ESP8684

Technical Reference Manual

Pre-release v0.3
Espressif Systems

About this Document

The **ESP8684** is targeted at developers working on low level software projects that use the ESP8684 SoC and other products in ESP32-C2 series. It describes the hardware modules listed below for the ESP8684 SoC and other products in ESP32-C2 series. The modules detailed in this document provide an overview, list of features, hardware architecture details, any necessary programming procedures, as well as register descriptions.

Navigation in This Document

Here are some tips on navigation through this extensive document:

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

Release Status at a Glance

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

No.	ESP8684 Chapters	Progress
1	ESP-RISC-V CPU (ESP-RV)	Published
2	GDMA Controller (GDMA)	Published
3	System and Memory (SYS_MEM)	Published
4	eFuse Controller (eFuse)	Published
5	IO MUX and GPIO Matrix (GPIO, IO_MUX)	Published
6	Reset and Clock (CLKRST)	Published
7	Chip Boot Control (BOOTCTRL)	Published
8	Interrupt Matrix (INTERRUPT)	Published
9	Low-Power Management (RTC_CNTL)	Published
10	System Timer (SYSTIMER)	Published
11	Timer Group (TIMG)	Published
12	Watchdog Timers (WDT)	Published
13	System Registers (SYSREG)	Published
14	Debug Assist (ASSIST_DEBUG)	Published
15	ECC Accelerator (ECC)	Published
16	SHA Accelerator (SHA)	Published
17	External Memory Encryption and Decryption (XTS_AES)	Published
18	Random Number Generator (RNG)	Published
19	Clock Glitch Detection (CLK_GLITCH)	0%
20	UART Controller (UART)	Published
21	SPI Controller (SPI)	Published
22	I2C Controller (I2C)	Published
23	LED PWM Controller (LEDC)	Published
24	On-Chip Sensors and Analog Signal Processing	Published

Note:

Check the link or the QR code to make sure that you use the latest version of this document:
https://www.espressif.com/documentation/esp8684_technical_reference_manual_en.pdf

Contents

List of Tables

List of Figures

1 ESP-RISC-V CPU

1.1 Overview

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

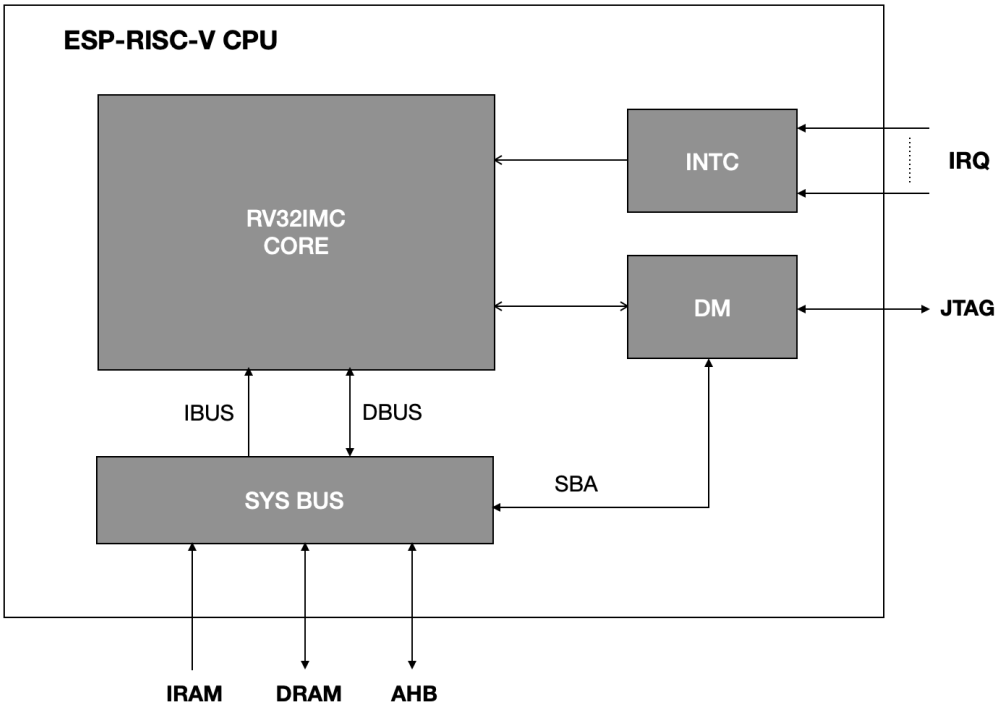


Figure 1-1. CPU Block Diagram

1.2 Features

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

- Operating clock frequency up to 120 MHz
- Zero wait cycle access to on-chip SRAM and Cache for program and data access over IRAM/DRAM interface
- Interrupt controller (INTC) with up to 31 vectored interrupts with programmable priority and threshold levels
- Debug module (DM) compliant with RISC-V debug specification v0.13 with external debugger support over an industry-standard JTAG/USB port
- Debugger direct system bus access (SBA) to memory and peripherals
- Hardware trigger compliant to RISC-V debug specification v0.13 with up to 2 breakpoints/watchpoints
- Physical memory protection (PMP) for up to 16 regions
- 32-bit AHB system bus for peripheral access

- Configurable events for core performance metrics

1.3 Address Map

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

Table 1-1. CPU Address Map

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

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

1.4 Configuration and Status Registers (CSRs)

1.4.1 Register Summary

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

Name	Description	Address	Access
Machine Information CSRs			
mvendorid	Machine Vendor ID	0xF11	RO
marchid	Machine Architecture ID	0xF12	RO
mimpid	Machine Implementation ID	0xF13	RO
mhartid	Machine Hart ID	0xF14	RO
Machine Trap Setup CSRs			
mstatus	Machine-Mode Status	0x300	R/W
misa [1]	Machine ISA	0x301	R/W
mtvec [2]	Machine Trap Vector	0x305	R/W
Machine Trap Handling CSRs			
mscratch	Machine Scratch	0x340	R/W
mepc	Machine Trap Program Counter	0x341	R/W
mcause [3]	Machine Trap Cause	0x342	R/W
mtval	Machine Trap Value	0x343	R/W

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

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

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

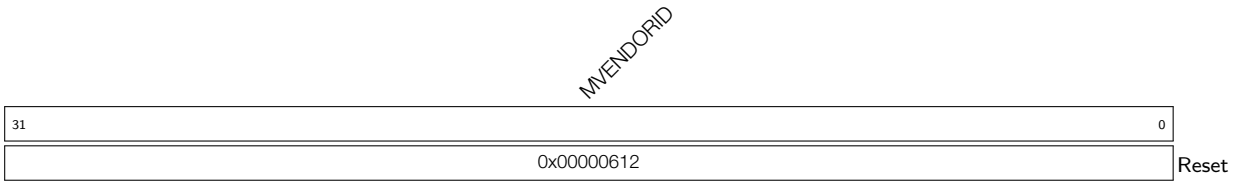
Name	Description	Address	Access
Physical Memory Protection (PMP) CSRs			
pmpcfg0	Physical memory protection configuration	0x3A0	R/W
pmpcfg1	Physical memory protection configuration	0x3A1	R/W
pmpcfg2	Physical memory protection configuration	0x3A2	R/W
pmpcfg3	Physical memory protection configuration	0x3A3	R/W
pmpaddr0	Physical memory protection address	0x3B0	R/W
pmpaddr1	Physical memory protection address	0x3B1	R/W
pmpaddr2	Physical memory protection address	0x3B2	R/W
pmpaddr3	Physical memory protection address	0x3B3	RO
pmpaddr4	Physical memory protection address	0x3B4	RO
pmpaddr5	Physical memory protection address	0x3B5	RO
pmpaddr6	Physical memory protection address	0x3B6	RO
pmpaddr7	Physical memory protection address	0x3B7	RO
pmpaddr8	Physical memory protection address	0x3B8	RO
pmpaddr9	Physical memory protection address	0x3B9	RO
pmpaddr10	Physical memory protection address	0x3BA	RO
pmpaddr11	Physical memory protection address	0x3BB	RO
pmpaddr12	Physical memory protection address	0x3BC	RO
pmpaddr13	Physical memory protection address	0x3BD	RO
pmpaddr14	Physical memory protection address	0x3BE	RO
pmpaddr15	Physical memory protection address	0x3BF	RO
Trigger Module CSRs (shared with Debug Mode)			
tselect	Trigger Select Register	0x7A0	R/W
tdata1	Trigger Abstract Data 1	0x7A1	R/W
tdata2	Trigger Abstract Data 2	0x7A2	R/W
tcontrol	Global Trigger Control	0x7A5	R/W
Debug Mode CSRs			
dcsr	Debug Control and Status	0x7B0	R/W
dpc	Debug PC	0x7B1	R/W
dscratch0	Debug Scratch Register 0	0x7B2	R/W
dscratch1	Debug Scratch Register 1	0x7B3	R/W
Performance Counter CSRs (Custom) [4]			
mpcer	Machine Performance Counter Event	0x7E0	R/W
mpcmr	Machine Performance Counter Mode	0x7E1	R/W
mpccr	Machine Performance Counter Count	0x7E2	R/W
GPIO Access CSRs (Custom)			
cpu_gpio_oen	GPIO Output Enable	0x803	R/W
cpu_gpio_in	GPIO Input Value	0x804	RO
cpu_gpio_out	GPIO Output Value	0x805	R/W

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

[4] These custom CSRs have been implemented in the address space reserved by RISC-V standard for custom use

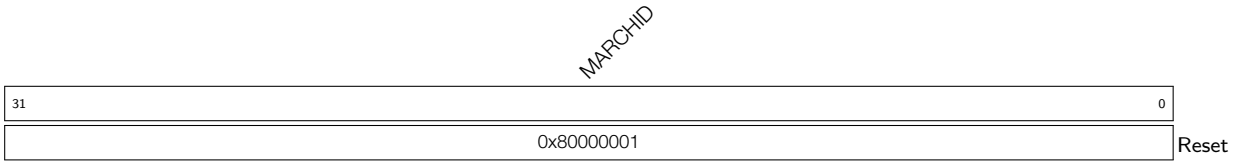
1.4.2 Register Description

Register 1.1. mvendorid (0xF11)



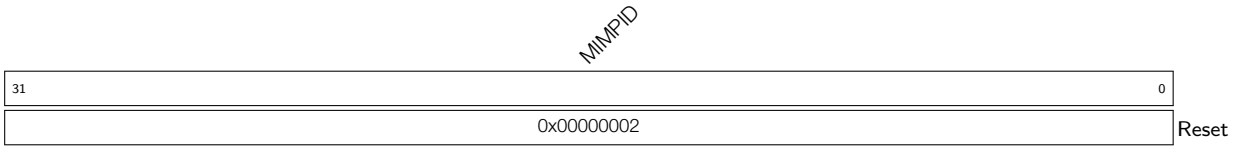
MVENDORID Vendor ID. (RO)

Register 1.2. marchid (0xF12)



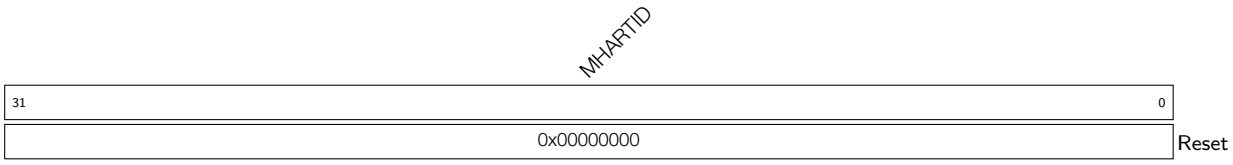
MARCHID Architecture ID. (RO)

Register 1.3. mimpid (0xF13)



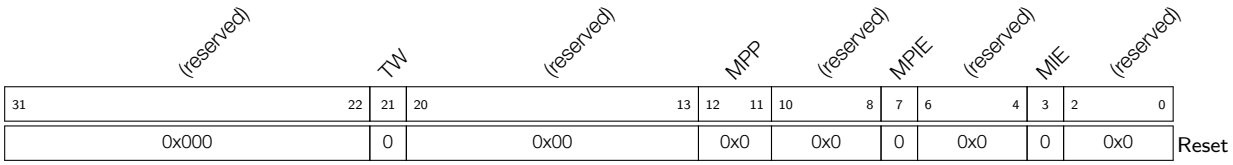
MIMPID Implementation ID. (RO)

Register 1.4. mhartid (0xF14)



MHARTID Hart ID. (RO)

Register 1.5. mstatus (0x300)



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

MPIE Previous MIE. (R/W)

MPP Machine previous privilege mode. (R/W)

Possible values:

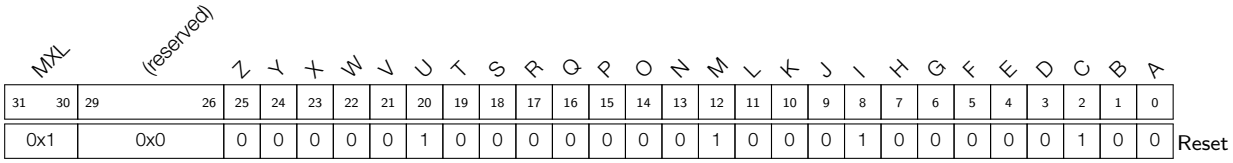
- 0x0: User-mode
- 0x3: Machine-mode

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

TW Timeout wait. (R/W)

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

Register 1.6. misa (0x301)



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

Reset

MXL Machine XLEN = 1 (32-bit). (RO)

Z Reserved = 0. (RO)

Y Reserved = 0. (RO)

X Non-standard extensions present = 0. (RO)

W Reserved = 0. (RO)

V Reserved = 0. (RO)

U User-mode implemented = 1. (RO)

T Reserved = 0. (RO)

S Supervisor-mode implemented = 0. (RO)

R Reserved = 0. (RO)

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

P Reserved = 0. (RO)

O Reserved = 0. (RO)

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

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

L Reserved = 0. (RO)

K Reserved = 0. (RO)

J Reserved = 0. (RO)

I RV32I base ISA = 1. (RO)

H Hypervisor extension = 0. (RO)

G Additional standard extensions present = 0. (RO)

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

E RV32E base ISA = 0. (RO)

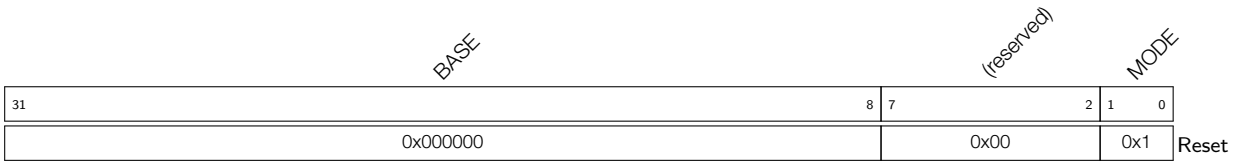
D Double-precision floating-point extension = 0. (RO)

C Compressed Extension = 1. (RO)

B Reserved = 0. (RO)

A Atomic Extension = 0. (RO)

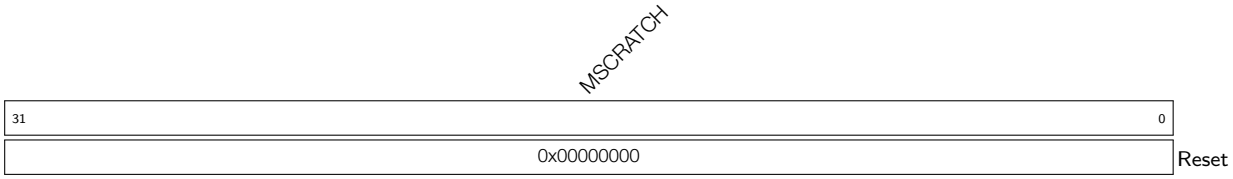
Register 1.7. mtvec (0x305)



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

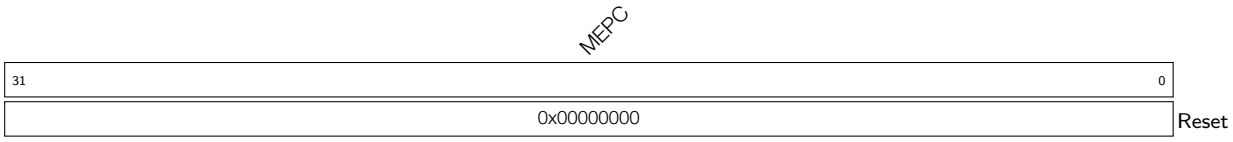
BASE Higher 24 bits of trap vector base address aligned to 256 bytes. (R/W)

Register 1.8. mscratch (0x340)



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

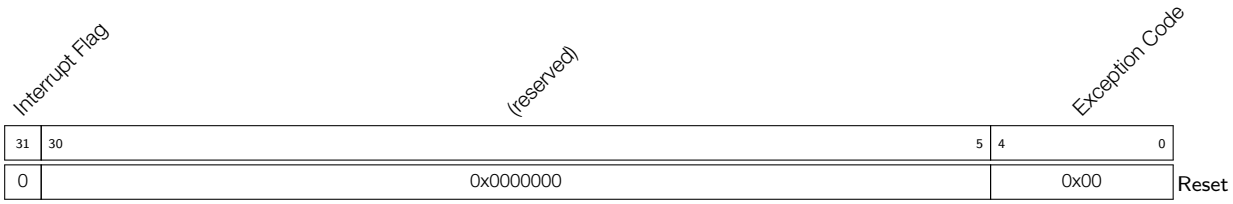
Register 1.9. mepc (0x341)



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

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

Register 1.10. mcause (0x342)



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

Possible exception IDs are:

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

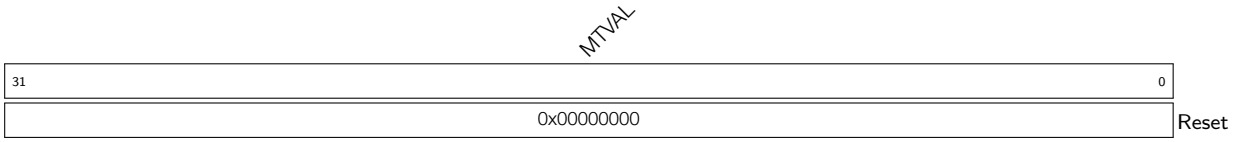
Note: Exception ID 0x0 (instruction access misaligned) is not present because CPU always masks the lowest bit of the address during instruction fetch.

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

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

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

Register 1.11. mtval (0x343)



MTVAL Machine trap value. (R/W)

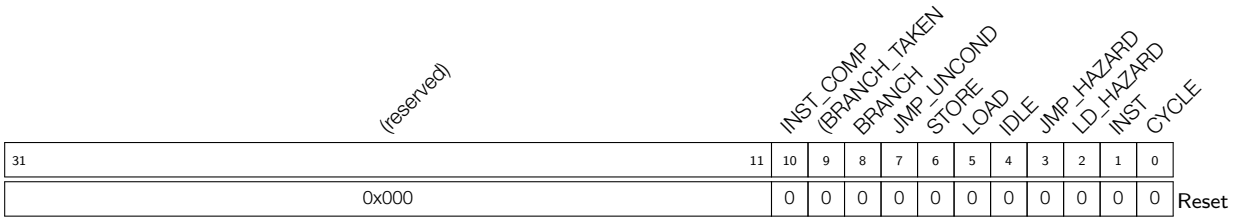
This is automatically updated with an exception dependent data which may be useful for handling that exception.

Data is to be interpreted depending upon exception IDs:

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

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

Register 1.12. mpcer (0x7E0)



INST_COMP Count Compressed Instructions. (R/W)

BRANCH_TAKEN Count Branches Taken. (R/W)

BRANCH Count Branches. (R/W)

JMP_UNCOND Count Unconditional Jumps. (R/W)

STORE Count Stores. (R/W)

LOAD Count Loads. (R/W)

IDLE Count IDLE Cycles. (R/W)

JMP_HAZARD Count Jump Hazards. (R/W)

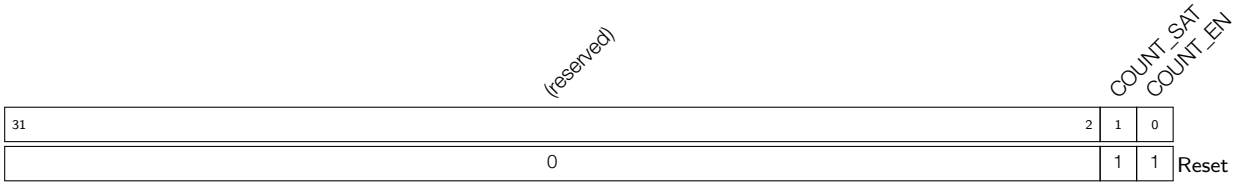
LD_HAZARD Count Load Hazards. (R/W)

INST Count Instructions. (R/W)

CYCLE Count Clock Cycles. (R/W)

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

Register 1.13. mpcmr (0x7E1)



COUNT_SAT Counter Saturation Control. (R/W)

Possible values:

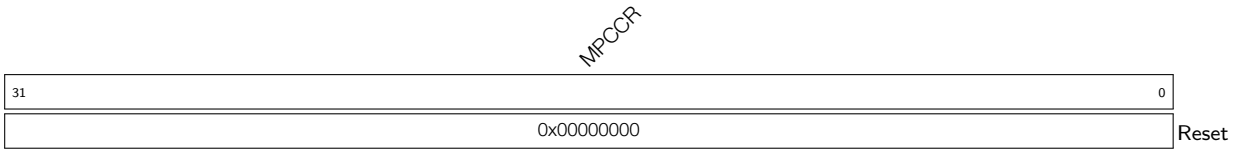
- 0: Overflow on maximum value
- 1: Halt on maximum value

COUNT_EN Counter Enable Control. (R/W)

Possible values:

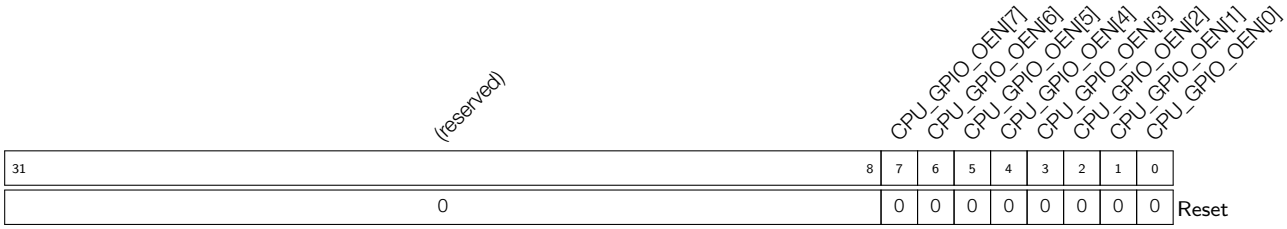
- 0: Disabled
- 1: Enabled

Register 1.14. mpccr (0x7E2)



MPCCR Machine Performance Counter Value. (R/W)

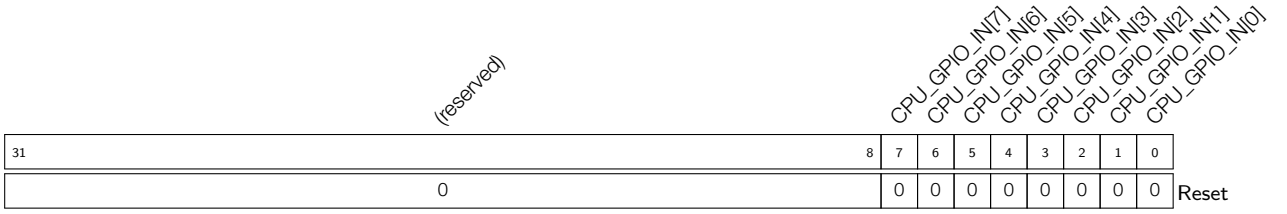
Register 1.15. cpu_gpio_oen (0x803)



CPU_GPIO_OEN GPIOn (n=0 ~ 21) Output Enable. CPU_GPIO_OEN[7:0] correspond to output enable signals cpu_gpio_out_oen[7:0] in Table 5-2 *Peripheral Signals via GPIO Matrix*. CPU_GPIO_OEN value matches that of cpu_gpio_out_oen.
CPU_GPIO_OEN is the enable signal of CPU_GPIO_OUT. (R/W)

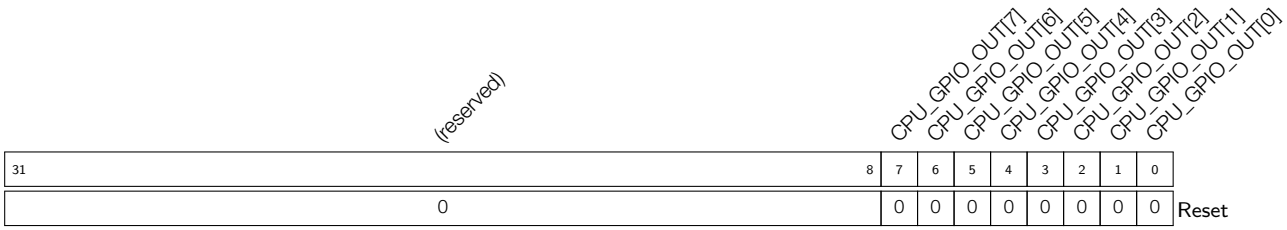
- 0: GPIO output disable
- 1: GPIO output enable

Register 1.16. cpu_gpio_in (0x804)



CPU_GPIO_IN GPIOn (n=0 ~ 21) Input Value. It is a CPU CSR to read input value (1=high, 0=low) from SoC GPIO pin.
CPU_GPIO_IN[7:0] correspond to input signals cpu_gpio_in[7:0] in Table 5-2 *Peripheral Signals via GPIO Matrix*.
CPU_GPIO_IN[7:0] can only be mapped to GPIO pins through GPIO matrix. For details please refer to Section 2 in Chapter *IO MUX and GPIO Matrix (GPIO, IO MUX)*. (RO)

Register 1.17. cpu_gpio_out (0x805)



CPU_GPIO_OUT GPIOn (n=0 ~ 21) Output Value. It is a CPU CSR to write value (1=high, 0=low) to SoC GPIO pin. The value takes effect only when CPU_GPIO_OEN is set.

CPU_GPIO_OUT[7:0] correspond to output signals cpu_gpio_out[7:0] in Table 5-2 *Peripheral Signals via GPIO Matrix*.

CPU_GPIO_OUT[7:0] can only be mapped to GPIO pins through GPIO matrix. For details please refer to Section 2 in Chapter *IO MUX and GPIO Matrix (GPIO, IO MUX)*. (R/W)

1.5 Interrupt Controller

1.5.1 Features

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

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

1.5.2 Functional Description

Each interrupt ID has 5 properties associated with it:

1. Enable State (0-1):
   - Determines if an interrupt is enabled to be captured and serviced by the CPU.
   - Programmed by writing the corresponding bit in INTERRUPT_CORE0_CPU_INT_ENABLE_REG.
2. Type (0-1):
   - Enables latching the state of an interrupt signal on its rising edge.
   - Programmed by writing the corresponding bit in INTERRUPT_CORE0_CPU_INT_TYPE_REG.
   - An interrupt for which type is kept 0 is referred as a 'level' type interrupt.
   - An interrupt for which type is set to 1 is referred as an 'edge' type interrupt.
3. Priority (1-15):
   - Determines which interrupt, among multiple pending interrupts, the CPU will service first.
   - Programmed by writing to the INTERRUPT_CORE0_CPU_INT_PRI_*n*_REG for a particular interrupt ID *n* in range (1-31).
   - Enabled interrupts with priorities zero or less than the threshold value in INTERRUPT_CORE0_CPU_INT_THRESH_REG are masked.
   - Priority levels increase from 1 (lowest) to 15 (highest).
   - Interrupts with same priority are statically prioritized by their IDs, lowest ID having highest priority.
4. Pending State (0-1):
   - Reflects the captured state of an enabled and unmasked interrupt signal.
   - For each interrupt ID, the corresponding bit in read-only INTERRUPT_CORE0_CPU_INT_EIP_STATUS_REG gives its pending state.
   - A pending interrupt will cause CPU to enter trap if no other pending interrupt has higher priority.

- A pending interrupt is said to be 'claimed' if it preempts the CPU and causes it to jump to the corresponding trap vector address.
- All pending interrupts which are yet to be serviced are termed as 'unclaimed'.

5. Clear State (0-1):
   - Toggling this will clear the pending state of claimed edge-type interrupts only.
   - Toggled by first setting and then clearing the corresponding bit in INTERRUPT_CORE0_CPU_INT_CLEAR_REG.
   - Pending state of a level type interrupt is unaffected by this and must be cleared from source.
   - Pending state of an unclaimed edge type interrupt can be flushed, if required, by first clearing the corresponding bit in INTERRUPT_CORE0_CPU_INT_ENABLE_REG and then toggling same bit in INTERRUPT_CORE0_CPU_INT_CLEAR_REG.

When CPU services a pending interrupt, it:

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

Table 1-3 shows the mapping of each interrupt ID with the corresponding trap-vector address. In short, the word aligned trap address for an interrupt with a certain $ID = i$ can be calculated as $(mtvec + 4i)$.

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

Table 1-3. ID wise map of Interrupt Trap-Vector Addresses

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

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

Upon execution of MRET instruction, the CPU:

- copies the state of MPIE back into MIE, and subsequently clears MPIE. This means that if previously MPIE was set, then, after MRET, MIE will be set, thereby enabling interrupts globally.
- jumps to the address stored in mepc and resumes execution.

It is possible to perform software assisted nesting of interrupts inside an ISR as explained in 1.5.3.

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

- Only if an interrupt has non-zero priority, higher or equal to the value in the threshold register, will it be reflected in INTERRUPT_CORE0_CPU_INT_EIP_STATUS_REG.
- If an interrupt is visible in INTERRUPT_CORE0_CPU_INT_EIP_STATUS_REG and has yet to be serviced, then it's possible to mask it (and thereby prevent the CPU from servicing it) by either lowering the value of its priority or increasing the global threshold.
- If an interrupt, visible in INTERRUPT_CORE0_CPU_INT_EIP_STATUS_REG, is to be flushed (and prevented from being serviced at all), then it must be disabled (and cleared if it is of edge type).

1.5.3 Suggested Operation

1.5.3.1 Latency Aspects

There is latency involved while configuring the Interrupt Controller.

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

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

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

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

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

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

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

1.5.3.2 Configuration Procedure

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

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

1. save the state of MIE and clear MIE to 0
2. depending upon the type of the interrupt (edge/level), set/unset the *n*th bit of INTERRUPT_CORE0_CPU_INT_TYPE_REG
3. set the priority by writing a value to INTERRUPT_CORE0_CPU_INT_PRI_*n*_REG in range 1(lowest) to 15 (highest)
4. set the *n*th bit of INTERRUPT_CORE0_CPU_INT_ENABLE_REG
5. execute FENCE instruction
6. restore the state of MIE

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

Upon entering into an ISR, software must toggle the *n*th bit of INTERRUPT_CORE0_CPU_INT_CLEAR_REG if the interrupt is of edge type, or clear the source of the interrupt if it is of level type.

Software may also update the value of INTERRUPT_CORE0_CPU_INT_THRESH_REG and program MIE=1 for allowing higher priority interrupts to preempt the current ISR (nesting), however, before doing so, all the state CSRs must be saved (mepc, mstatus, mcause, etc.) since they will get overwritten due to occurrence of such an interrupt. Later, when exiting the ISR, the values of these CSRs must be restored.

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

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

1. save the state of MIE and clear MIE to 0
2. check if the interrupt is pending in INTERRUPT_CORE0_CPU_INT_EIP_STATUS_REG
3. set/unset the *n*th bit of INTERRUPT_CORE0_CPU_INT_ENABLE_REG
4. if the interrupt is of edge type and was found to be pending in step 2 above, *n*th bit of INTERRUPT_CORE0_CPU_INT_CLEAR_REG must be toggled, so that its pending status gets flushed
5. execute FENCE instruction
6. restore the state of MIE

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

1.5.4 Register Summary

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

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

1.5.5 Register Description

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

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

1.6 Debug

1.6.1 Overview

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

Figure 1-2 below shows the main components of External Debug Support.

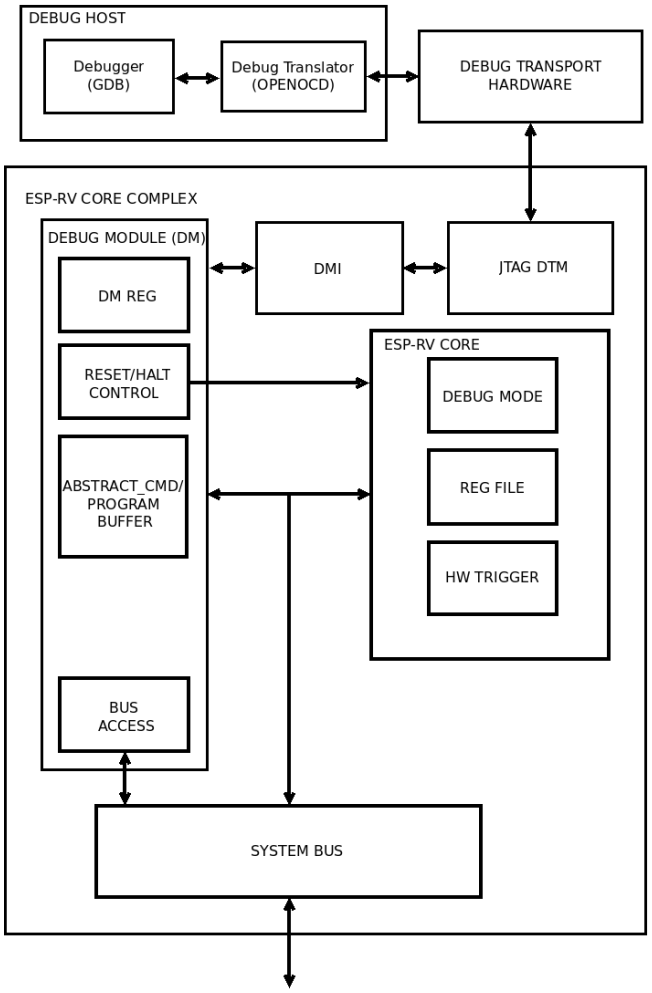


Figure 1-2. Debug System Overview

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

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

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

1.6.2 Features

Basic debug functionality has the following features:

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

1.6.3 Functional Description

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

1.6.4 Register Summary

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

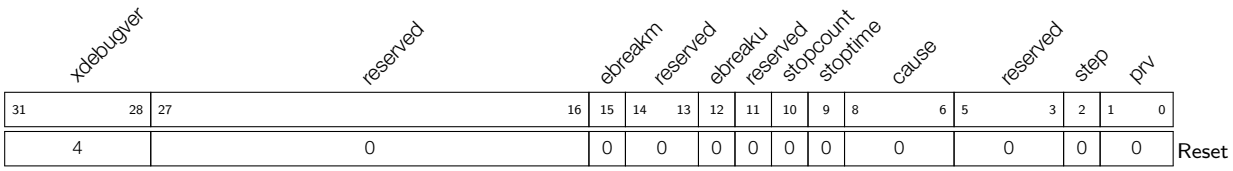
Name	Description	Address	Access
dcsr	Debug Control and Status	0x7B0	R/W
dpc	Debug PC	0x7B1	R/W
dscratch0	Debug Scratch Register 0	0x7B2	R/W
dscratch1	Debug Scratch Register 1	0x7B3	R/W

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

1.6.5 Register Description

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

Register 1.18. dcsr (0x7B0)



xdebugver Debug version. (RO)

- 4: External debug support exists

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

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

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

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

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

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

Other values are reserved for future use. (RO)

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

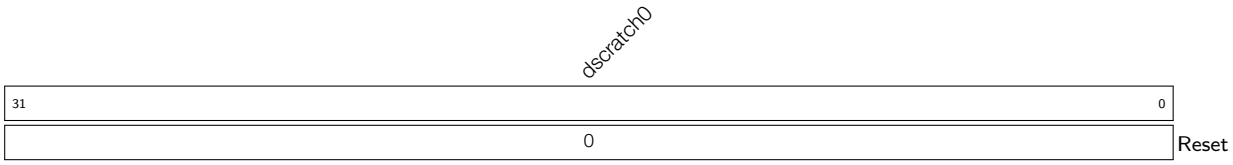
prv Contains the privilege level the core was operating in when Debug Mode was entered. A debugger can change this value to change the core's privilege level when exiting Debug Mode. Only **0x3** (machine-mode) and **0x0** (user-mode) are supported.

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

Register 1.19. dpc (0x7B1)

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

Register 1.20. dscratch0 (0x7B2)



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

Register 1.21. dscratch1 (0x7B3)

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

1.7 Hardware Trigger

1.7.1 Features

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

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

1.7.2 Functional Description

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

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

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

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

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

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

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

Each trigger unit only supports match on address, although this address could either be that of a load/store access or the virtual address of an instruction. The address and size of a region are specified by writing to maddress (tdata2) CSR for the selected trigger unit. Larger than 1 byte region sizes are specified through NAPOT (naturally aligned power of two) encoding (see Table 1-5) and enabled by setting match bit in mcontrol. Note that

for NAPOT encoded addresses, by definition, the start address is constrained to be aligned to (i.e. an integer multiple of) the region size.

Table 1-5. NAPOT encoding for maddress

maddress(31-0)	Start Address	Size (bytes)
aaa...aaaaaaaaa0	aaa...aaaaaaaaa0	2
aaa...aaaaaaaa01	aaa...aaaaaaaa00	4
aaa...aaaaaaa011	aaa...aaaaaaa000	8
aaa...aaaaaa0111	aaa...aaaaaa0000	16
	....	
a01...1111111111	a00...0000000000	2^{31}

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

1.7.3 Trigger Execution Flow

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

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

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

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

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

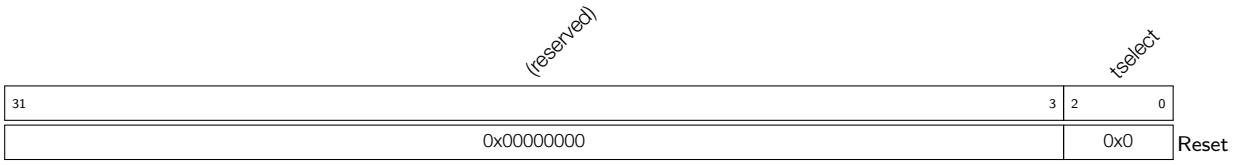
1.7.4 Register Summary

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

Name	Description	Address	Access
tselect	Trigger Select Register	0x7A0	R/W
tdata1	Trigger Abstract Data 1	0x7A1	R/W
tdata2	Trigger Abstract Data 2	0x7A2	R/W
tcontrol	Global Trigger Control	0x7A5	R/W

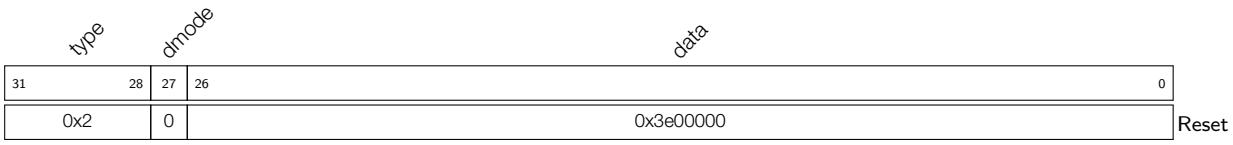
1.7.5 Register Description

Register 1.22. tselect (0x7A0)



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

Register 1.23. tdata1 (0x7A1)



type Type of trigger. (RO)

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

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

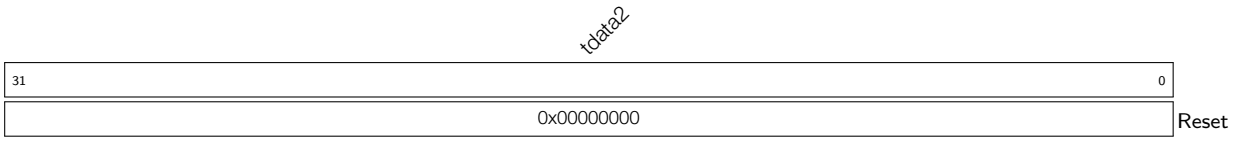
- 0: Both Debug and machine-mode can write the tdata1 and tdata2 registers at the selected tselect.
- 1: Only Debug Mode can write the tdata1 and tdata2 registers at the selected tselect. Writes from other modes are ignored.

** Note : Only writable from debug mode.*

data Abstract tdata1 content. (R/W)

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

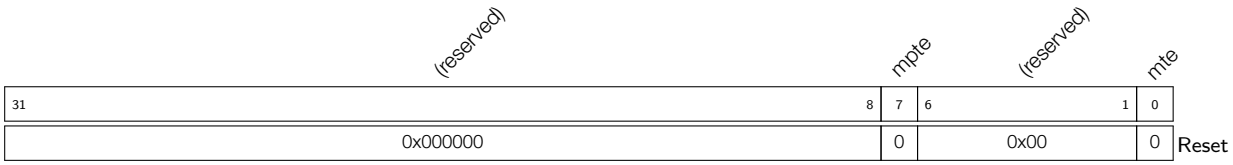
Register 1.24. tdata2 (0x7A2)



tdata2 Abstract tdata2 content. (R/W)

This will always be interpreted as maddress since only match type (0x2) triggers are supported.

Register 1.25. tcontrol (0x7A5)



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

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

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

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

Register 1.26. mcontrol (0x7A1)

(reserved)	dmode	(reserved)	hit	(reserved)	action	(reserved)	match	m	(reserved)	u	execute	store	load	
31 28	27	26 21	20	19 16	15 12	11	10 7	6	5 4	3	2	1	0	
0x2	0	0x1f	0	0	0	0	0	0	0	0	0	0	0	Reset

dmode Same as dmode in tdata1.

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

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

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

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

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

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

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

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

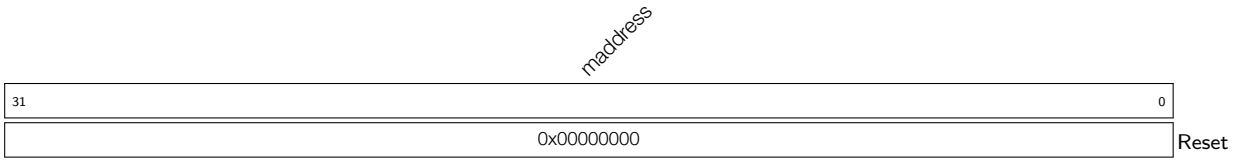
u Set this for enabling selected trigger to operate in user-mode. (R/W)

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

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

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

Register 1.27. maddress (0x7A2)



maddress Address used by the selected trigger when performing match operation. (R/W)

This is decoded as NAPOT when match=1 in mcontrol.

1.8 Memory Protection

1.8.1 Overview

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

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

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

1.8.2 Features

The memory protection unit has the following features:

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

1.8.3 Functional Description

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

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

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

1.8.4 Register Summary

Below is a list of PMP CSRs supported by the CPU. These are only accessible from machine-mode. As mentioned earlier, `pmpaddrX0-2` are kept programmable to split the SRAM region as per software requirements. `pmpaddrX3-15` are hard coded with values specified in "CSR Reset Value" column. These hard-coded values

have been derived to match the SoC memory map specified in ”PMP Region” column. To enable any PMP region, the A field in associated `pmpcfgX` register should always be programmed with value specified in ”Address Matching Mode” column.

Name	Description	CSR Address	CSR Reset Value	CSR Access	Address Matching Mode	PMP Region
pmpcfg0	PMP config register	0x3A0	0x0	R/W	-	-
pmpcfg1	PMP config register	0x3A1	0x0	R/W	-	-
pmpcfg2	PMP config register	0x3A2	0x0	R/W	-	-
pmpcfg3	PMP config register	0x3A3	0x0	R/W	-	-
pmpaddr0	PMP address register	0x3B0	0x0	R/W	OFF	IRAM Base Address
pmpaddr1	PMP address register	0x3B1	0x0	R/W	TOR	IRAM End Address
pmpaddr2	PMP address register	0x3B2	0x0	R/W	OFF	DRAM Base Address
pmpaddr3	PMP address register	0x3B3	0x0FF38000	RO	TOR	DRAM End Address 0x3FCDFFFF
pmpaddr4	PMP address register	0x3B4	0x08FFFFFF	RO	NAPOT	0x20000000 - 0x27FFFFFF (128 MB)
pmpaddr5	PMP address register	0x3B5	0x0F07FFFF	RO	NAPOT	0x3C000000 - 0x3C3FFFFF (4 MB)
pmpaddr6	PMP address register	0x3B6	0x0FFC0000	RO	OFF	0x3FF00000
pmpaddr7	PMP address register	0x3B7	0x0FFD4000	RO	TOR	0x3FF00000 - 0x3FF4FFFF (320 KB)
pmpaddr8	PMP address register	0x3B8	0x10000000	RO	OFF	0x40000000
pmpaddr9	PMP address register	0x3B9	0x10024000	RO	TOR	0x40000000 - 0x4008FFFF (576 KB)
pmpaddr10	PMP address register	0x3BA	0x1087FFFF	RO	NAPOT	0x42000000 - 0x423FFFFF (4 MB)
pmpaddr11	PMP address register	0x3BB	0x1801FFFF	RO	NAPOT	0x60000000 - 0x600FFFFF (1 MB)
pmpaddr12	PMP address register	0x3BC	0x100DF7FF	RO	NAPOT	0x4037C000 - 0x4037FFFF (16 KB)
pmpaddr13	PMP address register	0x3BD	0x3FFFFFFF	RO	NA4	0xFFFFFFFF (4 Byte)
pmpaddr14	PMP address register	0x3BE	0x0	RO	OFF	0x0

Name	Description	CSR Address	CSR Reset Value	CSR Access	Address Matching Mode	PMP Region
pmpaddr15	PMP address register	0x3BF	0x3FFFFFFF	RO	TOR	0xFFFFFFFE (4 GB)

1.8.5 Register Description

PMP unit implements all `pmpcfg0-3` and `pmpaddr0-15` CSRs as defined in RISC-V Instruction Set Manual Volume II: Privileged Architecture, Version 1.10.

2 GDMA Controller (GDMA)

2.1 Overview

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

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

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

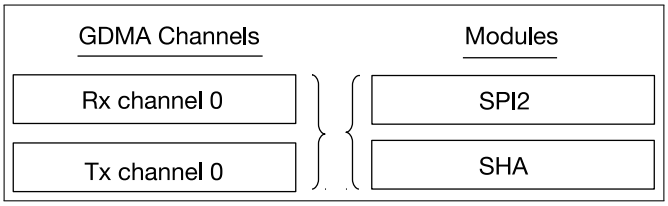


Figure 2-1. Modules with GDMA Feature and GDMA Channels

2.2 Features

The GDMA controller has the following features:

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

2.3 Architecture

In ESP8684, all modules that need high-speed data transfer support GDMA. The GDMA controller and CPU data bus have access to the same address space in internal RAM. Figure 2-2 shows the basic architecture of the GDMA engine.

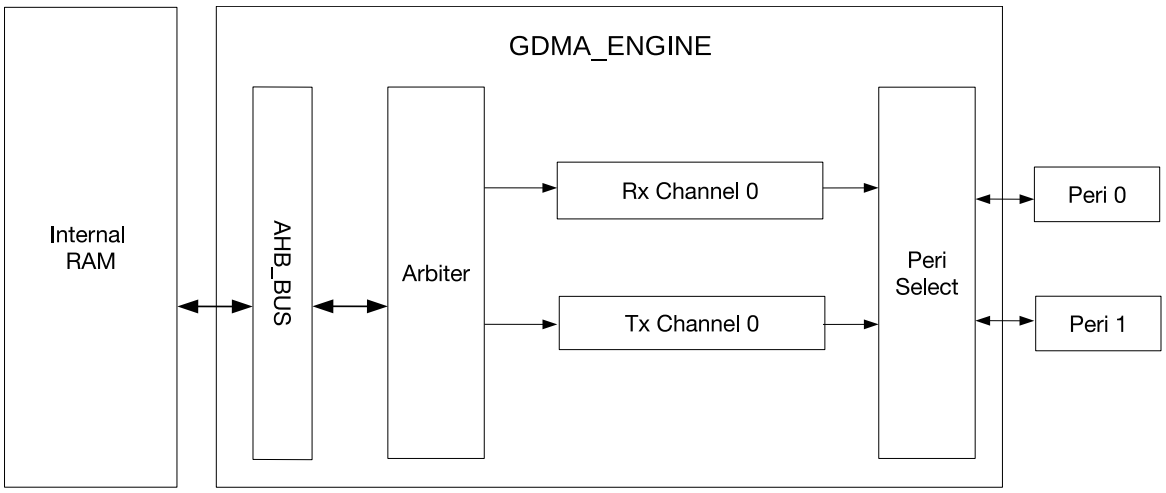


Figure 2-2. GDMA Engine Architecture

The GDMA controller has two independent channels, i.e. one transmit channel and one receive channel. Every channel can be connected to different peripherals. In other words, channels are general-purpose, shared by peripherals. The GDMA engine reads data from or writes data to internal RAM via the AHB_BUS. For available address range of Internal RAM, please see Chapter 3 *System and Memory*. Software can use the GDMA engine through linked lists. These linked lists, stored in internal RAM, consist of outlink and inlink. The GDMA controller reads an outlink (i.e. a linked list of transmit descriptors) from internal RAM and transmits data in corresponding RAM according to the outlink, or reads an inlink (i.e. a linked list of receive descriptors) and stores received data into specific address space in RAM according to the inlink.

2.4 Functional Description

2.4.1 Data Transfer Between Peripheral and Memory

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

Every transmit and receive channel can be connected to any peripheral with GDMA feature. Table 2-1 illustrates how to select the peripheral to be connected via registers. When one channel is connected to a peripheral, the other channel can not be connected to that peripheral.

Table 2-1. Selecting Peripherals via Register Configuration

GDMA_PERI_IN_SEL_CH0 GDMA_PERI_OUT_SEL_CH0	Peripheral
0	SPI2
1	Reserved
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	SHA
8	Reserved

2.4.2 Memory-to-Memory Data Transfer

The GDMA controller also allows memory-to-memory data transfer. Such data transfer can be enabled by setting GDMA_MEM_TRANS_EN_CH0, which connects the output of transmit channel 0 to the input of receive channel 0.

2.4.3 Linked List

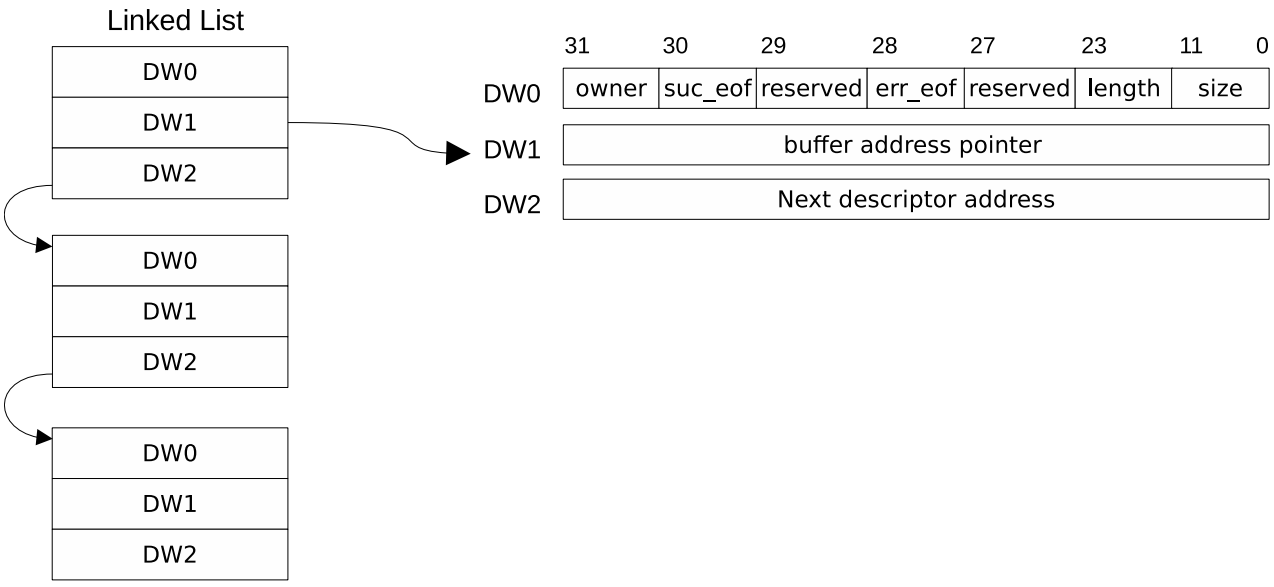


Figure 2-3. Structure of a Linked List

Figure 2-3 shows the structure of a linked list. An outlink and an inlink have the same structure. A linked list is formed by one or more descriptors, and each descriptor consists of three words. Linked lists should be in internal RAM for the GDMA engine to be able to use them. The meaning of each field is as follows:

- Owner (DW0) [31]: Specifies who is allowed to access the buffer that this descriptor points to.
  1'b0: CPU can access the buffer;
  1'b1: The GDMA controller can access the buffer.

When the GDMA controller stops using the buffer, this bit in a transmit descriptor is automatically cleared by hardware, and this bit in a receive descriptor is automatically cleared by hardware only if GDMA_OUT_AUTO_WRBACK_CH0 is set to 1. Before software loads a linked list, this bit should be set to 1.

Note: GDMA_OUT is the prefix of transmit channel registers, and GDMA_IN is the prefix of receive channel registers.

- suc_eof (DW0) [30]: Specifies whether this descriptor is the last descriptor in the list.
  1'b0: This descriptor is not the last one;
  1'b1: This descriptor is the last one.
  Software clears suc_eof bit in receive descriptors. When a frame or packet has been received, this bit in the last receive descriptor is set by hardware, and this bit in the last transmit descriptor is set by software.
- Reserved (DW0) [29]: Reserved. Value of this bit does not matter.
- err_eof (DW0) [28]: Specifies whether the received data has errors.
  When a frame or packet has been received and an error is detected in the received frame or packet, this bit in the receive descriptor is set to 1 by hardware.
- Reserved (DW0) [27:24]: Reserved.
- Length (DW0) [23:12]: Specifies the number of valid bytes in the buffer that this descriptor points to. This field in a transmit descriptor is written by software and indicates how many bytes can be read from the buffer; this field in a receive descriptor is written by hardware automatically and indicates how many valid bytes have been stored into the buffer.
- Size (DW0) [11:0]: Specifies the size of the buffer that this descriptor points to. Size should be larger than or equal to length.
- Buffer address pointer (DW1): Address of the buffer. This field can only point to internal RAM.
- Next descriptor address (DW2): Address of the next descriptor. If the current descriptor is the last one, this value is 0. This field can only point to internal RAM.

When a data frame or packet has been received, the suc_eof bit in the current receive descriptor will be set to 1, and the GDMA controller stops data transmission to the buffer pointed by the current receive descriptor. Even if the length of data received is smaller than the size of the buffer, data received in the next transaction would not be stored in the available space of this buffer. The data would rather be stored in the buffer pointed by the next receive descriptor.

2.4.4 Enabling GDMA

Software uses the GDMA controller through linked lists. When the GDMA controller receives data, software loads an inlink, configures GDMA_INLINK_ADDR_CH0 field with address of the first receive descriptor, and sets GDMA_INLINK_START_CH0 bit to enable GDMA. When the GDMA controller transmits data, software loads an outlink, prepares data to be transmitted, configures GDMA_OUTLINK_ADDR_CH0 field with address of the first transmit descriptor, and sets GDMA_OUTLINK_START_CH0 bit to enable GDMA. GDMA_INLINK_START_CH0 bit and GDMA_OUTLINK_START_CH0 bit are cleared automatically by hardware.

In some cases, you may want to append more descriptors to a DMA transfer that is already started. Naively, it would seem to be possible to do this only by setting the next descriptor address pointer field (DW2) at the end of the current list to the first descriptor of the to-be-added list. However, this strategy fails if the existing DMA transfer is almost or entirely finished. Instead, the GDMA engine has specialized logic to make sure a DMA

transfer can be continued or restarted: if it is still ongoing, it will make sure to take the appended descriptors into account; if the transfer has already finished, it will restart with the new descriptors. This is implemented in the Restart function.

When using the Restart function, software needs to rewrite address of the first descriptor in the new list to DW2 of the last descriptor in the loaded list, and set GDMA_INLINK_RESTART_CH0 bit or GDMA_OUTLINK_RESTART_CH0 bit (these two bits are cleared automatically by hardware). As shown in Figure 2-4, by doing so hardware can obtain the address of the first descriptor in the new list when reading the last descriptor in the loaded list, and then read the new list.

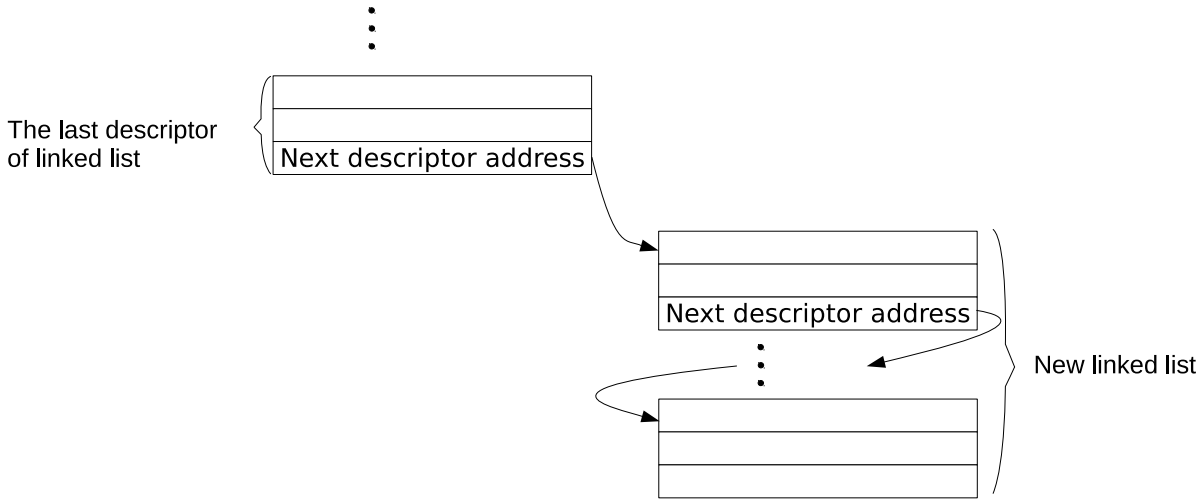


Figure 2-4. Relationship among Linked Lists

2.4.5 Linked List Reading Process

Once configured and enabled by software, the GDMA controller starts to read the linked list from internal RAM. The GDMA performs checks on descriptors in the linked list. Only if descriptors pass the checks, will the corresponding GDMA channel transfer data. If the descriptors fail any of the checks, hardware will trigger descriptor error interrupt (either GDMA_IN_DSCR_ERR_CH0_INT or GDMA_OUT_DSCR_ERR_CH0_INT), and the channel will halt.

The checks performed on descriptors are:

- Owner bit check when GDMA_IN_CHECK_OWNER_CH0 or GDMA_OUT_CHECK_OWNER_CH0 is set to 1. If the owner bit is 0, the buffer should be accessed by the CPU. In this case, the owner bit fails the check. The owner bit check will be skipped if GDMA_IN_CHECK_OWNER_CH0 or GDMA_OUT_CHECK_OWNER_CH0 is 0;
- Buffer address pointer (DW1) check. If the buffer address pointer does not point to 0x3FCA0000 ~ 0x3FCDFFFF (please refer to Section 2.4.7), it fails the check.

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

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

2.4.6 EOF

The GDMA controller uses EOF (end of frame) flags to indicate the end of data frame or packet transmission.

Before the GDMA controller transmits data, GDMA_OUT_TOTAL_EOF_CH0_INT_ENA bit should be set to enable GDMA_OUT_EOF_CH0_INT interrupt. If data in the buffer pointed by the last descriptor (with EOF) has been transmitted, a GDMA_OUT_EOF_CH0_INT interrupt is generated.

Before the GDMA controller receives data, GDMA_IN_SUC_EOF_CH0_INT_ENA bit should be set to enable GDMA_IN_SUC_EOF_CH0_INT interrupt. If a data frame or packet has been received successfully, a GDMA_IN_SUC_EOF_CH0_INT interrupt is generated. In addition, the GDMA controller also supports GDMA_IN_ERR_CH0_EOF_INT interrupt. This interrupt is enabled by setting GDMA_IN_ERR_EOF_CH0_INT_ENA bit, and it indicates that a data frame or packet has been received with errors.

When detecting a GDMA_OUT_TOTAL_EOF_CH0_INT or a GDMA_IN_SUC_EOF_CH0_INT interrupt, software can record the value of GDMA_OUT_EOF_DES_ADDR_CH0 or GDMA_IN_SUC_EOF_DES_ADDR_CH0 field, i.e. address of the last descriptor. Therefore, software can tell which descriptors have been used and reclaim them.

Note: In this chapter, EOF of transmit descriptors refers to suc_eof, while EOF of receive descriptors refers to both suc_eof and err_eof.

2.4.7 Accessing Internal RAM

Any transmit and receive channel of GDMA can access 0x3FCA0000 ~ 0x3FCDFFFF in internal RAM. To improve data transfer efficiency, GDMA can send data in burst mode, which is disabled by default. This mode is enabled for receive channel by setting GDMA_IN_DATA_BURST_EN_CH0, and enabled for transmit channel by setting GDMA_OUT_DATA_BURST_EN_CH0.

Table 2-2. Descriptor Field Alignment Requirements

Inlink/Outlink	Burst Mode	Size	Length	Buffer Address Pointer
Inlink	0	—[1]	—	—
	1	Word-aligned	—	Word-aligned
Outlink	0	—	—	—
	1	—	—	—

[1] "—" means no alignment requirements.

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

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

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

2.4.8 Arbitration

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

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

2.5 GDMA Interrupts

- GDMA_IN_DSCR_EMPTY_CH0_INT: Triggered when the size of the buffer pointed by receive descriptors is smaller than the length of data to be received via receive channel 0.
- GDMA_IN_DSCR_ERR_CH0_INT: Triggered when an error is detected in a receive descriptor on receive channel 0.
- GDMA_IN_ERR_EOF_CH0_INT: Triggered when an error is detected in the data frame or packet received via receive channel 0. This interrupt is used only for UHCI0 peripheral (UART0 or UART1).
- GDMA_IN_SUC_EOF_CH0_INT: Triggered when a data frame or packet has been received via receive channel 0.
- GDMA_IN_DONE_CH0_INT: Triggered when all data corresponding to a receive descriptor has been received via receive channel 0.
- GDMA_OUT_TOTAL_EOF_CH0_INT: Triggered when all data corresponding to a linked list (including multiple descriptors) has been sent via transmit channel 0.
- GDMA_OUT_DSCR_ERR_CH0_INT: Triggered when an error is detected in a transmit descriptor on transmit channel 0.
- GDMA_OUT_EOF_CH0_INT: Triggered when EOF in a transmit descriptor is 1 and data corresponding to this descriptor has been sent via transmit channel 0. If GDMA_OUT_EOF_MODE_CH0 is 0, this interrupt will be triggered when the last byte of data corresponding to this descriptor enters GDMA's transmit channel; if GDMA_OUT_EOF_MODE_CH0 is 1, this interrupt is triggered when the last byte of data is taken from GDMA's transmit channel.
- GDMA_OUT_DONE_CH0_INT: Triggered when all data corresponding to a transmit descriptor has been sent via transmit channel 0.

2.6 Programming Procedures

2.6.1 Programming Procedure for GDMA Clock and Reset

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

1. Set SYSTEM_DMA_CLK_EN to enable GDMA's clock;
2. Clear SYSTEM_DMA_RST to reset GDMA.

2.6.2 Programming Procedure for GDMA's Transmit Channel

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

1. Set GDMA_OUT_RST_CH0 first to 1 and then to 0, to reset the state machine of GDMA's transmit channel and FIFO pointer;
2. Load an outlink, and configure GDMA_OUTLINK_ADDR_CH0 with address of the first transmit descriptor;
3. Configure GDMA_PERI_OUT_SEL_CH0 with the value corresponding to the peripheral to be connected, as shown in Table 2-1;
4. Set GDMA_OUTLINK_START_CH0 to enable GDMA's transmit channel for data transfer;
5. Configure and enable the corresponding peripheral (SPI2 or SHA). See details in individual chapters of these peripherals;
6. Wait for GDMA_OUT_EOF_CH0_INT interrupt, which indicates the completion of data transfer.

2.6.3 Programming Procedure for GDMA's Receive Channel

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

1. Set GDMA_IN_RST_CH0 first to 1 and then to 0, to reset the state machine of GDMA's receive channel and FIFO pointer;
2. Load an inlink, and configure GDMA_INLINK_ADDR_CH0 with address of the first receive descriptor;
3. Configure GDMA_PERI_IN_SEL_CH0 with the value corresponding to the peripheral to be connected, as shown in Table 2-1;
4. Set GDMA_INLINK_START_CH0 to enable GDMA's receive channel for data transfer;
5. Configure and enable the corresponding peripheral (SPI2). See details in individual chapters of these peripherals;
6. Wait for GDMA_IN_SUC_EOF_CH0_INT interrupt, which indicates that a data frame or packet has been received.

2.6.4 Programming Procedure for Memory-to-Memory Transfer

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

1. Set GDMA_OUT_RST_CH0 first to 1 and then to 0, to reset the state machine of GDMA's transmit channel and FIFO pointer;
2. Set GDMA_IN_RST_CH0 first to 1 and then to 0, to reset the state machine of GDMA's receive channel and FIFO pointer;
3. Load an outlink, and configure GDMA_OUTLINK_ADDR_CH0 with address of the first transmit descriptor;
4. Load an inlink, and configure GDMA_INLINK_ADDR_CH0 with address of the first receive descriptor;
5. Set GDMA_MEM_TRANS_EN_CH0 to enable memory-to-memory transfer;
6. Set GDMA_OUTLINK_START_CH0 to enable GDMA's transmit channel for data transfer;
7. Set GDMA_INLINK_START_CH0 to enable GDMA's receive channel for data transfer;

8. Wait for GDMA_IN_SUC_EOF_CH0_INT interrupt, which indicates that which indicates that a data transaction has been completed.

2.7 Register Summary

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

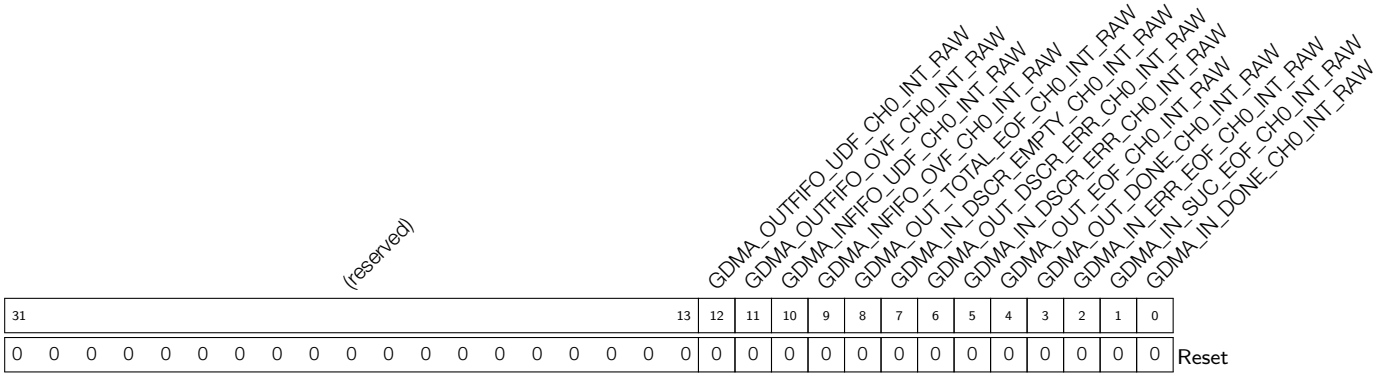
Name	Description	Address	Access
Interrupt Registers			
GDMA_INT_RAW_CH0_REG	Raw status interrupt of RX channel 0	0x0000	R/WTC/SS
GDMA_INT_ST_CH0_REG	Masked interrupt of RX channel 0	0x0004	RO
GDMA_INT_ENA_CH0_REG	Interrupt enable bits of RX channel 0	0x0008	R/W
GDMA_INT_CLR_CH0_REG	Interrupt clear bits of RX channel 0	0x000C	WT
Configuration Registers			
GDMA_MISC_CONF_REG	Miscellaneous register	0x0044	R/W
GDMA_IN_CONF0_CH0_REG	Configuration register 0 of RX channel 0	0x0070	R/W
GDMA_IN_CONF1_CH0_REG	Configuration register 1 of RX channel 0	0x0074	R/W
GDMA_IN_POP_CH0_REG	Pop control register of RX channel 0	0x007C	varies
GDMA_IN_LINK_CH0_REG	Link descriptor configuration and control register of RX channel 0	0x0080	varies
GDMA_OUT_CONF0_CH0_REG	Configuration register 0 of TX channel 0	0x00D0	R/W
GDMA_OUT_CONF1_CH0_REG	Configuration register 1 of TX channel 0	0x00D4	R/W
GDMA_OUT_PUSH_CH0_REG	Push control register of TX channel 0	0x00DC	varies
GDMA_OUT_LINK_CH0_REG	Link descriptor configuration and control register of TX channel 0	0x00E0	varies
Status Registers			
GDMA_INFIFO_STATUS_CH0_REG	RX FIFO status of RX channel 0	0x0078	RO
GDMA_IN_STATE_CH0_REG	Receive status of RX channel 0	0x0084	RO
GDMA_IN_SUC_EOF_DES_ADDR_CH0_REG	Inlink descriptor address when EOF occurs of RX channel 0	0x0088	RO
GDMA_IN_ERR_EOF_DES_ADDR_CH0_REG	Inlink descriptor address when errors occur of RX channel 0	0x008C	RO
GDMA_IN_DSCR_CH0_REG	Current inlink descriptor address of RX channel 0	0x0090	RO
GDMA_IN_DSCR_BF0_CH0_REG	The last inlink descriptor address of RX channel 0	0x0094	RO
GDMA_IN_DSCR_BF1_CH0_REG	The second-to-last inlink descriptor address of RX channel 0	0x0098	RO
GDMA_OUTFIFO_STATUS_CH0_REG	TX FIFO status of TX channel 0	0x00D8	RO
GDMA_OUT_STATE_CH0_REG	Transmit status of TX channel 0	0x00E4	RO
GDMA_OUT_EOF_DES_ADDR_CH0_REG	Outlink descriptor address when EOF occurs of TX channel 0	0x00E8	RO
GDMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	The last outlink descriptor address when EOF occurs of TX channel 0	0x00EC	RO
GDMA_OUT_DSCR_CH0_REG	Current inlink descriptor address of TX channel 0	0x00F0	RO
GDMA_OUT_DSCR_BF0_CH0_REG	The last inlink descriptor address of TX channel 0	0x00F4	RO

Name	Description	Address	Access
GDMA_OUT_DSCR_BF1_CH0_REG	The second-to-last inlink descriptor address of TX channel 0	0x00F8	RO
Priority Registers			
GDMA_IN_PRI_CH0_REG	Priority register of RX channel 0	0x009C	R/W
GDMA_OUT_PRI_CH0_REG	Priority register of TX channel 0	0x00FC	R/W
Peripheral Select Registers			
GDMA_IN_PERI_SEL_CH0_REG	Peripheral selection of RX channel 0	0x00A0	R/W
GDMA_OUT_PERI_SEL_CH0_REG	Peripheral selection of TX channel 0	0x0100	R/W
Version Registers			
GDMA_DATE_REG	Version control register	0x0048	R/W

2.8 Registers

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

Register 2.1. GDMA_INT_RAW_CH0_REG (0x0000)



GDMA_IN_DONE_CH0_INT_RAW The raw interrupt bit turns to high level when the last data pointed by one receive descriptor has been received for RX channel 0. (R/WTC/SS)

GDMA_IN_SUC_EOF_CH0_INT_RAW The raw interrupt bit turns to high level when the last data pointed by one receive descriptor has been received for RX channel 0. (R/WTC/SS)

GDMA_IN_ERR_EOF_CH0_INT_RAW Reserved. (R/WTC/SS)

GDMA_OUT_DONE_CH0_INT_RAW The raw interrupt bit turns to high level when the last data pointed by one transmit descriptor has been transmitted to peripherals for TX channel 0. (R/WTC/SS)

GDMA_OUT_EOF_CH0_INT_RAW The raw interrupt bit turns to high level when the last data pointed by one transmit descriptor has been read from memory for TX channel 0. (R/WTC/SS)

GDMA_IN_DSCR_ERR_CH0_INT_RAW The raw interrupt bit turns to high level when detecting receive descriptor error, including owner error, the second and third word error of receive descriptor for RX channel 0. (R/WTC/SS)

GDMA_OUT_DSCR_ERR_CH0_INT_RAW The raw interrupt bit turns to high level when detecting transmit descriptor error, including owner error, the second and third word error of transmit descriptor for TX channel 0. (R/WTC/SS)

Continued on the next page...

Register 2.1. GDMA_INT_RAW_CH0_REG (0x0000)

Continued from the previous page...

GDMA_IN_DSCR_EMPTY_CH0_INT_RAW The raw interrupt bit turns to high level when RX buffer pointed by inlink is full and receiving data is not completed, but there is no more inlink for RX channel 0. (R/WTC/SS)

GDMA_OUT_TOTAL_EOF_CH0_INT_RAW The raw interrupt bit turns to high level when data corresponding to an outlink (includes one descriptor or few descriptors) is transmitted out for TX channel 0. (R/WTC/SS)

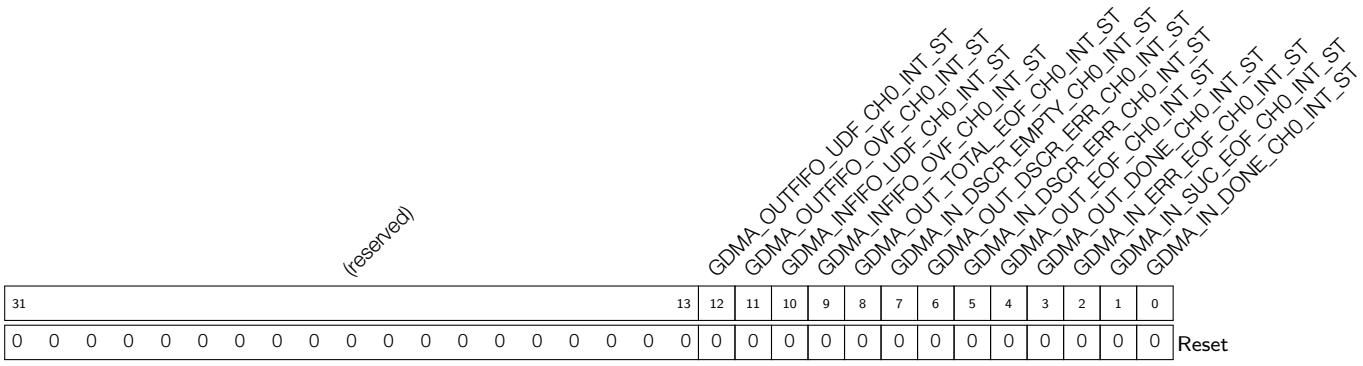
GDMA_INFIFO_OVF_CH0_INT_RAW This raw interrupt bit turns to high level when level 1 FIFO of RX channel 0 is overflow. (R/WTC/SS)

GDMA_INFIFO_UDF_CH0_INT_RAW This raw interrupt bit turns to high level when level 1 FIFO of RX channel 0 is underflow. (R/WTC/SS)

GDMA_OUTFIFO_OVF_CH0_INT_RAW This raw interrupt bit turns to high level when level 1 FIFO of TX channel 0 is overflow. (R/WTC/SS)

GDMA_OUTFIFO_UDF_CH0_INT_RAW This raw interrupt bit turns to high level when level 1 FIFO of TX channel 0 is underflow. (R/WTC/SS)

Register 2.2. GDMA_INT_ST_CH0_REG (0x0004)



GDMA_IN_DONE_CH0_INT_ST The raw interrupt status bit for the GDMA_IN_DONE_CH_INT interrupt. (RO)

GDMA_IN_SUC_EOF_CH0_INT_ST The raw interrupt status bit for the GDMA_IN_SUC_EOF_CH_INT interrupt. (RO)

GDMA_IN_ERR_EOF_CH0_INT_ST The raw interrupt status bit for the GDMA_IN_ERR_EOF_CH_INT interrupt. (RO)

GDMA_OUT_DONE_CH0_INT_ST The raw interrupt status bit for the GDMA_OUT_DONE_CH_INT interrupt. (RO)

GDMA_OUT_EOF_CH0_INT_ST The raw interrupt status bit for the GDMA_OUT_EOF_CH_INT interrupt. (RO)

GDMA_IN_DSCR_ERR_CH0_INT_ST The raw interrupt status bit for the GDMA_IN_DSCR_ERR_CH_INT interrupt. (RO)

GDMA_OUT_DSCR_ERR_CH0_INT_ST The raw interrupt status bit for the GDMA_OUT_DSCR_ERR_CH_INT interrupt. (RO)

GDMA_IN_DSCR_EMPTY_CH0_INT_ST The raw interrupt status bit for the GDMA_IN_DSCR_EMPTY_CH_INT interrupt. (RO)

GDMA_OUT_TOTAL_EOF_CH0_INT_ST The raw interrupt status bit for the GDMA_OUT_TOTAL_EOF_CH_INT interrupt. (RO)

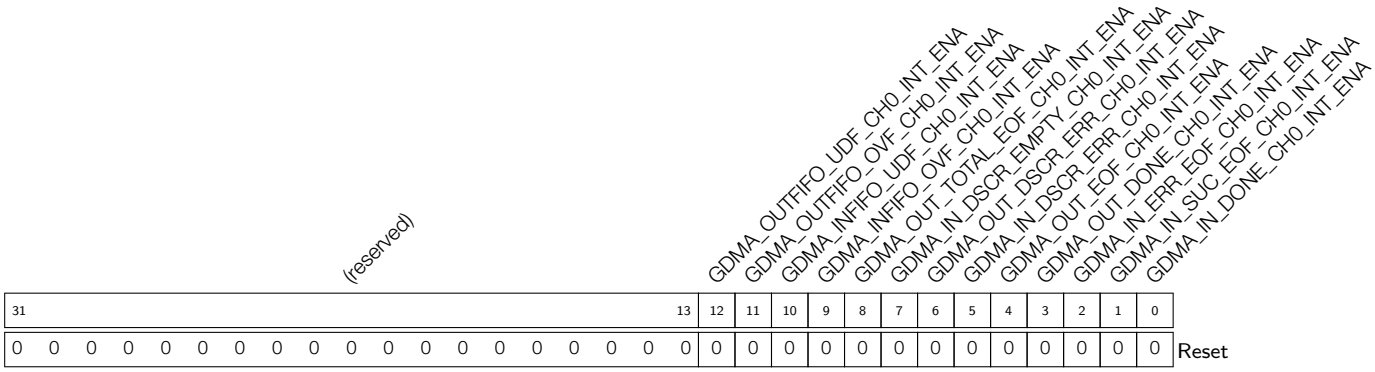
GDMA_INFIFO_OVF_CH0_INT_ST The raw interrupt status bit for the GDMA_INFIFO_OVF_L1_CH_INT interrupt. (RO)

GDMA_INFIFO_UDF_CH0_INT_ST The raw interrupt status bit for the GDMA_INFIFO_UDF_L1_CH_INT interrupt. (RO)

GDMA_OUTFIFO_OVF_CH0_INT_ST The raw interrupt status bit for the GDMA_OUTFIFO_OVF_L1_CH_INT interrupt. (RO)

GDMA_OUTFIFO_UDF_CH0_INT_ST The raw interrupt status bit for the GDMA_OUTFIFO_UDF_L1_CH_INT interrupt. (RO)

Register 2.3. GDMA_INT_ENA_CH0_REG (0x0008)



GDMA_IN_DONE_CH0_INT_ENA The interrupt enable bit for the GDMA_IN_DONE_CH_INT interrupt. (R/W)

GDMA_IN_SUC_EOF_CH0_INT_ENA The interrupt enable bit for the GDMA_IN_SUC_EOF_CH_INT interrupt. (R/W)

GDMA_IN_ERR_EOF_CH0_INT_ENA The interrupt enable bit for the GDMA_IN_ERR_EOF_CH_INT interrupt. (R/W)

GDMA_OUT_DONE_CH0_INT_ENA The interrupt enable bit for the GDMA_OUT_DONE_CH_INT interrupt. (R/W)

GDMA_OUT_EOF_CH0_INT_ENA The interrupt enable bit for the GDMA_OUT_EOF_CH_INT interrupt. (R/W)

GDMA_IN_DSCR_ERR_CH0_INT_ENA The interrupt enable bit for the GDMA_IN_DSCR_ERR_CH_INT interrupt. (R/W)

GDMA_OUT_DSCR_ERR_CH0_INT_ENA The interrupt enable bit for the GDMA_OUT_DSCR_ERR_CH_INT interrupt. (R/W)

GDMA_IN_DSCR_EMPTY_CH0_INT_ENA The interrupt enable bit for the GDMA_IN_DSCR_EMPTY_CH_INT interrupt. (R/W)

GDMA_OUT_TOTAL_EOF_CH0_INT_ENA The interrupt enable bit for the GDMA_OUT_TOTAL_EOF_CH_INT interrupt. (R/W)

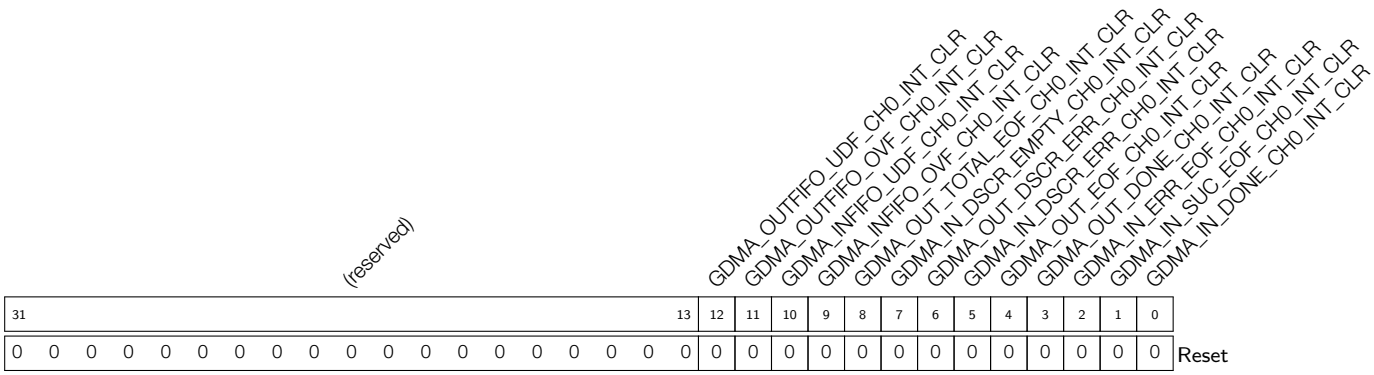
GDMA_INFIFO_OVF_CH0_INT_ENA The interrupt enable bit for the GDMA_INFIFO_OVF_L1_CH_INT interrupt. (R/W)

GDMA_INFIFO_UDF_CH0_INT_ENA The interrupt enable bit for the GDMA_INFIFO_UDF_L1_CH_INT interrupt. (R/W)

GDMA_OUTFIFO_OVF_CH0_INT_ENA The interrupt enable bit for the GDMA_OUTFIFO_OVF_L1_CH_INT interrupt. (R/W)

GDMA_OUTFIFO_UDF_CH0_INT_ENA The interrupt enable bit for the GDMA_OUTFIFO_UDF_L1_CH_INT interrupt. (R/W)

Register 2.4. GDMA_INT_CLR_CH0_REG (0x000C)



GDMA_IN_DONE_CH0_INT_CLR Set this bit to clear the GDMA_IN_DONE_CH_INT interrupt. (WT)

GDMA_IN_SUC_EOF_CH0_INT_CLR Set this bit to clear the GDMA_IN_SUC_EOF_CH_INT interrupt. (WT)

GDMA_IN_ERR_EOF_CH0_INT_CLR Set this bit to clear the GDMA_IN_ERR_EOF_CH_INT interrupt. (WT)

GDMA_OUT_DONE_CH0_INT_CLR Set this bit to clear the GDMA_OUT_DONE_CH_INT interrupt. (WT)

GDMA_OUT_EOF_CH0_INT_CLR Set this bit to clear the GDMA_OUT_EOF_CH_INT interrupt. (WT)

GDMA_IN_DSCR_ERR_CH0_INT_CLR Set this bit to clear the GDMA_IN_DSCR_ERR_CH_INT interrupt. (WT)

GDMA_OUT_DSCR_ERR_CH0_INT_CLR Set this bit to clear the GDMA_OUT_DSCR_ERR_CH_INT interrupt. (WT)

GDMA_IN_DSCR_EMPTY_CH0_INT_CLR Set this bit to clear the GDMA_IN_DSCR_EMPTY_CH_INT interrupt. (WT)

GDMA_OUT_TOTAL_EOF_CH0_INT_CLR Set this bit to clear the GDMA_OUT_TOTAL_EOF_CH_INT interrupt. (WT)

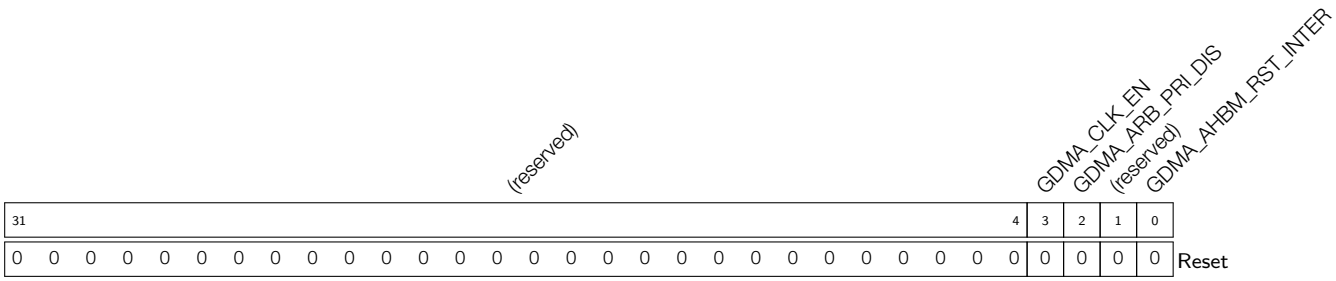
GDMA_INFIFO_OVF_CH0_INT_CLR Set this bit to clear the GDMA_INFIFO_OVF_L1_CH_INT interrupt. (WT)

GDMA_INFIFO_UDF_CH0_INT_CLR Set this bit to clear the GDMA_INFIFO_UDF_L1_CH_INT interrupt. (WT)

GDMA_OUTFIFO_OVF_CH0_INT_CLR Set this bit to clear the GDMA_OUTFIFO_OVF_L1_CH_INT interrupt. (WT)

GDMA_OUTFIFO_UDF_CH0_INT_CLR Set this bit to clear the GDMA_OUTFIFO_UDF_L1_CH_INT interrupt. (WT)

Register 2.5. GDMA_MISC_CONF_REG (0x0044)

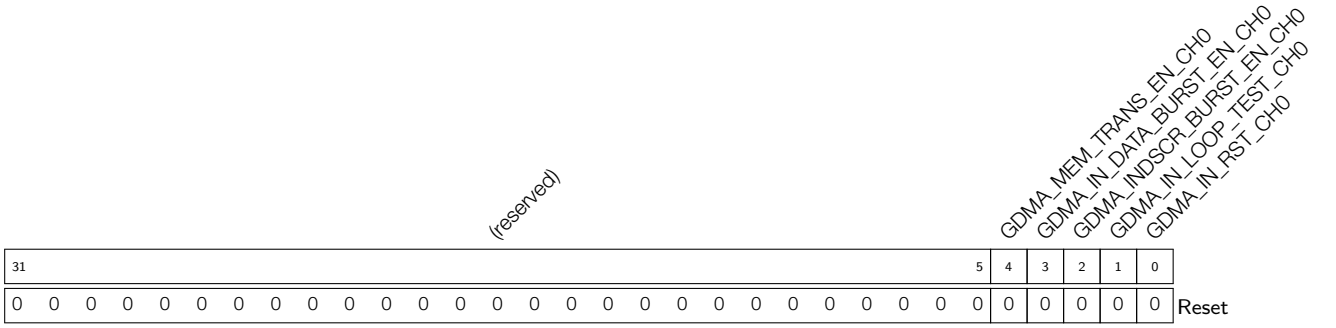


GDMA_AHBM_RST_INTER Set this bit, then clear this bit to reset the internal ahb FSM. (R/W)

GDMA_ARB_PRI_DIS Set this bit to disable priority arbitration function. (R/W)

GDMA_CLK_EN 0: Enable the clock only when application writes registers. 1: Force the clock on for registers. (R/W)

Register 2.6. GDMA_IN_CONF0_CH0_REG (0x0070)



GDMA_IN_RST_CH0 This bit is used to reset GDMA channel 0 RX FSM and RX FIFO pointer. (R/W)

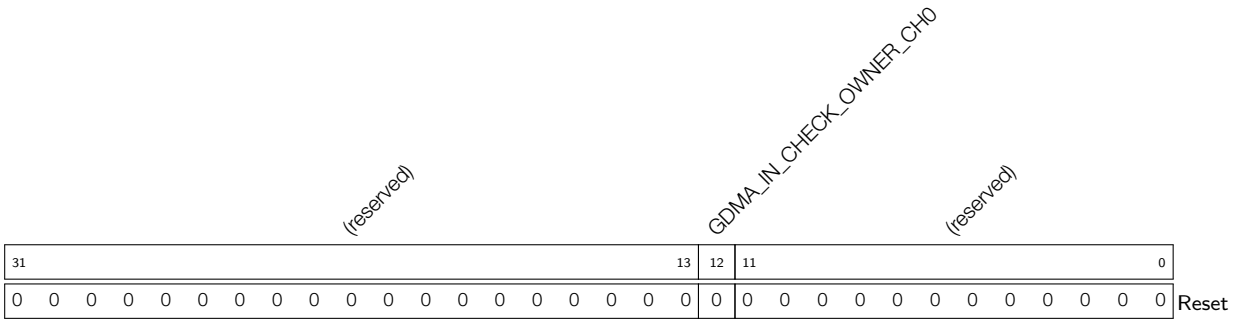
GDMA_IN_LOOP_TEST_CH0 Reserved. (R/W)

GDMA_INDSCR_BURST_EN_CH0 Set this bit to 1 to enable INCR burst transfer for RX channel 0 reading descriptor when accessing internal RAM. (R/W)

GDMA_IN_DATA_BURST_EN_CH0 Set this bit to 1 to enable INCR burst transfer for RX channel 0 receiving data when accessing internal RAM. (R/W)

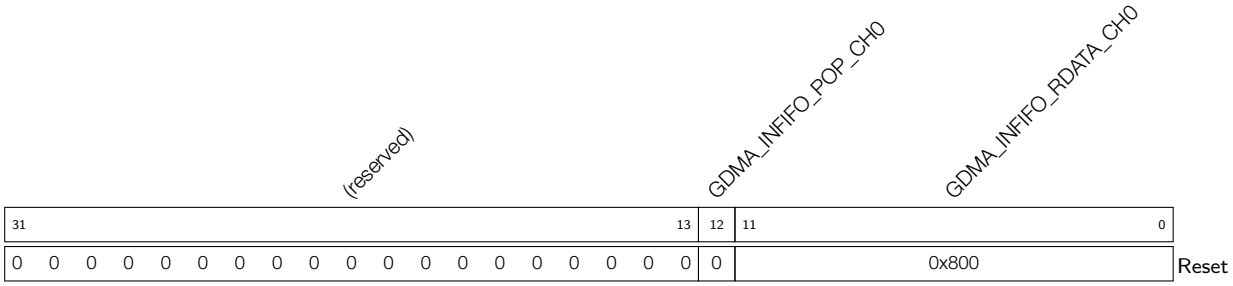
GDMA_MEM_TRANS_EN_CH0 Set this bit 1 to enable automatic transmitting data from memory to memory via GDMA. (R/W)

Register 2.7. GDMA_IN_CONF1_CH0_REG (0x0074)



GDMA_IN_CHECK_OWNER_CH0 Set this bit to enable checking the owner attribute of the descriptor. (R/W)

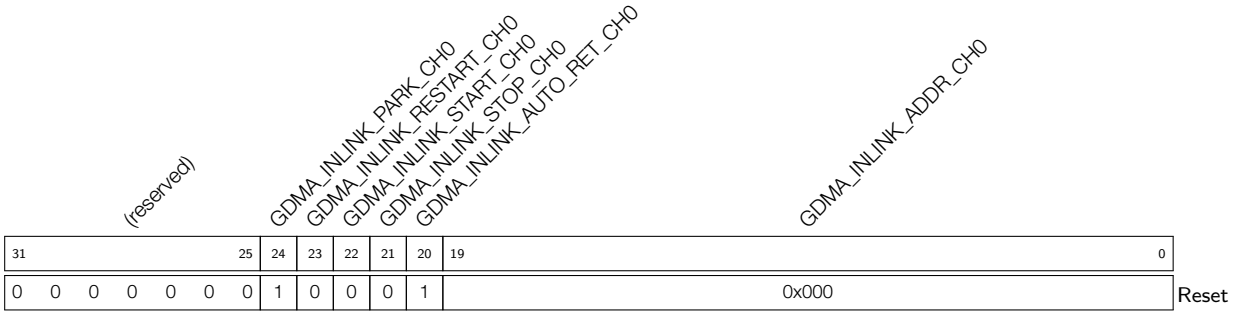
Register 2.8. GDMA_IN_POP_CH0_REG (0x007C)



GDMA_INFIFO_RDATA_CH0 This register stores the data popping from GDMA FIFO (intended for debugging). (RO)

GDMA_INFIFO_POP_CH0 Set this bit to pop data from GDMA FIFO (intended for debugging). (R/W/SC)

Register 2.9. GDMA_IN_LINK_CH0_REG (0x0080)



GDMA_INLINK_ADDR_CH0 This register stores the 20 least significant bits of the first receive descriptor's address. (R/W)

GDMA_INLINK_AUTO_RET_CH0 Set this bit to return to current receive descriptor's address, when there are some errors in current receiving data. (R/W)

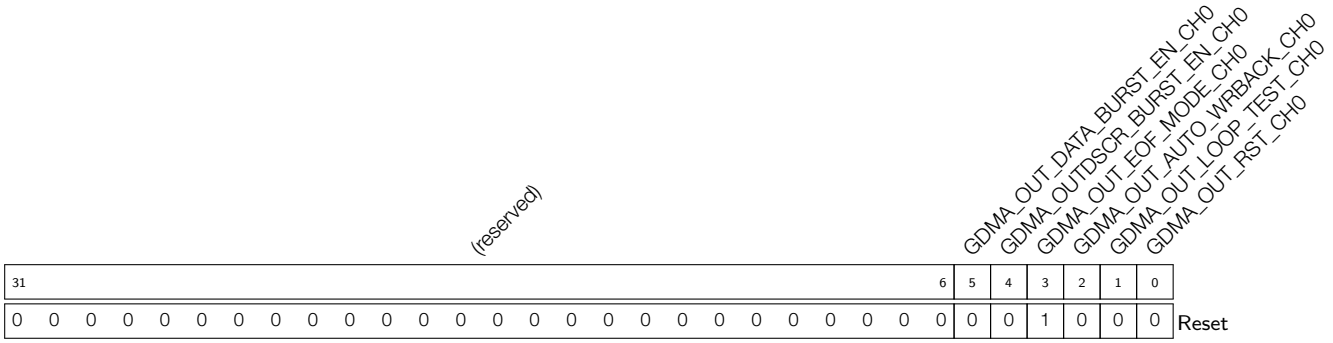
GDMA_INLINK_STOP_CH0 Set this bit to stop GDMA's receive channel from receiving data. (R/W/SC)

GDMA_INLINK_START_CH0 Set this bit to enable GDMA's receive channel for data transfer. (R/W/SC)

GDMA_INLINK_RESTART_CH0 Set this bit to mount a new receive descriptor. (R/W/SC)

GDMA_INLINK_PARK_CH0 1: the receive descriptor's FSM is in idle state. 0: the receive descriptor's FSM is working. (RO)

Register 2.10. GDMA_OUT_CONF0_CH0_REG (0x00D0)



GDMA_OUT_RST_CH0 This bit is used to reset GDMA channel 0 TX FSM and TX FIFO pointer. (R/W)

GDMA_OUT_LOOP_TEST_CH0 Reserved. (R/W)

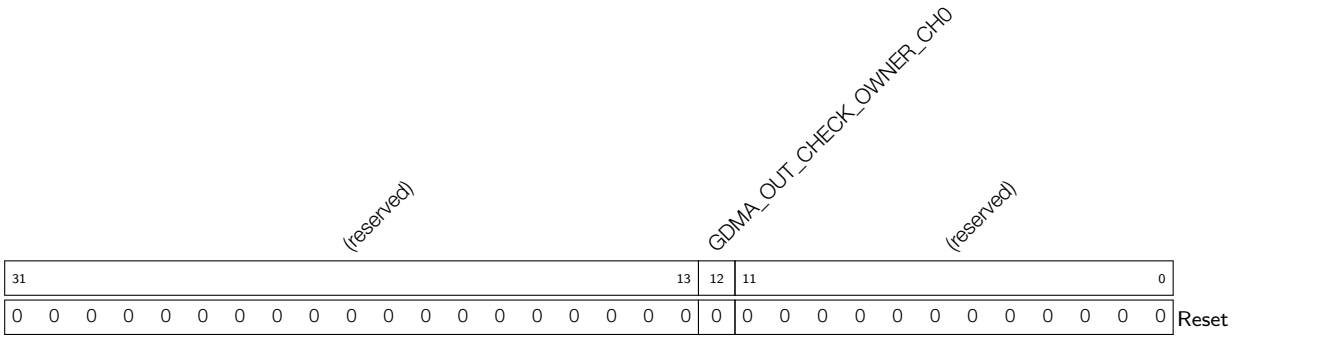
GDMA_OUT_AUTO_WRBACK_CH0 Set this bit to enable automatic outlink-writeback when all the data in TX buffer has been transmitted. (R/W)

GDMA_OUT_EOF_MODE_CH0 EOF flag generation mode when transmitting data. 1: EOF flag for TX channel 0 is generated when data need to transmit has been popped from FIFO in GDMA. (R/W)

GDMA_OUTDSCR_BURST_EN_CH0 Set this bit to 1 to enable INCR burst transfer for TX channel 0 reading descriptor when accessing internal RAM. (R/W)

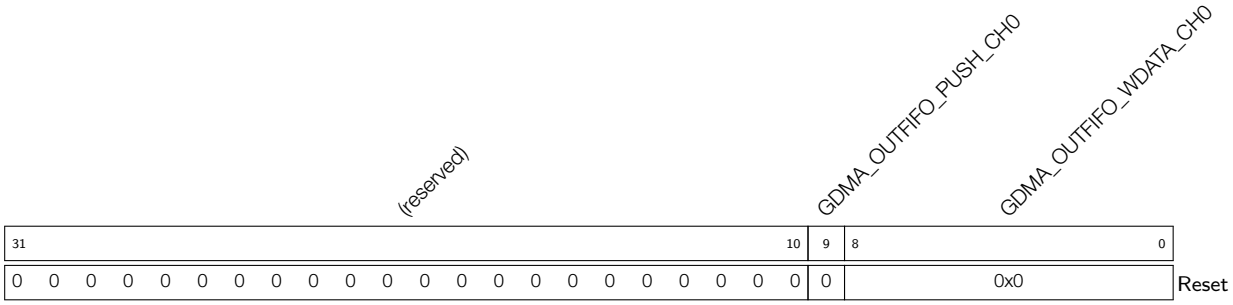
GDMA_OUT_DATA_BURST_EN_CH0 Set this bit to 1 to enable INCR burst transfer for TX channel 0 transmitting data when accessing internal RAM. (R/W)

Register 2.11. GDMA_OUT_CONF1_CH0_REG (0x00D4)



GDMA_OUT_CHECK_OWNER_CH0 Set this bit to enable checking the owner attribute of the descriptor. (R/W)

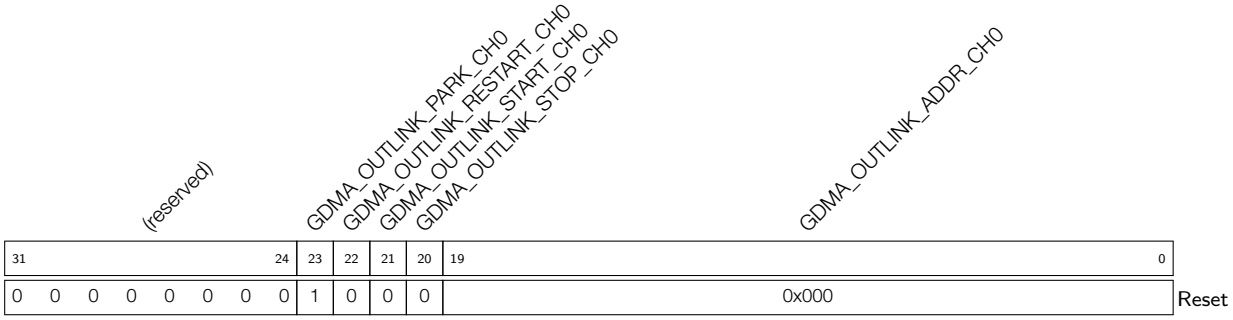
Register 2.12. GDMA_OUT_PUSH_CH0_REG (0x00DC)



GDMA_OUTFIFO_WDATA_CH0 This register stores the data that need to be pushed into GDMA FIFO. (R/W)

GDMA_OUTFIFO_PUSH_CH0 Set this bit to push data into GDMA FIFO. (R/W/SC)

Register 2.13. GDMA_OUT_LINK_CH0_REG (0x00E0)



GDMA_OUTLINK_ADDR_CH0 This register stores the 20 least significant bits of the first transmit descriptor's address. (R/W)

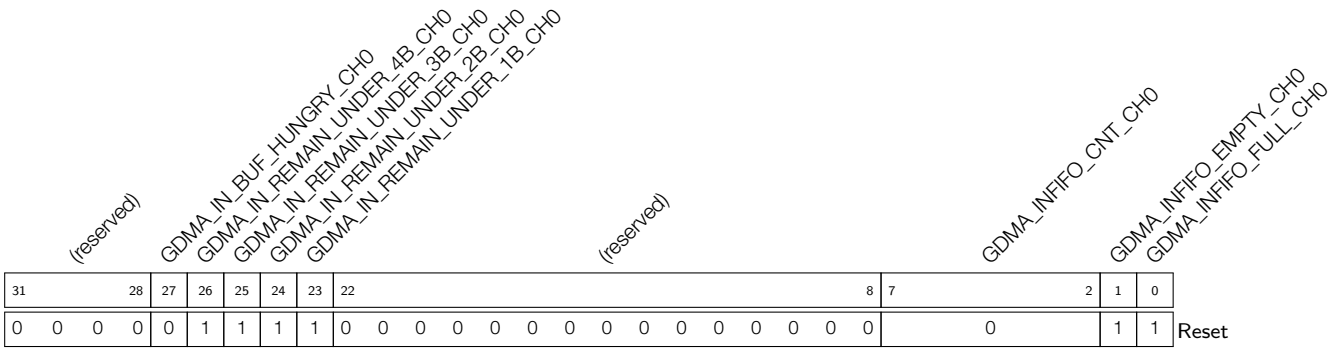
GDMA_OUTLINK_STOP_CH0 Set this bit to stop GDMA's receive channel from receiving data. (R/W/SC)

GDMA_OUTLINK_START_CH0 Set this bit to enable GDMA's transmit channel for data transfer. (R/W/SC)

GDMA_OUTLINK_RESTART_CH0 Set this bit to restart a new outlink from the last address. (R/W/SC)

GDMA_OUTLINK_PARK_CH0 1: the transmit descriptor's FSM is in idle state. 0: the transmit descriptor's FSM is working. (RO)

Register 2.14. GDMA_INFIFO_STATUS_CH0_REG (0x0078)



GDMA_INFIFO_FULL_CH0 L1 RX FIFO full signal for RX channel 0. (RO)

GDMA_INFIFO_EMPTY_CH0 L1 RX FIFO empty signal for RX channel 0. (RO)

GDMA_INFIFO_CNT_CH0 The register stores the byte number of the data in L1 RX FIFO for RX channel 0. (RO)

GDMA_IN_REMAIN_UNDER_1B_CH0 Reserved. (RO)

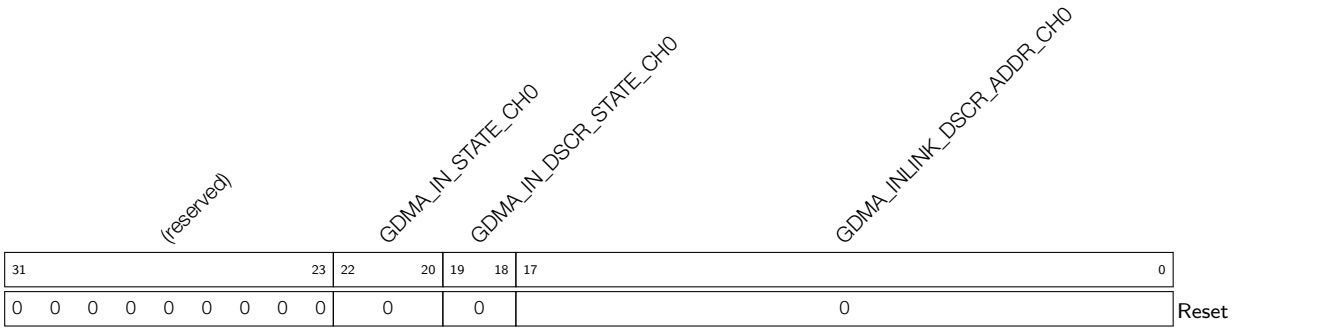
GDMA_IN_REMAIN_UNDER_2B_CH0 Reserved. (RO)

GDMA_IN_REMAIN_UNDER_3B_CH0 Reserved. (RO)

GDMA_IN_REMAIN_UNDER_4B_CH0 Reserved. (RO)

GDMA_IN_BUF_HUNGRY_CH0 Reserved. (RO)

Register 2.15. GDMA_IN_STATE_CH0_REG (0x0084)

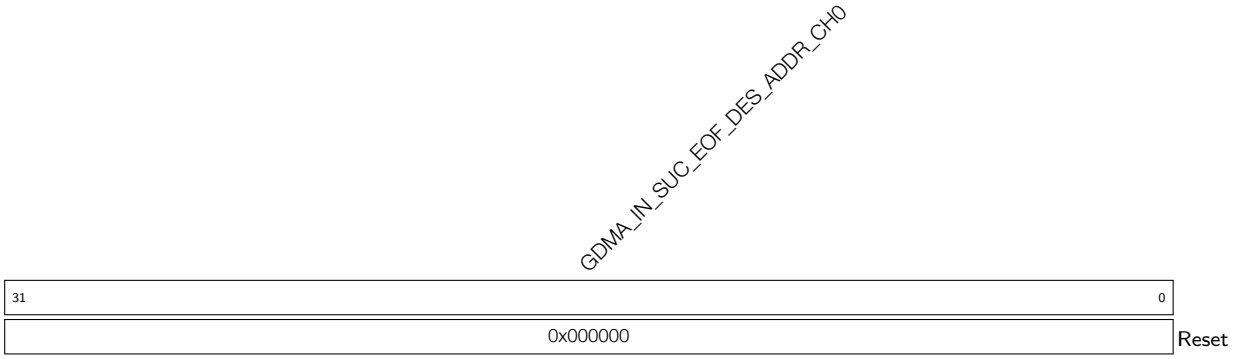


GDMA_INLINK_DSCR_ADDR_CH0 This register stores the current receive descriptor's address. (RO)

GDMA_IN_DSCR_STATE_CH0 Reserved. (RO)

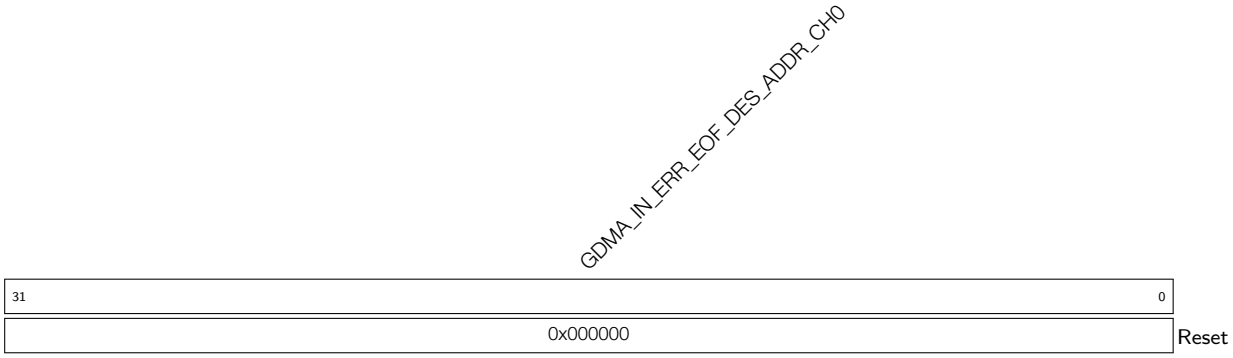
GDMA_IN_STATE_CH0 Reserved. (RO)

Register 2.16. GDMA_IN_SUC_EOF_DES_ADDR_CH0_REG (0x0088)



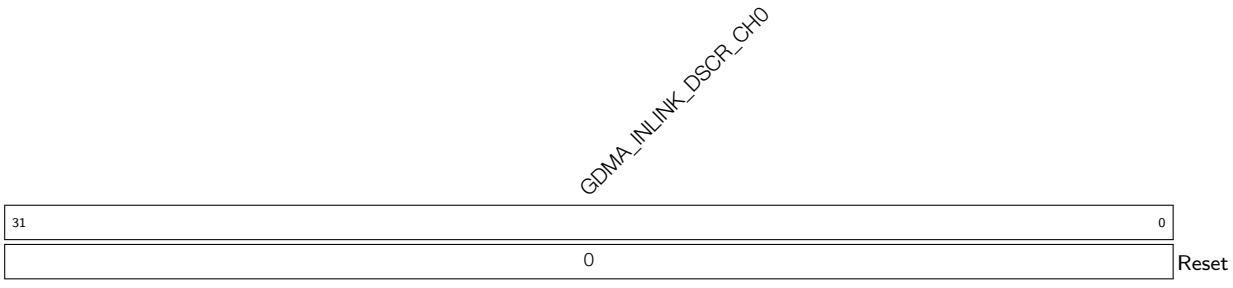
GDMA_IN_SUC_EOF_DES_ADDR_CH0 This register stores the address of the receive descriptor when the EOF bit in this descriptor is 1. (RO)

Register 2.17. GDMA_IN_ERR_EOF_DES_ADDR_CH0_REG (0x008C)



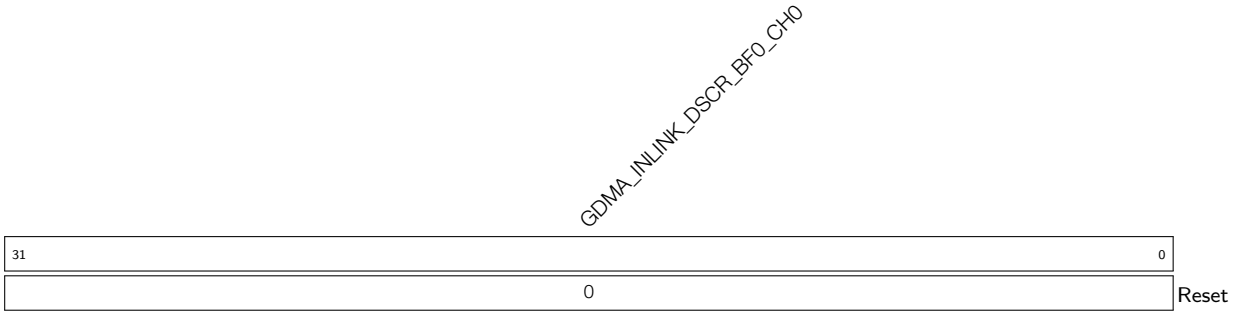
GDMA_IN_ERR_EOF_DES_ADDR_CH0 Reserved. (RO)

Register 2.18. GDMA_IN_DSCR_CH0_REG (0x0090)



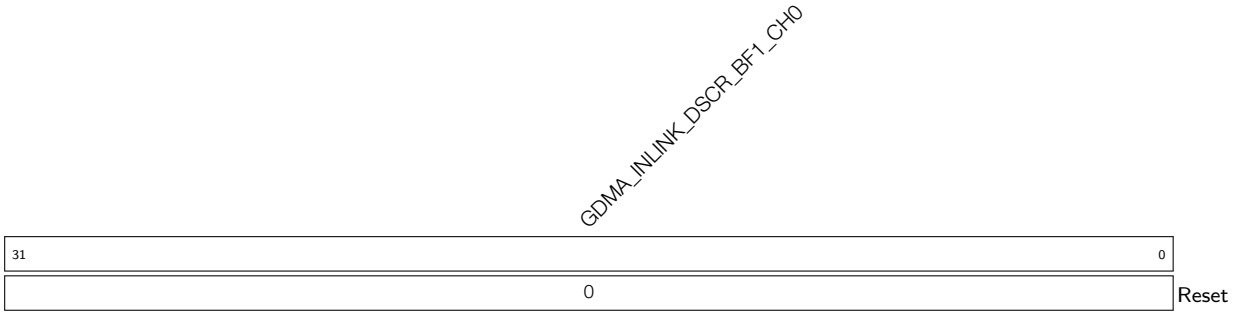
GDMA_INLINK_DSCR_CH0 The address of the current receive descriptor x. (RO)

Register 2.19. GDMA_IN_DSCR_BF0_CH0_REG (0x0094)



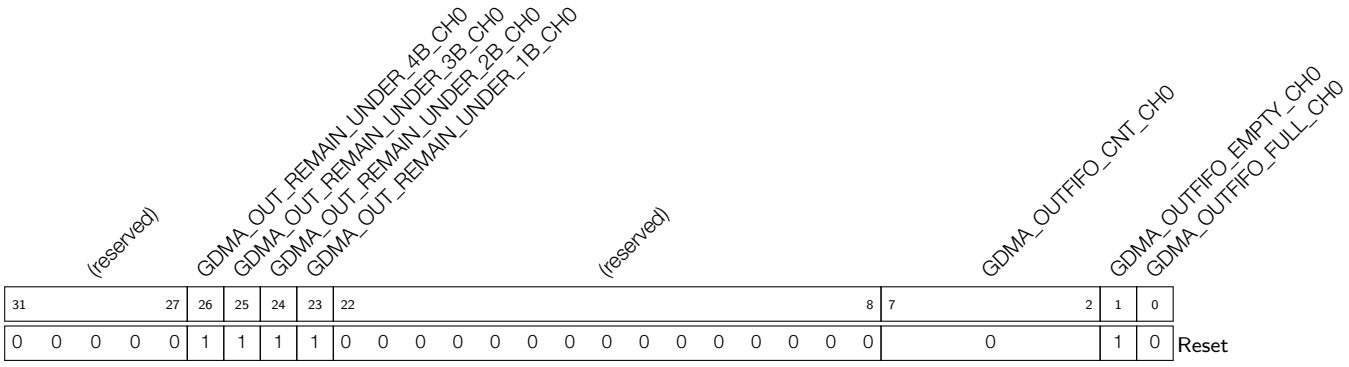
GDMA_INLINK_DSCR_BF0_CH0 The address of the last receive descriptor x-1. (RO)

Register 2.20. GDMA_IN_DSCR_BF1_CH0_REG (0x0098)



GDMA_INLINK_DSCR_BF1_CH0 The address of the second-to-last receive descriptor x-2. (RO)

Register 2.21. GDMA_OUTFIFO_STATUS_CH0_REG (0x00D8)



GDMA_OUTFIFO_FULL_CH0 L1 TX FIFO full signal for TX channel 0. (RO)

GDMA_OUTFIFO_EMPTY_CH0 L1 TX FIFO empty signal for TX channel 0. (RO)

GDMA_OUTFIFO_CNT_CH0 The register stores the byte number of the data in L1 TX FIFO for TX channel 0. (RO)

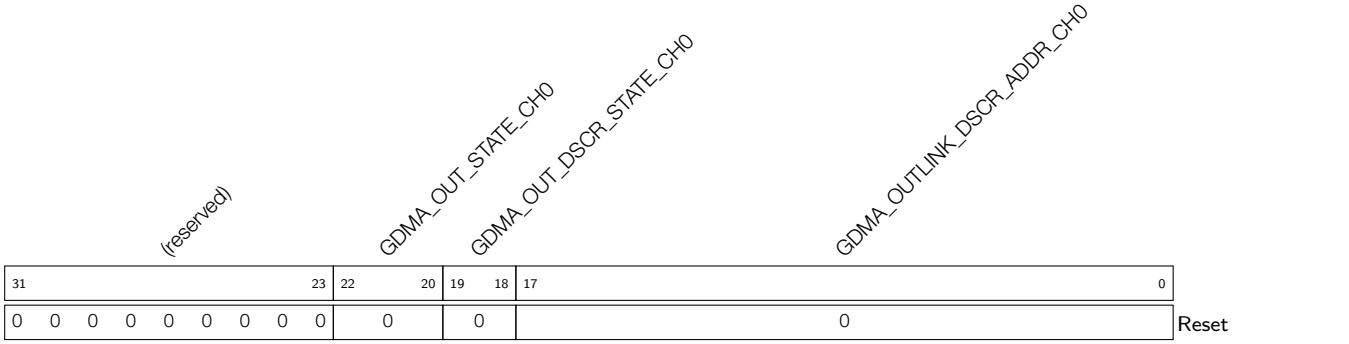
GDMA_OUT_REMAIN_UNDER_1B_CH0 Reserved. (RO)

GDMA_OUT_REMAIN_UNDER_2B_CH0 Reserved. (RO)

GDMA_OUT_REMAIN_UNDER_3B_CH0 Reserved. (RO)

GDMA_OUT_REMAIN_UNDER_4B_CH0 Reserved. (RO)

Register 2.22. GDMA_OUT_STATE_CH0_REG (0x00E4)

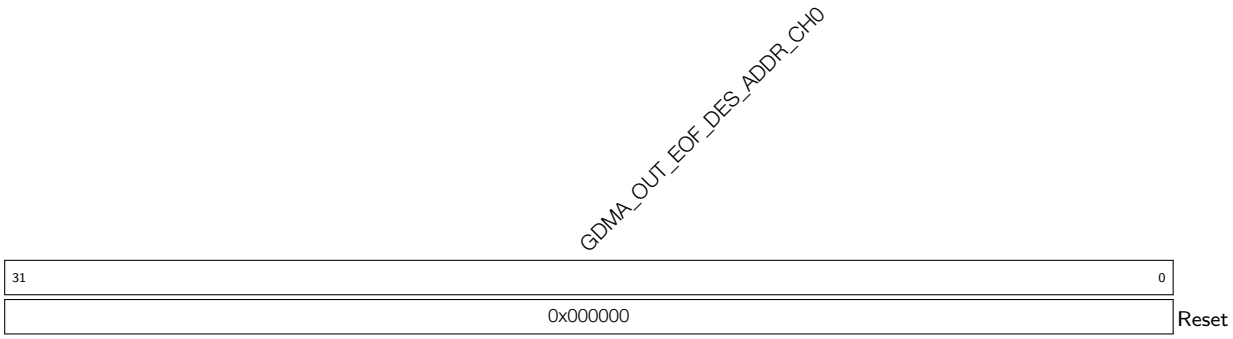


GDMA_OUTLINK_DSCR_ADDR_CH0 This register stores the current transmit descriptor's address. (RO)

GDMA_OUT_DSCR_STATE_CH0 Reserved. (RO)

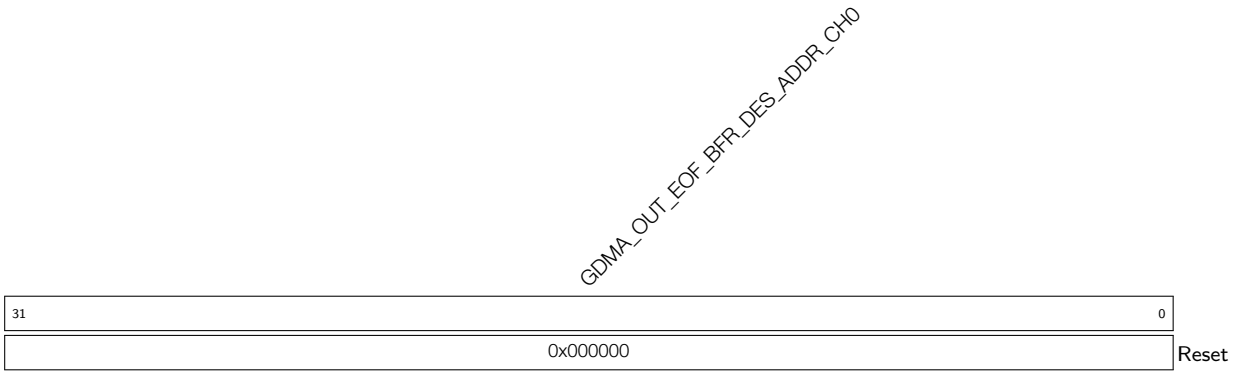
GDMA_OUT_STATE_CH0 Reserved. (RO)

Register 2.23. GDMA_OUT_EOF_DES_ADDR_CH0_REG (0x00E8)



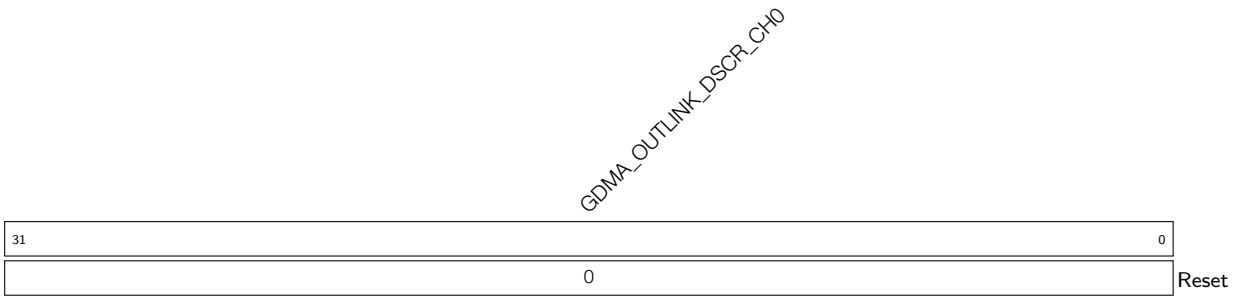
GDMA_OUT_EOF_DES_ADDR_CH0 This register stores the address of the transmit descriptor when the EOF bit in this descriptor is 1. (RO)

Register 2.24. GDMA_OUT_EOF_BFR_DES_ADDR_CH0_REG (0x00EC)



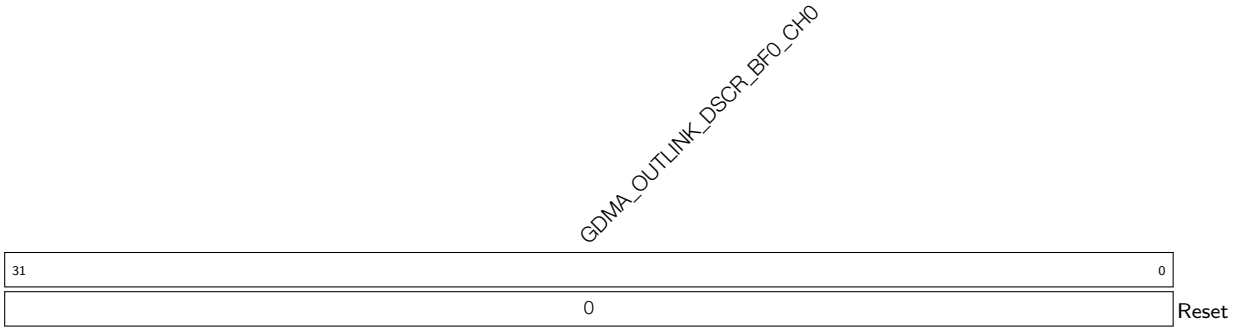
GDMA_OUT_EOF_BFR_DES_ADDR_CH0 This register stores the address of the transmit descriptor before the last transmit descriptor. (RO)

Register 2.25. GDMA_OUT_DSCR_CH0_REG (0x00F0)



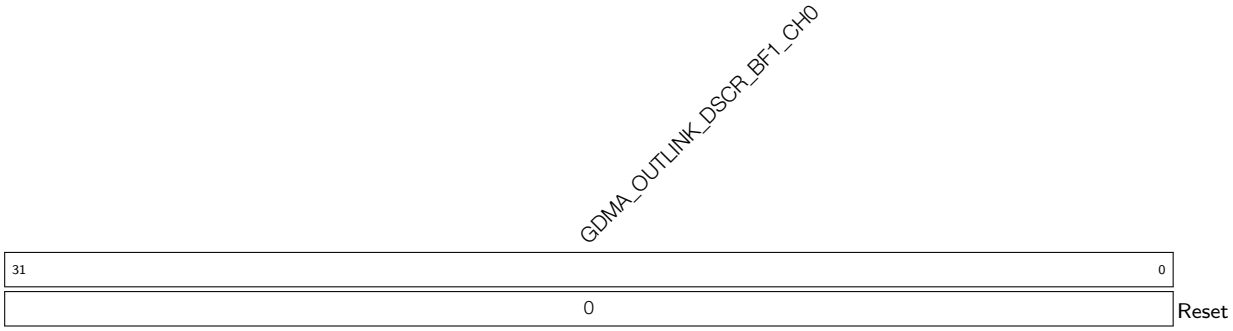
GDMA_OUTLINK_DSCR_CH0 The address of the current transmit descriptor y. (RO)

Register 2.26. GDMA_OUT_DSCR_BF0_CH0_REG (0x00F4)



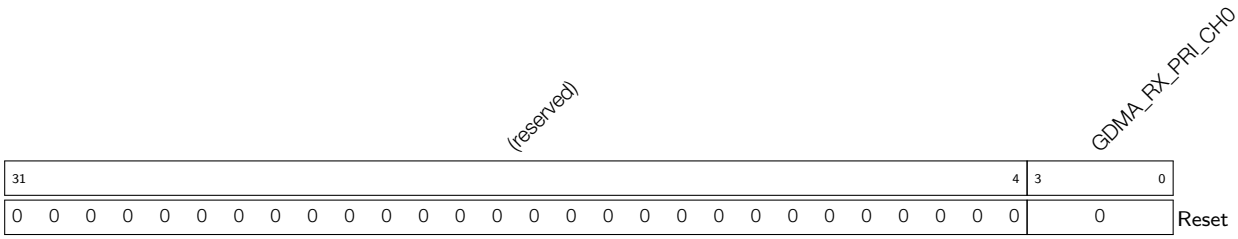
GDMA_OUTLINK_DSCR_BF0_CH0 The address of the last transmit descriptor y-1. (RO)

Register 2.27. GDMA_OUT_DSCR_BF1_CH0_REG (0x00F8)



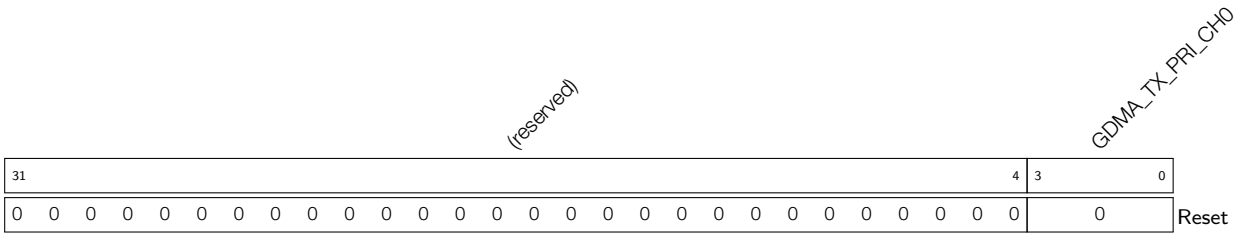
GDMA_OUTLINK_DSCR_BF1_CH0 The address of the second-to-last receive descriptor x-2. (RO)

Register 2.28. GDMA_IN_PRI_CH0_REG (0x009C)



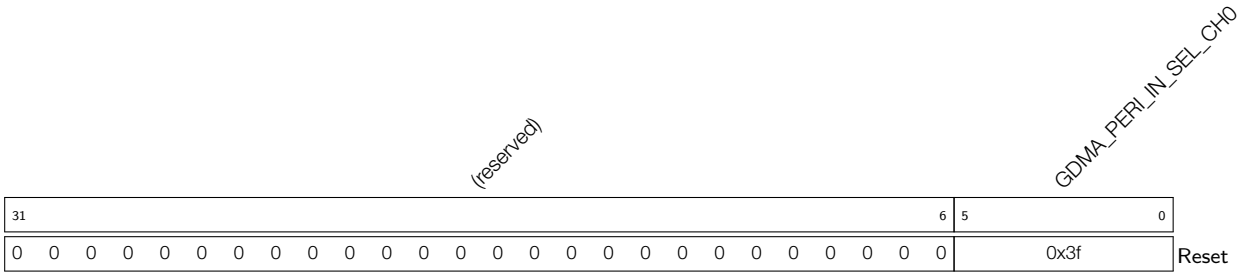
GDMA_RX_PRI_CH0 The priority of RX channel 0. The larger the value, the higher the priority. (R/W)

Register 2.29. GDMA_OUT_PRI_CH0_REG (0x00FC)



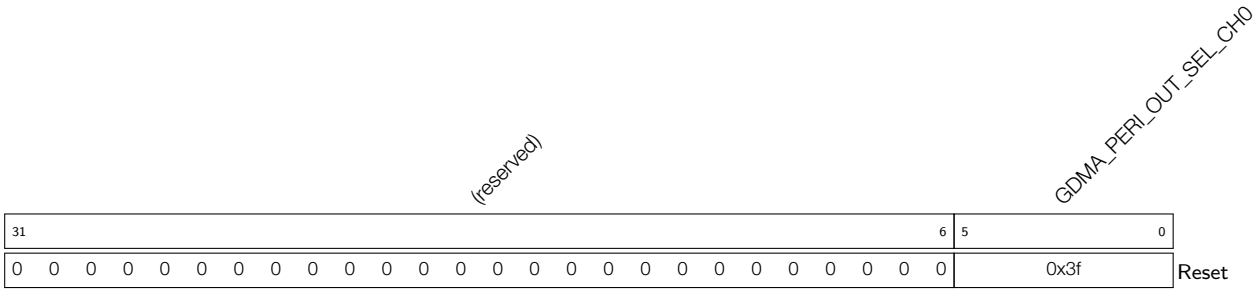
GDMA_TX_PRI_CH0 The priority of TX channel 0. The larger the value, the higher the priority. (R/W)

Register 2.30. GDMA_IN_PERI_SEL_CH0_REG (0x00A0)



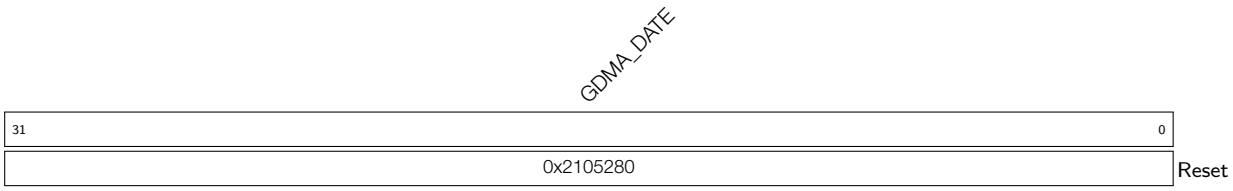
GDMA_PERI_IN_SEL_CH0 This register is used to select peripheral for RX channel 0. 0: SPI2. 1: reserved. 2: reserved. 3: reserved. 4: reserved. 5: reserved. 6: reserved. 7: reserved. 8: reserved. (R/W)

Register 2.31. GDMA_OUT_PERI_SEL_CH0_REG (0x0100)



GDMA_PERI_OUT_SEL_CH0 This register is used to select peripheral for TX channel 0. 0:SPI2. 1: reserved. 2: reserved. 3: reserved. 4: reserved. 5: reserved. 6: reserved. 7: SHA. 8: reserved. (R/W)

Register 2.32. GDMA_DATE_REG (0x0048)



GDMA_DATE This is the version control register. (R/W)

3 System and Memory

3.1 Overview

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

3.2 Features

ESP8684's system and memory has the following features:

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

Figure 3-1 illustrates the system structure and address mapping.

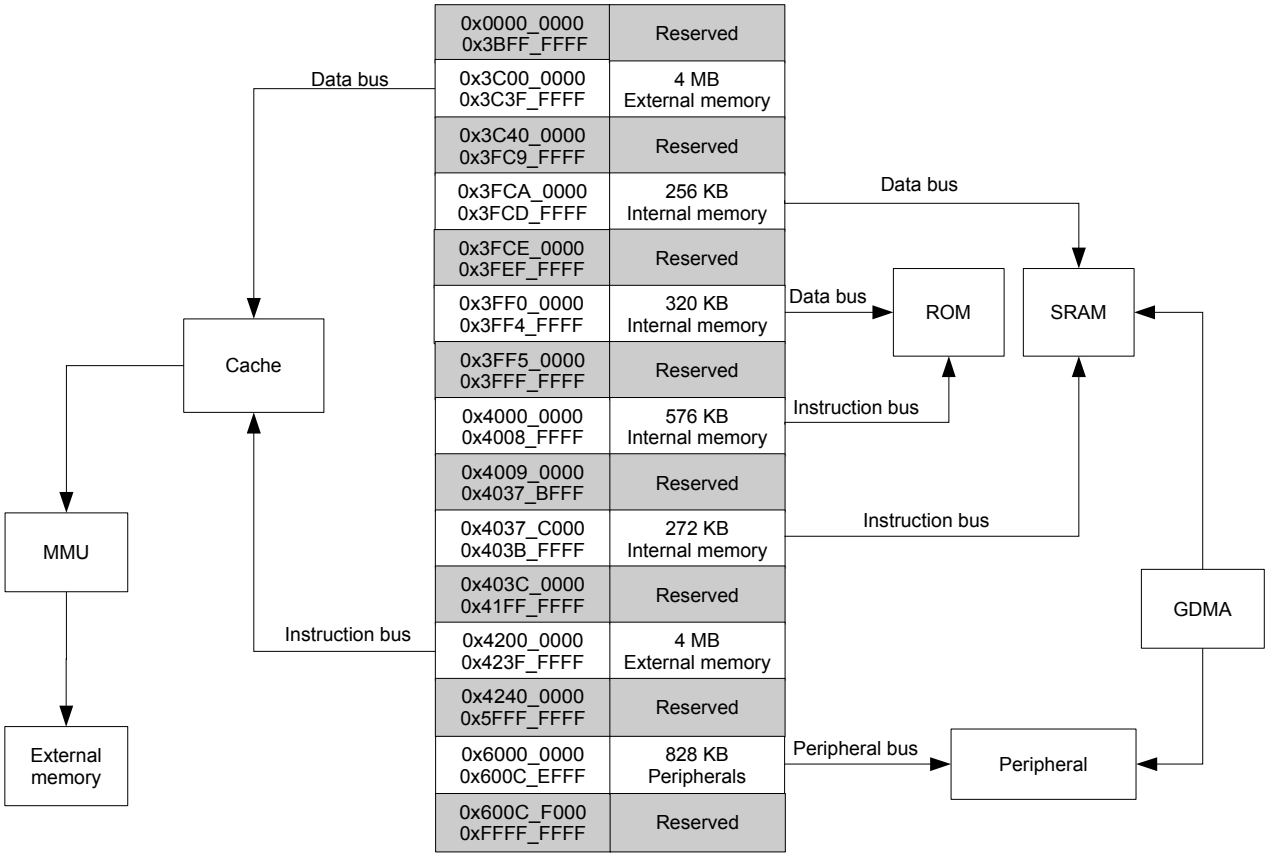


Figure 3-1. System Structure and Address Mapping

Note:

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

3.3 Functional Description

3.3.1 Address Mapping

Addresses below 0x4000_0000 are accessed using the data bus. Addresses in the range of 0x4000_0000 ~ 0x4FFF_FFFF are accessed using the instruction bus. Addresses over and including 0x5000_0000 are accessed using the peripheral bus.

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

The CPU can:

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

- directly access modules/peripherals via the peripheral bus.

Figure 3-1 shows the address ranges on the data bus, instruction bus and peripheral bus as well as their corresponding target memory.

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

3.3.2 Internal Memory

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

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

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

However, within each segment, there may be different bus access restrictions (e.g., some parts of the segment may only be accessible by the CPU's data bus). Therefore, segments are also further divided into parts. Table 3-1 describes each part of internal memory and their address ranges on the data bus and instruction bus.

Table 3-1. Internal Memory Address Mapping

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

1. Internal ROM 0

Internal ROM 0 is a 256 KB, read-only memory space, addressed by the CPU only through the instruction bus via 0x4000_0000 ~ 0x4003_FFFF, as shown in Table 3-1.

2. Internal ROM 1

Internal ROM 1 is a 320 KB, read-only memory space, addressed by the CPU through the instruction bus via 0x4004_0000 ~ 0x4008_FFFF or through the data bus via 0x3FF0_0000 ~ 0x3FF4_FFFF in the same order, as shown in Table 3-1.

This means, for example, address 04004_0000 and 0x3FF0_0000 correspond to the same word, 0x4004_0004

and 0x3FF0_0004 correspond to the same word, 0x4004_0008 and 0x3FF0_0008 correspond to the same word, etc.

3. Internal SRAM 0

Internal SRAM 0 is a 16 KB, read-and-write memory space, addressed by the CPU through the instruction bus via the range described in Table 3-1.

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

4. Internal SRAM 1

Internal SRAM 1 is a 256 KB, read-and-write memory space, addressed by the CPU through the data bus or instruction bus, in the same order (the same meaning as the explanation in 3.3.2 *Internal ROM 1*), via the ranges described in Table 3-1.

3.3.3 External Memory

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

3.3.3.1 External Memory Address Mapping

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

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

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

Table 3-2 lists the mapping between the cache and the corresponding address ranges on the data bus and instruction bus.

Table 3-2. External Memory Address Mapping

Bus Type	Boundary Address		Size (MB)	Target
	Low Address	High Address		
Data bus (read-only)	0x3C00_0000	0x3C3F_FFFF	4	Uniform Cache
Instruction bus	0x4200_0000	0x423F_FFFF	4	Uniform Cache

3.3.3.2 Cache

As shown in Figure 3-2, ESP8684 has a read-only uniform cache which is eight-way set-associative, its size is 16 KB and its block size is 32 bytes. When cache is active, some internal memory space will be occupied by cache (see Internal SRAM 0 in Section 3.3.2).

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

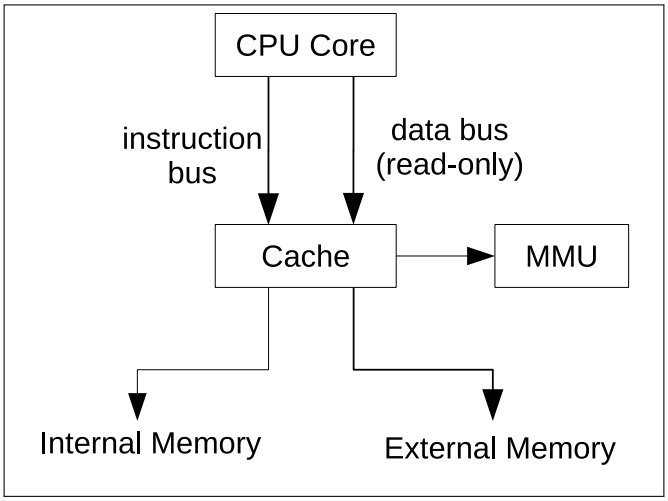


Figure 3-2. Cache Structure

3.3.3.3 Cache Operations

ESP8684 cache supports the following operations:

1. **Invalidate**: This operation is used to clear valid data in the cache. After this operation is completed, the data will only be stored in the external memory. The CPU needs to access the external memory in order to read this data. There are two types of invalidate-operation: automatic invalidation (Auto-Invalidate) and manual invalidation (Manual-Invalidate). Manual-Invalidate is performed only on data in the specified area in the cache, while Auto-Invalidate is performed on all data in the cache.
2. **Preload**: This operation is used to load instructions and data into the cache in advance. The minimum unit of preload-operation is one block (32 bytes). There are two types of preload-operation: manual preload (Manual-Preload) and automatic preload (Auto-Preload). Manual-Preload means that the hardware prefetches a piece of continuous data according to the virtual address specified by the software. Auto-Preload means the hardware prefetches a piece of continuous data according to the current address where the cache hits or misses (depending on configuration).
3. **Lock/Unlock**: The lock operation is used to prevent the data in the cache from being easily replaced. There are two types of lock: prelock and manual lock. When prelock is enabled, the cache locks the data in the specified area when filling the missing data to cache memory, while the data outside the specified area will not be locked. When manual lock is enabled, the cache checks the data that is already in the cache memory and only locks the data in the specified area, and leaves the data outside the specified area unlocked. When there are missing data, the cache will replace the data in the unlocked way first, so the data in the locked way is always stored in the cache and will not be replaced. But when all ways within the cache are locked, the cache will replace data, as if it was not locked. Unlocking is the reverse of locking, except that it only can be done manually.

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

3.3.4 GDMA Address Space

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

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

The GDMA can read and write to Internal SRAM 1 via the same address as the data bus. Specifically, GDMA accesses Internal SRAM 1 via 0x3FCA_0000 ~ 0x3FCD_FFFF. Note that GDMA cannot access the internal memory occupied by the cache.

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

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

3.3.5 Modules/Peripherals

The CPU can access modules/peripherals via 0x6000_0000 ~ 0x600C_EFFF shared by the peripheral bus.

3.3.5.1 Module/Peripheral Address Mapping

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

Table 3-3. Module/Peripheral Address Mapping

Target	Boundary Address		Size (KB)	Notes
	Low Address	High Address		
UART Controller 0	0x6000_0000	0x6000_0FFF	4	
Reserved	0x6000_1000	0x6000_1FFF		
SPI Controller 1	0x6000_2000	0x6000_2FFF	4	
SPI Controller 0	0x6000_3000	0x6000_3FFF	4	
GPIO	0x6000_4000	0x6000_4FFF	4	
Reserved	0x6000_5000	0x6000_7FFF		
Low-Power Management	0x6000_8000	0x6000_8FFF	4	
IO MUX	0x6000_9000	0x6000_9FFF	4	
Reserved	0x6000_A000	0x6000_CFFF		
MISC	0x6000_D000	0x6000_DFFF	4	
Reserved	0x6000_E000	0x6000_FFFF		
UART Controller 1	0x6001_0000	0x6001_0FFF	4	
Reserved	0x6001_1000	0x6001_2FFF		
I2C Controller	0x6001_3000	0x6001_3FFF	4	
Reserved	0x6001_4000	0x6001_8FFF		
LED PWM Controller	0x6001_9000	0x6001_9FFF	4	

Table 3-3 –

Target	Boundary Address		Size (KB)	Notes
	Low Address	High Address		
Reserved	0x6001_A000	0x6001_EFFF		
Timer Group 0	0x6001_F000	0x6001_FFFF	4	
Reserved	0x6002_0000	0x6002_2FFF		
System Timer	0x6002_3000	0x6002_3FFF	4	
SPI Controller 2	0x6002_4000	0x6002_4FFF	4	
Reserved	0x6002_5000	0x6002_5FFF		
APB Controller	0x6002_6000	0x6002_6FFF	4	
Reserved	0x6002_7000	0x6003_AFFF		
SHA Accelerator	0x6003_B000	0x6003_BFFF	4	
ECC Accelerator	0x6003_E000	0x6003_EFFF	4	
Reserved	0x6002_C000	0x6003_EFFF		
GDMA Controller	0x6003_F000	0x6003_FFFF	4	
ADC Controller	0x6004_0000	0x6004_0FFF	4	
Reserved	0x6004_1000	0x600B_FFFF		
System Registers	0x600C_0000	0x600C_0FFF	4	
Sensitive Register	0x600C_1000	0x600C_1FFF	4	
Interrupt Matrix	0x600C_2000	0x600C_2FFF	4	
Reserved	0x600C_3000	0x600C_3FFF		
Configure Cache	0x600C_4000	0x600C_DFFF	40	
Reserved	0x600C_E000	0x600C_DFFF		
Debug Assist	0x600C_E000	0x600C_EFFF	4	

4 eFuse Controller (eFuse)

4.1 Overview

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

4.2 Features

The eFuse controller has the following features:

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

4.3 Functional Description

4.3.1 Structure

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

BLOCK0 holds control parameters for most hardware modules.

Table 4-1 lists all the parameters in BLOCK0 that can be accessed (read and used) by users and their offsets, bit widths, as well as information on whether they can be used by hardware, which bits are write-protected, and corresponding descriptions.

The **EFUSE_WR_DIS** parameter is used to control the writing of other parameters, while **EFUSE_RD_DIS** is used to disable users from reading BLOCK3. For more information on these two parameters, please see Section 4.3.1.1 and Section 4.3.1.2.

Table 4-1. Parameters in BLOCK0

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

Table 4-2 lists parameter information stored in BLOCK1 ~ BLOCK3.

Table 4-2. Parameters in BLOCK1 to BLOCK3

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

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

4.3.1.1 EFUSE_WR_DIS

Parameter EFUSE_WR_DIS determines whether individual eFuse parameters are write-protected. After EFUSE_WR_DIS has been programmed, execute an eFuse read operation so that the write-protection status would take effect.

Column "Write Protection by EFUSE_WR_DIS Bit Number" in Table 4-1 and Table 4-2 lists the specific bits in EFUSE_WR_DIS that disable writing.

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

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

4.3.1.2 EFUSE_RD_DIS

Only parameters in BLOCK3 can be read-protected to prevent any access from outside of the chip as shown in column "Read Protection by EFUSE_RD_DIS Bit Number" of Table 4-2. After EFUSE_RD_DIS has been programmed, execute an eFuse read operation so that the read-protection status would take effect.

If the corresponding EFUSE_RD_DIS bit is 0, then the eFuse block can be read by users; if the corresponding EFUSE_RD_DIS bit is 1, then the parameter controlled by this bit is user protected.

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

4.3.1.3 Data Storage

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

All BLOCK0 parameters except for EFUSE_WR_DIS are stored with four backups, meaning each bit is stored four times. This backup scheme is not visible to users.

Except for EFUSE_WR_DIS which is 8-bit, all other parameters in BLOCK0 are 32-bit. Therefore, BLOCK0 occupies (8 + 32 * 4 = 136) bits of storage totally in eFuse memory.

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

The shift register circuit shown in Figure 4-1 and 4-2 processes 32-byte data using RS (44, 32). This coding scheme encodes 32 bytes of data into 44 bytes:

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

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

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

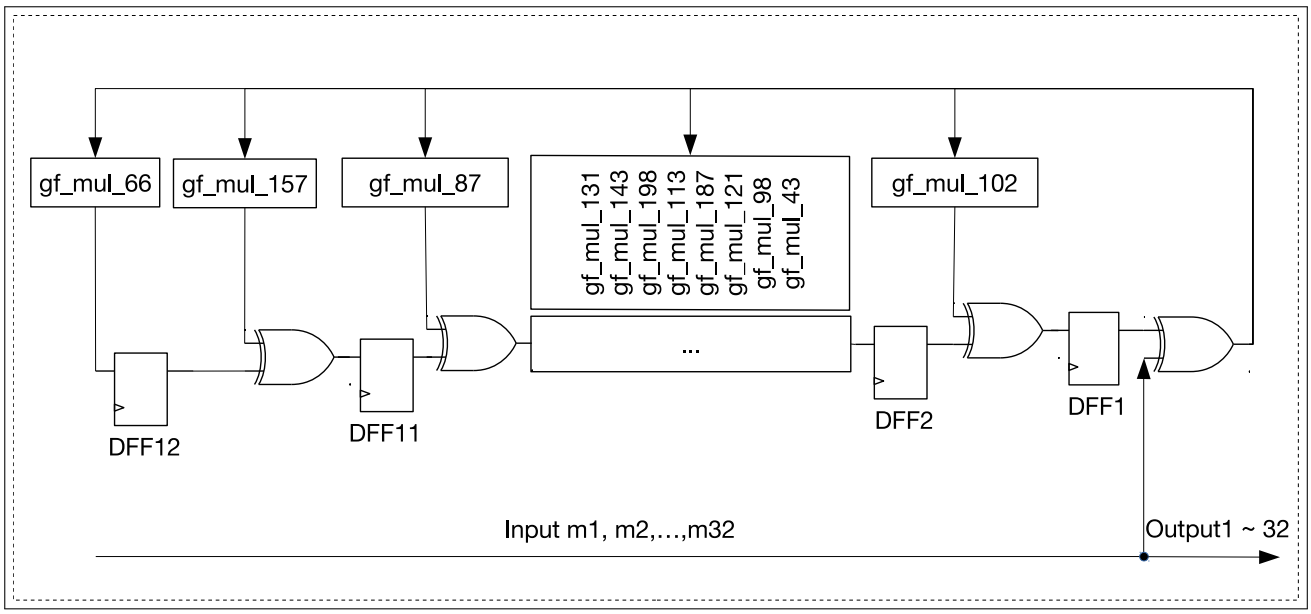


Figure 4-1. Shift Register Circuit (former 32-byte)

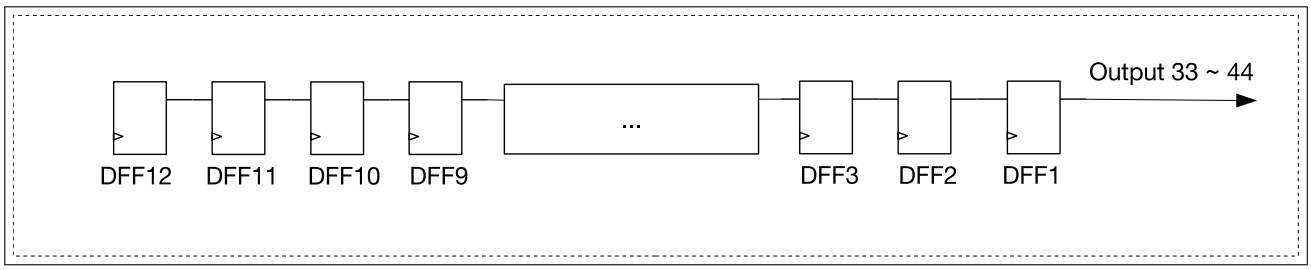


Figure 4-2. Shift Register Circuit (latter 12-byte)

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

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

4.3.2 Programming of Parameters

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

Programming BLOCK0

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

Programming BLOCK1

Set the EFUSE_BLK_NUM field to 1. The parameters to be programmed in BLOCK1 are stored in EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA2_REG, while the corresponding RS check codes are stored in EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG. The data in EFUSE_PGM_DATA3_REG ~ EFUSE_PGM_DATA7_REG registers does not affect the programming of BLOCK1.

Programming BLOCK2 ~ 3

Set the EFUSE_BLK_NUM field to 2 or 3, respectively. The parameters to be programmed are stored in EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG, while the corresponding RS check code is stored in EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG.

Programming process

The process of programming parameters is as follows:

1. Configure the value of parameter EFUSE_BLK_NUM to determine the block to be programmed.
2. Write the parameters to be programmed to registers EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG.
3. Make sure the eFuse programming voltage VDDQ is configured correctly as described in Section 4.3.4.
4. Set the field EFUSE_OP_CODE of register EFUSE_CONF_REG to 0x5A5A.
5. Set the field EFUSE_PGM_CMD of register EFUSE_CMD_REG to 1.
6. Poll register EFUSE_CMD_REG until it is 0x0, or wait for a PGM_DONE interrupt. For more information on how to identify a PGM/READ_DONE interrupt, please see the end of Section 4.3.3.
7. Clear the parameters in EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG.
8. Trigger an eFuse read operation (see Section 4.3.3) to update eFuse registers with the new values.
9. Check corresponding error registers. If the value read is not 0, above 1 ~ 7 steps should be executed again to prevent programming insufficiency. For different eFuse blocks, the corresponding error registers that need to be checked are listed as follows:
   - BLOCK0: EFUSE_RD_REPEAT_ERR_REG
   - BLOCK1: EFUSE_RD_RS_ERR_REG[3:0]
   - BLOCK2: EFUSE_RD_RS_ERR_REG[7:4]
   - BLOCK3: EFUSE_RD_RS_ERR_REG[11:8]

Restrictions

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

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

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

4.3.3 User Read of Parameters

Users cannot read data programmed in the eFuse controller directly. The eFuse controller reads all programmed data and stores the results to their corresponding programming registers in its memory space. Then, users can read eFuse bits by reading the registers that start with EFUSE_RD_. Details are provided in Table 4-3.

Table 4-3. Registers information

BLOCK	Read Registers	Programming Registers
0	EFUSE_RD_WR_DIS_REG	EFUSE_PGM_DATA0_REG
0	EFUSE_RD_REPEAT_DATA0_REG	EFUSE_PGM_DATA1_REG
1	EFUSE_RD_BLK1_DATA0 ~ 2_REG	EFUSE_PGM_DATA0 ~ 2_REG
2	EFUSE_RD_BLK2_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
3	EFUSE_RD_BLK3_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG

Updating eFuse controller read registers

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

1. Set the field EFUSE_OP_CODE of register EFUSE_CONF_REG to 0x5AA5.
2. Set the field EFUSE_READ_CMD of register EFUSE_CMD_REG to 1.
3. Poll register EFUSE_CMD_REG until it is 0x0, or wait for a READ_DONE interrupt. Information on how to identify a PGM/READ_DONE interrupt is provided below in this section.
4. Read the values of each parameter from eFuse memory.

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

Error detection

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

The EFUSE_RD_REPEAT_ERR_REG register indicates if there are any errors of programmed parameters (except for EFUSE_WR_DIS) in BLOCK0 (value 1 indicates an error is detected, and the bit becomes invalid; value 0 indicates no error).

The EFUSE_RD_RS_ERR_REG register stores the number of corrected bytes as well as the result of RS decoding during eFuse reading BLOCK1 ~ BLOCK3.

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

Identifying programming/read operation

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

- Method 1:

1. Poll bit 1/0 in register EFUSE_INT_RAW_REG until it becomes 1, which represents the completion of a program/read operation.

- Method 2:

1. Set bit 1/0 in register EFUSE_INT_ENA_REG to 1 to enable the eFuse controller to post a PGM/READ_DONE interrupt.
2. Configure the Interrupt Matrix to enable the CPU to respond to eFuse controller interrupt signals, see Chapter 8 *Interrupt Matrix (INTMTRX)*.
3. Wait for the PGM/READ_DONE interrupt.
4. Set bit 1/0 in register EFUSE_INT_CLR_REG to 1 to clear the PGM/READ_DONE interrupt.

Attention

When the eFuse controller updating registers, the EFUSE_PGM_DATA*n*_REG (n=0 1 .., 7) register will be re-used. Therefore, please do not write meaningful data to such register before the eFuse controller starts to update registers.

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

4.3.4 eFuse VDDQ Timing

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

- EFUSE_DAC_NUM (the rising period of VDDQ): The default value of VDDQ is 2.5 V and the voltage increases by 0.01 V in each clock cycle. Thus, the default value of this parameter is 255.
- EFUSE_DAC_CLK_DIV (the clock divisor of VDDQ): The clock period to program VDDQ should be larger than 1 μs.
- EFUSE_PWR_ON_NUM (the power-up time for VDDQ): The programming voltage should be stabilized after this time, which means the value of this parameter should be configured to exceed the result of EFUSE_DAC_CLK_DIV times EFUSE_DAC_NUM.
- EFUSE_PWR_OFF_NUM (the power-down time for VDDQ): The value of this parameter should be larger than 10 μs.

Table 4-4. Configuration of Default VDDQ Timing Parameters

EFUSE_DAC_NUM	EFUSE_DAC_CLK_DIV	EFUSE_PWR_ON_NUM	EFUSE_PWR_OFF_NUM
0xFF	0x28	0x3000	0x190

4.3.5 Parameters Used by Hardware Modules

Some hardware modules are directly connected to the eFuse peripheral in order to use the parameters listed in Table 4-1 and Table 4-2, specifically those marked with "Y" in columns "Accessible by Hardware". Users cannot intervene in this process.

4.3.6 Interrupts

- PGM_DONE interrupt: Triggered when eFuse programming has finished. To enable this interrupt, set the EFUSE_PGM_DONE_INT_ENA field of register EFUSE_INT_ENA_REG to 1.
- READ_DONE interrupt: Triggered when eFuse read has finished. To enable this interrupt, set the EFUSE_READ_DONE_INT_ENA field of register EFUSE_INT_ENA_REG to 1.

4.4 Register Summary

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

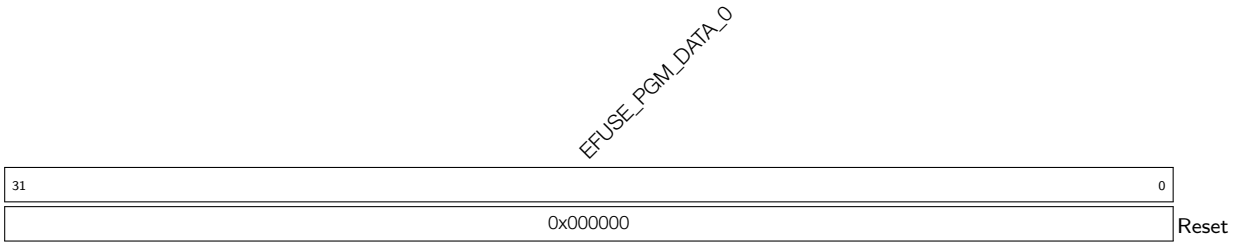
Name	Description	Address	Access
PGM Data Register			
EFUSE_PGM_DATA0_REG	Register 0 that configures data to be programmed	0x0000	R/W
EFUSE_PGM_DATA1_REG	Register 1 that configures data to be programmed	0x0004	R/W
EFUSE_PGM_DATA2_REG	Register 2 that configures data to be programmed	0x0008	R/W
EFUSE_PGM_DATA3_REG	Register 3 that configures data to be programmed	0x000C	R/W
EFUSE_PGM_DATA4_REG	Register 4 that configures data to be programmed	0x0010	R/W
EFUSE_PGM_DATA5_REG	Register 5 that configures data to be programmed	0x0014	R/W
EFUSE_PGM_DATA6_REG	Register 6 that configures data to be programmed	0x0018	R/W
EFUSE_PGM_DATA7_REG	Register 7 that configures data to be programmed	0x001C	R/W
EFUSE_PGM_CHECK_VALUE0_REG	Register 0 that configures the RS code to be programmed	0x0020	R/W
EFUSE_PGM_CHECK_VALUE1_REG	Register 1 that configures the RS code to be programmed	0x0024	R/W
EFUSE_PGM_CHECK_VALUE2_REG	Register 2 that configures the RS code to be programmed	0x0028	R/W
Read Data Register			
EFUSE_RD_WR_DIS_REG	Register 0 of BLOCK0 wr_dis data	0x002C	RO
EFUSE_RD_REPEAT_DATA0_REG	Register 1 of BLOCK0 data	0x0030	RO
EFUSE_RD_BLK1_DATA0_REG	Register 0 of BLOCK1 data	0x0034	RO
EFUSE_RD_BLK1_DATA1_REG	Register 1 of BLOCK1 data	0x0038	RO
EFUSE_RD_BLK1_DATA2_REG	Register 2 of BLOCK1 data	0x003C	RO
EFUSE_RD_BLK2_DATA0_REG	Register 0 of BLOCK2 data	0x0040	RO
EFUSE_RD_BLK2_DATA1_REG	Register 1 of BLOCK2 data	0x0044	RO
EFUSE_RD_BLK2_DATA2_REG	Register 2 of BLOCK2 data	0x0048	RO
EFUSE_RD_BLK2_DATA3_REG	Register 3 of BLOCK2 data	0x004C	RO
EFUSE_RD_BLK2_DATA4_REG	Register 4 of BLOCK2 data	0x0050	RO
EFUSE_RD_BLK2_DATA5_REG	Register 5 of BLOCK2 data	0x0054	RO
EFUSE_RD_BLK2_DATA6_REG	Register 6 of BLOCK2 data	0x0058	RO
EFUSE_RD_BLK2_DATA7_REG	Register 7 of BLOCK2 data	0x005C	RO
EFUSE_RD_BLK3_DATA0_REG	Register 0 of BLOCK3 data	0x0060	RO
EFUSE_RD_BLK3_DATA1_REG	Register 1 of BLOCK3 data	0x0064	RO
EFUSE_RD_BLK3_DATA2_REG	Register 2 of BLOCK3 data	0x0068	RO

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

4.5 Registers

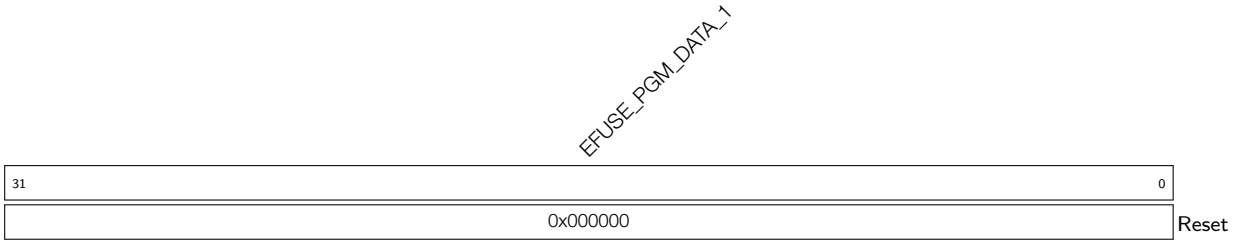
The addresses in this section are relative to eFuse controller base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Register 4.1. EFUSE_PGM_DATA0_REG (0x0000)



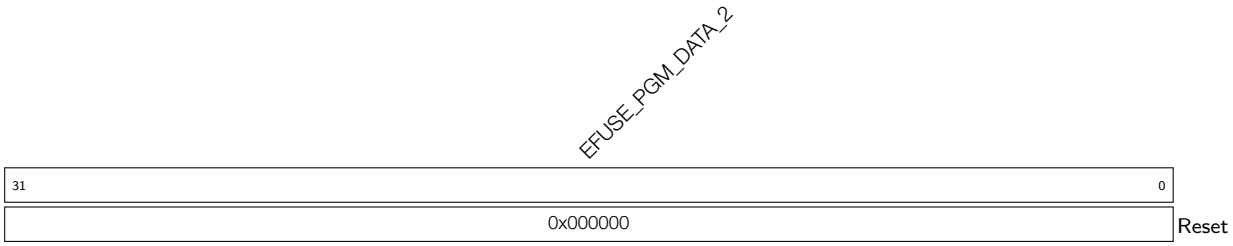
EFUSE_PGM_DATA_0 Configures the content of the 0th 32-bit data to be programmed. (R/W)

Register 4.2. EFUSE_PGM_DATA1_REG (0x0004)



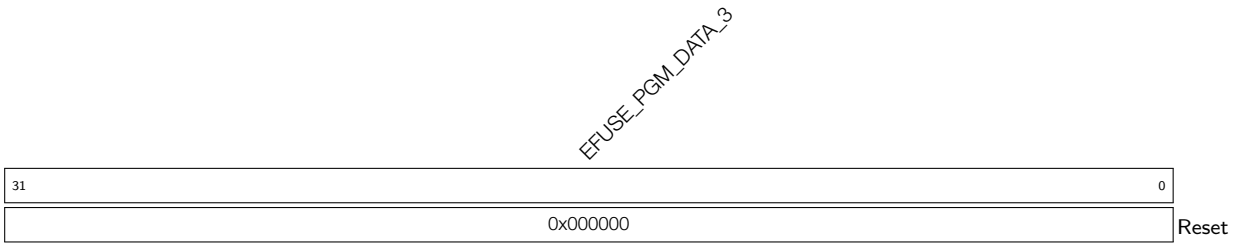
EFUSE_PGM_DATA_1 Configures the content of the 1st 32-bit data to be programmed. (R/W)

Register 4.3. EFUSE_PGM_DATA2_REG (0x0008)



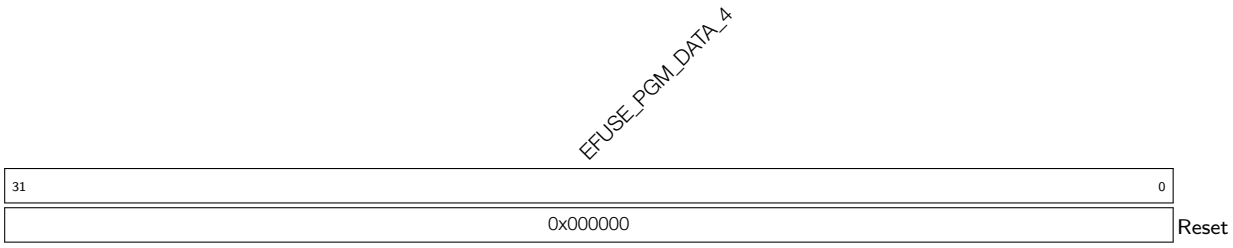
EFUSE_PGM_DATA_2 Configures the content of the 2nd 32-bit data to be programmed. (R/W)

Register 4.4. EFUSE_PGM_DATA3_REG (0x000C)



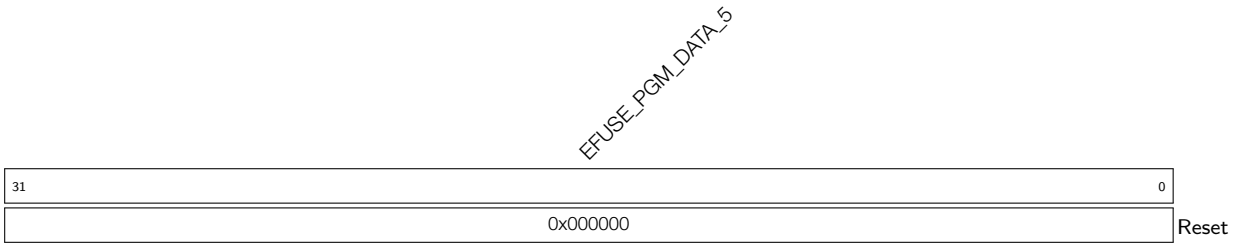
EFUSE_PGM_DATA_3 Configures the content of the 3rd 32-bit data to be programmed. (R/W)

Register 4.5. EFUSE_PGM_DATA4_REG (0x0010)



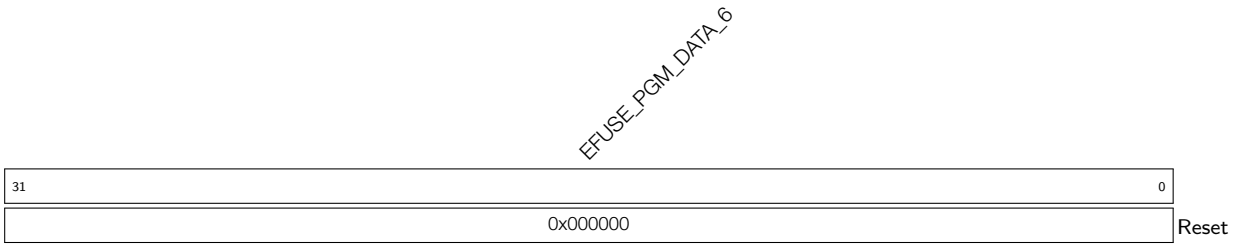
EFUSE_PGM_DATA_4 Configures the content of the 4th 32-bit data to be programmed. (R/W)

Register 4.6. EFUSE_PGM_DATA5_REG (0x0014)



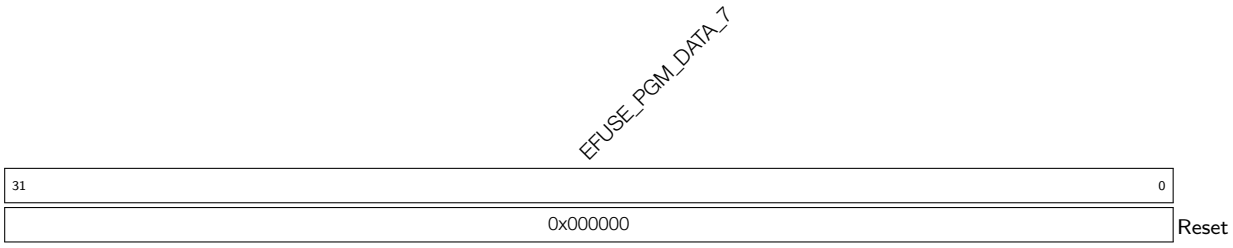
EFUSE_PGM_DATA_5 Configures the content of the 5th 32-bit data to be programmed. (R/W)

Register 4.7. EFUSE_PGM_DATA6_REG (0x0018)



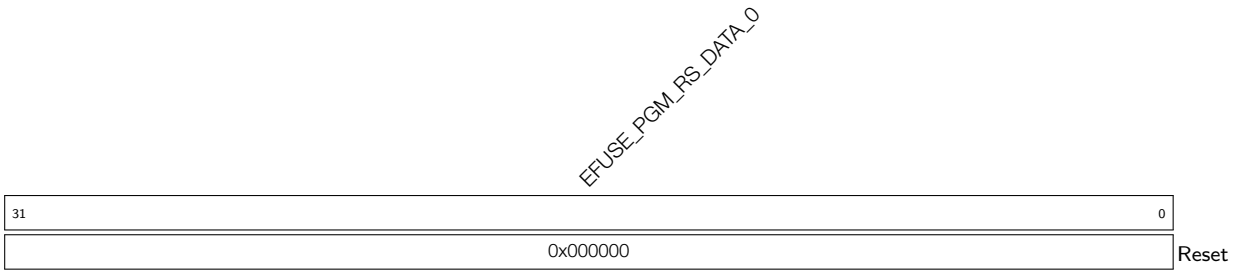
EFUSE_PGM_DATA_6 Configures the content of the 6th 32-bit data to be programmed. (R/W)

Register 4.8. EFUSE_PGM_DATA7_REG (0x001C)



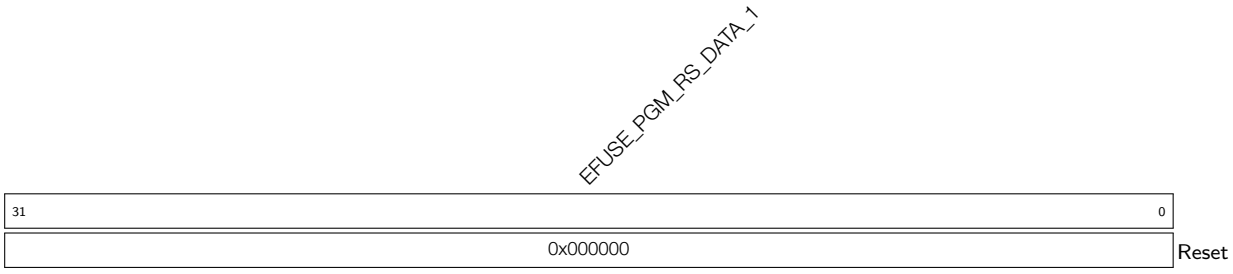
EFUSE_PGM_DATA_7 Configures the content of the 7th 32-bit data to be programmed. (R/W)

Register 4.9. EFUSE_PGM_CHECK_VALUE0_REG (0x0020)



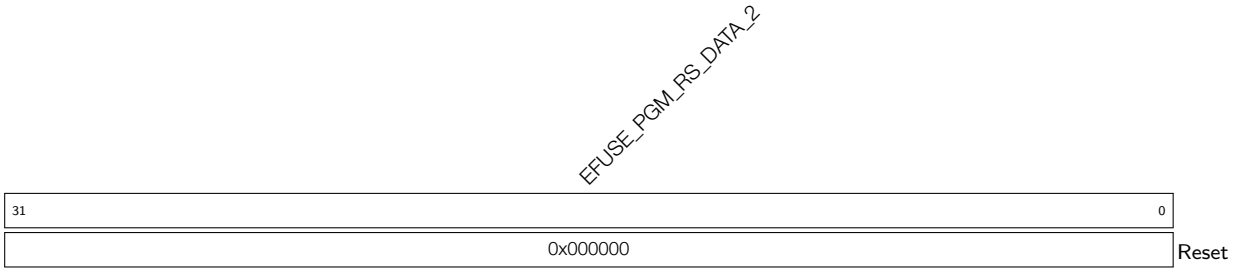
EFUSE_PGM_RS_DATA_0 Configures the content of the 0th 32-bit RS code to be programmed. (R/W)

Register 4.10. EFUSE_PGM_CHECK_VALUE1_REG (0x0024)



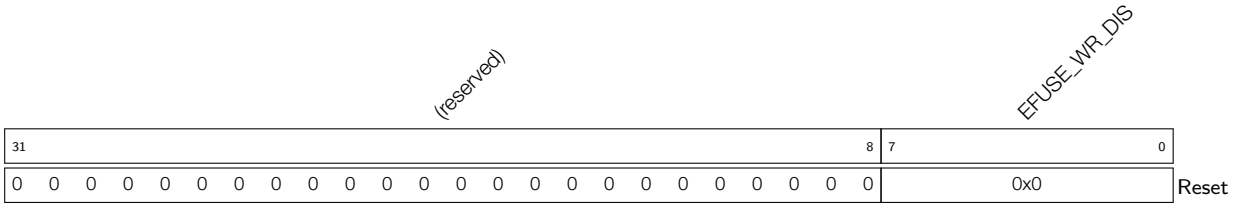
EFUSE_PGM_RS_DATA_1 Configures the content of the 1st 32-bit RS code to be programmed. (R/W)

Register 4.11. EFUSE_PGM_CHECK_VALUE2_REG (0x0028)



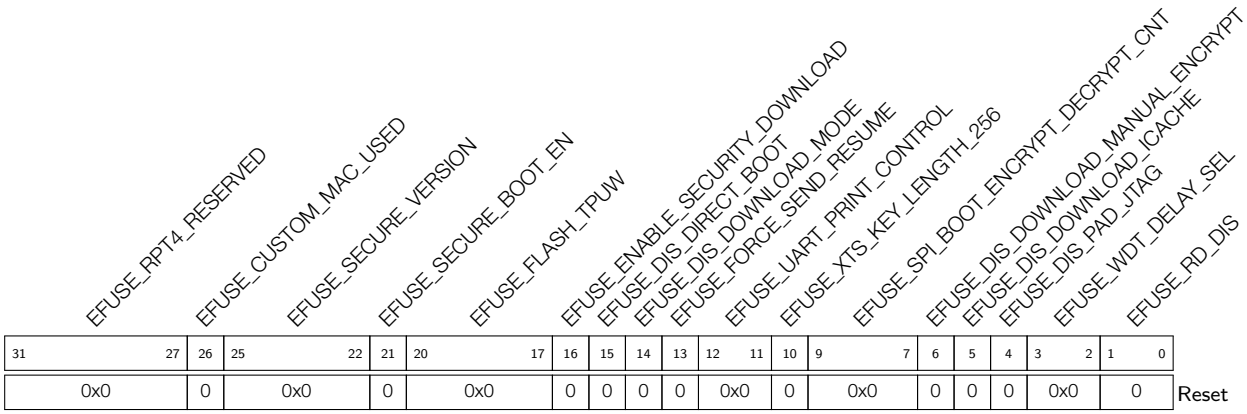
EFUSE_PGM_RS_DATA_2 Configures the content of the 2nd 32-bit RS code to be programmed. (R/W)

Register 4.12. EFUSE_RD_WR_DIS_REG (0x002C)



EFUSE_WR_DIS Represents whether programming of individual eFuses is disabled or enabled. 1: Disabled. 0: Enabled. (RO)

Register 4.13. EFUSE_RD_REPEAT_DATA0_REG (0x0030)



EFUSE_RD_DIS Represents whether reading of the high/low 128 bits is disabled or enabled. 1: Disabled. 0: Enabled. (RO)

EFUSE_WDT_DELAY_SEL Represents RTC watchdog timeout threshold. Measurement unit: slow clock cycle. 0: 40000. 1: 80000. 2: 160000. 3: 320000. (RO)

EFUSE_DIS_PAD_JTAG Represents whether pad JTAG is permanently disabled or enabled. 1: Disabled. 0: Enabled. (RO)

EFUSE_DIS_DOWNLOAD_ICACHE Represents whether iCache is disabled or enabled in download mode. 1: Disabled. 0: Enabled. (RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT Represents whether manual flash encryption is disabled or enabled in download boot modes. 1: Disabled. 0: Enabled. (RO)

EFUSE_SPI_BOOT_ENCRYPT_DECRYPT_CNT Represents whether SPI boot encryption and decryption are disabled or enabled. Odd number of 1: Enabled. Even number of 1: Disabled. (RO)

EFUSE_XTS_KEY_LENGTH_256 Represents key length for XTS_AES. 1: All 256 bits of BLOCK3. 0: The lower 128 bits of BLOCK3. (RO)

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

EFUSE_FORCE_SEND_RESUME Represents whether to force ROM code to send an SPI flash resume command during SPI boot. 1: Send. 0: Not send. (RO)

EFUSE_DIS_DOWNLOAD_MODE Represents whether all Download modes are disabled or enabled (boot_mode[3:0] = 0, 1, 2, 4, 5, 6, 7). 1: Disabled. 0: Enabled. (RO)

EFUSE_DIS_DIRECT_BOOT Represents whether Direct_boot mode is disabled or enabled. 1: Disabled. 0: Enabled. (RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD Represents whether UART secure download mode is enabled or disabled (read/write flash only). 1: Enabled. 0: Disabled. (RO)

Continued on the next page...

Register 4.13. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

Continued from the previous page...

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

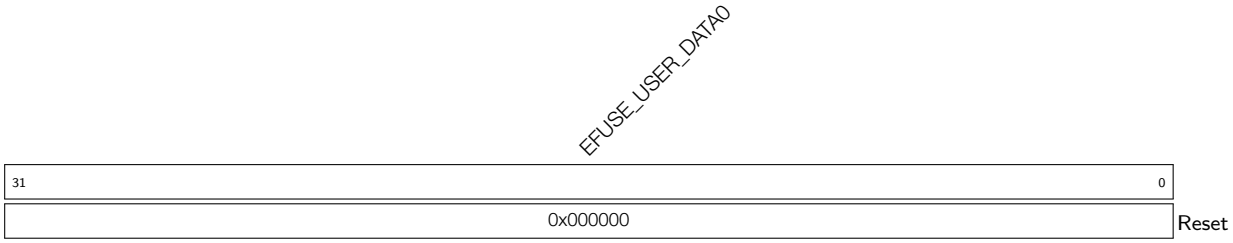
EFUSE_SECURE_BOOT_EN Represents whether secure boot is enabled or disabled. 1: Enabled. 0: Disabled. (RO)

EFUSE_SECURE_VERSION Represents the secure version used by ESP-IDF anti-rollback feature. (RO)

EFUSE_CUSTOM_MAC_USED Represents whether the MAC customized by users is used or not. 1: Used. 0: Not used. (RO)

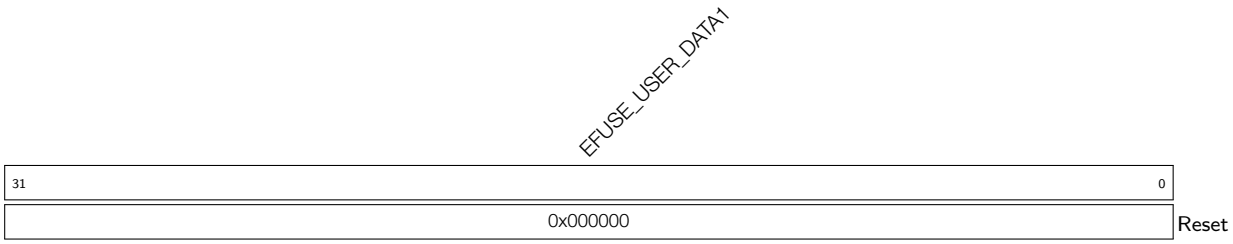
EFUSE_RPT4_RESERVED Reserved (used for four backups method). (RO)

Register 4.14. EFUSE_RD_BLK1_DATA0_REG (0x0034)



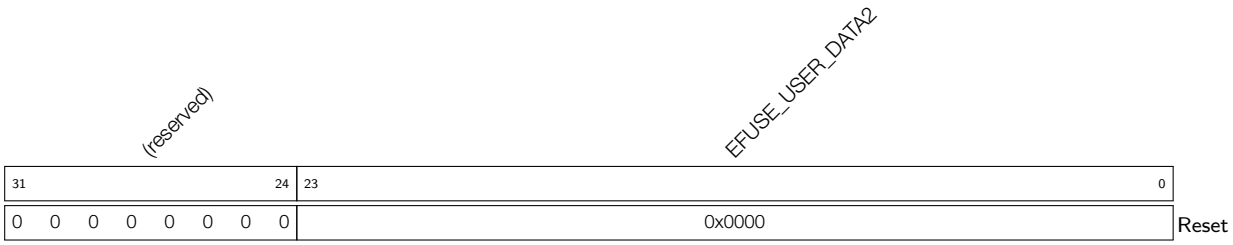
EFUSE_USER_DATA0 Stores the 0th 32-bit of user data. (RO)

Register 4.15. EFUSE_RD_BLK1_DATA1_REG (0x0038)



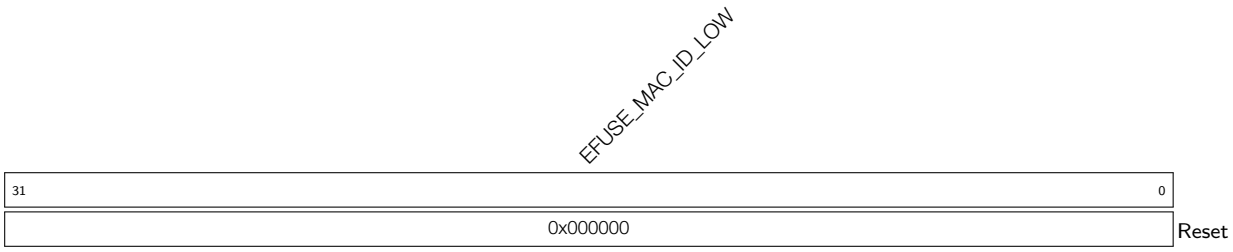
EFUSE_USER_DATA1 Stores the 1st 32-bit of user data. (RO)

Register 4.16. EFUSE_RD_BLK1_DATA2_REG (0x003C)



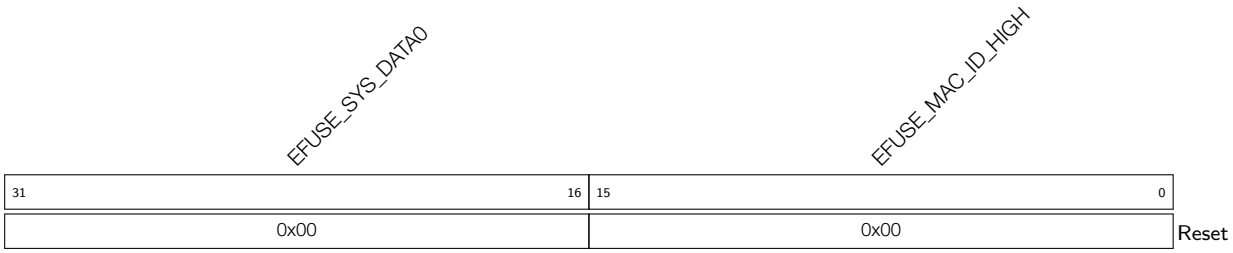
EFUSE_USER_DATA2 Stores the bits [64:87] of user data. (RO)

Register 4.17. EFUSE_RD_BLK2_DATA0_REG (0x0040)



EFUSE_MAC_ID_LOW Stores the lower 32-bit of MAC ID. (RO)

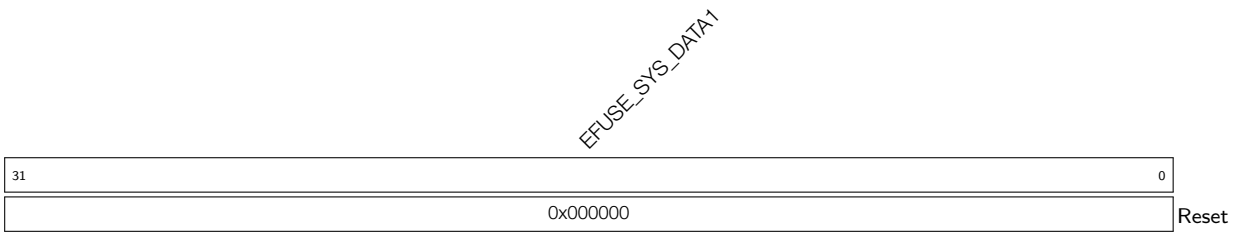
Register 4.18. EFUSE_RD_BLK2_DATA1_REG (0x0044)



EFUSE_MAC_ID_HIGH Stores the higher 16-bit of MAC ID. (RO)

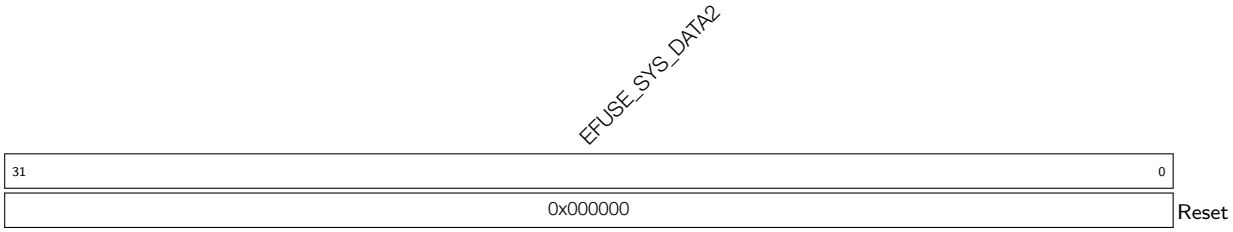
EFUSE_SYS_DATA0 Stores the 0th 16-bit of system data. (RO)

Register 4.19. EFUSE_RD_BLK2_DATA2_REG (0x0048)



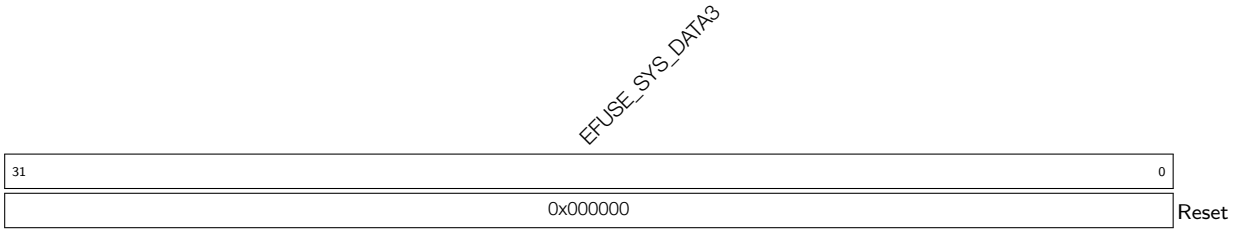
EFUSE_SYS_DATA1 Stores the 0th 32-bit of system data. (RO)

Register 4.20. EFUSE_RD_BLK2_DATA3_REG (0x004C)



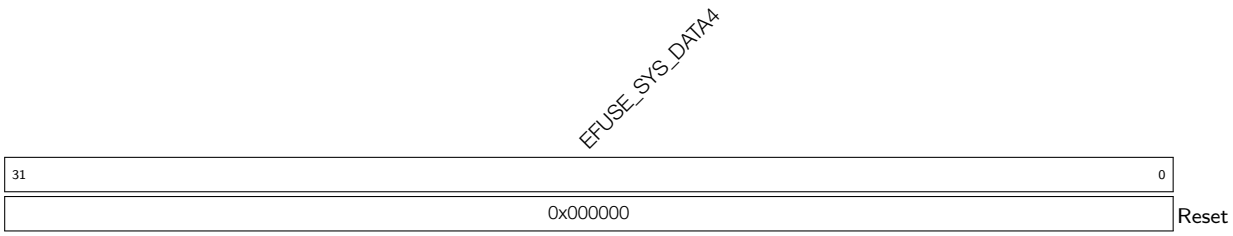
EFUSE_SYS_DATA2 Stores the 1st 32-bit of system data. (RO)

Register 4.21. EFUSE_RD_BLK2_DATA4_REG (0x0050)



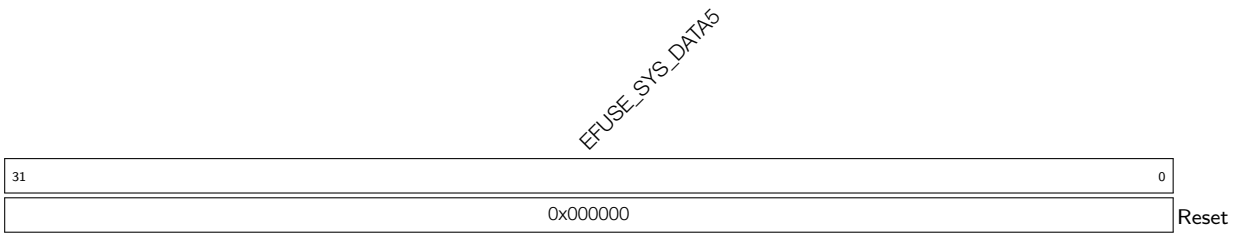
EFUSE_SYS_DATA3 Stores the 2nd 32-bit of system data. (RO)

Register 4.22. EFUSE_RD_BLK2_DATA5_REG (0x0054)



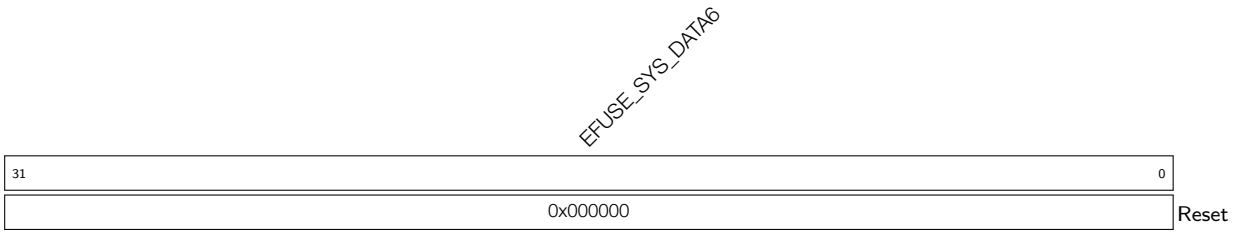
EFUSE_SYS_DATA4 Stores the 3rd 32-bit of system data. (RO)

Register 4.23. EFUSE_RD_BLK2_DATA6_REG (0x0058)



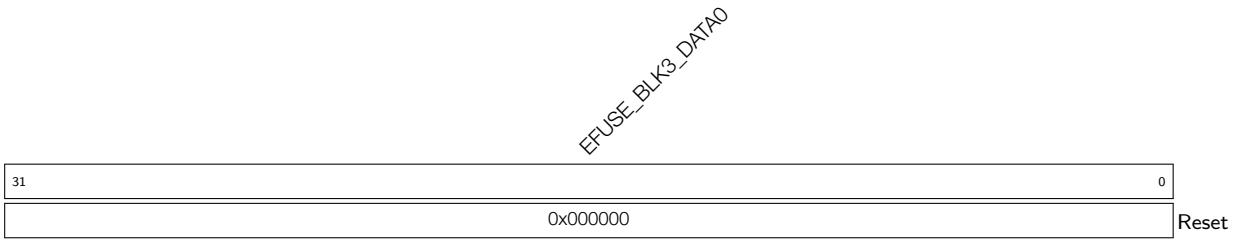
EFUSE_SYS_DATA5 Stores the 4th 32-bit of system data. (RO)

Register 4.24. EFUSE_RD_BLK2_DATA7_REG (0x005C)



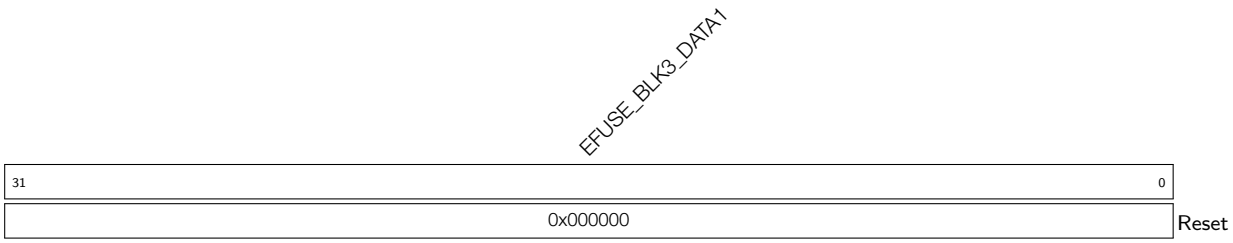
EFUSE_SYS_DATA6 Stores the 5th 32-bit of system data. (RO)

Register 4.25. EFUSE_RD_BLK3_DATA0_REG (0x0060)



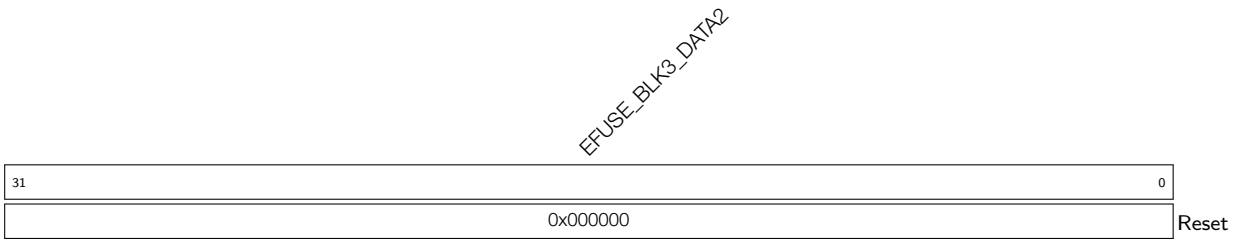
EFUSE_BLK3_DATA0 Stores the 0th 32-bit of BLOCK3. (RO)

Register 4.26. EFUSE_RD_BLK3_DATA1_REG (0x0064)



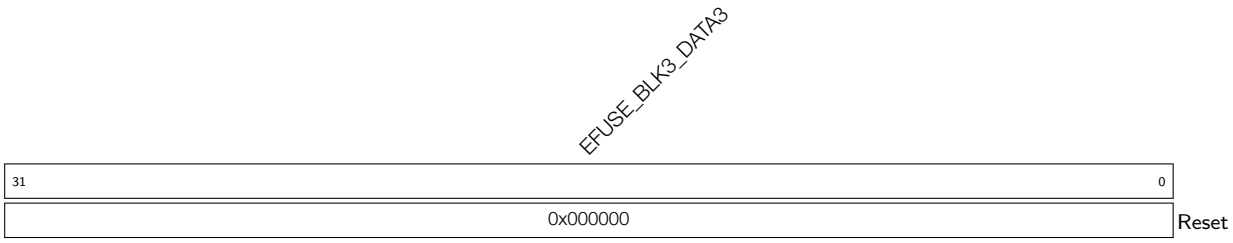
EFUSE_BLK3_DATA1 Stores the 1st 32-bit of BLOCK3. (RO)

Register 4.27. EFUSE_RD_BLK3_DATA2_REG (0x0068)



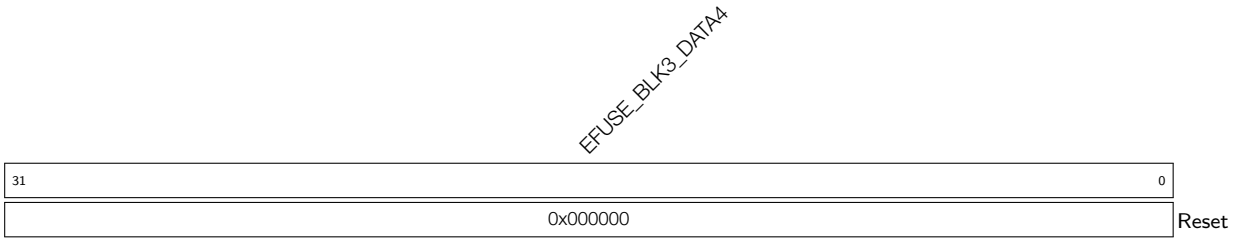
EFUSE_BLK3_DATA2 Stores the 2nd 32-bit of BLOCK3. (RO)

Register 4.28. EFUSE_RD_BLK3_DATA3_REG (0x006C)



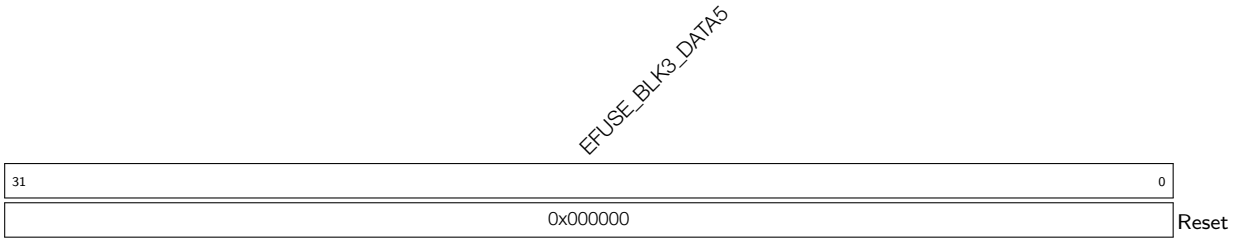
EFUSE_BLK3_DATA3 Stores the 3rd 32-bit of BLOCK3. (RO)

Register 4.29. EFUSE_RD_BLK3_DATA4_REG (0x0070)



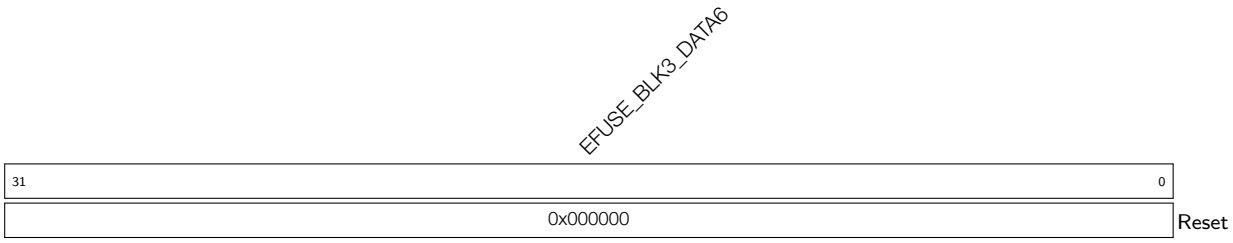
EFUSE_BLK3_DATA4 Stores the 4th 32-bit of BLOCK3. (RO)

Register 4.30. EFUSE_RD_BLK3_DATA5_REG (0x0074)



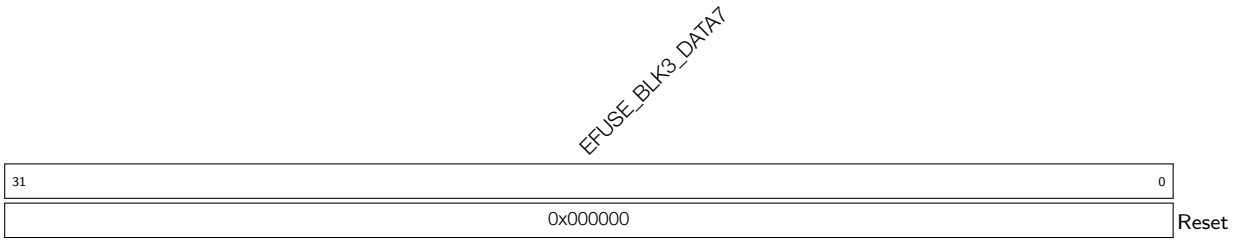
EFUSE_BLK3_DATA5 Stores the 5th 32-bit of BLOCK3. (RO)

Register 4.31. EFUSE_RD_BLK3_DATA6_REG (0x0078)



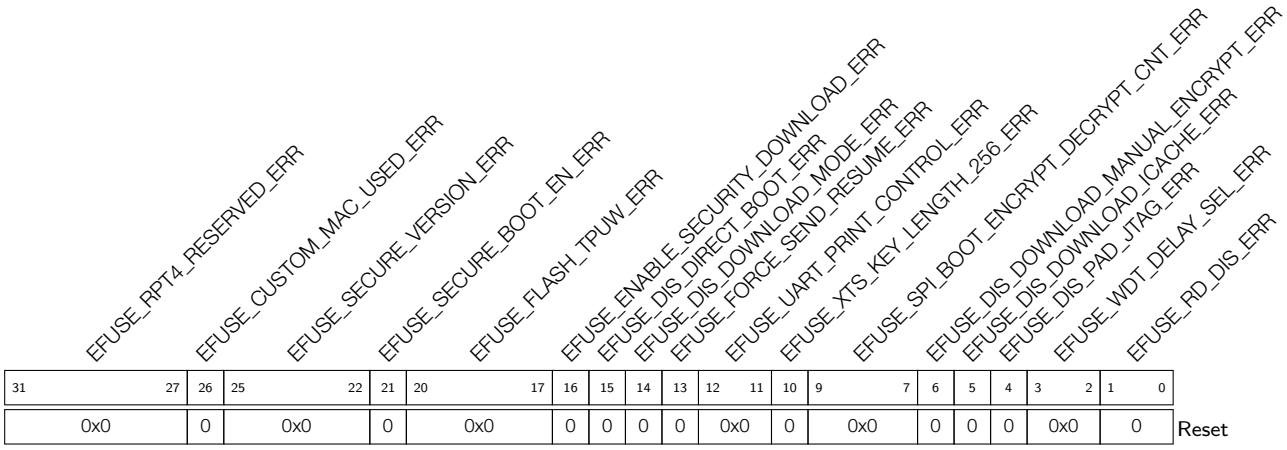
EFUSE_BLK3_DATA6 Stores the 6th 32-bit of BLOCK3. (RO)

Register 4.32. EFUSE_RD_BLK3_DATA7_REG (0x007C)



EFUSE_BLK3_DATA7 Stores the 7th 32-bit of BLOCK3. (RO)

Register 4.33. EFUSE_RD_REPEAT_ERR_REG (0x0080)



EFUSE_RD_DIS_ERR If any bit in RD_DIS_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_WDT_DELAY_SEL_ERR If any bit in WDT_DELAY_SEL_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_DIS_PAD_JTAG_ERR If any bit in DIS_PAD_JTAG_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_DIS_DOWNLOAD_ICACHE_ERR If any bit in DIS_DOWN_ICACHE_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR If any bit in DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_SPI_BOOT_ENCRYPT_DECRYPT_CNT_ERR If any bit in SPI_BOOT_ENCRYPT_DECRYPT_CNT_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_XTS_KEY_LENGTH_256_ERR If any bit in XTS_KEY_LENGTH_256_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_UART_PRINT_CONTROL_ERR If any bit in UART_PRINT_CONTROL_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_FORCE_SEND_RESUME_ERR If any bit in FORCE_SEND_RESUME_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_DIS_DOWNLOAD_MODE_ERR If any bit in DIS_DOWNLOAD_MODE_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_DIS_DIRECT_BOOT_ERR If any bit in DIS_DIRECT_BOOT_ERR is 1, then it indicates a programming error of this parameter. (RO)

Continued on the next page...

Register 4.33. EFUSE_RD_REPEAT_ERR_REG (0x0080)

Continued from the previous page...

EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR If any bit in ENABLE_SECURITY_DOWNLOAD_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_FLASH_TPUW_ERR If any bit in FLASH_TPUW_ERR is 1, then it indicates a programming error of this parameter. (RO)

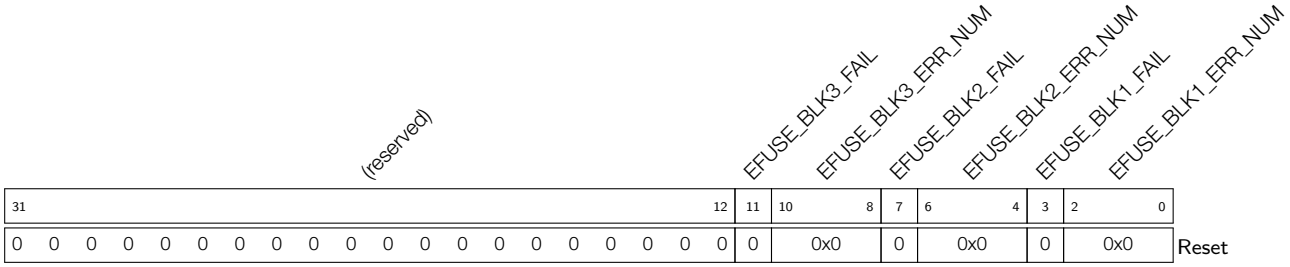
EFUSE_SECURE_BOOT_EN_ERR If any bit in SECURE_BOOT_EN_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_SECURE_VERSION_ERR If any bit in SECURE_VERSION_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_CUSTOM_MAC_USED_ERR If any bit in CUSTOM_MAC_USED_ERR is 1, then it indicates a programming error of this parameter. (RO)

EFUSE_RPT4_RESERVED_ERR Reserved. (RO)

Register 4.34. EFUSE_RD_RS_ERR_REG (0x0084)



EFUSE_BLK1_ERR_NUM The value of this signal means the number of error bytes in BLOCK1. (RO)

EFUSE_BLK1_FAIL 0: Means no failure and that the data of BLOCK1 is reliable. 1: Means that programming user data failed and the number of error bytes is over 6. (RO)

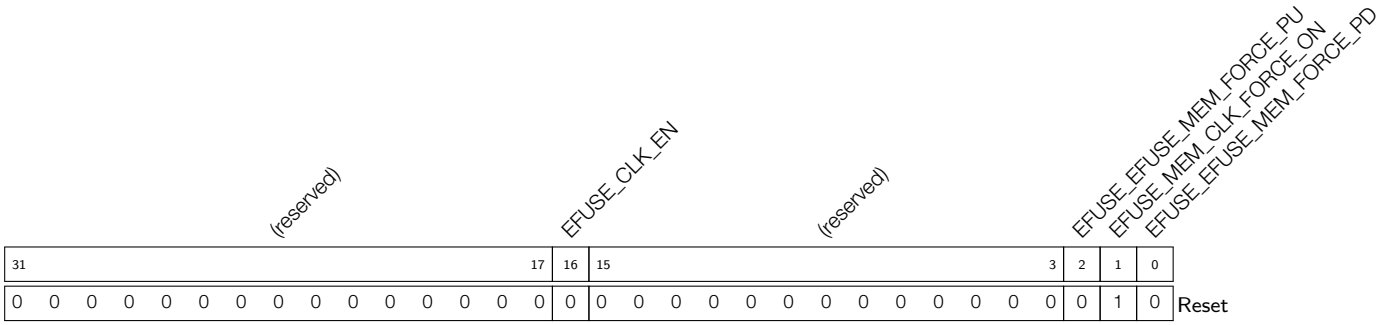
EFUSE_BLK2_ERR_NUM The value of this signal means the number of error bytes in BLOCK2. (RO)

EFUSE_BLK2_FAIL 0: Means no failure and that the data of BLOCK2 is reliable. 1: Means that programming user data failed and the number of error bytes is over 6. (RO)

EFUSE_BLK3_ERR_NUM The value of this signal means the number of error bytes in BLOCK3. (RO)

EFUSE_BLK3_FAIL 0: Means no failure and that the BLOCK3 data is reliable. 1: Means that programming user data failed and the number of error bytes is over 6. (RO)

Register 4.35. EFUSE_CLK_REG (0x0088)



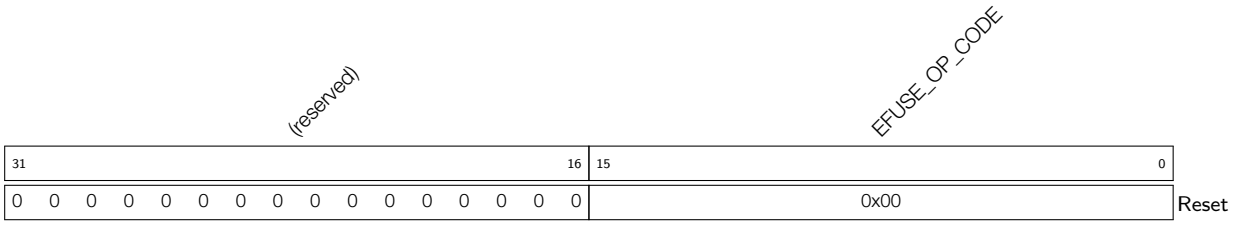
EFUSE_EFUSE_MEM_FORCE_PD Set this bit to force SRAM in eFuse controller into power-saving mode. (R/W)

EFUSE_MEM_CLK_FORCE_ON Set this bit to force to activate clock signal of SRAM in eFuse controller. (R/W)

EFUSE_EFUSE_MEM_FORCE_PU Set this bit to force SRAM in eFuse controller into working mode. (R/W)

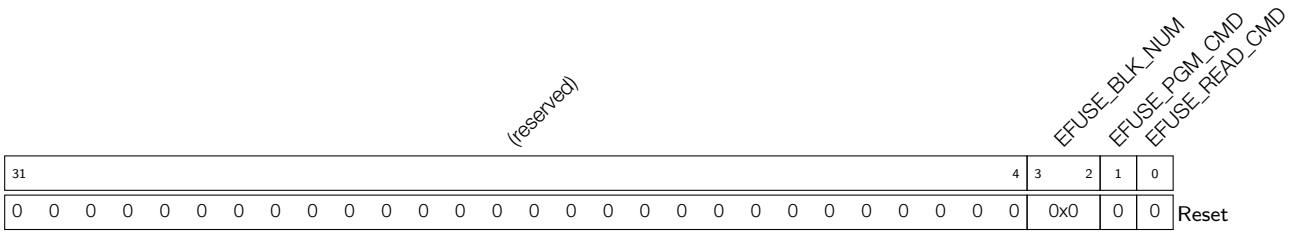
EFUSE_CLK_EN Set this bit to force enable clock signal of eFuse configuration register. (R/W)

Register 4.36. EFUSE_CONF_REG (0x008C)



EFUSE_OP_CODE 0x5A5A: Operate programming command. 0x5AA5: Operate read command. (R/W)

Register 4.37. EFUSE_CMD_REG (0x0094)

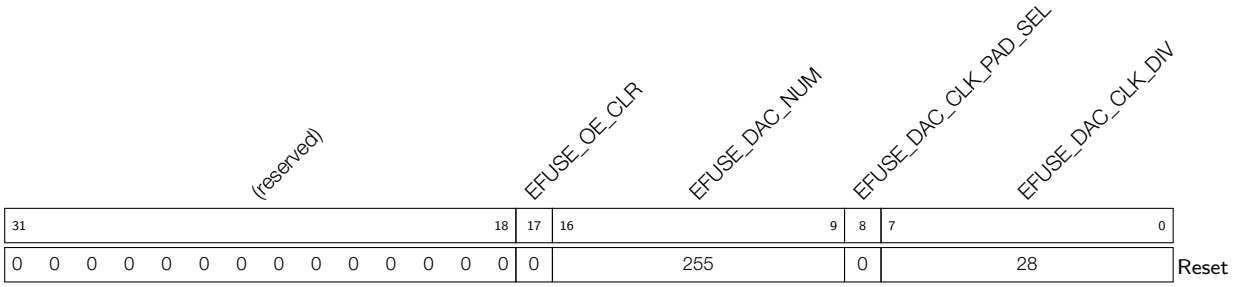


EFUSE_READ_CMD Set this bit to send read command. (R/W/SC)

EFUSE_PGM_CMD Set this bit to send programming command. (R/W/SC)

EFUSE_BLK_NUM The serial number of the block to be programmed. Value 0-3 corresponds to block number 0-3 respectively. (R/W)

Register 4.38. EFUSE_DAC_CONF_REG (0x0108)



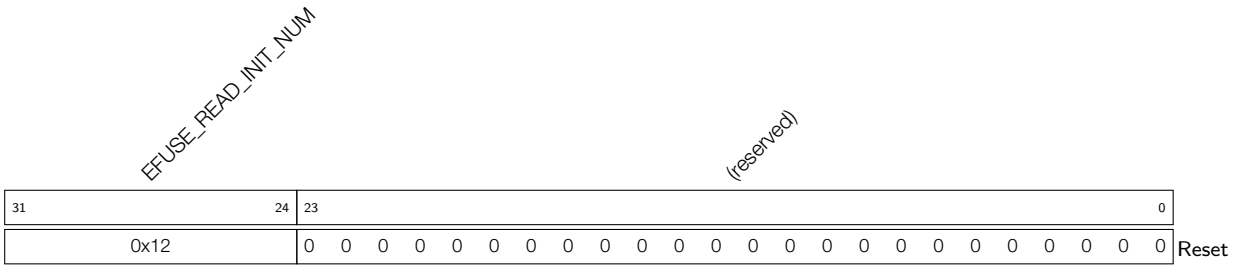
EFUSE_DAC_CLK_DIV Controls the division factor of the rising clock of the programming voltage. (R/W)

EFUSE_DAC_CLK_PAD_SEL Don't care. (R/W)

EFUSE_DAC_NUM Controls the rising period of the programming voltage. (R/W)

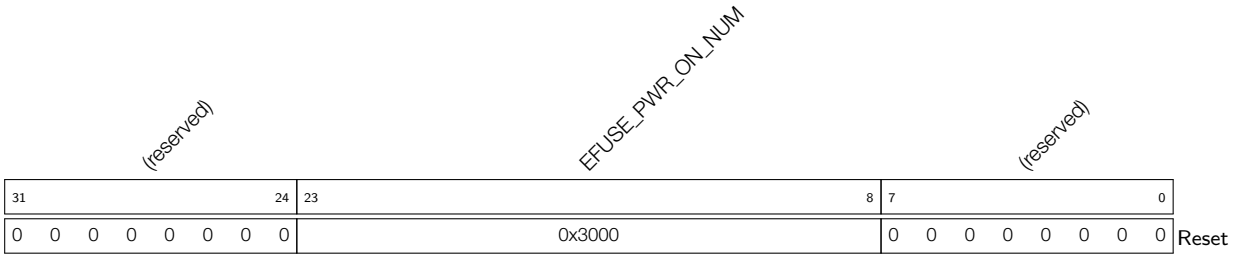
EFUSE_OE_CLR Reduces the power supply of the programming voltage. (R/W)

Register 4.39. EFUSE_RD_TIM_CONF_REG (0x010C)



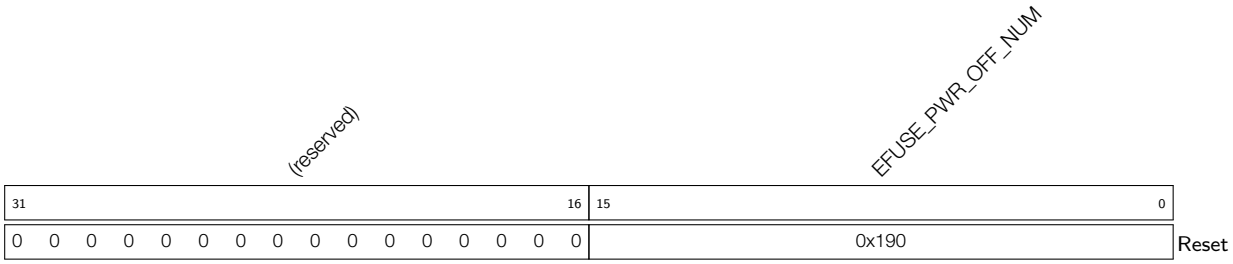
EFUSE_READ_INIT_NUM Configures the waiting time of reading eFuse memory. (R/W)

Register 4.40. EFUSE_WR_TIM_CONF1_REG (0x0114)



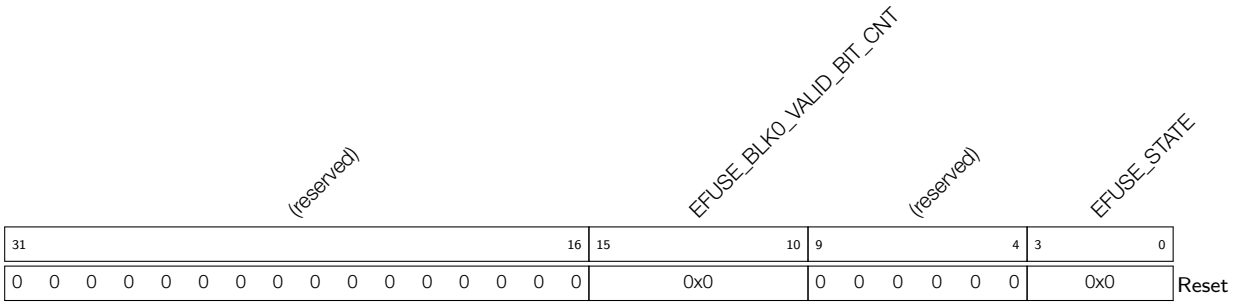
EFUSE_PWR_ON_NUM Configures the power up time for VDDQ. (R/W)

Register 4.41. EFUSE_WR_TIM_CONF2_REG (0x0118)



EFUSE_PWR_OFF_NUM Configures the power outage time for VDDQ. (R/W)

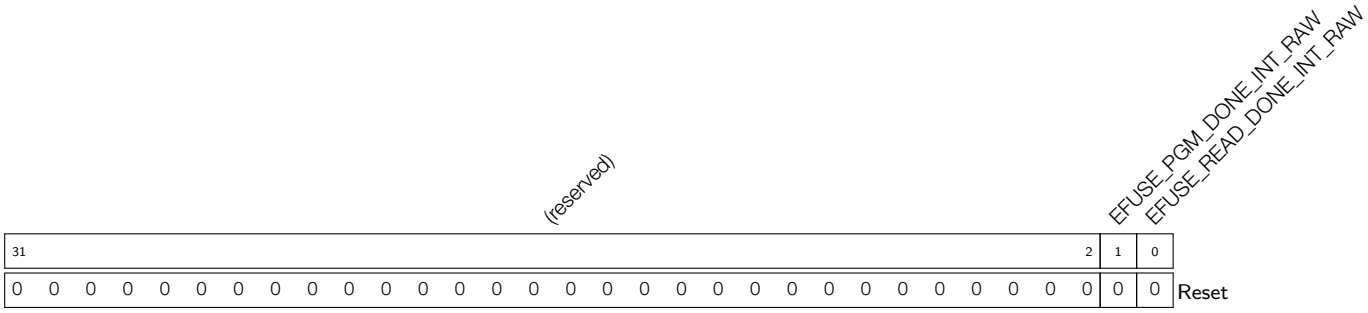
Register 4.42. EFUSE_STATUS_REG (0x0090)



EFUSE_STATE Indicates the state of the eFuse controller state machine. (RO)

EFUSE_BLK0_VALID_BIT_CNT Records the number of bits with a value of '1' in BLOCK0. (RO)

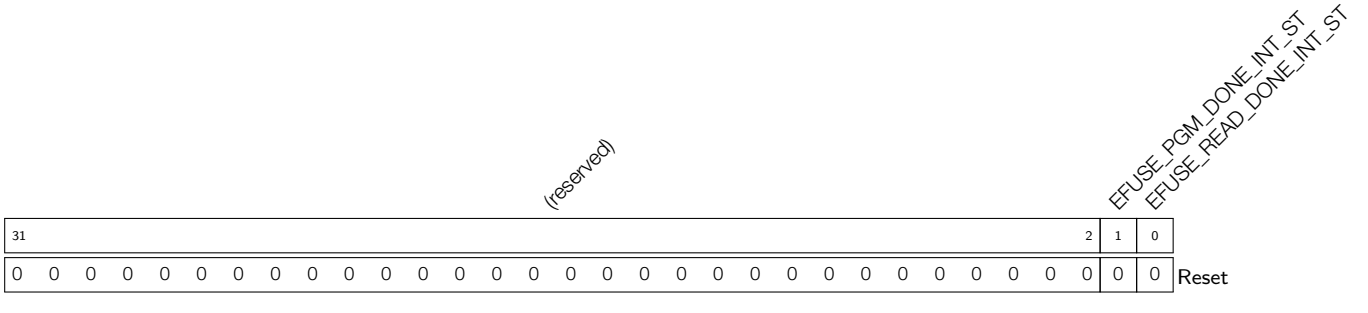
Register 4.43. EFUSE_INT_RAW_REG (0x0098)



EFUSE_READ_DONE_INT_RAW The raw bit signal for read_done interrupt. (R/WTC/SS)

EFUSE_PGM_DONE_INT_RAW The raw bit signal for pgm_done interrupt. (R/WTC/SS)

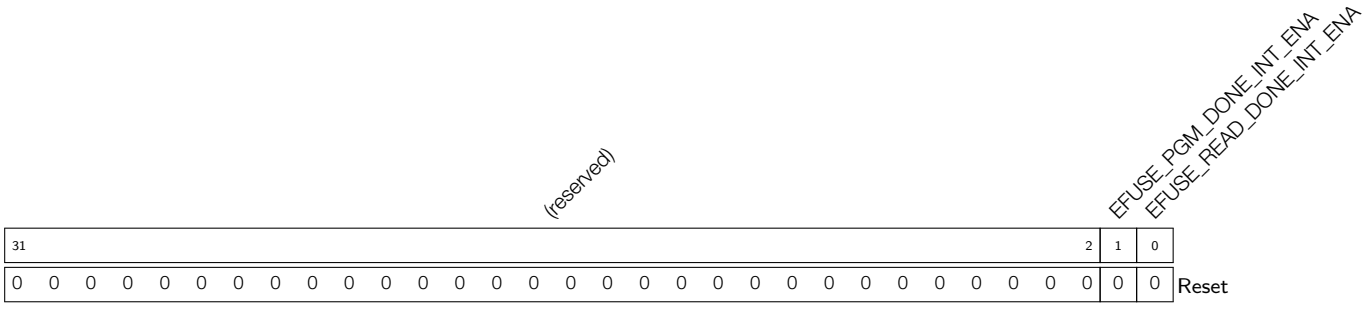
Register 4.44. EFUSE_INT_ST_REG (0x009C)



EFUSE_READ_DONE_INT_ST The status signal for read_done interrupt. (RO)

EFUSE_PGM_DONE_INT_ST The status signal for pgm_done interrupt. (RO)

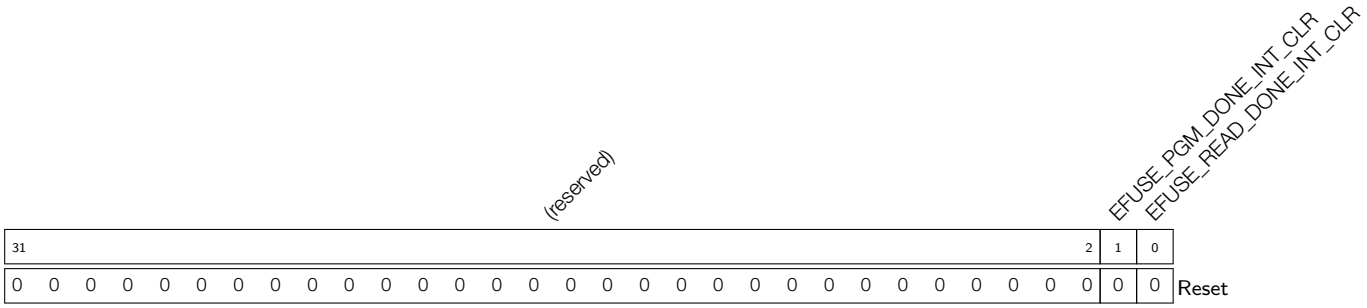
Register 4.45. EFUSE_INT_ENA_REG (0x0100)



EFUSE_READ_DONE_INT_ENA The enable signal for read_done interrupt. (R/W)

EFUSE_PGM_DONE_INT_ENA The enable signal for pgm_done interrupt. (R/W)

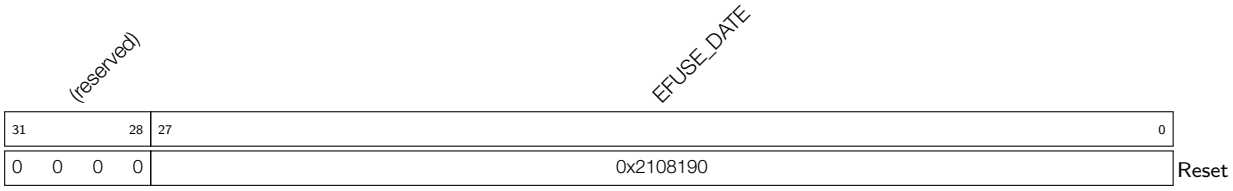
Register 4.46. EFUSE_INT_CLR_REG (0x0104)



EFUSE_READ_DONE_INT_CLR The clear signal for read_done interrupt. (WT)

EFUSE_PGM_DONE_INT_CLR The clear signal for pgm_done interrupt. (WT)

Register 4.47. EFUSE_DATE_REG (0x01FC)



EFUSE_DATE Stores eFuse controller register version. (R/W)

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

5.1 Overview

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

Note:

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

5.2 Features

GPIO matrix has the following features:

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

IO MUX has the following features:

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

5.3 Architectural Overview

Figure 5-1 shows in details how IO MUX and GPIO matrix route signals from pins to peripherals, and from peripherals to pins.

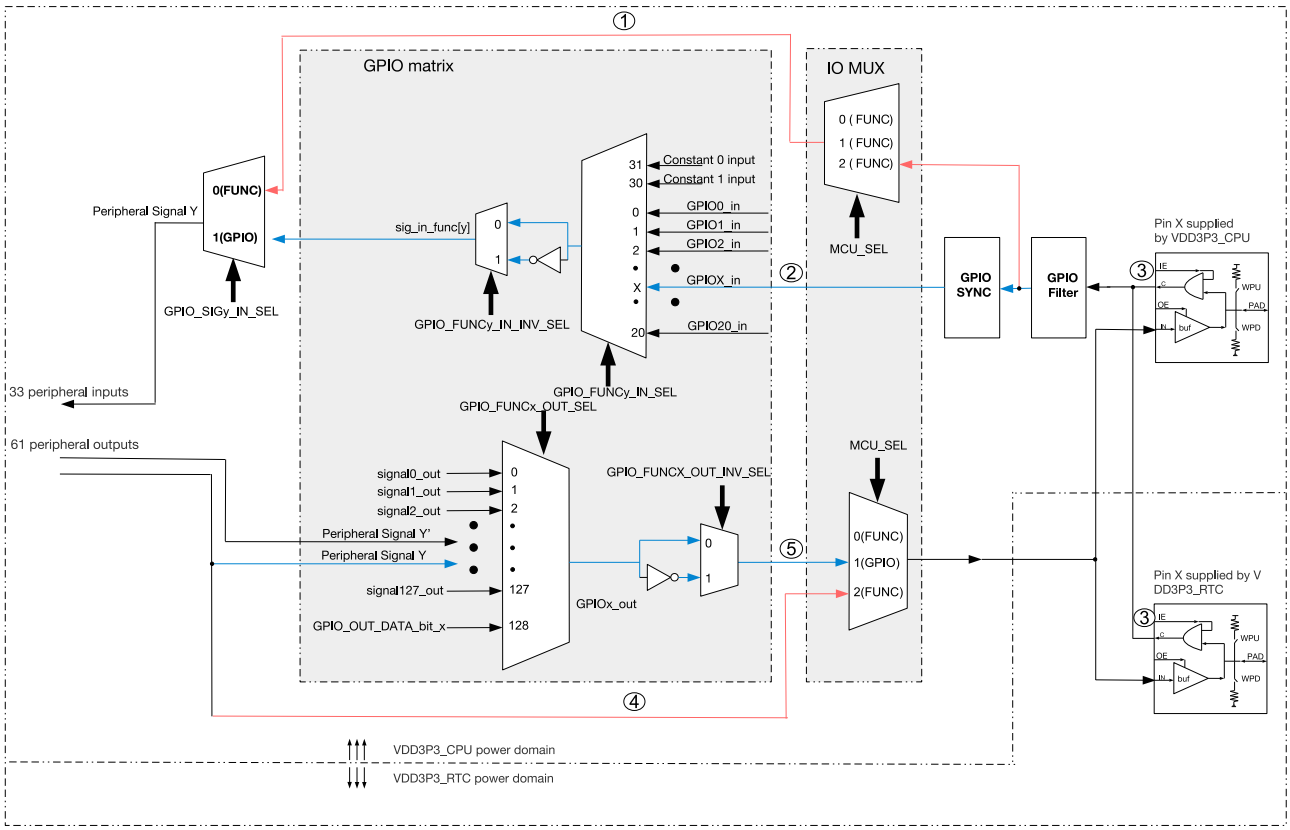


Figure 5-1. Architecture of IO MUX and GPIO Matrix

1. Only part of peripheral input signals (marked “yes” in column “Direct input through IO MUX” in Table 5-2) can bypass GPIO matrix. The other input signals can only be routed to peripherals via GPIO matrix.
2. There are only 21 inputs from GPIO SYNC to GPIO matrix, since ESP8684 provides 21 GPIO pins in total. Note, for chip variants with SiP flash, there are only 14 inputs from GPIO SYNC to GPIO matrix in total.
3. The pins supplied by VDD3P3_CPU or by VDD3P3_RTC are controlled by the signals: IE, OE, WPU, and WPD.
4. Only part of peripheral outputs (marked “yes” in column “Direct output through IO MUX” in Table 5-2) can be routed to pins bypassing GPIO matrix. The other output signals can only be routed to pins via GPIO matrix.
5. There are 21 outputs (corresponding to GPIO *X*: 0 ~ 20) from GPIO matrix to IO MUX. Note, for chip variants with SiP flash, there are only 14 outputs (corresponding to GPIO *X*: 0 ~ 10, 18 ~ 20) from GPIO matrix to IO MUX in total.

Figure 5-2 shows the internal structure of a pad, which is an electrical interface between the chip logic and the GPIO pin. The structure is applicable to all 21 GPIO pins and can be controlled by IE, OE, WPU, and WPD signals.

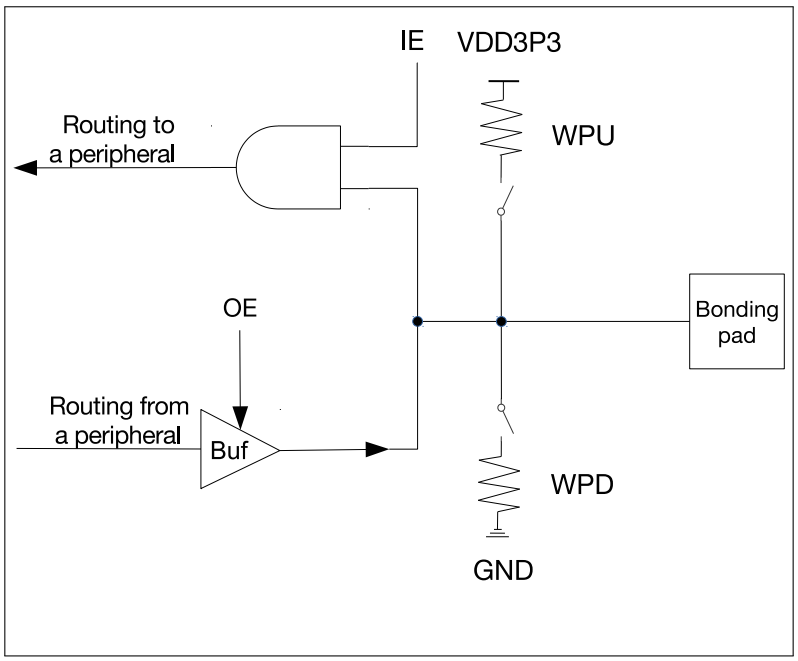


Figure 5-2. Internal Structure of a Pad

Note:

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

5.4 Peripheral Input via GPIO Matrix

5.4.1 Overview

To receive a peripheral input signal via GPIO matrix, the matrix is configured to source the peripheral input signal from one of the 21 GPIOs (0 ~ 20), see Table 5-2. Meanwhile, register corresponding to the peripheral signal should be set to receive input signal via GPIO matrix.

5.4.2 Signal Synchronization

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

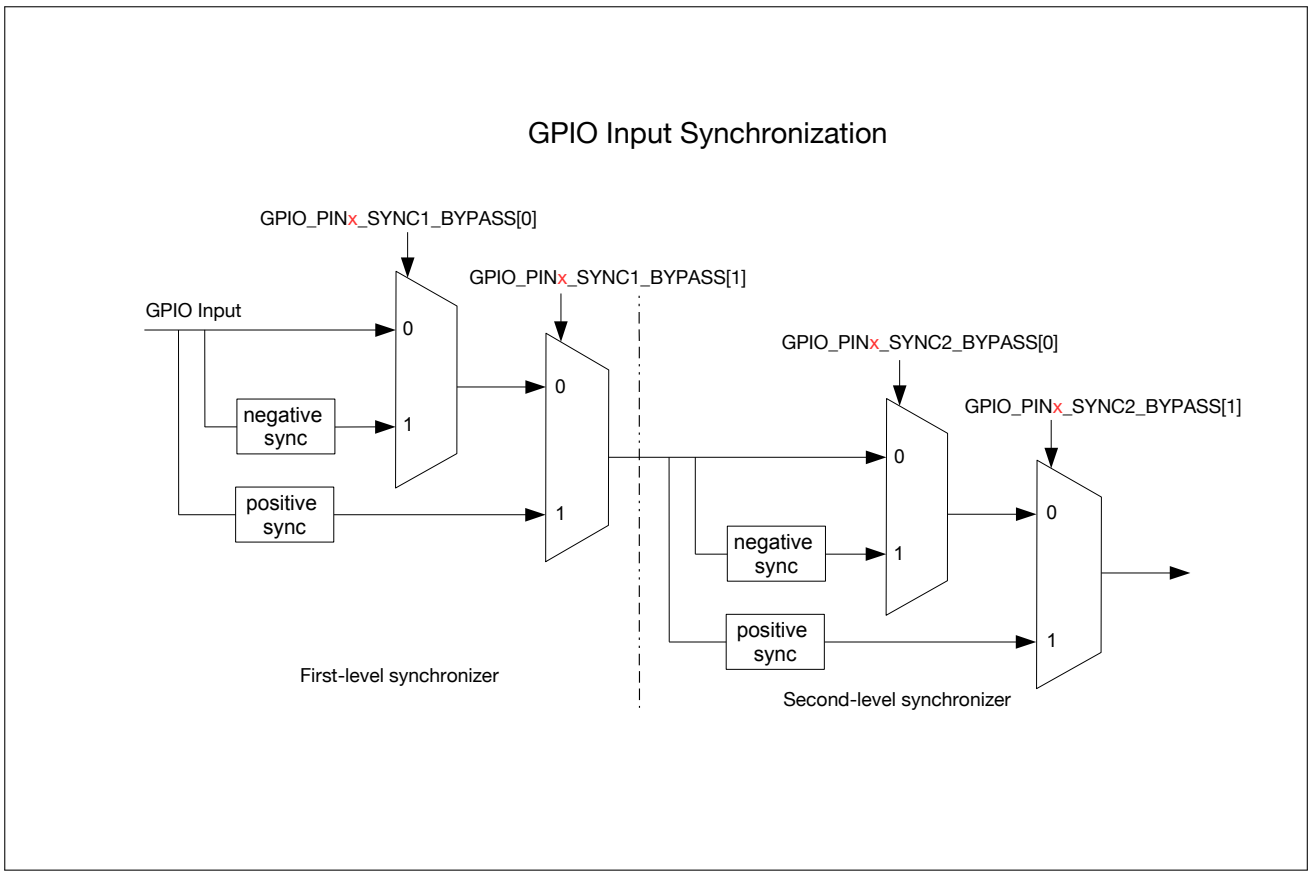


Figure 5-3. GPIO Input Synchronized on APB Clock Rising Edge or on Falling Edge

Figure 5-3 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.

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

5.4.3 Functional Description

To read GPIO pin *X*[1] into peripheral signal *Y*, follow the steps below:

1. Configure register GPIO_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 IO_MUX_GPIO*n*_FILTER_EN. Only the signals with a valid width of more than two clock cycles can be sampled, see Figure 5-4.

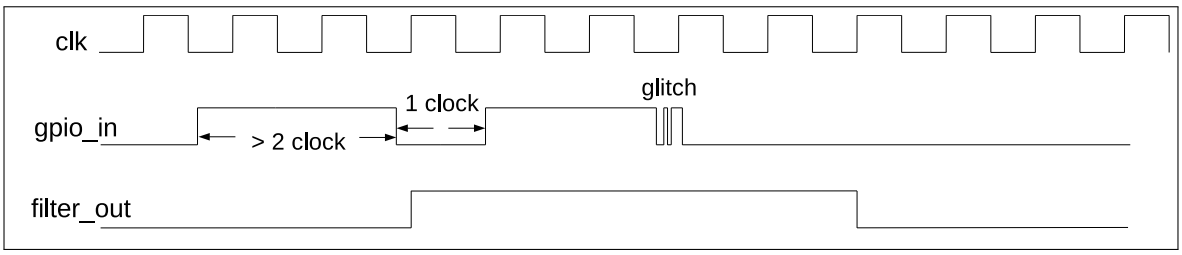


Figure 5-4. Filter Timing of GPIO Input Signals

3. Synchronize GPIO input. To do so, please set GPIO_PINx_REG corresponding to GPIO pin X as follows:
   - Set GPIO_PINx_SYNC1_BYPASS to enable input signal synchronized on rising edge or on falling edge in the first-level synchronizer, see Figure 5-3.
   - Set GPIO_PINx_SYNC2_BYPASS to enable input signal synchronized on rising edge or on falling edge in the second-level synchronizer, see Figure 5-3.
4. Configure IO MUX register to enable pin input. For this end, please set IO_MUX_GPIOx_REG corresponding to GPIO pin X as follows:
   - Set IO_MUX_GPIOx_FUN_IE to enable input[2].
   - Set or clear IO_MUX_GPIOx_FUN_WPU and IO_MUX_GPIOx_FUN_WPD, as desired, to enable or disable pull-up and pull-down resistors.

For example, to connect UART0 DSR input signal [3] (U0DSR_in, signal index 8) to GPIO7, please follow the steps below. Note that GPIO7 is also named as MTDO pin.

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

Note:

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

5.4.4 Simple GPIO Input

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

To implement simple GPIO input, follow the steps below:

- Set IO_MUX_GPIO*x*_FUN_IE in register IO_MUX_GPIO*x*_REG, to enable pin input.
- Read the GPIO input from GPIO_IN_REG[*x*].

5.5 Peripheral Output via GPIO Matrix

5.5.1 Overview

To output a signal from a peripheral via GPIO matrix, the matrix is configured to route peripheral output signals (only signals with a name assigned in the column "Output signal" in Table 5-2) to one of the 21 GPIOs (0 ~ 20). Note, for chip variants with SiP flash, output signals can only be mapped to 14 GPIO pins, i.e. GPIO0 ~ GPIO10, GPIO18 ~ GPIO20.

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

Note:
There is a range of peripheral output signals (97 ~ 100 in Table 5-2) which are not connected to any peripheral, but to the input signals (97 ~ 100) directly. This feature can be used to input a signal from one GPIO pin and output directly to another GPIO pin.

5.5.2 Functional Description

The 61 output signals (signals with a name assigned in the column "Output signal" in Table 5-2) can be set to go through GPIO matrix into IO MUX and then to a pin. Figure 5-1 illustrates the configuration.

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

1. Configure register 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 GPIO_FUNC*x*_OEN_SEL bit in register GPIO_FUNC*x*_OUT_SEL_CFG_REG and the bit in register GPIO_ENABLE_W1TS_REG, corresponding to GPIO pin *X*. To have the output enable signal decided by internal logic (for example, the SPIQ_oe in column "Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0" in Table 5-2), clear GPIO_FUNC*x*_OEN_SEL bit instead.
   - Set the corresponding bit in register GPIO_ENABLE_W1TC_REG to disable the output from the GPIO pin.
2. For an open drain output, set the GPIO_PIN*x*_PAD_DRIVER bit in register GPIO_PIN*x*_REG corresponding to GPIO pin *X*.
3. Configure IO MUX register to enable output via GPIO matrix. Set the IO_MUX_GPIO*x*_REG corresponding to GPIO pin *X* as follows:
   - Set the field IO_MUX_GPIO*x*_MCU_SEL to desired IO MUX function corresponding to GPIO pin*X*. This is Function 1 (GPIO function), numeric value 1, for all pins.

- Set the IO_MUX_GPIO*x*_FUN_DRV field to the desired value for output strength (0 ~ 3). The higher the driver strength, the more current can be sourced/sunk from the pin.
  - 0: ~5 mA
  - 1: ~10 mA
  - 2: ~20 mA (default)
  - 3: ~40 mA
- If using open drain mode, set/clear the IO_MUX_GPIO*x*_FUN_WPU and IO_MUX_GPIO*x*_FUN_WPD bits to enable/disable the internal pull-up/pull-down resistors.

Note:

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

5.5.3 Simple GPIO Output

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

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

Note:

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

5.6 Direct Input and Output via IO MUX

5.6.1 Overview

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

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

5.6.2 Functional Description

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

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

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

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

5.7 Analog Functions of GPIO Pins

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

- Set IO_MUX_GPIO*n*_MCU_SEL to 1, and clear IO_MUX_GPIO*n*_FUN_IE, IO_MUX_GPIO*n*_FUN_WPU, IO_MUX_GPIO*n*_FUN_WPD.
- Write 1 to GPIO_ENABLE_W1TC*[n]*, to clear output enable.

See Table 5-4 for analog functions of ESP8684 pins.

5.8 Pin Functions in Light-sleep

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

Table 5-1. 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_MCU_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 5.5.2 for how to enable output in normal execution.

5.9 Pin Hold Feature

Each GPIO pin (including the RTC pins: GPIO0 ~ GPIO5) has an individual hold function controlled by a RTC register. When the pin is set to hold, the state is latched at that moment and will not change no matter how the internal signals change or how the IO MUX/GPIO configuration is modified. Users can use the hold function for the pins to retain the pin state through a core reset and system reset triggered by watchdog time-out or Deep-sleep events.

Note:

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

5.10 Power Supplies and Management of GPIO Pins

5.10.1 Power Supplies of GPIO Pins

For more information on the power supply for GPIO pins, please refer to Pin Definition in *ESP8684 Datasheet*. All the pins can be used to wake up the chip from Light-sleep mode, but only the pins (GPIO0 ~ GPIO5) in VDD3P3_RTC domain can be used to wake up the chip from Deep-sleep mode.

5.10.2 Power Supply Management

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

- VDD3P3_RTC: the input power supply for both RTC and CPU
- VDD3P3_CPU: the input power supply for CPU

5.11 Peripheral Signal List

Table 5-2 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 5-2. 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.

- Only the signals with a name assigned in the column "Input signal" in Table 5-2 are valid input signals.
- Only the signals with a name assigned in the column "Output signal" in Table 5-2 are valid output signals.

Table 5-2. 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	U0RXD_in	0	yes	U0TXD_out	1'd1	yes
7	U0CTS_in	0	no	U0RTS_out	1'd1	no
8	U0DSR_in	0	no	U0DTR_out	1'd1	no
9	U1RXD_in	0	no	U1TXD_out	1'd1	no
10	U1CTS_in	0	no	U1RTS_out	1'd1	no
11	U1DSR_in	0	no	U1DTR_out	1'd1	no
12	-	-	-	-	-	-
13	-	-	-	-	-	-
14	-	-	-	-	-	-
15	-	-	-	SPIQ_monitor	1'd1	no
16	-	-	-	SPID_monitor	1'd1	no
17	-	-	-	SPIHD_monitor	1'd1	no
18	-	-	-	SPIWP_monitor	1'd1	no
19	-	-	-	SPICS1_out	SPICS1_oe	no
20	-	-	-	-	-	-
21	-	-	-	-	-	-
22	-	-	-	-	-	-
23	-	-	-	-	-	-
24	-	-	-	-	-	-

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

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

GoBack

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	-	-	-	-	-	-
80	-	-	-	-	-	-
81	-	-	-	-	-	-
82	-	-	-	-	-	-
83	-	-	-	-	-	-
84	-	-	-	-	-	-
85	-	-	-	-	-	-
86	-	-	-	-	-	-
87	-	-	-	-	-	-
88	-	-	-	-	-	-
89	-	-	-	ant_sel0	1'd1	no
90	-	-	-	ant_sel1	1'd1	no
91	-	-	-	ant_sel2	1'd1	no
92	-	-	-	ant_sel3	1'd1	no
93	-	-	-	ant_sel4	1'd1	no
94	-	-	-	ant_sel5	1'd1	no
95	-	-	-	ant_sel6	1'd1	no
96	-	-	-	ant_sel7	1'd1	no
97	sig_in_func_97	0	no	sig_in_func97	1'd1	no
98	sig_in_func_98	0	no	sig_in_func98	1'd1	no
99	sig_in_func_99	0	no	sig_in_func99	1'd1	no
100	sig_in_func_100	0	no	sig_in_func100	1'd1	no
101	-	-	-	-	-	-
102	-	-	-	-	-	-
103	-	-	-	-	-	-
104	-	-	-	-	-	-
105	-	-	-	-	-	-

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

5.12 IO MUX Functions List

Table 5-3 shows the IO MUX functions of each pin.

Table 5-3. IO MUX Pin Functions

Pin No.	Pin Name	Function 0	Function 1	Function 2	Function 3	DRV	Reset	Note
0	GPIO0	GPIO0	GPIO0	-	-	2	0	R
1	GPIO1	GPIO1	GPIO1	-	-	2	0	R
2	GPIO2	GPIO2	GPIO2	FSPIQ	-	2	1	R
3	GPIO3	GPIO3	GPIO3	-	-	2	1	R
4	MTMS	MTMS	GPIO4	FSPIHD	-	2	1	R
5	MTDI	MTDI	GPIO5	FSPIWP	-	2	1	R
6	MTCK	MTCK	GPIO6	FSPICLK	-	2	1*	-
7	MTDO	MTDO	GPIO7	FSPID	-	2	1	-
8	GPIO8	GPIO8	GPIO8	-	-	2	1	-
9	GPIO9	GPIO9	GPIO9	-	-	2	3	-
10	GPIO10	GPIO10	GPIO10	FSPICS0	-	2	1	-
11	VDD_SPI	GPIO11	GPIO11	-	-	2	0	S
12	SPIHD	SPIHD	GPIO12	-	-	2	3	S
13	SPIWP	SPIWP	GPIO13	-	-	2	3	S
14	SPICS0	SPICS0	GPIO14	-	-	2	3	S
15	SPICLK	SPICLK	GPIO15	-	-	2	3	S
16	SPID	SPID	GPIO16	-	-	2	3	S
17	SPIQ	SPIQ	GPIO17	-	-	2	3	S
18	GPIO18	GPIO18	GPIO18	-	-	2	0	-
19	U0RXD	U0RXD	GPIO19	-	-	2	3	-
20	U0TXD	U0TXD	GPIO20	-	-	2	4	-

Drive Strength

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

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

Reset Configurations

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

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

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

Note:

- **R** - Pins in VDD3P3_RTC domain, and part of them have analog functions, see Table 5-4.
- **S** - For chip variants with SiP flash, these pins are only used to connect SiP flash, i.e. only Function 0 is available. For chip variants without SiP flash, these pins can be used as normal pins, i.e. all the functions are available.

5.13 Analog Functions List

Table 5-4 shows the IO MUX pins with analog functions.

Table 5-4. Analog Functions of IO MUX Pins

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

5.14 Register Summary

5.14.1 GPIO Matrix Register Summary

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

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

- **Configuration Registers**: can only be configured for GPIO0 ~ GPIO10 and GPIO18 ~ GPIO20;
- **Pin Configuration Registers**: only GPIO_PIN0_REG ~ GPIO_PIN10_REG and GPIO_PIN18_REG ~ GPIO_PIN20_REG are available;
- **Input Configuration Registers**: can only be configured for GPIO0 ~ GPIO10 and GPIO18 ~ GPIO20;
- **Output Configuration Registers**: only GPIO_FUNC0_OUT_SEL_CFG_REG ~ GPIO_FUNC10_OUT_SEL_CFG_REG and GPIO_PIN18_OUT_SEL_CFG_REG ~ GPIO_PIN20_OUT_SEL_CFG_REG are available;

Name	Description	Address	Access
Configuration Registers			
GPIO_OUT_REG	GPIO output register	0x0004	R/W/SS
GPIO_OUT_W1TS_REG	GPIO output set register	0x0008	WT
GPIO_OUT_W1TC_REG	GPIO output clear register	0x000C	WT

Name	Description	Address	Access
GPIO_ENABLE_REG	GPIO output enable register	0x0020	R/W/SS
GPIO_ENABLE_W1TS_REG	GPIO output enable set register	0x0024	WT
GPIO_ENABLE_W1TC_REG	GPIO output enable clear register	0x0028	WT
GPIO_STRAP_REG	Pin strapping register	0x0038	RO
GPIO_IN_REG	GPIO input register	0x003C	RO
GPIO_STATUS_REG	GPIO interrupt status register	0x0044	R/W/SS
GPIO_STATUS_W1TS_REG	GPIO interrupt status set register	0x0048	WT
GPIO_STATUS_W1TC_REG	GPIO interrupt status clear register	0x004C	WT
GPIO_PCPU_INT_REG	GPIO CPU interrupt status register	0x005C	RO
GPIO_PCPU_NMI_INT_REG	GPIO CPU (non-maskable) interrupt status register	0x0060	RO
GPIO_STATUS_NEXT_REG	GPIO interrupt source register	0x014C	RO
Pin Configuration Registers			
GPIO_PIN0_REG	GPIO pin 0 configuration register	0x0074	R/W
GPIO_PIN1_REG	GPIO pin 1 configuration register	0x0078	R/W
GPIO_PIN2_REG	GPIO pin 2 configuration register	0x007C	R/W
GPIO_PIN3_REG	GPIO pin 3 configuration register	0x0080	R/W
GPIO_PIN4_REG	GPIO pin 4 configuration register	0x0084	R/W
GPIO_PIN5_REG	GPIO pin 5 configuration register	0x0088	R/W
GPIO_PIN6_REG	GPIO pin 6 configuration register	0x008C	R/W
GPIO_PIN7_REG	GPIO pin 7 configuration register	0x0090	R/W
GPIO_PIN8_REG	GPIO pin 8 configuration register	0x0094	R/W
GPIO_PIN9_REG	GPIO pin 9 configuration register	0x0098	R/W
GPIO_PIN10_REG	GPIO pin 10 configuration register	0x009C	R/W
GPIO_PIN11_REG	GPIO pin 11 configuration register	0x00A0	R/W
GPIO_PIN12_REG	GPIO pin 12 configuration register	0x00A4	R/W
GPIO_PIN13_REG	GPIO pin 13 configuration register	0x00A8	R/W
GPIO_PIN14_REG	GPIO pin 14 configuration register	0x00AC	R/W
GPIO_PIN15_REG	GPIO pin 15 configuration register	0x00B0	R/W
GPIO_PIN16_REG	GPIO pin 16 configuration register	0x00B4	R/W
GPIO_PIN17_REG	GPIO pin 17 configuration register	0x00B8	R/W
GPIO_PIN18_REG	GPIO pin 18 configuration register	0x00BC	R/W
GPIO_PIN19_REG	GPIO pin 19 configuration register	0x00C0	R/W
GPIO_PIN20_REG	GPIO pin 20 configuration register	0x00C4	R/W
Input Function Configuration Registers			
GPIO_FUNC0_IN_SEL_CFG_REG	Configuration register for input signal 0	0x0154	R/W
GPIO_FUNC1_IN_SEL_CFG_REG	Configuration register for input signal 1	0x0158	R/W
...	...	...	...
GPIO_FUNC126_IN_SEL_CFG_REG	Configuration register for input signal 126	0x034C	R/W
GPIO_FUNC127_IN_SEL_CFG_REG	Configuration register for input signal 127	0x0350	R/W
Output Function Configuration Registers			
GPIO_FUNC0_OUT_SEL_CFG_REG	Configuration register for GPIO0 output	0x0554	R/W
GPIO_FUNC1_OUT_SEL_CFG_REG	Configuration register for GPIO1 output	0x0558	R/W

Name	Description	Address	Access
GPIO_FUNC2_OUT_SEL_CFG_REG	Configuration register for GPIO2 output	0x055C	R/W
GPIO_FUNC3_OUT_SEL_CFG_REG	Configuration register for GPIO3 output	0x0560	R/W
GPIO_FUNC4_OUT_SEL_CFG_REG	Configuration register for GPIO4 output	0x0564	R/W
GPIO_FUNC5_OUT_SEL_CFG_REG	Configuration register for GPIO5 output	0x0568	R/W
GPIO_FUNC6_OUT_SEL_CFG_REG	Configuration register for GPIO6 output	0x056C	R/W
GPIO_FUNC7_OUT_SEL_CFG_REG	Configuration register for GPIO7 output	0x0570	R/W
GPIO_FUNC8_OUT_SEL_CFG_REG	Configuration register for GPIO8 output	0x0574	R/W
GPIO_FUNC9_OUT_SEL_CFG_REG	Configuration register for GPIO9 output	0x0578	R/W
GPIO_FUNC10_OUT_SEL_CFG_REG	Configuration register for GPIO10 output	0x057C	R/W
GPIO_FUNC11_OUT_SEL_CFG_REG	Configuration register for GPIO11 output	0x0580	R/W
GPIO_FUNC12_OUT_SEL_CFG_REG	Configuration register for GPIO12 output	0x0584	R/W
GPIO_FUNC13_OUT_SEL_CFG_REG	Configuration register for GPIO13 output	0x0588	R/W
GPIO_FUNC14_OUT_SEL_CFG_REG	Configuration register for GPIO14 output	0x058C	R/W
GPIO_FUNC15_OUT_SEL_CFG_REG	Configuration register for GPIO15 output	0x0590	R/W
GPIO_FUNC16_OUT_SEL_CFG_REG	Configuration register for GPIO16 output	0x0594	R/W
GPIO_FUNC17_OUT_SEL_CFG_REG	Configuration register for GPIO17 output	0x0598	R/W
GPIO_FUNC18_OUT_SEL_CFG_REG	Configuration register for GPIO18 output	0x059C	R/W
GPIO_FUNC19_OUT_SEL_CFG_REG	Configuration register for GPIO19 output	0x05A0	R/W
GPIO_FUNC20_OUT_SEL_CFG_REG	Configuration register for GPIO20 output	0x05A4	R/W
Version Register			
GPIO_DATE_REG	GPIO version register	0x06FC	R/W
Clock Gate Register			
GPIO_CLOCK_GATE_REG	GPIO clock gate register	0x062C	R/W

5.14.2 IO MUX Register Summary

The addresses in this section are relative to the IO MUX base address provided in Table 3-3 in Chapte 3 *System and Memory* .

Note: For chip variants with SiP flash, only 14 GPIO pins are available, i.e. GPIO0 ~ GPIO10 and GPIO18 ~ GPIO20. For this case, IO_MUX_GPIO11_REG ~ IO_MUX_GPIO17_REG are not configurable.

Name	Description	Address	Access
Configuration Registers			
IO_MUX_PIN_CTRL_REG	Clock output configuration Register	0x0000	R/W
IO_MUX_GPIO0_REG	IO MUX configuration register for pin GPIO0	0x0004	R/W
IO_MUX_GPIO1_REG	IO MUX configuration register for pin GPIO1	0x0008	R/W
IO_MUX_GPIO2_REG	IO MUX configuration register for pin GPIO2	0x000C	R/W
IO_MUX_GPIO3_REG	IO MUX configuration register for pin GPIO3	0x0010	R/W
IO_MUX_GPIO4_REG	IO MUX configuration register for pin MTMS	0x0014	R/W
IO_MUX_GPIO5_REG	IO MUX configuration register for pin MTDI	0x0018	R/W
IO_MUX_GPIO6_REG	IO MUX configuration register for pin MTCK	0x001C	R/W
IO_MUX_GPIO7_REG	IO MUX configuration register for pin MTDO	0x0020	R/W
IO_MUX_GPIO8_REG	IO MUX configuration register for pin GPIO8	0x0024	R/W
IO_MUX_GPIO9_REG	IO MUX configuration register for pin GPIO9	0x0028	R/W

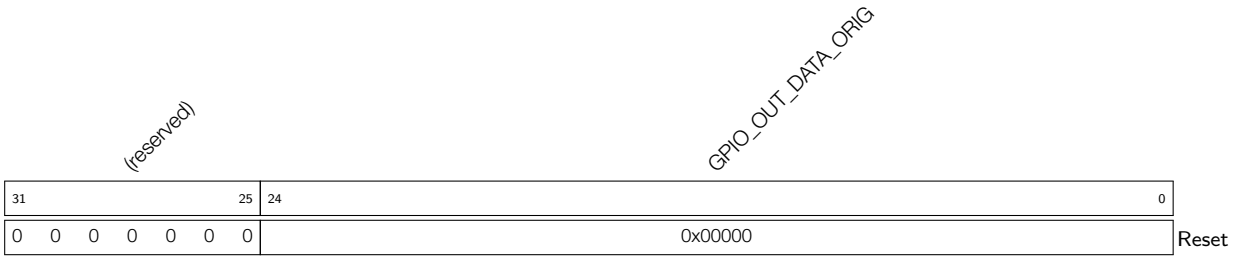
Name	Description	Address	Access
IO_MUX_GPIO10_REG	IO MUX configuration register for pin GPIO10	0x002C	R/W
IO_MUX_GPIO11_REG	IO MUX configuration register for pin VDD_SPI	0x0030	R/W
IO_MUX_GPIO12_REG	IO MUX configuration register for pin SPIHD	0x0034	R/W
IO_MUX_GPIO13_REG	IO MUX configuration register for pin SPIWP	0x0038	R/W
IO_MUX_GPIO14_REG	IO MUX configuration register for pin SPICS0	0x003C	R/W
IO_MUX_GPIO15_REG	IO MUX configuration register for pin SPICLK	0x0040	R/W
IO_MUX_GPIO16_REG	IO MUX configuration register for pin SPID	0x0044	R/W
IO_MUX_GPIO17_REG	IO MUX configuration register for pin SPIQ	0x0048	R/W
IO_MUX_GPIO18_REG	IO MUX configuration register for pin GPIO18	0x004C	R/W
IO_MUX_GPIO19_REG	IO MUX configuration register for pin U0RXD	0x0050	R/W
IO_MUX_GPIO20_REG	IO MUX configuration register for pin U0TXD	0x0054	R/W
Version Register			
IO_MUX_DATE_REG	IO MUX Version Control Register	0x00FC	R/W

5.15 Registers

5.15.1 GPIO Matrix Registers

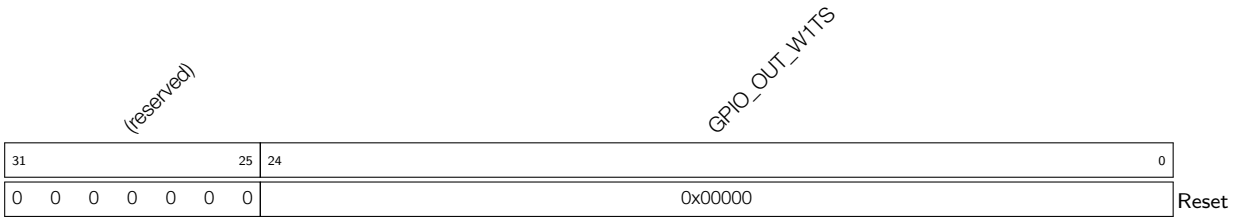
The addresses in this section are relative to GPIO base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Register 5.1. GPIO_OUT_REG (0x0004)



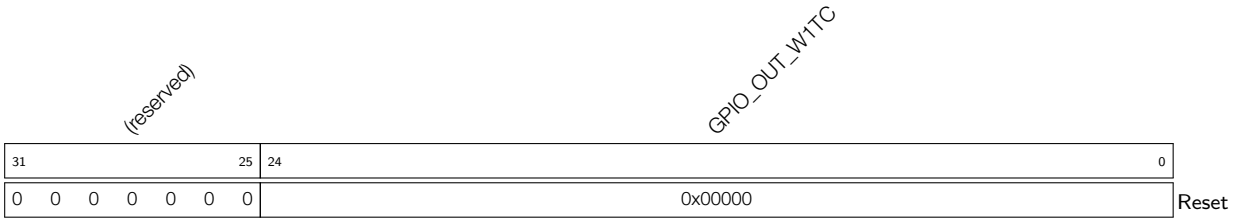
GPIO_OUT_DATA_ORIG GPIO0 ~ 20 output value in simple GPIO output mode. The values of bit0 ~ bit20 correspond to the output value of GPIO0 ~ GPIO20 respectively, and bit21 ~ bit24 are invalid. (R/W/SS)

Register 5.2. GPIO_OUT_W1TS_REG (0x0008)



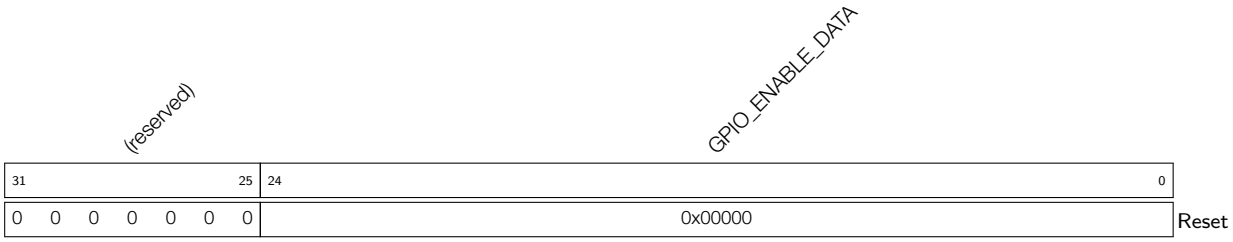
GPIO_OUT_W1TS GPIO0 ~ 20 output set register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT_REG will be set to 1. Recommended operation: use this register to set GPIO_OUT_REG. (WT)

Register 5.3. GPIO_OUT_W1TC_REG (0x000C)



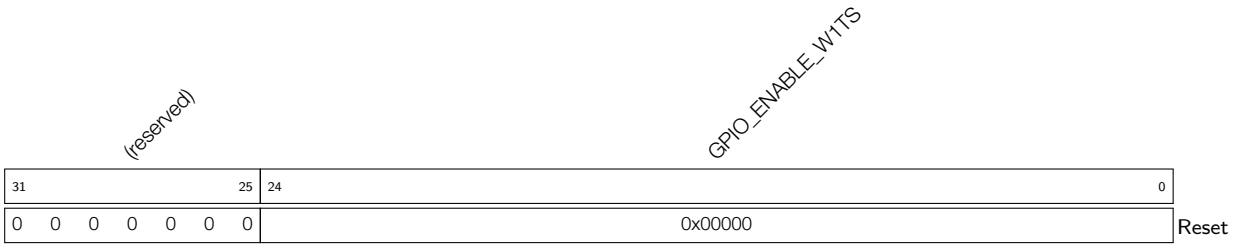
GPIO_OUT_W1TC GPIO0 ~ 20 output clear register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT_REG will be cleared. Recommended operation: use this register to clear GPIO_OUT_REG. (WT)

Register 5.4. GPIO_ENABLE_REG (0x0020)



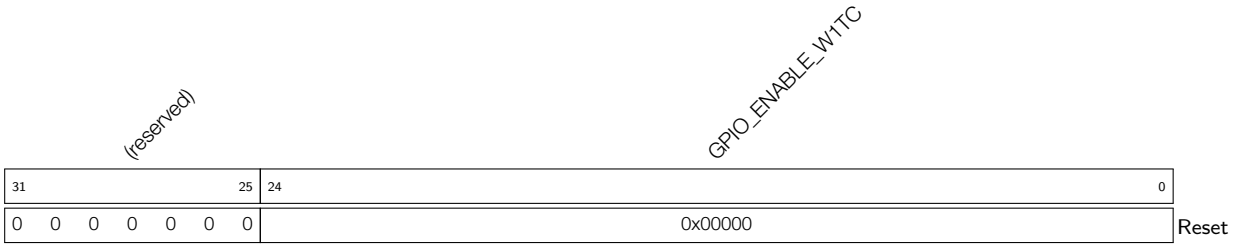
GPIO_ENABLE_DATA GPIO output enable register for GPIO0 ~ 20. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. (R/W/SS)

Register 5.5. GPIO_ENABLE_W1TS_REG (0x0024)



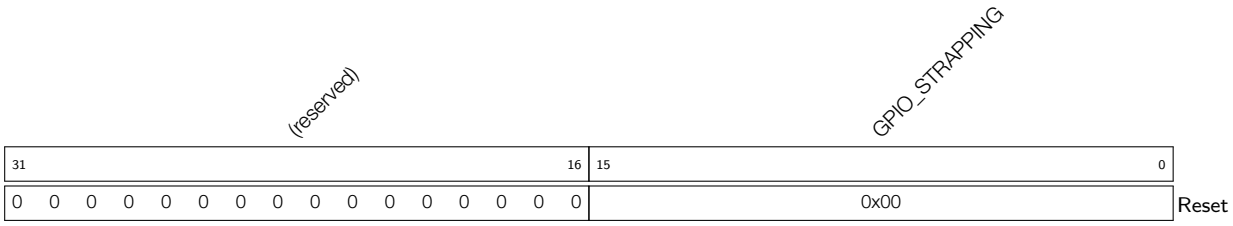
GPIO_ENABLE_W1TS GPIO0 ~ 20 output enable set register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE_REG will be set to 1. Recommended operation: use this register to set GPIO_ENABLE_REG. (WT)

Register 5.6. GPIO_ENABLE_W1TC_REG (0x0028)



GPIO_ENABLE_W1TC GPIO0 ~ 20 output enable clear register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE_REG will be cleared. Recommended operation: use this register to clear GPIO_ENABLE_REG. (WT)

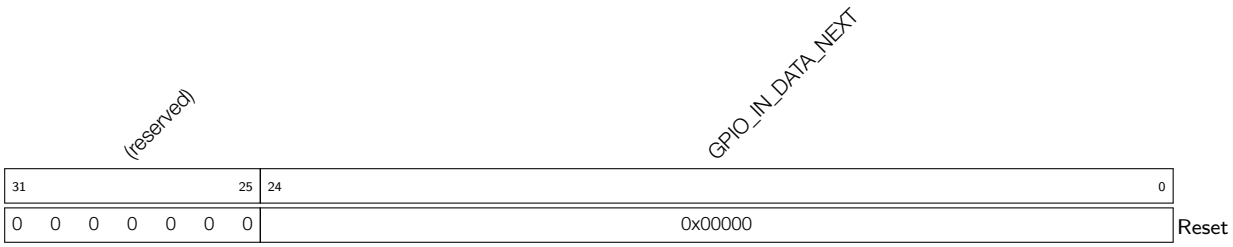
Register 5.7. GPIO_STRAP_REG (0x0038)



GPIO_STRAPPING GPIO strapping values. (RO)

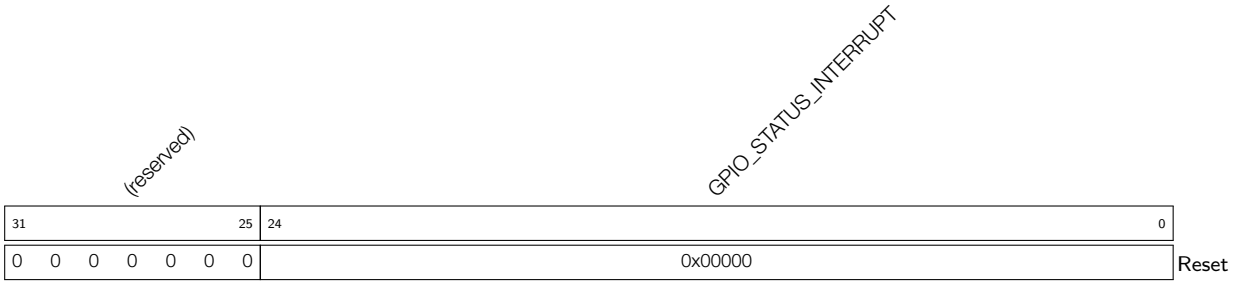
- bit 2: GPIO8
- bit 3: GPIO9

Register 5.8. GPIO_IN_REG (0x003C)



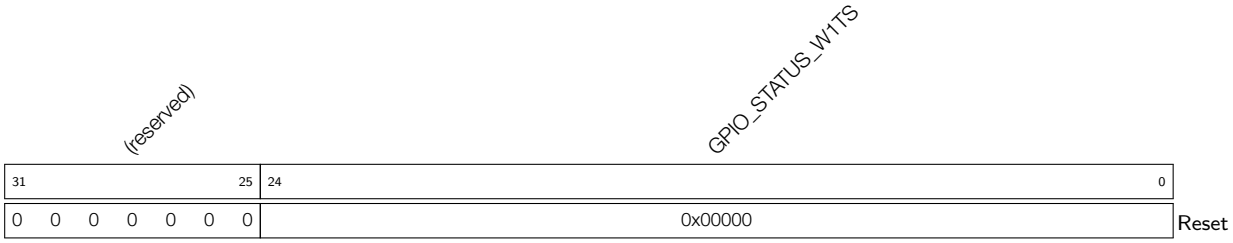
GPIO_IN_DATA_NEXT GPIO0 ~ 20 input value. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. Each bit represents a pin input value, 1 for high level and 0 for low level. (RO)

Register 5.9. GPIO_STATUS_REG (0x0044)



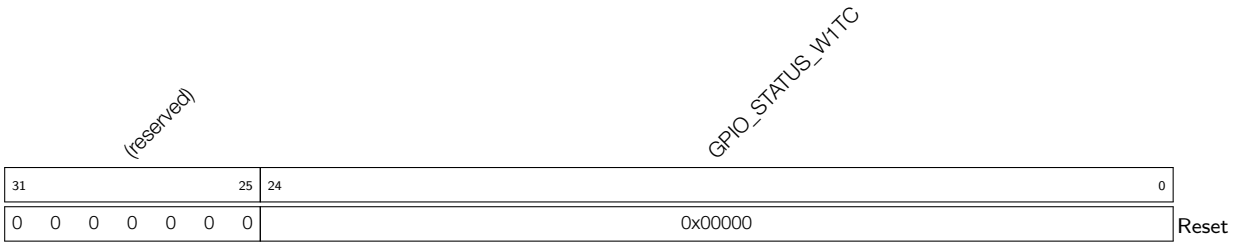
GPIO_STATUS_INTERRUPT GPIO0 ~ 20 interrupt status register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. (R/W/SS)

Register 5.10. GPIO_STATUS_W1TS_REG (0x0048)



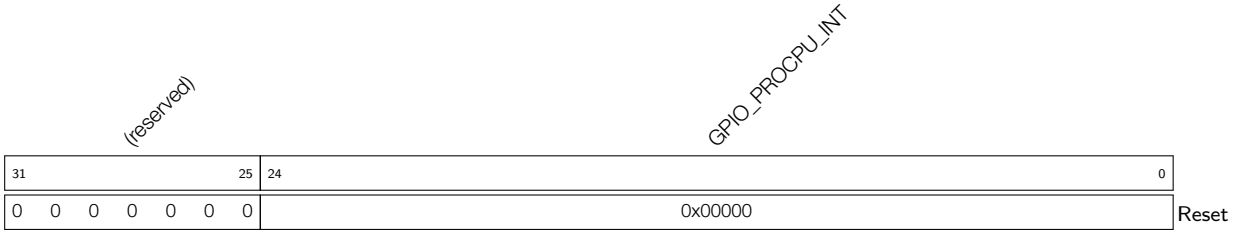
GPIO_STATUS_W1TS GPIO0 ~ 20 interrupt status set register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be set to 1. Recommended operation: use this register to set GPIO_STATUS_INTERRUPT. (WT)

Register 5.11. GPIO_STATUS_W1TC_REG (0x004C)



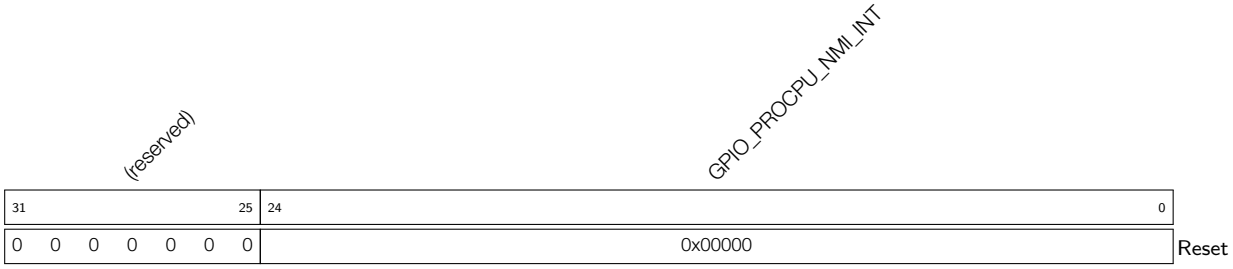
GPIO_STATUS_W1TC GPIO0 ~ 20 interrupt status clear register. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be cleared. Recommended operation: use this register to clear GPIO_STATUS_INTERRUPT. (WT)

Register 5.12. GPIO_PCPU_INT_REG (0x005C)



GPIO_PROCPU_INT GPIO0 ~ 20 CPU interrupt status. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit13 of GPIO_PIN*n*_REG). (RO)

Register 5.13. GPIO_PCPU_NMI_INT_REG (0x0060)



GPIO_PROCPU_NMI_INT GPIO0 ~ 20 CPU non-maskable interrupt status. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. This interrupt status is corresponding to the bit in GPIO_STATUS_REG when assert (high) enable signal (bit 14 of GPIO_PIN*n*_REG). (RO)

Register 5.14. GPIO_PIN*n*_REG (*n*: 0-20) (0x0074+4n*)**

GPIO_PIN*n*_SYNC2_BYPASS For the second-level synchronization, GPIO input data can be synchronized on either edge of the APB clock. 0: no synchronization; 1: synchronized on falling edge; 2 and 3: synchronized on rising edge. (R/W)

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

GPIO_PIN*n*_SYNC1_BYPASS For the first-level synchronization, GPIO input data can be synchronized on either edge of the APB clock. 0: no synchronization; 1: synchronized on falling edge; 2 and 3: synchronized on rising edge. (R/W)

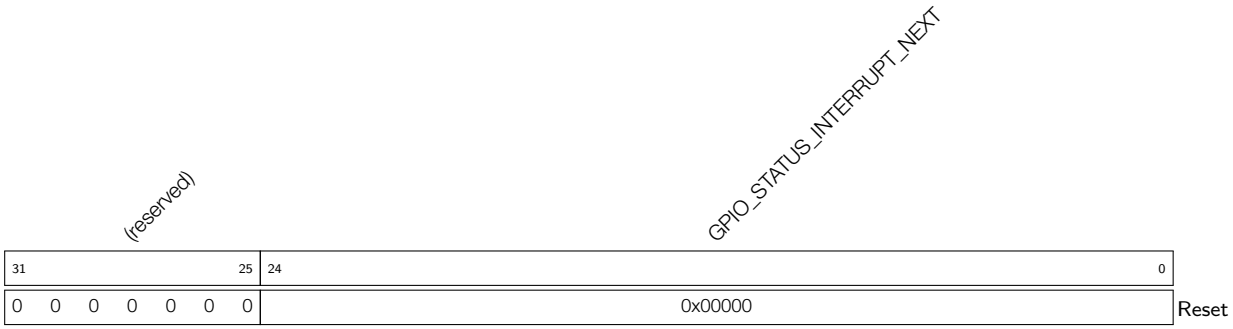
GPIO_PIN*n*_INT_TYPE Interrupt type selection. 0: GPIO interrupt disabled; 1: rising edge trigger; 2: falling edge trigger; 3: any edge trigger; 4: low level trigger; 5: high level trigger. (R/W)

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

GPIO_PIN*n*_CONFIG reserved (R/W)

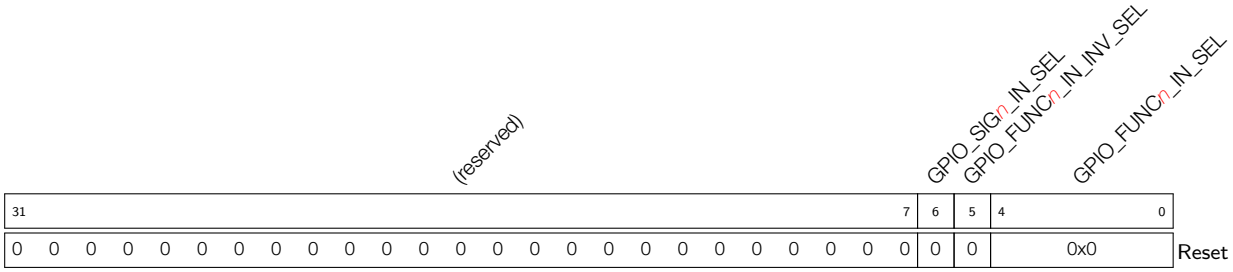
GPIO_PIN*n*_INT_ENA Interrupt enable bits. bit13: CPU interrupt enabled; bit14: CPU non-maskable interrupt enabled. (R/W)

Register 5.15. GPIO_STATUS_NEXT_REG (0x014C)



GPIO_STATUS_INTERRUPT_NEXT Interrupt source signal of GPIO0 ~ 20, could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt. Bit0 ~ bit20 are corresponding to GPIO0 ~ 20, and bit21 ~ bit24 are invalid. (RO)

Register 5.16. GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0-127) (0x0154+4n*)**

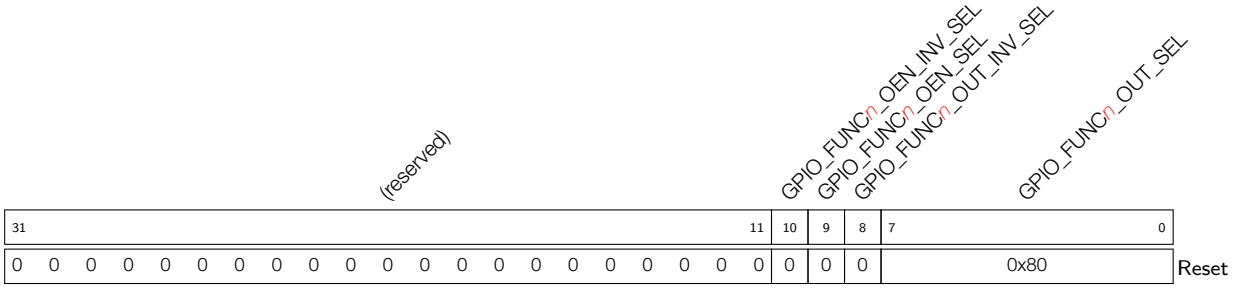


GPIO_FUNC*n*_IN_SEL Selection control for peripheral input signal *n*, selects a pin from the 21 GPIO matrix pins to connect this input signal. Or selects 0x1E for a constantly high input or 0x1F for a constantly low input. (R/W)

GPIO_FUNC*n*_IN_INV_SEL Invert the input value. 1: invert enabled; 0: invert disabled. (R/W)

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

Register 5.17. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-20) (0x0554+4n*)**



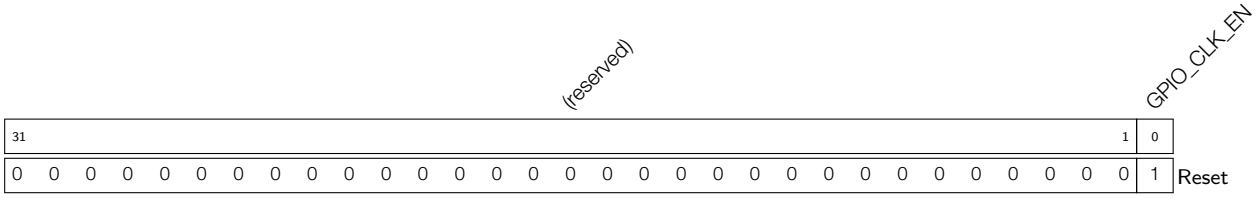
GPIO_FUNC*n*_OUT_SEL Selection control for GPIO output *n*. If a value *Y* (0<=Y<128) is written to this field, the peripheral output signal *Y* will be connected to GPIO output *n*. If a value 128 is written to this field, bit *n* of GPIO_OUT_REG and GPIO_ENABLE_REG will be selected as the output value and output enable. (R/W)

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

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

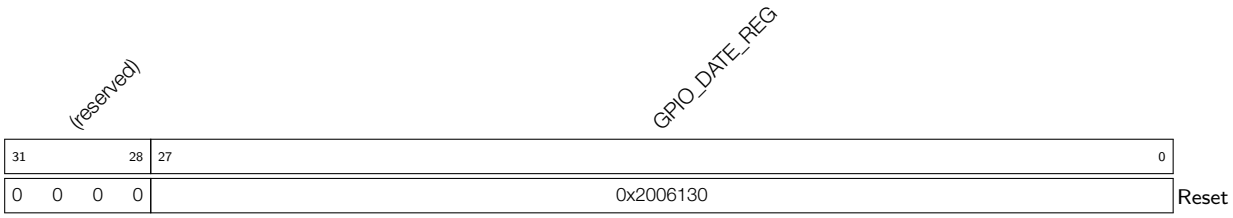
GPIO_FUNC*n*_OEN_INV_SEL 0: Do not invert the output enable signal; 1: Invert the output enable signal. (R/W)

Register 5.18. GPIO_CLOCK_GATE_REG (0x062C)



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

Register 5.19. GPIO_DATE_REG (0x06FC)

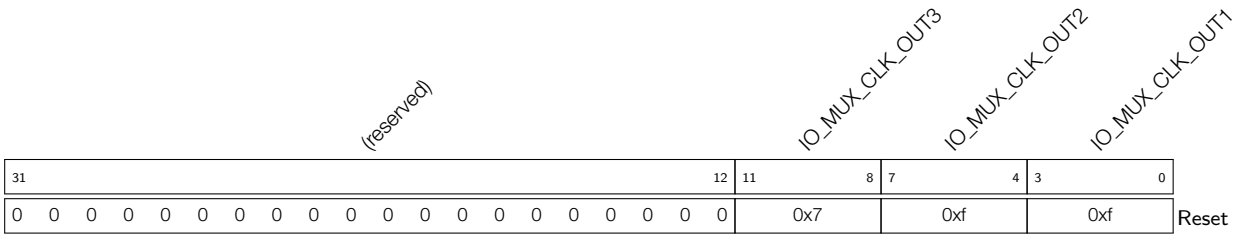


GPIO_DATE_REG Version control register (R/W)

5.15.2 IO MUX Registers

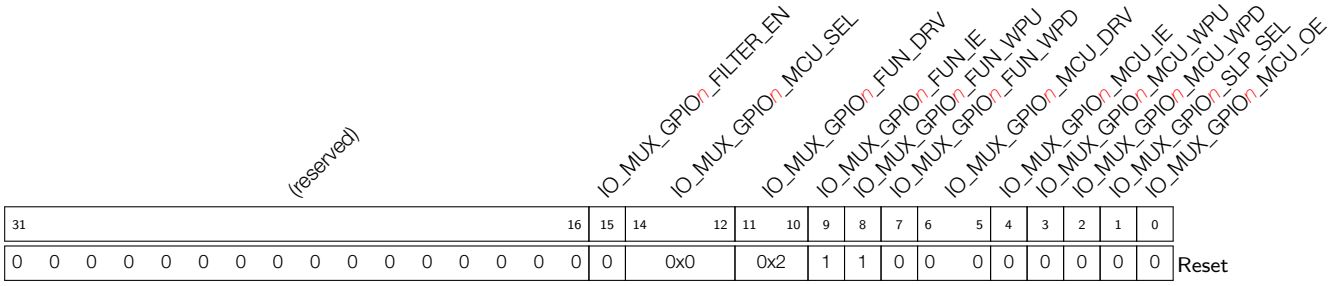
The addresses in this section are relative to the IO MUX base address provided in Table 3-3 in Chapte 3 *System and Memory* .

Register 5.20. IO_MUX_PIN_CTRL_REG (0x0000)



IO_MUX_CLK_OUT*x* If you want to output clock for I2S to CLK_OUT_out*x*, set IO_MUX_CLK_OUT*x* to 0x0. CLK_OUT_out*x* can be found in Table 5-2. (R/W)

Register 5.21. IO_MUX_GPIO*n*_REG (*n*: 0-20) (0x0004+4n*)**



IO_MUX_GPIO*n*_MCU_OE Output enable of the pin in sleep mode. 1: output enabled; 0: output disabled. (R/W)

IO_MUX_GPIO*n*_SLP_SEL Sleep mode selection of this pin. Set to 1 to put the pin in sleep mode. (R/W)

IO_MUX_GPIO*n*_MCU_WPD Pull-down enable of the pin in sleep mode. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

IO_MUX_GPIO*n*_MCU_WPU Pull-up enable of the pin during sleep mode. 1: internal pull-up enabled; 0: internal pull-up disabled. (R/W)

IO_MUX_GPIO*n*_MCU_IE Input enable of the pin during sleep mode. 1: input enabled; 0: input disabled. (R/W)

IO_MUX_GPIO*n*_MCU_DRV Configures the drive strength of GPIO*n* during sleep mode.
0: ~5 mA
1: ~ 10 mA
2: ~ 20 mA
3: ~40 mA
(R/W)

IO_MUX_GPIO*n*_FUN_WPD Pull-down enable of the pin. 1: internal pull-down enabled; 0: internal pull-down disabled. (R/W)

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

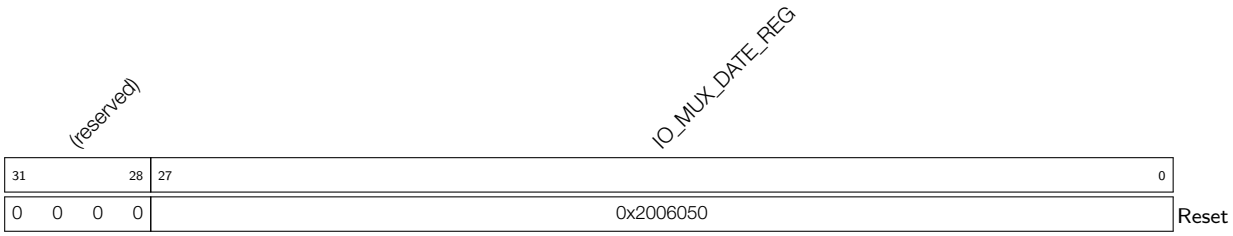
IO_MUX_GPIO*n*_FUN_IE Input enable of the pin. 1: input enabled; 0: input disabled. (R/W)

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

IO_MUX_GPIO*n*_MCU_SEL Select IO MUX function for this pin. 0: Select Function 0; 1: Select Function 1; etc. (R/W)

IO_MUX_GPIO*n*_FILTER_EN Enable filter for pin input signals. 1: filter enabled; 0: filter disabled. (R/W)

Register 5.22. IO_MUX_DATE_REG (0x00FC)



IO_MUX_DATE_REG Version control register (R/W)

6 Reset and Clock

6.1 Reset

6.1.1 Overview

ESP8684 provides four types of reset that occur at different levels, namely CPU Reset, Core Reset, System Reset, and Chip Reset. All reset types mentioned above (except Chip Reset) maintain the data stored in internal memory. Figure 6-1 shows the scope of affected subsystems by each type of reset.

6.1.2 Architectural Overview

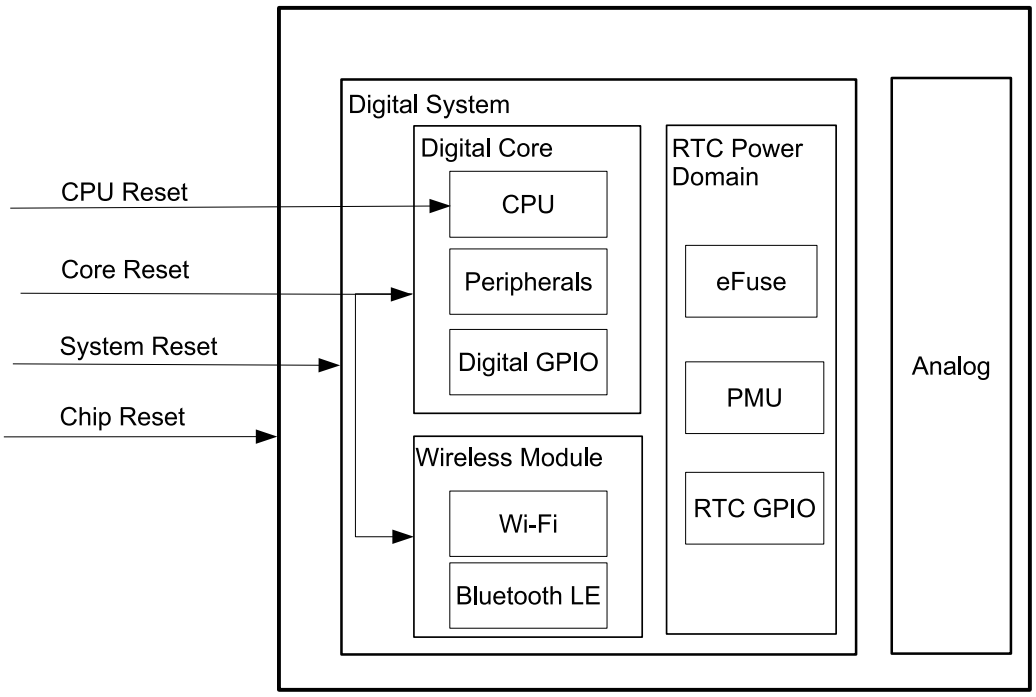


Figure 6-1. Reset Types

6.1.3 Features

- Support four reset types:
  - CPU Reset: Only resets CPU core. Once such reset is released, the instructions from the CPU reset vector will be executed.
  - Core Reset: Resets the whole digital system except RTC, including CPU, peripherals, Wi-Fi, Bluetooth® LE, and digital GPIOs.
  - System Reset: Resets the whole digital system, including RTC.
  - Chip Reset: Resets the whole chip.
- Support software reset and hardware reset:
  - Software Reset: triggered via software by configuring the corresponding registers of CPU, see Chapter 9 *Low-power Management (RTC_CNTL)*.

– Hardware Reset: triggered directly by the hardware.

Note:
If CPU is reset, SENSITIVE registers will be reset, too.

6.1.4 Functional Description

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

Table 6-1 lists possible reset sources and the types of reset they trigger.

Table 6-1. 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 12 *Watchdog Timers (WDT)*
0x12	Analog Super Watchdog reset	System Reset	See Chapter 12 *Watchdog Timers (WDT)*
0x13	Clock glitch reset	System Reset	See Chapter 1 *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 9 *Low-power Management (RTC_CNTL)*
0x07	MWDT0 core reset	Core Reset	See Chapter 12 *Watchdog Timers (WDT)*
0x09	RWDT core reset	Core Reset	See Chapter 12 *Watchdog Timers (WDT)*
0x14	eFuse reset	Core Reset	Triggered by eFuse CRC error
0x18	JTAG reset	CPU Reset	Triggered by JTAG
0x0B	MWDT0 CPU reset	CPU Reset	See Chapter 12 *Watchdog Timers (WDT)*
0x0C	Software CPU reset	CPU Reset	Triggered by configuring RTC_CNTL_SW_PROCPU_RST
0x0D	RWDT CPU reset	CPU Reset	See Chapter 12 *Watchdog Timers (WDT)*

[1] Chip Reset can be triggered by the following two sources:
- Triggered by chip power-on.
- Triggered by brown-out detector.

[2] Once brown-out status is detected, the detector will trigger System Reset or Chip Reset, depending on the configuration of RTC_CNTL_BROWN_OUT_RST_SEL. See Chapter 9 *Low-power Management (RTC_CNTL)*.

6.2 Clock

6.2.1 Overview

ESP8684 clocks are mainly sourced from external oscillator (OSC), RC, and PLL circuit, and then processed by the dividers or selectors, which allows most functional modules to select their working clock according to their

power consumption and performance requirements. Figure 6-2 shows the system clock structure.

6.2.2 Architectural Overview

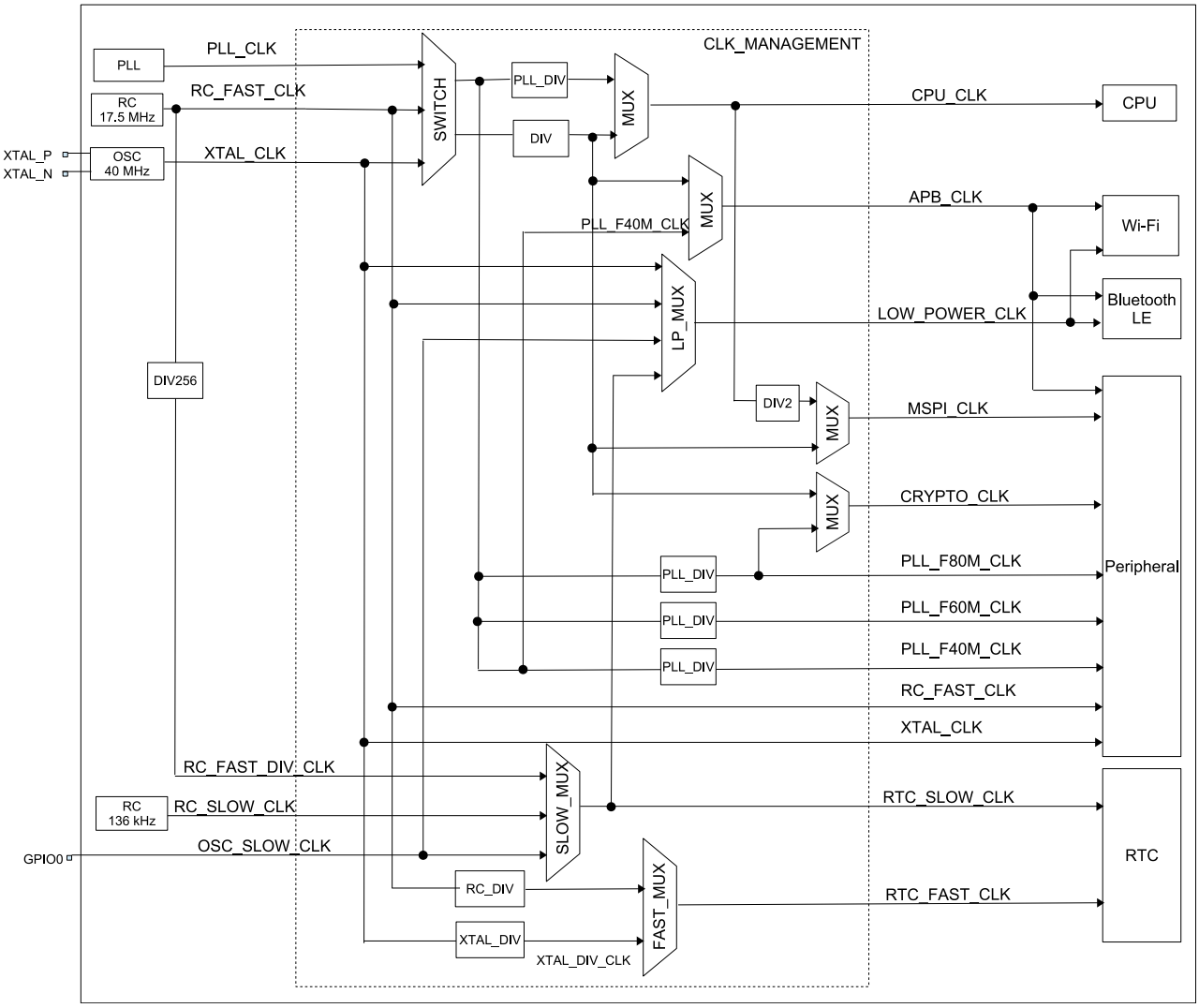


Figure 6-2. System Clock

6.2.3 Features

ESP8684 clocks can be classified in two types depending on their frequencies:

- High speed clocks for devices working at a higher frequency, such as CPU and digital peripherals
  - PLL_CLK (480 MHz): internal PLL clock
  - XTAL_CLK (40 MHz): external crystal clock
- Slow speed clocks for low-power devices, such as RTC module and low-power peripherals
  - OSC_SLOW_CLK (usually 32 kHz): external slow clock from GPIO0
  - RC_FAST_CLK (17.5 MHz by default): internal fast RC oscillator with adjustable frequency
  - FOSC_DIV_CLK: internal fast RC oscillator derived from RC_FAST_CLK divided by 256
  - RC_SLOW_CLK (136 kHz by default): internal slow RC oscillator with adjustable frequency

6.2.4 Functional Description

6.2.4.1 CPU Clock

As Figure 6-2 shows, CPU_CLK is the master clock for CPU and it can be as high as 120 MHz when CPU works in high performance mode. Alternatively, CPU can run at lower frequencies, such as at 2 MHz, to lower power consumption. Users can set PLL_CLK, RC_FAST_CLK or XTAL_CLK as CPU_CLK clock source by configuring SYSTEM_SOC_CLK_SEL, see Table 6-2 and Table 6-3. By default, the CPU clock is sourced from XTAL_CLK with a divider of 2, i.e. the CPU clock is 20 MHz.

Table 6-2. CPU Clock Source

SYSTEM_SOC_CLK_SEL	CPU Clock Source
0	XTAL_CLK
1	PLL_CLK
2	RC_FAST_CLK

Table 6-3. CPU Clock Frequency

CPU Clock Source	SEL_0*	SEL_1*	CPU Clock Frequency
XTAL_CLK	0	-	CPU_CLK = XTAL_CLK/(SYSTEM_PRE_DIV_CNT + 1) SYSTEM_PRE_DIV_CNT ranges from 0 ~ 1023. Default is 1
PLL_CLK	1	0	CPU_CLK = PLL_CLK/6 CPU_CLK frequency is 80 MHz
PLL_CLK	1	1	CPU_CLK = PLL_CLK/4 CPU_CLK frequency is 120 MHz
RC_FAST_CLK	2	-	CPU_CLK = RC_FAST_CLK/(SYSTEM_PRE_DIV_CNT + 1) SYSTEM_PRE_DIV_CNT ranges from 0 ~ 1023. Default is 1

* The value of SYSTEM_SOC_CLK_SEL

* The value of SYSTEM_CPUPERIOD_SEL

6.2.4.2 Peripheral Clock

Peripheral clocks are classified into two categories:

- Bus clock: APB_CLK
- Functional clocks: CRYPTO_CLK, PLL_F80M_CLK, PLL_F60M_CLK, PLL_F40M_CLK, MSPI_CLK, XTAL_CLK, and RC_FAST_CLK.

Table 6-4 shows which clock can be used by each peripheral.

Table 6-4. Peripheral Clocks

Peripheral	XTAL_CLK	RC_FAST_CLK	PLL_F40M_CLK	PLL_F60M_CLK	PLL_F80M_CLK	(RTC) RTC_FAST_CLK	CRYPTO_CLK	MSPI_CLK
Timer Group	Y		Y					
UART	Y	Y	Y					
I2C	Y	Y						
SPI	Y		Y					
LEDC	Y	Y		Y				
SAR ADC	Y				Y			
Temperature sensor	Y	Y						
System Timer	Y							
Crypto							Y	
MSPI								Y
eFuse						Y		

APB_CLK

The frequency of APB_CLK is determined by the clock source of CPU_CLK as shown in Table 6-5.

Table 6-5. APB_CLK Clock Frequency

CPU_CLK Source	APB_CLK Frequency
PLL_CLK	40 MHz
XTAL_CLK	CPU_CLK
RC_FAST_CLK	CPU_CLK

CRYPTO_CLK

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

Table 6-6. CRYPTO_CLK Frequency

CPU_CLK Source	CRYPTO_CLK Frequency
PLL_CLK	80 MHz
XTAL_CLK	CPU_CLK
RC_FAST_CLK	CPU_CLK

MSPI_CLK

The frequency of MSPI_CLK is determined by the CPU_CLK source, as shown in Table 6-7.

Table 6-7. MSPI_CLK Frequency

CPU_CLK Source	MSPI_CLK Frequency
PLL_CLK	CPU_CLK/2
XTAL_CLK	CPU_CLK
RC_FAST_CLK	CPU_CLK

PLL_F80M_CLK, PLL_F60M_CLK, PLL_F40M_CLK

PLL_F80M_CLK, PLL_F60M_CLK, and PLL_F40M_CLK are divided from PLL_CLK according to current PLL frequency.

6.2.4.3 Wireless Clock

The wireless clock (LOW_POWER_CLK) in ESP8684 is used for Wi-Fi and Bluetooth LE in low-power mode. The clock source of LOW_POWER_CLK can be:

- OSC_SLOW_CLK
- XTAL_CLK
- RC_FAST_CLK
- RTC_SLOW_CLK (the low clock selected by RTC)

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

6.2.4.4 RTC Clock

RTC module can operate when most other clocks are stopped. RTC clocks include RTC_SLOW_CLK and RTC_FAST_CLK.

The clock sources for RTC_SLOW_CLK and RTC_FAST_CLK are low-frequency clocks:

- RTC_SLOW_CLK, used to clock RTC timer, RTC watch dog, and low-power controller, can be derived from:
  - OSC_SLOW_CLK
  - RC_SLOW_CLK
  - or FOSC_DIV_CLK
- RTC_FAST_CLK, used to clock RTC peripherals and on-chip sensor module, can be derived from
  - XTAL_CLK divided by 2
  - or RC_FAST_CLK divided by N

7 Chip Boot Control

7.1 Overview

Strapping pins are the specific chip pins used to control the following functions during chip power-on or hardware reset of ESP8684:

- control chip boot mode
- enable or disable ROM code printing to UART

ESP8684 has two strapping pins:

- GPIO8
- GPIO9

During power-on reset, RTC watchdog reset, and brownout reset, (see Chapter 6 *Reset and Clock*), hardware captures samples and stores the voltage level of strapping pins as strapping bit of "0" or "1" in latches, and holds these bits until the chip is powered down. Software can read the latch status (strapping value) from GPIO_STRAPPING.

7.2 Features

- Control of chip function on boot with strapping pins:
  - GPIO8
  - GPIO9
- Able to control chip boot mode:
  - SPI Boot mode
  - Download Boot mode
- Able to control ROM code printing to UART
- Allow the reading of strapping pin values from GPIO_STRAPPING

7.3 Functional Description

This section provides description of the chip functions and the pattern of the strapping pins values to invoke each function.

Notice:

Only documented patterns should be used. If some pattern is not documented, it may trigger unexpected behavior.

7.3.1 Default Configuration

By default, GPIO9 is connected to the chip's internal pull-up resistor. If GPIO9 is not connected or connected to an external high-impedance circuit, the internal weak pull-up determines the default input level of this strapping

pin (see Table 7-1).

Table 7-1. Default Configuration of Strapping Pins

Strapping Pin	Default Configuration
GPIO8	N/A
GPIO9	Pull-up

To change the strapping bit values, users can apply external pull-down/pull-up resistors, or use host MCU GPIOs to control the voltage level of these pins when powering on ESP8684. After the reset is released, the strapping pins work as normal-function pins.

7.3.2 Boot Mode Control

The values of GPIO8 and GPIO9 at reset determine the boot mode after the reset is released.

Table 7-2. Boot Mode Control

Boot Mode	GPIO8	GPIO9
SPI Boot	x	1
Download Boot	1	0

Table 7-2 shows the strapping pin values of GPIO8 and GPIO9, and the associated boot modes. "x" means that this value is ignored.

In SPI Boot mode, the ROM bootloader loads and executes the program from SPI flash. SPI Boot mode can be further classified as follows:

- Normal Flash Boot: supports Secure Boot. The ROM bootloader loads the program from flash into RAM and executes it. In most practical scenarios, this program is the 2nd stage bootloader, which in turn boots the final application.
- Direct Boot: does not support Secure Boot and programs run directly from flash. To enable this mode, make sure that the first two words of the bin file downloaded to flash are 0xaedb041d. For more detailed process, see Figure 7-1.

In Download Boot mode, users can download code into flash using UART0 interface. It is also possible to load a program into SRAM and execute it from SRAM.

Figure 7-1 shows the detailed boot flow of the chip.

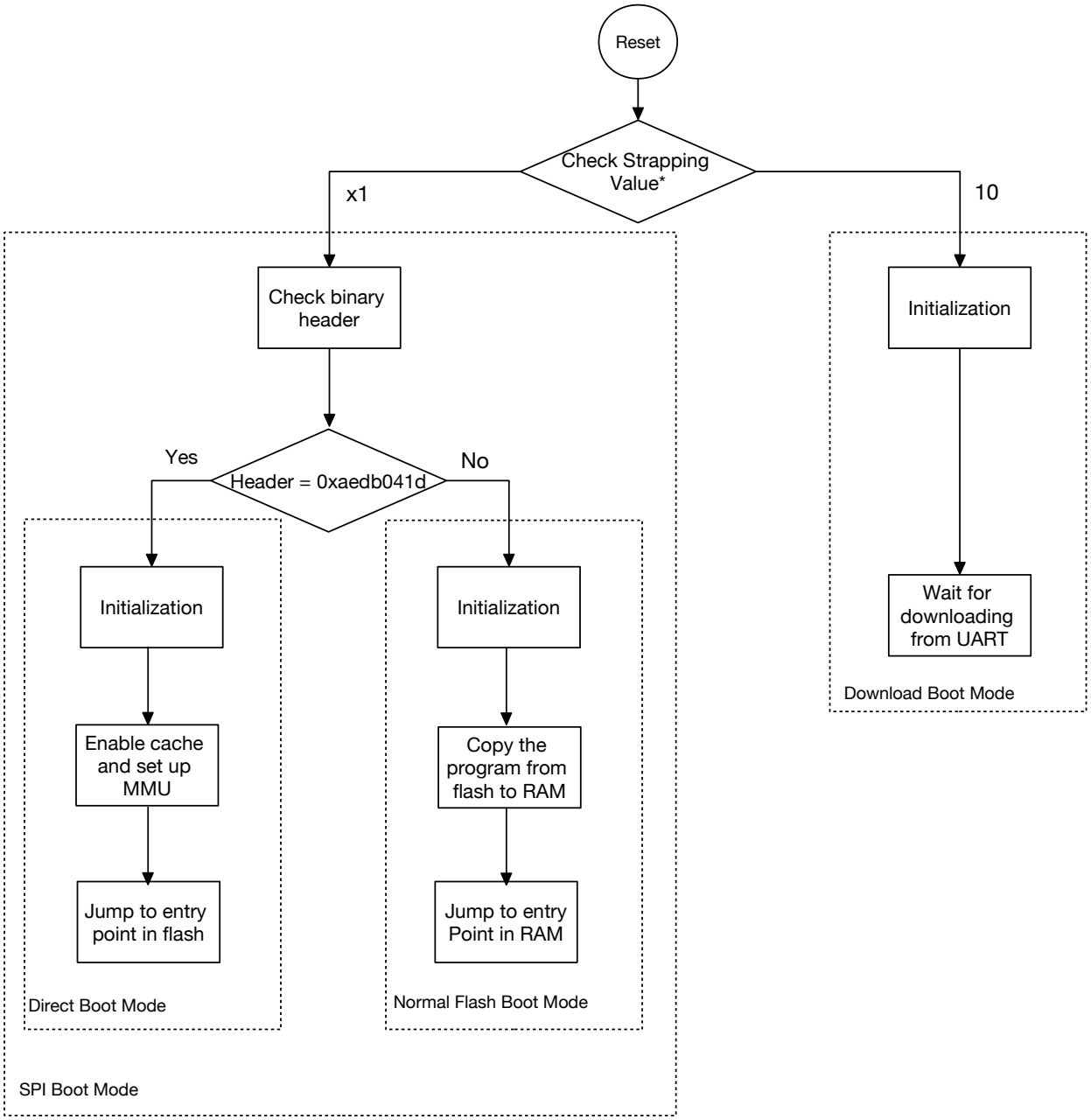


Figure 7-1. Chip Boot Flow

The following registers control boot mode behaviors:

- RTC_CNTL_FORCE_DOWNLOAD_BOOT

  Software can force switch the chip from SPI Boot mode to Download Boot mode by setting register RTC_CNTL_FORCE_DOWNLOAD_BOOT and triggering a CPU reset.

- 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.

7.3.3 ROM Code Printing Control

GPIO8 controls ROM code printing during the early SPI boot process. This GPIO is used together with EFUSE_UART_PRINT_CONTROL.

Table 7-3. ROM Code Printing Control

eFuse[1]	GPIO8	ROM Code Printing Behavior
0	x	ROM code always prints to UART during boot. The value of GPIO8 is ignored.
1	0	Print is enabled during boot.
	1	Print is disabled during boot.
2	0	Print is disabled during boot.
	1	Print is enabled during boot.
3	x	Print is always disabled during boot. The value of GPIO8 is ignored.

[1] eFuse: EFUSE_UART_PRINT_CONTROL

8 Interrupt Matrix (INTMTRX)

8.1 Overview

The interrupt matrix embedded in ESP8684 independently routes peripheral interrupt sources to the ESP-RISC-V CPU's peripheral interrupts, to timely inform CPU to process the coming interrupts.

The ESP8684 has 43 peripheral interrupt sources. To map them to 31 CPU interrupts, this interrupt matrix is needed.

Note:
This chapter focuses on how to map peripheral interrupt sources to CPU interrupts. For more details about interrupt configuration, vector, and ISA suggested operations, please refer to Chapter 1 *ESP-RISC-V CPU*.

8.2 Features

Interrupt matrix has the following features:

- 43 peripheral interrupt sources as input
- 31 CPU peripheral interrupts as output
- Able to query current status of peripheral interrupt sources
- Configurable priority, type, threshold, and enable signal of CPU interrupts

Figure 8-1 shows the structure of the interrupt matrix.

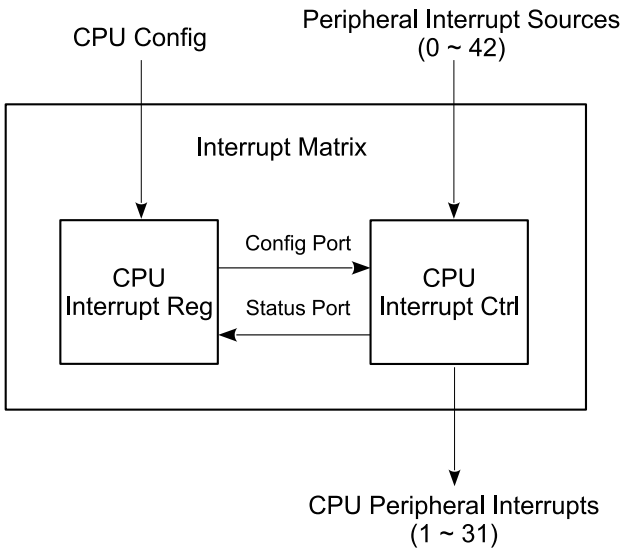


Figure 8-1. Interrupt Matrix Structure

8.3 Functional Description

8.3.1 Peripheral Interrupt Sources

The ESP8684 has 43 peripheral interrupt sources in total. Table 8-1 lists all these sources and their configuration/status registers.

- Column “Index”: Peripheral interrupt source index, can be 0 ~ 42.
- Column “Chapter”: In which chapter the interrupt source is described in details.
- Column “Source”: Name of the peripheral interrupt source.
- Column “Configuration Register”: Registers used to configure routing of the peripheral interrupt sources to CPU peripheral interrupts
- Column “Status Register”: Registers used for indicating the interrupt status of peripheral interrupt sources.
  - Column “Status Register - Bit”: Bit position in status register, indicating the interrupt status.
  - Column “Status Register - Name”: Name of status registers.

Table 8-1. CPU Peripheral Interrupt Configuration/Status Registers and Peripheral Interrupt Sources

Index	Chapter	Source	Configuration Register	Status Register	
				Bit	Name
0	N/A	reserved	reserved	0	
1	N/A	reserved	reserved	1	
2	N/A	reserved	reserved	2	
3	N/A	reserved	reserved	3	
4	N/A	reserved	reserved	4	
5	N/A	reserved	reserved	5	
6	N/A	reserved	reserved	6	
7	N/A	reserved	reserved	7	
8	N/A	reserved	reserved	8	
9	N/A	reserved	reserved	9	
10	N/A	reserved	reserved	10	
11	N/A	reserved	reserved	11	
12	N/A	reserved	reserved	12	
13	*IO MUX and GPIO Matrix (GPIO, IO MUX)*	GPIO_PROCPU_INTR	INTERRUPT_CORE0_GPIO_INTERRUPT_PRO_MAP_REG	13	
14	*IO MUX and GPIO Matrix (GPIO, IO MUX)*	GPIO_PROCPU_NMI_INTR	INTERRUPT_CORE0_GPIO_INTERRUPT_PRO_NMI_MAP_REG	14	INTERRUPT_CORE0_INTR_STATUS_0_REG
15	N/A	reserved	reserved	15	
16	*SPI Controller (SPI)*	GPSPI2_INTR_2	INTERRUPT_CORE0_SPI_INTR_2_MAP_REG	16	
17	*UART Controller (UART)*	UART_INTR	INTERRUPT_CORE0_UART_INTR_MAP_REG	17	
18	*UART Controller (UART)*	UART1_INTR	INTERRUPT_CORE0_UART1_INTR_MAP_REG	18	
19	*LED PWM Controller (LEDC)*	LEDC_INTR	INTERRUPT_CORE0_LEDC_INT_MAP_REG	19	
20	*eFuse Controller (eFuse)*	EFUSE_INTR	INTERRUPT_CORE0_EFUSE_INT_MAP_REG	20	
21	*Low-power Management (RTC_CNTL)*	RTC_CNTL_INTR	INTERRUPT_CORE0_RTC_CORE_INTR_MAP_REG	21	
22	*I2C Master Controller (I2C)*	I2C_EXT0_INTR	INTERRUPT_CORE0_I2C_EXT0_INTR_MAP_REG	22	
23	*Timer Group (TIMG)*	TG_T0_INTR	INTERRUPT_CORE0_TG_T0_INT_MAP_REG	23	
24	*Timer Group (TIMG)*	TG_WDT_INTR	INTERRUPT_CORE0_TG_WDT_INT_MAP_REG	24	
25	N/A	reserved	reserved	25	
26	*System Timer (SYSTIMER)*	SYSTIMER_TARGET0_INTR	INTERRUPT_CORE0_SYSTIMER_TARGET0_INT_MAP_REG	26	
27	*System Timer (SYSTIMER)*	SYSTIMER_TARGET1_INTR	INTERRUPT_CORE0_SYSTIMER_TARGET1_INT_MAP_REG	27	
28	*System Timer (SYSTIMER)*	SYSTIMER_TARGET2_INTR	INTERRUPT_CORE0_SYSTIMER_TARGET2_INT_MAP_REG	28	
29	N/A	reserved	reserved	29	
30	N/A	reserved	reserved	30	

Index	Chapter	Source	Configuration Register	Status Register	
				Bit	Name
31	N/A	reserved	reserved	31	
32	*On-Chip Sensor and Analog Signal Processing*	DIGTAL_ADC_INTR	INTERRUPT_CORE0_APB_ADC_INT_MAP_REG	0	INTERRUPT_CORE0_INTR_STATUS_1_REG
33	*GDMA Controller (GDMA)*	GDMA_CH0_INTR	INTERRUPT_CORE0_DMA_CH0_INT_MAP_REG	1	
34	*SHA Accelerator (SHA)*	SHA_INTR	INTERRUPT_CORE0_SHA_INTR_MAP_REG	2	
35	*ECC Hardware Accelerator (ECC)*	ECC_INTR	INTERRUPT_CORE0_ECC_INTR_MAP_REG	3	
36	*System Registers (SYSTEM)*	SW_INTR_0	INTERRUPT_CORE0_CPU_INTR_FROM_CPU_0_MAP_REG	4	
37	*System Registers (SYSTEM)*	SW_INTR_1	INTERRUPT_CORE0_CPU_INTR_FROM_CPU_1_MAP_REG	5	
38	*System Registers (SYSTEM)*	SW_INTR_2	INTERRUPT_CORE0_CPU_INTR_FROM_CPU_2_MAP_REG	6	
39	*System Registers (SYSTEM)*	SW_INTR_3	INTERRUPT_CORE0_CPU_INTR_FROM_CPU_3_MAP_REG	7	
40	*Debug Assistant (ASSIST_DEBUG)*	ASSIST_DEBUG_INTR	INTERRUPT_CORE0_ASSIST_DEBUG_INTR_MAP_REG	8	
41	N/A	PERI_VIO_SIZE_INTR	INTERRUPT_CORE0_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR_MAP_REG	9	
42	N/A	reserved	reserved	10	

8.3.2 CPU Interrupts

The ESP8684 implements its interrupt mechanism using an interrupt controller instead of RISC-V Privileged ISA specification. The ESP-RISC-V CPU has 31 interrupts, with unique IDs (1 ~ 31). Each CPU interrupt has the following properties.

- Priority levels from 1 (lowest) to 15 (highest).
- Configurable as high-level triggered or rising-edge triggered.
- Programmable threshold for masking interrupts with lower priority.

Note:
For detailed information about how to configure CPU interrupts, see Chapter 1 *ESP-RISC-V CPU*.

8.3.3 Allocate Peripheral Interrupt Source to CPU Interrupt

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

- Source_*X*: stands for a peripheral interrupt source, wherein *X* means the index of this interrupt source in Table 8-1.
- INTERRUPT_CORE0_SOURCE_*X*_MAP_REG: stands for a configuration register, mapping Source_*X* to CPU interrupt.
- Num_P: the ID of CPU interrupts, can be 1 ~ 31.
- Interrupt_P: stands for the CPU interrupt with ID = Num_P.

8.3.3.1 Allocate one peripheral interrupt source (Source_*X*) to CPU

Setting the corresponding configuration register INTERRUPT_CORE0_SOURCE_*X*_MAP_REG of Source_*X* to Num_P allocates this interrupt source to Interrupt_P.

8.3.3.2 Allocate multiple peripheral interrupt sources (Source_X*n*) to CPU

Setting the corresponding configuration register INTERRUPT_CORE0_SOURCE_X*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 Interrupt_P. When an interrupt signal is generated, interrupt service routine (ISR) should check the interrupt status registers to figure out which peripheral generated the interrupt. For more information, see Chapter 1 *ESP-RISC-V CPU*.

8.3.3.3 Disable CPU peripheral interrupt source (Source_*X*)

Clearing the configuration register INTERRUPT_CORE0_SOURCE_*X*_MAP_REG disables the corresponding interrupt source.

8.3.4 Query Current Interrupt Status of Peripheral Interrupt Source

Users can query current interrupt status of a peripheral interrupt source by reading the bit value in INTERRUPT_CORE0_INTR_STATUS_*n*_REG (read only). For the mapping between INTERRUPT_CORE0_INTR_STATUS_*n*_REG and peripheral interrupt sources, please refer to Table 8-1.

8.4 Register Summary

The addresses in this section are relative to the interrupt matrix base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Name	Description	Address	Access
Interrupt source mapping register			
INTERRUPT_CORE0_GPIO_INTERRUPT_PRO_MAP_REG	GPIO_INTERRUPT_PRO mapping register	0x0034	R/W
INTERRUPT_CORE0_GPIO_INTERRUPT_PRO_NMI_MAP_REG	GPIO_INTERRUPT_PRO_NMI mapping register	0x0038	R/W
INTERRUPT_CORE0_SPI_INTR_2_MAP_REG	SPI_INTR_2 mapping register	0x0040	R/W
INTERRUPT_CORE0_UART_INTR_MAP_REG	UART_INTR mapping register	0x0044	R/W
INTERRUPT_CORE0_UART1_INTR_MAP_REG	UART1_INTR mapping register	0x0048	R/W
INTERRUPT_CORE0_LEDC_INT_MAP_REG	LEDC_INT mapping register	0x004C	R/W
INTERRUPT_CORE0_EFUSE_INT_MAP_REG	EFUSE_INT mapping register	0x0050	R/W
INTERRUPT_CORE0_RTC_CORE_INTR_MAP_REG	RTC_CORE_INTR mapping register	0x0054	R/W
INTERRUPT_CORE0_I2C_EXT0_INTR_MAP_REG	I2C_EXT0_INTR mapping register	0x0058	R/W
INTERRUPT_CORE0_TG_T0_INT_MAP_REG	TG_T0_INT mapping register	0x005C	R/W
INTERRUPT_CORE0_TG_WDT_INT_MAP_REG	TG_WDT_INT mapping register	0x0060	R/W
INTERRUPT_CORE0_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0_INT mapping register	0x0068	R/W
INTERRUPT_CORE0_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1_INT mapping register	0x006C	R/W
INTERRUPT_CORE0_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2_INT mapping register	0x0070	R/W
INTERRUPT_CORE0_APB_ADC_INT_MAP_REG	APB_ADC_INT mapping register	0x0080	R/W
INTERRUPT_CORE0_DMA_CH0_INT_MAP_REG	DMA_CH0_INT mapping register	0x0084	R/W
INTERRUPT_CORE0_SHA_INT_MAP_REG	SHA_INT mapping register	0x0088	R/W
INTERRUPT_CORE0_ECC_INT_MAP_REG	ECC_INT mapping register	0x008C	R/W
INTERRUPT_CORE0_CPU_INTR_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 mapping register	0x0090	R/W
INTERRUPT_CORE0_CPU_INTR_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 mapping register	0x0094	R/W
INTERRUPT_CORE0_CPU_INTR_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 mapping register	0x0098	R/W
INTERRUPT_CORE0_CPU_INTR_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 mapping register	0x009C	R/W
INTERRUPT_CORE0_ASSIST_DEBUG_INTR_MAP_REG	ASSIST_DEBUG_INTR mapping register	0x00A0	R/W
INTERRUPT_CORE0_SIZE_INTR_MAP_REG	PIF_PMS_MONITOR_VIOLATE_SIZE_INTR mapping register	0x00A4	R/W
Interrupt source status register			

Name	Description	Address	Access
INTERRUPT_CORE0_INTR_STATUS_0_REG	Interrupt source status register 0	0x00AC	RO
INTERRUPT_CORE0_INTR_STATUS_1_REG	Interrupt source status register 1	0x00B0	RO
Clock register			
INTERRUPT_CORE0_CLOCK_GATE_REG	Clock register	0x00B4	R/W
CPU interrupt register			
INTERRUPT_CORE0_CPU_INT_ENABLE_REG	Enable register for CPU interrupts	0x00B8	R/W
INTERRUPT_CORE0_CPU_INT_TYPE_REG	Type configuration register for CPU interrupts	0x00BC	R/W
INTERRUPT_CORE0_CPU_INT_CLEAR_REG	CPU interrupt clear register	0x00C0	R/W
INTERRUPT_CORE0_CPU_INT_EIP_STATUS_REG	Pending status register for CPU interrupts	0x00C4	RO
INTERRUPT_CORE0_CPU_INT_PRI_1_REG	Priority configuration register for CPU interrupt 1	0x00CC	R/W
INTERRUPT_CORE0_CPU_INT_PRI_2_REG	Priority configuration register for CPU interrupt 2	0x00D0	R/W
INTERRUPT_CORE0_CPU_INT_PRI_3_REG	Priority configuration register for CPU interrupt 3	0x00D4	R/W
INTERRUPT_CORE0_CPU_INT_PRI_4_REG	Priority configuration register for CPU interrupt 4	0x00D8	R/W
INTERRUPT_CORE0_CPU_INT_PRI_5_REG	Priority configuration register for CPU interrupt 5	0x00DC	R/W
INTERRUPT_CORE0_CPU_INT_PRI_6_REG	Priority configuration register for CPU interrupt 6	0x00E0	R/W
INTERRUPT_CORE0_CPU_INT_PRI_7_REG	Priority configuration register for CPU interrupt 7	0x00E4	R/W
INTERRUPT_CORE0_CPU_INT_PRI_8_REG	Priority configuration register for CPU interrupt 8	0x00E8	R/W
INTERRUPT_CORE0_CPU_INT_PRI_9_REG	Priority configuration register for CPU interrupt 9	0x00EC	R/W
INTERRUPT_CORE0_CPU_INT_PRI_10_REG	Priority configuration register for CPU interrupt 10	0x00F0	R/W
INTERRUPT_CORE0_CPU_INT_PRI_11_REG	Priority configuration register for CPU interrupt 11	0x00F4	R/W
INTERRUPT_CORE0_CPU_INT_PRI_12_REG	Priority configuration register for CPU interrupt 12	0x00F8	R/W
INTERRUPT_CORE0_CPU_INT_PRI_13_REG	Priority configuration register for CPU interrupt 13	0x00FC	R/W
INTERRUPT_CORE0_CPU_INT_PRI_14_REG	Priority configuration register for CPU interrupt 14	0x0100	R/W
INTERRUPT_CORE0_CPU_INT_PRI_15_REG	Priority configuration register for CPU interrupt 15	0x0104	R/W
INTERRUPT_CORE0_CPU_INT_PRI_16_REG	Priority configuration register for CPU interrupt 16	0x0108	R/W
INTERRUPT_CORE0_CPU_INT_PRI_17_REG	Priority configuration register for CPU interrupt 17	0x010C	R/W
INTERRUPT_CORE0_CPU_INT_PRI_18_REG	Priority configuration register for CPU interrupt 18	0x0110	R/W
INTERRUPT_CORE0_CPU_INT_PRI_19_REG	Priority configuration register for CPU interrupt 19	0x0114	R/W
INTERRUPT_CORE0_CPU_INT_PRI_20_REG	Priority configuration register for CPU interrupt 20	0x0118	R/W

Name	Description	Address	Access
INTERRUPT_CORE0_CPU_INT_PRI_21_REG	Priority configuration register for CPU interrupt 21	0x011C	R/W
INTERRUPT_CORE0_CPU_INT_PRI_22_REG	Priority configuration register for CPU interrupt 22	0x0120	R/W
INTERRUPT_CORE0_CPU_INT_PRI_23_REG	Priority configuration register for CPU interrupt 23	0x0124	R/W
INTERRUPT_CORE0_CPU_INT_PRI_24_REG	Priority configuration register for CPU interrupt 24	0x0128	R/W
INTERRUPT_CORE0_CPU_INT_PRI_25_REG	Priority configuration register for CPU interrupt 25	0x012C	R/W
INTERRUPT_CORE0_CPU_INT_PRI_26_REG	Priority configuration register for CPU interrupt 26	0x0130	R/W
INTERRUPT_CORE0_CPU_INT_PRI_27_REG	Priority configuration register for CPU interrupt 27	0x0134	R/W
INTERRUPT_CORE0_CPU_INT_PRI_28_REG	Priority configuration register for CPU interrupt 28	0x0138	R/W
INTERRUPT_CORE0_CPU_INT_PRI_29_REG	Priority configuration register for CPU interrupt 29	0x013C	R/W
INTERRUPT_CORE0_CPU_INT_PRI_30_REG	Priority configuration register for CPU interrupt 30	0x0140	R/W
INTERRUPT_CORE0_CPU_INT_PRI_31_REG	Priority configuration register for CPU interrupt 31	0x0144	R/W
INTERRUPT_CORE0_CPU_INT_THRESH_REG	Threshold configuration register for CPU interrupts	0x0148	R/W
Version register			
INTERRUPT_CORE0_INTERRUPT_DATE_REG	Version control register	0x07FC	R/W

8.5 Registers

The addresses in this section are relative to the interrupt matrix base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Register 8.1. INTERRUPT_CORE0_*GPIO_INTERRUPT_PRO_MAP*_REG (0x0034)

Register 8.2. INTERRUPT_CORE0_*GPIO_INTERRUPT_PRO_NMI_MAP*_REG (0x0038)

Register 8.3. INTERRUPT_CORE0_*SPI_INTR_2_MAP*_REG (0x0040)

Register 8.4. INTERRUPT_CORE0_*UART_INTR_MAP*_REG (0x0044)

Register 8.5. INTERRUPT_CORE0_*UART1_INTR_MAP*_REG (0x0048)

Register 8.6. INTERRUPT_CORE0_*LEDC_INT_MAP*_REG (0x004C)

Register 8.7. INTERRUPT_CORE0_*EFUSE_INT_MAP*_REG (0x0050)

Register 8.8. INTERRUPT_CORE0_*RTC_CORE_INTR_MAP*_REG (0x0054)

Register 8.9. INTERRUPT_CORE0_*I2C_EXT0_INTR_MAP*_REG (0x058)

Register 8.10. INTERRUPT_CORE0_*TG_T0_INT_MAP*_REG (0x005C)

Register 8.11. INTERRUPT_CORE0_*TG_WDT_INT_MAP*_REG (0x0060)

Register 8.12. INTERRUPT_CORE0_*SYSTIMER_TARGET0_INT_MAP*_REG (0x0068)

Register 8.13. INTERRUPT_CORE0_*SYSTIMER_TARGET1_INT_MAP*_REG (0x006C)

Register 8.14. INTERRUPT_CORE0_*SYSTIMER_TARGET2_INT_MAP*_REG (0x0070)

Register 8.15. INTERRUPT_CORE0_*APB_ADC_INT_MAP*_REG (0x0080)

Register 8.16. INTERRUPT_CORE0_*DMA_CH0_INT_MAP*_REG (0x0084)

Register 8.17. INTERRUPT_CORE0_*SHA_INT_MAP*_REG (0x0088)

Register 8.18. INTERRUPT_CORE0_*ECC_INT_MAP*_REG (0x008C)

Register 8.19. INTERRUPT_CORE0_*CPU_INTR_FROM_CPU_0_MAP*_REG (0x0090)

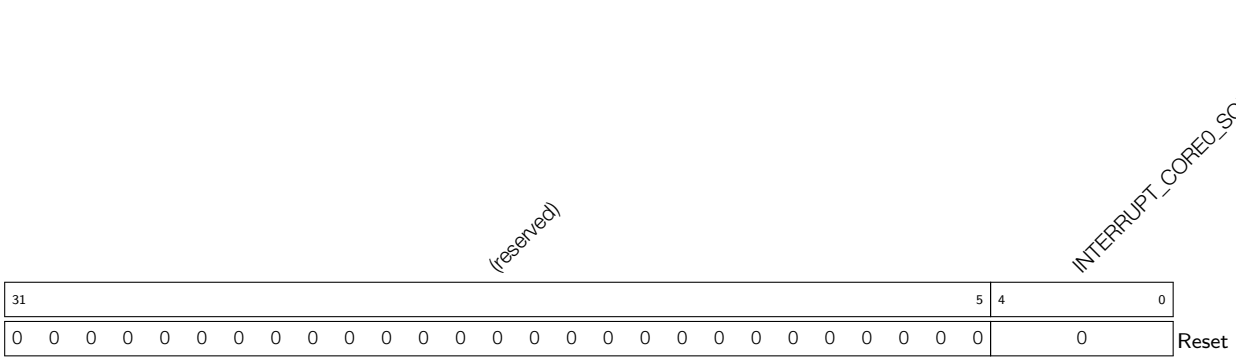
Register 8.20. INTERRUPT_CORE0_*CPU_INTR_FROM_CPU_1_MAP*_REG (0x0094)

Register 8.21. INTERRUPT_CORE0_*CPU_INTR_FROM_CPU_2_MAP*_REG (0x0098)

Register 8.22. INTERRUPT_CORE0_*CPU_INTR_FROM_CPU_3_MAP*_REG (0x009C)

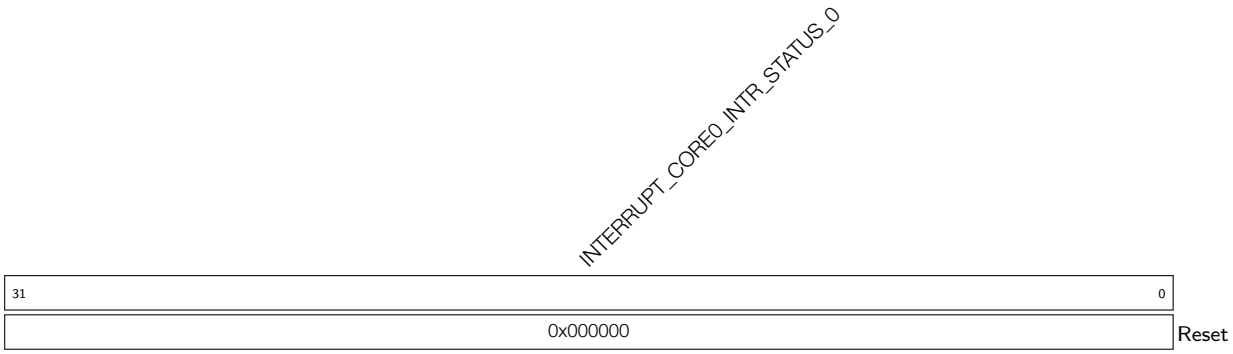
Register 8.23. INTERRUPT_CORE0_*ASSIST_DEBUG_INTR_MAP*_REG (0x00A0)

Register 8.24. INTERRUPT_CORE0_*PIF_PMS_MONITOR_VIOLATE_SIZE_INTR_MAP*_REG (0x00A4)



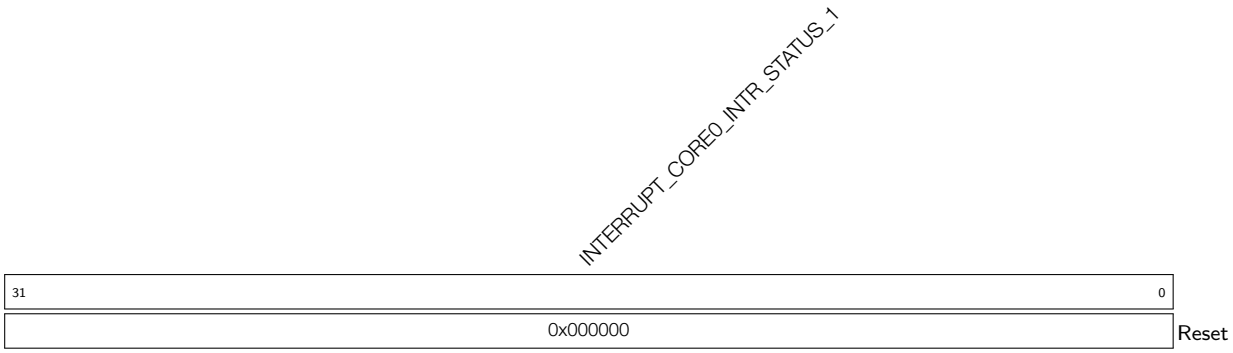
INTERRUPT_CORE0_SOURCE_*X*_MAP Map the interrupt source (SOURCE_*X*) into one CPU interrupt. For the information of SOURCE_*X*, see Table 8-1. (R/W)

Register 8.25. INTERRUPT_CORE0_INTR_STATUS_0_REG (0x00AC)



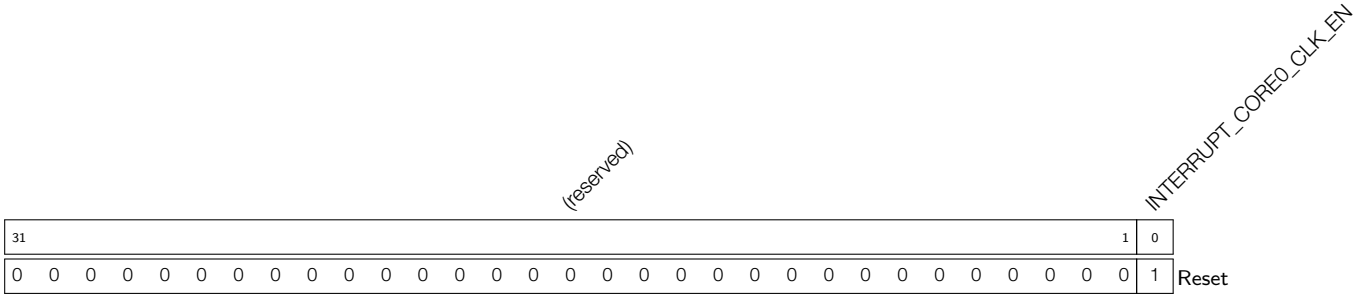
INTERRUPT_CORE0_INTR_STATUS_0 This register stores the status of the first 32 interrupt sources: 0 ~ 31. If the bit is 1 here, it means the corresponding source triggered an interrupt. (RO)

Register 8.26. INTERRUPT_CORE0_INTR_STATUS_1_REG (0x00B0)



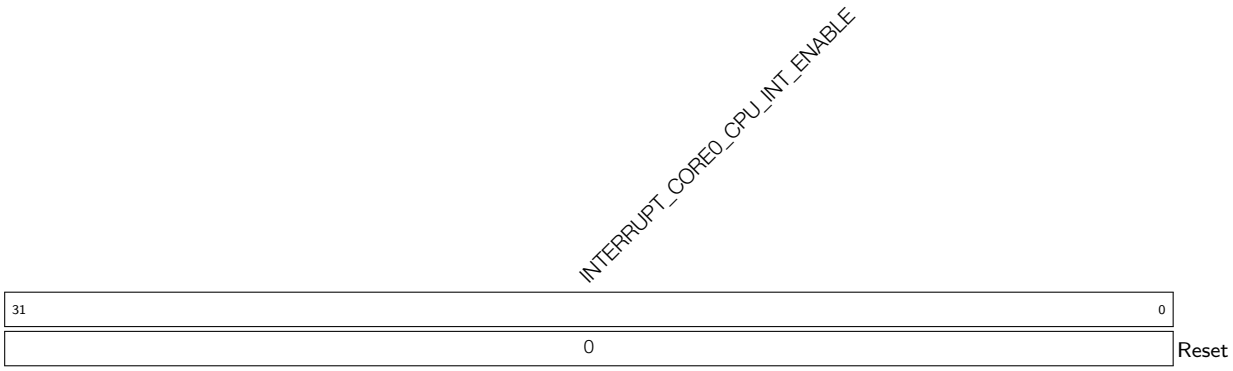
INTERRUPT_CORE0_INTR_STATUS_1 This register stores the status of interrupt sources: 32 ~ 42. If the bit is 1 here, it means the corresponding source triggered an interrupt. (RO)

Register 8.27. INTERRUPT_CORE0_CLOCK_GATE_REG (0x00B4)



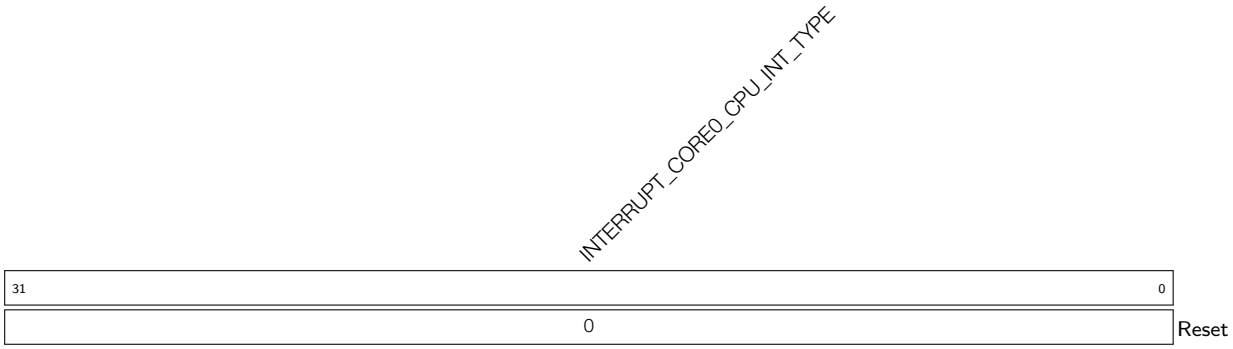
INTERRUPT_CORE0_CLK_EN Set 1 to force interrupt register clock-gate on. (R/W)

Register 8.28. INTERRUPT_CORE0_CPU_INT_ENABLE_REG (0x00B8)



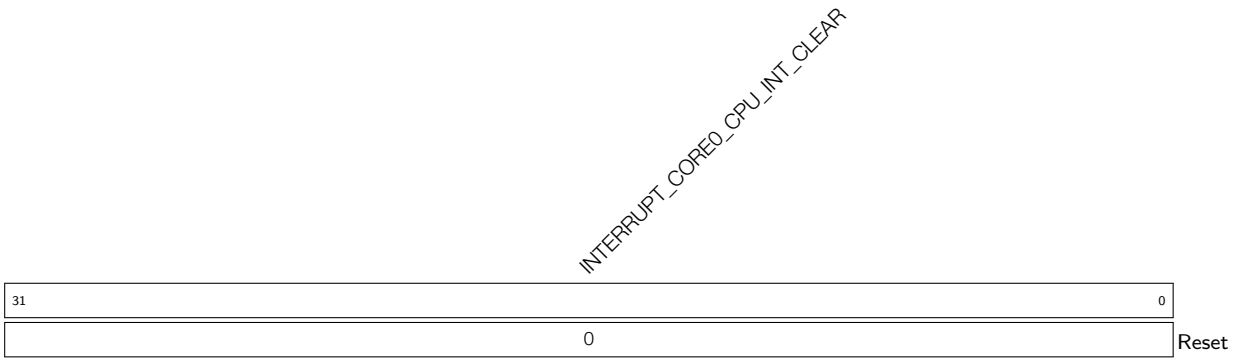
INTERRUPT_CORE0_CPU_INT_ENABLE Writing 1 to the bit here enables its corresponding CPU interrupt. For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU*. (R/W)

Register 8.29. INTERRUPT_CORE0_CPU_INT_TYPE_REG (0x00BC)



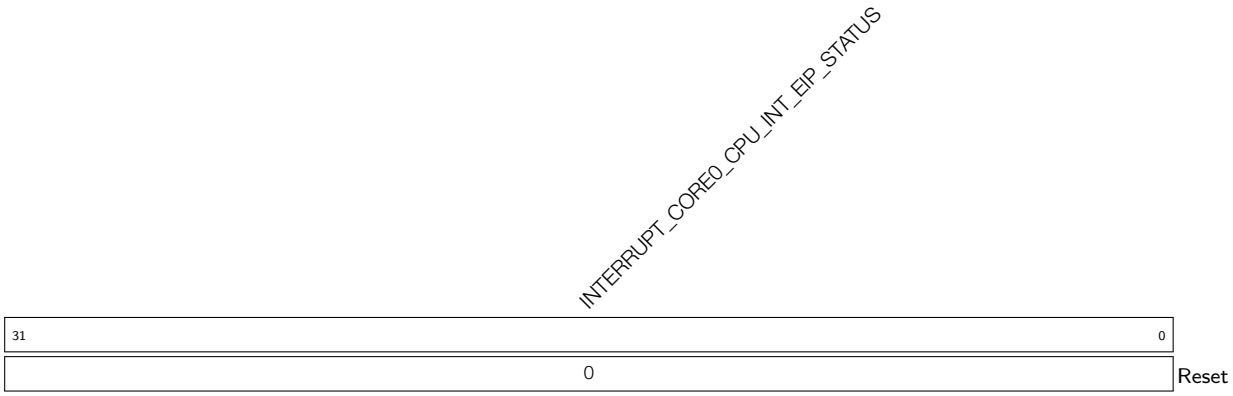
INTERRUPT_CORE0_CPU_INT_TYPE Configure CPU interrupt type. 0: level-triggered; 1: edge-triggered. For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU*. (R/W)

Register 8.30. INTERRUPT_CORE0_CPU_INT_CLEAR_REG (0x00C0)



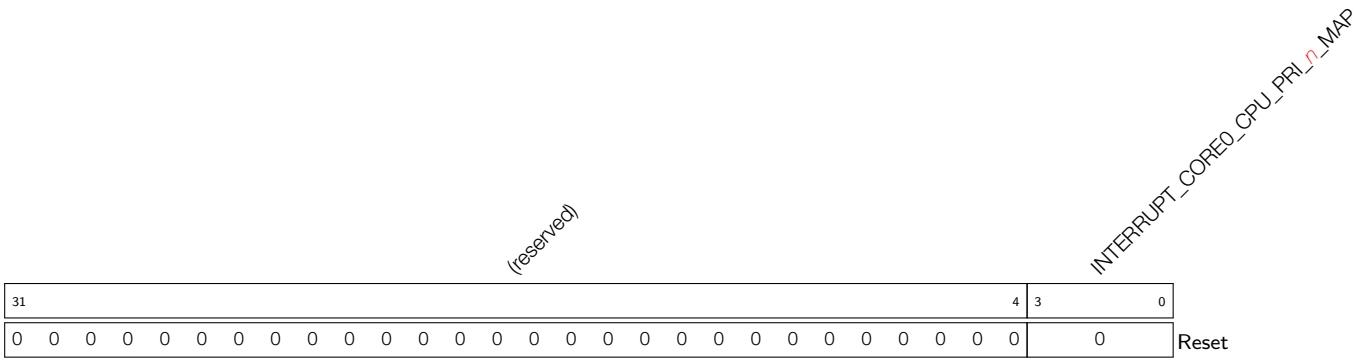
INTERRUPT_CORE0_CPU_INT_CLEAR Writing 1 to the bit here clears its corresponding CPU interrupt. For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU*. (R/W)

Register 8.31. INTERRUPT_CORE0_CPU_INT_EIP_STATUS_REG (0x00C4)



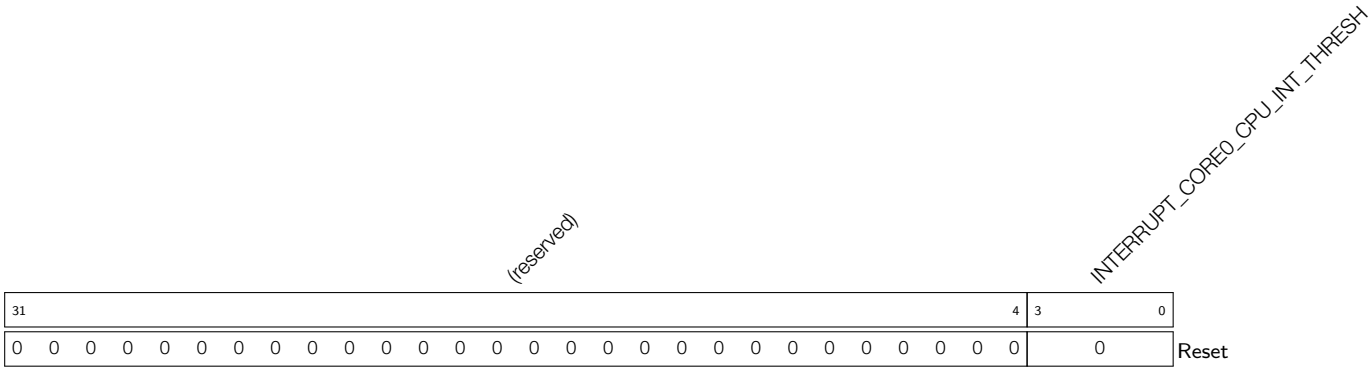
INTERRUPT_CORE0_CPU_INT_EIP_STATUS Store the pending status of CPU interrupts. For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU*. (RO)

Register 8.32. INTERRUPT_CORE0_CPU_INT_PRI_*n*_REG (*n*: 1 - 31)(0x00C8 + 0x4n*)**



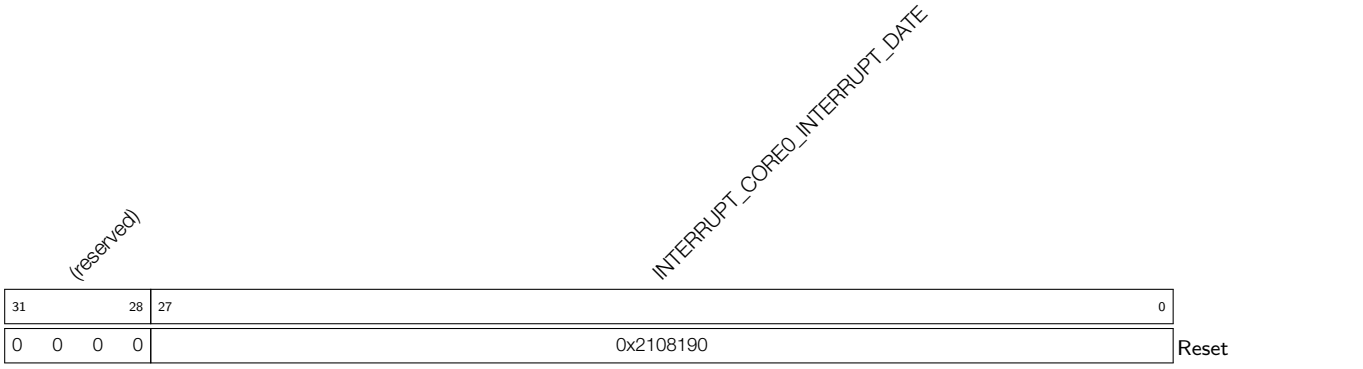
INTERRUPT_CORE0_CPU_PRI_*n*_MAP Set the priority for CPU interrupt *n*. The priority here can be 1 (lowest) ~ 15 (highest). For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU*. (R/W)

Register 8.33. INTERRUPT_CORE0_CPU_INT_THRESH_REG (0x0148)



INTERRUPT_CORE0_CPU_INT_THRESH Set threshold for interrupt assertion to CPU. Only when the interrupt priority is equal to or higher than this threshold, CPU will respond to this interrupt. For more information about how to use this register, see Chapter 1 *ESP-RISC-V CPU*. (R/W)

Register 8.34. INTERRUPT_CORE0_INTERRUPT_DATE_REG (0x07FC)



INTERRUPT_CORE0_INTERRUPT_DATE Version control register. (R/W)

9 Low-power Management (RTC_CNTL)

9.1 Introduction

ESP8684 has an advanced Power Management Unit (PMU), which can flexibly power up different power domains of the chip, to achieve the best balance among chip performance, power consumption, and wakeup latency. To simplify power management for typical scenarios, ESP8684 has predefined four power modes, which are preset configurations that power up different combinations of power domains. On top of that, the chip also allows the users to independently power up any particular power domain to meet more complex requirements.

9.2 Features

ESP8684's low-power management supports the following features:

- 4 x predefined power modes to simplify power management for typical scenarios
- 8 x 32-bit retention registers

In this chapter, we first introduce the working process of ESP8684's low-power management, then introduce the predefined power modes of the chip.

9.3 Functional Description

ESP8684's low-power management involves the following components:

- Power management unit: controls the power supply to three power domain categories:
  - Real Time Controller (RTC)
  - Digital
  - Analog

  For a complete list of 6 power domains grouped in these three power domain categories, see Section 9.5.1.
- Power isolation unit: isolates different power domains, so any powered down power domain does not affect the powered up ones.
- Low-power clocks: provide clocks to power domains working in low-power modes.
- RTC timer: logs the status of the RTC main state machine in dedicated registers.
- 8 x 32-bit "always-on" retention registers: These registers are always powered up and are not affected by any low-power modes, thus can be used for storing data that cannot be lost.
- 6 x "always-on" pins: These pins are always powered up and are not affected by any low-power modes, which makes them suitable for working as wakeup sources when the chip is working in the low-power modes (for details, please refer to Section 9.5.3), or can be used as regular GPIOs (for details, please refer to Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO MUX)*).
- Voltage regulators: regulate the power supply to different power domains.

The schematic diagram of ESP8684's low-power management is shown in Figure 9-1.

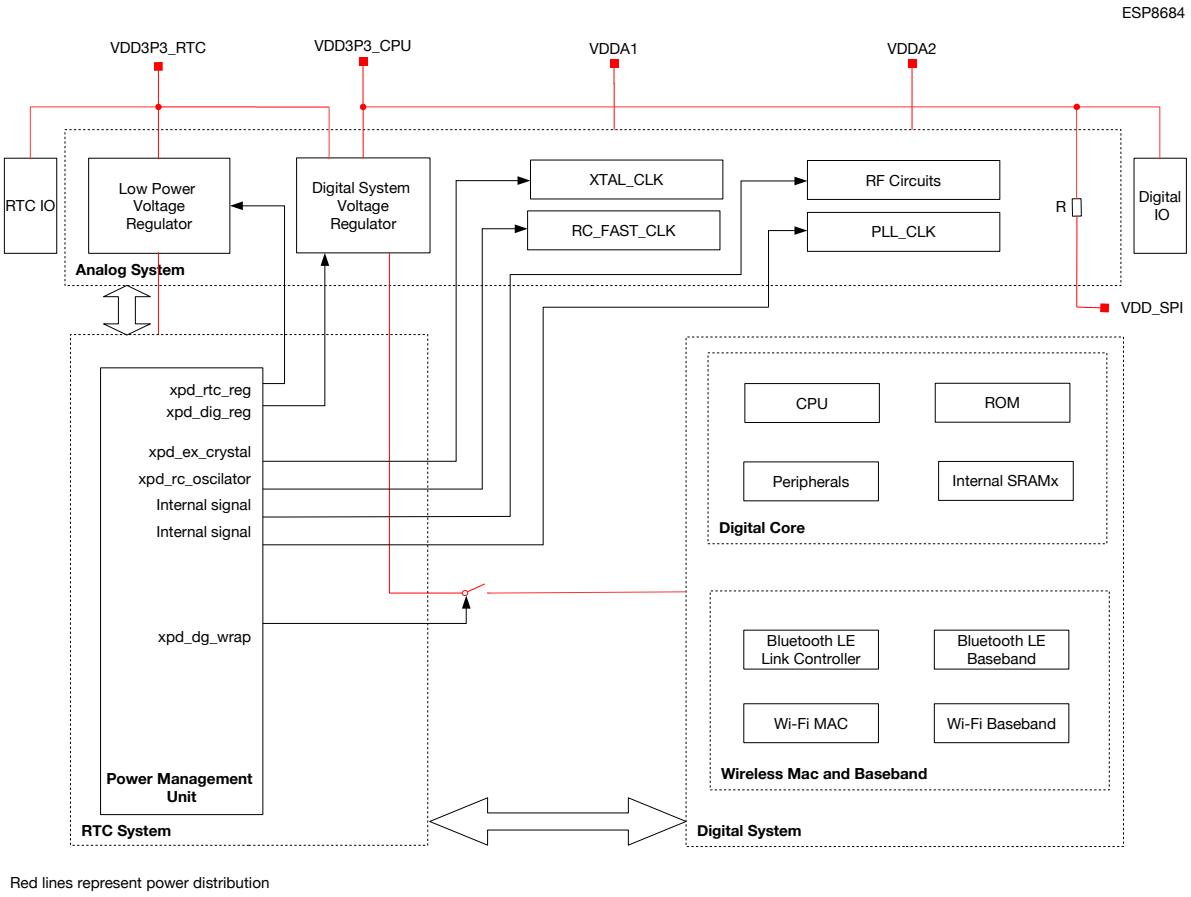


Figure 9-1. Low-power Management Schematics

Note:

- Power domains are enclosed with dashed lines. For more information about different power domains, please check Section 9.5.1.
- Switches in the above diagram can be controlled by Register RTC_CNTL_DIG_PWC_REG.
- Signals in the above diagram are described below:
  - xpd_rtc_reg:
    - When RTC_CNTL_REGULATOR_FORCE_PU is set to 1, low power voltage regulator is always-on;
    - Otherwise, the low power voltage regulator is off when chip enters sleep.
  - xpd_dig_reg:
    - When RTC_CNTL_DG_WRAP_PD_EN is enabled, the digital system voltage regulator is off when the chip enters sleep;
    - Otherwise, the digital system voltage regulator is always-on.
  - xpd_ex_crystal:
    - When RTC_CNTL_XTL_FORCE_PU is set to 1, the external main crystal clock is always-on;
    - Otherwise, the external main crystal clock is off when chip enters sleep.
  - xpd_rc_oscilator:
    - when RTC_CNTL_FOSC_FORCE_PU is set to 1, the fast RC oscillator is always-on;
    - Otherwise, the fast RC oscillator is off when chip enters sleep.

9.3.1 Power Management Unit (PMU)

ESP8684's power management unit controls the power supply to different power domains. The main components of the power management unit include:

- RTC main state machine: generates power gating, clock gating, and reset signals.
- Power controllers: power up and power down different power domains, according to the power gating signals from the main state machine.
- Sleep / wakeup controllers: send sleep or wakeup requests to the RTC main state machine.
- Clock controller: selects and powers up/down clock sources.
- Protection Timer: controls the transition interval between main state machine states.

In ESP8684's power management unit, the sleep / wakeup controllers send sleep or wakeup requests to the RTC main state machine, which then generates power gating, clock gating, and reset signals. Then, the power controller and clock controller power up and power down different power domains and clock sources, according to the signals generated by the RTC main state machine, so that the chip enters or exits the low-power modes. The main workflow is shown in Figure 9-2.

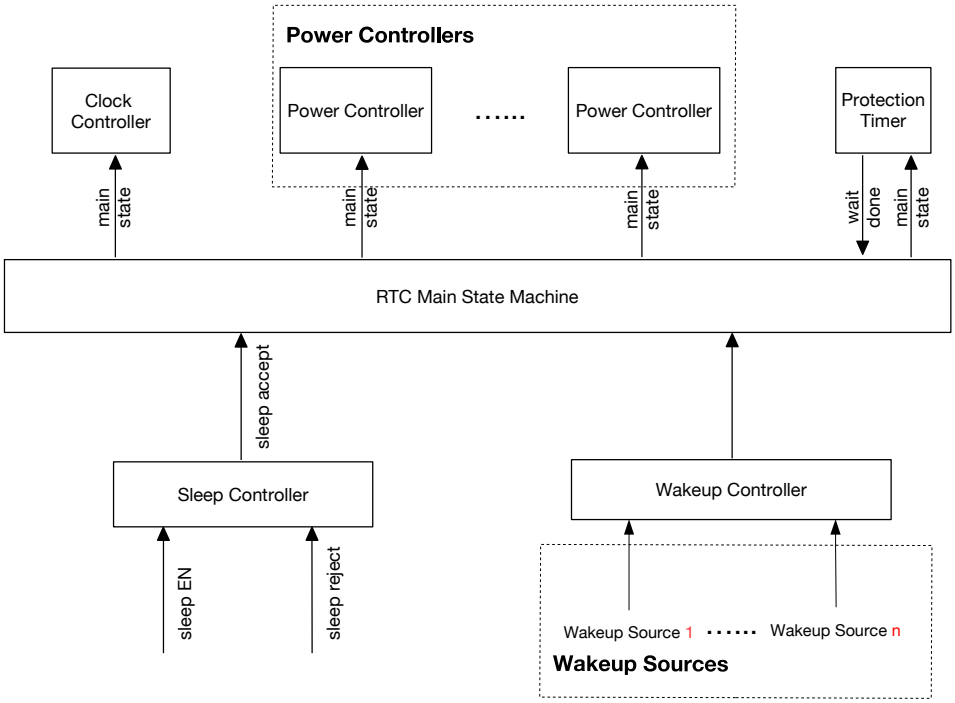


Figure 9-2. Power Management Unit Workflow

Note:

1. Each power domain has its own power controller. For a complete list of all the available power controllers controlling different power domains, please refer to Section 9.5.1.
2. For a complete list of all the available wakeup sources, please refer to Table 9-4.

9.3.2 Low-Power Clocks

In general, ESP8684 powers down its External Main Crystal Clock (XTAL_CLK) and PLL Clock (PLL_CLK) to reduce power consumption when working in low-power modes. During this time, the chip's low-power clocks remain on to provide clocks to low power domains, such as the power management unit.

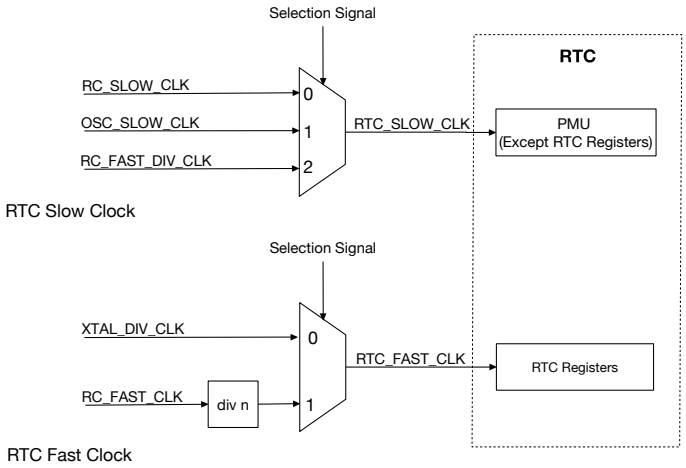


Figure 9-3. RTC_SLOW_CLK and RTC_FAST_CLK

Table 9-1. Low-power Clocks

Clock Type	Clock Source	Selection Signal	Power Domain
RTC_SLOW_CLK	OSC_SLOW_CLK	RTC_CNTL_ANA_CLK_RTC_SEL	Power Management System (except RTC registers)
	RC_FAST_DIV_CLK		
	RC_SLOW_CLK (default)		
RTC_FAST_CLK	RC_FAST_CLK divided by n (default)	RTC_CNTL_FAST_CLK_RTC_SEL	RTC Registers
	XTAL_DIV_CLK		

For more detailed description about clocks, please refer to 6 *Reset and Clock*.

9.3.3 Timers

ESP8684's low-power management uses RTC timer. The readable 48-bit RTC timer is a real-time counter (using RTC slow clock) that can be configured to log the time when one of the following events happens. For details, see Table 9-2.

Table 9-2. The Triggering Conditions for the RTC Timer

Enabling Options	Descriptions
RTC_CNTL_TIMER_XTL_OFF	RTC main state machine powers down or XTAL_CLK powers up.
RTC_CNTL_TIMER_SYS_STALL	CPU enters or exits the stall state. This is to ensure the SYS_TIMER is continuous in time.
RTC_CNTL_TIMER_SYS_RST	Resetting digital system completes.

RTC_CNTL_TIME_UPDATE	Register RTC_CNTL_TIME_UPDATE is configured by CPU (i.e. users).

The RTC timer updates two groups of registers upon any new trigger. The first group logs the time of the current trigger, and the other logs the previous trigger. Detailed information about these two register groups is shown below:

- Register group 0: logs the status of RTC timer at the current trigger.
  - RTC_CNTL_TIME_HIGH0_REG
  - RTC_CNTL_TIME_LOW0_REG
- Register group 1: logs the status of RTC timer at the previous trigger.
  - RTC_CNTL_TIME_HIGH1_REG
  - RTC_CNTL_TIME_LOW1_REG

On a new trigger, information on previous trigger is moved from register group 0 to register group 1 (and the original trigger logged in register group 1 is overwritten), and this new trigger is logged in register group 0. Therefore, only the last two triggers can be logged at any time.

It should be noted that any reset / sleep other than power-up reset will not stop or reset the RTC timer.

Also, the RTC timer can be used as a wakeup source. For details, see Section 9.5.3.

9.3.4 Voltage Regulators

ESP8684 has two regulators to maintain a constant power supply voltage to different power domains:

- Digital system voltage regulator for digital power domains;
- Low-power voltage regulator for RTC power domains.

Note:
For more detailed description about power domains, please refer to Section 9.5.1.

9.3.4.1 Digital System Voltage Regulator

ESP8684's built-in digital system voltage regulator converts the external power supply (typically 3.3 V) to 1.1 V for digital power domains. This regulator is controlled by the xpd_dig_reg signal. For details, see description of Figure 9-1. For the architecture of the ESP8684 digital system voltage regulator, see Figure 9-4.

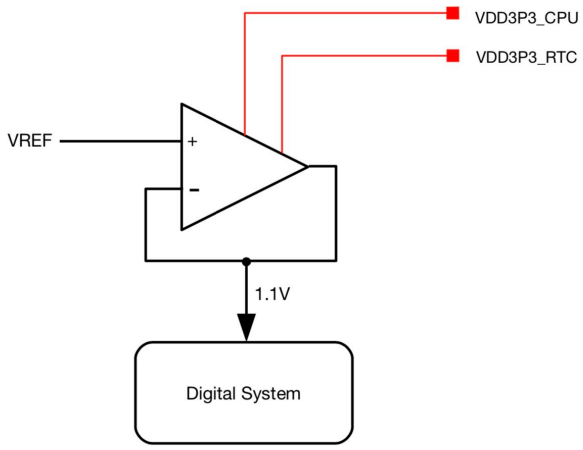


Figure 9-4. Digital System Regulator

9.3.4.2 Low-power Voltage Regulator

ESP8684's built-in low-power voltage regulator converts the external power supply (typically 3.3 V) to 1.1 V for RTC power domains. Note when the pin CHIP_EN is at a high level, the low-power voltage regulator cannot be turned off, but only switching between normal-work mode and Deep-sleep mode.

For the architecture of the ESP8684 low-power voltage regulator, see Figure 9-5.

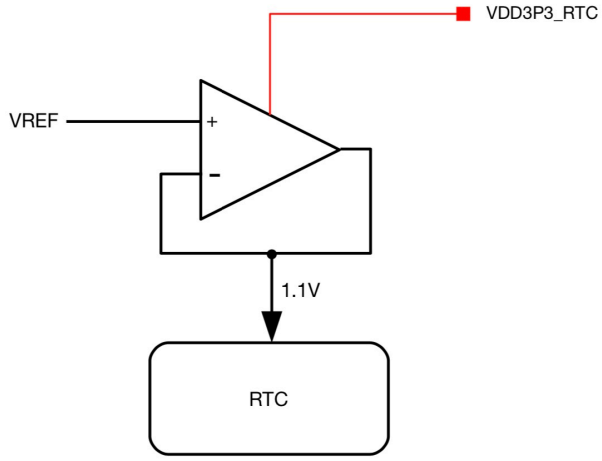


Figure 9-5. Low-power voltage regulator

9.4 Brownout Detector

The brownout detector checks the voltage of pins VDDA, VDDA3P3, VDD3P3_RTC and VDD3P3_CPU. If the voltage of these pins drops below the predefined threshold (2.7 V by default), the detector would trigger a signal to shut down some power-consuming blocks (such as LNA, PA, etc.) to allow extra time for the digital system to save and transfer important data.

The brownout detector has ultra-low power consumption and remains enabled whenever the chip is powered up.

For the architecture of the ESP8684 brownout detector, see Figure 9-6.

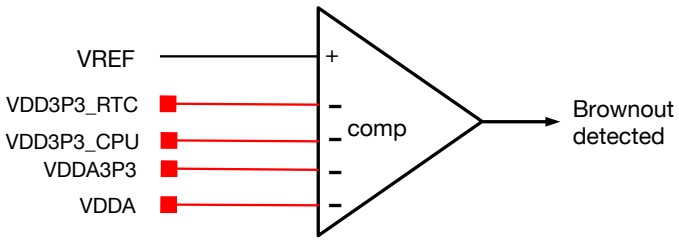


Figure 9-6. Brownout detector

RTC_CNTL_BROWN_OUT_DET indicates the output level of brownout detector. This register is low level by default, and outputs high level when the voltage on any of monitored pins drops below the predefined threshold.

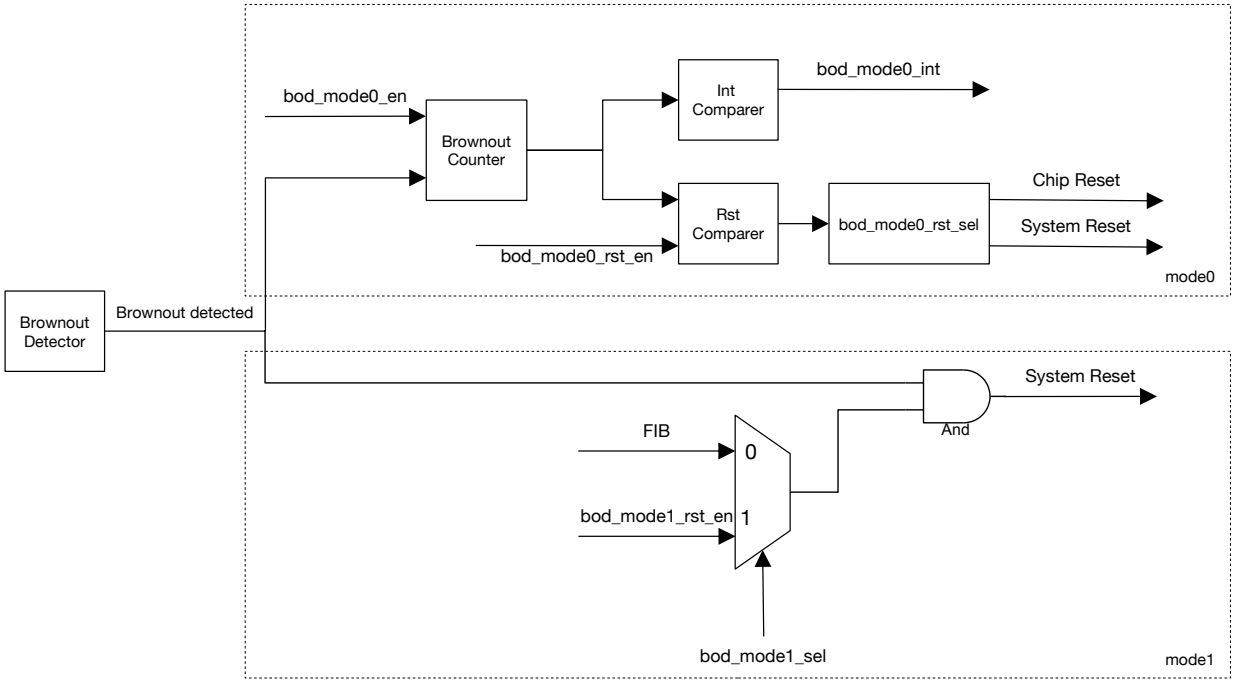


Figure 9-7. Brownout handling

As shown in the Figure 9-7, the brownout detector can handle the detected brownout signal in one of these two methods described below according to user configuration:

- mode0: triggers an interrupt when the brownout counter counts to the threshold pre-defined in the interrupt comparator (configured with RTC_CNTL_BROWN_OUT_INT_WAIT)
  - Additionally, when bod_mode0_rst_en (RTC_CNTL_BROWN_OUT_RST_ENA) is enabled, the brownout detector also resets the chip when the brownout counter counts to the threshold pre-defined in the reset comparator (configured with RTC_CNTL_BROWN_OUT_RST_WAIT) based on the rst_sel (configured with RTC_CNTL_BROWN_OUT_RST_SEL):

    * 0: resets the chip
    * 1: resets the system

  For more information regarding chip reset and system reset, please refer to Chapter 6 *Reset and Clock*.

- mode1: resets the system directly.

To choose how the brownout detector handles the detected brownout signal:

- mode0: set the bod_mode0_en signal (configured with RTC_CNTL_BROWN_OUT_ENA).
- mode1: bod_mode1_sel
  - 0: set the bod_mode1_rst_en signal (configured with RTC_CNTL_BROWN_OUT_ANA_RST_EN)
  - 1: FIB bus
- Note that mode1 prevails mode0 when both options are enabled at the same time

9.5 Power Modes Management

9.5.1 Power Domains

ESP8684 has 6 power domains in three power domain categories:

- RTC
  - Power management unit (PMU), including RTC timer and always-on registers
- Digital
  - Digital including digital core and Wireless digital circuit
- Analog
  - RC_FAST_CLK
  - XTAL_CLK
  - PLL_CLK
  - RF Circuits

9.5.2 Pre-defined Power Modes

As mentioned earlier, ESP8684 has four power modes, which are predefined configurations that power up different combinations of power domains. For details, please refer to Table 9-3.

Table 9-3. Predefined Power Modes

Power Mode	Power Domain					
	PMU	Digital	RC_FAST_CLK	XTAL_CLK	PLL_CLK	RF Circuits
Active	ON	ON	ON	ON	ON	ON
Modem-sleep	ON	ON	ON	ON	ON	OFF
Light-sleep	ON	ON	OFF	OFF	OFF	OFF
Deep-sleep	ON	OFF	OFF	OFF	OFF	OFF

By default, ESP8684 first enters the Active mode after system resets, then enters different low-power modes (including Modem-sleep, Light-sleep, and Deep-sleep) to save power after the CPU stalls for a specific time (For example, when CPU is waiting to be wakened up by an external event). From modes Active to Deep-sleep, the number of available functionalities[1] and power consumption[2] decreases and wakeup latency increases. Also, the supported wakeup sources for different power modes are different[3]. Users can choose a power mode based on their requirements of functionality, power consumption, wakeup latency, and available wakeup sources.

Note:

1. For details, please refer to Table 9-3.
2. For details on power consumption, please refer to the Current Consumption Characteristics in *ESP8684 Datasheet*.
3. For details on the supported wakeup sources, please refer to Section 9.5.3.

9.5.3 Wakeup Sources

The ESP8684 supports various wakeup sources, which could wake up the CPU in different sleep modes. The wakeup source is determined by RTC_CNTL_WAKEUP_ENA as shown in Table 9-4.

Table 9-4. Wakeup Source

WAKEUP_ENA	Wakeup Source	Light-sleep	Deep-sleep
0x4	GPIO[1]	Y	Y
0x8	RTC Timer	Y	Y
0x20	Wi-Fi[2]	Y	-
0x40	UART0[3]	Y	-
0x80	UART1[3]	Y	-
0x400	Bluetooth	Y	-

[1] In Deep-sleep mode, only the RTC GPIOs (not regular GPIOs) can work as a wakeup source.

[2] To wake up the chip with a Wi-Fi source, the chip switches between the Active, Modem-sleep, and Light-sleep modes. The CPU and RF modules are woken up at predetermined intervals to keep Wi-Fi connections active.

[3] A wakeup is triggered when the number of RX pulses received exceeds the setting in the threshold register UART_SLEEP_CONF_REG. For details, please refer to Chapter 19 *UART Controller (UART)*.

9.5.4 Reject Sleep

ESP8684 implements a hardware mechanism that equips the chip with the ability to reject to sleep, which prevents the chip from going to sleep unexpectedly when some peripherals are still working but not detected by the CPU, thus guaranteeing the proper functioning of the peripherals.

Table 9-5. Reject Source

REJECT_ENA	Reject Source
0x4	GPIO
0x8	RTC Timer
0x20	Wi-Fi
0x400	Bluetooth

Users can configure the reject to sleep option according to Table 9-5 via the following registers.

- Configure the RTC_CNTL_SLEEP_REJECT_ENA field to enable or disable the option to reject to sleep:
  - Set RTC_CNTL_LIGHT_SLP_REJECT_EN to enable reject-to-light-sleep.
  - Set RTC_CNTL_DEEP_SLP_REJECT_EN to enable reject-to-deep-sleep.
- Read RTC_CNTL_SLP_REJECT_CAUSE_REG to check the reason for rejecting to sleep.

9.6 Register Summary

The addresses in this section are relative to low-power management base address provided in Table 3-3 in Chapter 3 *System and Memory*.

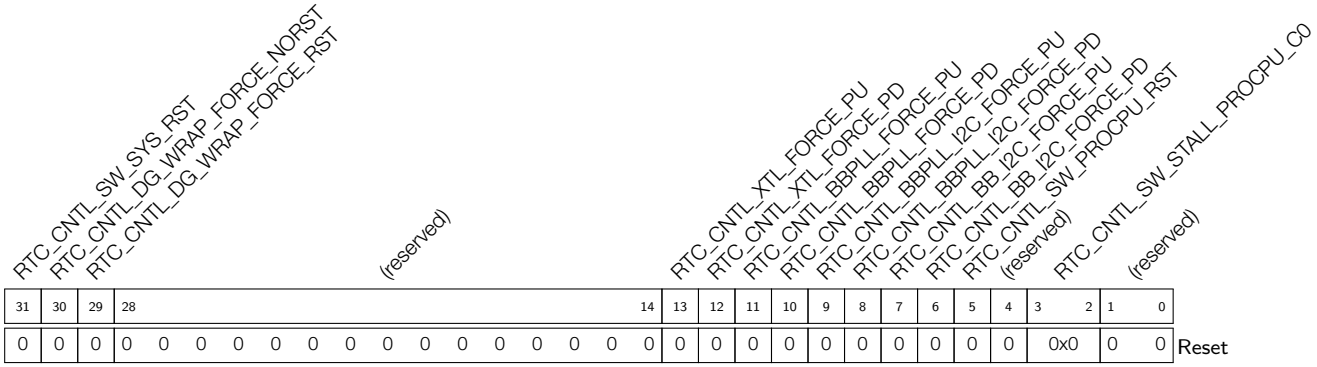
Name	Description	Address	Access
Control / Configuration Registers			
RTC_CNTL_OPTIONS0_REG	Configures the power options of crystal and PLL clocks, and initiates reset by software	0x0000	varies
RTC_CNTL_SLP_TIMER0_REG	RTC timer threshold register 0	0x0004	R/W
RTC_CNTL_SLP_TIMER1_REG	RTC timer threshold register 1	0x0008	R/W
RTC_CNTL_TIME_UPDATE_REG	RTC timer update control register	0x000C	R/W
RTC_CNTL_TIME_LOW0_REG	Represents the lower 32 bits of RTC timer 0	0x0010	R/W
RTC_CNTL_TIME_HIGH0_REG	Represents the higher 16 bits of RTC timer 0	0x0014	R/W
RTC_CNTL_STATE0_REG	Configures the sleep / reject / wakeup state	0x0018	R/W
RTC_CNTL_TIMER1_REG	Configures CPU stall options	0x001C	R/W
RTC_CNTL_TIMER2_REG	Configures RTC_SLOW_CLK and touch controller	0x0020	R/W
RTC_CNTL_ANA_CONF_REG	Configures the power options for I2C and PLLA	0x002C	R/W
RTC_CNTL_WAKEUP_STATE_REG	Wakeup bitmap enabling register	0x0034	R/W
RTC_CNTL_STORE0_REG	Reservation register 0	0x0048	R/W
RTC_CNTL_STORE1_REG	Reservation register 1	0x004C	R/W
RTC_CNTL_STORE2_REG	Reservation register 2	0x0050	R/W
RTC_CNTL_STORE3_REG	Reservation register 3	0x0054	R/W
RTC_CNTL_EXT_WAKEUP_CONF_REG	GPIO wakeup configuration register	0x005C	R/W
RTC_CNTL_SLP_REJECT_CONF_REG	Configures sleep / reject options	0x0060	R/W
RTC_CNTL_CLK_CONF_REG	RTC timer configuration register	0x0068	R/W
RTC_CNTL_REG	RTC configuration register	0x0074	R/W
RTC_CNTL_PWC_REG	RTC power configuration register	0x0078	R/W
RTC_CNTL_DIG_PWC_REG	Digital system power configuration register	0x007C	R/W
RTC_CNTL_DIG_ISO_REG	Digital system isolation configuration register	0x0080	R/W
RTC_CNTL_WDTCONFIG0_REG	RTC watchdog configuration register	0x0084	R/W
RTC_CNTL_WDTCONFIG1_REG	Configures the hold time of RTC watchdog in stage 0	0x0088	R/W
RTC_CNTL_WDTCONFIG2_REG	Configures the hold time of RTC watchdog in stage 1	0x008C	R/W
RTC_CNTL_WDTCONFIG3_REG	Configures the hold time of RTC watchdog in stage 2	0x0090	R/W
RTC_CNTL_WDTCONFIG4_REG	Configures the hold time of RTC watchdog in stage 3	0x0094	R/W

Name	Description	Address	Access
RTC_CNTL_WDTFEED_REG	RTC watchdog SW feed configuration register	0x0098	R/W
RTC_CNTL_WDTWPROTECT_REG	RTC watchdog write protection configuration register	0x009C	R/W
RTC_CNTL_SWD_CONF_REG	Super watchdog configuration register	0x00A0	R/W
RTC_CNTL_SWD_WPROTECT_REG	Super watchdog write protection configuration register	0x00A4	R/W
RTC_CNTL_SW_CPU_STALL_REG	CPU stall configuration register	0x00A8	R/W
RTC_CNTL_STORE4_REG	Reservation register 4	0x00AC	R/W
RTC_CNTL_STORE5_REG	Reservation register 5	0x00B0	R/W
RTC_CNTL_STORE6_REG	Reservation register 6	0x00B4	R/W
RTC_CNTL_STORE7_REG	Reservation register 7	0x00B8	R/W
RTC_CNTL_PAD_HOLD_REG	Configures the hold options for RTC GPIOs	0x00C4	R/W
RTC_CNTL_DIG_PAD_HOLD_REG	Configures the hold options for digital GPIOs	0x00C8	R/W
RTC_CNTL_BROWN_OUT_REG	Brownout configuration register	0x00CC	R/W
RTC_CNTL_TIME_LOW1_REG	Represents the lower 32 bits of RTC timer 1	0x00D0	R/W
RTC_CNTL_TIME_HIGH1_REG	Represents the higher 16 bits of RTC timer 1	0x00D4	R/W
RTC_CNTL_USB_CONF_REG	IO_MUX configuration register	0x00D8	R/W
RTC_CNTL_SLP_REJECT_CAUSE_REG	Represents the reject-to-sleep cause	0x00DC	R/W
RTC_CNTL_OPTION1_REG	RTC option register	0x00E0	R/W
RTC_CNTL_SLP_WAKEUP_CAUSE_REG	Represents the sleep-to-wakeup cause.	0x00E4	R/W
RTC_CNTL_CNTL_GPIO_WAKEUP_REG	GPIO wakeup configuration register	0x00FC	R/W
RTC_CNTL_CNTL_SENSOR_CTRL_REG	SAR ADC control register	0x0108	R/W
RTC_CNTL_FIB_SEL_REG	Brownout detector configuration register	0x00F8	R/W
Status Registers			
RTC_CNTL_RESET_STATE_REG	Represents the CPU reset source	0x0030	R/W
RTC_CNTL_LOW_POWER_ST_REG	Represents the RTC state	0x00BC	R/W
Interrupt Registers			
RTC_CNTL_INT_ENA_RTC_REG	RTC interrupt enabling register	0x0038	R/W
RTC_CNTL_INT_RAW_RTC_REG	RTC interrupt raw register	0x003C	R/W
RTC_CNTL_INT_ST_RTC_REG	RTC interrupt state register	0x0040	R/W
RTC_CNTL_INT_CLR_RTC_REG	RTC interrupt clear register	0x0044	R/W
RTC_CNTL_INT_ENA_RTC_W1TS_REG	RTC interrupt enabling register (W1TS)	0x00EC	R/W
RTC_CNTL_INT_ENA_RTC_W1TC_REG	RTC interrupt clear register (W1TC)	0x00F0	R/W

9.7 Registers

The addresses in this section are relative to low-power management base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Register 9.1. RTC_CNTL_OPTIONS0_REG (0x0000)



RTC_CNTL_SW_STALL_PROCPU_C0 Write 0x2 to stall the CPU by SW. Valid only when RTC_CNTL_SW_STALL_PROCPU_C1 is configured to 0x21. (R/W)

RTC_CNTL_SW_PROCPU_RST Write 1 to reset the CPU by SW. (WO)

RTC_CNTL_BB_I2C_FORCE_PD Write 1 to FPD BB_I2C. (R/W)

RTC_CNTL_BB_I2C_FORCE_PU Write 1 to FPU BB_I2C. (R/W)

RTC_CNTL_BBPLL_I2C_FORCE_PD Write 1 to FPD BB_PLL_I2C. (R/W)

RTC_CNTL_BBPLL_I2C_FORCE_PU Write 1 to FPU BB_PLL_I2C. (R/W)

RTC_CNTL_BBPLL_FORCE_PD Write 1 to FPD BB_PLL. (R/W)

RTC_CNTL_BBPLL_FORCE_PU Write 1 to FPU BB_PLL. (R/W)

RTC_CNTL_XTL_FORCE_PD Write 1 to FPD the XTAL_CLK. (R/W)

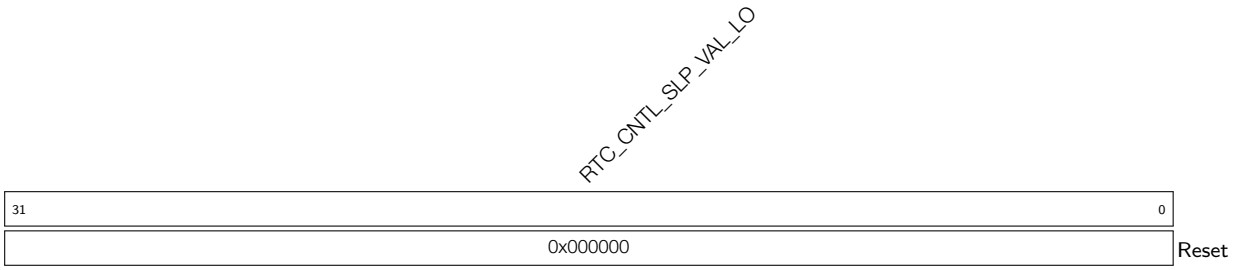
RTC_CNTL_XTL_FORCE_PU Write 1 to FPU the XTAL_CLK. (R/W)

RTC_CNTL_DG_WRAP_FORCE_RST Write 1 to force reset the digital system in deep-sleep. (R/W)

RTC_CNTL_DG_WRAP_FORCE_NORST Write 1 to disable force reset to the digital system in deep-sleep. (R/W)

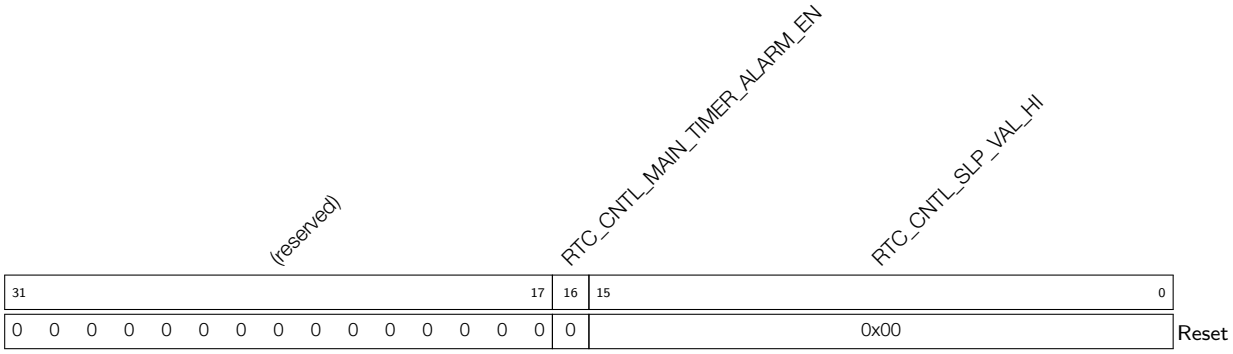
RTC_CNTL_SW_SYS_RST Write 1 to reset the digital power category via SW. (WO)

Register 9.2. RTC_CNTL_SLP_TIMER0_REG (0x0004)



RTC_CNTL_SLP_VAL_LO Configures the lower 32 bits of the trigger threshold for the RTC timer. (R/W)

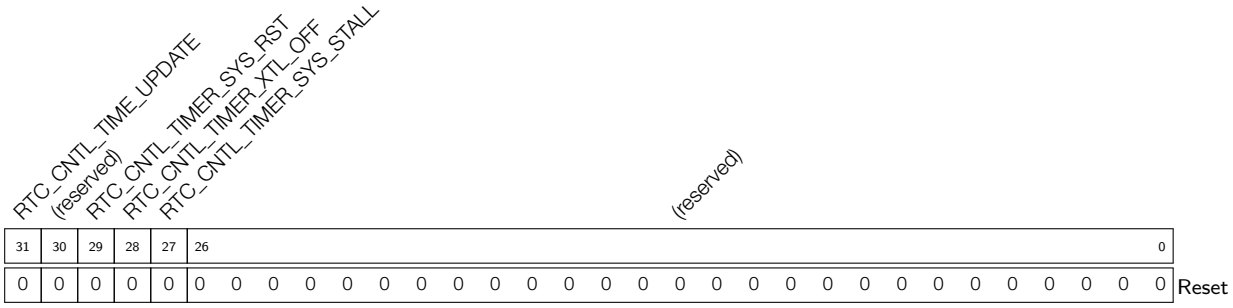
Register 9.3. RTC_CNTL_SLP_TIMER1_REG (0x0008)



RTC_CNTL_SLP_VAL_HI Configures the higher 16 bits of the trigger threshold for the RTC timer. (R/W)

RTC_CNTL_MAIN_TIMER_ALARM_EN Write 1 to enable the timer alarm. (R/W)

Register 9.4. RTC_CNTL_TIME_UPDATE_REG (0x000C)



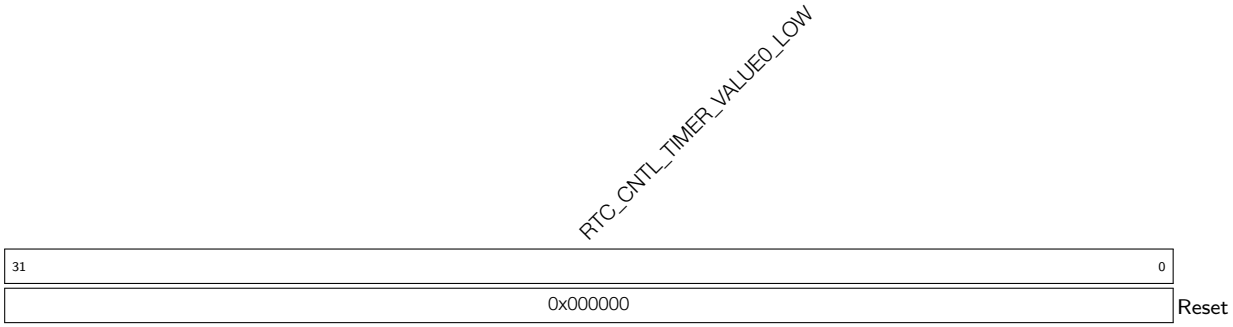
RTC_CNTL_TIMER_SYS_STALL Write 1 to enable recording digital system stall time. (R/W)

RTC_CNTL_TIMER_XTL_OFF Write 1 to enable recording XTAL_CLK OFF time. (R/W)

RTC_CNTL_TIMER_SYS_RST Write 1 to enable recording digital system reset time. (R/W)

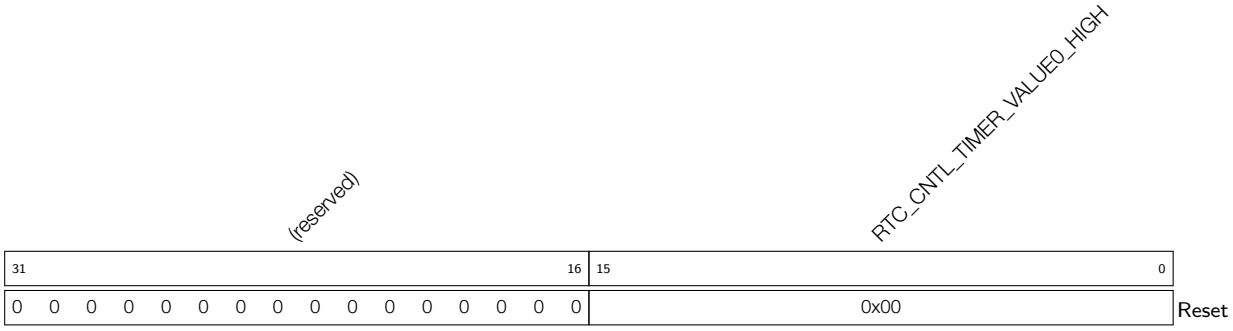
RTC_CNTL_TIME_UPDATE Write 1 to update register with RTC timer. (R/W)

Register 9.5. RTC_CNTL_TIME_LOW0_REG (0x0010)



RTC_CNTL_TIMER_VALUE0_LOW Represents the lower 32 bits of RTC timer 0. (R/W)

Register 9.6. RTC_CNTL_TIME_HIGH0_REG (0x0014)



RTC_CNTL_TIMER_VALUE0_HIGH Represents the higher 16 bits of RTC timer 0. (R/W)

Register 9.7. RTC_CNTL_STATE0_REG (0x0018)

RTC_CNTL_SLEEP_EN	RTC_CNTL_SLP_REJECT	RTC_CNTL_SLP_WAKEUP	RTC_CNTL_SDIO_ACTIVE_IND	(reserved)	RTC_CNTL_APB2RTC_BRIDGE_SEL	(reserved)	RTC_CNTL_SLP_REJECT_CAUSE_CLR	RTC_CNTL_SW_CPU_INT	
31	30	29	28	27 … 23	22	21 … 2	1	0	
0	0	0	0	0 0 0 0 0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	Reset

RTC_CNTL_SW_CPU_INT Write 1 to send a SW RTC interrupt to CPU. (R/W)

RTC_CNTL_SLP_REJECT_CAUSE_CLR Write 1 to clear the RTC reject-to-sleep cause. (R/W)

RTC_CNTL_APB2RTC_BRIDGE_SEL Configures the APB to RTC option.

0x0: APB to RTC using sync

0x1: APB to RTC using bridge

(R/W)

RTC_CNTL_SDIO_ACTIVE_IND Represents the SDIO is active. (R/W)

RTC_CNTL_SLP_WAKEUP Represents wakeup events. (R/W)

RTC_CNTL_SLP_REJECT Represents reject-to-sleep events. (R/W)

RTC_CNTL_SLEEP_EN Write 1 to send the chip to sleep. (R/W)

Register 9.8. RTC_CNTL_TIMER1_REG (0x001C)

RTC_CNTL_PLL_BUF_WAIT	RTC_CNTL_XTL_BUF_WAIT	RTC_CNTL_FOSC_WAIT	RTC_CNTL_CPU_STALL_WAIT	RTC_CNTL_CPU_STALL_EN	
31 … 24	23 … 14	13 … 6	5 … 1	0	
40	80	0x10	1	1	Reset

RTC_CNTL_CPU_STALL_EN Write 1 to enable CPU stalling. (R/W)

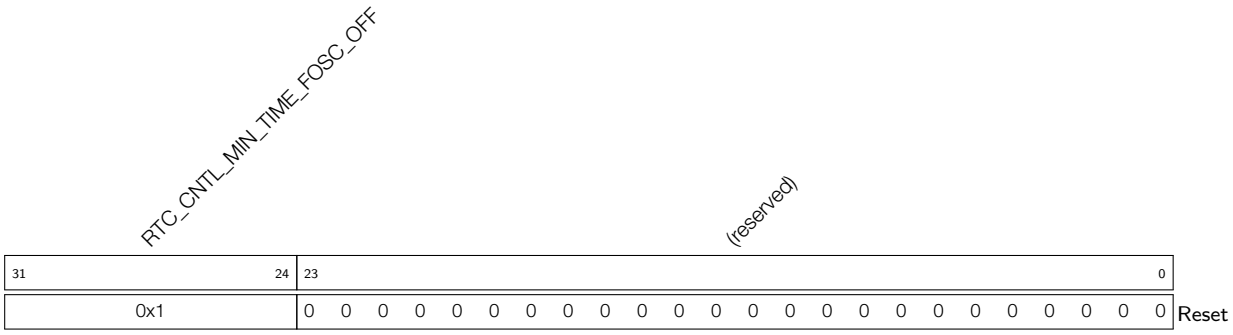
RTC_CNTL_CPU_STALL_WAIT Configures CPU stall wait cycles using RTC_FAST_CLK. (R/W)

RTC_CNTL_FOSC_WAIT Configures the RC_FAST_CLK wait cycles using RTC_SLOW_CLK. (R/W)

RTC_CNTL_XTL_BUF_WAIT Configures XTAL_CLK wait cycles using RTC_SLOW_CLK. (R/W)

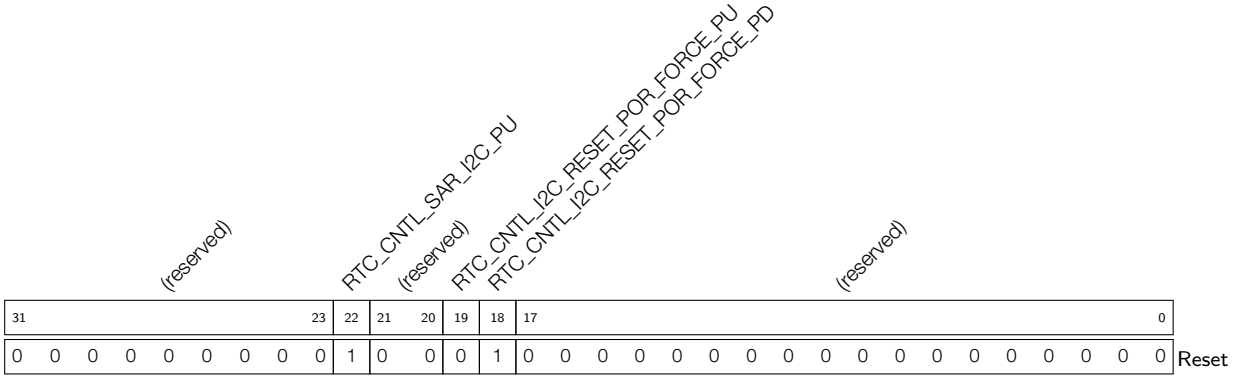
RTC_CNTL_PLL_BUF_WAIT Configures the PLL_CLK wait cycles using RTC_SLOW_CLK. (R/W)

Register 9.9. RTC_CNTL_TIMER2_REG (0x0020)



RTC_CNTL_MIN_TIME_FOSC_OFF Configures the minimal cycle for RC_FAST_CLK (using RTC_SLOW_CLK) when powered down. (R/W)

Register 9.10. RTC_CNTL_ANA_CONF_REG (0x002C)

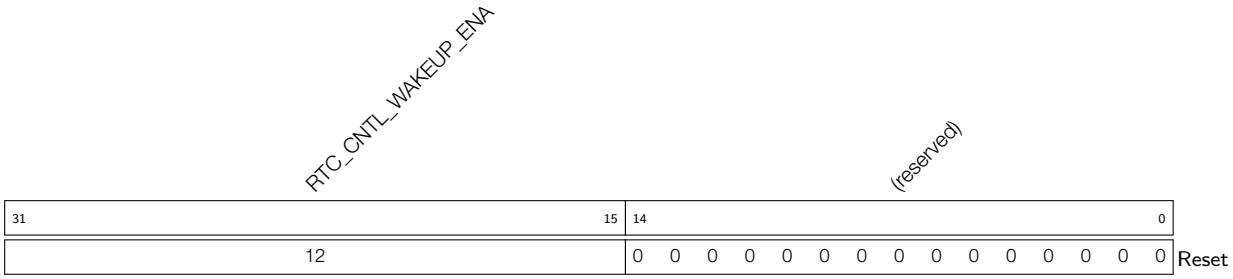


RTC_CNTL_I2C_RESET_POR_FORCE_PD Write 1 to force not bypass I2C power-on reset. (R/W)

RTC_CNTL_I2C_RESET_POR_FORCE_PU Write 1 to force bypass I2C power-on reset. (R/W)

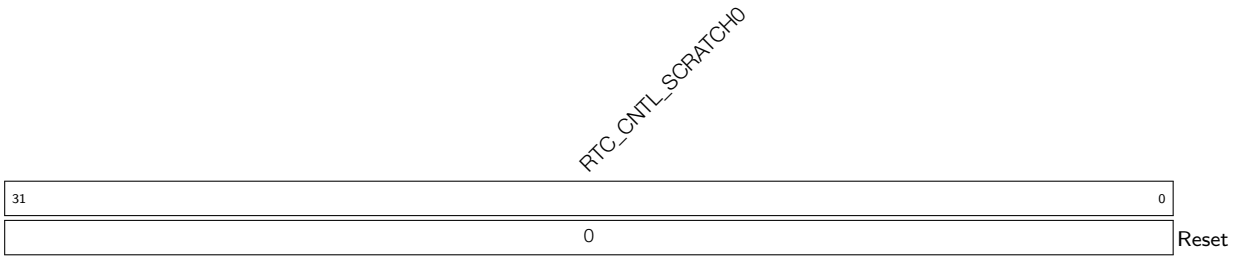
RTC_CNTL_SAR_I2C_PU Write 1 to FPU the SAR_I2C. (R/W)

Register 9.11. RTC_CNTL_WAKEUP_STATE_REG (0x0034)



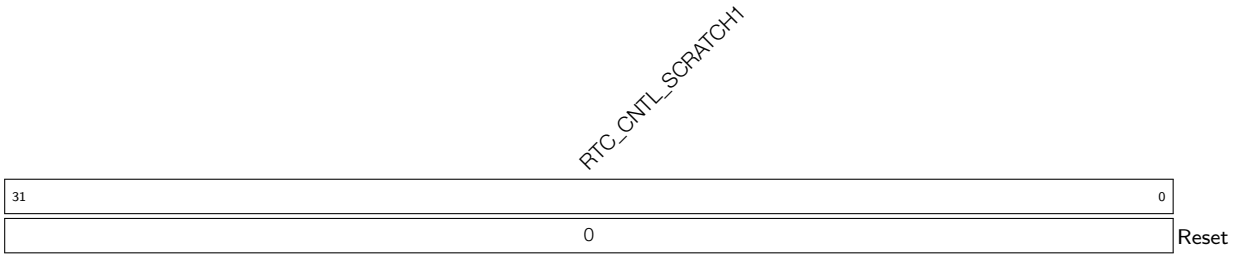
RTC_CNTL_WAKEUP_ENA Configures the wakeup source. For details, please refer to Table 9-4. (R/W)

Register 9.12. RTC_CNTL_STORE0_REG (0x0048)



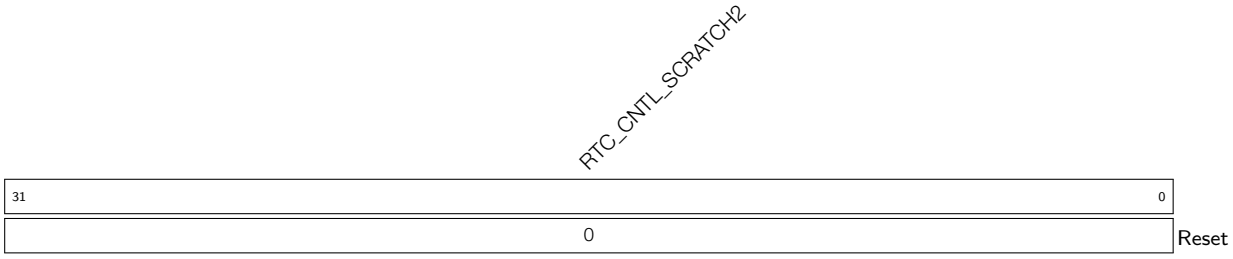
RTC_CNTL_SCRATCH0 Reservation register 0. (R/W)

Register 9.13. RTC_CNTL_STORE1_REG (0x004C)



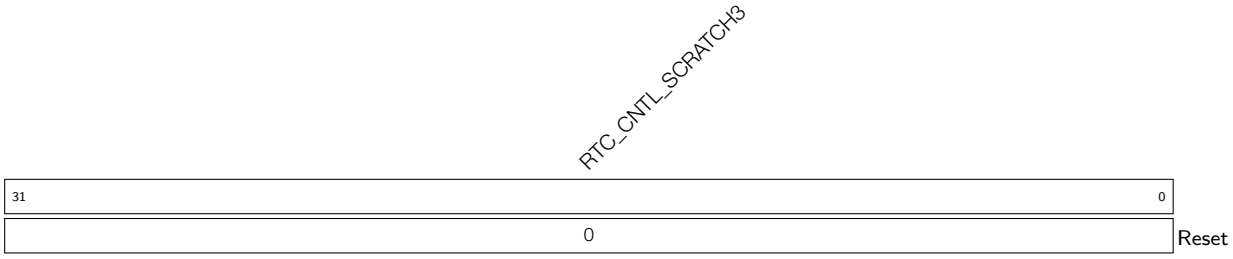
RTC_CNTL_SCRATCH1 Reservation register 1. (R/W)

Register 9.14. RTC_CNTL_STORE2_REG (0x0050)



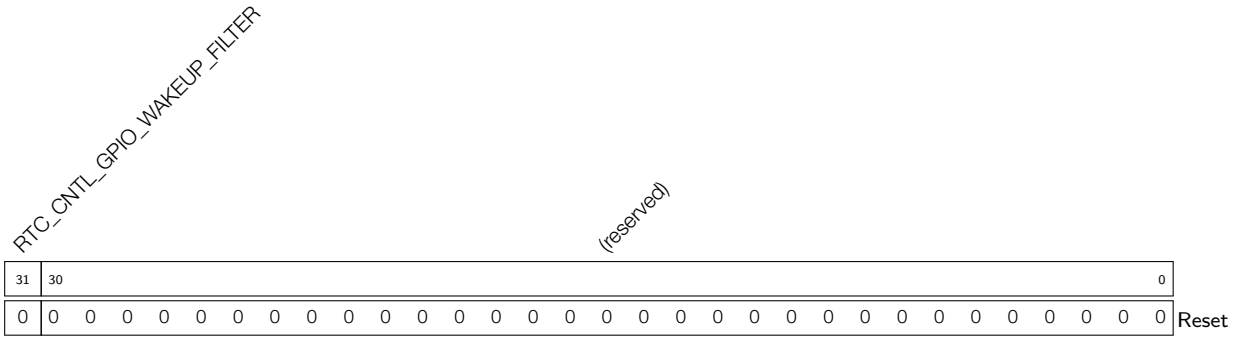
RTC_CNTL_SCRATCH2 Reservation register 2. (R/W)

Register 9.15. RTC_CNTL_STORE3_REG (0x0054)



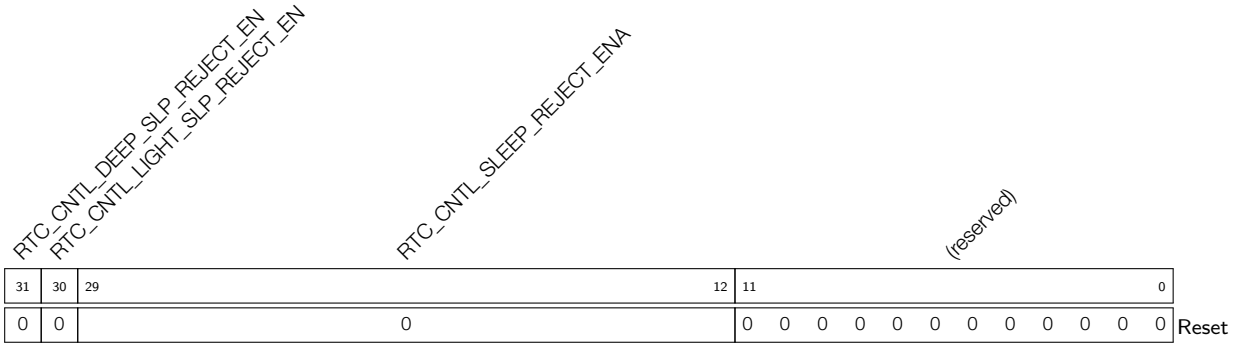
RTC_CNTL_SCRATCH3 Reservation register 3. (R/W)

Register 9.16. RTC_CNTL_EXT_WAKEUP_CONF_REG (0x005C)



RTC_CNTL_GPIO_WAKEUP_FILTER Write 1 to enable the GPIO wakeup event filter. (R/W)

Register 9.17. RTC_CNTL_SLP_REJECT_CONF_REG (0x0060)

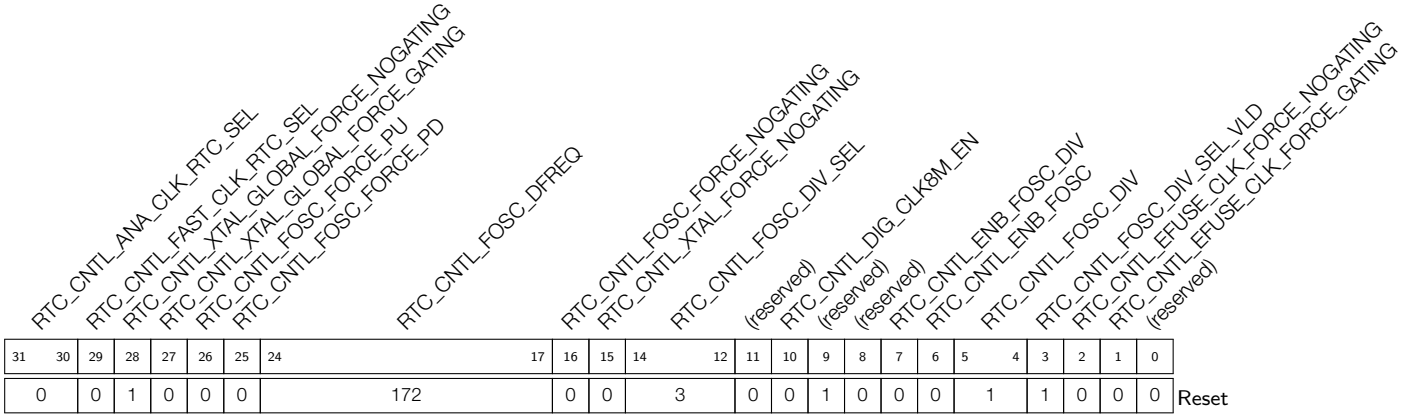


RTC_CNTL_SLEEP_REJECT_ENA Write 1 to enable reject-to-sleep. (R/W)

RTC_CNTL_LIGHT_SLP_REJECT_EN Write 1 to enable reject-to-light-sleep. (R/W)

RTC_CNTL_DEEP_SLP_REJECT_EN Write 1 to enable reject-to-deep-sleep. (R/W)

Register 9.18. RTC_CNTL_CLK_CONF_REG (0x0068)



RTC_CNTL_EFUSE_CLK_FORCE_GATING Write 1 to FPU the eFuse clock gating. (R/W)

RTC_CNTL_EFUSE_CLK_FORCE_NOGATING Write 1 to FPD the eFuse clock gating. (R/W)

RTC_CNTL_FOSC_DIV_SEL_VLD Write 1 to synchronize RTC_CNTL_FOSC_DIV_SEL. Note that you have to invalidate the bus before modifying the frequency divider, then validate the new divider clock. (R/W)

RTC_CNTL_FOSC_DIV Configures the RC_FAST_DIV_CLK divider.

0x0: divided by 128
0x1: divided by 256
0x2: divided by 512
0x3: divided by 1024
(R/W)

RTC_CNTL_ENB_FOSC Write 1 to disable RC_FAST_CLK and RC_FAST_DIV_CLK. (R/W)

RTC_CNTL_ENB_FOSC_DIV Configures the RC_FAST_CLK divider.

0x0: RC_FAST_CLK divided by 256
0x1: RC_FAST_CLK
(R/W)

RTC_CNTL_DIG_FOSC_EN Write 1 to enable RC_FAST_CLK for the digital system. (R/W)

RTC_CNTL_FOSC_DIV_SEL Represents the RC_FAST_CLK divider, which is RTC_CNTL_FOSC_DIV_SEL + 1. (R/W)

Continued on the next page...

Register 9.18. RTC_CNTL_CLK_CONF_REG (0x0068)

Continued from the previous page...

RTC_CNTL_XTAL_FORCE_NOGATING Write 1 to force no gating to XTAL_CLK during sleep. (R/W)

RTC_CNTL_FOSC_FORCE_NOGATING Write 1 to disable force gating to XTAL_CLK during sleep. (R/W)

RTC_CNTL_FOSC_DFREQ Configures the RC_FAST_CLK frequency. (R/W)

RTC_CNTL_FOSC_FORCE_PD Write 1 to FPD RC_FAST_CLK. (R/W)

RTC_CNTL_FOSC_FORCE_PU Write 1 to FPU RC_FAST_CLK. (R/W)

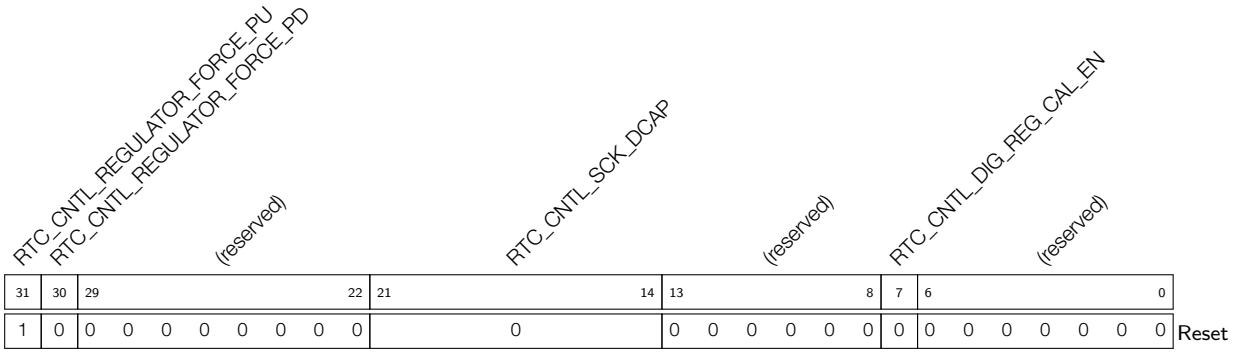
RTC_CNTL_XTAL_GLOBAL_FORCE_GATING Write 1 to force enable XTAL_CLK clock gating. (R/W)

RTC_CNTL_XTAL_GLOBAL_FORCE_NOGATING Write 1 to force bypass the XTAL_CLK clock gating. (R/W)

RTC_CNTL_FAST_CLK_RTC_SEL Configures the RTC_FAST_CLK.
0x0: XTAL_DIV_CLK
0x1: FOSC_DIV
(R/W)

RTC_CNTL_ANA_CLK_RTC_SEL Configures the RTC_SLOW_CLK.
0x0: RC_SLOW_CLK
0x1: OSC_SLOW_CLK
0x2: RC_FAST_DIV_CLK
0x3: Reserved
(R/W)

Register 9.19. RTC_CNTL_REG (0x0074)



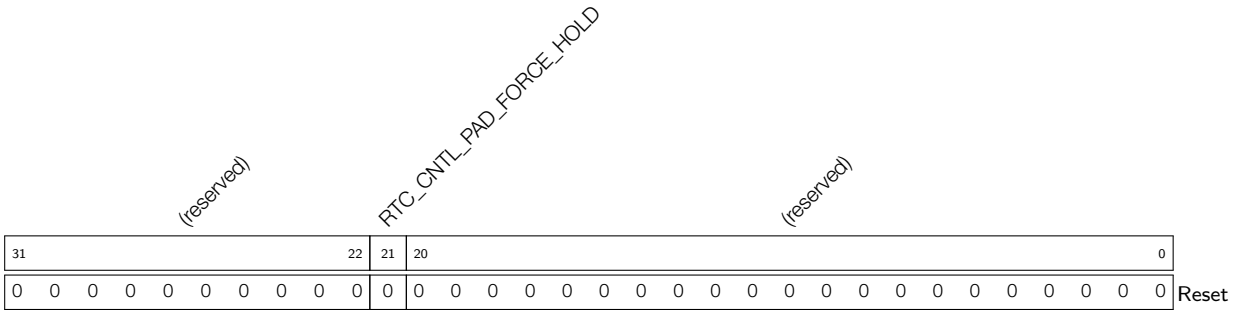
RTC_CNTL_DIG_REG_CAL_EN Write 1 to enable digital regulator calibration by SW. (R/W)

RTC_CNTL_SCK_DCAP Configures the RC_SLOW_CLK frequency. (R/W)

RTC_CNTL_REGULATOR_FORCE_PD Write 1 to FPD the low-power voltage regulator, which means decreasing its voltage to 0.8 V or lower. (R/W)

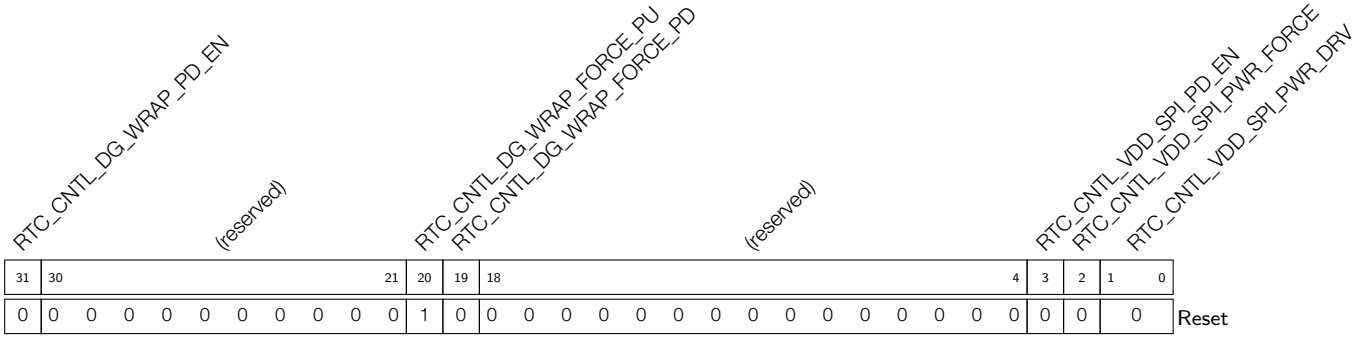
RTC_CNTL_REGULATOR_FORCE_PU Write 1 to FPU the low-power voltage regulator, which means increasing its voltage to higher than 0.8 V. (R/W)

Register 9.20. RTC_CNTL_PWC_REG (0x0078)



RTC_CNTL_PAD_FORCE_HOLD Write 1 to force RTC pad into hold state. (R/W)

Register 9.21. RTC_CNTL_DIG_PWC_REG (0x007C)



RTC_CNTL_VDD_SPI_PWR_DRV Configures the vdd_spi's drive intensity. (R/W)

RTC_CNTL_VDD_SPI_PWR_FORCE Write 1 to allow software to configure vdd_spi's drive intensity. (R/W)

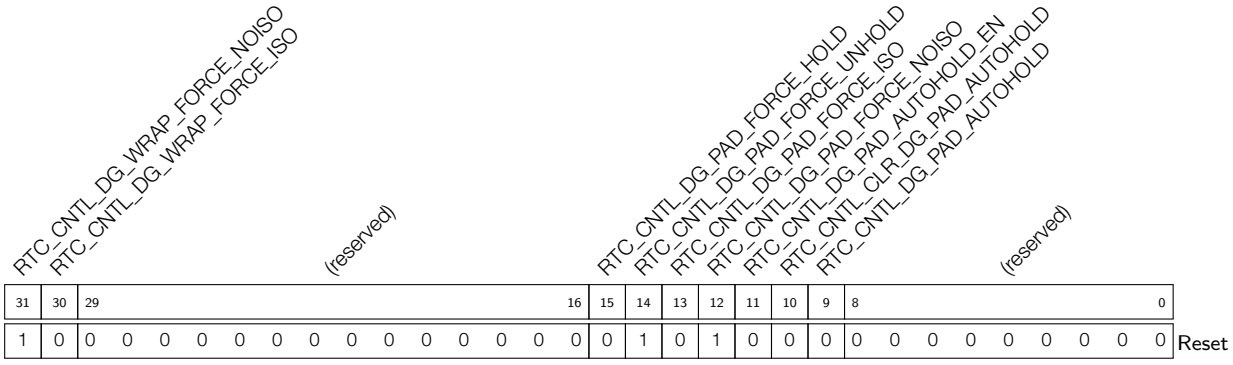
RTC_CNTL_VDD_SPI_PD_EN Write 1 to power down VDD_SPI in sleep. (R/W)

RTC_CNTL_DG_WRAP_FORCE_PD Write 1 to FPD the digital system. (R/W)

RTC_CNTL_DG_WRAP_FORCE_PU Write 1 to FPU the digital system. (R/W)

RTC_CNTL_DG_WRAP_PD_EN Write 1 to enable FPD digital system in sleep. (R/W)

Register 9.22. RTC_CNTL_DIG_ISO_REG (0x0080)



RTC_CNTL_DG_PAD_AUTOHOLD Indicates the auto-hold status of the digital GPIOs. (R/W)

RTC_CNTL_CLR_DG_PAD_AUTOHOLD Write 1 to clear the auto-hold enabler for the digital GPIOs. (R/W)

RTC_CNTL_DG_PAD_AUTOHOLD_EN Write 1 to allow the digital GPIOs to enter the auto-hold status. (R/W)

RTC_CNTL_DG_PAD_FORCE_NOISO Write 1 to disable the force isolation of the digital GPIOs. (R/W)

RTC_CNTL_DG_PAD_FORCE_ISO Write 1 to force isolation of the digital GPIOs. (R/W)

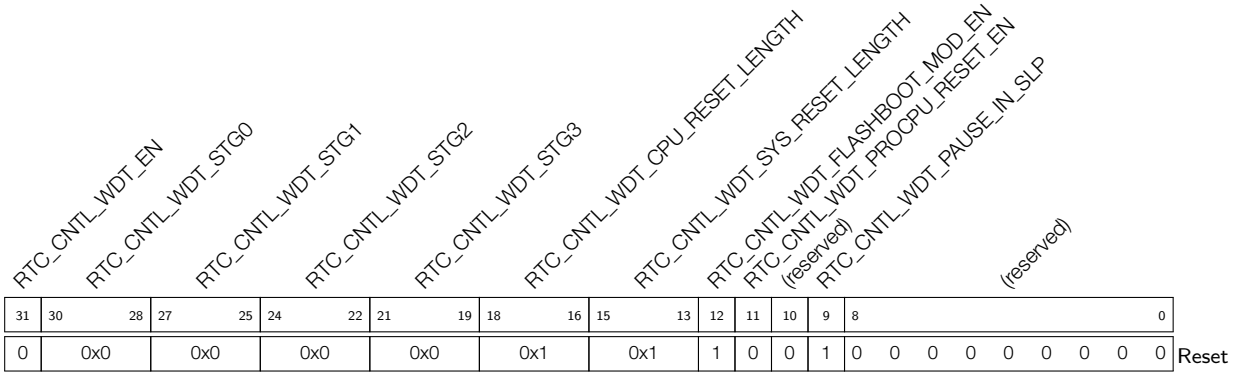
RTC_CNTL_DG_PAD_FORCE_UNHOLD Write 1 the force unhold the digital GPIOs. (R/W)

RTC_CNTL_DG_PAD_FORCE_HOLD Write 1 the force hold the digital GPIOs. (R/W)

RTC_CNTL_DG_WRAP_FORCE_ISO Write 1 to force isolation of the digital system. (R/W)

RTC_CNTL_DG_WRAP_FORCE_NOISO Write 1 to disable the force isolation of the digital system. (R/W)

Register 9.23. RTC_CNTL_WDTCONFIG0_REG (0x0084)



RTC_CNTL_WDT_PAUSE_IN_SLP Write 1 to pause the watchdog in sleep. (R/W)

RTC_CNTL_WDT_PROCPU_RESET_EN Write 1 to enable RTC WDT to reset CPU. (R/W)

RTC_CNTL_WDT_FLASHBOOT_MOD_EN Write 1 to enable watchdog when the chip boots from flash. (R/W)

RTC_CNTL_WDT_SYS_RESET_LENGTH Configures the length of the digital system reset counter. (R/W)

RTC_CNTL_WDT_CPU_RESET_LENGTH Configures the length of the CPU reset counter. (R/W)

RTC_CNTL_WDT_STG3 Configures the timeout action for RTC watchdog timer at stage 3.
0x1: triggers an interrupt
0x2: resets the CPU core
0x3: resets the digital system excluding RTC
0x4: resets the digital system including RTC
(R/W)

RTC_CNTL_WDT_STG2 Configures the timeout action for RTC watchdog timer at stage 2.
0x1: triggers an interrupt
0x2: resets the CPU core
0x3: resets the digital system excluding RTC
0x4: resets the digital system including RTC
(R/W)

Continued on the next page...

Register 9.23. RTC_CNTL_WDTCONFIG0_REG (0x0084)

Continued from the previous page...

RTC_CNTL_WDT_STG1 Configures the timeout action for RTC watchdog timer at stage 1.

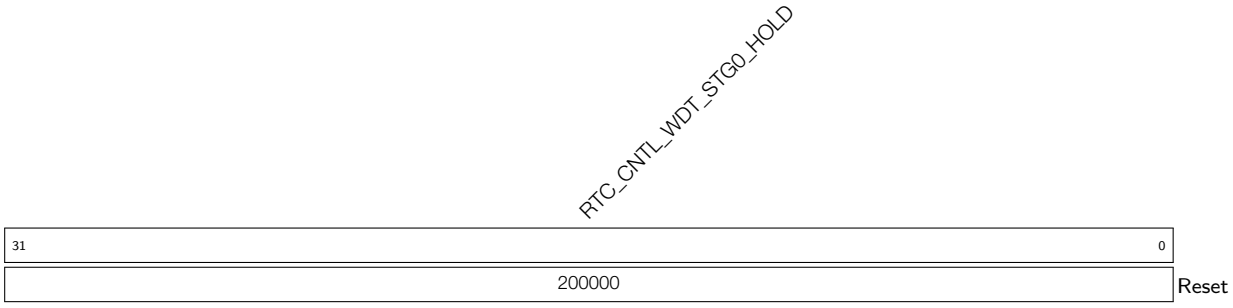
0x1: triggers an interrupt
0x2: resets the CPU core
0x3: resets the digital system excluding RTC
0x4: resets the digital system including RTC
(R/W)

RTC_CNTL_WDT_STG0 Configures the timeout action for RTC watchdog timer at stage 0.

0x1: triggers an interrupt
0x2: resets the CPU core
0x3: resets the digital system excluding RTC
0x4: resets the digital system including RTC
(R/W)

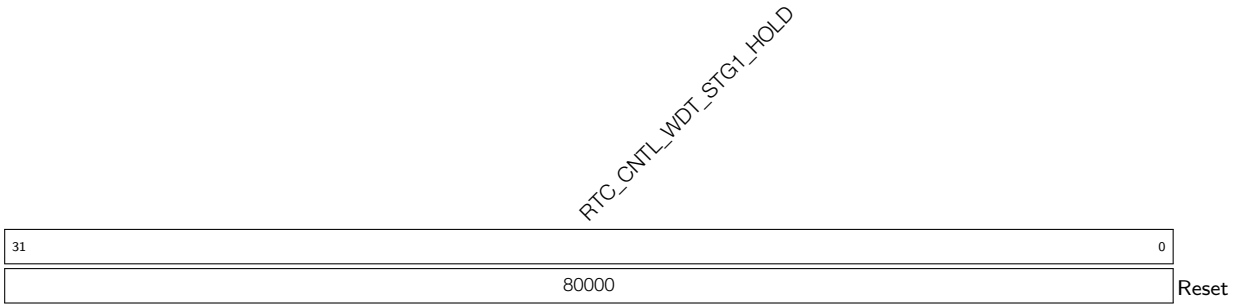
RTC_CNTL_WDT_EN Write 1 to enable the RTC watchdog. (R/W)

Register 9.24. RTC_CNTL_WDTCONFIG1_REG (0x0088)



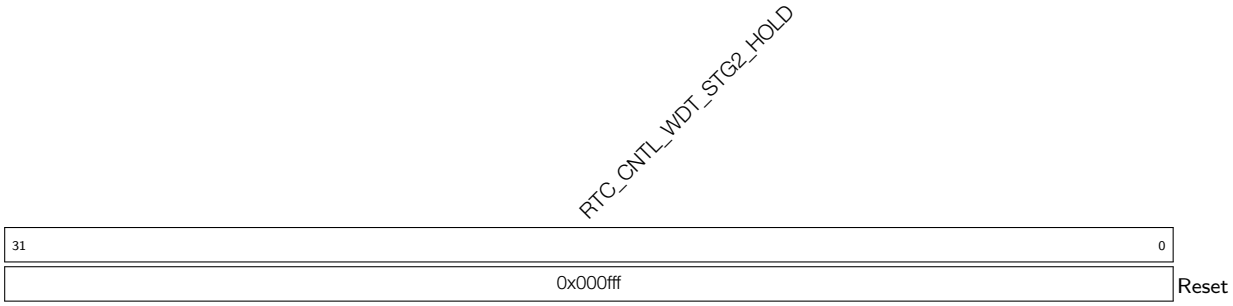
RTC_CNTL_WDT_STG0_HOLD Configures the hold time of RTC watchdog in stage 0. (R/W)

Register 9.25. RTC_CNTL_WDTCONFIG2_REG (0x008C)



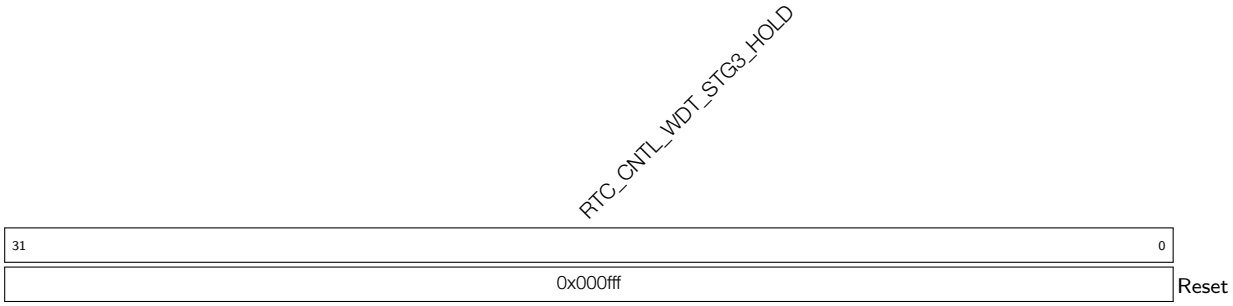
RTC_CNTL_WDT_STG1_HOLD Configures the hold time of RTC watchdog in stage 1. (R/W)

Register 9.26. RTC_CNTL_WDTCONFIG3_REG (0x0090)



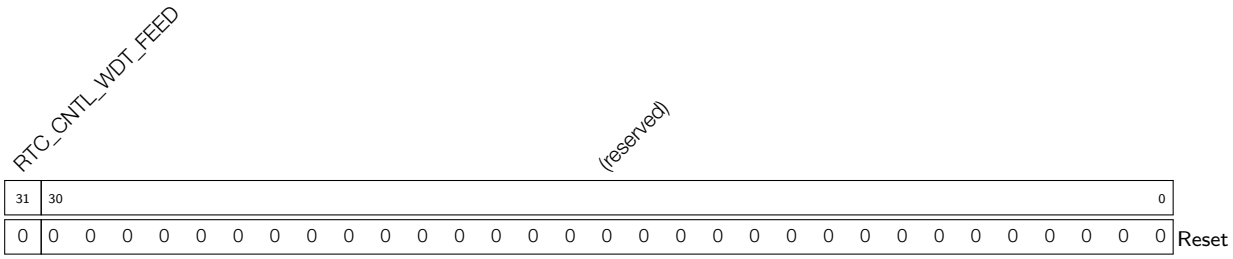
RTC_CNTL_WDT_STG2_HOLD Configures the hold time of RTC watchdog in stage 2. (R/W)

Register 9.27. RTC_CNTL_WDTCONFIG4_REG (0x0094)



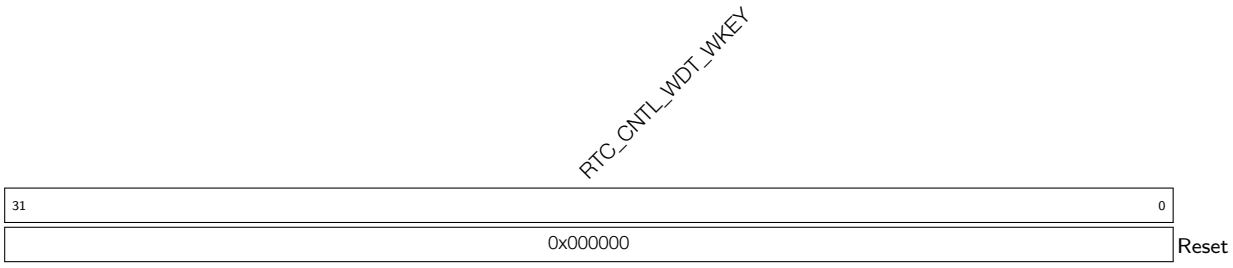
RTC_CNTL_WDT_STG3_HOLD Configures the hold time of RTC watchdog in stage 3. (R/W)

Register 9.28. RTC_CNTL_WDTFEED_REG (0x0098)



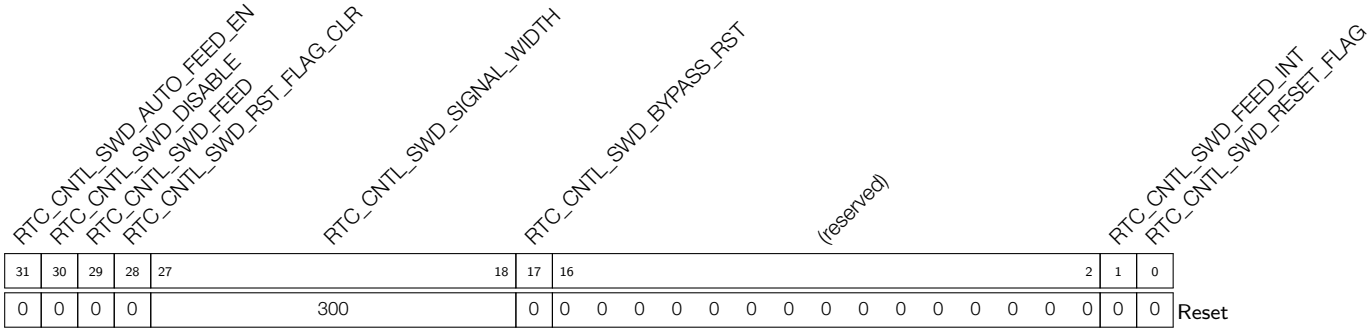
RTC_CNTL_WDT_FEED Write 1 to feed the RTC watchdog. (R/W)

Register 9.29. RTC_CNTL_WDTWPROTECT_REG (0x009C)



RTC_CNTL_WDT_WKEY Configures the write protection key of the RTC watchdog. (R/W)

Register 9.30. RTC_CNTL_SWD_CONF_REG (0x00A0)



RTC_CNTL_SWD_RESET_FLAG Represents the super watchdog reset flag. (R/W)

RTC_CNTL_SWD_FEED_INT Represents super watchdog will be fed via SW. (R/W)

RTC_CNTL_SWD_BYPASS_RST Write 1 to bypass super watchdog reset. (R/W)

RTC_CNTL_SWD_SIGNAL_WIDTH Configures the signal width sent to the super watchdog. (R/W)

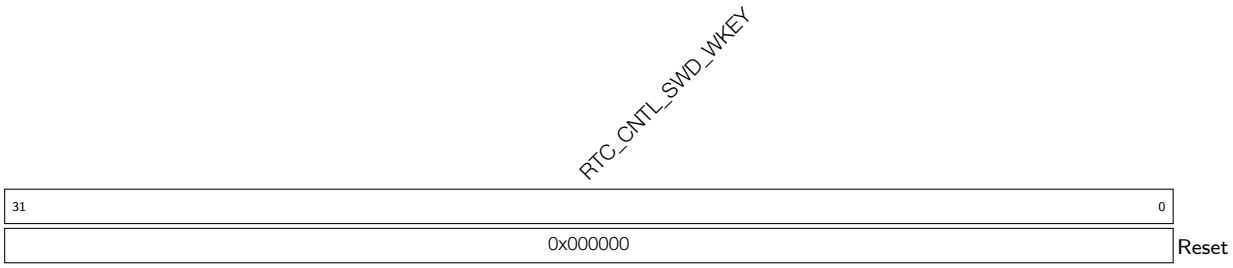
RTC_CNTL_SWD_RST_FLAG_CLR Write 1 to reset the super watchdog reset flag. (R/W)

RTC_CNTL_SWD_FEED Write 1 to feed the super watchdog via SW. (R/W)

RTC_CNTL_SWD_DISABLE Write 1 to disable super watchdog. (R/W)

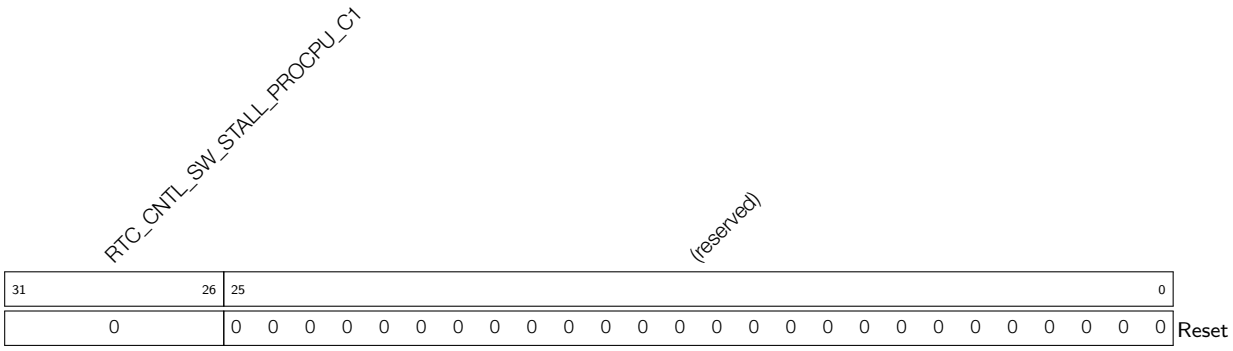
RTC_CNTL_SWD_AUTO_FEED_EN Write 1 to enable automatic watchdog feeding upon interrupt. (R/W)

Register 9.31. RTC_CNTL_SWD_WPROTECT_REG (0x00A4)



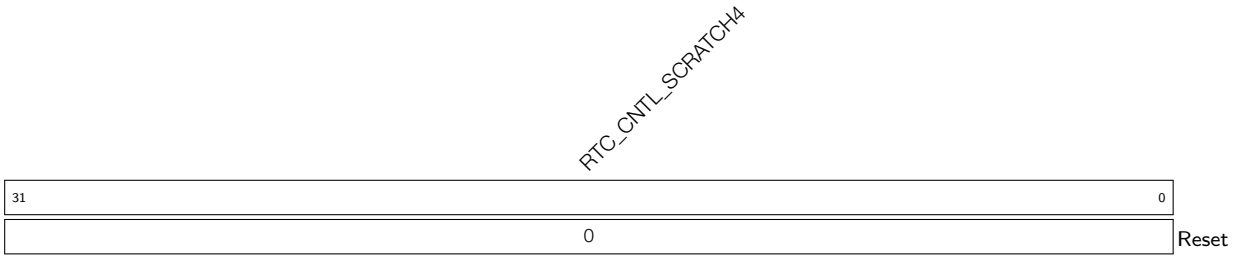
RTC_CNTL_SWD_WKEY Configures the write protection key of the super watchdog. (R/W)

Register 9.32. RTC_CNTL_SW_CPU_STALL_REG (0x00A8)



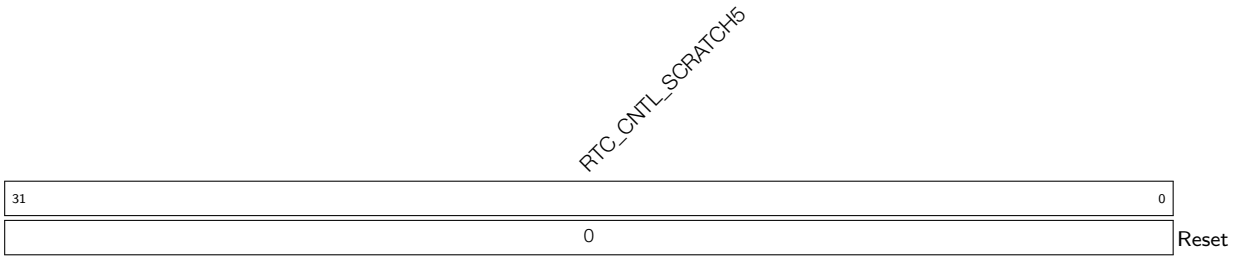
RTC_CNTL_SW_STALL_PROCPU_C1 Write 0x21 to stall the CPU by SW. Valid only when RTC_CNTL_SW_STALL_PROCPU_C0 is configured to 0x2. (R/W)

Register 9.33. RTC_CNTL_STORE4_REG (0x00AC)



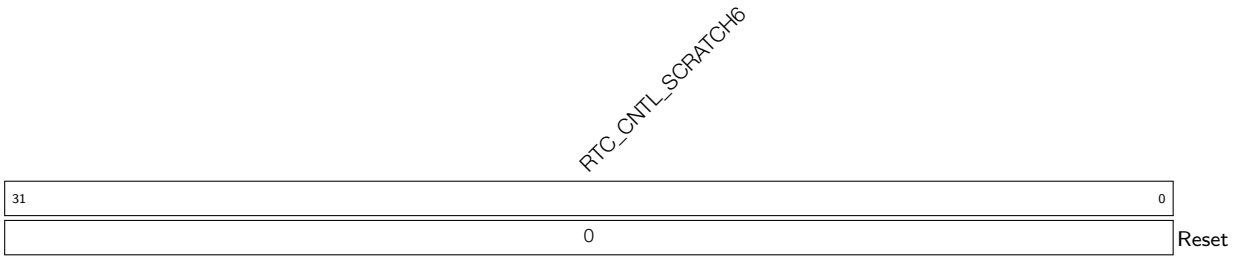
RTC_CNTL_SCRATCH4 Reservation register 4. (R/W)

Register 9.34. RTC_CNTL_STORE5_REG (0x00B0)



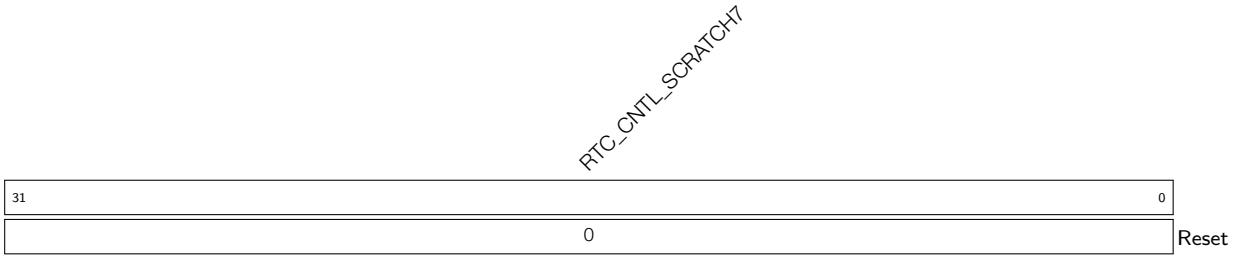
RTC_CNTL_SCRATCH5 Reservation register 5. (R/W)

Register 9.35. RTC_CNTL_STORE6_REG (0x00B4)



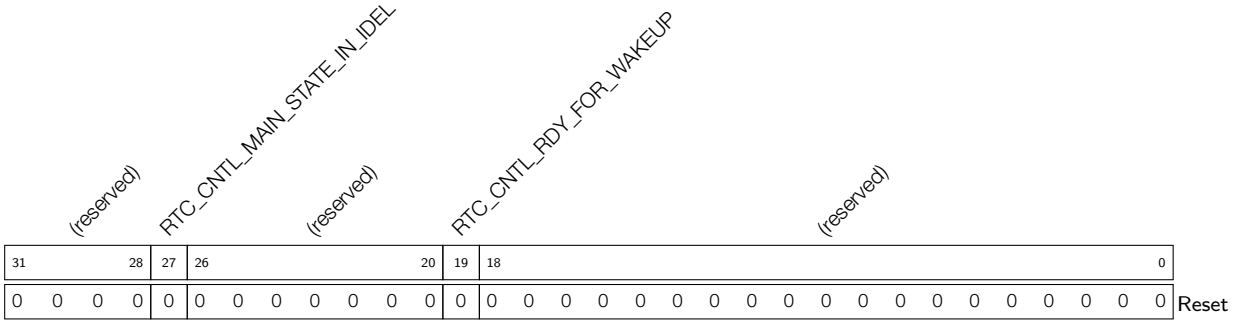
RTC_CNTL_SCRATCH6 Reservation register 6. (R/W)

Register 9.36. RTC_CNTL_STORE7_REG (0x00B8)



RTC_CNTL_SCRATCH7 Reservation register 7. (R/W)

Register 9.37. RTC_CNTL_LOW_POWER_ST_REG (0x00BC)

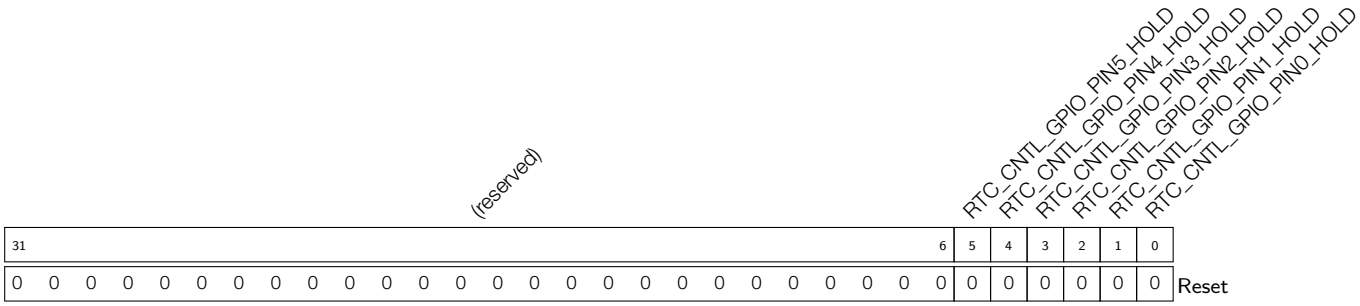


RTC_CNTL_RDY_FOR_WAKEUP Indicates the RTC is ready to be triggered by any wakeup source. (RO)

RTC_CNTL_MAIN_STATE_IN_IDLE Indicates the RTC state.

- 0: the chip can be either
  - in sleep modes.
  - entering sleep modes. In this case, wait until RTC_CNTL_RDY_FOR_WAKEUP bit is set, then you can wake up the chip.
  - exiting sleep mode. In this case, RTC_CNTL_MAIN_STATE_IN_IDLE will eventually become 1.
- 1: the chip is not in sleep modes (i.e. running normally).

Register 9.38. RTC_CNTL_PAD_HOLD_REG (0x00C4)



RTC_CNTL_GPIO_PIN0_HOLD Sets the GPIO 0 to the holding state. (R/W)

RTC_CNTL_GPIO_PIN1_HOLD Sets the GPIO 1 to the holding state. (R/W)

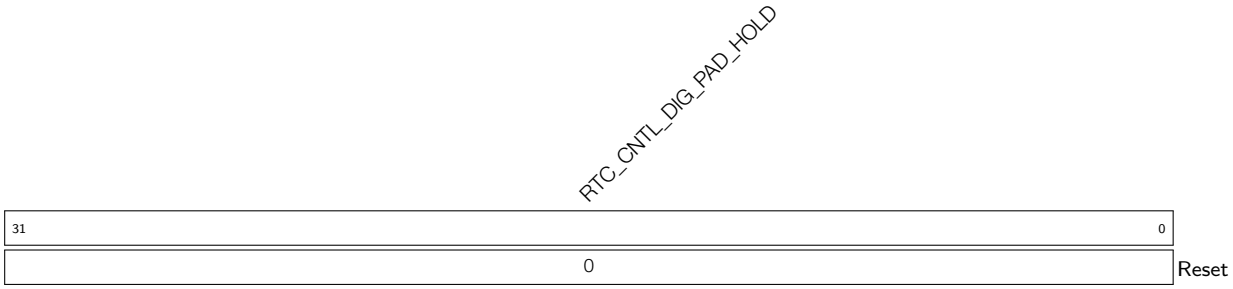
RTC_CNTL_GPIO_PIN2_HOLD Sets the GPIO 2 to the holding state. (R/W)

RTC_CNTL_GPIO_PIN3_HOLD Sets the GPIO 3 to the holding state. (R/W)

RTC_CNTL_GPIO_PIN4_HOLD Sets the GPIO 4 to the holding state. (R/W)

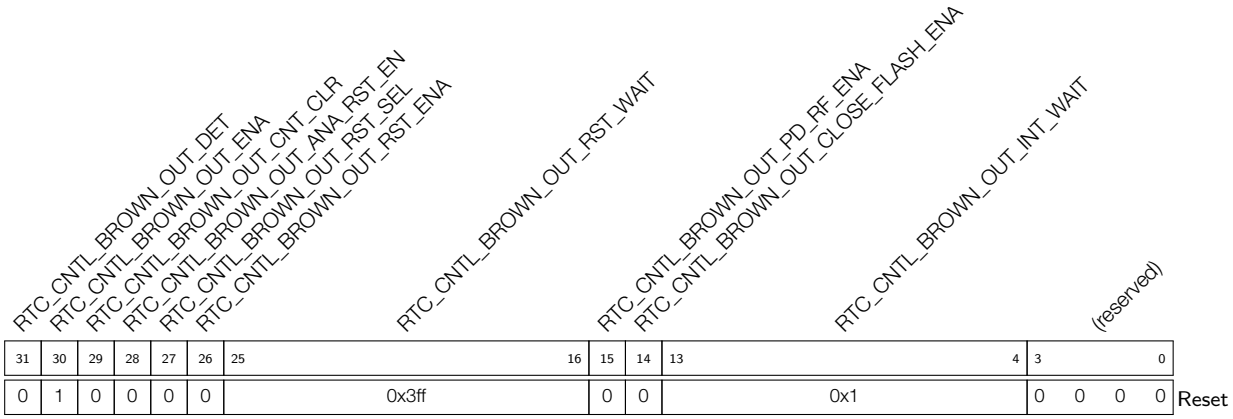
RTC_CNTL_GPIO_PIN5_HOLD Sets the GPIO 5 to the holding state. (R/W)

Register 9.39. RTC_CNTL_DIG_PAD_HOLD_REG (0x00C8)



RTC_CNTL_DIG_PAD_HOLD Set GPIO 6 to GPIO 20 to the holding state. (See bitmap to locate any GPIO). (R/W)

Register 9.40. RTC_CNTL_BROWN_OUT_REG (0x00CC)



RTC_CNTL_BROWN_OUT_INT_WAIT Configures the waiting cycles before sending an interrupt. (R/W)

RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA Write 1 to enable PD the flash when a brownout happens. (R/W)

RTC_CNTL_BROWN_OUT_PD_RF_ENA Write 1 to enable PD the RF circuits when a brownout happens. (R/W)

RTC_CNTL_BROWN_OUT_RST_WAIT Configures the waiting cycles before the reset after a brownout. (R/W)

RTC_CNTL_BROWN_OUT_RST_ENA Write 1 to reset brown-out. (R/W)

RTC_CNTL_BROWN_OUT_RST_SEL Configures the reset type when a brownout happens in mode0.
0x0: system reset
0x1: chip reset
(R/W)

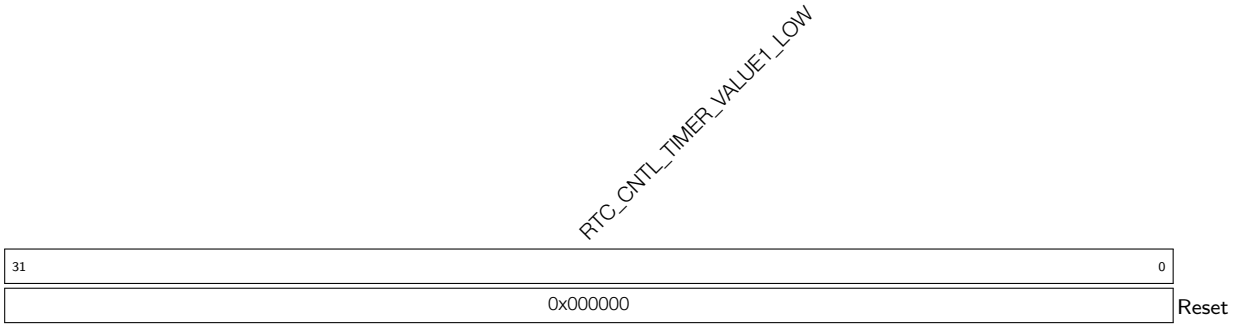
RTC_CNTL_BROWN_OUT_ANA_RST_EN Write 1 to enable brownout detection mode1. (R/W)

RTC_CNTL_BROWN_OUT_CNT_CLR Write 1 to clear the brownout counter. (R/W)

RTC_CNTL_BROWN_OUT_ENA Write 1 to enable brownout detection mode0. (R/W)

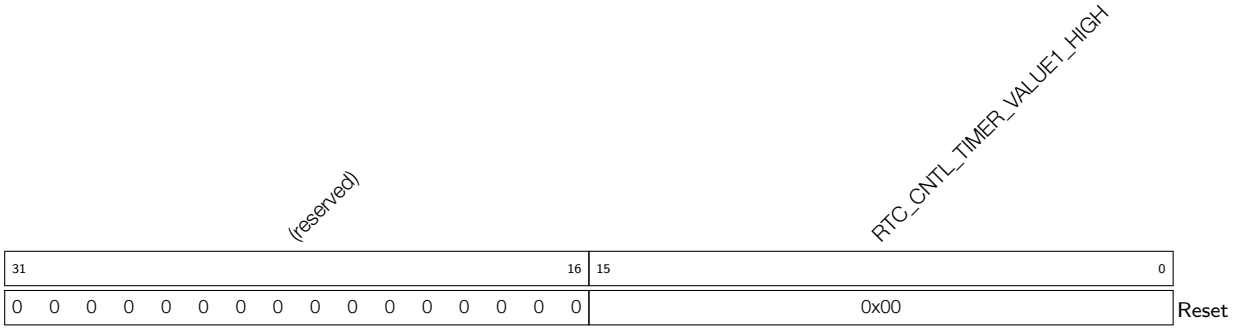
RTC_CNTL_BROWN_OUT_DET Represents the status of the brownout signal. (R/W)

Register 9.41. RTC_CNTL_TIME_LOW1_REG (0x00D0)



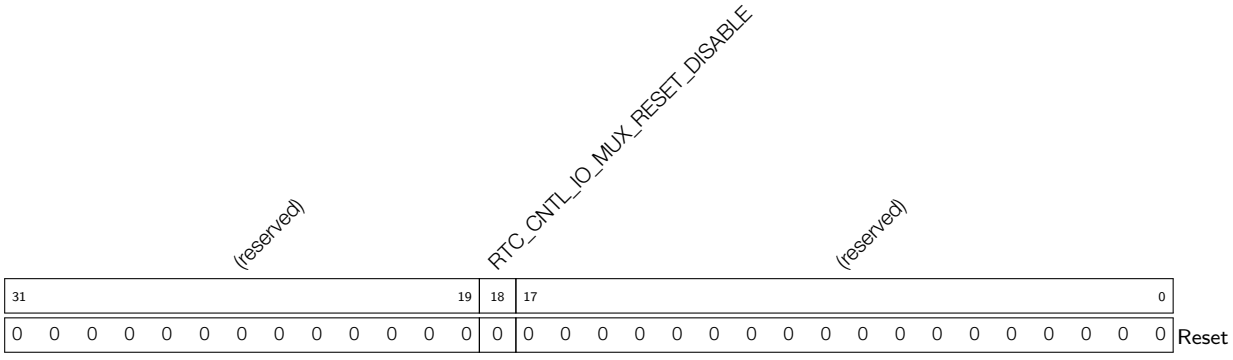
RTC_CNTL_TIMER_VALUE1_LOW Represents the lower 32 bits of RTC timer 1. (R/W)

Register 9.42. RTC_CNTL_TIME_HIGH1_REG (0x00D4)



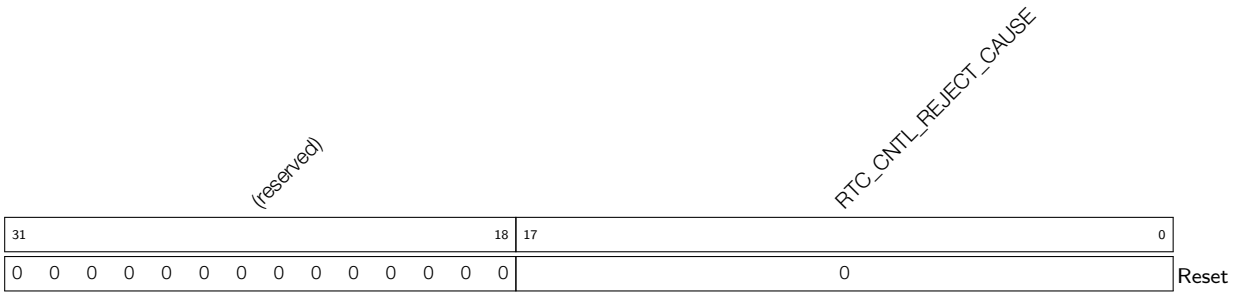
RTC_CNTL_TIMER_VALUE1_HIGH Represents the higher 16 bits of RTC timer 1. (R/W)

Register 9.43. RTC_CNTL_USB_CONF_REG (0x00D8)



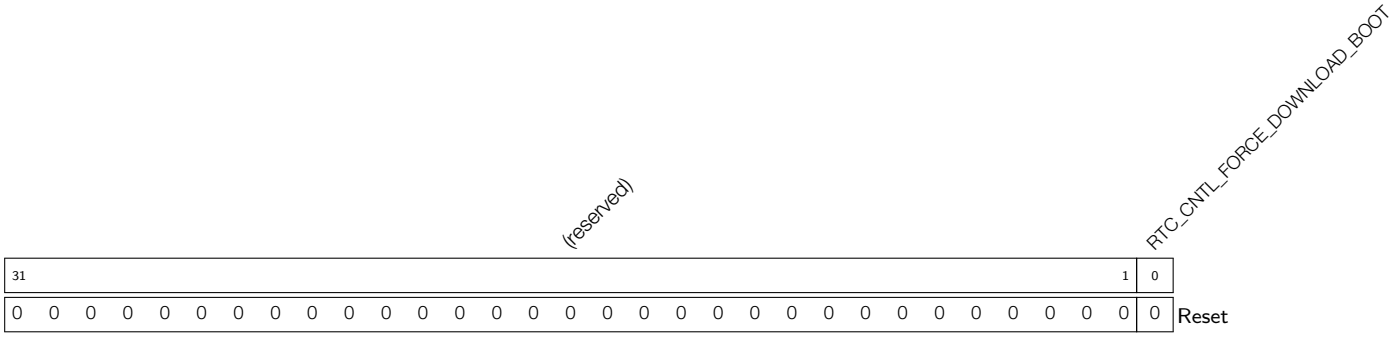
RTC_CNTL_IO_MUX_RESET_DISABLE Write 1 to disable io_mux reset. (R/W)

Register 9.44. RTC_CNTL_SLP_REJECT_CAUSE_REG (0x00DC)



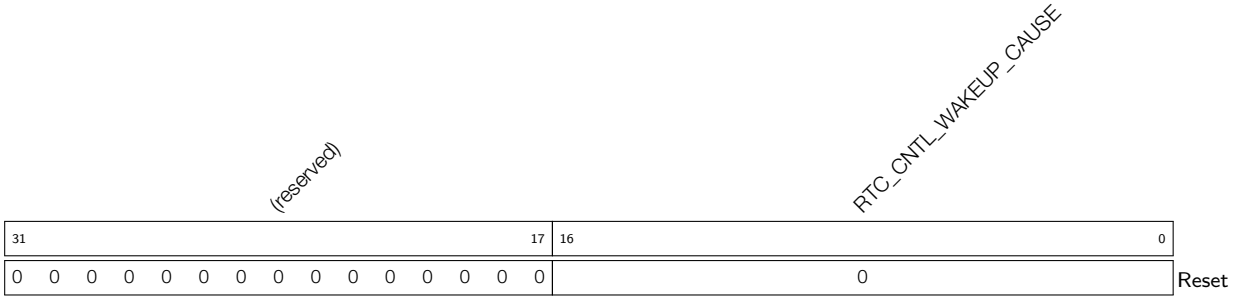
RTC_CNTL_REJECT_CAUSE Represents the reject-to-sleep cause. (R/W)

Register 9.45. RTC_CNTL_OPTION1_REG (0x00E0)



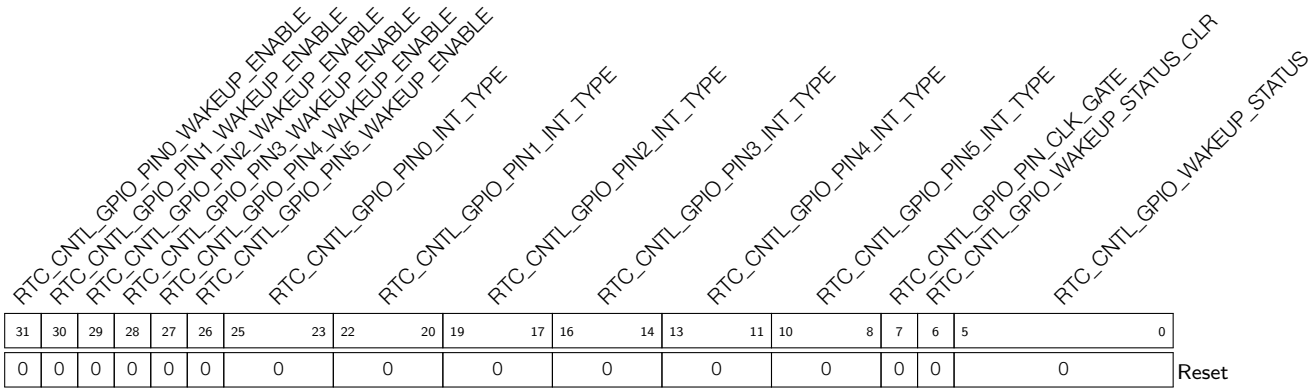
RTC_CNTL_FORCE_DOWNLOAD_BOOT Write 1 to force the chip to boot from the download mode. (R/W)

Register 9.46. RTC_CNTL_SLP_WAKEUP_CAUSE_REG (0x00E4)



RTC_CNTL_WAKEUP_CAUSE Represents the wakeup cause. (R/W)

Register 9.47. RTC_CNTL_CNTL_GPIO_WAKEUP_REG (0x00FC)



RTC_CNTL_GPIO_WAKEUP_STATUS Write 1 to set the RTC GPIO wakeup flag. (R/W)

RTC_CNTL_GPIO_WAKEUP_STATUS_CLR Write 1 to clear the RTC GPIO flag. (R/W)

RTC_CNTL_GPIO_PIN_CLK_GATE Write 1 to enable the RTC GPIO clock gating. (R/W)

RTC_CNTL_GPIO_PIN5_INT_TYPE Configures GPIO 5 wakeup type. (R/W)

RTC_CNTL_GPIO_PIN4_INT_TYPE Configures GPIO 4 wakeup type. (R/W)

RTC_CNTL_GPIO_PIN3_INT_TYPE Configures GPIO 3 wakeup type. (R/W)

RTC_CNTL_GPIO_PIN2_INT_TYPE Configures GPIO 2 wakeup type. (R/W)

RTC_CNTL_GPIO_PIN1_INT_TYPE Configures GPIO 1 wakeup type. (R/W)

RTC_CNTL_GPIO_PIN0_INT_TYPE Configures GPIO 0 wakeup type. (R/W)

RTC_CNTL_GPIO_PIN5_WAKEUP_ENABLE Write 1 to enable wakeup from RTC GPIO 5. (R/W)

RTC_CNTL_GPIO_PIN4_WAKEUP_ENABLE Write 1 to enable wakeup from RTC GPIO 4. (R/W)

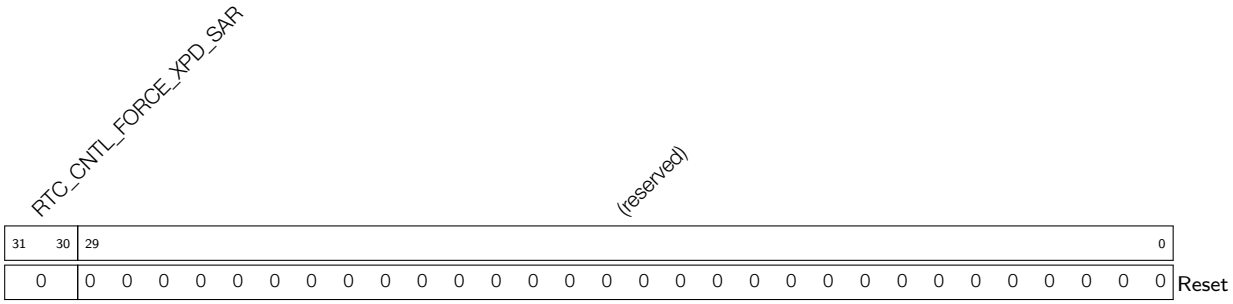
RTC_CNTL_GPIO_PIN3_WAKEUP_ENABLE Write 1 to enable wakeup from RTC GPIO 3. (R/W)

RTC_CNTL_GPIO_PIN2_WAKEUP_ENABLE Write 1 to enable wakeup from RTC GPIO 2. (R/W)

RTC_CNTL_GPIO_PIN1_WAKEUP_ENABLE Write 1 to enable wakeup from RTC GPIO 1. (R/W)

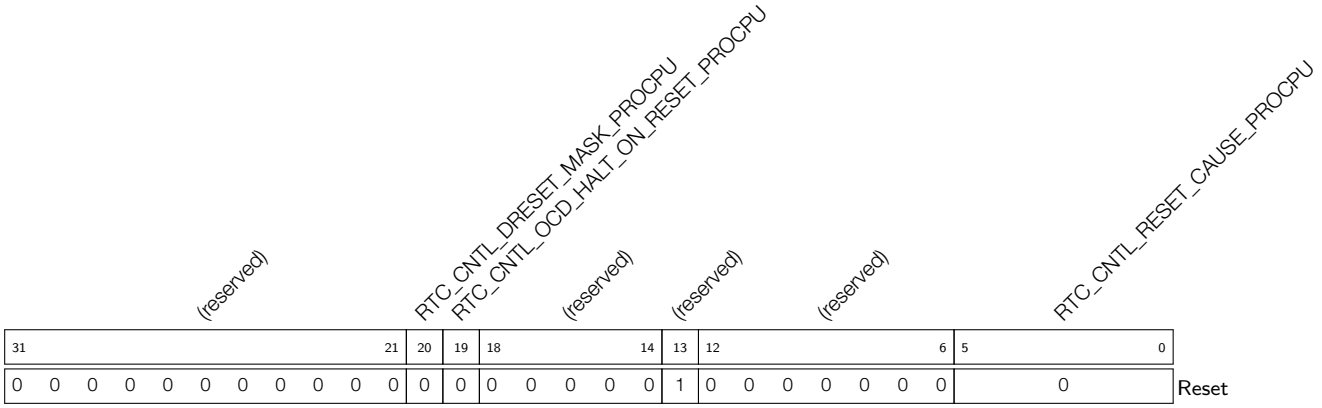
RTC_CNTL_GPIO_PIN0_WAKEUP_ENABLE Write 1 to enable wakeup from RTC GPIO 0. (R/W)

Register 9.48. RTC_CNTL_CNTL_SENSOR_CTRL_REG (0x0108)



RTC_CNTL_FORCE_XPD_SAR Set this field to FPU SAR ADC. (R/W)

Register 9.49. RTC_CNTL_RESET_STATE_REG (0x0030)

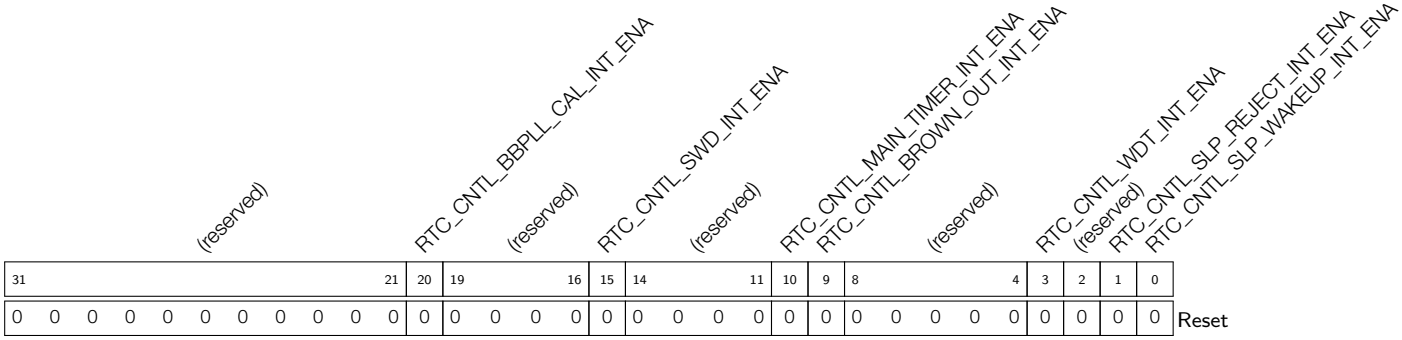


RTC_CNTL_RESET_CAUSE_PROCPU Represents the CPU reset cause. (R/W)

RTC_CNTL_OCD_HALT_ON_RESET_PROCPU Write 1 to send CPU into halt state upon CPU reset. (R/W)

RTC_CNTL_DRESET_MASK_PROCPU Write 1 to bybass D-reset. (R/W)

Register 9.50. RTC_CNTL_INT_ENA_RTC_REG (0x0038)



RTC_CNTL_SLP_WAKEUP_INT_ENA Write 1 to enable interrupt when chip wakes up from sleep. (R/W)

RTC_CNTL_SLP_REJECT_INT_ENA Write 1 to enable interrupt when chip rejects to go to sleep. (R/W)

RTC_CNTL_WDT_INT_ENA Write 1 to enable RTC WDT interrupt. (R/W)

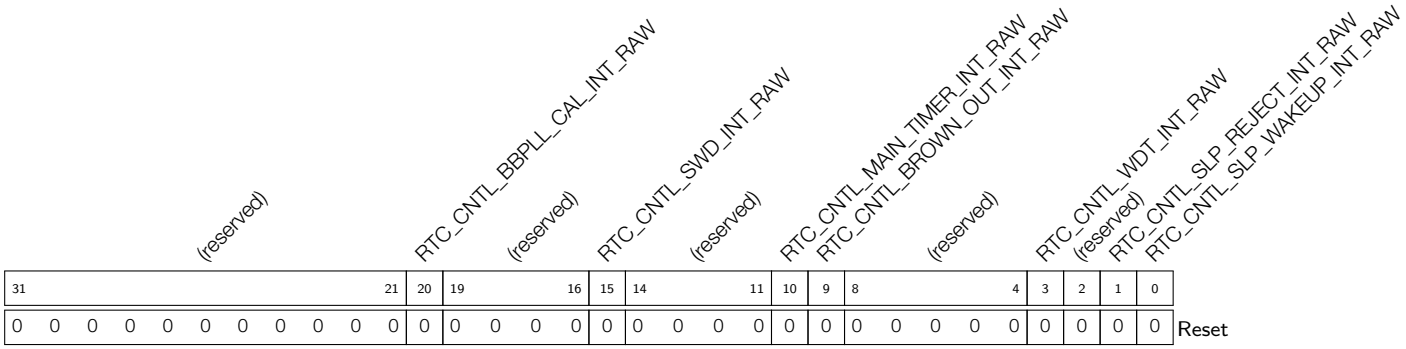
RTC_CNTL_BROWN_OUT_INT_ENA Write 1 to enable the brownout interrupt. (R/W)

RTC_CNTL_MAIN_TIMER_INT_ENA Write 1 to enable the RTC timer interrupt. (R/W)

RTC_CNTL_SWD_INT_ENA Write 1 to enable the super watchdog interrupt. (R/W)

RTC_CNTL_BBPLL_CAL_INT_ENA Write 1 to enable interrupt upon the ending of a BB_PLL call. (R/W)

Register 9.51. RTC_CNTL_INT_RAW_RTC_REG (0x003C)



RTC_CNTL_SLP_WAKEUP_INT_RAW Represents the raw interrupt triggered when the chip wakes up from sleep. (R/W)

RTC_CNTL_SLP_REJECT_INT_RAW Represents the raw interrupt triggered when the chip rejects to go to sleep. (R/W)

RTC_CNTL_WDT_INT_RAW Represents the raw watchdog interrupt. (R/W)

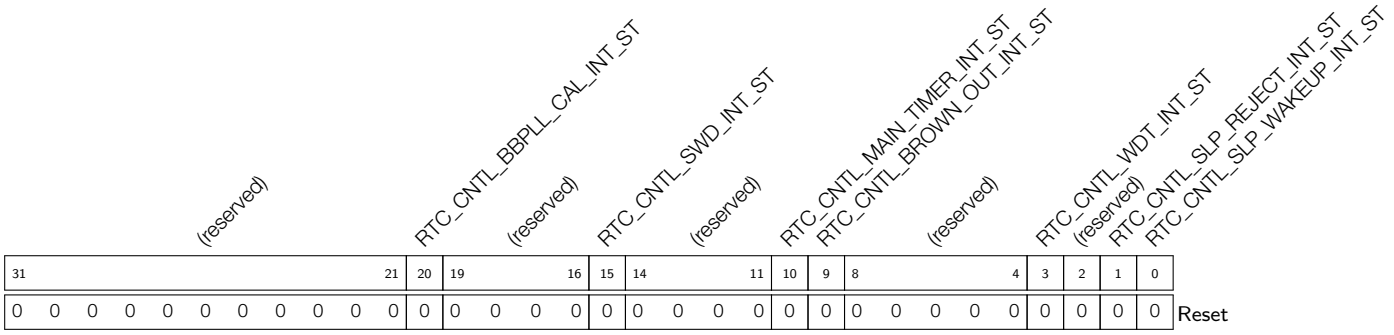
RTC_CNTL_BROWN_OUT_INT_RAW Represents the raw brownout interrupt. (R/W)

RTC_CNTL_MAIN_TIMER_INT_RAW Represents the raw RTC main timer interrupt. (R/W)

RTC_CNTL_SWD_INT_RAW Represents the raw super watchdog interrupt. (R/W)

RTC_CNTL_BBPLL_CAL_INT_RAW Represents the raw interrupt upon the ending of a BB_PLL call. (R/W)

Register 9.52. RTC_CNTL_INT_ST_RTC_REG (0x0040)



RTC_CNTL_SLP_WAKEUP_INT_ST Represents the status of the interrupt triggered when the chip wakes up from sleep. (R/W)

RTC_CNTL_SLP_REJECT_INT_ST Represents the status of the interrupt triggered when the chip rejects to go to sleep. (R/W)

RTC_CNTL_WDT_INT_ST Represents the status of the RTC watchdog interrupt. (R/W)

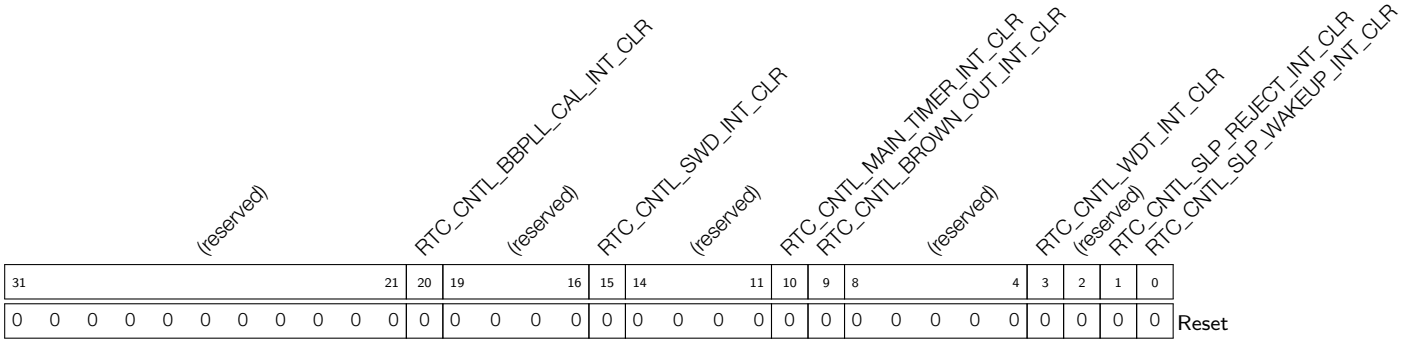
RTC_CNTL_BROWN_OUT_INT_ST Represents the status of the brownout interrupt. (R/W)

RTC_CNTL_MAIN_TIMER_INT_ST Represents the status of the RTC main timer interrupt. (R/W)

RTC_CNTL_SWD_INT_ST Represents the status of the super watchdog interrupt. (R/W)

RTC_CNTL_BBPLL_CAL_INT_ST Represents the status of the interrupt triggered upon the ending of a BB_PLL call. (R/W)

Register 9.53. RTC_CNTL_INT_CLR_RTC_REG (0x0044)



RTC_CNTL_SLP_WAKEUP_INT_CLR Write 1 to clear the interrupt triggered when the chip wakes up from sleep. (R/W)

RTC_CNTL_SLP_REJECT_INT_CLR Write 1 to clear the interrupt triggered when the chip rejects to go to sleep. (R/W)

RTC_CNTL_WDT_INT_CLR Write 1 to clear the RTC watchdog interrupt. (R/W)

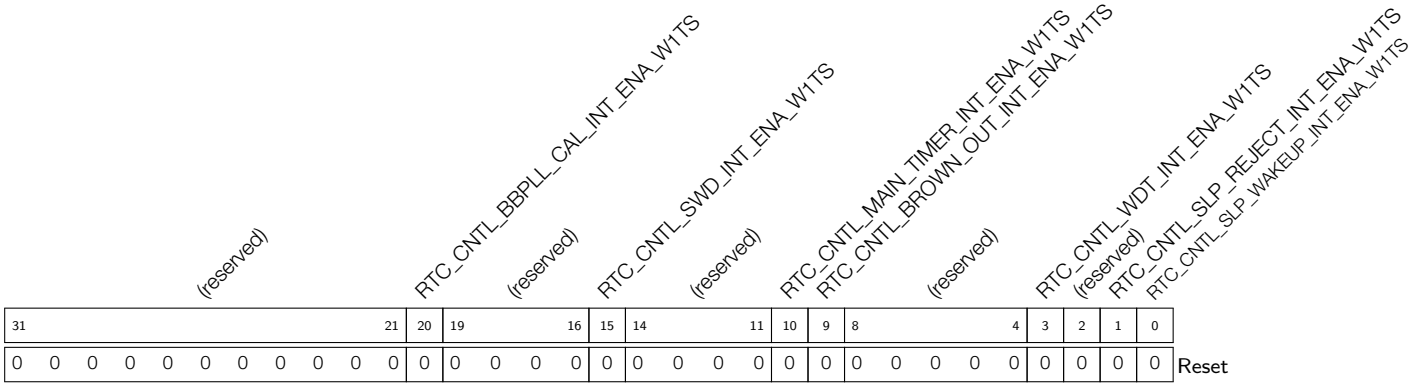
RTC_CNTL_BROWN_OUT_INT_CLR Write 1 to clear the brownout interrupt. (R/W)

RTC_CNTL_MAIN_TIMER_INT_CLR Write 1 to clear the RTC main timer interrupt. (R/W)

RTC_CNTL_SWD_INT_CLR Write 1 to clear the super watch dog interrupt state. (R/W)

RTC_CNTL_BBPLL_CAL_INT_CLR Write 1 to clear the interrupt triggered upon the ending of a BB_PLL call. (R/W)

Register 9.54. RTC_CNTL_INT_ENA_RTC_W1TS_REG (0x00EC)



RTC_CNTL_SLP_WAKEUP_INT_ENA_W1TS Write 1 to enable interrupt when the chip wakes up from sleep. If the value 1 is written to this bit, the RTC_CNTL_SLP_WAKEUP_INT_ENA field will be set to 1. (R/W)

RTC_CNTL_SLP_REJECT_INT_ENA_W1TS Write 1 to enable interrupt when the chip rejects to go to sleep. If the value 1 is written to this bit, the RTC_CNTL_SLP_REJECT_INT_ENA field will be set to 1. (R/W)

RTC_CNTL_WDT_INT_ENA_W1TS Write 1 to enable the RTC watchdog interrupt. If the value 1 is written to this bit, the RTC_CNTL_WDT_INT_ENA field will be set to 1. (R/W)

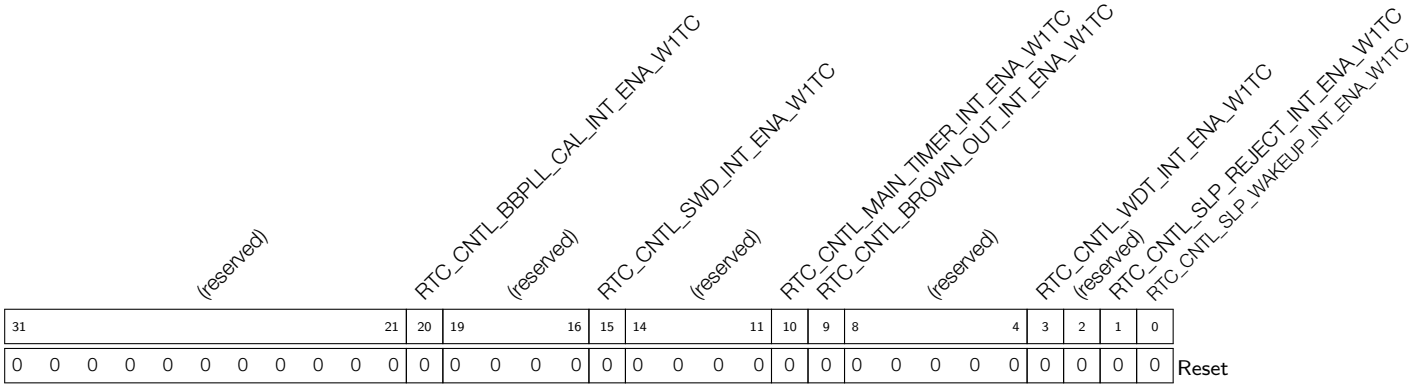
RTC_CNTL_BROWN_OUT_INT_ENA_W1TS Write 1 to enable the brownout interrupt. If the value 1 is written to this bit, the RTC_CNTL_BROWN_OUT_INT_ENA field will be set to 1. (R/W)

RTC_CNTL_MAIN_TIMER_INT_ENA_W1TS Write 1 to enable the RTC main timer interrupt. If the value 1 is written to this bit, the RTC_CNTL_MAIN_TIMER_INT_ENA field will be set to 1. (R/W)

RTC_CNTL_SWD_INT_ENA_W1TS Write 1 to enable the super watchdog interrupt. If the value 1 is written to this bit, the RTC_CNTL_SWD_INT_ENA field will be set to 1. (R/W)

RTC_CNTL_BBPLL_CAL_INT_ENA_W1TS Write 1 to enable interrupt upon the ending of a BB_PLL call. If the value 1 is written to this bit, the RTC_CNTL_BBPLL_CAL_INT_ENA field will be set to 1. (R/W)

Register 9.55. RTC_CNTL_INT_ENA_RTC_W1TC_REG (0x00F0)



RTC_CNTL_SLP_WAKEUP_INT_ENA_W1TC Write 1 to disable interrupt when the chip wakes up from sleep. If the value 1 is written to this bit, the RTC_CNTL_SLP_WAKEUP_INT_CLR field will be cleared. (R/W)

RTC_CNTL_SLP_REJECT_INT_ENA_W1TC Write 1 to disable interrupt when the chip rejects to go to sleep. If the value 1 is written to this bit, the RTC_CNTL_SLP_REJECT_INT_CLR field will be cleared. (R/W)

RTC_CNTL_WDT_INT_ENA_W1TC Write 1 to disable the RTC watchdog interrupt. If the value 1 is written to this bit, the RTC_CNTL_WDT_INT_CLR field will be cleared. (R/W)

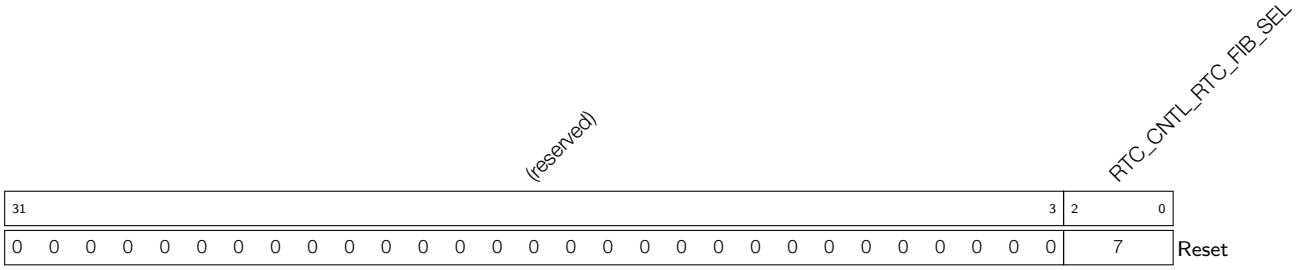
RTC_CNTL_BROWN_OUT_INT_ENA_W1TC Write 1 to disable the brownout interrupt. If the value 1 is written to this bit, the RTC_CNTL_BROWN_OUT_INT_CLR field will be cleared. (R/W)

RTC_CNTL_MAIN_TIMER_INT_ENA_W1TC Write 1 to disable the RTC timer interrupt. If the value 1 is written to this bit, the RTC_CNTL_MAIN_TIMER_INT_CLR field will be cleared. (R/W)

RTC_CNTL_SWD_INT_ENA_W1TC Write 1 to disable the super watchdog interrupt enable bit. If the value 1 is written to this bit, the RTC_CNTL_SWD_INT_CLR field will be cleared. (R/W)

RTC_CNTL_BBPLL_CAL_INT_ENA_W1TC Write 1 to disable the interrupt upon the ending of a BB_PLL call. If the value 1 is written to this bit, the RTC_CNTL_BBPLL_CAL_INT_CLR field will be cleared. (R/W)

Register 9.56. RTC_CNTL_FIB_SEL_REG (0x00F8)



RTC_CNTL_RTC_FIB_SEL Configures the brownout detector. (R/W)

10 System Timer (SYSTIMER)

10.1 Overview

ESP8684 provides a 52-bit timer, which can be used to generate tick interrupts for operating system, or be used as a general timer to generate periodic interrupts or one-time interrupts. With the help of RTC timer, system timer can be kept up to date after Light-sleep or Deep-sleep.

The timer consists of two counters UNIT0 and UNIT1. The counter values can be monitored by three comparators COMP0, COMP1 and COMP2. See the timer block diagram on Figure 10-1.

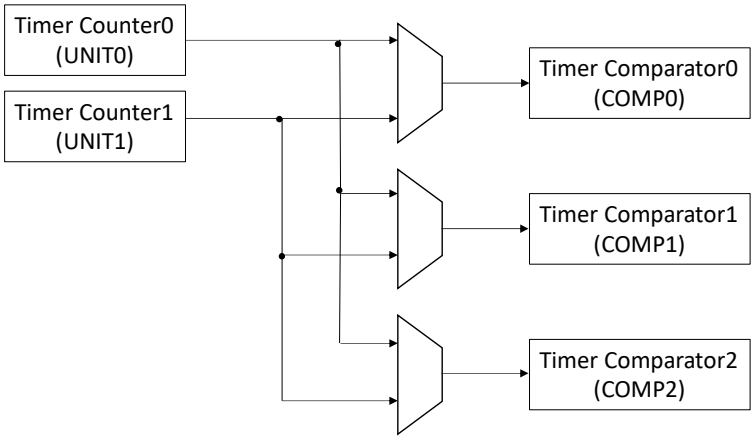


Figure 10-1. System Timer Structure

10.2 Features

The system timer has the following features:

- Two 52-bit counters and three 52-bit comparators
- Software accessing registers clocked by APB_CLK
- 40 MHz XTAL_CLK as the clock source of CNT_CLK
- 52-bit alarm values (t) and 26-bit alarm periods (δt)
- Two modes to generate alarms:
  - Target mode: only a one-time alarm is generated based on the alarm value (t)
  - Period mode: periodic alarms are generated based on the alarm period (δt)
- Three comparators generating three independent interrupts based on configured alarm value (t) or alarm period (δt)
- Able to load back sleep time recorded by RTC timer via software after Deep-sleep or Light-sleep
- Able to stall or continue running when CPU stalls or enters on-chip-debugging mode
- CNT_CLK used for counting, with an average frequency of 16 MHz in two counting cycles

10.3 Clock Source Selection

The counters and comparators are driven using XTAL_CLK. After scaled by a fractional divider, a $f_{XTAL\_CLK}/3$ clock is generated in one count cycle and a $f_{XTAL\_CLK}/2$ clock in another count cycle. The average clock frequency is $f_{XTAL\_CLK}/2.5$, which is 16 MHz, i.e. the CNT_CLK in Figure 10-2. The timer counting is incremented by 1/16 μs on each CNT_CLK cycle.

Software operation such as configuring registers is clocked by APB_CLK. For more information about APB_CLK, see Chapter 6 *Reset and Clock*.

The following two bits of system registers are also used to control the system timer:

- SYSTEM_SYSTIMER_CLK_EN in register SYSTEM_PERIP_CLK_EN0_REG: enable APB_CLK signal to system timer.
- SYSTEM_SYSTIMER_RST in register SYSTEM_PERIP_RST_EN0_REG: reset system timer.

Note that if the timer is reset, its registers will be restored to their default values. For more information, please refer to Table Peripheral Clock Gating and Reset in Chapter 13 *System Registers (SYSTEM)*.

10.4 Functional Description

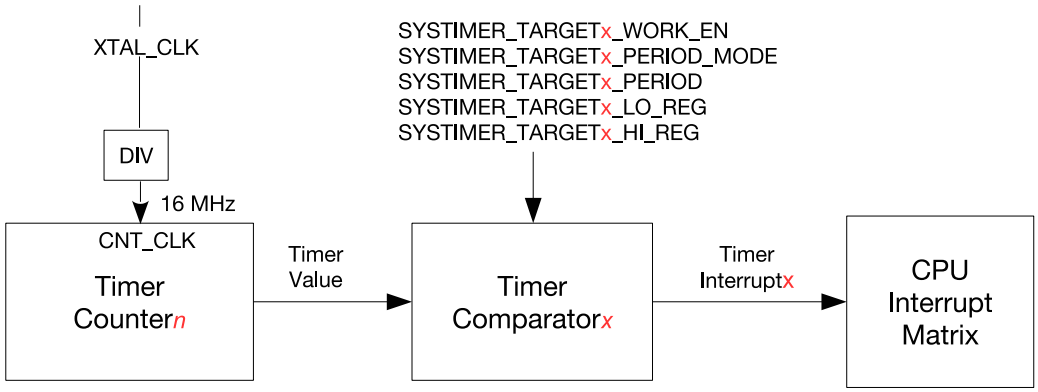


Figure 10-2. System Timer Alarm Generation

Figure 10-2 shows the procedure to generate alarm in system timer. In this process, one timer counter and one timer comparator are used. An alarm interrupt will be generated accordingly based on the comparison result in comparator.

10.4.1 Counter

The system timer has two 52-bit timer counters, shown as UNIT*n* (*n* = 0 or 1). Their counting clock source is a 16 MHz clock, i.e. CNT_CLK. Whether UNIT*n* works or not is controlled by two bits in register SYSTIMER_CONF_REG:

- SYSTIMER_TIMER_UNIT*n*_WORK_EN: set this bit to enable the counter UNIT*n* in system timer.
- SYSTIMER_TIMER_UNIT*n*_CORE0_STALL_EN: if this bit is set, the counter UNIT*n* stops when CPU is stalled. The counter continues its counting after the CPU resumes.

The configuration of the two bits to control the counter UNIT*n* is shown below, assuming that CPU is stalled.

Table 10-1. UNIT*n* Configuration Bits

SYSTIMER_TIMER_UNIT*n*_WORK_EN	SYSTIMER_TIMER_UNIT*n*_CORE0_STALL_EN	Counter UNIT*n*
0	x *	Not at work
1	1	Stop counting, but will continue its counting after the CPU resumes.
1	0	Keep counting

* x: Don't-care.

When the counter UNIT*n* is at work, the count value is incremented on each counting cycle. When the counter UNIT*n* is stopped, the count value stops increasing and keeps unchanged.

The lower 32 and higher 20 bits of initial count value are loaded from the registers SYSTIMER_TIMER_UNIT*n*_LOAD_LO and SYSTIMER_TIMER_UNIT*n*_LOAD_HI. Writing 1 to the bit SYSTIMER_TIMER_UNIT*n*_LOAD will trigger a reload event, and the current count value will be changed immediately. If UNIT*n* is at work, the counter will continue to count up from the new reloaded value.

Writing 1 to SYSTIMER_TIMER_UNIT*n*_UPDATE will trigger an update event. The lower 32 and higher 20 bits of current count value will be locked into the registers SYSTIMER_TIMER_UNIT*n*_VALUE_LO and SYSTIMER_TIMER_UNIT*n*_VALUE_HI, and then SYSTIMER_TIMER_UNIT*n*_VALUE_VALID is asserted. Before the next update event, the values of SYSTIMER_TIMER_UNIT*n*_VALUE_LO and SYSTIMER_TIMER_UNIT*n*_VALUE_HI remain unchanged.

10.4.2 Comparator and Alarm

The system timer has three 52-bit comparators, shown as COMP*x* (*x* = 0, 1, or 2). The comparators can generate independent interrupts based on different alarm values (t) or alarm periods (δt).

Configure SYSTIMER_TARGET*x*_PERIOD_MODE to choose from the two alarm modes for each COMP*x*:

- 1: period mode
- 0: target mode

In period mode, the alarm period (δt) is provided by the register SYSTIMER_TARGET*x*_PERIOD. Assuming that current count value is t1, when it reaches (t1 + δt), an alarm interrupt will be generated. Another alarm interrupt also will be generated when the counter value reaches (t1 + 2*δt). By such way, periodic alarms are generated.

In target mode, the lower 32 bits and higher 20 bits of the alarm value (t) are provided by SYSTIMER_TIMER_TARGET*x*_LO and SYSTIMER_TIMER_TARGET*x*_HI. Assuming that current count value is t2 (t2 <= t), an alarm interrupt will be generated when the count value reaches the alarm value (t). Unlike in period mode, only one alarm interrupt is generated in target mode.

SYSTIMER_TARGET*x*_TIMER_UNIT_SEL is used to choose the count value from which timer counter to be compared for alarm:

- 1: use the count value from UNIT*1*
- 0: use the count value from UNIT*0*

Finally, set SYSTIMER_TARGETx_WORK_EN and COMPx starts to compare the count value with the alarm value (t) in target mode or with the alarm period (t1 + n*δt) in period mode.

An alarm is generated when the count value equals to the alarm value (t) in target mode or to the start value + n*alarm period δt (n = 1,2,3...) in period mode. But if the alarm value (t) set in registers is less than current count value, i.e. the target has already passed, or current count value is larger than the target value (t) within a range (0 ~ 2^{51} -1), an alarm interrupt also is generated immediately. The relationship between current count value t_c, the alarm value t_t and alarm trigger point is shown below.

Table 10-2. Trigger Point

Relationship Between t_c and t_t	Trigger Point
t_c- t_t <= 0	t_c = t_t, an alarm is triggered.
0 <= t_c - t_t < 2^{51} - 1 (t_c < 2^{51} and t_t < 2^{51}, or t_c >= 2^{51} and t_t >= 2^{51})	An alarm is triggered immediately.
t_c - t_t >= 2^{51} - 1	t_c overflows after counting to its maximum value 52'hfffffffffffff, and then starts counting up from 0. When its value reaches t_t, an alarm is triggered.

10.4.3 Synchronization Operation

The clock (APB_CLK) used in software operation is not the same one as the timer counters and comparators working on CNT_CLK. Synchronization is needed for some configuration registers. A complete synchronization action takes two steps:

1. Software writes suitable values to configuration fields, see the first column in Table 10-3.
2. Software writes 1 to corresponding bits to start synchronization, see the second column in Table 10-3.

Table 10-3. Synchronization Operation

Configuration Fields	Synchronization Enable Bit
SYSTIMER_TIMER_UNITn_LOAD_LO SYSTIMER_TIMER_UNITn_LOAD_HI	SYSTIMER_TIMER_UNITn_LOAD
SYSTIMER_TARGETx_PERIOD SYSTIMER_TIMER_TARGETx_HI SYSTIMER_TIMER_TARGETx_LO	SYSTIMER_TIMER_COMPx_LOAD

10.4.4 Interrupt

Each comparator has one level-type alarm interrupt, named as SYSTIMER_TARGETx_INT. Interrupts signal is asserted high when the comparator starts to alarm. Until the interrupt is cleared by software, it remains high. To enable interrupts, set the bit SYSTIMER_TARGETx_INT_ENA.

10.5 Programming Procedure

10.5.1 Read Current Count Value

1. Set SYSTIMER_TIMER_UNITn_UPDATE to update the current count value into SYSTIMER_TIMER_UNITn_

VALUE_HI and SYSTIMER_TIMER_UNIT*n*_VALUE_LO.

2. Poll the reading of SYSTIMER_TIMER_UNIT*n*_VALUE_VALID, till it's 1, which means user now can read the count value from SYSTIMER_TIMER_UNIT*n*_VALUE_HI and SYSTIMER_TIMER_UNIT*n*_VALUE_LO.
3. Read the lower 32 bits and higher 20 bits from SYSTIMER_TIMER_UNIT*n*_VALUE_LO and SYSTIMER_TIMER_UNIT*n*_VALUE_HI.

10.5.2 Configure One-Time Alarm in Target Mode

1. Set SYSTIMER_TARGET*x*_TIMER_UNIT_SEL to select the counter (UNIT*0* or UNIT*1*) used for COMP*x*.
2. Read current count value, see Section 10.5.1. This value will be used to calculate the alarm value (t) in Step 4.
3. Clear SYSTIMER_TARGET*x*_PERIOD_MODE to enable target mode.
4. Set an alarm value (t), and fill its lower 32 bits to SYSTIMER_TIMER_TARGET*x*_LO, and the higher 20 bits to SYSTIMER_TIMER_TARGET*x*_HI.
5. Set SYSTIMER_TIMER_COMP*x*_LOAD to synchronize the alarm value (t) to COMP*x*, i.e. load the alarm value (t) to the COMP*x*.
6. Set SYSTIMER_TARGET*x*_WORK_EN to enable the selected COMP*x*. COMP*x* starts comparing the count value with the alarm value (t).
7. Set SYSTIMER_TARGET*x*_INT_ENA to enable timer interrupt. When Unit*n* counts to the alarm value (t), a SYSTIMER_TARGET*x*_INT interrupt is triggered.

10.5.3 Configure Periodic Alarms in Period Mode

1. Set SYSTIMER_TARGET*x*_TIMER_UNIT_SEL to select the counter (UNIT*0* or UNIT*1*) used for COMP*x*.
2. Set an alarm period (δt), and fill it to SYSTIMER_TARGET*x*_PERIOD.
3. Set SYSTIMER_TIMER_COMP*x*_LOAD to synchronize the alarm period (δt) to COMP*x*, i.e. load the alarm period (δt) to COMP*x*.
4. Set SYSTIMER_TARGET*x*_PERIOD_MODE to configure COMP*x* into period mode.
5. Set SYSTIMER_TARGET*x*_WORK_EN to enable the selected COMP*x*. COMP*x* starts comparing the count value with the sum of start value + n*δt (n = 1, 2, 3...).
6. Set SYSTIMER_TARGET*x*_INT_ENA to enable timer interrupt. A SYSTIMER_TARGET*x*_INT interrupt is triggered when Unit*n* counts to start value + n*δt (n = 1, 2, 3...) set in Step 2.

10.5.4 Update After Deep-sleep and Light-sleep

1. Configure RTC timer before the chip goes to Deep-sleep or Light-sleep, to record the exact sleep time. For more information, see Chapter 9 *Low-power Management (RTC_CNTL)*.
2. Read the sleep time from RTC timer when the chip is woken up from Deep-sleep or Light-sleep.
3. Read current count value of system timer, see Section 10.5.1.

4. Convert the time value recorded by RTC timer from the clock cycles based on RTC_SLOW_CLK to that based on 16 MHz CNT_CLK. For example, if the frequency of RTC_SLOW_CLK is 32 KHz, the recorded RTC timer value should be converted by multiplying by 500.
5. Add the converted RTC value to current count value of system timer:
   - Fill the new value into SYSTIMER_TIMER_UNIT*n*_LOAD_LO (low 32 bits) and SYSTIMER_TIMER_UNIT*n*_LOAD_HI (high 20 bits).
   - Set SYSTIMER_TIMER_UNIT*n*_LOAD to load new timer value into system timer. By such way, the system timer is updated.

10.6 Register Summary

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

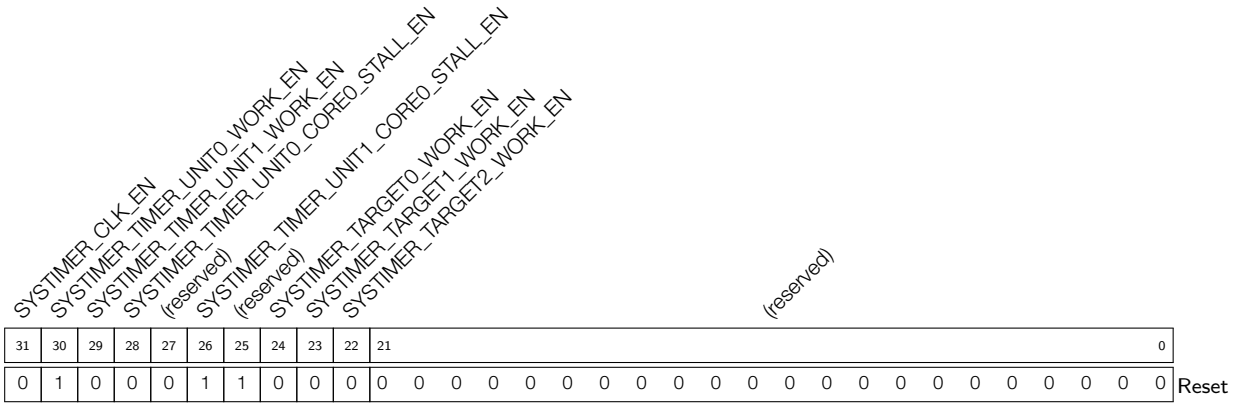
Name	Description	Address	Access
Clock Control Register			
SYSTIMER_CONF_REG	Configure system timer clock	0x0000	R/W
UNIT*0* Control and Configuration Registers			
SYSTIMER_UNIT0_OP_REG	Read UNIT0 value to registers	0x0004	varies
SYSTIMER_UNIT0_LOAD_HI_REG	High 20 bits to be loaded to UNIT0	0x000C	R/W
SYSTIMER_UNIT0_LOAD_LO_REG	Low 32 bits to be loaded to UNIT0	0x0010	R/W
SYSTIMER_UNIT0_VALUE_HI_REG	UNIT0 value, high 20 bits	0x0040	RO
SYSTIMER_UNIT0_VALUE_LO_REG	UNIT0 value, low 32 bits	0x0044	RO
SYSTIMER_UNIT0_LOAD_REG	UNIT0 synchronization register	0x005C	WT
UNIT*1* Control and Configuration Registers			
SYSTIMER_UNIT1_OP_REG	Read UNIT1 value to registers	0x0008	varies
SYSTIMER_UNIT1_LOAD_HI_REG	High 20 bits to be loaded to UNIT1	0x0014	R/W
SYSTIMER_UNIT1_LOAD_LO_REG	Low 32 bits to be loaded to UNIT1	0x0018	R/W
SYSTIMER_UNIT1_VALUE_HI_REG	UNIT1 value, high 20 bits	0x0048	RO
SYSTIMER_UNIT1_VALUE_LO_REG	UNIT1 value, low 32 bits	0x004C	RO
SYSTIMER_UNIT1_LOAD_REG	UNIT1 synchronization register	0x0060	WT
Comparator*0* Control and Configuration Registers			
SYSTIMER_TARGET0_HI_REG	Alarm value to be loaded to COMP0, high 20 bits	0x001C	R/W
SYSTIMER_TARGET0_LO_REG	Alarm value to be loaded to COMP0, low 32 bits	0x0020	R/W
SYSTIMER_TARGET0_CONF_REG	Configure COMP0 alarm mode	0x0034	R/W
SYSTIMER_COMP0_LOAD_REG	COMP0 synchronization register	0x0050	WT
Comparator*1* Control and Configuration Registers			
SYSTIMER_TARGET1_HI_REG	Alarm value to be loaded to COMP1, high 20 bits	0x0024	R/W
SYSTIMER_TARGET1_LO_REG	Alarm value to be loaded to COMP1, low 32 bits	0x0028	R/W
SYSTIMER_TARGET1_CONF_REG	Configure COMP1 alarm mode	0x0038	R/W

Name	Description	Address	Access
SYSTIMER_COMP1_LOAD_REG	COMP1 synchronization register	0x0054	WT
Comparator2 Control and Configuration Registers			
SYSTIMER_TARGET2_HI_REG	Alarm value to be loaded to COMP2, high 20 bits	0x002C	R/W
SYSTIMER_TARGET2_LO_REG	Alarm value to be loaded to COMP2, low 32 bits	0x0030	R/W
SYSTIMER_TARGET2_CONF_REG	Configure COMP2 alarm mode	0x003C	R/W
SYSTIMER_COMP2_LOAD_REG	COMP2 synchronization register	0x0058	WT
Interrupt Registers			
SYSTIMER_INT_ENA_REG	Interrupt enable register of system timer	0x0064	R/W
SYSTIMER_INT_RAW_REG	Interrupt raw register of system timer	0x0068	R/WTC/SS
SYSTIMER_INT_CLR_REG	Interrupt clear register of system timer	0x006C	WT
SYSTIMER_INT_ST_REG	Interrupt status register of system timer	0x0070	RO
Version Register			
SYSTIMER_DATE_REG	Version control register	0x00FC	R/W

10.7 Registers

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

Register 10.1. SYSTIMER_CONF_REG (0x0000)



SYSTIMER_TARGET2_WORK_EN COMP2 work enable bit. (R/W)

SYSTIMER_TARGET1_WORK_EN COMP1 work enable bit. (R/W)

SYSTIMER_TARGET0_WORK_EN COMP0 work enable bit. (R/W)

SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN UNIT1 is stalled when CPU stalled. (R/W)

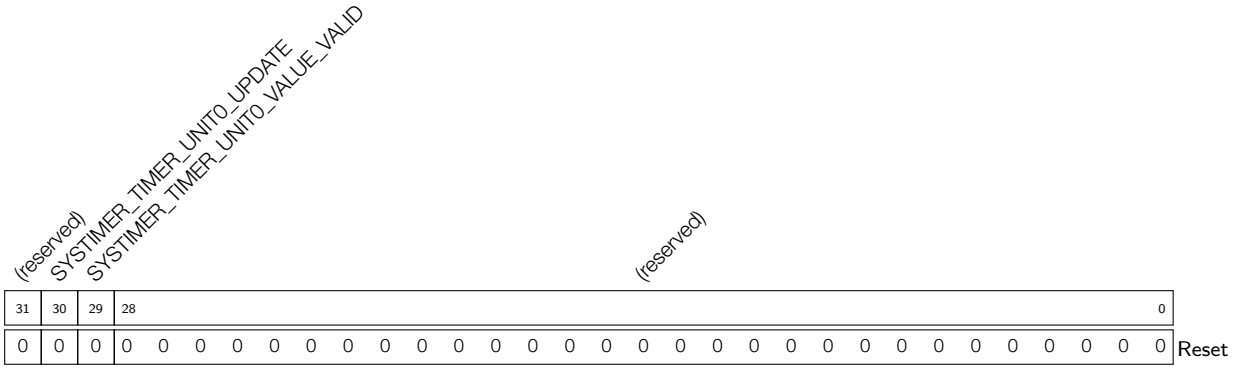
SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN UNIT0 is stalled when CPU stalled. (R/W)

SYSTIMER_TIMER_UNIT1_WORK_EN UNIT1 work enable bit. (R/W)

SYSTIMER_TIMER_UNIT0_WORK_EN UNIT0 work enable bit. (R/W)

SYSTIMER_CLK_EN Register clock gating. 1: Register clock is always enabled for read and write operations. 0: Only enable needed clock for register read or write operations. (R/W)

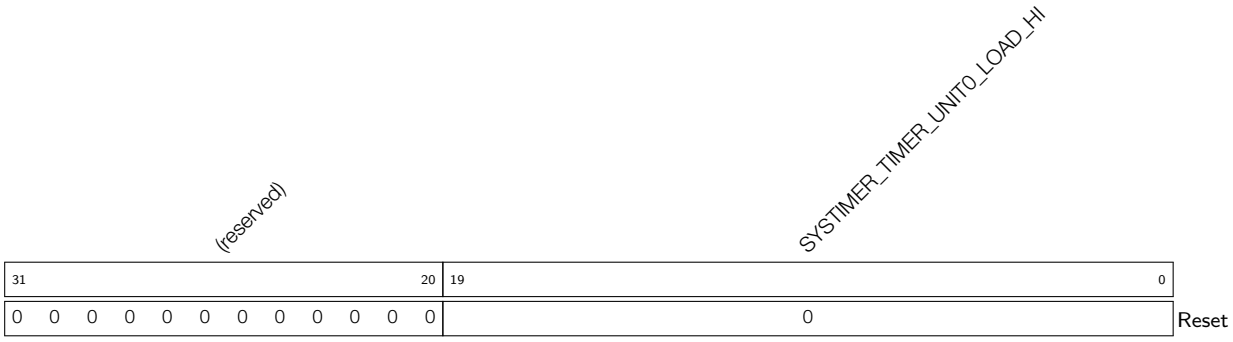
Register 10.2. SYSTIMER_UNIT0_OP_REG (0x0004)



SYSTIMER_TIMER_UNIT0_VALUE_VALID Timer value is synchronized and valid. (R/SS/WTC)

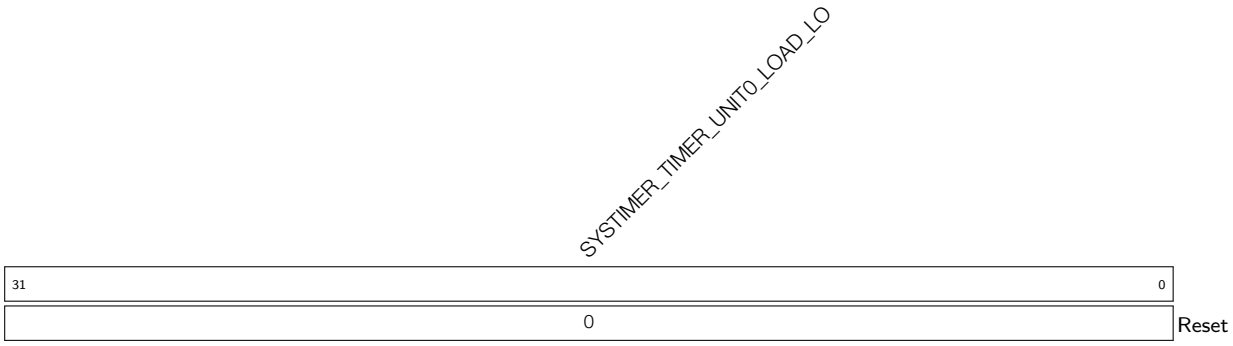
SYSTIMER_TIMER_UNIT0_UPDATE Update timer UNIT0, i.e. read the UNIT0 count value to SYSTIMER_TIMER_UNIT0_VALUE_HI and SYSTIMER_TIMER_UNIT0_VALUE_LO. (WT)

Register 10.3. SYSTIMER_UNIT0_LOAD_HI_REG (0x000C)



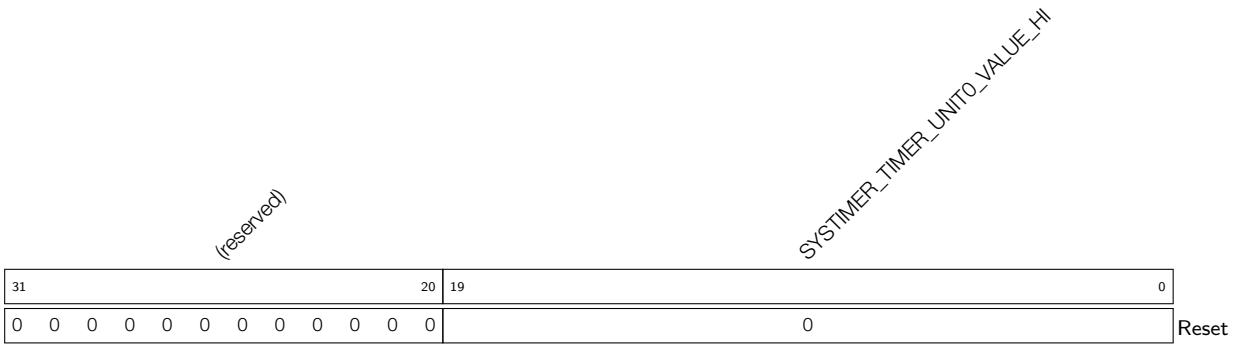
SYSTIMER_TIMER_UNIT0_LOAD_HI The value to be loaded to UNIT0, high 20 bits. (R/W)

Register 10.4. SYSTIMER_UNIT0_LOAD_LO_REG (0x0010)



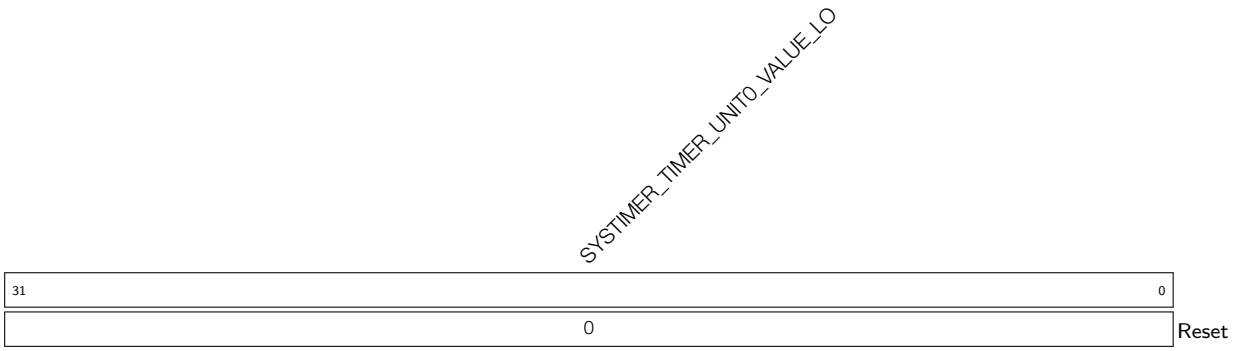
SYSTIMER_TIMER_UNIT0_LOAD_LO The value to be loaded to UNIT0, low 32 bits. (R/W)

Register 10.5. SYSTIMER_UNIT0_VALUE_HI_REG (0x0040)



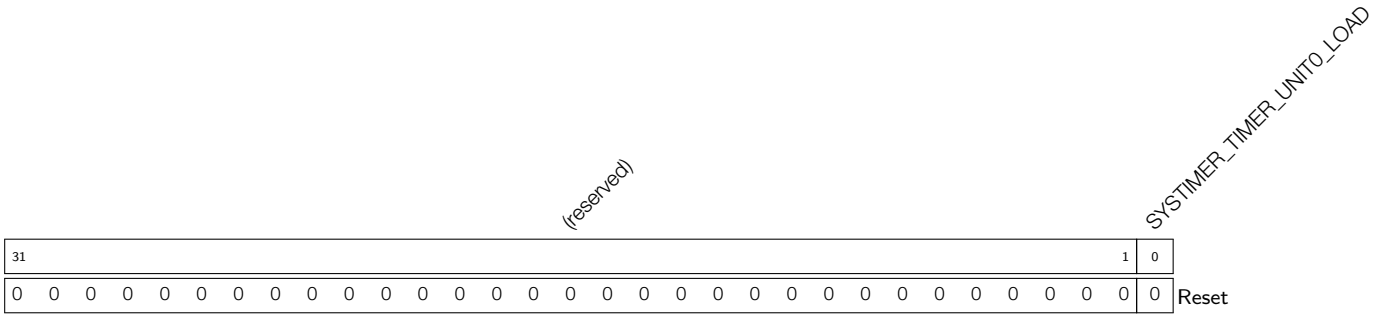
SYSTIMER_TIMER_UNIT0_VALUE_HI UNIT0 read value, high 20 bits. (RO)

Register 10.6. SYSTIMER_UNIT0_VALUE_LO_REG (0x0044)



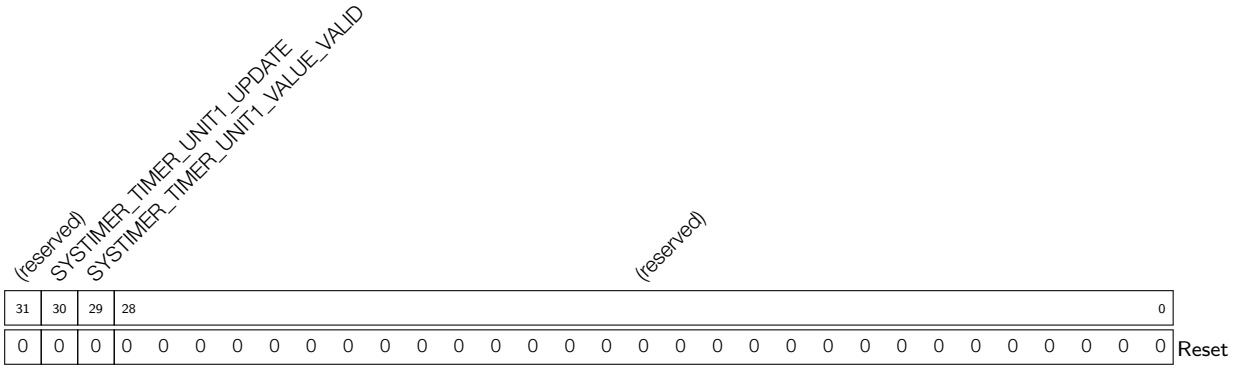
SYSTIMER_TIMER_UNIT0_VALUE_LO UNIT0 read value, low 32 bits. (RO)

Register 10.7. SYSTIMER_UNIT0_LOAD_REG (0x005C)



SYSTIMER_TIMER_UNIT0_LOAD UNIT0 synchronization enable signal. Set this bit to reload the values of SYSTIMER_TIMER_UNIT0_LOAD_HI and SYSTIMER_TIMER_UNIT0_LOAD_LO to UNIT0. (WT)

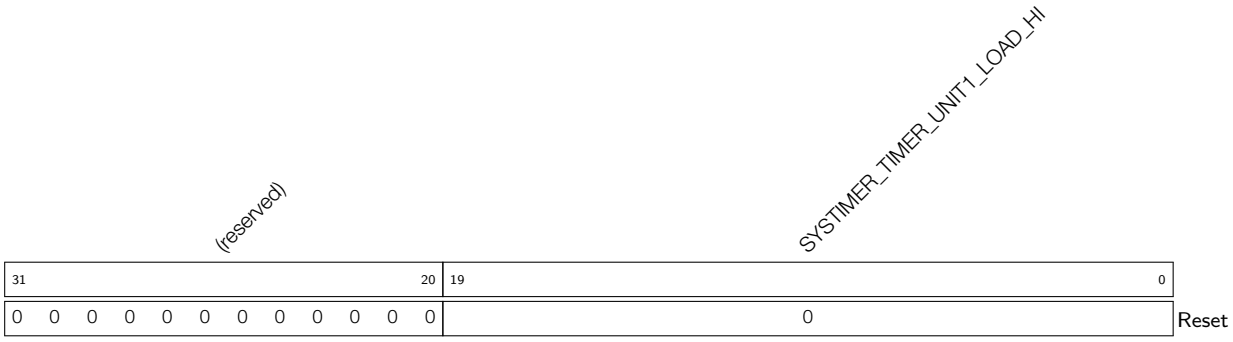
Register 10.8. SYSTIMER_UNIT1_OP_REG (0x0008)



SYSTIMER_TIMER_UNIT1_VALUE_VALID UNIT1 value is synchronized and valid. (R/SS/WTC)

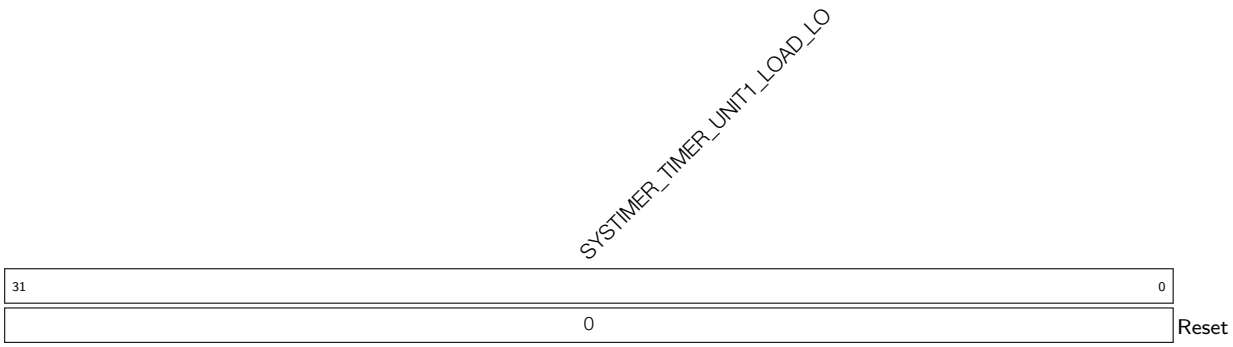
SYSTIMER_TIMER_UNIT1_UPDATE Update timer UNIT1, i.e. read the UNIT1 count value to SYSTIMER_TIMER_UNIT1_VALUE_HI and SYSTIMER_TIMER_UNIT1_VALUE_LO. (WT)

Register 10.9. SYSTIMER_UNIT1_LOAD_HI_REG (0x0014)



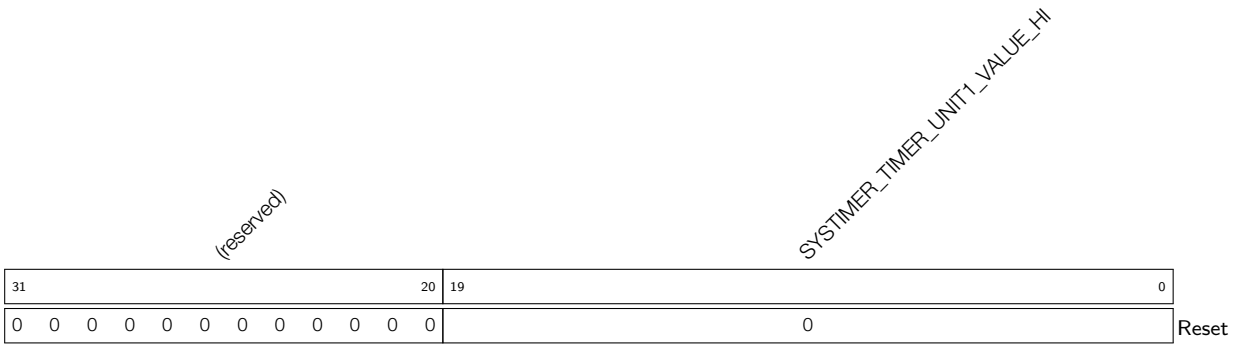
SYSTIMER_TIMER_UNIT1_LOAD_HI The value to be loaded to UNIT1, high 20 bits. (R/W)

Register 10.10. SYSTIMER_UNIT1_LOAD_LO_REG (0x0018)



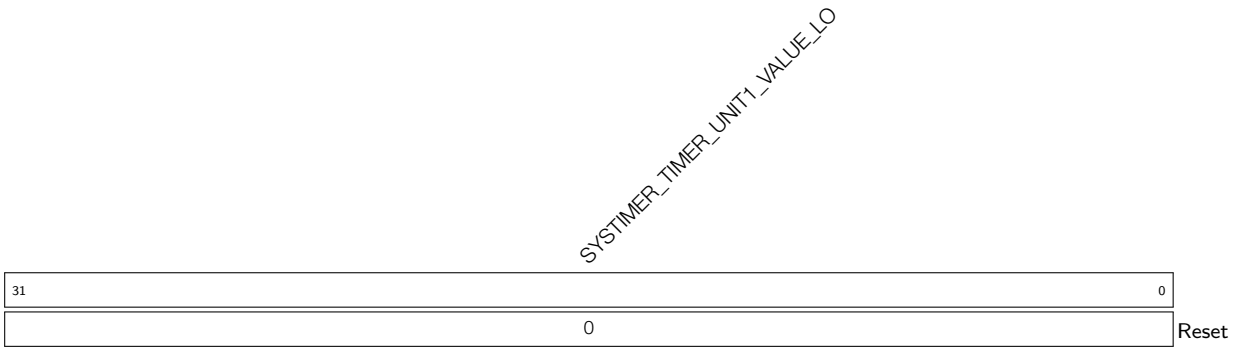
SYSTIMER_TIMER_UNIT1_LOAD_LO The value to be loaded to UNIT1, low 32 bits. (R/W)

Register 10.11. SYSTIMER_UNIT1_VALUE_HI_REG (0x0048)



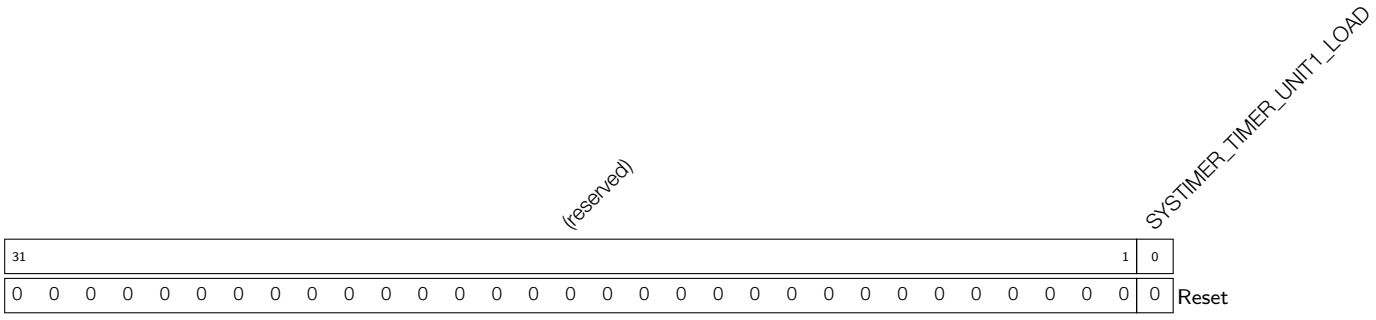
SYSTIMER_TIMER_UNIT1_VALUE_HI UNIT1 read value, high 20 bits. (RO)

Register 10.12. SYSTIMER_UNIT1_VALUE_LO_REG (0x004C)



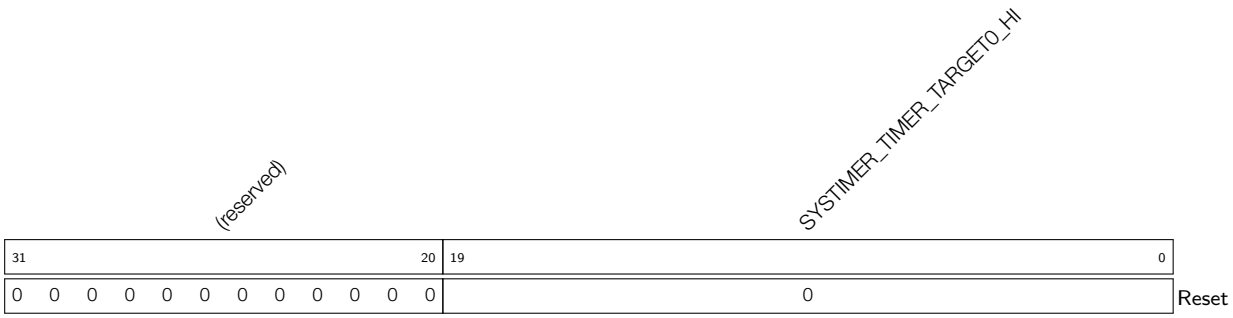
SYSTIMER_TIMER_UNIT1_VALUE_LO UNIT1 read value, low 32 bits. (RO)

Register 10.13. SYSTIMER_UNIT1_LOAD_REG (0x0060)



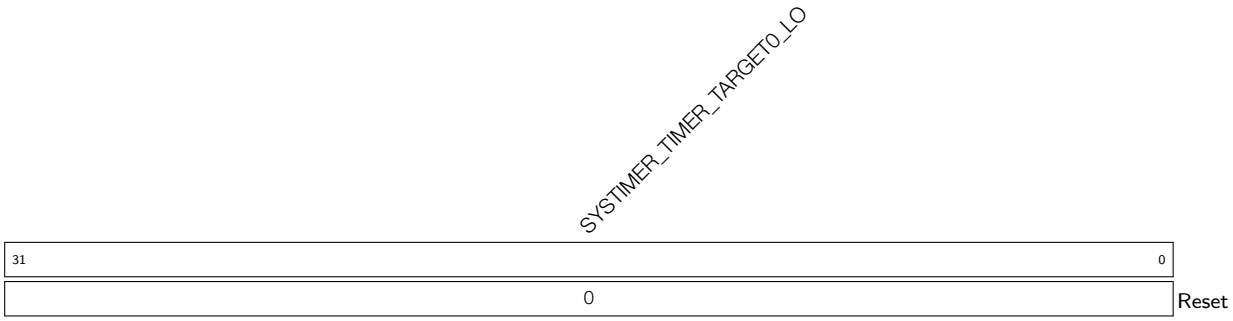
SYSTIMER_TIMER_UNIT1_LOAD UNIT1 synchronization enable signal. Set this bit to reload the values of SYSTIMER_TIMER_UNIT1_LOAD_HI and SYSTIMER_TIMER_UNIT1_LOAD_LO to UNIT1. (WT)

Register 10.14. SYSTIMER_TARGET0_HI_REG (0x001C)



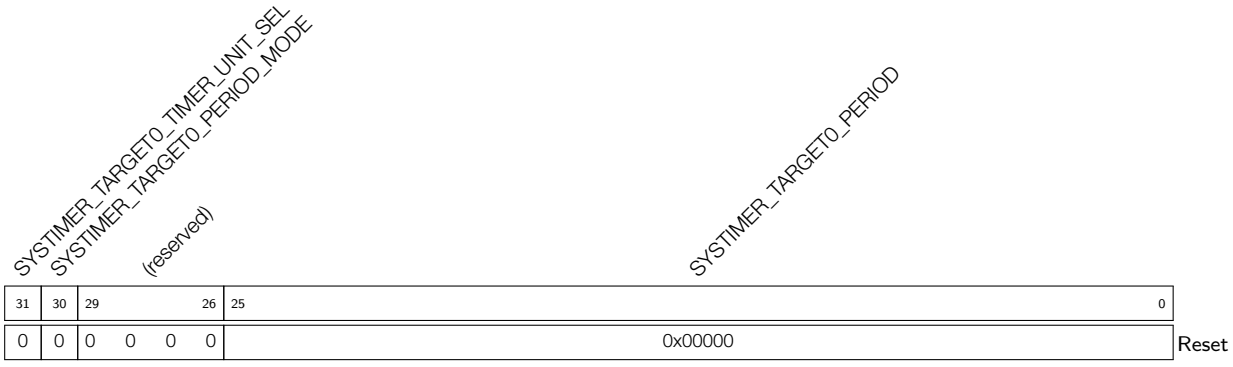
SYSTIMER_TIMER_TARGET0_HI The alarm value to be loaded to COMP0, high 20 bits. (R/W)

Register 10.15. SYSTIMER_TARGET0_LO_REG (0x0020)



SYSTIMER_TIMER_TARGET0_LO The alarm value to be loaded to COMP0, low 32 bits. (R/W)

Register 10.16. SYSTIMER_TARGET0_CONF_REG (0x0034)

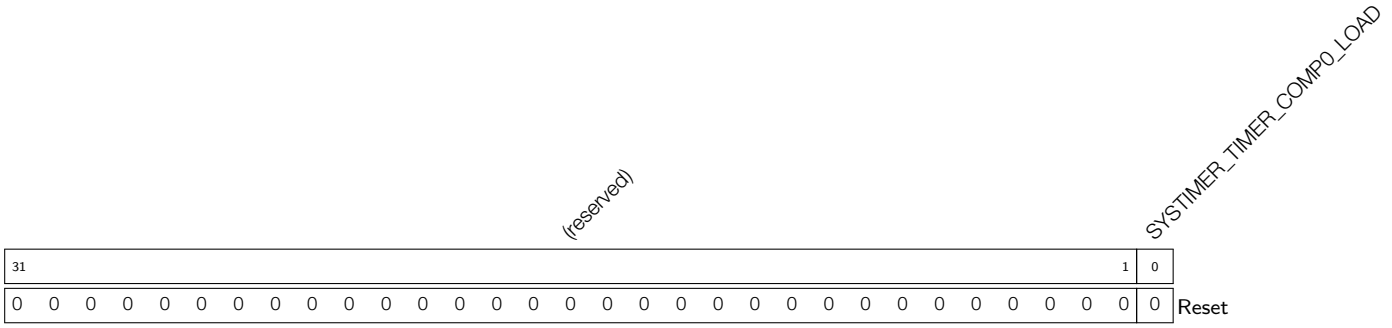


SYSTIMER_TARGET0_PERIOD COMP0 alarm period. (R/W)

SYSTIMER_TARGET0_PERIOD_MODE Set COMP0 to period mode. (R/W)

SYSTIMER_TARGET0_TIMER_UNIT_SEL Select which unit to compare for COMP0. (R/W)

Register 10.17. SYSTIMER_COMP0_LOAD_REG (0x0050)

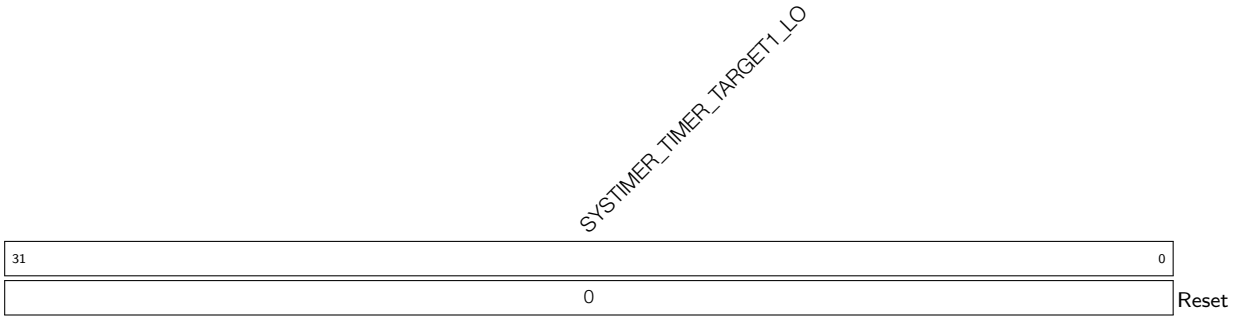


SYSTIMER_TIMER_COMP0_LOAD COMP0 synchronization enable signal. Set this bit to reload the alarm value/period to COMP0. (WT)

Register 10.18. SYSTIMER_TARGET1_HI_REG (0x0024)

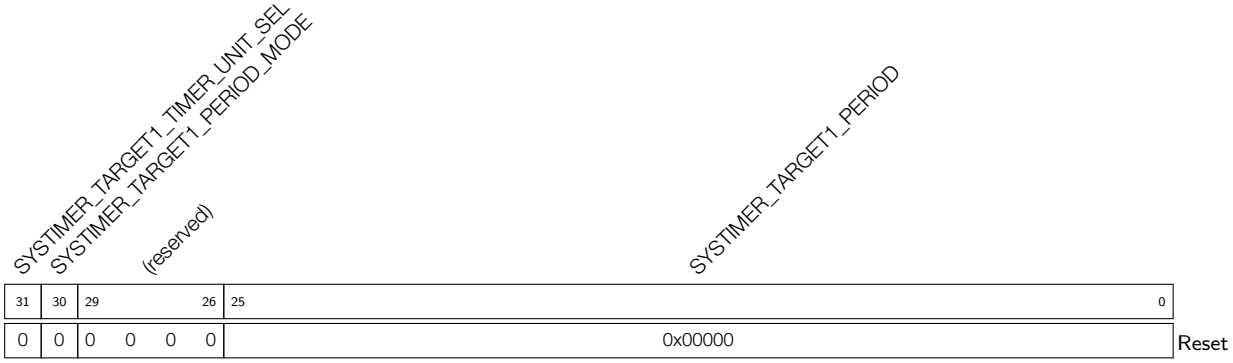
SYSTIMER_TIMER_TARGET1_HI The alarm value to be loaded to COMP1, high 20 bits. (R/W)

Register 10.19. SYSTIMER_TARGET1_LO_REG (0x0028)



SYSTIMER_TIMER_TARGET1_LO The alarm value to be loaded to COMP1, low 32 bits. (R/W)

Register 10.20. SYSTIMER_TARGET1_CONF_REG (0x0038)

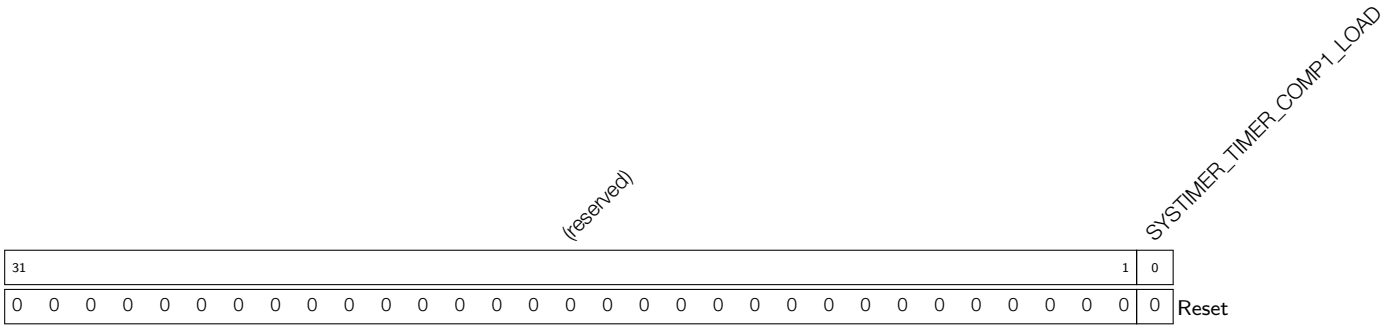


SYSTIMER_TARGET1_PERIOD COMP1 alarm period. (R/W)

SYSTIMER_TARGET1_PERIOD_MODE Set COMP1 to period mode. (R/W)

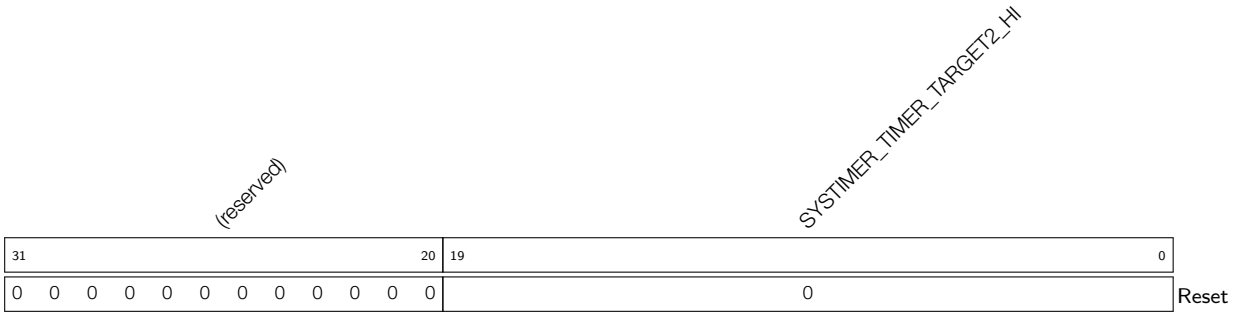
SYSTIMER_TARGET1_TIMER_UNIT_SEL Select which unit to compare for COMP1. (R/W)

Register 10.21. SYSTIMER_COMP1_LOAD_REG (0x0054)



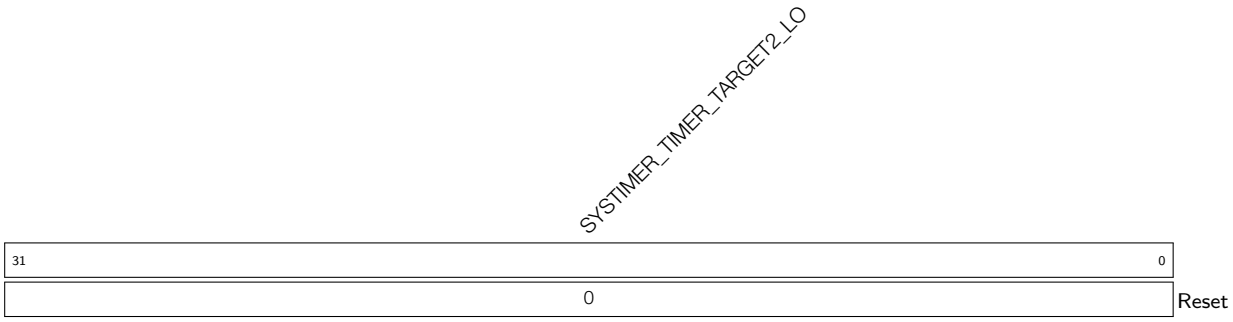
SYSTIMER_TIMER_COMP1_LOAD COMP1 synchronization enable signal. Set this bit to reload the alarm value/period to COMP1. (WT)

Register 10.22. SYSTIMER_TARGET2_HI_REG (0x002C)



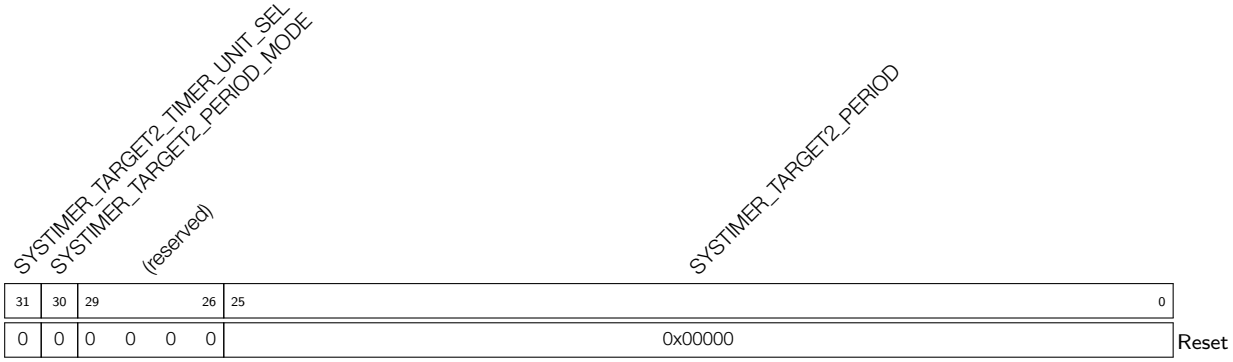
SYSTIMER_TIMER_TARGET2_HI The alarm value to be loaded to COMP2, high 20 bits. (R/W)

Register 10.23. SYSTIMER_TARGET2_LO_REG (0x0030)



SYSTIMER_TIMER_TARGET2_LO The alarm value to be loaded to COMP2, low 32 bits. (R/W)

Register 10.24. SYSTIMER_TARGET2_CONF_REG (0x003C)

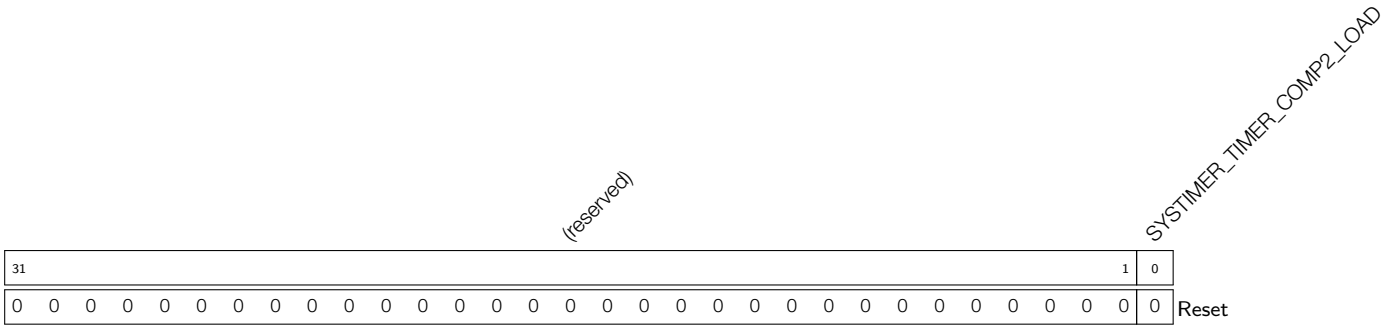


SYSTIMER_TARGET2_PERIOD COMP2 alarm period. (R/W)

SYSTIMER_TARGET2_PERIOD_MODE Set COMP2 to period mode. (R/W)

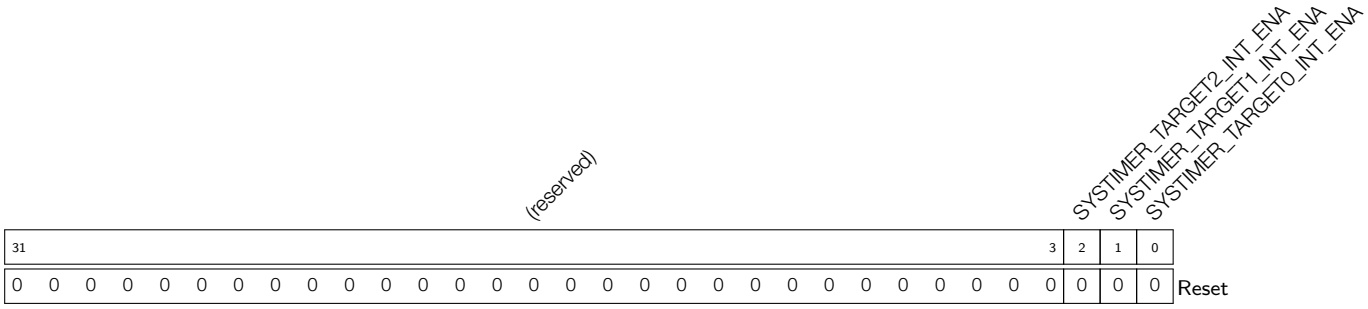
SYSTIMER_TARGET2_TIMER_UNIT_SEL Select which unit to compare for COMP2. (R/W)

Register 10.25. SYSTIMER_COMP2_LOAD_REG (0x0058)



SYSTIMER_TIMER_COMP2_LOAD COMP2 synchronization enable signal. Set this bit to reload the alarm value/period to COMP2. (WT)

Register 10.26. SYSTIMER_INT_ENA_REG (0x0064)

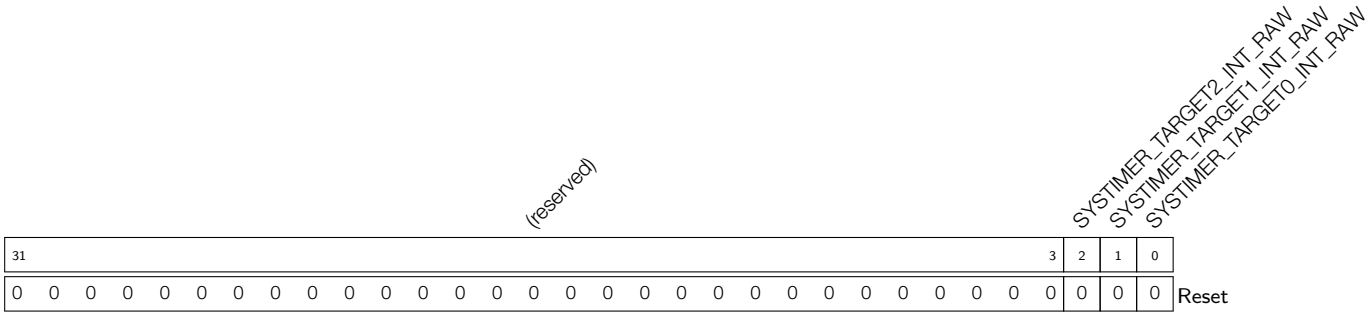


SYSTIMER_TARGET0_INT_ENA SYSTIMER_TARGET0_INT enable bit. (R/W)

SYSTIMER_TARGET1_INT_ENA SYSTIMER_TARGET1_INT enable bit. (R/W)

SYSTIMER_TARGET2_INT_ENA SYSTIMER_TARGET2_INT enable bit. (R/W)

Register 10.27. SYSTIMER_INT_RAW_REG (0x0068)

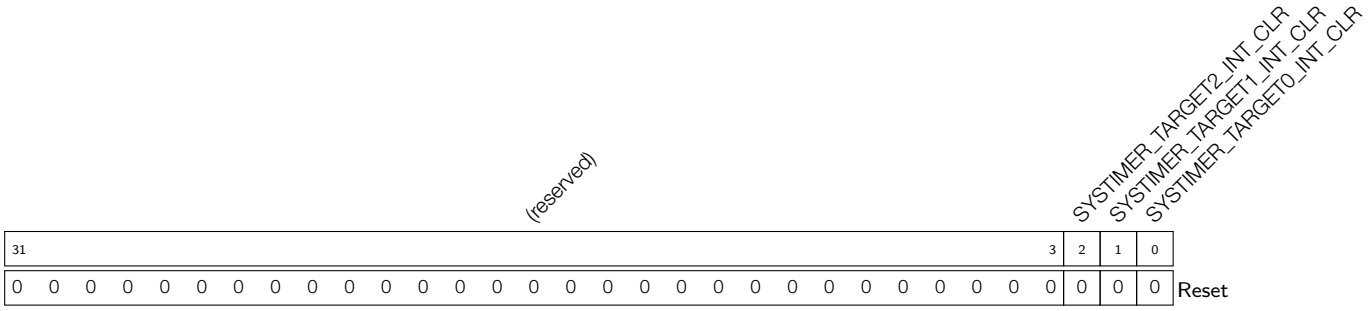


SYSTIMER_TARGET0_INT_RAW SYSTIMER_TARGET0_INT raw bit. (R/WTC/SS)

SYSTIMER_TARGET1_INT_RAW SYSTIMER_TARGET1_INT raw bit. (R/WTC/SS)

SYSTIMER_TARGET2_INT_RAW SYSTIMER_TARGET2_INT raw bit. (R/WTC/SS)

Register 10.28. SYSTIMER_INT_CLR_REG (0x006C)

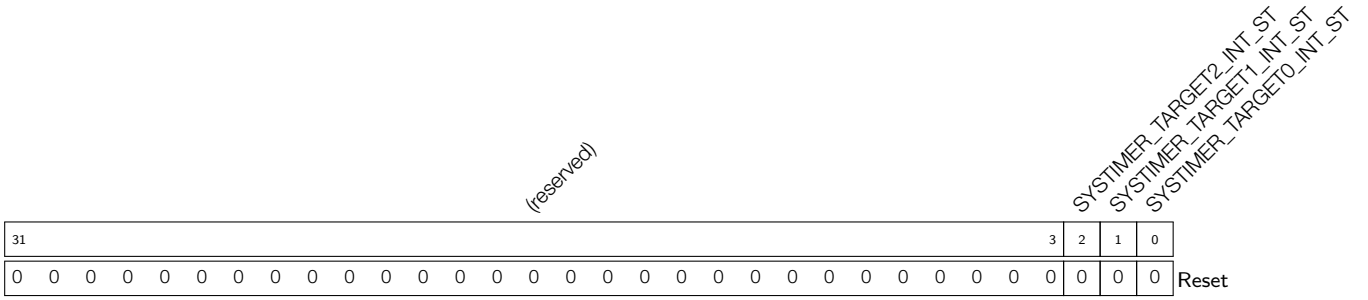


SYSTIMER_TARGET0_INT_CLR SYSTIMER_TARGET0_INT clear bit. (WT)

SYSTIMER_TARGET1_INT_CLR SYSTIMER_TARGET1_INT clear bit. (WT)

SYSTIMER_TARGET2_INT_CLR SYSTIMER_TARGET2_INT clear bit. (WT)

Register 10.29. SYSTIMER_INT_ST_REG (0x0070)

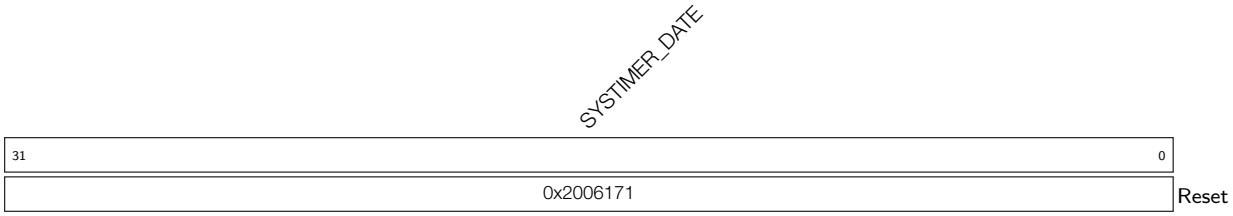


SYSTIMER_TARGET0_INT_ST SYSTIMER_TARGET0_INT status bit. (RO)

SYSTIMER_TARGET1_INT_ST SYSTIMER_TARGET1_INT status bit. (RO)

SYSTIMER_TARGET2_INT_ST SYSTIMER_TARGET2_INT status bit. (RO)

Register 10.30. SYSTIMER_DATE_REG (0x00FC)



SYSTIMER_DATE Version control register. (R/W)

11 Timer Group (TIMG)

11.1 Overview

General-purpose timers can be used to precisely time an interval, trigger an interrupt after a particular interval (periodically and aperiodically), or act as a hardware clock. As shown in Figure 11-1, the ESP8684 chip contains one timer group, namely timer group 0. The timer group consists of one general-purpose timer referred to as T0 and one Main System Watchdog Timer. The general-purpose timer is based on a 16-bit prescaler and a 54-bit auto-reload-capable up-down counter.

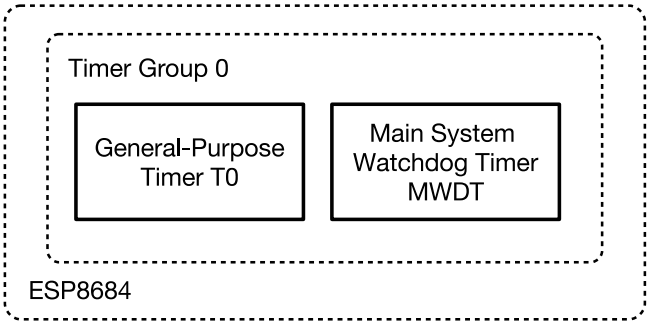


Figure 11-1. Timer Group Overview

Note that while the Main System Watchdog Timer registers are described in this chapter, their functional description is included in the Chapter 12 *Watchdog Timers (WDT)*. Therefore, the term 'timer' within this chapter refers to the general-purpose timer.

11.2 Features

The timer's features are summarized as follows:

- A 54-bit time-base counter programmable to incrementing or decrementing
- Two clock sources: 40 MHz PLL_40M_CLK or XTAL_CLK
- A 16-bit clock prescaler, from 2 to 65536
- Able to read real-time value of the time-base counter
- Able to halt and resume the time-base counter
- Programmable alarm generation
- Timer value reload — Auto-reload at alarm or software-controlled instant reload
- RTC slow clock SLOW_CLK frequency calculation
- Level interrupt generation

11.3 Functional Description

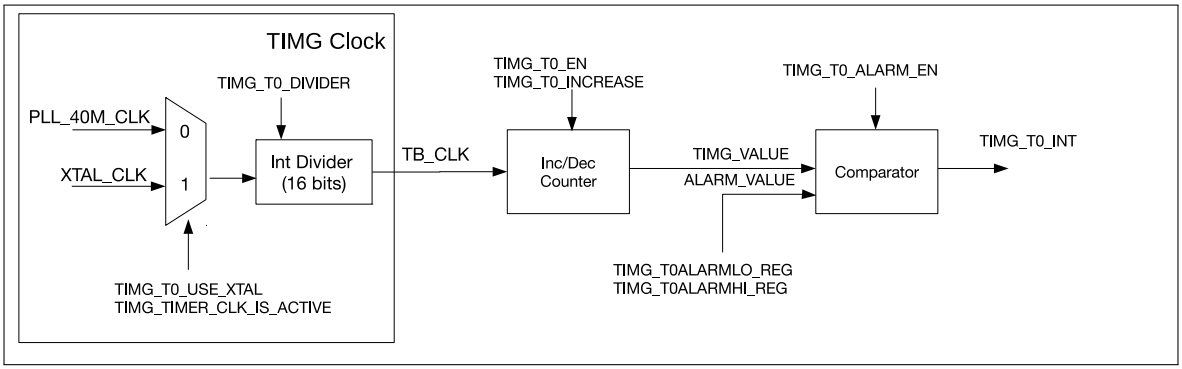


Figure 11-2. Timer Group Architecture

Figure 11-2 is a diagram of timer T0 in a timer group. T0 contains a clock selector, a 16-bit integer divider as a prescaler, a timer-based counter and a comparator for alarm generation.

11.3.1 16-bit Prescaler and Clock Selection

The timer can select between the PLL_40M_CLK clock or external clock (XTAL_CLK) as its clock source by setting the TIMG_T0_USE_XTAL field of the TIMG_T0CONFIG_REG register. Note that when the chip is in low-power mode and the clock source of CPU_CLK is not PLL_CLK (i.e. when SYSTEM_SOC_CLK_SEL is not 1, see details in Table 6-2 of Chapter 6 *Reset and Clock*), the timer can only select XTAL_CLK.

The selected clock can be switched on by setting TIMG_TIMER_CLK_IS_ACTIVE field of the TIMG_REGCLK_REG register to 1 and switched off by setting it to 0. The clock is then divided by a 16-bit prescaler to generate the time-base counter clock (TB_CLK) used by the time-base counter. When the TIMG_T0_DIVIDER field is configured as 2 ~ 65536, the divisor of the prescaler would be 2 ~ 65536. Note that programming value 0 to TIMG_T0_DIVIDER will result in the divisor being 65536. When the TIMG_T0_DIVIDER is set to 1, the actual divisor is 2 so the timer counter value represents the half of real time.

To modify the 16-bit prescaler, please first configure the TIMG_T0_DIVIDER field, and then set TIMG_T0_DIVIDER_RST to 1. Meanwhile, the timer must be disabled (i.e. TIMG_T0_EN should be cleared). Otherwise, the result can be unpredictable.

11.3.2 54-bit Time-base Counter

The 54-bit time-base counter is based on TB_CLK and can be configured to increment or decrement via the TIMG_T0_INCREASE field. The time-base counter can be enabled or disabled by setting or clearing the TIMG_T0_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_T0_INCREASE field can be changed no matter whether TIMG_T0_EN is set or not, and this will cause the time-base counter to change direction instantly.

To read the 54-bit value of the time-base counter, the timer value must be latched to two registers before being read by the CPU (due to the CPU being 32-bit). By writing any value to the TIMG_T0UPDATE_REG, the current value of the 54-bit timer is instantly latched into the TIMG_T0LO_REG and TIMG_T0HI_REG registers containing the lower 32-bits and higher 22-bits, respectively. TIMG_T0LO_REG and TIMG_T0HI_REG registers will remain unchanged for the CPU to read in its own time until TIMG_T0UPDATE_REG is written to again.

11.3.3 Alarm Generation

A timer can be configured to trigger an alarm when the timer's current value matches the alarm value. An alarm will cause an interrupt to occur and (optionally) an automatic reload of the timer's current value (see Section 11.3.4).

The 54-bit alarm value is configured using TIMG_T0ALARMLO_REG and TIMG_T0ALARMHI_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_T0_ALARM_EN field. To avoid alarm being enabled 'too late' (i.e. the timer value has already passed the alarm value when the alarm is enabled), the hardware will trigger the alarm immediately if the current timer value is:

- higher than the alarm value (within a defined range) when the up-down counter increments
- lower than the alarm value (within a defined range) when the up-down counter decrements

Table 11-1 and Table 11-2 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_T0HI_REG, TIMG_T0LO_REG}
- ALARM_VALUE = {TIMG_T0ALARMHI_REG, TIMG_T0ALARMLO_REG}

Table 11-1. Alarm Generation When Up-Down Counter Increments

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

Table 11-2. 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_T0_ALARM_EN field is automatically cleared and no alarm will occur again until the TIMG_T0_ALARM_EN is set next time.

11.3.4 Timer Reload

A timer is reloaded when a timer's current value is overwritten with a reload value stored in the TIMG_T0_LOAD_LO and TIMG_T0_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_T0_LOAD_LO and TIMG_T0_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_T0LOAD_REG, which causes the timer's current value to be instantly reloaded. If TIMG_T0_EN is set, the timer will continue incrementing or decrementing from the new value. In this case if TIMG_T0_ALARM_EN is set, the timer will still trigger alarms in scenarios listed in Table 11-1 and 11-2. If TIMG_T0_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_T0_AUTORELOAD field should be set. If not enabled, the timer's value will continue to increment or decrement past the alarm value after an alarm.

11.3.5 SLOW_CLK Frequency Calculation

Using XTAL_CLK as a reference, it is possible to calculate the frequency of clock sources for SLOW_CLK (i.e. RTC_CLK, FOSC_DIV_CLK, and XTAL32K_CLK) as follows:

1. Start periodic or one-shot frequency calculation (see Section 11.4.4 for details);
2. Once receiving the signal to start calculation, the counter of XTAL_CLK and the counter of 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: $f\_rtc = \frac{C0 \times f\_XTAL\_CLK}{C1}$

11.3.6 Interrupts

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

Interrupts are triggered after an alarm (or stage timeout for watchdog timers) occurs. Level interrupts will be held high after an alarm (or stage timeout) occurs, and will remain so until manually cleared. To enable a timer's interrupt, the TIMG_T0_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_T0_INT_RAW : An alarm event sets it to 1. The bit will remain set until the timer's corresponding bit in TIMG_T0_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_T0_INT_ST : Reflects the status of each timer's interrupt and is generated by masking the bits of TIMG_T0_INT_RAW with TIMG_T0_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_T0_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_T0_INT_CLR : Used to clear a timer's interrupt by setting its corresponding bit to 1. The timer's corresponding bit in TIMG_T0_INT_RAW and TIMG_T0_INT_ST will be cleared as a result. Note that a timer's interrupt must be cleared before the next interrupt occurs.
- TIMG_WDT_INT_CLR : Used to clear a timer's interrupt by setting its corresponding bit to 1. The watchdog timer's corresponding bit in TIMG_WDT_INT_RAW and TIMG_WDT_INT_ST will be cleared as a result. Note that a watchdog timer's interrupt must be cleared before the next interrupt occurs.

11.4 Configuration and Usage

11.4.1 Timer as a Simple Clock

1. Configure the time-base counter
   - Select clock source by setting or clearing TIMG_T0_USE_XTAL field. When CPU works in high performance mode, any value can be written to this field. When CPU works at low frequencies (i.e. when SYSTEM_SOC_CLK_SEL is not 1), this field must be set to 1.
   - Configure the 16-bit prescaler by setting TIMG_T0_DIVIDER.
   - Configure the timer direction by setting or clearing TIMG_T0_INCREASE.
   - Set the timer's starting value by writing the starting value to TIMG_T0_LOAD_LO and TIMG_T0_LOAD_HI, then reloading it into the timer by writing any value to TIMG_T0LOAD_REG.
2. Start the timer by setting TIMG_T0_EN.
3. Get the timer's current value.
   - Write any value to TIMG_T0UPDATE_REG to latch the timer's current value.
   - Read the latched timer value from TIMG_T0LO_REG and TIMG_T0HI_REG.

11.4.2 Timer as One-shot Alarm

1. Configure the time-base counter following step 1 of Section 11.4.1.
2. Configure the alarm.
   - Configure the alarm value by setting TIMG_T0ALARMLO_REG and TIMG_T0ALARMHI_REG.
   - Enable interrupt by setting TIMG_T0_INT_ENA.
3. Disable auto reload by clearing TIMG_T0_AUTORELOAD.
4. Start the alarm by setting TIMG_T0_ALARM_EN.
5. Handle the alarm interrupt.

- Clear the interrupt by setting the timer's corresponding bit in TIMG_T0_INT_CLR.
- Disable the timer by clearing TIMG_T0_EN.

11.4.3 Timer as Periodic Alarm

1. Configure the time-base counter following step 1 in Section 11.4.1.
2. Configure the alarm following step 2 in Section 11.4.2.
3. Enable auto reload by setting TIMG_T0_AUTORELOAD and configure the reload value via TIMG_T0_LOAD_LO and TIMG_T0_LOAD_HI.
4. Start the alarm by setting TIMG_T0_ALARM_EN.
5. Handle the alarm interrupt (repeat on each alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T0_INT_CLR.
   - If the next alarm requires a new alarm value and reload value (i.e. different alarm interval per iteration), then TIMG_T0ALARMLO_REG, TIMG_T0ALARMHI_REG, TIMG_T0_LOAD_LO, and TIMG_T0_LOAD_HI should be reconfigured as needed. Otherwise, the aforementioned registers should remain unchanged.
   - Re-enable the alarm by setting TIMG_T0_ALARM_EN.
6. Stop the timer (on final alarm iteration).
   - Clear the interrupt by setting the timer's corresponding bit in TIMG_T0_INT_CLR.
   - Disable the timer by clearing TIMG_T0_EN.

11.4.4 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 according to the formula in Section 11.3.5.
2. Periodic frequency calculation
   - Select the clock whose frequency is to be calculated (clock source of 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.

11.5 Register Summary

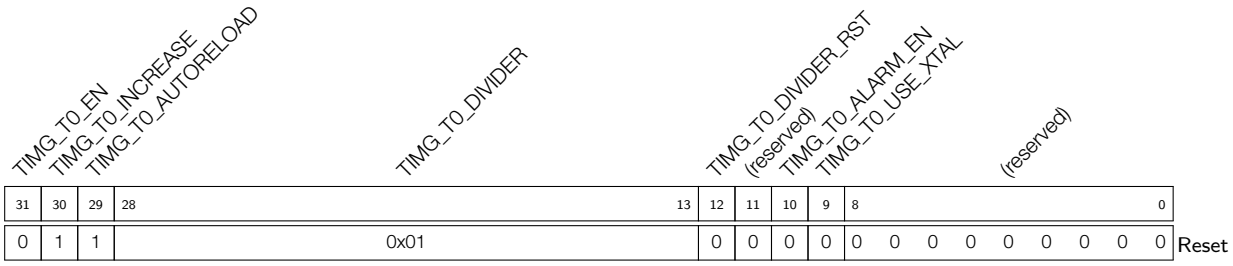
The addresses in this section are relative to Timer Group base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Name	Description	Address	Access
T0 control and configuration registers			
TIMG_T0CONFIG_REG	Timer 0 configuration register	0x0000	varies
TIMG_T0LO_REG	Timer 0 current value, low 32 bits	0x0004	RO
TIMG_T0HI_REG	Timer 0 current value, high 22 bits	0x0008	RO
TIMG_T0UPDATE_REG	Write to copy current timer value to TIMG_T0LO_REG or TIMG_T0HI_REG	0x000C	R/W/SC
TIMG_T0ALARMLO_REG	Timer 0 alarm value, low 32 bits	0x0010	R/W
TIMG_T0ALARMHI_REG	Timer 0 alarm value, high 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
WDT control and configuration registers			
TIMG_WDTCONFIG0_REG	Watchdog timer configuration register	0x0048	varies
TIMG_WDTCONFIG1_REG	Watchdog timer prescaler register	0x004C	varies
TIMG_WDTCONFIG2_REG	Watchdog timer stage 0 timeout value	0x0050	R/W
TIMG_WDTCONFIG3_REG	Watchdog timer stage 1 timeout value	0x0054	R/W
TIMG_WDTCONFIG4_REG	Watchdog timer stage 2 timeout value	0x0058	R/W
TIMG_WDTCONFIG5_REG	Watchdog timer stage 3 timeout value	0x005C	R/W
TIMG_WDTFEED_REG	Write to feed the watchdog timer	0x0060	WT
TIMG_WDTWPROTECT_REG	Watchdog write protect register	0x0064	R/W
RTC frequency calculation control and configuration registers			
TIMG_RTCCALICFG_REG	RTC frequency calculation configuration register 0	0x0068	varies
TIMG_RTCCALICFG1_REG	RTC frequency calculation configuration register 1	0x006C	RO
TIMG_RTCCALICFG2_REG	RTC frequency calculation configuration 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/SS/WTC
TIMG_INT_ST_TIMERS_REG	Masked interrupt status	0x0078	RO
TIMG_INT_CLR_TIMERS_REG	Interrupt clear bits	0x007C	WT
Version register			
TIMG_NTIMERS_DATE_REG	Timer version control register	0x00F8	R/W
Clock configuration registers			
TIMG_REGCLK_REG	Timer group clock gate register	0x00FC	R/W

11.6 Registers

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

Register 11.1. TIMG_T0CONFIG_REG (0x0000)



TIMG_T0_USE_XTAL 1: Use XTAL_CLK as the source clock of timer group. 0: Use PLL_40M_CLK as the source clock of timer group. (R/W)

TIMG_T0_ALARM_EN When set, the alarm is enabled. This bit is automatically cleared once an alarm occurs. (R/W/SC)

TIMG_T0_DIVIDER_RST When set, timer 0 's clock divider counter will be reset. (WT)

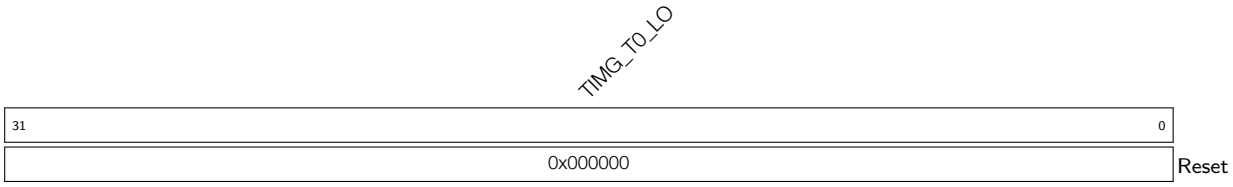
TIMG_T0_DIVIDER Timer 0 clock (T0_clk) prescaler value. (R/W)

TIMG_T0_AUTORELOAD When set, timer 0 auto-reload at alarm is enabled. (R/W)

TIMG_T0_INCREASE When set, the timer 0 time-base counter will increment every clock tick. When cleared, the timer 0 time-base counter will decrement. (R/W)

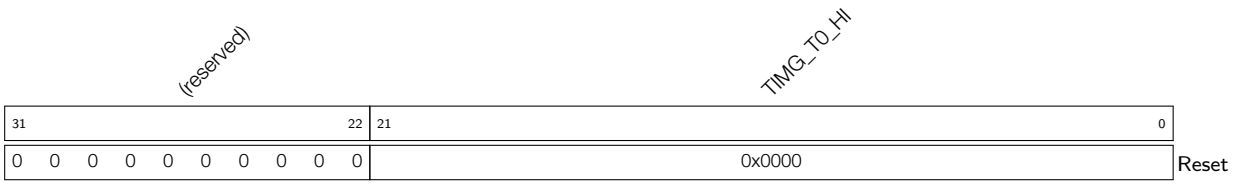
TIMG_T0_EN When set, the timer 0 time-base counter is enabled. (R/W)

Register 11.2. TIMG_T0LO_REG (0x0004)



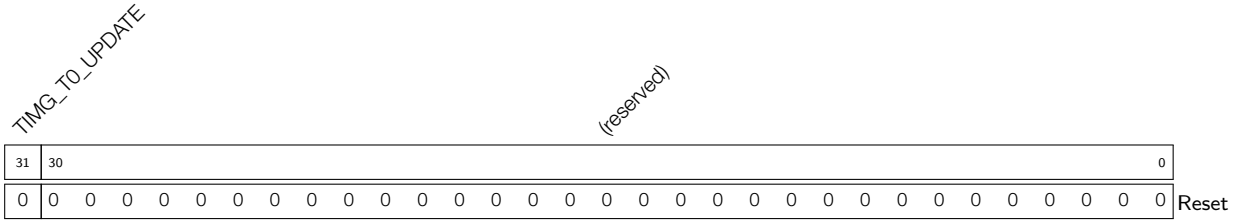
TIMG_T0_LO After writing to TIMG_T0UPDATE_REG, the low 32 bits of the time-base counter of timer 0 can be read here. (RO)

Register 11.3. TIMG_T0HI_REG (0x0008)



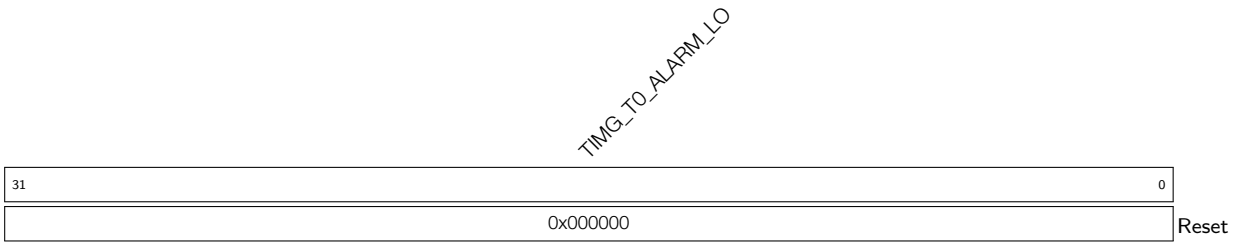
TIMG_T0_HI After writing to TIMG_T0UPDATE_REG, the high 22 bits of the time-base counter of timer 0 can be read here. (RO)

Register 11.4. TIMG_T0UPDATE_REG (0x000C)



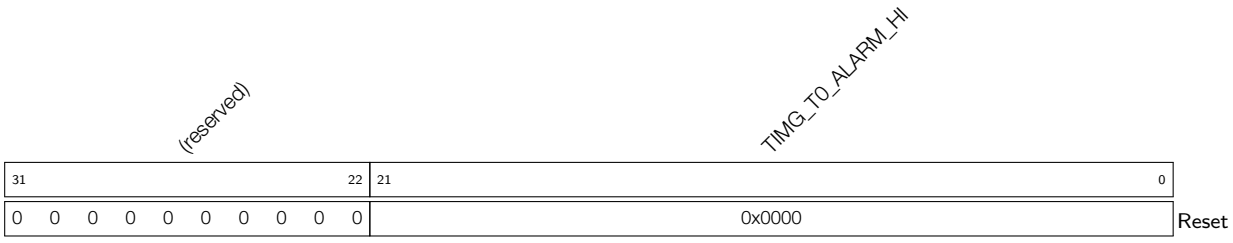
TIMG_T0_UPDATE After writing 0 or 1 to TIMG_T0UPDATE_REG, the counter value is latched. (R/W/SC)

Register 11.5. TIMG_T0ALARMLO_REG (0x0010)



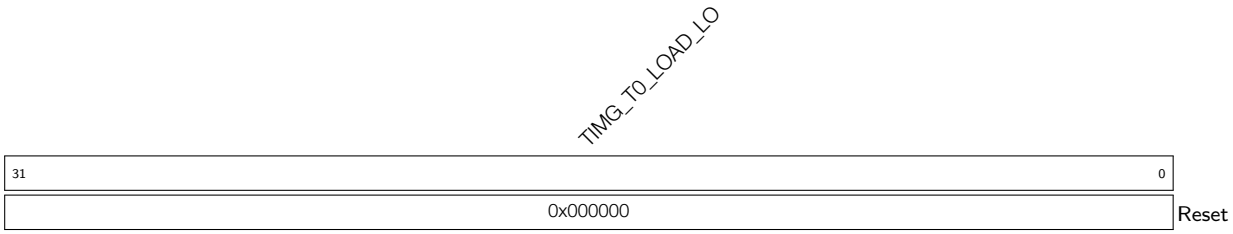
TIMG_T0_ALARM_LO Timer 0 alarm trigger time-base counter value, low 32 bits. (R/W)

Register 11.6. TIMG_T0ALARMHI_REG (0x0014)



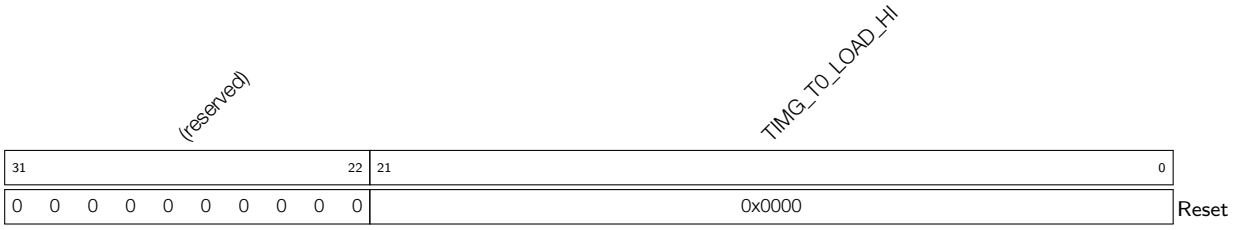
TIMG_T0_ALARM_HI Timer 0 alarm trigger time-base counter value, high 22 bits. (R/W)

Register 11.7. TIMG_T0LOADLO_REG (0x0018)



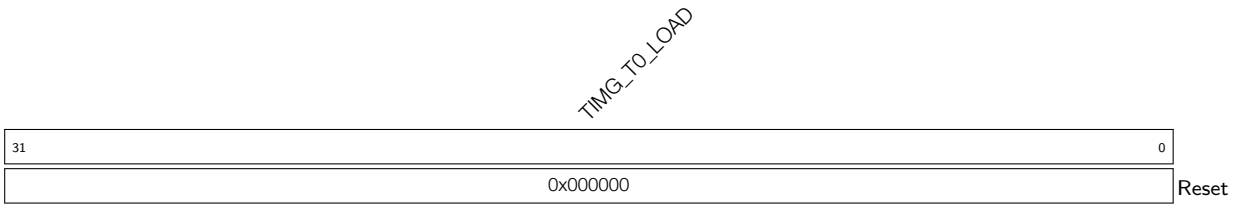
TIMG_T0_LOAD_LO Low 32 bits of the value that a reload will load onto timer 0 time-base counter. (R/W)

Register 11.8. TIMG_T0LOADHI_REG (0x001C)



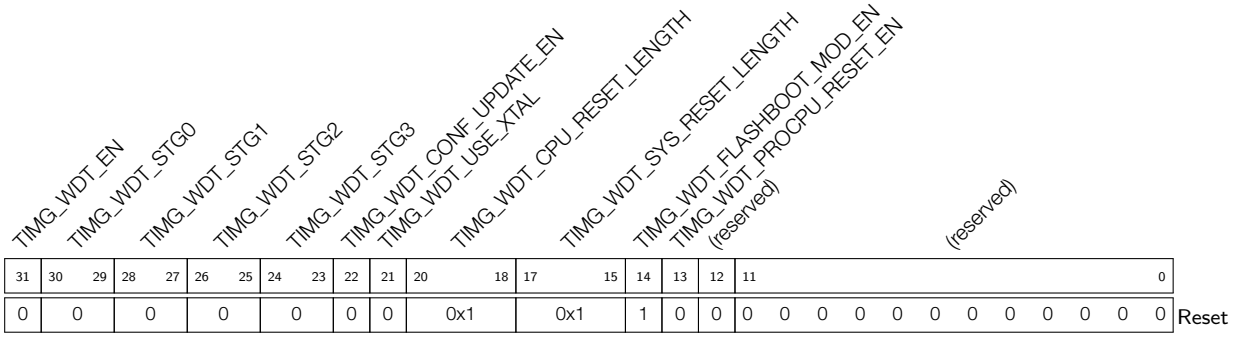
TIMG_T0_LOAD_HI High 22 bits of the value that a reload will load onto timer 0 time-base counter. (R/W)

Register 11.9. TIMG_T0LOAD_REG (0x0020)



TIMG_T0_LOAD Write any value to trigger a timer 0 time-base counter reload. (WT)

Register 11.10. TIMG_WDTCONFIG0_REG (0x0048)



TIMG_WDT_PROCPU_RESET_EN WDT reset CPU enable. (R/W)

TIMG_WDT_FLASHBOOT_MOD_EN When set, Flash boot protection is enabled. (R/W)

TIMG_WDT_SYS_RESET_LENGTH System reset signal length selection. 0: 100 ns, 1: 200 ns, 2: 300 ns, 3: 400 ns, 4: 500 ns, 5: 800 ns, 6: 1.6 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_USE_XTAL Chooses WDT clock. 0: PLL_40M_CLK, 1: XTAL_CLK. (R/W)

TIMG_WDT_CONF_UPDATE_EN Updates the WDT configuration registers. (WT)

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

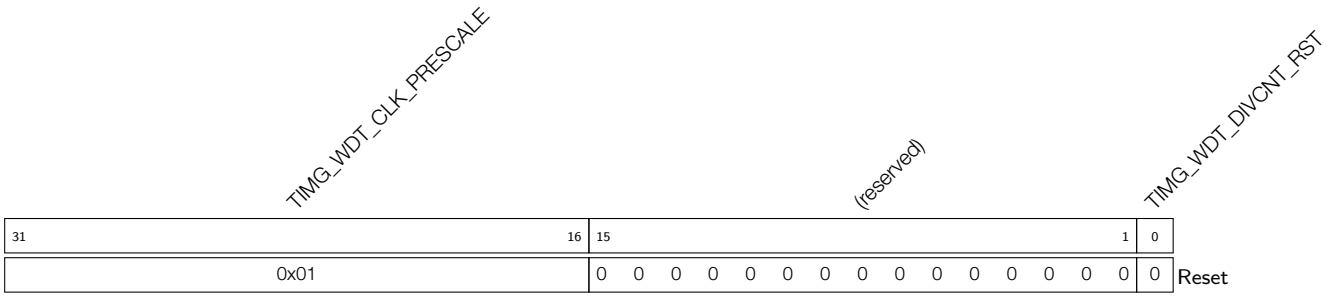
TIMG_WDT_STG2 Stage 2 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG_WDT_STG1 Stage 1 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG_WDT_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)

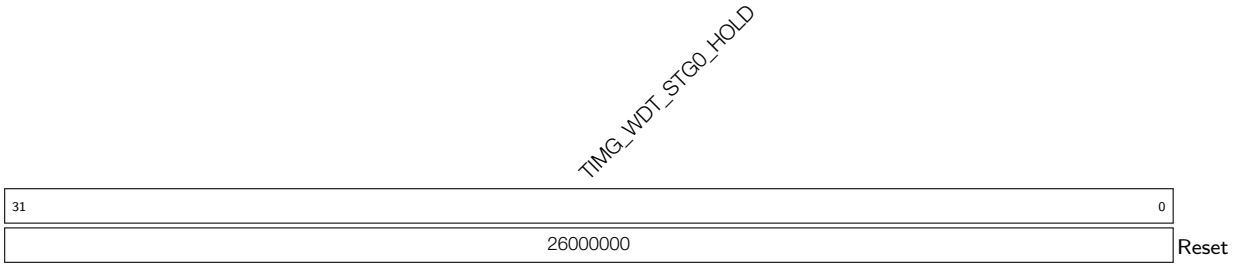
Register 11.11. TIMG_WDTCONFIG1_REG (0x004C)



TIMG_WDT_DIVCNT_RST When set, WDT 's clock divider counter will be reset. (WT)

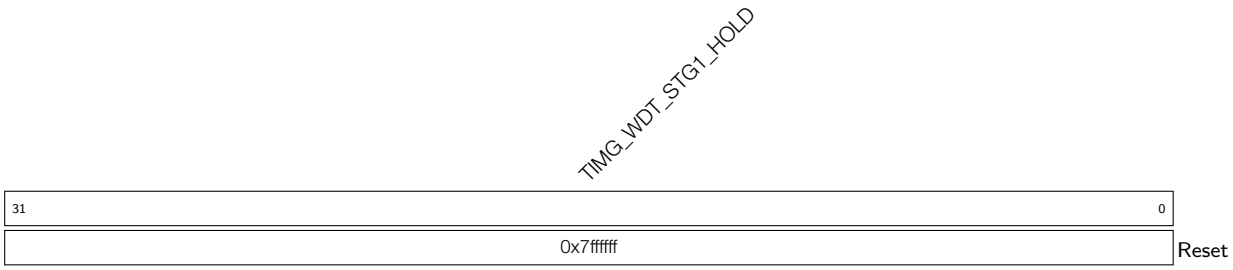
TIMG_WDT_CLK_PRESCALE MWDT clock prescaler value. MWDT clock period = 12.5 ns * TIMG_WDT_CLK_PRESCALE. (R/W)

Register 11.12. TIMG_WDTCONFIG2_REG (0x0050)



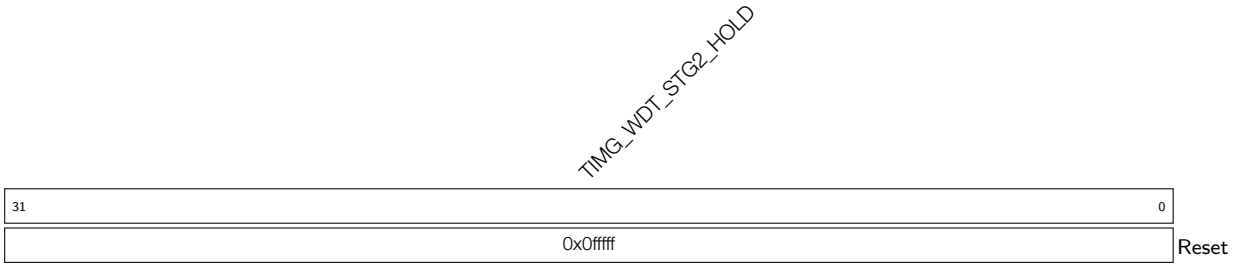
TIMG_WDT_STG0_HOLD Stage 0 timeout value, in MWDT clock cycles. (R/W)

Register 11.13. TIMG_WDTCONFIG3_REG (0x0054)



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

Register 11.14. TIMG_WDTCONFIG4_REG (0x0058)

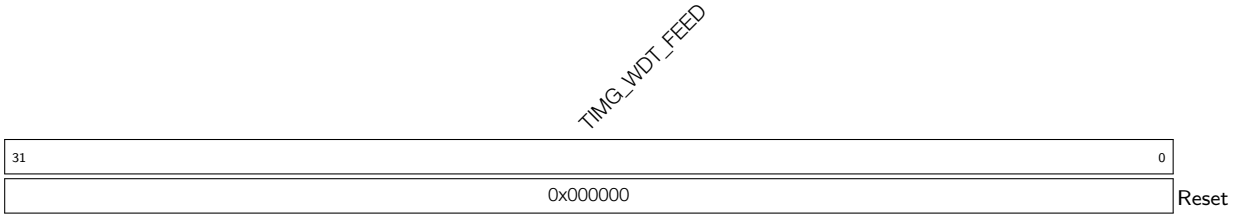


TIMG_WDT_STG2_HOLD Stage 2 timeout value, in MWDT clock cycles. (R/W)

Register 11.15. TIMG_WDTCONFIG5_REG (0x005C)

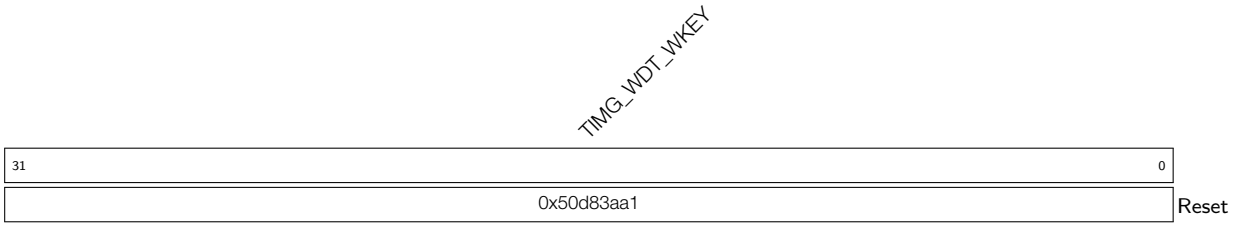
TIMG_WDT_STG3_HOLD Stage 3 timeout value, in MWDT clock cycles. (R/W)

Register 11.16. TIMG_WDTFEED_REG (0x0060)



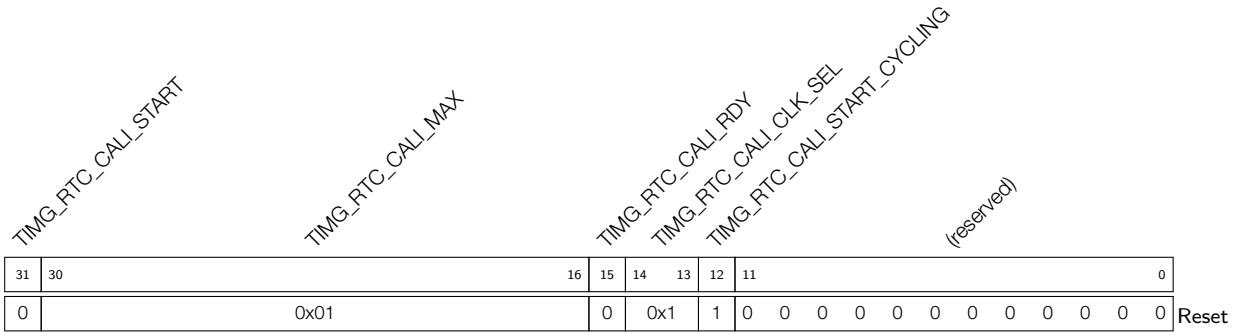
TIMG_WDT_FEED Write any value to feed the MWDT. (WO) (WT)

Register 11.17. TIMG_WDTWPROTECT_REG (0x0064)



TIMG_WDT_WKEY If the register contains a different value than its reset value, write protection is enabled. (R/W)

Register 11.18. TIMG_RTCCALICFG_REG (0x0068)



TIMG_RTC_CALI_START_CYCLING 0: one-shot frequency calculation mode, 1: periodic frequency calculation mode. (R/W)

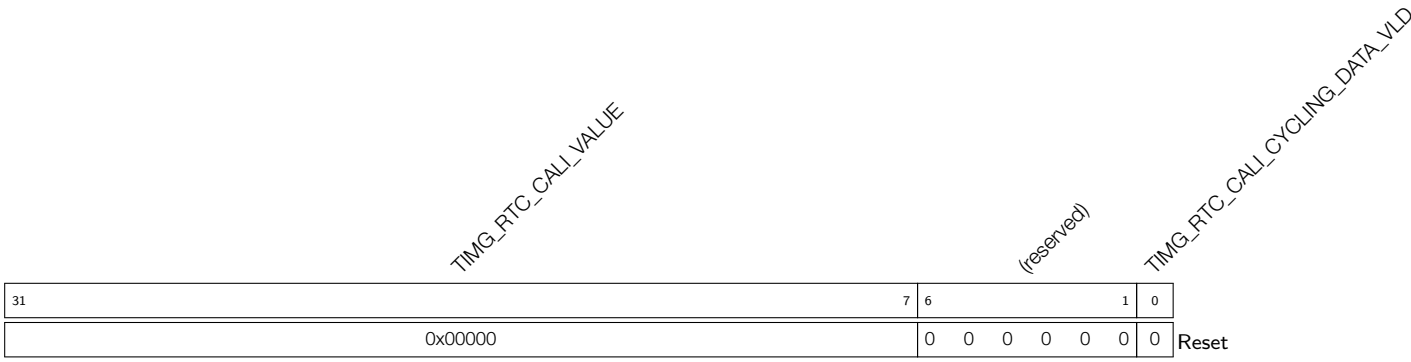
TIMG_RTC_CALI_CLK_SEL 0: RTC_CLK, 1: FOSC_DIV_CLK, 2: XTAL32K_CLK. (R/W)

TIMG_RTC_CALI_RDY Marks the completion of one-shot frequency calculation. (RO)

TIMG_RTC_CALI_MAX Configures the time to calculate the frequency of RTC slow clock SLOW_CLK. Measurement unit: SLOW_CLK cycle. (R/W)

TIMG_RTC_CALI_START Set this bit to start one-shot frequency calculation. (R/W)

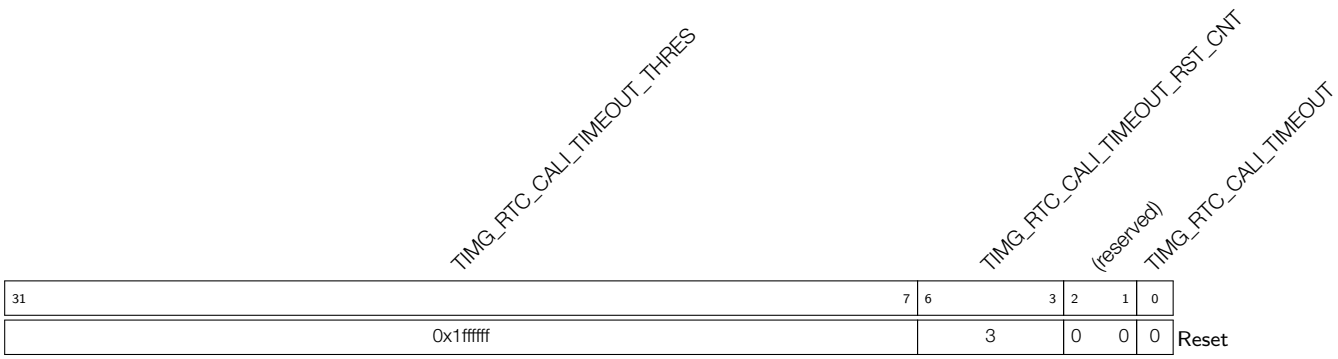
Register 11.19. TIMG_RTCCALICFG1_REG (0x006C)



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

TIMG_RTC_CALI_VALUE When one-shot or periodic frequency calculation completes, read this value to calculate the frequency of RTC slow clock SLOW_CLK. Measurement unit: XTAL_CLK cycle. (RO)

Register 11.20. TIMG_RTCCALICFG2_REG (0x0080)

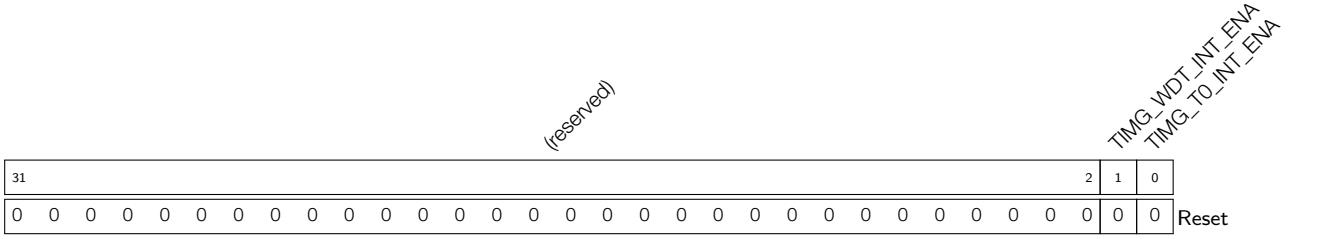


TIMG_RTC_CALI_TIMEOUT Indicates frequency calculation timeout. (RO)

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)

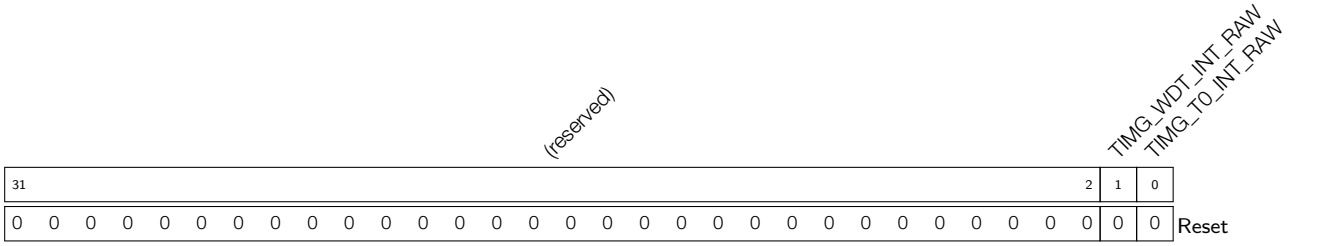
Register 11.21. TIMG_INT_ENA_TIMERS_REG (0x0070)



TIMG_T0_INT_ENA The interrupt enable bit for the TIMG_T0_INT interrupt. (R/W)

TIMG_WDT_INT_ENA The interrupt enable bit for the TIMG_WDT_INT interrupt. (R/W)

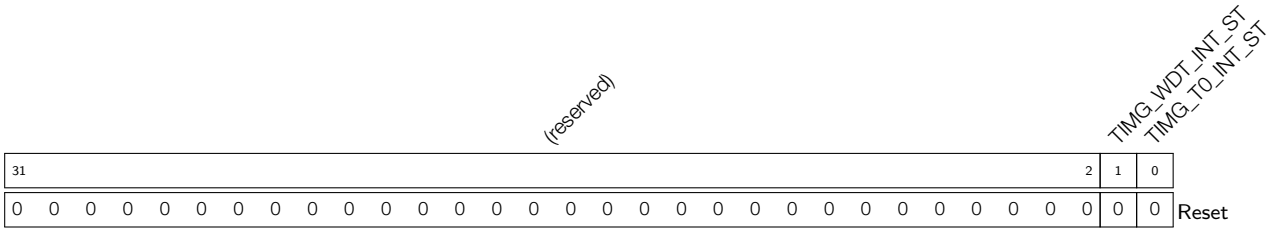
Register 11.22. TIMG_INT_RAW_TIMERS_REG (0x0074)



TIMG_T0_INT_RAW The raw interrupt status bit for the TIMG_T0_INT interrupt. (R/SS/WTC)

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

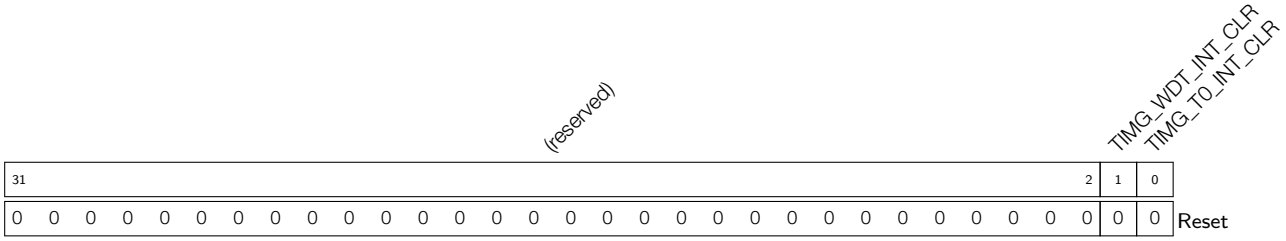
Register 11.23. TIMG_INT_ST_TIMERS_REG (0x0078)



TIMG_T0_INT_ST The masked interrupt status bit for the TIMG_T0_INT interrupt. (RO)

TIMG_WDT_INT_ST The masked interrupt status bit for the TIMG_WDT_INT interrupt. (RO)

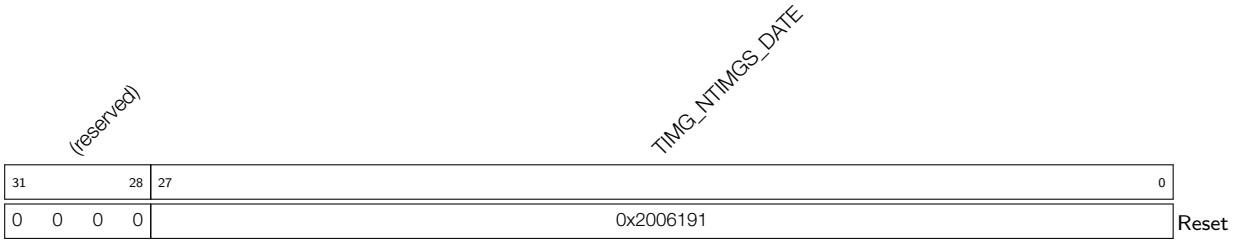
Register 11.24. TIMG_INT_CLR_TIMERS_REG (0x007C)



TIMG_T0_INT_CLR Set this bit to clear the TIMG_T0_INT interrupt. (WT)

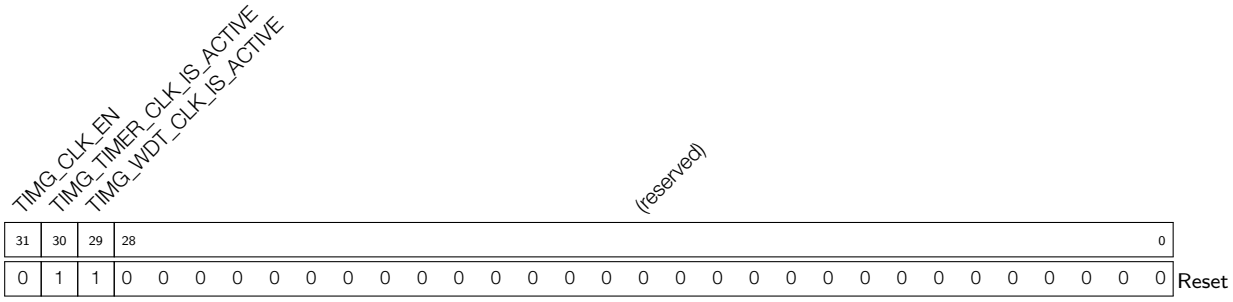
TIMG_WDT_INT_CLR Set this bit to clear the TIMG_WDT_INT interrupt. (WT)

Register 11.25. TIMG_NTIMERS_DATE_REG (0x00F8)



TIMG_NTIMGS_DATE This is the version control register. (R/W)

Register 11.26. TIMG_REGCLK_REG (0x00FC)



TIMG_WDT_CLK_IS_ACTIVE Enable WDT's clock. (R/W)

TIMG_TIMER_CLK_IS_ACTIVE Enable timer 0's clock. (R/W)

TIMG_CLK_EN Register clock gate signal. 1: Registers can be read and written to by software. 0: Registers can not be read or written to by software. (R/W)

12 Watchdog Timers (WDT)

12.1 Overview

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

As shown in Figure 12-1, ESP8684 contains two digital watchdog timers: one in the timer group in Chapter 11 *Timer Group (TIMG)* (called Main System Watchdog Timer, or MWDT) and one in the RTC Module (called the RTC Watchdog Timer, or RWDT). Each digital watchdog timer allows for four separately configurable stages and each stage can be programmed to take one action upon timeout, unless the watchdog is fed or disabled. MWDT supports three timeout actions: interrupt, CPU reset, and core reset, while RWDT supports four timeout actions: interrupt, CPU reset, core reset, and system reset (see details in Section 12.2.2.2 *Stages and Timeout Actions*). A timeout value can be set for each stage individually.

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

ESP8684 also has one analog watchdog timer: Super watchdog (SWD). It is an ultra-low-power circuit in analog domain that helps to prevent the system from operating in a sub-optimal state and resets the system if required.

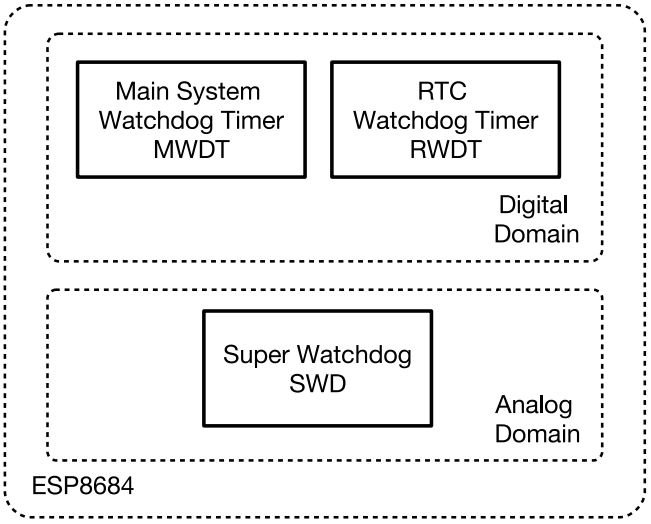


Figure 12-1. Watchdog Timers Overview

Note that while this chapter provides the functional descriptions of the watchdog timer's, their register descriptions are provided in Chapter 11 *Timer Group (TIMG)* and Chapter 9 *Low-power Management (RTC_CNTL)*.

12.2 Digital Watchdog Timers

12.2.1 Features

Watchdog timers have the following features:

- Four stages, each with a separately programmable timeout value and timeout action
- Timeout actions:
  - MWDT: interrupt, CPU reset, core reset
  - RWDT: interrupt, CPU reset, core reset, system reset
- Flash boot protection at stage 0:
  - MWDT: core reset upon timeout
  - RWDT: system reset upon timeout
- Write protection that makes WDT register read only unless unlocked
- 32-bit timeout counter
- Clock source:
  - MWDT: 40 MHz PLL_40M_CLK or XTAL_CLK
  - RWDT: RTC SLOW_CLK

12.2.2 Functional Description

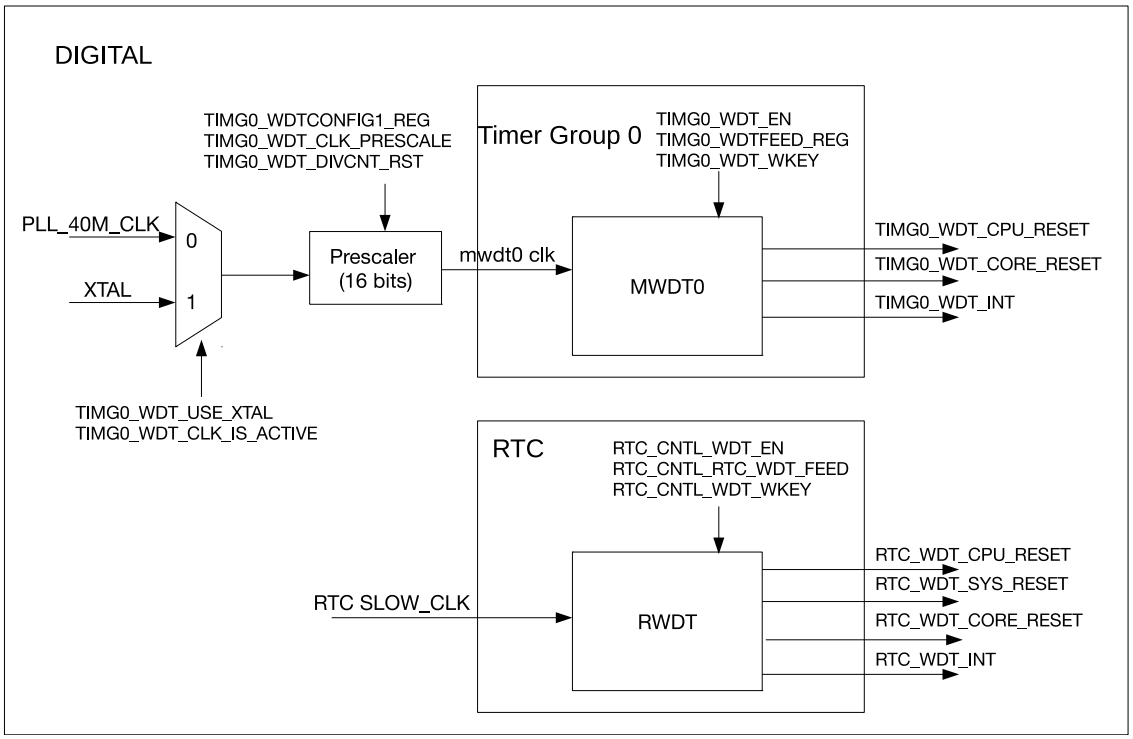


Figure 12-2. Digital Watchdog Timers in ESP8684

Figure 12-2 shows the two watchdog timers in ESP8684 digital systems.

12.2.2.1 Clock Source and 32-Bit Counter

At the core of each watchdog timer is a 32-bit counter.

MWDT can select between the PLL_40M_CLK clock or external clock (XTAL_CLK) as its clock source by setting the TIMG_WDT_USE_XTAL field of the TIMG_WDTCONFIG0_REG register. Note that when the chip is in low-power mode and the clock source of CPU_CLK is not PLL_CLK (i.e. when SYSTEM_SOC_CLK_SEL is not 1, see details in Table 6-2 of Chapter 6 *Reset and Clock*), MWDT can only select XTAL_CLK. The selected clock is switched on by setting TIMG_WDT_CLK_IS_ACTIVE field of the TIMG_REGCLK_REG register to 1 and switched off by setting it to 0. Then the selected clock is divided by a 16-bit configurable prescaler. The 16-bit prescaler for MWDT is configured via the TIMG_WDT_CLK_PRESCALE field of TIMG_WDTCONFIG1_REG. When TIMG_WDT_DIVCNT_RST field is set, the prescaler is reset and it can be re-configured at once.

In contrast, the clock source of RWDT is derived directly from RTC SLOW_CLK (see details in Chapter 6 *Reset and Clock*).

MWDT and RWDT are enabled by setting the TIMG_WDT_EN and RTC_CNTL_WDT_EN fields respectively. When enabled, the 32-bit counters of the watchdog will increment on each source clock cycle until the timeout value of the current stage is reached (i.e. timeout of the current stage). When this occurs, the current counter value is reset to zero and the next stage will become active. If a watchdog timer is fed by software, the timer will return to stage 0 and reset its counter value to zero. Software can feed a watchdog timer by writing any value to TIMG_WDTFEED_REG for MDWT and RTC_CNTL_RTC_WDT_FEED for RWDT.

12.2.2.2 Stages and Timeout Actions

Timer stages allow for a timer to have a series of different timeout values and corresponding timeout action. When one stage times out, the timeout action is triggered, the counter value is reset to zero, and the next stage becomes active. MWDT/ RWDT provide four stages (called stages 0 to 3). The watchdog timers will progress through each stage in a loop (i.e. from stage 0 to 3, then back to stage 0).

Timeout values of each stage for MWDT are configured in TIMG_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_DATA0_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 timeout of each stage, one of the following timeout actions will be executed:

Table 12-1. Timeout Actions

Timeout Action	Description
Interrupt	Trigger an interrupt
CPU reset	Reset the CPU core
Core reset	Reset the main system (which includes MWDT, CPU, and all peripherals). The power management unit and RTC peripherals will not be reset
System reset	Reset the main system, power management unit and RTC peripherals (see details in Chapter 9 *Low-power Management (RTC_CNTL)*). This action is only available in RWDT
Disabled	No effect on the system

For MWDT, the timeout action of all stages is configured in TIMG_WDTCONFIG0_REG. Likewise for RWDT, the timeout action is configured in RTC_CNTL_WDTCONFIG0_REG.

12.2.2.3 Write Protection

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

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

12.2.2.4 Flash Boot Protection

During flash booting process, MWDT as well as RWDT, are automatically enabled. Stage 0 for the enabled MWDT is automatically configured to reset the system upon timeout, known as core reset. Likewise, stage 0 for RWDT is configured to system reset, which resets the main system and RTC when it times out. After booting, TIMG_WDT_FLASHBOOT_MOD_EN and RTC_CNTL_WDT_FLASHBOOT_MOD_EN should be cleared to stop the flash boot protection procedure for both MWDT and RWDT respectively. After this, MWDT and RWDT can be configured by software.

12.3 Super Watchdog

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

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

The source of the clock for SWD is constant and can not be selected.

12.3.1 Features

SWD has the following features:

- Ultra-low power
- Interrupt to indicate that the SWD is about to time out
- Various dedicated methods for software to feed SWD, which enables SWD to monitor the working state of the whole operating system

12.3.2 Super Watchdog Controller

12.3.2.1 Structure

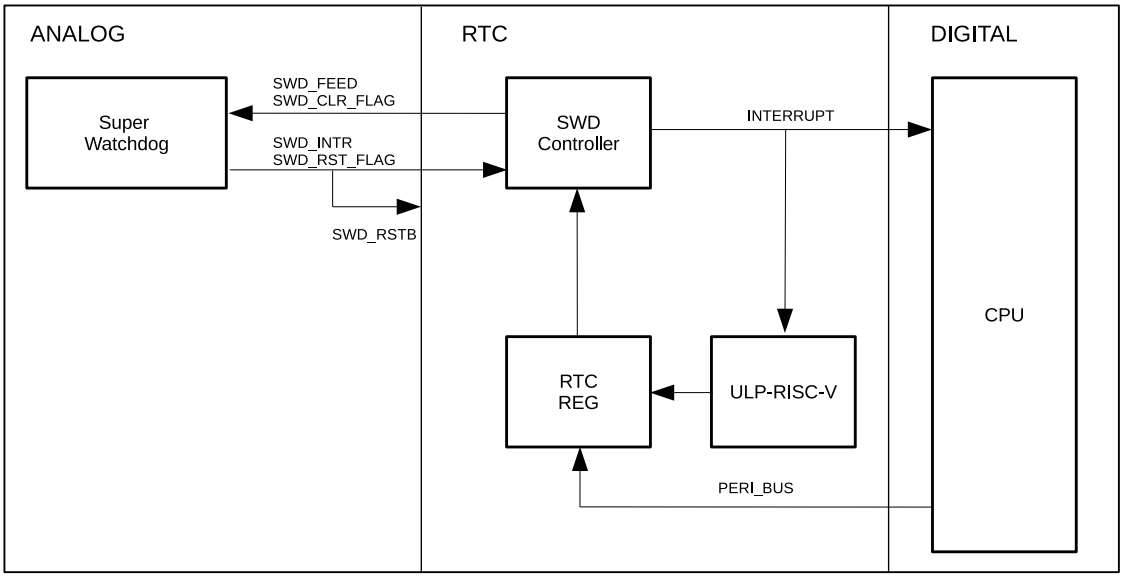


Figure 12-3. Super Watchdog Controller Structure

12.3.2.2 Workflow

In normal state:

- SWD controller receives feed request from SWD.
- SWD controller can send an interrupt to main CPU 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.

12.4 Interrupts

For watchdog timer interrupts, please refer to Section 11.3.6 *Interrupts* in Chapter 11 *Timer Group (TIMG)*.

12.5 Registers

MWDT registers are part of the timer submodule and are described in Section 11.5 *Register Summary* in Chapter 11 *Timer Group (TIMG)*. RWDT and SWD registers are part of the RTC submodule and are described in Section 9.6 *Register Summary* in Chapter 9 *Low-power Management (RTC_CNTL)*.

13 System Registers (SYSTEM)

13.1 Overview

The ESP8684 integrates a large number of peripherals, and enables the control of individual peripherals to achieve optimal characteristics in performance-vs-power-consumption scenarios. Specifically, ESP8684 has various system configuration registers that can be used for the chip's clock management (clock gating), power management, and the configuration of peripherals and core-system modules. This chapter lists all these system registers and their functions.

13.2 Features

ESP8684 system registers can be used to control the following peripheral blocks and core modules:

- System and memory
- Clock
- Software interrupts
- Peripheral clock gating and reset

13.3 Function Description

13.3.1 System and Memory Registers

13.3.1.1 Internal Memory

The following registers can be used to control ESP8684's internal memory:

- In register SYSCON_CLKGATE_FORCE_ON_REG:
  - Setting different bits of the SYSCON_ROM_CLKGATE_FORCE_ON field forces on the clock gates of different blocks of Internal ROM 0 and Internal ROM 1.
  - Setting different bits of the SYSCON_SRAM_CLKGATE_FORCE_ON field forces on the clock gates of different blocks of Internal SRAM.
  - This means when the respective bits of this register are set to 1, the clock gate of the corresponding ROM or SRAM blocks will always be on. Otherwise, the clock gate will turn on automatically when the corresponding ROM or SRAM blocks are accessed and turn off automatically when the corresponding ROM or SRAM blocks are not accessed. Therefore, it's recommended to configure these bits to 0 to lower power consumption.
- In register SYSCON_MEM_POWER_DOWN_REG:
  - Setting different bits of the SYSCON_ROM_POWER_DOWN field sends different blocks of Internal ROM 0 and Internal ROM 1 into retention state.
  - Setting different bits of the SYSCON_SRAM_POWER_DOWN field sends different blocks of Internal SRAM into retention state.
  - The "Retention" state is a low-power state of a memory block. In this state, the memory block still holds all the data stored but cannot be accessed, thus reducing the power consumption. Therefore,

you can send a certain block of memory into the retention state to reduce power consumption if you know you are not going to use such memory block for some time.

- In register SYSCON_MEM_POWER_UP_REG:
  - By default, all memory enters low-power state when the chip enters the Light-sleep mode.
  - Setting different bits of the SYSCON_ROM_POWER_UP field forces different blocks of Internal ROM 0 and Internal ROM 1 to work as normal (do not enter the retention state) when the chip enters Light-sleep.
  - Setting different bits of the SYSCON_SRAM_POWER_UP field forces different blocks of Internal SRAM to work as normal (do not enter the retention state) when the chip enters Light-sleep.

For detailed information about controlling different blocks using different controlling bits in the above-mentioned registers, please see Table 13-1 below.

Table 13-1. Memory Controlling Bit

Memory	Instruction Lowest Address	Instruction Highest Address	Data Lowest Address	Data Highest Address	Controlling Bit
ROM 0	0x4000_0000	0x4003_FFFF	-	-	Bit0
ROM 1	0x4004_0000	0x4007_FFFF	0x3FF0_0000	0x3FF3_FFFF	Bit1
	0x4008_0000	0x4008_FFFF	0x3FF4_0000	0x3FF4_FFFF	Bit2
SRAM Block 0	0x4037_C000	0x4037_FFFF	-	-	Bit0
SRAM Block 1	0x4038_0000	0x4038_FFFF	0x3FCA_0000	0x3FCA_FFFF	Bit1
SRAM Block 2	0x4039_0000	0x4039_FFFF	0x3FCB_0000	0x3FCB_FFFF	Bit2
SRAM Block 3	0x403A_0000	0x403B_FFFF	0x3FCC_0000	0x3FCD_FFFF	Bit3

For more information, please refer to Chapter 3 *System and Memory*.

13.3.1.2 External Memory

SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG configures encryption and decryption options of the external memory. For details, please refer to Chapter 17 *External Memory Encryption and Decryption (XTS_AES)*.

13.3.2 Clock Registers

The following registers are used to set clock sources and frequency. For more information, please refer to Chapter 6 *Reset and Clock*.

- SYSTEM_CPU_PER_CONF_REG
- SYSTEM_SYSCLK_CONF_REG

13.3.3 Interrupt Signal Registers

The following registers are used for generating the interrupt signals (software interrupt), which then can be routed to the CPU peripheral interrupts via the interrupt matrix. To be more specific, writing 1 to any of the following registers generates an interrupt signal. Writing 0 clears the interrupt signal. Therefore, these registers can be

used by software to control interrupts. The following registers correspond to the interrupt source SW_INTR_0/1/2/3. For more information, please refer to Chapter 8 *Interrupt Matrix (INTMTRX)*.

- SYSTEM_CPU_INTR_FROM_CPU_0_REG
- SYSTEM_CPU_INTR_FROM_CPU_1_REG
- SYSTEM_CPU_INTR_FROM_CPU_2_REG
- SYSTEM_CPU_INTR_FROM_CPU_3_REG

13.3.4 Peripheral Clock Gating and Reset Registers

The following registers are used for controlling the clock gating and reset of different peripherals. Details can be seen in Table 13-2.

- SYSTEM_CACHE_CONTROL_REG
- SYSTEM_PERIP_CLK_EN0_REG
- SYSTEM_PERIP_RST_EN0_REG
- SYSTEM_PERIP_CLK_EN1_REG
- SYSTEM_PERIP_RST_EN1_REG

Table 13-2. Clock Gating and Reset Bits

Component	Clock Enabling Bit [1]	Reset Controlling Bit [2 3]
CACHE Control	**SYSTEM_CACHE_CONTROL_REG**	
DCACHE	SYSTEM_DCACHE_CLK_ON	SYSTEM_DCACHE_RESET
ICACHE	SYSTEM_ICACHE_CLK_ON	SYSTEM_ICACHE_RESET
GDMA	**SYSTEM_GDMA_CTRL_REG**	
GDMA	SYSTEM_GDMA_CLK_ON	SYSTEM_GDMA_RESET
CPU	**SYSTEM_CPU_PERI_CLK_EN_REG**	**SYSTEM_CPU_PERI_RST_EN_REG**
DEBUG_ASSIST	SYSTEM_CLK_EN_ASSIST_DEBUG	SYSTEM_RST_EN_ASSIST_DEBUG
Peripherals	**SYSTEM_PERIP_CLK_EN0_REG**	**SYSTEM_PERIP_RST_EN0_REG**
SPI0 / SPI1	SYSTEM_SPI01_CLK_EN	SYSTEM_SPI01_RST
UART0	SYSTEM_UART_CLK_EN	SYSTEM_UART_RST
UART1	SYSTEM_UART1_CLK_EN	SYSTEM_UART1_RST
SPI2	SYSTEM_SPI2_CLK_EN	SYSTEM_SPI2_RST
I2C0	SYSTEM_I2C_EXT0_CLK_EN	SYSTEM_I2C_EXT0_RST
LED PWM Controller	SYSTEM_LEDC_CLK_EN	SYSTEM_LEDC_RST
Timer Group0	SYSTEM_TIMERGROUP_CLK_EN	SYSTEM_TIMERGROUP_RST
UART MEM	SYSTEM_UART_MEM_CLK_EN [4]	SYSTEM_UART_MEM_RST
APB SARADC	SYSTEM_APB_SARADC_CLK_EN	SYSTEM_APB_SARADC_RST
System Timer	SYSTEM_SYSTIMER_CLK_EN	SYSTEM_SYSTIMER_RST
ADC Controller	SYSTEM_ADC2_ARB_CLK_EN	SYSTEM_ADC2_ARB_RST
Accelerators	**SYSTEM_PERIP_CLK_EN1_REG**	**SYSTEM_PERIP_RST_EN1_REG**
ECC Accelerator	SYSTEM_CRYPTO_ECC_CLK_EN	SYSTEM_CRYPTO_ECC_RST
SHA Accelerator	SYSTEM_CRYPTO_SHA_CLK_EN	SYSTEM_CRYPTO_SHA_RST

Cont'd on next page

Table 13-2 – cont'd from previous page

Component	Clock Enabling Bit [1]	Reset Controlling Bit [2 3]
DMA	SYSTEM_DMA_CLK_EN	SYSTEM_DMA_RST[5]
TSENS	SYSTEM_TSENS_CLK_EN	SYSTEM_TSENS_RST

[1] Set the clock enabling bit to 1 to enable the clock, and to 0 to disable the clock.

[2] Set the reset controlling bit to 1 to reset a peripheral, and to 0 to disable the reset.

[3] Reset registers cannot be cleared by hardware. Therefore, SW reset clear is required after setting the reset registers.

[4] UART memory is shared by all UART peripherals, meaning having any active UART peripherals will prevent the UART memory from entering the clock-gated state.

[5] When DMA is required for peripheral communications, for example, SPI and SHA, DMA clock should also be enabled.

13.4 Register Summary

The addresses in this section are relative to the base address of System Registers provided in Table 3-3 in Chapter 3 *System and Memory*.

Name	Description	Address	Access
Peripheral Clock Control Registers			
SYSTEM_CPU_PERI_CLK_EN_REG	CPU peripheral clock enable register	0x0000	Varies
SYSTEM_CPU_PERI_RST_EN_REG	CPU peripheral reset register	0x0004	R/W
SYSTEM_PERIP_CLK_EN0_REG	SYSTEM peripheral clock enable register 1	0x0010	R/W
SYSTEM_PERIP_CLK_EN1_REG	SYSTEM peripheral clock enable register 1	0x0014	R/W
SYSTEM_PERIP_RST_EN0_REG	SYSTEM peripheral reset register 0	0x0018	R/W
SYSTEM_PERIP_RST_EN1_REG	SYSTEM peripheral reset register 1	0x001C	R/W
SYSTEM_GDMA_CTRL_REG	GDMA clock control register	0x003C	R/W
SYSTEM_CACHE_CONTROL_REG	Cache clock control register	0x0040	R/W
Clock Configuration Registers			
SYSTEM_CPU_PER_CONF_REG	CPU clock configuration register	0x0008	Varies
SYSTEM_SYSCLK_CONF_REG	System clock configuration register	0x0058	R/W
CPU Interrupt Control Registers			
SYSTEM_CPU_INTR_FROM_CPU_0_REG	CPU interrupt control register 0	0x0028	R/W
SYSTEM_CPU_INTR_FROM_CPU_1_REG	CPU interrupt control register 1	0x002C	R/W
SYSTEM_CPU_INTR_FROM_CPU_2_REG	CPU interrupt control register 2	0x0030	R/W
SYSTEM_CPU_INTR_FROM_CPU_3_REG	CPU interrupt control register 3	0x0034	R/W
System and Memory Control Registers			
SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG	External memory encryption and decryption control register	0x0044	R/W
Clock Gate Control Registers			
SYSTEM_CLOCK_GATE_REG	Clock gate control register	0x0054	R/W
Date Register			
SYSTEM_DATE_REG	Version register	0x0FFC	R/W

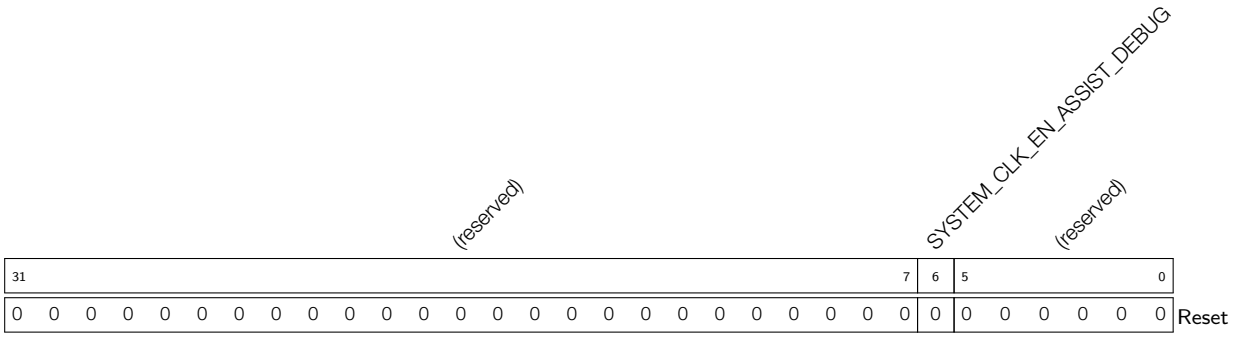
The addresses in this section are relative to the base address of APB Controller provided in Table 3-3 in Chapter 3 *System and Memory*.

Name	Description	Address	Access
Configuration Register			
SYSCON_CLKGATE_FORCE_ON_REG	Internal memory clock gate enable register	0x00A4	R/W
SYSCON_MEM_POWER_DOWN_REG	Internal memory control register	0x00A8	R/W
SYSCON_MEM_POWER_UP_REG	Internal memory control register	0x00AC	R/W

13.5 Registers

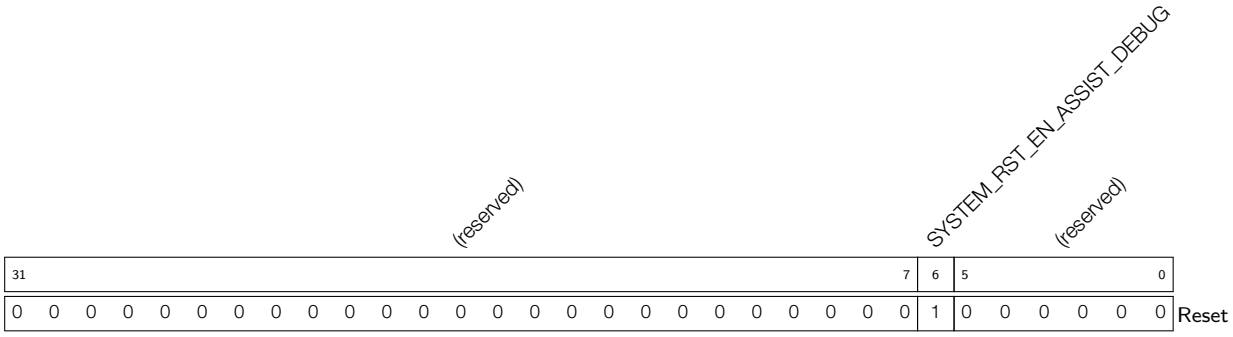
The addresses in this section are relative to the base address of System Registers provided in Table 3-3 in Chapter 3 *System and Memory*.

Register 13.1. SYSTEM_CPU_PERI_CLK_EN_REG (0x0000)



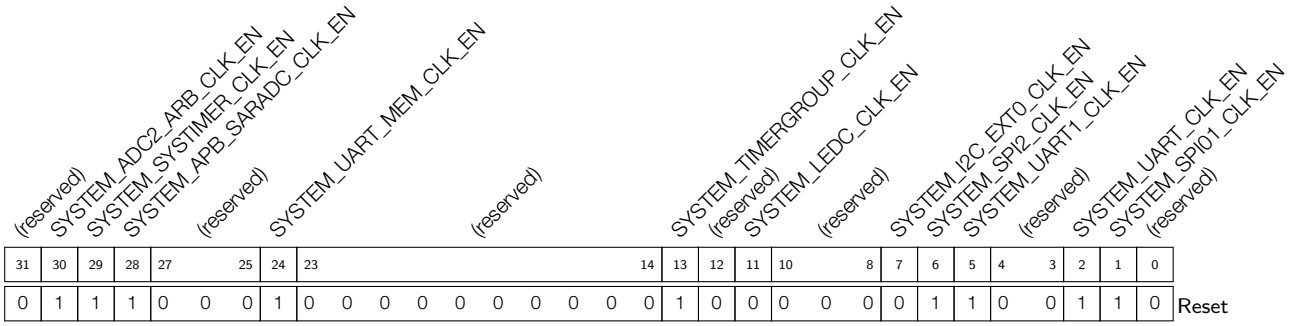
SYSTEM_CLK_EN_ASSIST_DEBUG Set this bit to enable ASSIST_DEBUG clock. Please see Chapter 14 *Debug Assistant (ASSIST_DEBUG)* for more information about ASSIST_DEBUG. (R/W)

Register 13.2. SYSTEM_CPU_PERI_RST_EN_REG (0x0004)



SYSTEM_RST_EN_ASSIST_DEBUG Set this bit to reset ASSIST_DEBUG. Please see Chapter 14 *Debug Assistant (ASSIST_DEBUG)* for more information about ASSIST_DEBUG. (R/W)

Register 13.3. SYSTEM_PERIP_CLK_EN0_REG (0x0010)



SYSTEM_SPI01_CLK_EN Set this bit to enable SPI0/SPI1 clock. (R/W)

SYSTEM_UART_CLK_EN Set this bit to enable UART clock. (R/W)

SYSTEM_UART1_CLK_EN Set this bit to enable UART1 clock. (R/W)

SYSTEM_SPI2_CLK_EN Set this bit to enable SPI2 clock. (R/W)

SYSTEM_I2C_EXT0_CLK_EN Set this bit to enable I2C_EXT0 clock. (R/W)

SYSTEM_LEDC_CLK_EN Set this bit to enable LEDC clock. (R/W)

SYSTEM_TIMERGROUP_CLK_EN Set this bit to enable TIMERGROUP clock. (R/W)

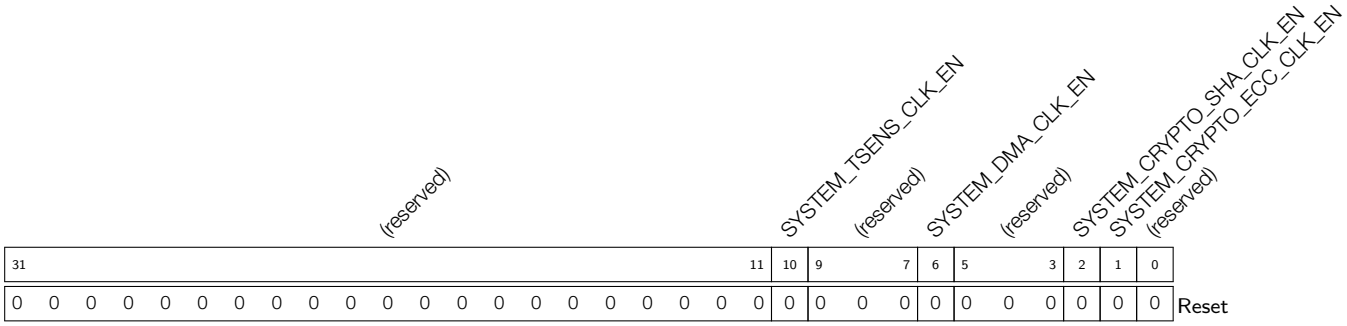
SYSTEM_UART_MEM_CLK_EN Set this bit to enable UART_MEM clock. (R/W)

SYSTEM_APB_SARADC_CLK_EN Set this bit to enable APB_SARADC clock. (R/W)

SYSTEM_SYSTIMER_CLK_EN Set this bit to enable SYSTEMTIMER clock. (R/W)

SYSTEM_ADC2_ARB_CLK_EN Set this bit to enable ADC2_ARB clock. (R/W)

Register 13.4. SYSTEM_PERIP_CLK_EN1_REG (0x0014)



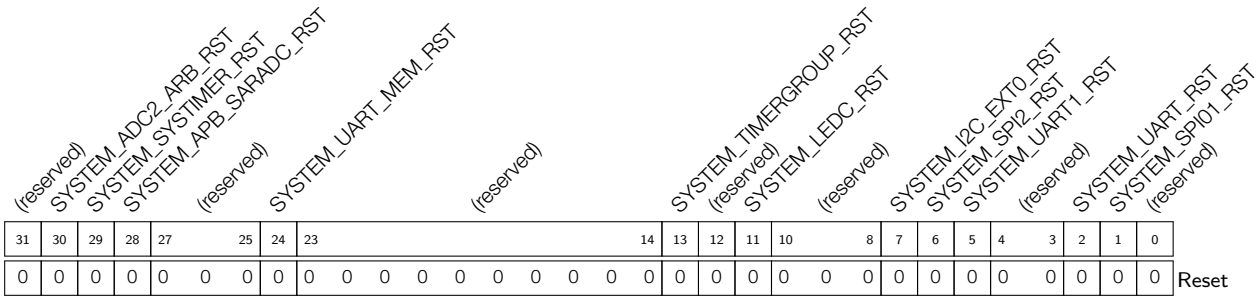
SYSTEM_CRYPTO_ECC_CLK_EN Set this bit to enable ECC clock. (R/W)

SYSTEM_CRYPTO_SHA_CLK_EN Set this bit to enable SHA clock. (R/W)

SYSTEM_DMA_CLK_EN Set this bit to enable DMA clock. (R/W)

SYSTEM_TSENS_CLK_EN Set this bit to enable TSENS clock. (R/W)

Register 13.5. SYSTEM_PERIP_RST_EN0_REG (0x0018)



SYSTEM_SPI01_RST Set this bit to reset SPI0/SPI1. (R/W)

SYSTEM_UART_RST Set this bit to reset UART. (R/W)

SYSTEM_UART1_RST Set this bit to reset UART1. (R/W)

SYSTEM_SPI2_RST Set this bit to reset SPI2. (R/W)

SYSTEM_I2C_EXT0_RST Set this bit to reset I2C_EXT0. (R/W)

SYSTEM_LEDC_RST Set this bit to reset LEDC. (R/W)

SYSTEM_TIMERGROUP_RST Set this bit to reset TIMERGROUP. (R/W)

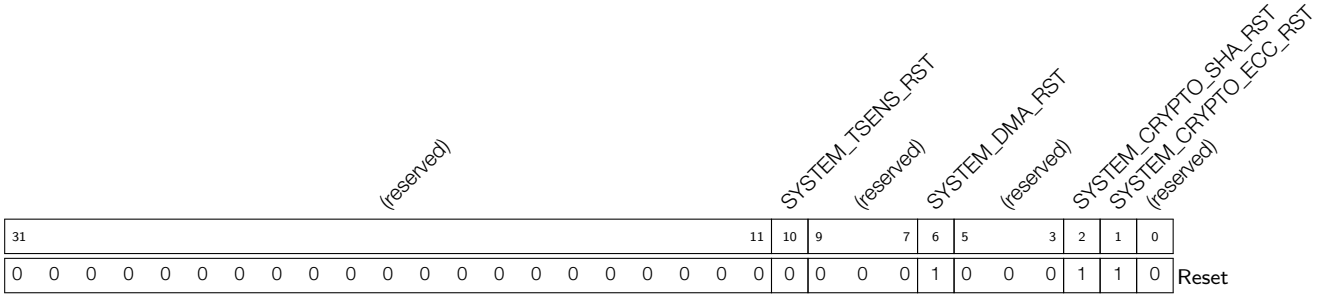
SYSTEM_UART_MEM_RST Set this bit to reset UART_MEM. (R/W)

SYSTEM_APB_SARADC_RST Set this bit to reset APB_SARADC. (R/W)

SYSTEM_SYSTIMER_RST Set this bit to reset SYSTIMER. (R/W)

SYSTEM_ADC2_ARB_RST Set this bit to reset ADC2_ARB. (R/W)

Register 13.6. SYSTEM_PERIP_RST_EN1_REG (0x001C)



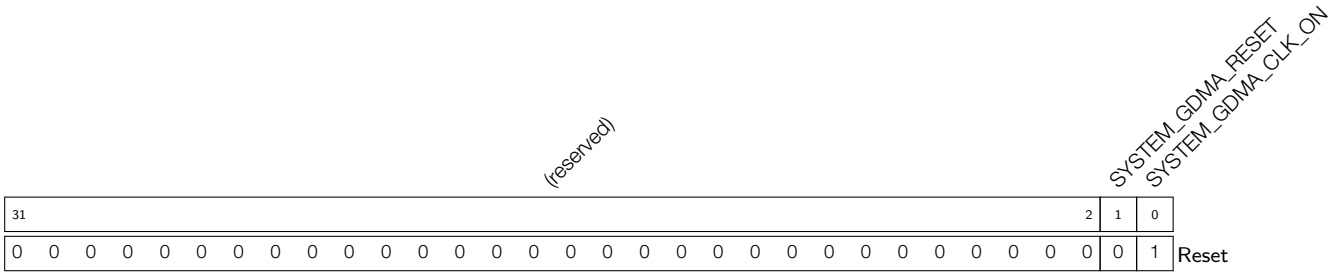
SYSTEM_CRYPTO_ECC_RST Set this bit to reset CRYPTO_ECC. (R/W)

SYSTEM_CRYPTO_SHA_RST Set this bit to reset CRYPTO_SHA. (R/W)

SYSTEM_DMA_RST Set this bit to reset DMA. (R/W)

SYSTEM_TSENS_RST Set this bit to reset TSENS. (R/W)

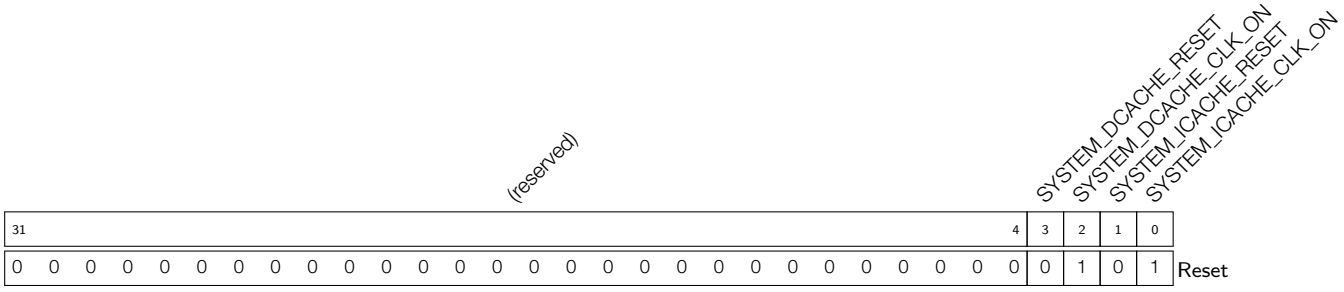
Register 13.7. SYSTEM_GDMA_CTRL_REG (0x003C)



SYSTEM_GDMA_CLK_ON Set this bit to enable GDMA clock. (R/W)

SYSTEM_GDMA_RESET Set this bit to reset GDMA. (R/W)

Register 13.8. SYSTEM_CACHE_CONTROL_REG (0x0040)



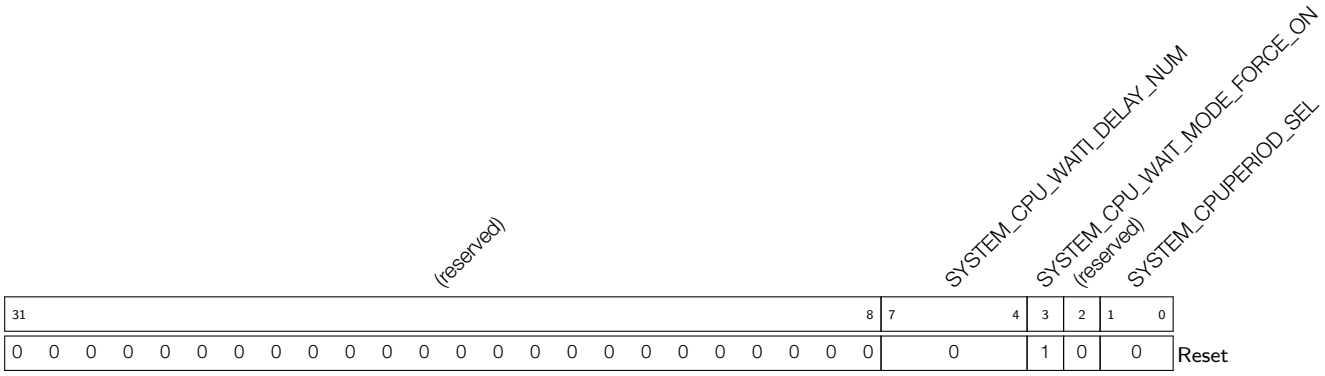
SYSTEM_ICACHE_CLK_ON Set this bit to enable i-cache clock. (R/W)

SYSTEM_ICACHE_RESET Set this bit to reset i-cache. (R/W)

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

SYSTEM_DCACHE_RESET Set this bit to reset d-cache. (R/W)

Register 13.9. SYSTEM_CPU_PER_CONF_REG (0x0008)

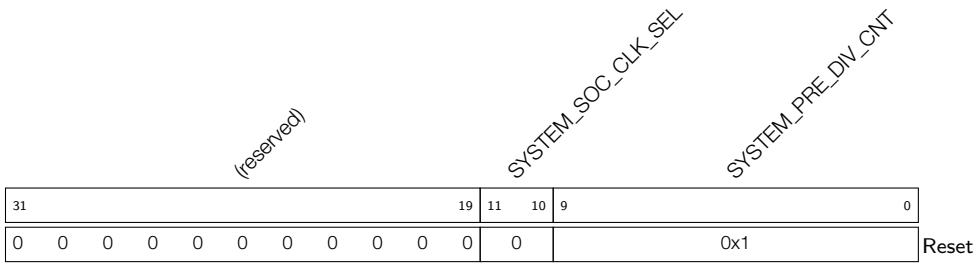


SYSTEM_CPUPERIOD_SEL Set this field to select the CPU clock frequency. For details, please refer to Table 6-4 in Chapter 6 *Reset and Clock*. (R/W)

SYSTEM_CPU_WAIT_MODE_FORCE_ON Set this bit to force on the clock gate of CPU wait mode. Usually, after executing the WFI (Wait-for-Interrupt) instruction, CPU enters the wait mode, during which the clock gate of CPU is turned off until any interrupts occur. In this way, power consumption is saved. However, if this bit is set, the clock gate of CPU is always on and will not be turned off by the WFI instruction. (R/W)

SYSTEM_CPU_WAITI_DELAY_NUM Set the number of delay cycles to turn off the CPU clock gate after the CPU enters the wait mode because of a WFI instruction. (R/W)

Register 13.10. SYSTEM_SYSCLK_CONF_REG (0x0058)



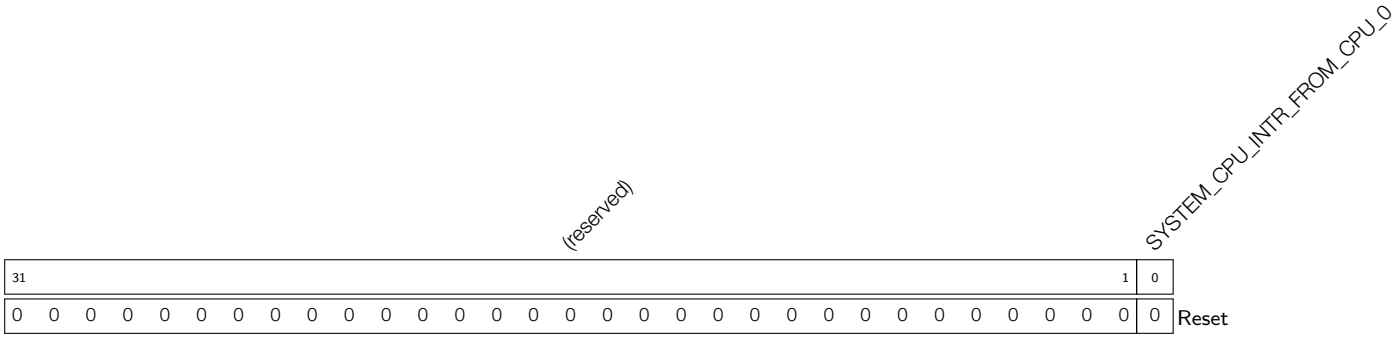
SYSTEM_PRE_DIV_CNT This field is used to set the count of prescaler of XTAL_CLK. For details, please refer to Table 6-3 in Chapter 6 *Reset and Clock*. (R/W)

SYSTEM_SOC_CLK_SEL This field is used to select SOC clock. For details, please refer to Table 6-4 in Chapter 6 *Reset and Clock*. (R/W)

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

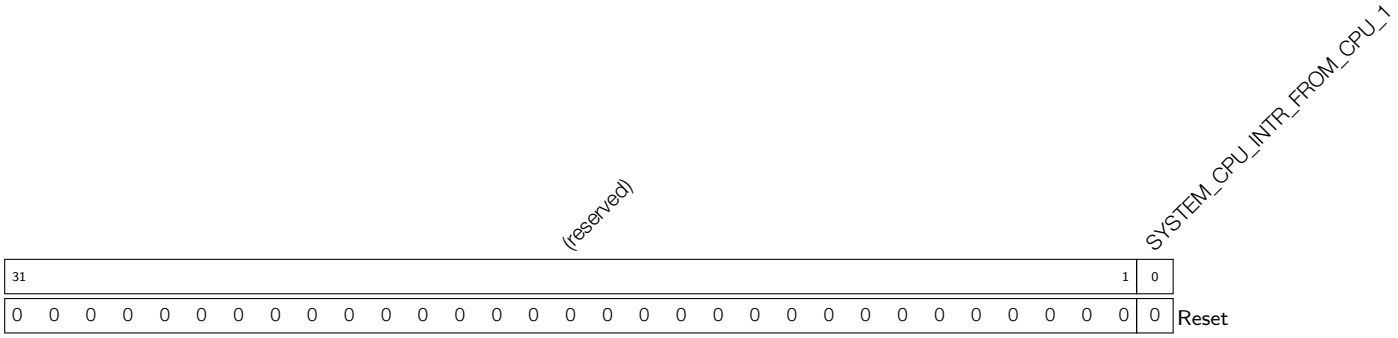
SYSTEM_CLK_DIV_EN This field is used to set clock divider. (RO)

Register 13.11. SYSTEM_CPU_INTR_FROM_CPU_0_REG (0x0028)



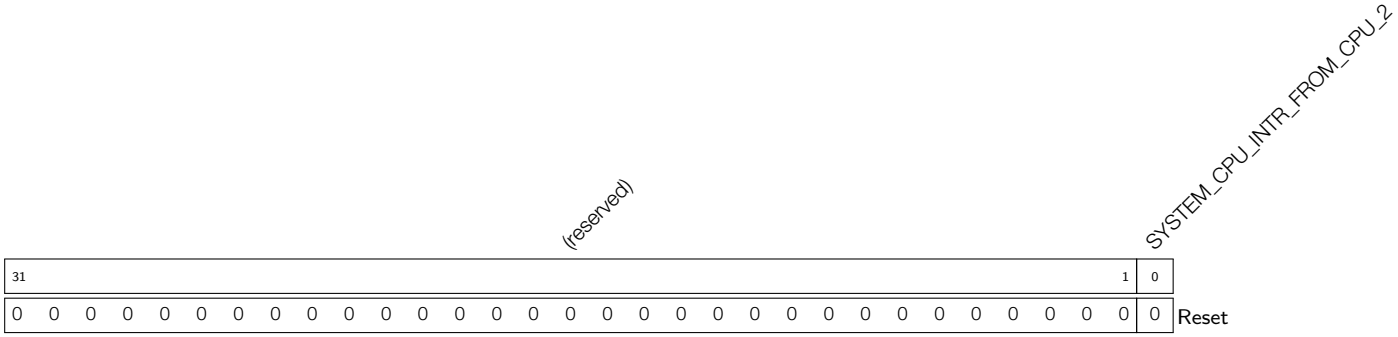
SYSTEM_CPU_INTR_FROM_CPU_0 Set this bit to generate CPU interrupt 0. This bit needs to be reset by software in the ISR process. (R/W)

Register 13.12. SYSTEM_CPU_INTR_FROM_CPU_1_REG (0x002C)



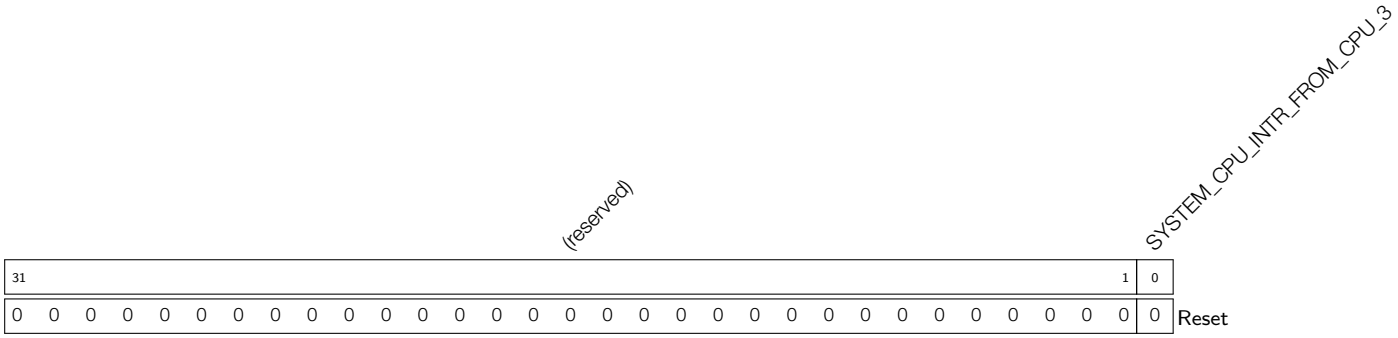
SYSTEM_CPU_INTR_FROM_CPU_1 Set this bit to generate CPU interrupt 1. This bit needs to be reset by software in the ISR process. (R/W)

Register 13.13. SYSTEM_CPU_INTR_FROM_CPU_2_REG (0x0030)



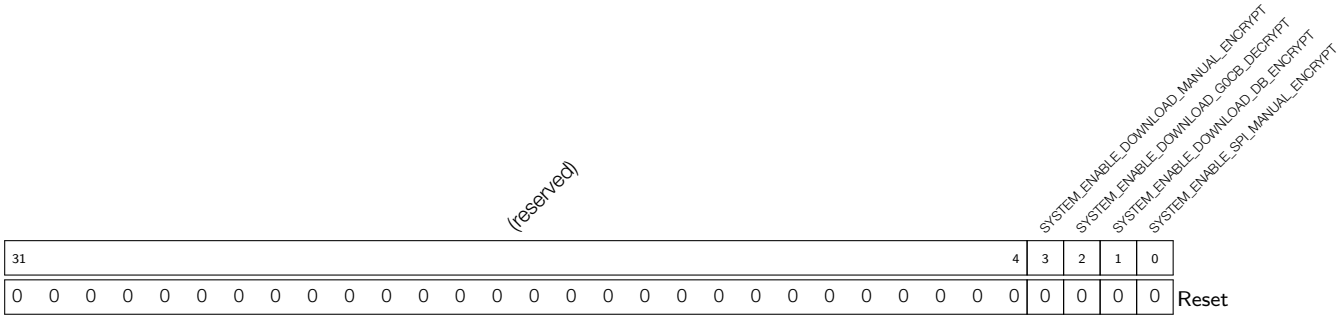
SYSTEM_CPU_INTR_FROM_CPU_2 Set this bit to generate CPU interrupt 2. This bit needs to be reset by software in the ISR process. (R/W)

Register 13.14. SYSTEM_CPU_INTR_FROM_CPU_3_REG (0x0034)



SYSTEM_CPU_INTR_FROM_CPU_3 Set this bit to generate CPU interrupt 3. This bit needs to be reset by software in the ISR process. (R/W)

Register 13.15. SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG (0x0044)



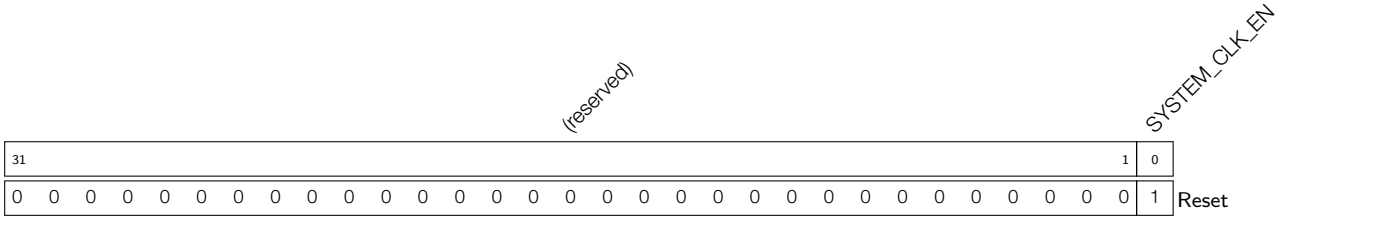
SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT Set this bit to enable Manual Encryption under SPI Boot mode. (R/W)

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

SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT Set this bit to enable Auto Decryption under Download Boot mode. (R/W)

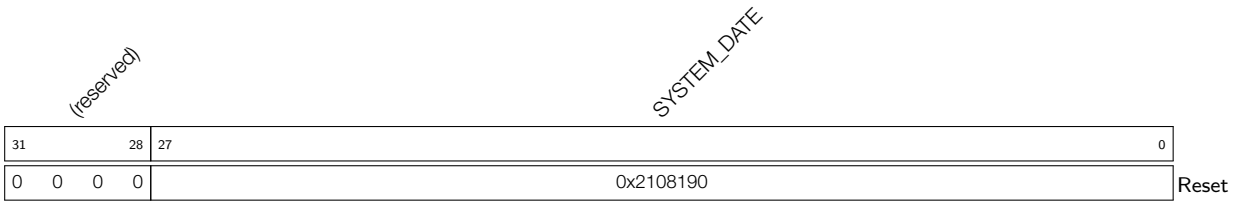
SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT Set this bit to enable Manual Encryption under Download Boot mode. (R/W)

Register 13.16. SYSTEM_CLOCK_GATE_REG (0x0054)



SYSTEM_CLK_EN Set this bit to enable the system clock. (R/W)

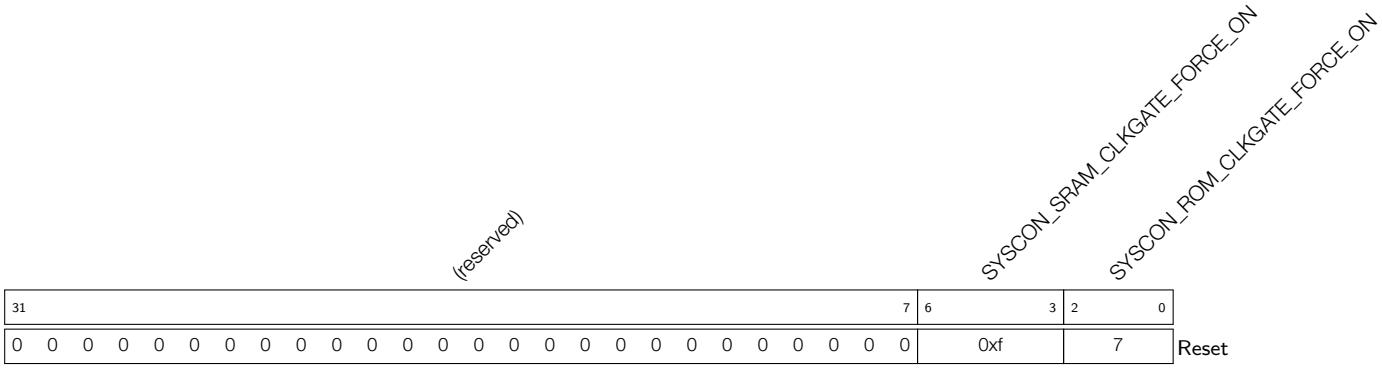
Register 13.17. SYSTEM_DATE_REG (0x0FFC)



SYSTEM_DATE Version control register. (R/W)

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

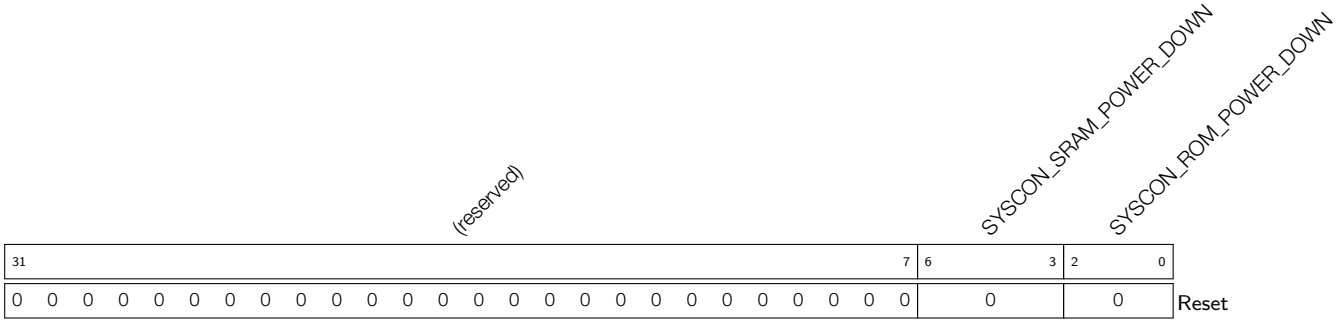
Register 13.18. SYSCON_CLKGATE_FORCE_ON_REG (0x00A4)



SYSCON_ROM_CLKGATE_FORCE_ON Set this field to configure the ROM clock gate to be always on; Set 0 to configure the clock gate to turn on automatically when ROM is accessed and turn off automatically when ROM is not accessed. (R/W)

SYSCON_SRAM_CLKGATE_FORCE_ON Set this field to configure the SRAM clock gate to be always on; Set 0 to configure the clock gate to turn on automatically when SRAM is accessed and turn off automatically when SRAM is not accessed. (R/W)

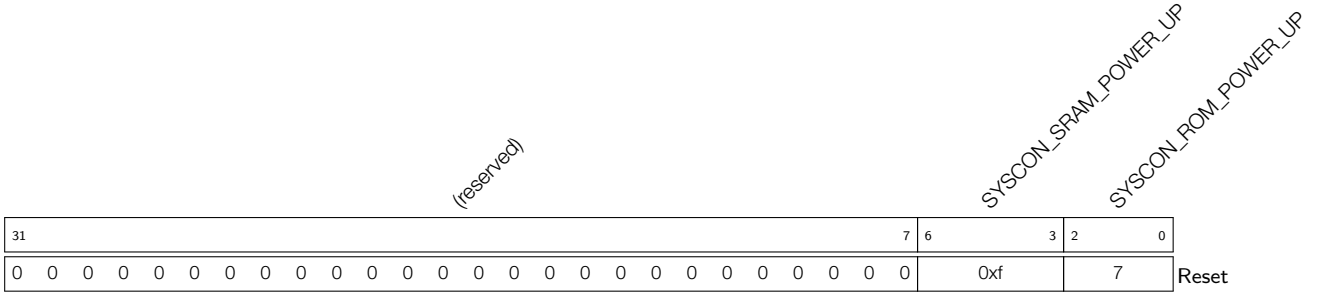
Register 13.19. SYSCON_MEM_POWER_DOWN_REG (0x00A8)



SYSCON_ROM_POWER_DOWN Set this field to send the internal ROM into retention state. (R/W)

SYSCON_SRAM_POWER_DOWN Set this field to send the internal SRAM into retention state. (R/W)

Register 13.20. SYSCON_MEM_POWER_UP_REG (0x00AC)



SYSCON_ROM_POWER_UP Set this field to force the internal ROM to work as normal (do not enter the retention state) when the chip enters light sleep. (R/W)

SYSCON_SRAM_POWER_UP Set this field to force the internal SRAM to work as normal (do not enter the retention state) when the chip enters light sleep. (R/W)

14 Debug Assistant (ASSIST_DEBUG)

14.1 Overview

Debug Assistant is an auxiliary module that features a set of functions to help locate bugs and issues during software debugging.

14.2 Features

The Debug Assistant module has the following features:

- Stack pointer (SP) monitoring
- Program counter (PC) logging before the CPU resets occurs
- CPU debugging status logging

14.3 Functional Description

14.3.1 SP Monitoring

The Debug Assistant module can monitor the SP so as to prevent stack overflow or erroneous push/pop. When the stack pointer exceeds the minimum or maximum thresholds, the Debug Assistant will record the PC's current value and generate an interrupt. Users can then read the recorded PC value to determine which instruction caused the out of bounds access. The minimum and maximum thresholds must be configured by software.

14.3.2 PC Logging

In some cases, software developers want to know the PC at the last CPU reset. For instance, when the program is stuck and can only be reset, the developer may want to know where the program got stuck in order to debug. The Debug Assistant module can record the PC at the last CPU reset, which can be then read for software debugging.

14.3.3 CPU Debugging Status Logging

The Debug Assistant module records the CPU debugging status by providing a set of read-only registers. Please refer to 1 *ESP-RISC-V CPU* for more information.

14.4 Recommended Operation

14.4.1 SP Monitoring

SP bounds check monitoring:

- SP exceeds the upper bound address
- SP exceeds the lower bound address

The configuration process for SP monitoring is as follows:

1. Configure the monitored SP threshold with ASSIST_DEBUG_CORE_0_SP_MIN_REG and ASSIST_DEBUG_CORE_0_SP_MAX_REG.
2. Configure interrupts.
   - Configure ASSIST_DEBUG_CORE_0_INTR_EN_REG to enable the interrupt of a monitoring mode.
   - Read ASSIST_DEBUG_CORE_0_INTR_RAW_REG to get the interrupt status of a monitoring mode.
   - Configure ASSIST_DEBUG_CORE_0_INTR_CLR_REG to clear their interrupts.
3. Configure ASSIST_DEBUG_CORE_0_SP_MONITOR_EN_REG to enable the monitoring mode(s). Various monitoring modes can be enabled at the same time.

Read ASSIST_DEBUG_CORE_0_SP_PC to get the PC value when an interrupt is triggered.

The interrupt of the Debug Assistant module corresponds to the interrupt source ASSIST_DEBUG_INTR of the interrupt matrix. For how to map the interrupt source to the CPU interrupt, please refer to the 8 *Interrupt Matrix (INTMTRX)*.

14.4.2 PC Logging Configuration Process

The CPU sends PC value to Debug Assistant. Only when ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN is 1, the PC is valid, otherwise, it is always 0. Only when ASSIST_DEBUG_CORE_0_RCD_RECORDEN is 1, ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG samples the CPU's PC, otherwise, it keeps the original value.

The description of ASSIST_DEBUG_CORE_0_RCD_EN_REG and ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG can be found in section 14.8 and 14.9.

When the CPU resets, ASSIST_DEBUG_CORE_0_RCD_EN_REG will reset, while ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG will not. Therefore, the latter will keep the PC value at the CPU reset. ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG records the SP value at the reset.

14.5 Register Summary

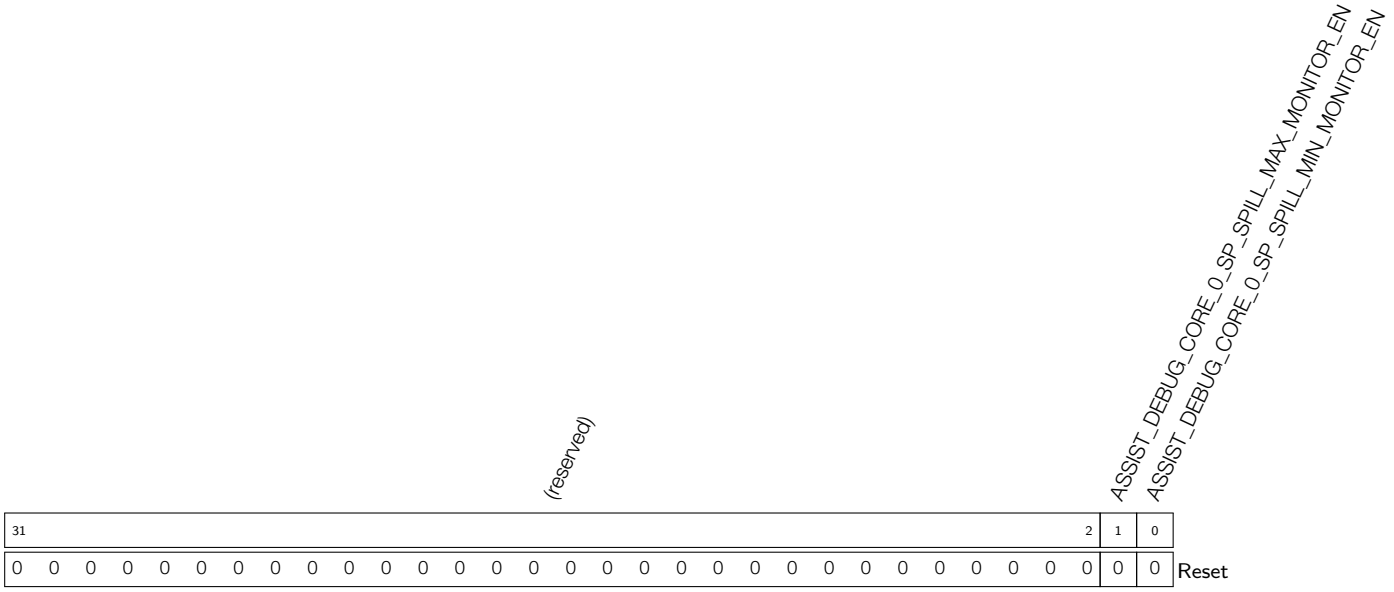
The addresses in this section are relative to the Debug Assistant base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Name	Description	Address	Access
Monitor configuration registers			
ASSIST_DEBUG_CORE_0_SP_MONITOR_EN_REG	Configure monitoring modes	0x0000	R/W
ASSIST_DEBUG_CORE_0_SP_MIN_REG	Configure stack min value	0x0010	R/W
ASSIST_DEBUG_CORE_0_SP_MAX_REG	Configure stack max value	0x0014	R/W
ASSIST_DEBUG_CORE_0_SP_PC_REG	Store PC value when an interrupt occurs	0x0018	RO
Interrupt configuration registers			
ASSIST_DEBUG_CORE_0_INTR_RAW_REG	Store interrupt status of monitoring modes	0x0004	RO
ASSIST_DEBUG_CORE_0_INTR_EN_REG	Enable interrupt of monitoring modes	0x0008	R/W
ASSIST_DEBUG_CORE_0_INTR_CLR_REG	Clear interrupt of monitoring modes	0x000C	WT
PC logging configuration register			
ASSIST_DEBUG_CORE_0_RCD_EN_REG	Enable PC logging	0x001C	R/W
PC logging status registers			
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG	Record PC value	0x0020	RO
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG	Record SP value	0x0024	RO
CPU status registers			
ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG	Store PC of the last command before CPU enters exception	0x0028	RO
ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG	Store CPU debug mode status	0x002C	RO
Clock gate register			
ASSIST_DEBUG_CLOCK_GATE_REG	Clock gate register	0x0030	R/W
Version register			
ASSIST_DEBUG_DATE_REG	Version control register	0x01FC	R/W

14.6 Registers

The addresses in this section are relative to the Debug Assistant base address provided in Table 3-3 in Chapter 3 *System and Memory*.

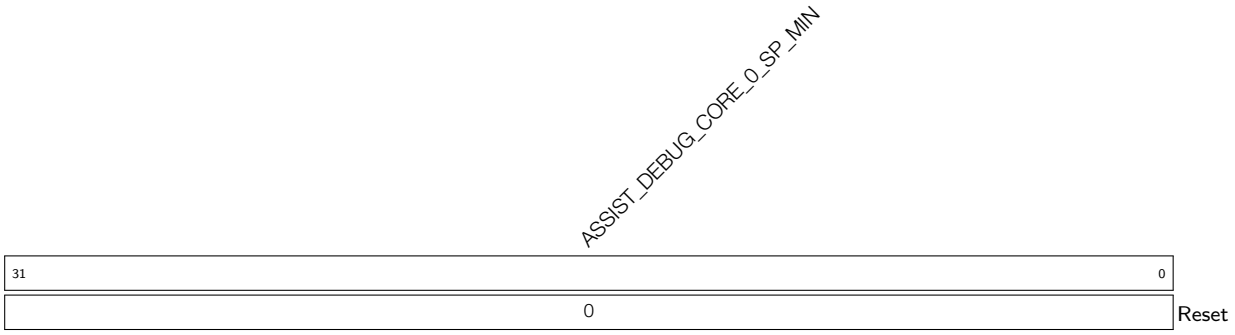
Register 14.1. ASSIST_DEBUG_CORE_0_SP_MONITOR_EN_REG (0x0000)



ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_MONITOR_EN Set 1 to enable SP underflow monitor. (R/W)

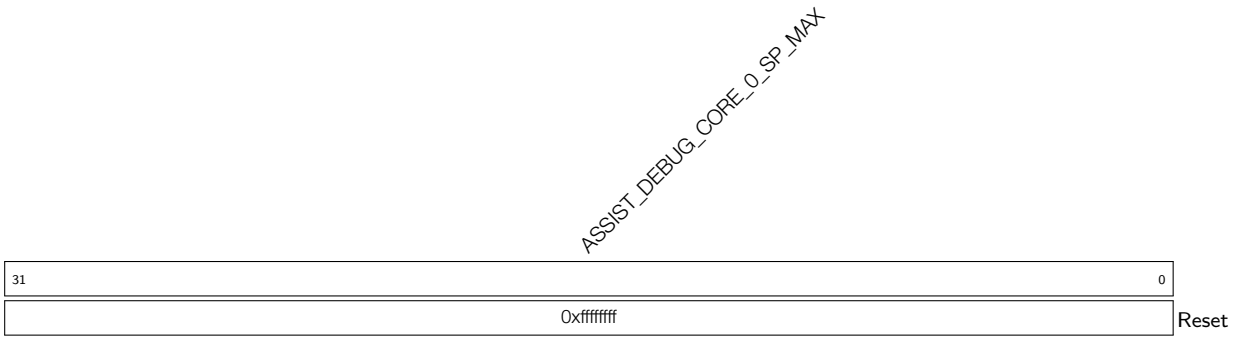
ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_MONITOR_EN Set 1 to enable SP overflow monitor. (R/W)

Register 14.2. ASSIST_DEBUG_CORE_0_SP_MIN_REG (0x0010)



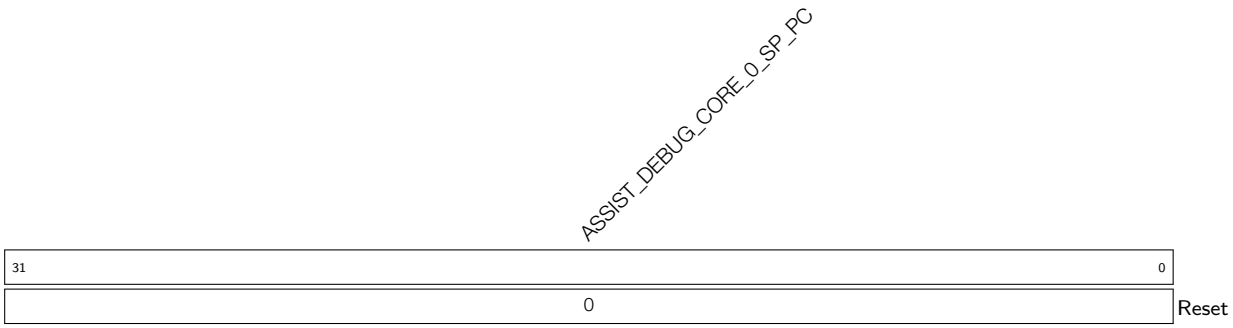
ASSIST_DEBUG_CORE_0_SP_MIN Records the lower bound address of SP. (R/W)

Register 14.3. ASSIST_DEBUG_CORE_0_SP_MAX_REG (0x0014)



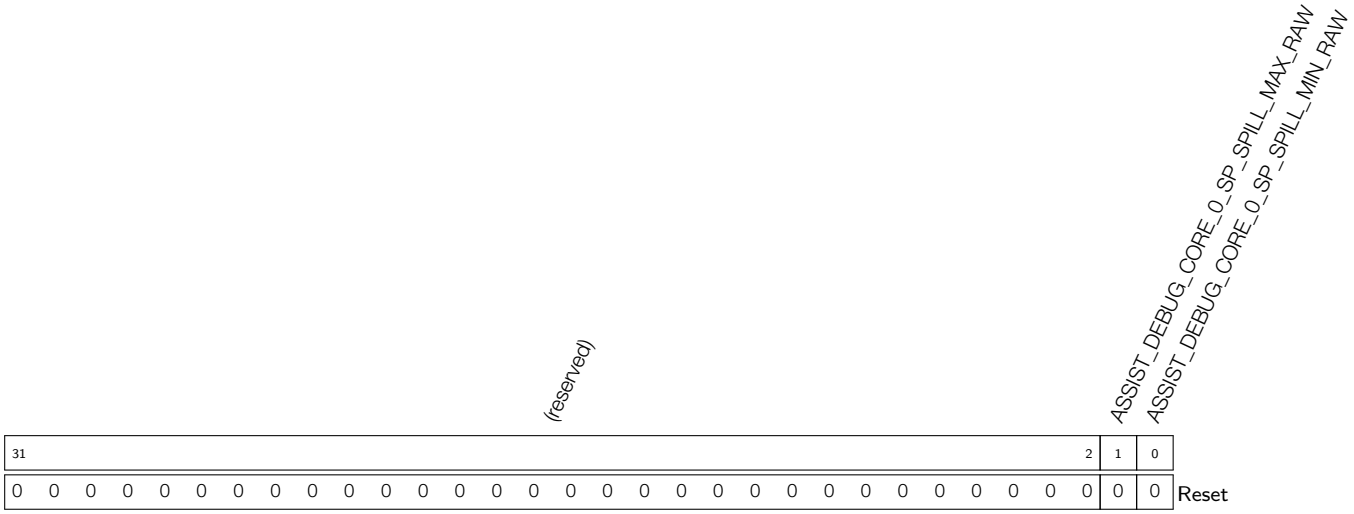
ASSIST_DEBUG_CORE_0_SP_MAX Records the upper bound address of SP. (R/W)

Register 14.4. ASSIST_DEBUG_CORE_0_SP_PC_REG (0x0018)



ASSIST_DEBUG_CORE_0_SP_PC Records the PC value during stack monitoring. (RO)

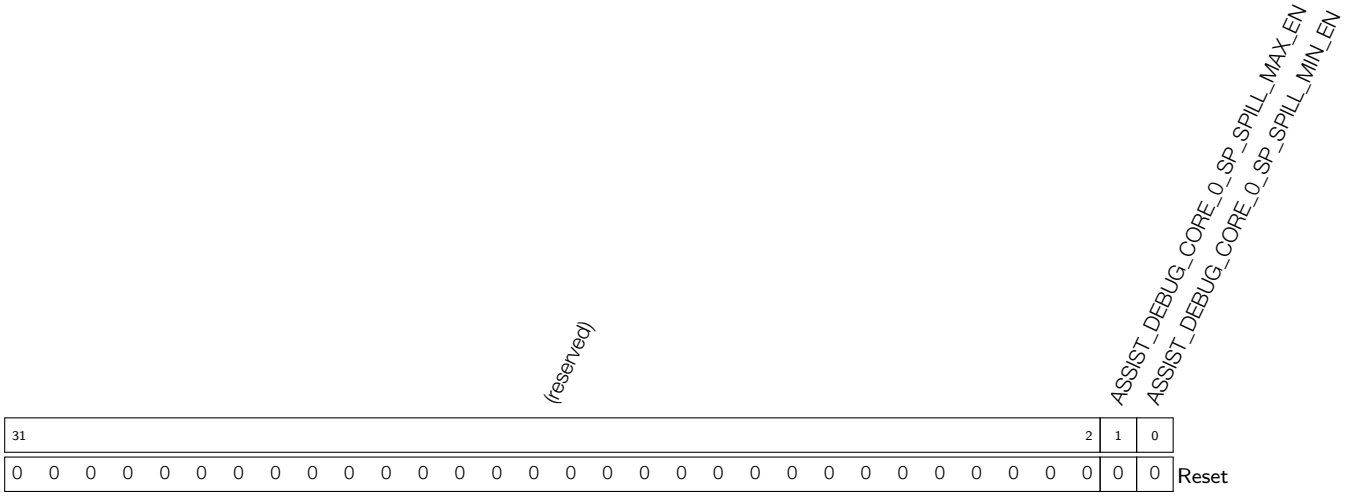
Register 14.5. ASSIST_DEBUG_CORE_0_INTR_RAW_REG (0x0004)



ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_RAW Stores the interrupt status of SP underflow monitoring. (RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_RAW Stores the interrupt status of SP overflow monitoring. (RO)

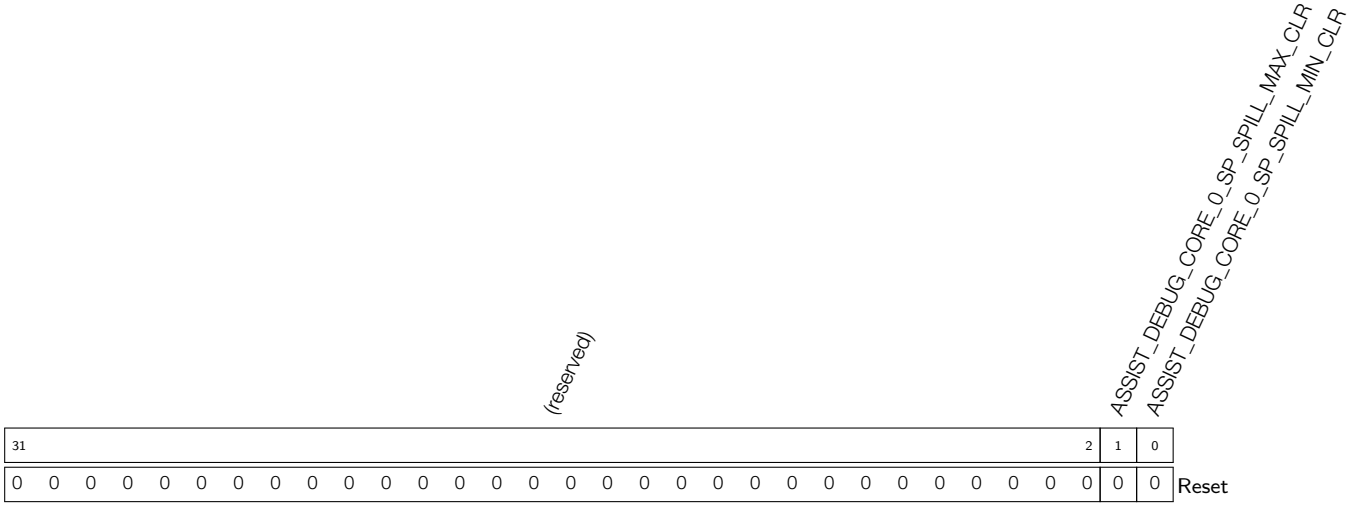
Register 14.6. ASSIST_DEBUG_CORE_0_INTR_EN_REG (0x0008)



ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_EN SP underflow monitor interrupt enable bit, 1: interrupt enabled, 0: interrupt disabled. (R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_EN SP overflow monitor interrupt enable bit, 1: interrupt enabled, 0: interrupt disabled. (R/W)

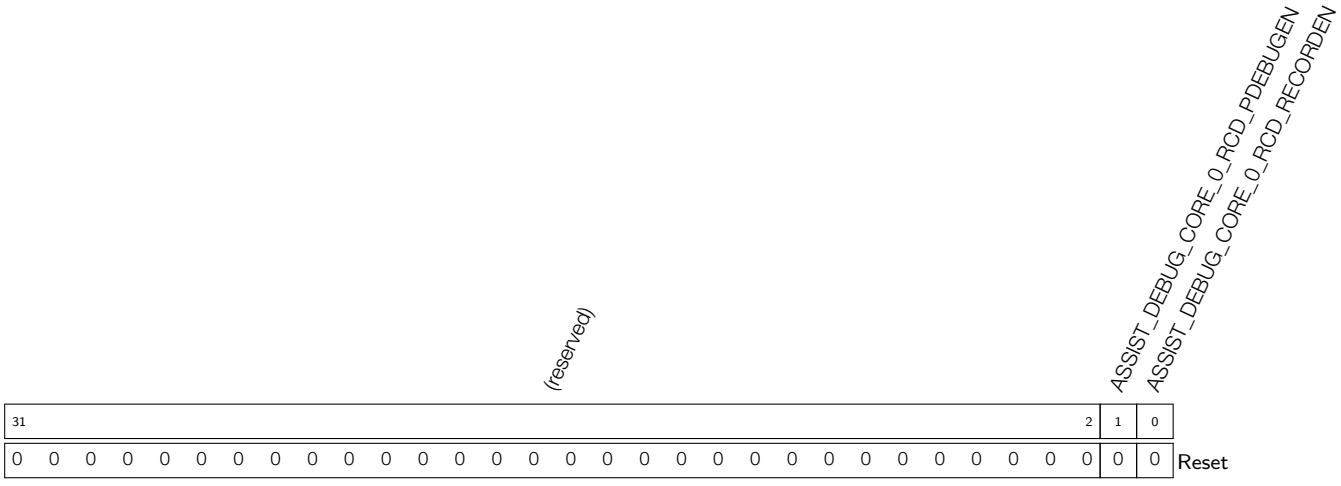
Register 14.7. ASSIST_DEBUG_CORE_0_INTR_CLR_REG (0x000C)



ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_CLR Set 1 to clear SP underflow monitor interrupt. (WT)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_CLR Set 1 to clear SP overflow monitor interrupt. (WT)

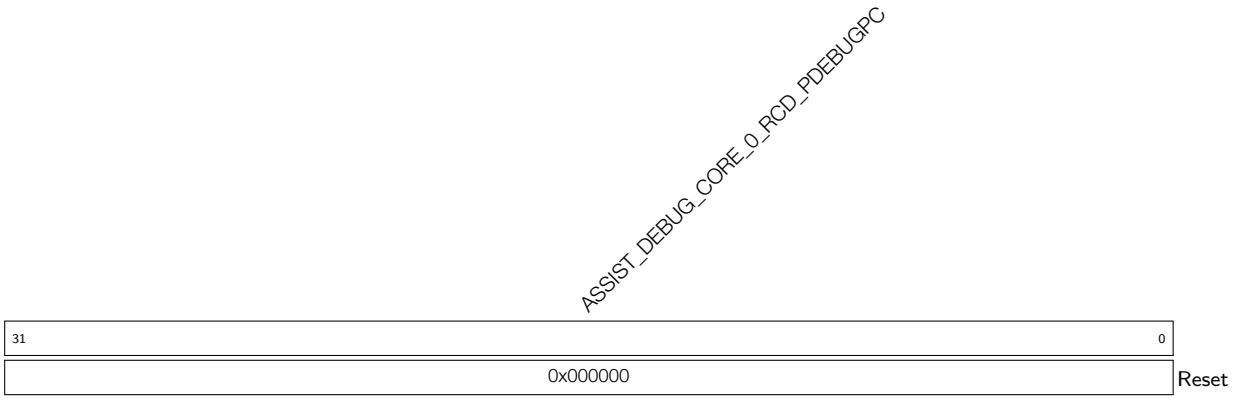
Register 14.8. ASSIST_DEBUG_CORE_0_RCD_EN_REG (0x001C)



ASSIST_DEBUG_CORE_0_RCD_RECORDEN Set 1 to enable ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG to record PC in real time. (R/W)

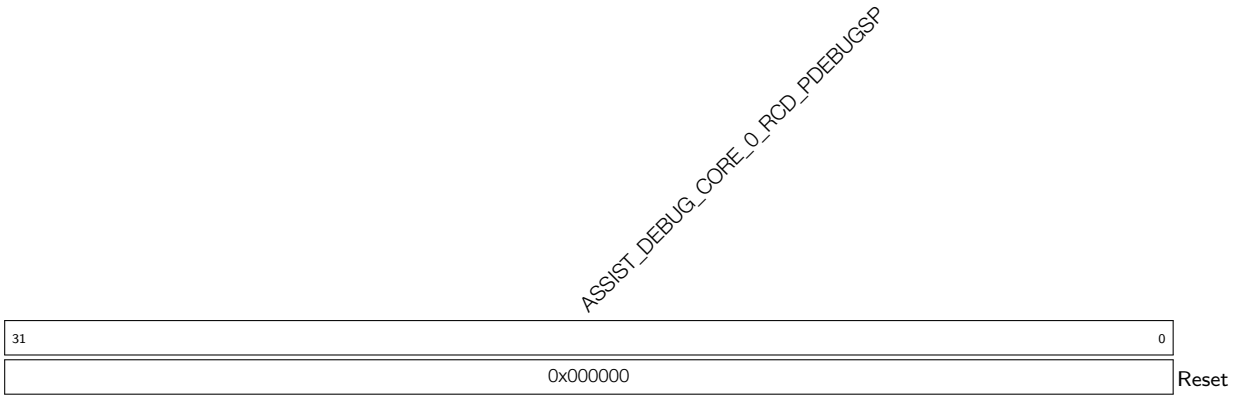
ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN Set 1 to enable CPU debug function. The CPU outputs PC only when this field is set to 1. (R/W)

Register 14.9. ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG (0x0020)



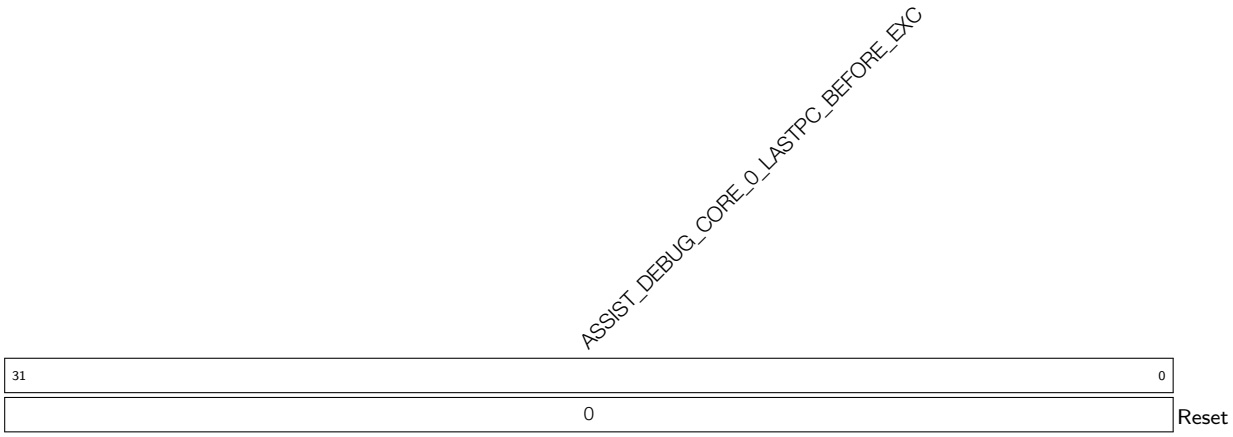
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC Records the PC value at CPU reset. (RO)

Register 14.10. ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG (0x0024)



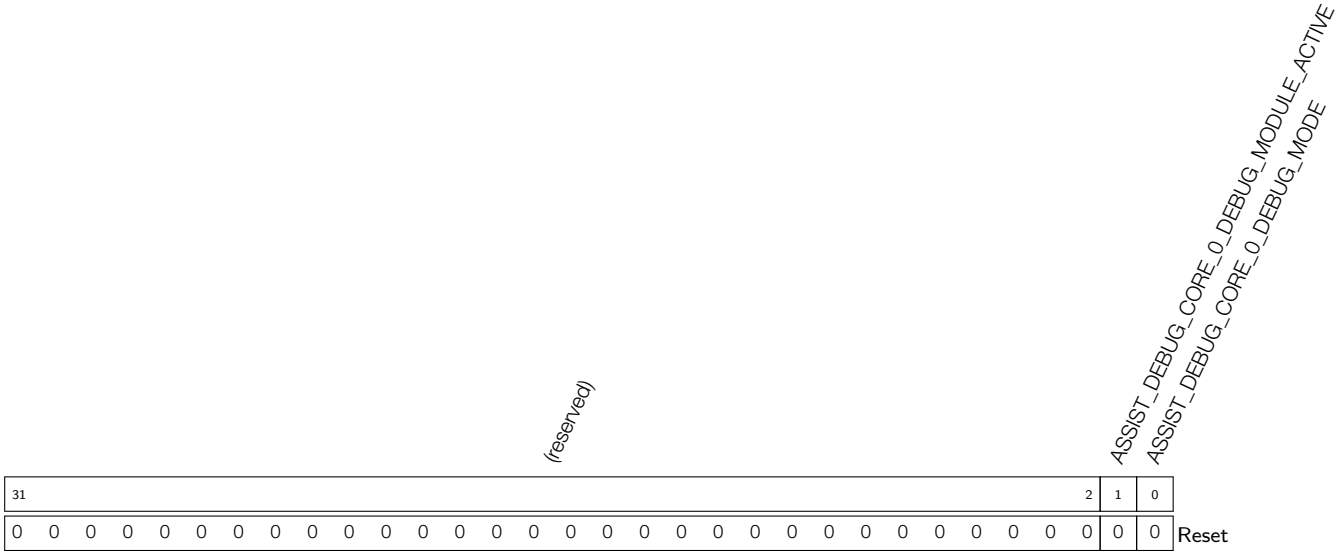
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP Records SP. (RO)

Register 14.11. ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG (0x0028)



ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXC Records the PC of the last instruction before the CPU enters exception. (RO)

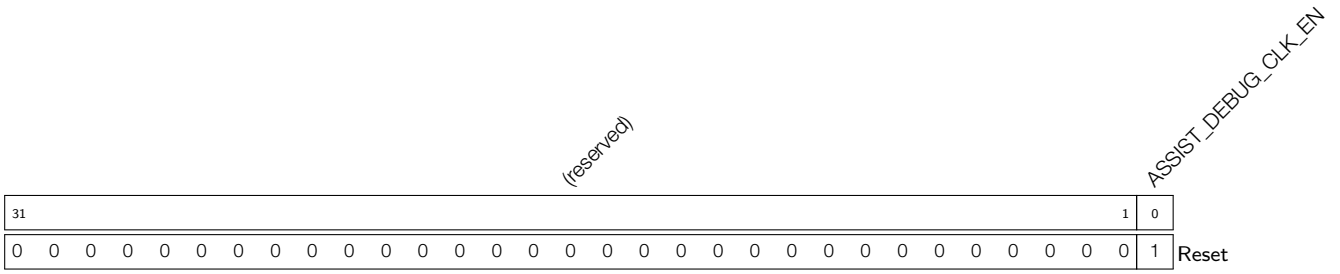
Register 14.12. ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG (0x002C)



ASSIST_DEBUG_CORE_0_DEBUG_MODE Indicates whether the RISC-V CPU is in debug mode. 1: in debug mode; 0: not in debug mode. (RO)

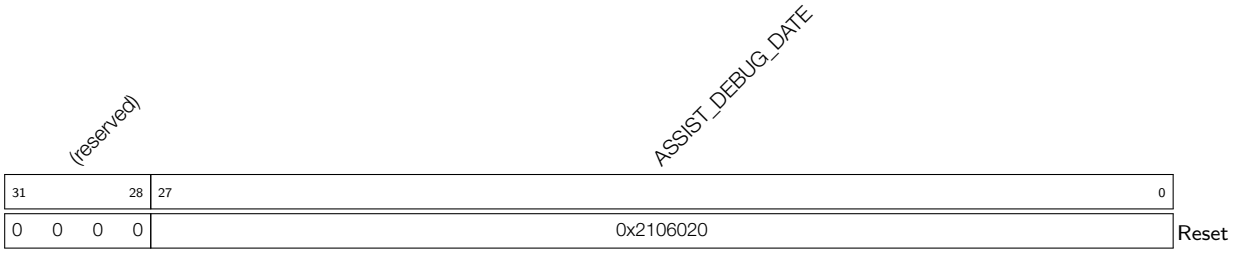
ASSIST_DEBUG_CORE_0_DEBUG_MODULE_ACTIVE Indicates the status of the RISC-V CPU debug module. 1: active status; 0: inactive status. (RO)

Register 14.13. ASSIST_DEBUG_CLOCK_GATE_REG (0x0030)



ASSIST_DEBUG_CLK_EN Clock gate register. (R/W)

Register 14.14. ASSIST_DEBUG_DATE_REG (0x01FC)



ASSIST_DEBUG_DATE Version control register. (R/W)

15 ECC Hardware Accelerator (ECC)

15.1 Introduction

Elliptic Curve Cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves. ECC allows smaller keys compared to RSA cryptography while providing equivalent security.

ESP8684's ECC Accelerator can complete various calculation based on different elliptic curves, thus accelerating ECC algorithm and ECC-derived algorithms (such as ECDSA).

15.2 Features

ESP8684's ECC Accelerator supports:

- Two different elliptic curves, namely P-192 and P-256 defined in FIPS 186-3
- Seven working modes
- Interrupt upon completion of calculation

15.3 Terminology

To better illustrate the ECC accelerator, we will first introduce the terminology used in this chapter.

15.3.1 ECC Basics

15.3.1.1 Elliptic Curve and Points on the Curves

The ECC algorithm is based on elliptic curves over prime fields, which can be represented as:

$$y^2 = x^3 + ax + b \bmod p$$

where,

- p is a prime number.
- a and b are two non-negative integers smaller than p.
- (x, y) is a point on the curve satisfying the representation.

15.3.1.2 Affine Coordinates and Jacobian Coordinates

An elliptic curve can be represented as below:

- In affine coordinates:

$$y^2 = x^3 + ax + b \bmod p$$

- In a Jacobian coordinates:

$$Y^2 = X^3 + aXZ^4 + bZ^6 \bmod p$$

To convert affine coordinates (x, y) to/from Jacobian coordinates (X, Y, Z):

- From Affine to Jacobian coordinates

$$x = X/Z^2 \bmod p$$

$$y = Y/Z^3 \bmod p$$

- From Jacobian to affine coordinates

$$X = x$$

$$Y = y$$

$$Z = 1$$

15.3.2 ECC Definitions

15.3.2.1 Memory Blocks

ECC's memory blocks store input date and output data of the ECC operation.

Table 15-1. ECC Accelerator Memory Blocks

Memory	Size (byte)	Starting Address*	Ending Address *	Access
ECC_Mem_k	32	0x100	0x11F	R/W
ECC_Mem_Px	32	0x120	0x13F	R/W
ECC_Mem_Py	32	0x140	0x15F	R/W

* Address offset relative to ECC accelerator base address provided in Table 3-3 in Chapter 3 *System and Memory*.

15.3.2.2 Data and Data Block

ESP8684's ECC operates on data of 256 bits. This data ($D[255:0]$) can be divided into eight 32-bit data blocks $D[n][31:0](n = 0, 1, \cdots, 7)$. To be specific:

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

15.3.2.3 Write Data

Write data means writing data to an ECC memory block and using this data as the input to the ECC algorithm. To be more specific, write data to an ECC memory block means write $D[n][31:0](n = 0, 1, \cdots, 7)$ to the "starting address of this ECC memory block $+4 \times n$" successively:

- write $D[0]$ to "starting address"
- write $D[1]$ to "starting address + 4"
- ...
- write $D[7]$ to "starting address + 28"

Note:
When the key size of 192 bits is used, you need to append 0 before 192 bits of data and write 256 bits of data.

15.3.2.4 Read Data

Read data means reading data from the starting address and using this data as the output from the ECC algorithm. To be more specific, read data from an ECC memory block means read $D[n][31:0](n = 0, 1, \cdots, 7)$ from the "starting address of this ECC memory block + $4 \times n$" successively:

- read $D[0]$ from "starting address"
- read $D[1]$ from "starting address + 4"
- ...
- read $D[7]$ from "starting address + 28"

Note:
When the key size of 192 bits is used, only read 192 bit (6 blocks) of data.

15.3.2.5 Standard Calculation and Jacobian Calculation

ESP8684's ECC performs Base Point Calculation (including Base Point Verification and Base Point Multiplication) using the affine coordinates and Jacobian Calculation (including Jacobian Point Verification and Jacobian Point Multiplication) using the Jacobian coordinates.

15.4 Function Description

15.4.1 Key Size

ESP8684's ECC supports acceleration based on two key sizes (corresponding to two different elliptic curves). By configuring ECC_KEY_LENGTH field, users can choose desired key size. Details can be seen in Table 15-2 below.

Table 15-2. Choose ECC Accelerator Key Size

ECC_KEY_LENGTH	Elliptic Curves
1'b0	FIPS P-192
1'b1	FIPS P-256

[1] See definition of FIPS P-192 and P-256 in FIPS 186-3.

15.4.2 Working Modes

ESP8684's ECC accelerator supports 7 working modes based on two elliptic curves described in the above section. By configuring ECC_WORK_MODE field, users can choose desired working mode. Details can be seen in Table 15-3.

Table 15-3. ECC Accelerator's Working Modes

ECC_WORK_MODE	Working Modes	ECC_WORK_MODE	Working Modes
3'd0	Point Multi Mode	3'd4	Jacobian Point Multi
3'd1	Division Mode	3'd5	**Reserved**
3'd2	Point Verif	3'd6	Jacobian Point Verif
3'd3	Point Verif + Multi	3'd7	Point Verification + Jacobian Multi

Detailed description about each working modes is provided in the following sections.

15.4.2.1 Base Point Multiplication (Point Multi Mode)

Base Point Multiplication can be represented as:

$$(Q_x, Q_y) = k \cdot (P_x, P_y)$$

where,

- Input: P_x, P_y, and k are stored in ECC_Mem_Px, ECC_Mem_Py, and ECC_Mem_k respectively.
- Output: Q_x and Q_y are stored in ECC_Mem_Px and ECC_Mem_Py respectively.

15.4.2.2 Finite Field Division (Division Mode)

Finite Field Division can be represented as:

$$\text{Result} = P_y \cdot k^{-1}$$

where,

- Input: P_y and k are stored in ECC_Mem_Py and ECC_Mem_k.
- Output: Result is stored in ECC_Mem_Py.

15.4.2.3 Base Point Verification (Point Verif Mode)

Base Point Verification can be used to verify if a point (P_x, P_y) is on a selected elliptic curve.

- Input: P_x and P_y are stored in ECC_Mem_Px and ECC_Mem_Py, respectively.
- Output: verification result is stored in ECC_VERIFICATION_RESULT field.

15.4.2.4 Base Point Verification + Base Point Multiplication (Point Verif + Multi Mode)

In this working mode, ECC first verifies if Point (P_x, P_y) is on the selected elliptic curve or not. If yes, then perform the multiplication:

$$(Q_x, Q_y) = k \cdot (P_x, P_y)$$

where,

- Input: P_x, P_y and k are stored at ECC_Mem_Px, ECC_Mem_Py, and ECC_Mem_k respectively.
- Output:
  - verification result is stored in ECC_VERIFICATION_RESULT field.

  - Q_x and Q_y are stored in ECC_Mem_Px and ECC_Mem_Py respectively.

15.4.2.5 Jacobian Point Multiplication (Jacobian Point Multi Mode)

Jacobian Point Multiplication can be represented as:

$$(Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

where,

- (Q_x, Q_y, Q_z) is a Jacobian point on the selected elliptic curve.
- 1 in the point's Jacobian coordinates is auto completed by hardware.
- Input: P_x, P_y and k are stored in ECC_Mem_Px, ECC_Mem_Py, and ECC_Mem_k respectively.
- Output: Q_x, Q_y, and Q_z are stored in ECC_Mem_Px, ECC_Mem_Py, and ECC_Mem_k, respectively.

15.4.2.6 Jacobian Point Verification (Jacobian Point Verif Mode)

Jacobian Point Verification can be used to verify if a point (Q_x, Q_y, Q_z) is on a selected elliptic curve.

- (Q_x, Q_y, Q_z) is the point in Jacobian Coordinates.
- Input: Q_x, Q_y, and Q_z are stored in ECC_Mem_Px, ECC_Mem_Py, and ECC_Mem_k, respectively.
- Output: verification result is stored in ECC_VERIFICATION_RESULT field.

15.4.2.7 Base Point Verification + Jacobian Point Multiplication (Point Verif + Jacobian Point Multi Mode)

In this working mode, ECC first verifies if Point (P_x, P_y) is on the selected elliptic curve or not. If yes, then perform the multiplication:

$$(Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

where,

- (Q_x, Q_y, Q_z) is a Jacobian point on the selected elliptic curve.
- 1 in the point's Jacobian coordinates is auto completed by hardware.
- Input: P_x, P_y, and k are stored in ECC_Mem_Px, ECC_Mem_Py, and ECC_Mem_k.
- Output:
  - verification result is stored in ECC_VERIFICATION_RESULT field.
  - Q_x, Q_y, and Q_z are stored in ECC_Mem_Px, ECC_Mem_Py, and ECC_Mem_k.

15.5 Clocks and Resets

ESP8684's ECC only has one clock module (crypo_ecc_clk) and one reset module (crypto_ecc_rst). Users should enable the ECC clock and disable the ECC reset before starting the ECC accelerator. For details on how to configure the ECC clock and reset, please refer to Chapter 6 *Reset and Clock*.

15.6 Interrupts

ESP8684's ECC accelerator can generate one interrupt signal ECC_INTR and send it to Interrupt Matrix.

Note:
Each interrupt signal is generated by any of its interrupts: any of its interrupt triggered can generate the interrupt signal.

ECC_INTR has only one interrupt ECC_CALC_DONE_INT, which is triggered on the completion of an ECC computation.

This ECC interrupt ECC_CALC_DONE_INT is configured by the following registers:

- ECC_CALC_DONE_INT_RAW: stores the raw interrupt of ECC_CALC_DONE_INT.
- ECC_CALC_DONE_INT_ST: indicates the status of the ECC_CALC_DONE_INT interrupt. This field is generated by enabling/disabling ECC_CALC_DONE_INT_RAW field via ECC_CALC_DONE_INT_ENA.
- ECC_CALC_DONE_INT_ENA: enables/disables the ECC_CALC_DONE_INT interrupt.
- ECC_CALC_DONE_INT_CLR: set this bit to clear the ECC_CALC_DONE_INT interrupt status. By setting this bit to 1, fields ECC_CALC_DONE_INT_RAW and ECC_CALC_DONE_INT_ST will be cleared.

15.7 Programming Procedures

The programming procedures for configuring ECC are described below:

1. Configure the ECC clock and reset.
2. Choose key size and working mode as described in Section 15.4.
3. Enable ECC_CALC_DONE_INT interrupt as described in Section 15.6.
4. Set ECC_START field to start ECC calculation.
5. Wait for the ECC_CALC_DONE_INT interrupt, which indicates the completion of the ECC calculation.
6. Check the result as described in Section 15.4.

15.8 Register Summary

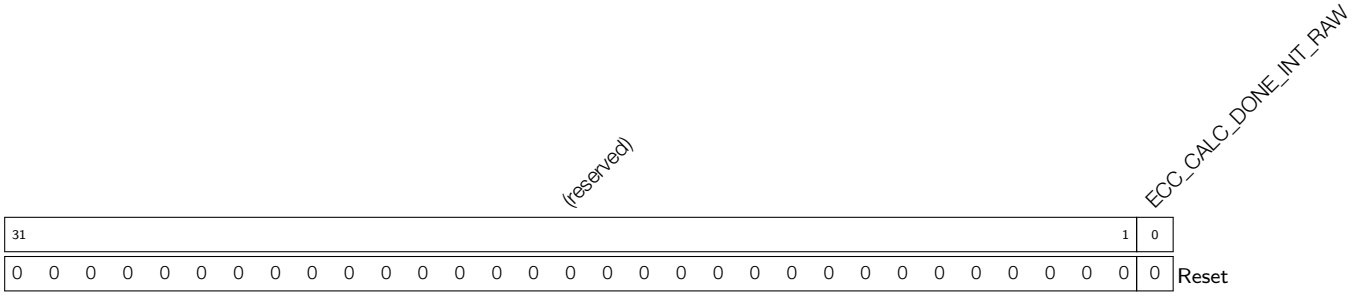
The addresses in this section are relative to ECC accelerator base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Name	Description	Address	Access
Interrupt Registers			
ECC_MULT_INT_RAW_REG	ECC raw interrupt status register	0x000C	RO/WTC/SS
ECC_MULT_INT_ST_REG	ECC masked interrupt status register	0x0010	RO
ECC_MULT_INT_ENA_REG	ECC interrupt enable register	0x0014	R/W
ECC_MULT_INT_CLR_REG	ECC interrupt clear register	0x0018	WT
Configuration Register			
ECC_MULT_CONF_REG	ECC configuration register	0x001C	varies
Version Register			
ECC_MULT_DATE_REG	Version control register	0x00FC	R/W

15.9 Registers

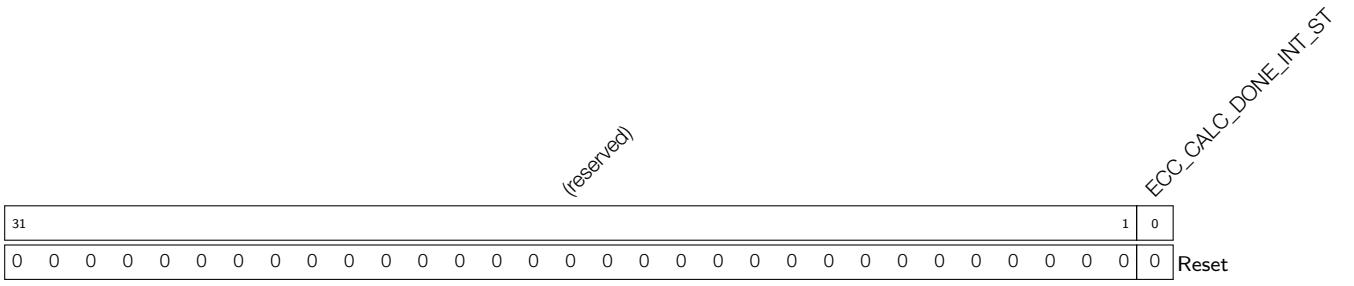
The addresses in this section are relative to ECC accelerator base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Register 15.1. ECC_MULT_INT_RAW_REG (0x000C)



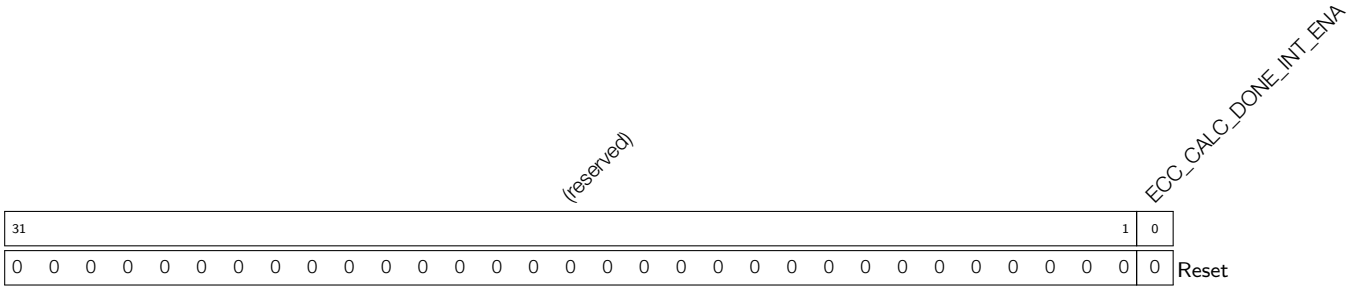
ECC_CALC_DONE_INT_RAW The raw interrupt status of ECC_CALC_DONE_INT. (RO/WTC/SS)

Register 15.2. ECC_MULT_INT_ST_REG (0x0010)



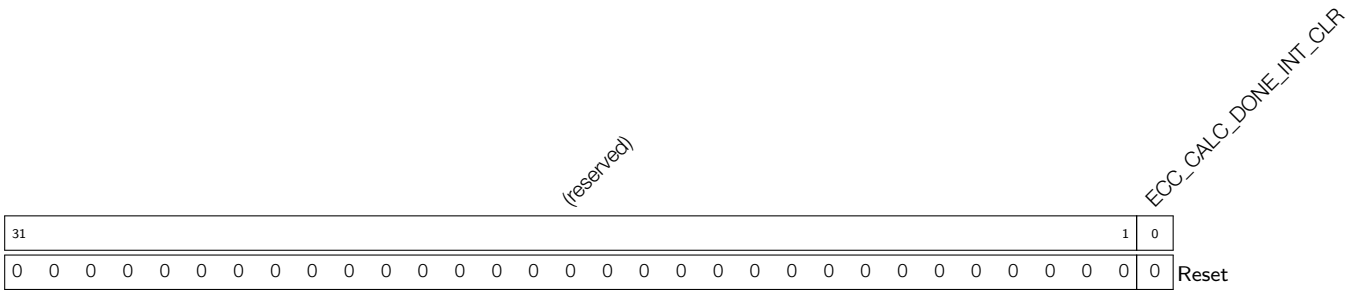
ECC_CALC_DONE_INT_ST The masked interrupt status of ECC_CALC_DONE_INT. (RO)

Register 15.3. ECC_MULT_INT_ENA_REG (0x0014)



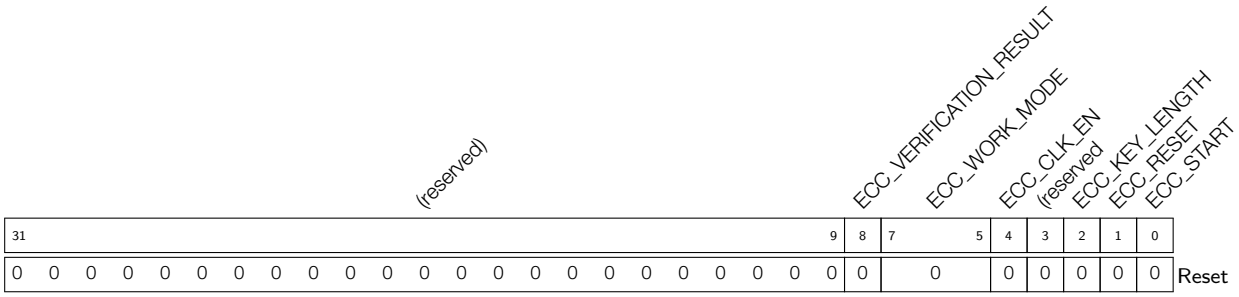
ECC_CALC_DONE_INT_ENA Write 1 to enable the ECC_CALC_DONE_INT interrupt. (R/W)

Register 15.4. ECC_MULT_INT_CLR_REG (0x0018)



ECC_CALC_DONE_INT_CLR Write 1 to clear the ECC_CALC_DONE_INT interrupt. (WT)

Register 15.5. ECC_MULT_CONF_REG (0x001C)



ECC_START Write 1 to start calculation of ECC Accelerator. This bit will be self-cleared after the calculation is done. (R/W/SC)

ECC_RESET Write 1 to reset ECC Accelerator. (WT)

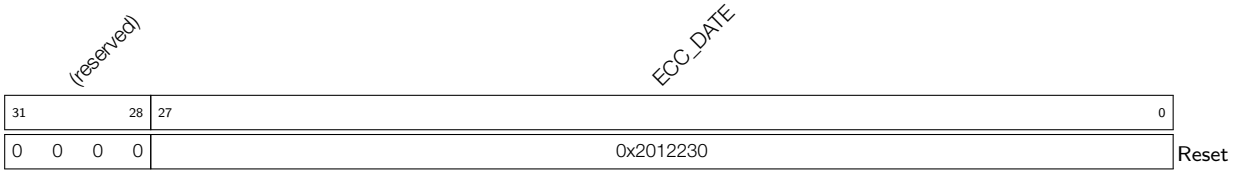
ECC_KEY_LENGTH The key length mode bit of ECC Accelerator. 1'b0: P-192. 1'b1: P-256. (R/W)

ECC_CLK_EN Write 1 to force on register clock gate. (R/W)

ECC_WORK_MODE The work mode bits of ECC Accelerator. 3'd0: Point Multi Mode. 3'd1: Division mode. 3'd2: Point verification mode. 3'd3: Point Verif+Multi mode. 3'd4: Jacobian Point Multi Mode. 3'd5: Reserved. 3'd6: Jacobian Point Verification Mode. 3'd7: Point Verif + Jacobian Multi Mode. (R/W)

ECC_VERIFICATION_RESULT The verification result bit of ECC Accelerator, only valid when calculation is done. (RO/SS)

Register 15.6. ECC_MULT_DATE_REG (0x00FC)



ECC_DATE ECC Version control register. (R/W)

16 SHA Accelerator (SHA)

16.1 Introduction

ESP8684 integrates an SHA accelerator, which is a hardware device that speeds up SHA algorithm significantly, compared to SHA algorithm implemented solely in software. The SHA accelerator integrated in ESP8684 has two working modes, which are Typical SHA and DMA-SHA.

16.2 Features

ESP8684 's SHA accelerator supports:

- Hash algorithms introduced in FIPS PUB 180-4 Spec.
  - SHA-1
  - SHA-224
  - SHA-256
- Two working modes
  - Typical SHA
  - DMA-SHA
- Interleaved function
- Interrupt function when working in DMA-SHA working mode

16.3 Working Modes

The SHA accelerator integrated in ESP8684 has two working modes.

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

Users can start the SHA accelerator with different working modes by configuring registers SHA_START_REG and SHA_DMA_START_REG. For details, please see Table 16-1.

Table 16-1. 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 16-2.

Table 16-2. SHA Hash Algorithm Selection

Hash Algorithm	SHA_MODE_REG Configuration
SHA-1	0
SHA-224	1
SHA-256	2

16.4 Function Description

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

16.4.1 Preprocessing

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

16.4.1.1 Padding the Message

The SHA accelerator can only process message blocks of 512 bits. Thus, all the messages should be padded to a multiple of 512 bits before the hash task.

Suppose that the length of the message M is m bits. Then M shall be padded as introduced below:

1. First, append the bit "1" to the end of the message;
2. Second, append k bits of zeros, where k is the smallest, non-negative solution to the equation $m + 1 + k \equiv 448 \ mod \ 512$;
3. Last, append the 64-bit block of value equal to the number m expressed using a binary representation.

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

16.4.1.2 Parsing the Message

The message and its padding must be parsed into N 512-bit blocks, $M^{(1)}$, $M^{(2)}$, ..., $M^{(N)}$. Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block i are denoted $M_0^{(i)}$, the next 32 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

During the task, all the message blocks are written into the SHA_M_n_REG: $M_0^{(i)}$ is stored in SHA_M_0_REG, $M_1^{(i)}$ stored in SHA_M_1_REG, ..., and $M_{15}^{(i)}$ stored in SHA_M_15_REG.

Note:

For more information about "message block", please refer to Section "2.1 Glossary of Terms and Acronyms" in FIPS PUB 180-4 Spec.

16.4.1.3 Setting the Initial Hash Value

Before hash task begins for any secure hash algorithms, the initial Hash value H(0) must be set based on different algorithms. However, the SHA accelerator uses the initial Hash values (constant C) stored in the hardware for hash tasks.

16.4.2 Hash Operation

After the preprocessing, the ESP8684 SHA accelerator starts to hash a message M and generates message digest of different lengths, depending on different hash algorithms. As described above, the ESP8684 SHA accelerator supports two working modes, which are Typical SHA and DMA-SHA. The operation process for the SHA accelerator under two working modes is described in the following subsections.

16.4.2.1 Typical SHA Mode Process

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

However, ESP8684 SHA also supports optional "interleaved" message digest calculation. Users can insert new calculation each time the SHA accelerator completes a sequence of operations.

- In Typical SHA mode, this can be done after each individual message block.
- In DMA-SHA mode, this can be done after a full sequence of DMA operations is complete.

Specifically, users can read out the message digest from registers SHA_H_*n*_REG after completing part of a message digest calculation, and use the SHA accelerator for a different calculation. After the different calculation completes, users can restore the previous message digest to registers SHA_H_*n*_REG, and resume the accelerator with the previously paused calculation.

Typical SHA Process

1. Select a hash algorithm.
   - Configure the SHA_MODE_REG register based on Table 16-2.
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.

Note:

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

As mentioned above, ESP8684 SHA accelerator supports **"interleaving" calculation under the Typical SHA working mode**.

The process to implement interleaved calculation is described below.

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

16.4.2.2 DMA-SHA Mode Process

ESP8684 SHA accelerator does not support "interleaving" message digest calculation at the level of individual message blocks when using DMA, which means you cannot insert new calculation before a complete DMA-SHA process (of one or more message blocks) completes. In this case, users who need interleaved operation are recommended to divide the message blocks and perform several DMA-SHA calculations, instead of trying to compute all the messages in one go.

Single DMA-SHA calculation supports up to 63 data blocks.

In contrast to the Typical SHA working mode, when the SHA accelerator is working under the DMA-SHA mode, all data read are completed via DMA. Therefore, users are required to configure the DMA controller following the description in Chapter 2 *GDMA Controller (GDMA)*.

DMA-SHA process

1. Select a hash algorithm.
   - Select a hash algorithm by configuring the SHA_MODE_REG register. For details, please refer to Table 16-2.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Configure the number of message blocks.
   - Write the number of message blocks M to the SHA_DMA_BLOCK_NUM_REG register.
4. Start the DMA-SHA calculation.
   - If the current DMA-SHA calculation follows a previous calculation, firstly write the message digest from the previous calculation to registers SHA_H_*n*_REG, then write 1 to register SHA_DMA_CONTINUE_REG to start SHA accelerator;
   - Otherwise, write 1 to register SHA_DMA_START_REG to start the accelerator.
5. Wait till the completion of the DMA-SHA calculation, which happens when:
   - The content of SHA_BUSY_REG register becomes 0, or
   - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.
6. Obtain the message digest:
   - Read the message digest from registers SHA_H_*n*_REG.

16.4.3 Message Digest

After the hash task completes, the SHA accelerator writes the message digest from the task to registers SHA_H_*n*_REG (*n*: 0~7). The lengths of the generated message digest are different depending on different hash algorithms. For details, see Table 16-3 below:

Table 16-3. 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

[1] The message digest is stored in registers from most significant bits to the least significant bits, with the first word stored in register SHA_H_0_REG and the second word stored in register SHA_H_1_REG... For details, please see subsection 16.4.1.2.

16.4.4 Interrupt

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

16.5 Register Summary

The addresses in this section are relative to the SHA accelerator base address provided in Table 3-3 in Chapter 3 *System and Memory*.

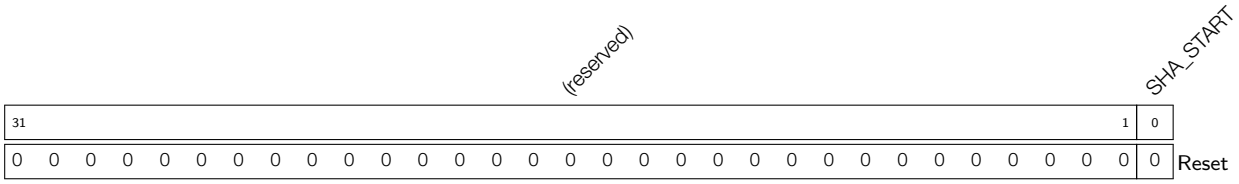
Name	Description	Address	Access
Control/Status registers			
SHA_CONTINUE_REG	Continues SHA operation (only effective in Typical SHA mode)	0x0014	WO
SHA_BUSY_REG	Indicates if SHA Accelerator is busy or not	0x0018	RO
SHA_DMA_START_REG	Starts the SHA accelerator for DMA-SHA operation	0x001C	WO
SHA_START_REG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

Name	Description	Address	Access
SHA_M_12_REG	Message	0x00B0	R/W
SHA_M_13_REG	Message	0x00B4	R/W
SHA_M_14_REG	Message	0x00B8	R/W
SHA_M_15_REG	Message	0x00BC	R/W

16.6 Registers

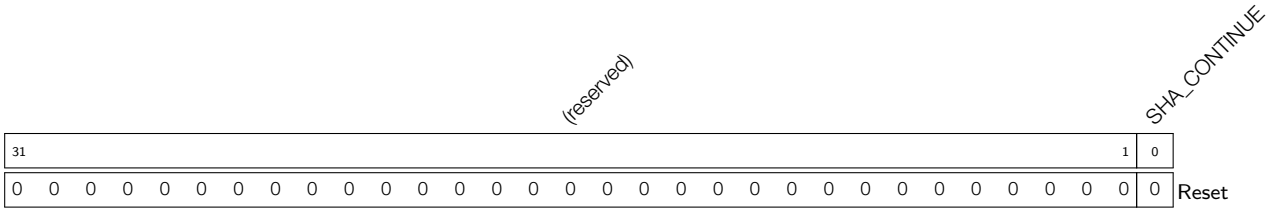
The addresses in this section are relative to the SHA accelerator base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Register 16.1. SHA_START_REG (0x0010)



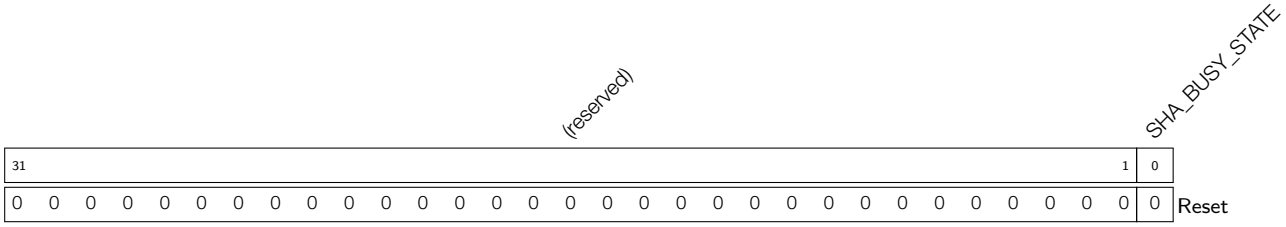
SHA_START Write 1 to start Typical SHA calculation. (WO)

Register 16.2. SHA_CONTINUE_REG (0x0014)



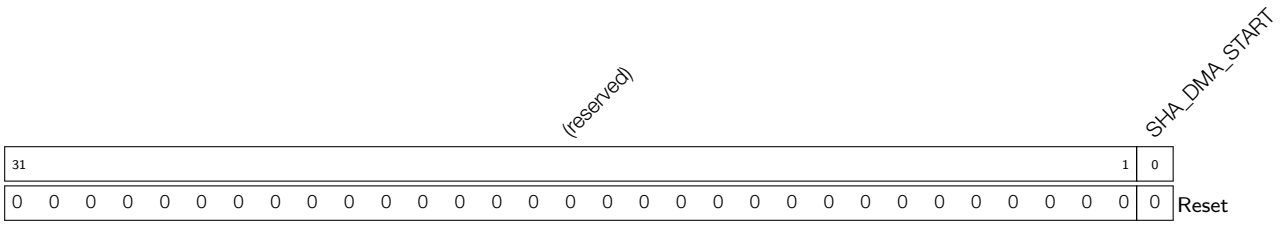
SHA_CONTINUE Write 1 to continue Typical SHA calculation. (WO)

Register 16.3. SHA_BUSY_REG (0x0018)



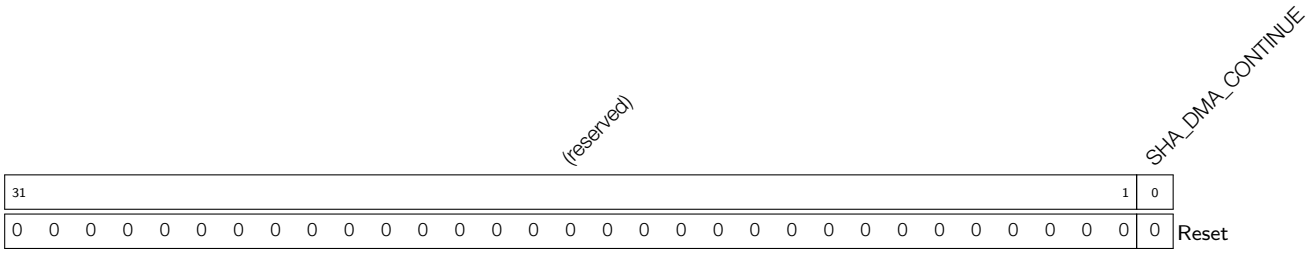
SHA_BUSY_STATE Indicates the states of SHA accelerator. (RO) 1'h0: idle 1'h1: busy

Register 16.4. SHA_DMA_START_REG (0x001C)



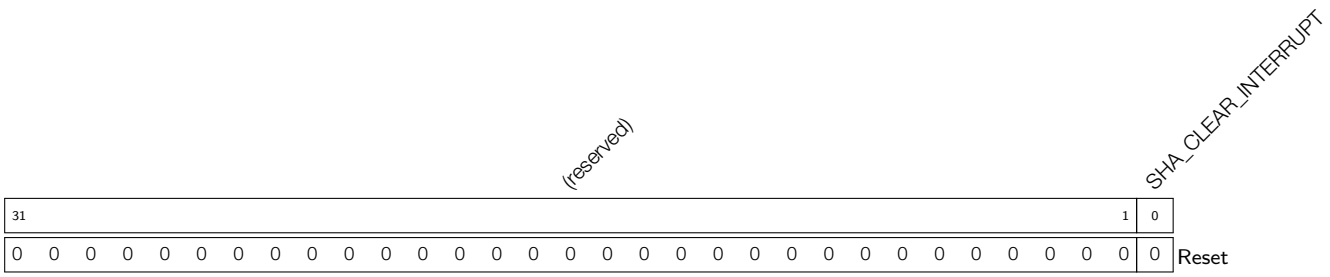
SHA_DMA_START Write 1 to start DMA-SHA calculation. (WO)

Register 16.5. SHA_DMA_CONTINUE_REG (0x0020)



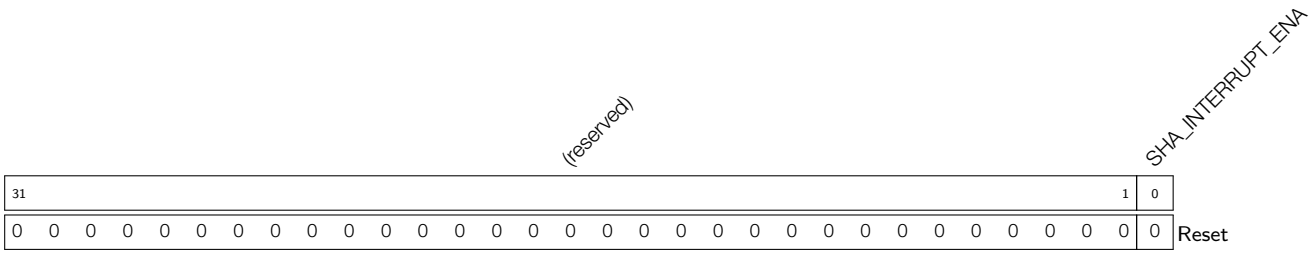
SHA_DMA_CONTINUE Write 1 to continue DMA-SHA calculation. (WO)

Register 16.6. SHA_INT_CLEAR_REG (0x0024)



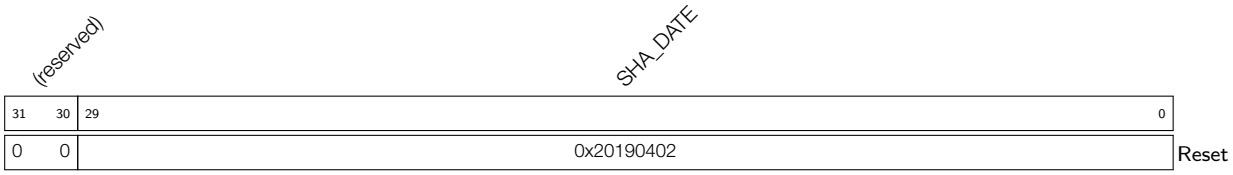
SHA_CLEAR_INTERRUPT Clears DMA-SHA interrupt. (WO)

Register 16.7. SHA_INT_ENA_REG (0x0028)



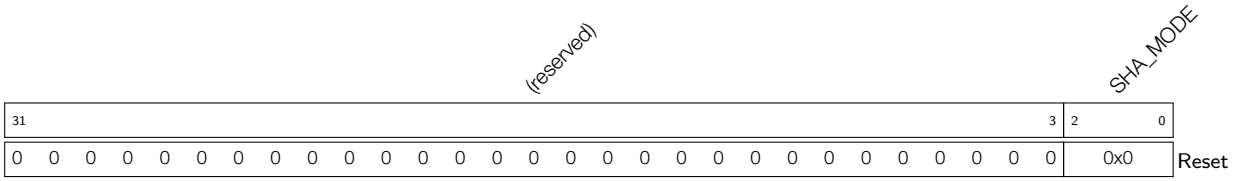
SHA_INTERRUPT_ENA Enables DMA-SHA interrupt. (R/W)

Register 16.8. SHA_DATE_REG (0x002C)



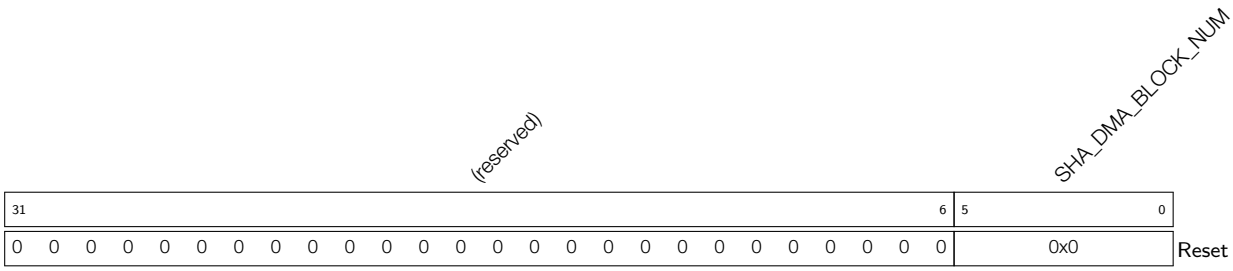
SHA_DATE Version control register. (R/W)

Register 16.9. SHA_MODE_REG (0x0000)



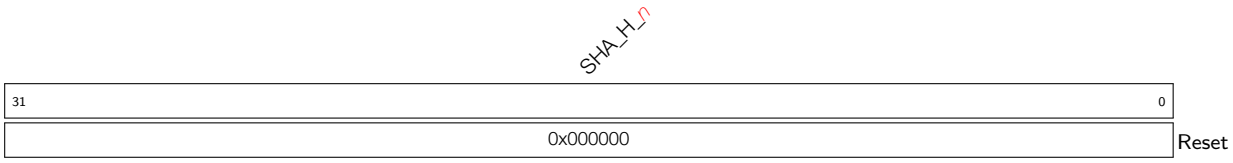
SHA_MODE Defines the SHA algorithm. For details, please see Table 16-2. (R/W)

Register 16.10. SHA_DMA_BLOCK_NUM_REG (0x000C)



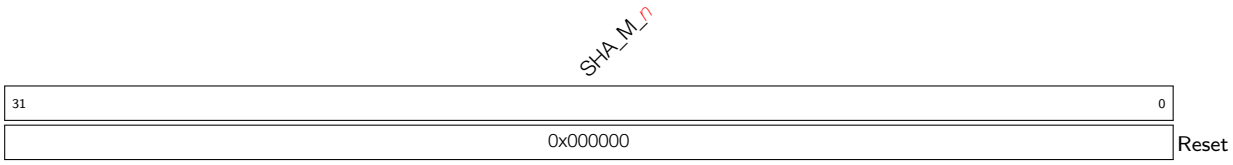
SHA_DMA_BLOCK_NUM Defines the DMA-SHA block number. (R/W)

Register 16.11. SHA_H_*n*_REG (*n*: 0-7) (0x0040+4n*)**



SHA_H_*n* Stores the *n*th 32-bit piece of the Hash value. (R/W)

Register 16.12. SHA_M_*n*_REG (*n*: 0-15) (0x0080+4n*)**



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

17 External Memory Encryption and Decryption (XTS_AES)

17.1 Overview

The ESP8684 integrates an External Memory Encryption and Decryption module that complies with the XTS_AES standard algorithm specified in IEEE Std 1619-2007, providing security for users' application code and data stored in the external memory (flash). Users can store proprietary firmware and sensitive data (e.g., credentials for gaining access to a private network) to the external flash.

17.2 Features

This module supports the following features:

- General XTS_AES algorithm, compliant with IEEE Std 1619-2007
- Software-triggered manual encryption
- High-speed auto decryption, without software's participation
- Encryption and decryption functions jointly determined by register configurations, eFuse parameters, and Boot modes

17.3 Module Structure

The External Memory Encryption and Decryption module consists of two blocks, namely the Manual Encryption block and Auto Decryption block. The module architecture is shown in Figure 17-1.

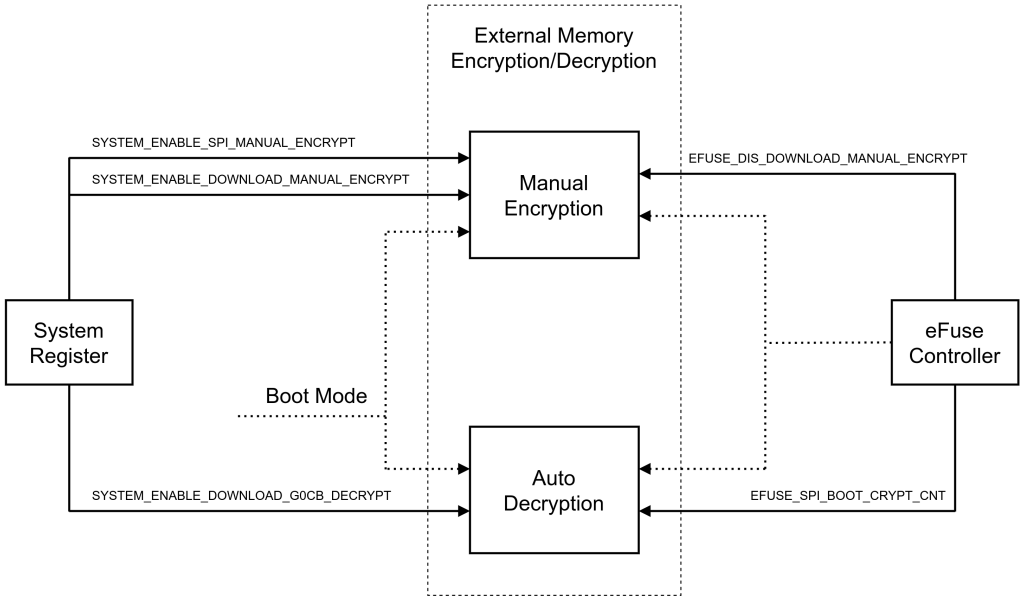


Figure 17-1. Architecture of the External Memory Encryption and Decryption

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

In the System Registers (SYSREG) peripheral (see Chapter 13 *System Registers (SYSTEM)*), the following four bits in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG are relevant to the external

memory encryption and decryption:

- SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT
- SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT
- SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT
- SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT

The XTS_AES module also fetches two parameters from the peripheral eFuse Controller, which are: EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT and EFUSE_SPI_BOOT_ENCRYPT_DECRYPT_CNT. For detailed information, please see Chapter 4 *eFuse Controller (eFuse)*.

17.4 Functional Description

17.4.1 XTS Algorithm

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

17.4.2 Key

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

The Key is 256-bit long. The value of the Key is determined by the eFuse parameters. For easier description, we define:

- Key_A: the lowest 128-bit of BLOCK3 in eFuse.
- Key_B: the highest 128-bit of BLOCK3 in eFuse.

There are two possibilities of how the Key is generated depending on the value of EFUSE_XTS_KEY_LENGTH_256. In each case, the Key can be uniquely determined by Key_A and Key_B as shown in Table 17-1.

Table 17-1. Key Generated Based on Key_A, Key_B

EFUSE_XTS_KEY_LENGTH_256	Key	Key Length (bit)
1	$\{Key_B, Key_A\}$	256
0	$SHA-256(Key_A)$[1]	256

[1] "SHA-256" indicates the SHA-256 algorithm, please refer to Chapter 16 *SHA Accelerator (SHA)*.

17.4.3 Target Memory Space

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

- Target size: the $size$ of the target memory space, indicating the number of bytes encrypted in one

encryption operation, which supports 16 or 32 bytes.

- Base address: the $base\_addr$ of the target memory space. It is a 24-bit physical address, with range of 0x0000_0000 ~ 0x00FF_FFFF. It should be aligned to $size$, i.e., $base\_addr\%size == 0$.

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

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

For Auto Decryption blocks, these parameters are automatically determined by hardware. For Manual Encryption blocks, these parameters should be configured by users.

Note:

The "tweak" defined in Section *Data units and tweaks* of IEEE Std 1619-2007 is a 128-bit non-negative integer ($tweak$), which can be generated according to $tweak$ = ($base\_addr$ & 0x00FFFF80). The lowest 7 bits and the highest 97 bits in $tweak$ are always zero.

17.4.4 Data Writing

For Auto Decryption blocks, data writing is automatically applied in hardware. For Manual Encryption blocks, data writing should be applied by users. The Manual Encryption block has a register block which consists of 8 registers, i.e., XTS_AES_PLAIN_*n*_REG (*n*: 0 ~ 7), that are dedicated to data writing and can store up to 256 bits of plaintext at a time.

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

How mapping between target memory space and registers works:

Assume a word in the target memory space is stored in $address$, define $offset = address\%32$, $n = \frac{offset}{4}$, then the word will be stored in register XTS_AES_PLAIN_*n*_REG.

For example, when the target size is 32, all registers in the register block will be used. The mapping between $offset$ and registers is shown in Table 17-2.

Table 17-2. Mapping Between Offsets and Registers

$offset$	Register	$offset$	Register
0x00	XTS_AES_PLAIN_0_REG	0x10	XTS_AES_PLAIN_4_REG
0x04	XTS_AES_PLAIN_1_REG	0x14	XTS_AES_PLAIN_5_REG
0x08	XTS_AES_PLAIN_2_REG	0x18	XTS_AES_PLAIN_6_REG
0x0C	XTS_AES_PLAIN_3_REG	0x1C	XTS_AES_PLAIN_7_REG

17.4.5 Manual Encryption Block

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

Manual encryption is allowed only when the Manual Encryption block has operation permissions. Whether the Manual Encryption block has operation permissions depends on:

- In SPI Boot mode

  If bit SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1, the Manual Encryption block has operation permissions. Otherwise, it is not operational.

- In Download Boot mode

  If bit SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1 and the eFuse parameter EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT is 0, the Manual Encryption block has operation permissions. Otherwise, it is not operational.

Note:

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

17.4.6 Auto Decryption Block

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

Auto decryption is allowed only when the Auto Decryption block has operation permissions. Whether the Auto Decryption block has operation permissions depends on:

- In SPI Boot mode

  If the first bit or the third bit in the eFuse parameter EFUSE_SPI_BOOT_ENCRYPT_DECRYPT_CNT (3 bits) is set to 1, then the Auto Decryption block has operation permissions. Otherwise, it is not operational.

- In Download Boot mode

  If bit SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT in register SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG is 1, the Auto Decryption block has operation permissions. Otherwise, it is not operational.

Note:

- When the Auto Decryption block has operation permissions, it will automatically decrypt the ciphertext if the CPU reads instructions/data from the external memory via cache to retrieve the instructions/data. The entire decryption process does not need software participation and is transparent to the cache. Users can by no means obtain the

decryption Key during the process.

- When the Auto Decryption block has no operation permissions, it does not have any effect on the contents stored in the external memory, no matter if they are encrypted or not. Therefore, what the CPU reads via cache is the original information stored in the external memory.

17.5 Software Process

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

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

   For definitions of $base\_addr$ and $size$, please refer to Section 17.4.3.
2. Write plaintext data to the registers block XTS_AES_PLAIN_*n*_REG (*n*: 0 ~ 7). For detailed information, please refer to Section 17.4.4.
   Please write data to registers according to your actual needs, and the unused ones could be set to arbitrary values.
3. Wait for Manual Encryption block to be idle. Poll register XTS_AES_STATE_REG until it reads 0 that indicates the Manual Encryption block is idle.
4. Trigger manual encryption by writing 1 to register XTS_AES_TRIGGER_REG.
5. Wait for the encryption process completion. Poll register XTS_AES_STATE_REG until it reads 2.
   Step 1 to 5 are the steps of encrypting plaintext instructions with the Manual Encryption block using the Key.
6. Write 1 to register XTS_AES_RELEASE_REG to grant SPI1 the access to the encrypted ciphertext. Then, poll register XTS_AES_STATE_REG until it reads 3.
7. Call SPI1 to write the ciphertext in the external flash (see Chapter 20 *SPI Controller (SPI)*).
8. Write 1 to register XTS_AES_DESTROY_REG to destroy the ciphertext. After this, the value of register XTS_AES_STATE_REG will become 0.

Repeat above steps according to the amount of plaintext instructions/data that need to be encrypted.

17.6 Register Summary

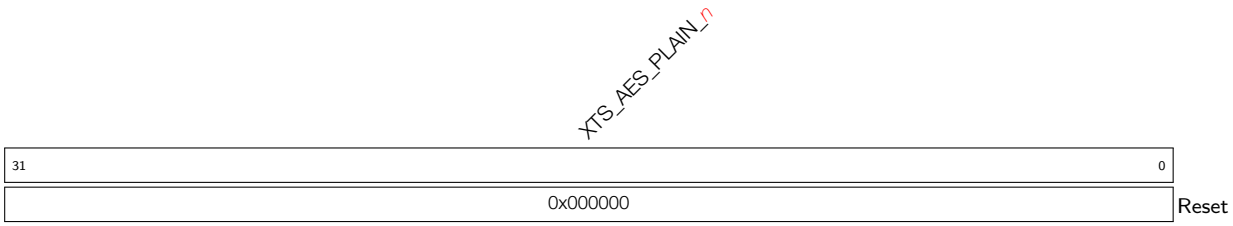
The addresses in this section are relative to External Memory Encryption and Decryption base address provided in Table 3-3 in Chapter 3 *System and Memory*.

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
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
Control/Status Registers			
XTS_AES_TRIGGER_REG	Activates AES algorithm	0x004C	WO
XTS_AES_RELEASE_REG	Release control	0x0050	WO
XTS_AES_DESTROY_REG	Destroy control	0x0054	WO
XTS_AES_STATE_REG	Status register	0x0058	RO
Version Register			
XTS_AES_DATE_REG	Version control register	0x005C	RO

17.7 Registers

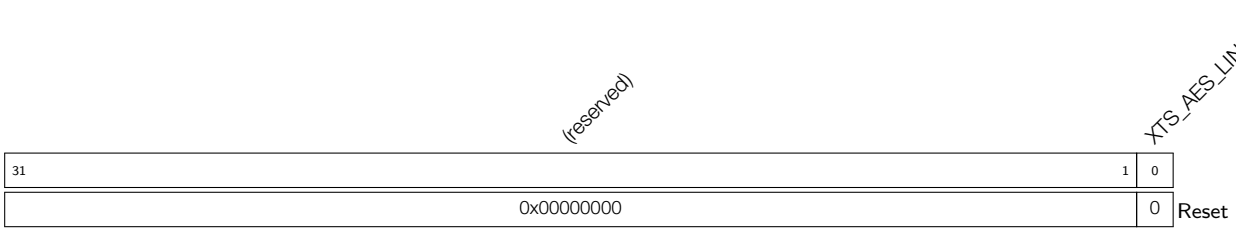
The addresses in this section are relative to External Memory Encryption and Decryption base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Register 17.1. XTS_AES_PLAIN_*n*_REG (*n*: 0-7) (0x0000+4n*)**



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

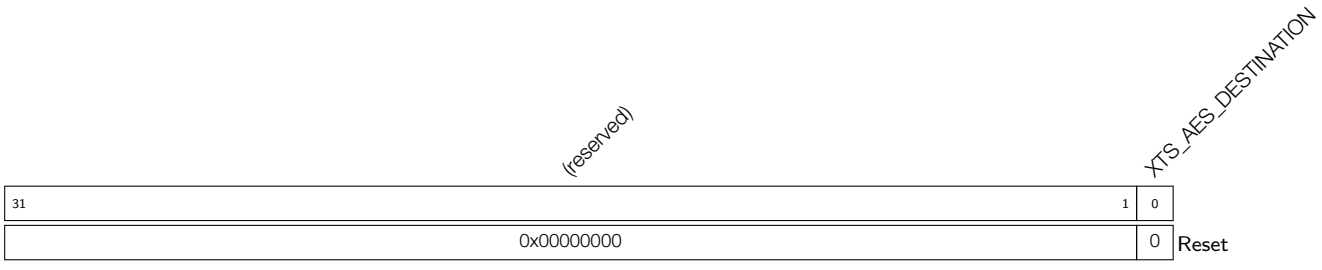
Register 17.2. XTS_AES_LINESIZE_REG (0x0040)



XTS_AES_LINESIZE Configures the data size of one encryption operation. (R/W)

- 0: 16 bytes;
- 1: 32 bytes.

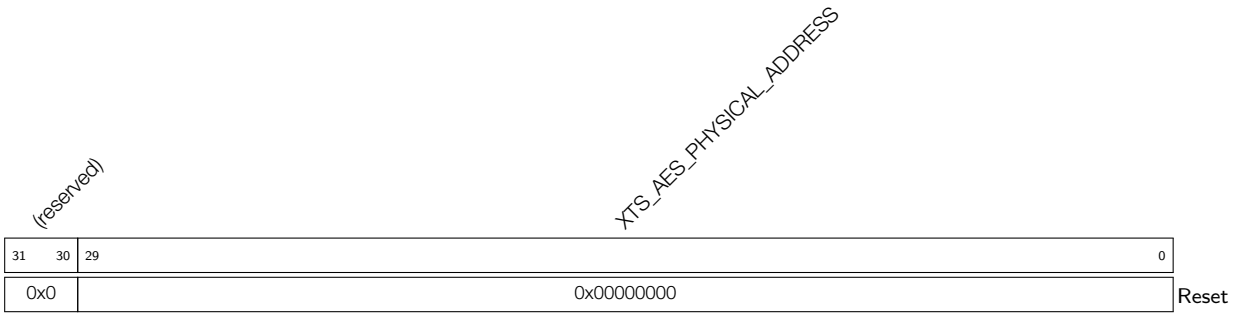
Register 17.3. XTS_AES_DESTINATION_REG (0x0044)



XTS_AES_DESTINATION Configures the type of the external memory. Currently, it must be set to 0, as the Manual Encryption block only supports flash encryption. Errors may occur if users write 1. (R/W)

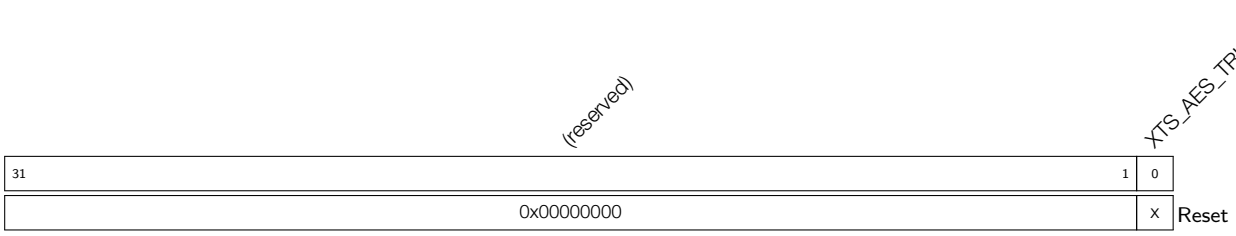
- 0: flash;
- 1: external RAM.

Register 17.4. XTS_AES_PHYSICAL_ADDRESS_REG (0x0048)



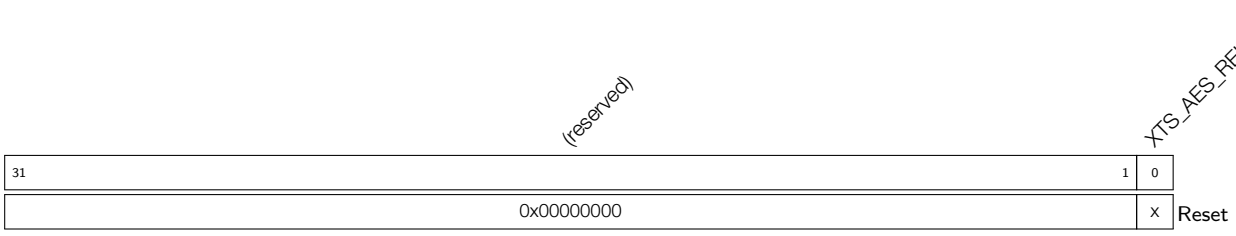
XTS_AES_PHYSICAL_ADDRESS Pysical address. (Note that its value should be within the range of 0x0000_0000 and 0x00FF_FFFF). (R/W)

Register 17.5. XTS_AES_TRIGGER_REG (0x004C)



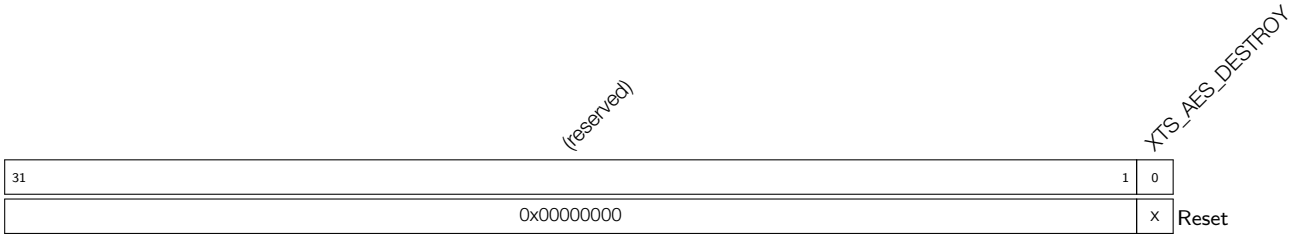
XTS_AES_TRIGGER Write 1 to trigger maunal encryption. (WO)

Register 17.6. XTS_AES_RELEASE_REG (0x0050)



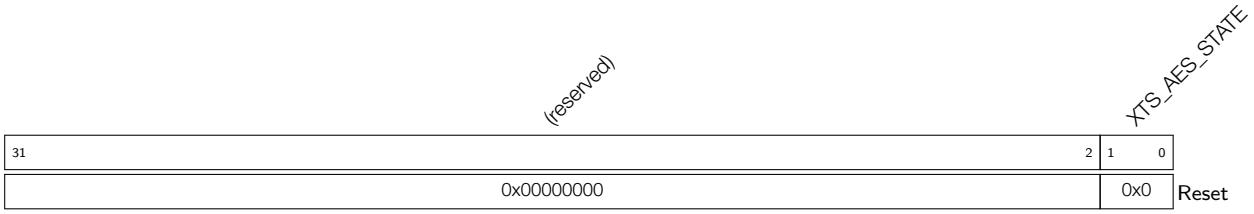
XTS_AES_RELEASE Write 1 to grant SPI1 access to the encrypted result. (WO)

Register 17.7. XTS_AES_DESTROY_REG (0x0054)



XTS_AES_DESTROY Write 1 to destroy encrypted result. (WO)

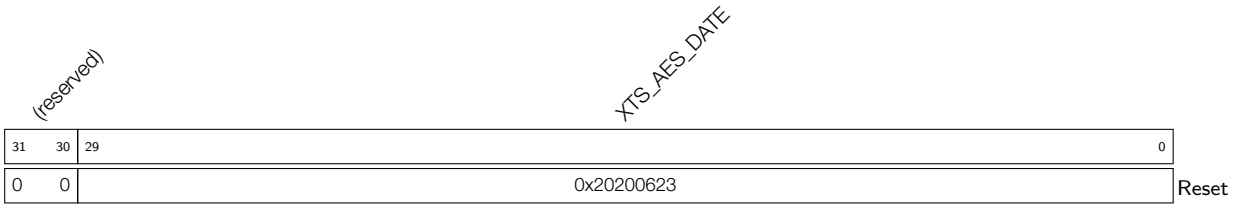
Register 17.8. XTS_AES_STATE_REG (0x0058)



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 17.9. XTS_AES_DATE_REG (0x005C)



XTS_AES_DATE Version control register. (R/W)

18 Random Number Generator (RNG)

18.1 Introduction

The ESP8684 contains a true random number generator, which generates 32-bit random numbers that can be used for cryptographical operations, among other things.

18.2 Features

The random number generator in ESP8684 generates true random numbers, which means random number generated from a physical process, rather than by means of an algorithm. No number generated within the specified range is more or less likely to appear than any other number.

18.3 Functional Description

Every 32-bit value that the system reads from the RNG_DATA_REG register of the random number generator is a true random number. These true random numbers are generated based on the **thermal noise** in the system and the **asynchronous clock mismatch**.

- **Thermal noise** comes from the high-speed ADC or SAR ADC or both. Whenever the high-speed ADC or SAR ADC is enabled, bit streams will be generated and fed into the random number generator through an XOR logic gate as random bit seeds.
- Internal fast RC oscillator clock RC_FAST_CLK (typically about 17.5 MHz, and adjustable) is an **asynchronous clock** source and it increases the RNG entropy by introducing circuit metastability.

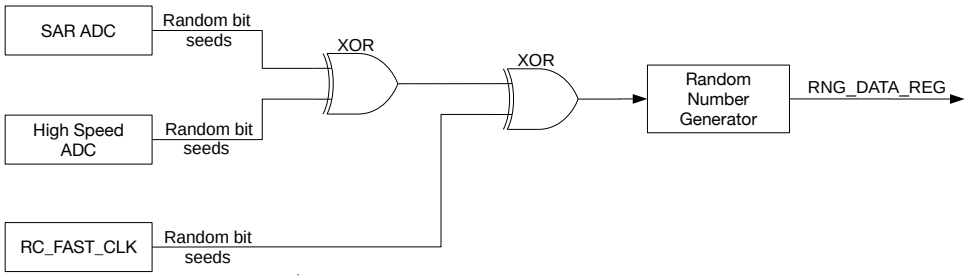


Figure 18-1. Noise Source

When there is noise coming from the SAR ADC, the random number generator is fed with a 1-bit entropy in one ADC sampling cycle. Considering the maximum ADC sample rate is 128 KHz, it is thus advisable to read the RNG_DATA_REG register also at a maximum rate of 128 kHz.

When there is noise coming from the high-speed ADC, the random number generator is fed with a 2-bit entropy in one APB clock cycle, which is normally 80 MHz. Thus, it is advisable to read the RNG_DATA_REG register at a maximum rate of 5 MHz to obtain the maximum entropy.

18.4 Programming Procedure

When using the random number generator, make sure at least either the SAR ADC or high-speed ADC[1] is enabled. Otherwise, pseudo-random numbers will be returned.

- SAR ADC can be enabled by using the DIG ADC controller. For details, please refer to Chapter 23 *On-Chip Sensor and Analog Signal Processing*.
- High-speed ADC is enabled automatically when the wireless module is enabled.
- RC_FAST_CLK[2] is always enabled when the chip is on. Therefore, no need to enable this clock specifically.

Note:

1. Note that, when the wireless module is enabled, the value read from the high-speed ADC can be saturated in some extreme cases, which lowers the entropy. Thus, it is advisable to also enable the SAR ADC as the noise source for the random number generator for such cases.
2. RC_FAST_CLK increases the RNG entropy. However, to ensure maximum entropy, it's recommended to always enable an ADC source as well.

When using the random number generator, read the RNG_DATA_REG register multiple times until sufficient random numbers have been generated. Ensure the rate at which the register is read does not exceed the frequencies described in section 18.3 above.

18.5 Register Summary

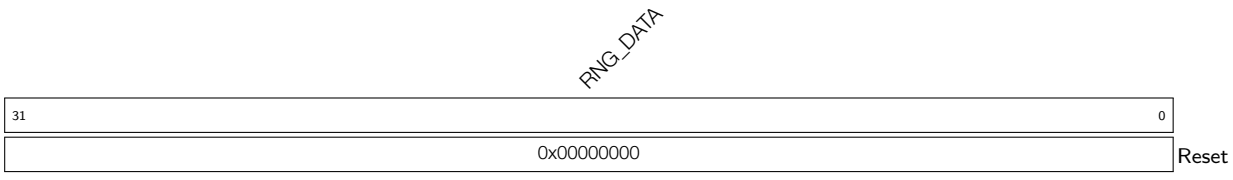
The address in the following table is relative to the random number generator base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Name	Description	Address	Access
RNG_DATA_REG	Random number data	0x00B0	RO

18.6 Register

The address in this section is relative to the random number generator base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Register 18.1. RNG_DATA_REG (0x00B0)



RNG_DATA Random number source. (RO)

19 UART Controller (UART)

19.1 Overview

In embedded system applications, data is required to be transferred in a simple way with minimal system resources. This can be achieved by a Universal Asynchronous Receiver/Transmitter (UART), which flexibly exchanges data with other peripheral devices in full-duplex mode. ESP8684 has two UART controllers compatible with various UART devices. They support Infrared Data Association (IrDA) and RS485 transmission.

Each of the two UART controllers has a group of registers that function identically. In this chapter, the two UART controllers are referred to as UART*n*, in which *n* denotes 0 or 1.

A UART is a character-oriented data link for asynchronous communication between devices. Such communication does not add clock signals to data sent. Therefore, in order to communicate successfully, the transmitter and the receiver must operate at the same baud rate with the same stop bit and parity bit.

A UART data frame usually begins with one start bit, followed by data bits, one parity bit (optional) and one or more stop bits. UART controllers on ESP8684 support various lengths of data bits and stop bits. These controllers also support software and hardware flow control.

19.2 Features

Each UART controller has the following features:

- Full-duplex asynchronous communication
- Configurable baud rate, up to 2.5 Mbaud
- Automatic baud rate detection of input signals
- Data frame format:
  - a START bit
  - data bits, ranging from 5 ~ 8
  - a parity bit
  - stop bits, whose length can be 1, 1.5, 2, or 3 bits
- Special character AT_CMD detection
- Supported protocols: RS485, IrDA
- UART as wake-up source
- Software and hardware flow control
- Three clock sources that can be divided:
  - 40 MHz PLL_40M_CLK
  - internal fast RC oscillator FOSC_CLK
  - external crystal clock XTAL_CLK
- 512 x 8-bit RAM shared by TX FIFOs and RX FIFOs of the two UART controllers

19.3 UART Architecture

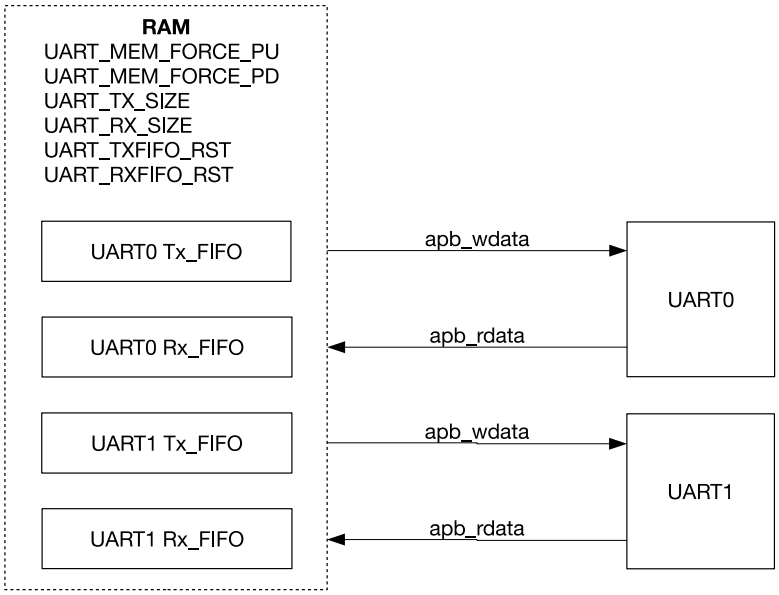


Figure 19-1. UART Architecture Overview

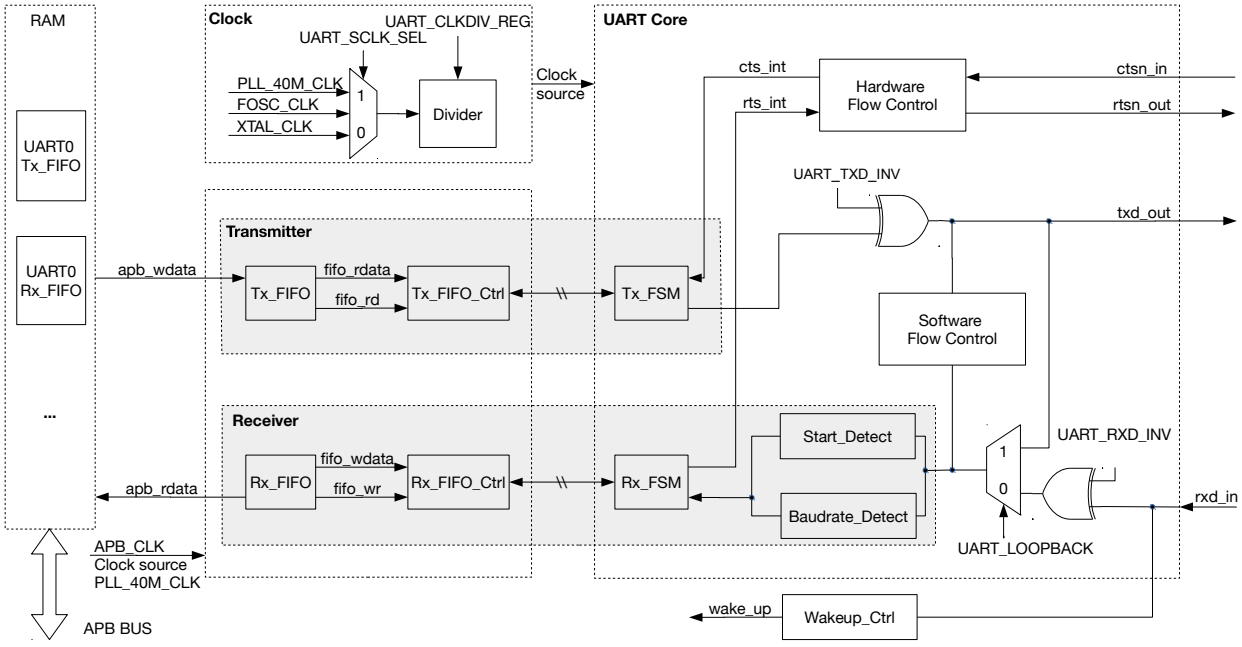


Figure 19-2. UART Architecture

Figure 19-2 shows the basic architecture of a UART controller. A UART controller works in two clock domains, namely APB_CLK domain and Core Clock domain (the UART Core's clock domain).

APB_CLK is derived from PLL_40M_CLK.

The UART Core's clock is derived from the 40 MHz PLL_40M_CLK, FOSC_CLK, or external crystal clock XTAL_CLK (for details, please refer to Chapter 6 *Reset and Clock*), which are selected by configuring UART_SCLK_SEL. The selected clock source is divided by a divider to generate clock signals that drive the

UART Core. The divisor is configured by UART_CLKDIV_REG: UART_CLKDIV for the integral part, and UART_CLKDIV_FRAG for the fractional part.

A UART controller is broken down into two parts according to functions: a transmitter and a receiver.

The transmitter contains a TX FIFO, which buffers data to be sent. Software can write data to Tx_FIFO via the APB bus. Tx_FIFO_Ctrl controls writing and reading Tx_FIFO. When Tx_FIFO is not empty, Tx_FSM reads data bits in the data frame via Tx_FIFO_Ctrl, and converts them into a bitstream. The levels of output signal txd_out can be inverted by configuring UART_TXD_INV field.

The receiver contains a RX FIFO, which buffers data to be processed. The levels of input signal rxd_in can be inverted by configuring UART_RXD_INV field. Baudrate_Detect measures the baud rate of input signal rxd_in by detecting its minimum pulse width. Start_Detect detects the start bit in a data frame. If the start bit is detected, Rx_FSM stores data bits in the data frame into Rx_FIFO by Rx_FIFO_Ctrl. Software can read data from Rx_FIFO via the APB bus.

HW_Flow_Ctrl controls rxd_in and txd_out data flows by standard UART RTS and CTS flow control signals (rtsn_out and ctsn_in). SW_Flow_Ctrl controls data flows by automatically adding special characters to outgoing data and detecting special characters in incoming data. When a UART controller is in Light-sleep mode (see Chapter 9 *Low-power Management (RTC_CNTL)* for more details), Wakeup_Ctrl counts up rising edges of rxd_in. When the number reaches (UART_ACTIVE_THRESHOLD + 2), a wake_up signal is generated and sent to RTC, which then wakes up the ESP8684 chip.

19.4 Functional Description

19.4.1 Clock and Reset

Specific functional blocks of UART controllers are asynchronous. Their register configuration module, TX FIFO and RX FIFO are in APB_CLK domain, while the UART Core that controls transmission and reception is in Core Clock domain. The three clock sources of the UART core, namely PLL_40M_CLK, FOSC_CLK and external crystal clock XTAL_CLK, are selected by configuring UART_SCLK_SEL. The selected clock source is divided by a divider. This divider supports fractional frequency division: UART_SCLK_DIV_NUM field is the integral part, UART_SCLK_DIV_B field is the numerator of the fractional part, and UART_SCLK_DIV_A is the denominator of the fractional part. The divisor ranges from 1 to 256.

When the frequency of the UART Core's clock is higher than the frequency needed to generate baud rate, the UART Core can be clocked at a lower frequency by the divider, in order to reduce power consumption. Usually, the UART Core's clock frequency is lower than the APB_CLK's frequency, and can be divided by the largest divisor value when higher than the frequency needed to generate baud rate. The frequency of the UART Core's clock can also be at most twice higher than the APB_CLK. The clock for the UART transmitter and the UART receiver can be controlled independently. To enable the clock for the UART transmitter, UART_TX_SCLK_EN shall be set; to enable the clock for the UART receiver, UART_RX_SCLK_EN shall be set.

To ensure that the configured register values are synchronized from APB_CLK domain to Core Clock domain, please follow procedures in Section19.5.

To reset the whole UART, please:

- enable the clock for UART RAM by setting SYSTEM_UART_MEM_CLK_EN to 1;
- enable APB_CLK for UART*n* by setting SYSTEM_UART*n*_CLK_EN to 1;

- clear SYSTEM_UART*n*_RST to 0;
- write 1 to UART_RST_CORE;
- write 1 to SYSTEM_UART*n*_RST;
- clear SYSTEM_UART*n*_RST to 0;
- clear UART_RST_CORE to 0.

Note:
It is not recommended to reset the APB clock domain module (SYSTEM_UART*n*_RST) or UART Core (UART_RST_CORE) only.

19.4.2 UART RAM

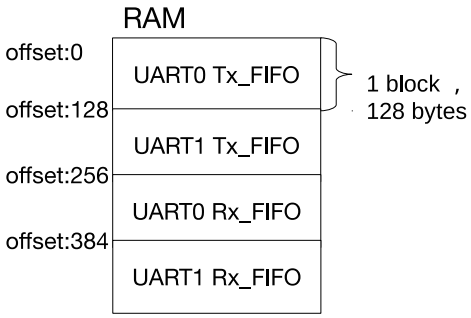


Figure 19-3. UART Controllers Sharing RAM

The two UART controllers on ESP8684 share 512 × 8 bits of FIFO RAM. As Figure 19-3 illustrates, RAM is divided into 4 blocks, each has 128 × 8 bits. Figure 19-3 shows how many RAM blocks are allocated to TX FIFOs and RX FIFOs of the two UART controllers by default. UART*n* Tx_FIFO can be expanded by configuring UART_TX_SIZE, while UART*n* Rx_FIFO can be expanded by configuring UART_RX_SIZE. Some limits are imposed:

- UART0 Tx_FIFO can be increased up to 4 blocks (the whole RAM);
- UART1 Tx_FIFO can be increased up to 3 blocks (from offset 128 to the end address);
- UART0 Rx_FIFO can be increased up to 2 blocks (from offset 256 to the end address);
- UART1 Rx_FIFO cannot be increased.

Please note that starting addresses of all FIFOs are fixed, so expanding one FIFO may take up the default space of other FIFOs. For example, by setting UART_TX_SIZE of UART0 to 2, the size of UART0 Tx_FIFO is increased by 128 bytes (from offset 0 to offset 255). In this case, UART0 Tx_FIFO takes up the default space for UART1 Tx_FIFO, and UART1's transmitting function cannot be used as a result.

When neither of the two UART controllers is active, RAM could enter low-power mode by setting UART_MEM_FORCE_PD.

UART0 Tx_FIFO and UART1 Tx_FIFO are reset by setting UART_TXFIFO_RST. UART0 Rx_FIFO and UART1 Rx_FIFO are reset by setting UART_RXFIFO_RST.

Data to be sent is written to TX FIFO via the APB bus, read automatically and converted from a frame into a bitstream by hardware Tx_FSM; data received is converted from a bitstream into a frame by hardware Rx_FSM,

written into RX FIFO, and then stored into RAM via the APB bus.

The empty signal threshold for Tx_FIFO is configured by setting UART_TXFIFO_EMPTY_THRHD. When data stored in Tx_FIFO is less than UART_TXFIFO_EMPTY_THRHD, a UART_TXFIFO_EMPTY_INT interrupt is generated. The full signal threshold for Rx_FIFO is configured by setting UART_RXFIFO_FULL_THRHD. When data stored in Rx_FIFO is greater than UART_RXFIFO_FULL_THRHD, a UART_RXFIFO_FULL_INT interrupt is generated. In addition, when Rx_FIFO receives more data than its capacity, a UART_RXFIFO_OVF_INT interrupt is generated.

UART*n* can access FIFO via register UART_FIFO_REG. You can put data into TX FIFO by writing UART_RXFIFO_RD_BYTE, and get data in RX FIFO by reading UART_RXFIFO_RD_BYTE.

19.4.3 Baud Rate Generation and Detection

19.4.3.1 Baud Rate Generation

Before a UART controller sends or receives data, the baud rate should be configured by setting corresponding registers. The baud rate generator of a UART controller functions by dividing the input clock source. It can divide the clock source by a fractional amount. The divisor is configured by UART_CLKDIV_REG: UART_CLKDIV for the integral part, and UART_CLKDIV_FRAG for the fractional part. When using the 40 MHz input clock, the UART controller supports a maximum baud rate of 2.5 Mbaud.

The divisor of the baud rate divider is equal to

$$UART\_CLKDIV + \frac{UART\_CLKDIV\_FRAG}{16}$$

meaning that the final baud rate is equal to

$$\frac{INPUT\_FREQ}{UART\_CLKDIV + \frac{UART\_CLKDIV\_FRAG}{16}}$$

where INPUT_FREQ is the frequency of UART Core's source clock. For example, if UART_CLKDIV = 694 and UART_CLKDIV_FRAG = 7, then the divisor value is

$$694 + \frac{7}{16} = 694.4375$$

When UART_CLKDIV_FRAG is 0, the baud rate generator is an integer clock divider where an output pulse is generated every UART_CLKDIV input pulses.

When UART_CLKDIV_FRAG is not 0, the divider is fractional and the output baud rate clock pulses are not strictly uniform. As shown in Figure 19-4, for every 16 output pulses, the generator divides either (UART_CLKDIV + 1) input pulses or UART_CLKDIV input pulses per output pulse. A total of UART_CLKDIV_FRAG output pulses are generated by dividing (UART_CLKDIV + 1) input pulses, and the remaining (16 - UART_CLKDIV_FRAG) output pulses are generated by dividing UART_CLKDIV input pulses.

The output pulses are interleaved as shown in Figure 19-4 below, to make the output timing more uniform:

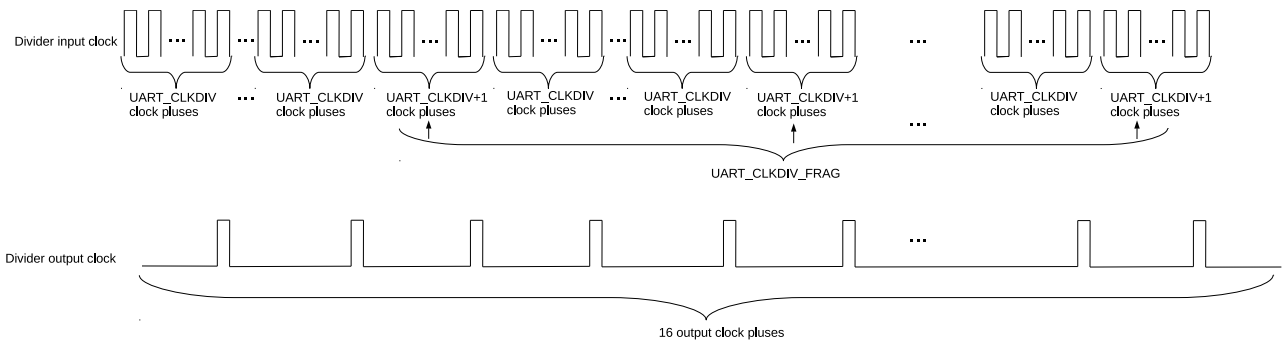


Figure 19-4. UART Controllers Division

To support IrDA (see Section 19.4.6 for details), the fractional clock divider for IrDA data transmission generates clock signals divided by 16 × UART_CLKDIV_REG. This divider works similarly as the one elaborated above: it takes UART_CLKDIV/16 as the integer value and the lowest four bits of UART_CLKDIV as the fractional value.

19.4.3.2 Baud Rate Detection

Automatic baud rate detection (Autobaud) on UARTs is enabled by setting UART_AUTOBAUD_EN. The Baudrate_Detect module shown in Figure 19-2 filters any noise whose pulse width is shorter than UART_GLITCH_FILT.

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

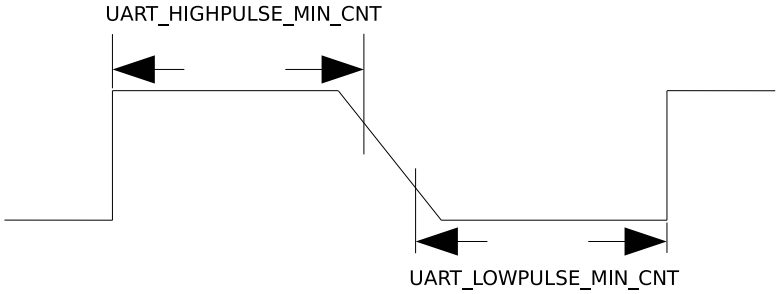


Figure 19-5. The Timing Diagram of Weak UART Signals Along Falling Edges

Baud rate B_{uart} can be determined in the following three ways:

1. Normally, to avoid sampling erroneous data along rising or falling edges in metastable state, which results in inaccuracy of UART_LOWPULSE_MIN_CNT or UART_HIGHPULSE_MIN_CNT, use a weighted average of these two values to eliminate errors. In this case, baud rate is calculated as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_LOWPULSE\_MIN\_CNT} + \text{UART\_HIGHPULSE\_MIN\_CNT} + 2)/2}$$

   where f_{clk} stands for clock frequency.

2. If UART signals are weak along falling edges as shown in Figure 19-5, which leads to inaccurate average of UART_LOWPULSE_MIN_CNT and UART_HIGHPULSE_MIN_CNT, use UART_POSEDGE_MIN_CNT to determine the transmitter's baud rate as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_POSEDGE\_MIN\_CNT} + 1)/2}$$

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

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_NEGEDGE\_MIN\_CNT} + 1)/2}$$

19.4.4 UART Data Frame

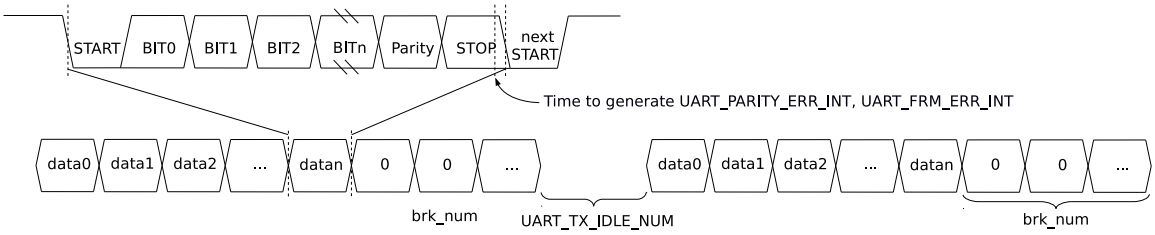


Figure 19-6. Structure of UART Data Frame

Figure 19-6 shows the basic structure of a data frame. A frame starts with one START bit, and ends with STOP bits which can be 1, 1.5, 2 or 3 bits long, configured by UART_STOP_BIT_NUM, UART_DL1_EN and UART_DL0_EN. The START bit is logical low, whereas STOP bits are logical high.

The actual data length can be anywhere between 5 ~ 8 bit, configured by UART_BIT_NUM. When UART_PARITY_EN is set, a parity bit is added after data bits. UART_PARITY is used to choose even parity or odd parity. When the receiver detects a parity bit error in data received, a UART_PARITY_ERR_INT interrupt is generated, and the data received is still stored into RX FIFO. When the receiver detects a framing error (i.e. the sampled stop bit is not 1), a UART_FRM_ERR_INT interrupt is generated, and the data received by default is stored into RX FIFO.

If all data in Tx_FIFO has been sent, a UART_TX_DONE_INT interrupt is generated. After this, if the UART_TXD_BRK bit is set, then the transmitter will send several NULL characters in which the TX data line is logical low. The number of NULL characters is configured by UART_TX_BRK_NUM. Once the transmitter has sent all NULL characters, a UART_TX_BRK_DONE_INT interrupt is generated. The minimum interval between data frames can be configured using UART_TX_IDLE_NUM. If the transmitter stays idle for UART_TX_IDLE_NUM or more time, a UART_TX_BRK_IDLE_DONE_INT interrupt is generated.

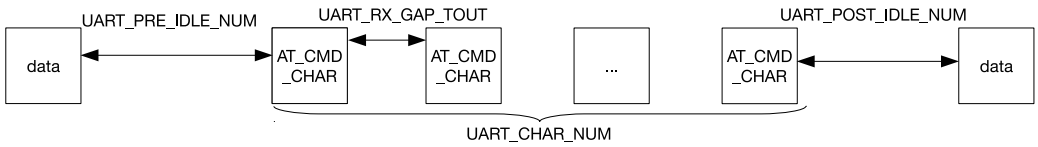


Figure 19-7. AT_CMD Character Structure

Figure 19-7 is the structure of a special character AT_CMD. If the receiver constantly receives AT_CMD_CHAR and the following conditions are met, a UART_AT_CMD_CHAR_DET_INT interrupt is generated.

- The interval between the first AT_CMD_CHAR and the last non-AT_CMD_CHAR character is at least UART_PRE_IDLE_NUM cycles.
- The interval between two AT_CMD_CHAR characters is less than UART_RX_GAP_TOUT cycles.
- The number of AT_CMD_CHAR characters is equal to or greater than UART_CHAR_NUM.
- The interval between the last AT_CMD_CHAR character and next non-AT_CMD_CHAR character is at least UART_POST_IDLE_NUM cycles.

19.4.5 RS485

The two UART controllers support RS485 protocol. This protocol uses differential signals to transmit data, so it can communicate over longer distances at higher bit rates than RS232. RS485 has two-wire half-duplex mode and four-wire full-duplex mode. UART controllers support two-wire half-duplex transmission and bus snooping. In a two-wire RS485 multidrop network, there can be 32 slaves at most.

19.4.5.1 Driver Control

As shown in Figure 19-8, in a two-wire multidrop network, an external RS485 transceiver is needed for differential to single-ended conversion. A RS485 transceiver contains a driver (D) and a receiver (R). When a UART controller is not in transmitter mode, the connection to the differential line can be broken by disabling the driver (D). When DE is 1, the driver is enabled; when DE is 0, the driver is disabled.

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

DE can be controlled by either software or hardware. To reduce the cost of software, in our design, DE is controlled by hardware (can still be controlled by software). As shown in Figure 19-8, DE is connected to dtrn_out of UART (please refer to Section 19.4.8.1 for more details).

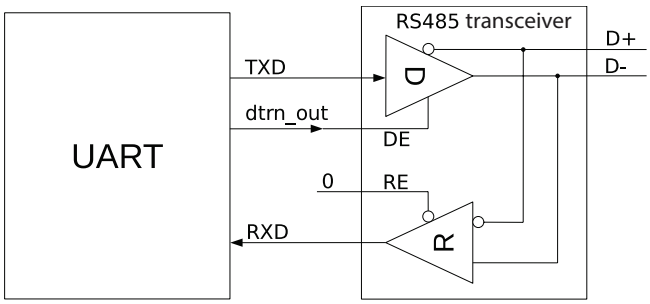


Figure 19-8. Driver Control Diagram in RS485 Mode

19.4.5.2 Turnaround Delay

By default, the two UART controllers work in receiver mode. When a UART controller is switched from transmitter mode to receiver mode, the RS485 protocol requires a turnaround delay of one cycle after the stop bit for stable data transmission. The UART transmitter supports adding a turnaround delay of one cycle before the start bit or after the stop bit. When UART_DL0_EN is set, a turnaround delay of one cycle is added before the start bit; when UART_DL1_EN is set, a turnaround delay of one cycle is added after the stop bit.

19.4.5.3 Bus Snooping

In a two-wire multidrop network, UART controllers support bus snooping if RE of the external RS485 transceiver is 0. By default, a UART controller is not allowed to transmit and receive data simultaneously. If UART_RS485TX_RX_EN is set and the external RS485 transceiver is configured as in Figure 19-8, a UART controller may receive data in transmitter mode and snoop the bus. If UART_RS485RXBY_TX_EN is set, a UART controller may transmit data in receiver mode.

The two UART controllers can snoop data sent by themselves. In transmitter mode, when a UART controller detects a collision between data sent and data received, a UART_RS485_CLASH_INT is generated; when a UART controller detects a framing error, a UART_RS485_FRM_ERR_INT interrupt is generated; when a UART controller detects a polarity error, a UART_RS485_PARITY_ERR_INT is generated.

19.4.6 IrDA

IrDA protocol consists of three layers, namely the physical layer, the link access protocol, and the link management protocol. The two UART controllers implement IrDA's physical layer. In IrDA encoding, a UART controller supports data rates up to 115.2 kbit/s (SIR, or serial infrared mode). As shown in Figure 19-9, the IrDA encoder converts a NRZ (non-return to zero code) signal to a RZI (return to zero inverted code) signal and sends it to the external driver and infrared LED. This encoder uses modulated signals whose pulse width is 3/16 bits high level to indicate logic "0", and low levels to indicate logic "1". The IrDA decoder receives signals from the infrared receiver and converts them to NRZ signals. In most cases, the receiver is high when it is idle, and the encoder output polarity is the opposite of the decoder input polarity. If a low pulse is detected, it indicates that a start bit has been received.

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

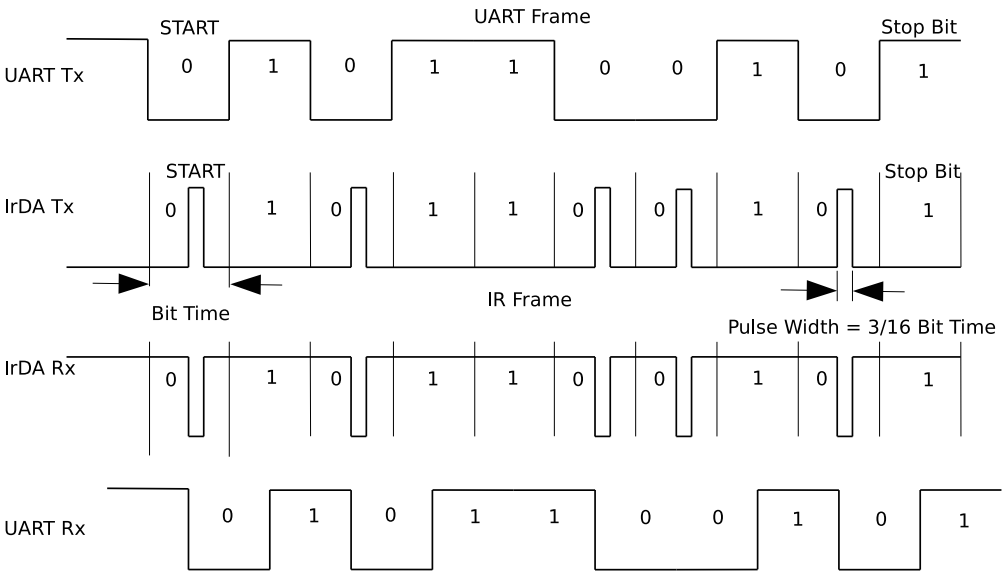


Figure 19-9. The Timing Diagram of Encoding and Decoding in SIR mode

The IrDA transceiver is half-duplex, meaning that it cannot send and receive data simultaneously. As shown in Figure 19-10, IrDA function is enabled by setting UART_IRDA_EN. When UART_IRDA_TX_EN is set (high), the IrDA transceiver is enabled to send data and not allowed to receive data; when UART_IRDA_TX_EN is reset (low),

the IrDA transceiver is enabled to receive data and not allowed to send data.

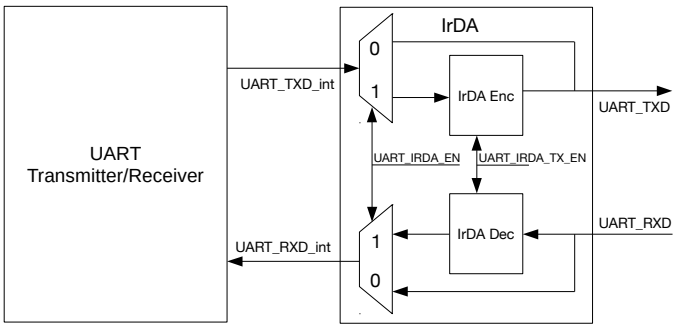


Figure 19-10. IrDA Encoding and Decoding Diagram

19.4.7 Wake-up

UART0 and UART1 can be set as wake-up source. When a UART controller is in Light-sleep mode, Wakeup_Ctrl counts up the rising edges of rxd_in. When the number of rising edges is greater than (UART_ACTIVE_THRESHOLD + 2), a wake_up signal is generated and sent to RTC, which then wakes up ESP8684.

19.4.8 Flow Control

UART controllers have two ways to control data flow, namely hardware flow control and software flow control. Hardware flow control is achieved using output signal rtsn_out and input signal dsrn_in. Software flow control is achieved by inserting special characters in data flow sent and detecting special characters in data flow received.

19.4.8.1 Hardware Flow Control

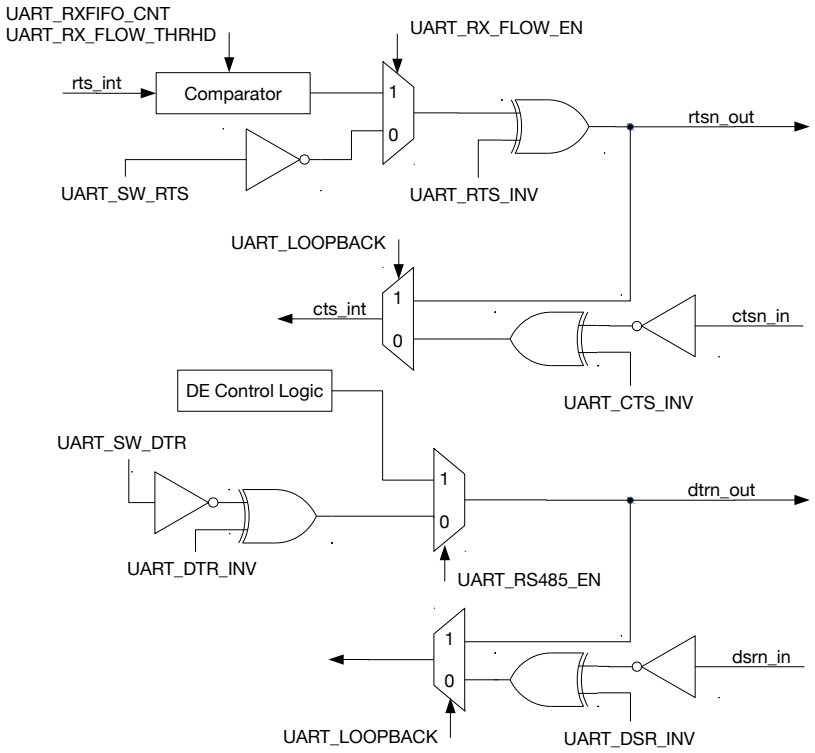


Figure 19-11. Hardware Flow Control Diagram

Figure 19-11 shows hardware flow control of a UART controller. Hardware flow control uses output signal rtsn_out and input signal dsrn_in. Figure 19-12 illustrates how these signals are connected between UART on ESP8684 (hereinafter referred to as IU0) and the external UART (hereinafter referred to as EU0).

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

- Software control: Enter this mode by clearing UART_RX_FLOW_EN to 0. In this mode, the level of rtsn_out is changed by configuring UART_SW_RTS.
- Hardware control: Enter this mode by setting UART_RX_FLOW_EN to 1. In this mode, rtsn_out is pulled high when data in Rx_FIFO exceeds UART_RX_FLOW_THRHD.

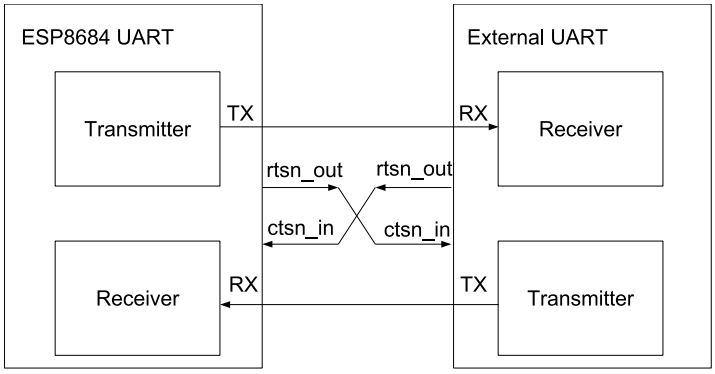


Figure 19-12. Connection between Hardware Flow Control Signals

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

If dtrn_out of IU0 is high, it indicates that IU0 is ready to transmit data. dtrn_out is generated by configuring the UART_SW_DTR field. When the IU0 transmitter detects a edge change of dsrn_in, a UART_DSR_CHG_INT interrupt is generated. After this interrupt is detected, software can obtain the level of input signal dsrn_in by reading UART_DSRN. If dsrn_in is high, it indicates that EU0 is ready to transmit data.

In a two-wire RS485 multidrop network enabled by setting UART_RS485_EN, dtrn_out is generated by hardware (i.e. DE control logic in Figure 19-12) and used for transmit/receive turnaround. When data transmission starts, dtrn_out is pulled high and the external driver is enabled; when data transmission completes, dtrn_out is pulled low and the external driver is disabled. Please note that when there is turnaround delay of one cycle added after the stop bit, dtrn_out is pulled low after the delay.

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

19.4.8.2 Software Flow Control

Instead of ctsn_in and rtsn_out signals, software flow control uses XON/XOFF characters to start or stop data transmission. Such flow control is enabled by setting UART_SW_FLOW_CON_EN to 1.

When using software flow control, hardware automatically detects if there are XON/XOFF characters in data flow received, and generate a UART_SW_XOFF_INT or a UART_SW_XON_INT interrupt accordingly. If an XOFF character is detected, the transmitter stops data transmission once the current byte has been transmitted; if an XON character is detected, the transmitter starts data transmission. In addition, software can force the transmitter to stop sending data by setting UART_FORCE_XOFF, or to start sending data by setting UART_FORCE_XON.

Software determines whether to insert flow control characters according to the remaining room in RX FIFO. When UART_SEND_XOFF is set, the transmitter sends an XOFF character configured by UART_XOFF_CHAR after the current byte in transmission; when UART_SEND_XON is set, the transmitter sends an XON character configured by UART_XON_CHAR after the current byte in transmission. If the RX FIFO of a UART controller stores more data than UART_XOFF_THRESHOLD, UART_SEND_XOFF is set by hardware. As a result, the transmitter sends an XOFF character configured by UART_XOFF_CHAR after the current byte in transmission. If the RX FIFO of a UART controller stores less data than UART_XON_THRESHOLD, UART_SEND_XON is set by hardware. As a result, the transmitter sends an XON character configured by UART_XON_CHAR after the current byte in transmission.

19.4.9 UART Interrupts

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

- UART_TX_DONE_INT: Triggered when all data in the transmitter's TX FIFO has been sent.
- UART_TX_BRK_IDLE_DONE_INT: Triggered when the transmitter stays idle for the minimum interval (threshold) after sending the last data bit.
- UART_TX_BRK_DONE_INT: Triggered when the transmitter has sent all NULL characters after all data in TX FIFO had been sent.
- UART_GLITCH_DET_INT: Triggered when the receiver detects a glitch in the middle of the start bit.
- UART_SW_XOFF_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XOFF character.
- UART_SW_XON_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XON character.
- UART_RXFIFO_TOUT_INT: Triggered when the receiver takes more time than UART_RX_TOUT_THRHD to receive one byte.
- UART_BRK_DET_INT: Triggered when the receiver detects a NULL character after stop bits.
- UART_CTS_CHG_INT: Triggered when the receiver detects an edge change of CTSn signals.
- UART_DSR_CHG_INT: Triggered when the receiver detects an edge change of DSRn signals.
- UART_RXFIFO_OVF_INT: Triggered when the receiver receives more data than the capacity of RX FIFO.
- UART_FRM_ERR_INT: Triggered when the receiver detects a framing error.
- UART_PARITY_ERR_INT: Triggered when the receiver detects a parity error.
- UART_TXFIFO_EMPTY_INT: Triggered when TX FIFO stores less data than what UART_TXFIFO_EMPTY_THRHD specifies.
- UART_RXFIFO_FULL_INT: Triggered when the receiver receives more data than what UART_RXFIFO_FULL_THRHD specifies.
- UART_WAKEUP_INT: Triggered when UART is woken up.

19.5 Programming Procedures

19.5.1 Register Type

All UART registers are in APB_CLK domain. According to whether clock domain crossing and synchronization are required, UART registers that can be configured by software are classified into three types, namely immediate registers, synchronous registers, and static registers. Immediate registers are read in APB_CLK domain, and take effect after configured via the APB bus. Synchronous registers are read in Core Clock domain, and take effect after synchronization. Static registers are also read in Core Clock domain, but would not change dynamically. Therefore, for static registers clock domain crossing is not required, and software can turn on and off the clock for the UART transmitter or receiver to ensure that the configuration sampled in Core Clock domain is correct.

19.5.1.1 Synchronous Registers

Read in Core Clock domain, synchronous registers implement the clock domain crossing design to ensure that their values sampled in Core Clock domain are correct. These registers as listed in Table 19-1 are configured as follows:

- Enable register synchronization by clearing UART_UPDATE_CTRL to 0;
- Wait for UART_REG_UPDATE to become 0, which indicates the completion of last synchronization;
- Configure synchronous registers;
- Synchronize the configured values to Core Clock domain by writting 1 to UART_REG_UPDATE.

Table 19-1. UART*n* Synchronous Registers

Register	Field
UART_CLKDIV_REG	UART_CLKDIV_FRAG[3:0]
	UART_CLKDIV[11:0]
UART_CONF0_REG	UART_AUTOBAUD_EN
	UART_ERR_WR_MASK
	UART_TXD_INV
	UART_RXD_INV
	UART_IRDA_EN
	UART_TX_FLOW_EN
	UART_LOOPBACK
	UART_IRDA_RX_INV
	UART_IRDA_TX_EN
	UART_IRDA_WCTL
	UART_IRDA_TX_EN
	UART_IRDA_DPLX
	UART_STOP_BIT_NUM
	UART_BIT_NUM
	UART_PARITY_EN
	UART_PARITY
UART_FLOW_CONF_REG	UART_SEND_XOFF
	UART_SEND_XON
	UART_FORCE_XOFF
	UART_FORCE_XON
	UART_XONOFF_DEL
	UART_SW_FLOW_CON_EN
UART_TXBRK_CONF_REG	UART_RS485_TX_DLY_NUM[3:0]
	UART_RS485_RX_DLY_NUM
	UART_RS485RXBY_TX_EN
	UART_RS485TX_RX_EN
	UART_DL1_EN
	UART_DL0_EN
	UART_RS485_EN

19.5.1.2 Static Registers

Static registers, though also read in Core Clock domain, would not change dynamically when UART controllers are at work, so they do not implement the clock domain crossing design. These registers must be configured when the UART transmitter or receiver is not at work. In this case, software can turn off the clock for the UART

transmitter or receiver, so that static registers are not sampled in their metastable state. When software turns on the clock, the configured values are stable to be correctly sampled. Static registers as listed in Table 19-2 are configured as follows:

- Turn off the clock for the UART transmitter by clearing UART_TX_SCLK_EN, or the clock for the UART receiver by clearing UART_RX_SCLK_EN, depending on which one (transmitter or receiver) is not at work;
- Configure static registers;
- Turn on the clock for the UART transmitter by writing 1 to UART_TX_SCLK_EN, or the clock for the UART receiver by writing 1 to UART_RX_SCLK_EN.

Table 19-2. UART*n* Static Registers

Register	Field
UART_RX_FILT_REG	UART_GLITCH_FILT_EN
	UART_GLITCH_FILT[7:0]
UART_SLEEP_CONF_REG	UART_ACTIVE_THRESHOLD[9:0]
UART_SWFC_CONF0_REG	UART_XOFF_CHAR[7:0]
UART_SWFC_CONF1_REG	UART_XON_CHAR[7:0]
UART_IDLE_CONF_REG	UART_TX_IDLE_NUM[9:0]
UART_AT_CMD_PRECNT_REG	UART_PRE_IDLE_NUM[15:0]
UART_AT_CMD_POSTCNT_REG	UART_POST_IDLE_NUM[15:0]
UART_AT_CMD_GAPTOUT_REG	UART_RX_GAP_TOUT[15:0]
UART_AT_CMD_CHAR_REG	UART_CHAR_NUM[7:0]
	UART_AT_CMD_CHAR[7:0]

19.5.1.3 Immediate Registers

Except those listed in Table 19-1 and Table 19-2, registers that can be configured by software are immediate registers read in APB_CLK domain, such as interrupt and FIFO configuration registers.

19.5.2 Detailed Steps

Figure 19-13 illustrates the process to program UART controllers, namely initialize UART, configure registers, enable the UART transmitter or receiver, and finish data transmission.

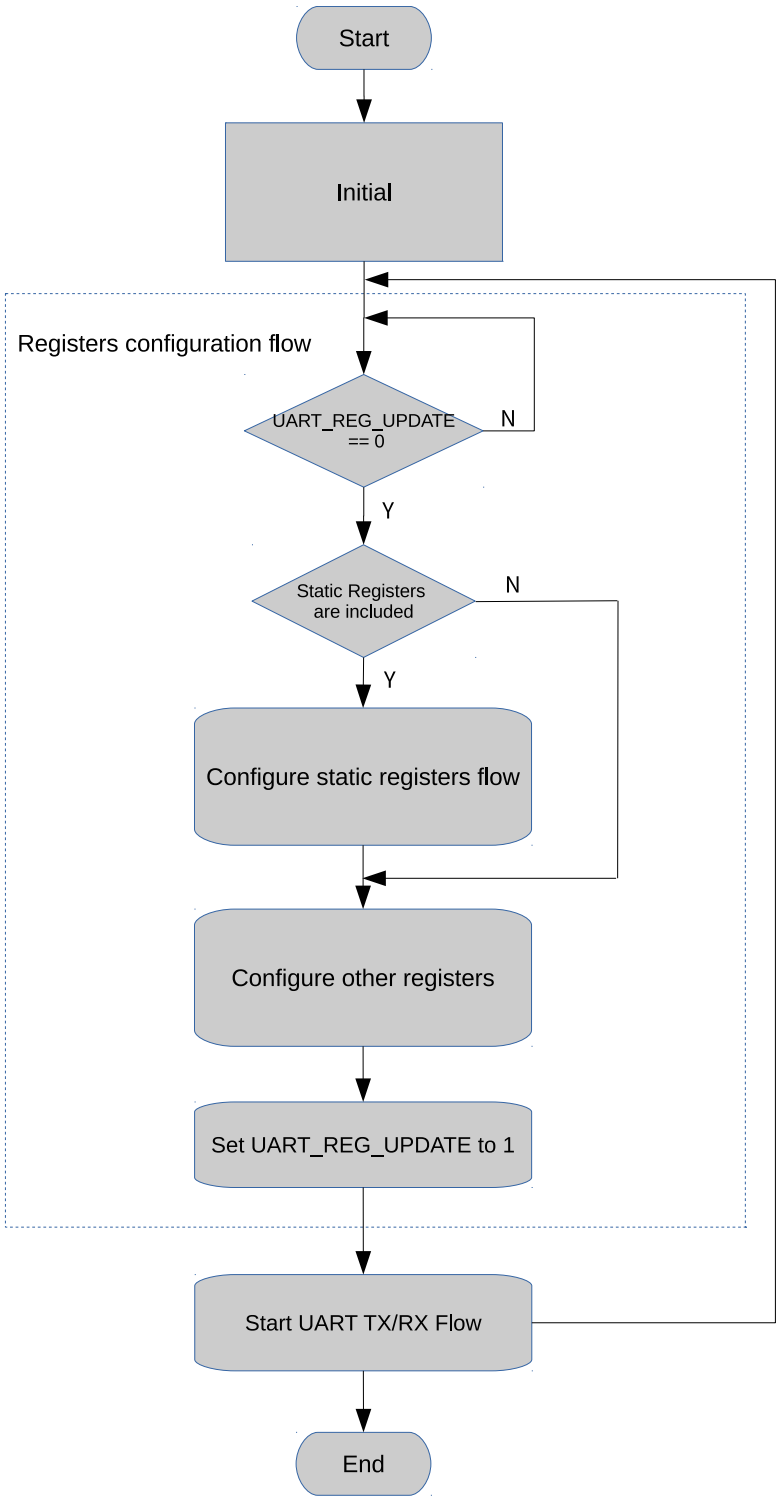


Figure 19-13. UART Programming Procedures

19.5.2.1 Initializing UART*n*

To initialize UART*n*:

- enable the clock for UART RAM by setting SYSTEM_UART_MEM_CLK_EN to 1;
- enable APB_CLK for UART*n* by setting SYSTEM_UART*n*_CLK_EN to 1;
- clear SYSTEM_UART*n*_RST;

- write 1 to UART_RST_CORE;
- write 1 to SYSTEM_UART*n*_RST;
- clear SYSTEM_UART*n*_RST;
- clear UART_RST_CORE;
- enable register synchronization by clearing UART_UPDATE_CTRL.

19.5.2.2 Configuring UART*n* Communication

To configure UART*n* communication:

- wait for UART_REG_UPDATE to become 0, which indicates the completion of last synchronization;
- configure static registers (if any) following Section 19.5.1.2;
- select the clock source via UART_SCLK_SEL;
- configure divisor of the divider via UART_SCLK_DIV_NUM, UART_SCLK_DIV_A, and UART_SCLK_DIV_B;
- configure the baud rate for transmission via UART_CLKDIV and UART_CLKDIV_FRAG;
- configure data length via UART_BIT_NUM;
- configure odd or even parity check via UART_PARITY_EN and UART_PARITY;
- optional steps depending on your application ...
- synchronize the configured values to Core Clock domain by writing 1 to UART_REG_UPDATE.

19.5.2.3 Enabling UART*n*

To enable UART*n* transmitter:

- configure TX FIFO's empty threshold via UART_TXFIFO_EMPTY_THRHD;
- disable UART_TXFIFO_EMPTY_INT interrupt by clearing UART_TXFIFO_EMPTY_INT_ENA;
- write data to be sent to UART_RXFIFO_RD_BYTE;
- clear UART_TXFIFO_EMPTY_INT interrupt by setting UART_TXFIFO_EMPTY_INT_CLR;
- enable UART_TXFIFO_EMPTY_INT interrupt by setting UART_TXFIFO_EMPTY_INT_ENA;
- detect UART_TXFIFO_EMPTY_INT and wait for the completion of data transmission.

To enable UART*n* receiver:

- configure RX FIFO's full threshold via UART_RXFIFO_FULL_THRHD;
- enable UART_RXFIFO_FULL_INT interrupt by setting UART_RXFIFO_FULL_INT_ENA;
- detect UART_TXFIFO_FULL_INT and wait until the RX FIFO is full;
- read data from RX FIFO via UART_RXFIFO_RD_BYTE, and obtain the number of bytes received in RX FIFO via UART_RXFIFO_CNT.

19.6 Register Summary

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

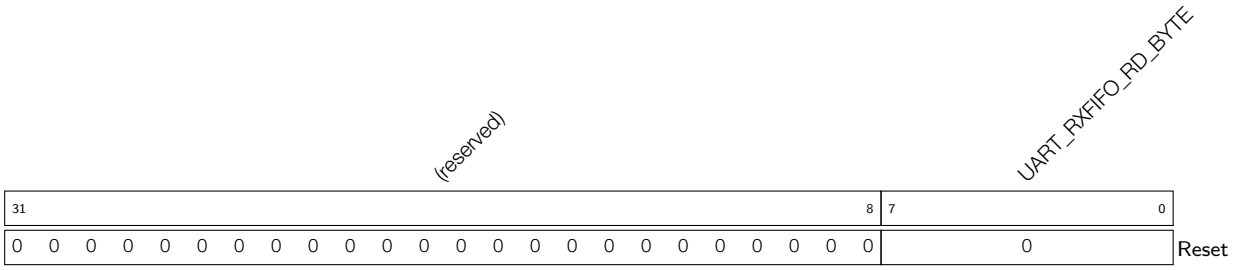
Name	Description	Address	Access
FIFO Configuration			
UART_FIFO_REG	FIFO data register	0x0000	RO
UART_MEM_CONF_REG	UART threshold and allocation configuration	0x0060	R/W
UART Interrupt Register			
UART_INT_RAW_REG	Raw interrupt status	0x0004	R/WTC/SS
UART_INT_ST_REG	Masked interrupt status	0x0008	RO
UART_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UART_INT_CLR_REG	Interrupt clear bits	0x0010	WT
Configuration Register			
UART_CLKDIV_REG	Clock divider configuration	0x0014	R/W
UART_RX_FILT_REG	RX filter configuration	0x0018	R/W
UART_CONF0_REG	Configuration register 0	0x0020	R/W
UART_CONF1_REG	Configuration register 1	0x0024	R/W
UART_FLOW_CONF_REG	Software flow control configuration	0x0034	varies
UART_SLEEP_CONF_REG	Sleep mode configuration	0x0038	R/W
UART_SWFC_CONF0_REG	Software flow control character configuration register 0	0x003C	R/W
UART_SWFC_CONF1_REG	Software flow control character configuration register 1	0x0040	R/W
UART_TXBRK_CONF_REG	TX break character configuration	0x0044	R/W
UART_IDLE_CONF_REG	Frame end idle time configuration	0x0048	R/W
UART_RS485_CONF_REG	RS485 mode configuration	0x004C	R/W
UART_CLK_CONF_REG	UART core clock configuration	0x0078	R/W
Status Register			
UART_STATUS_REG	UART status register	0x001C	RO
UART_MEM_TX_STATUS_REG	TX FIFO write and read offset address	0x0064	RO
UART_MEM_RX_STATUS_REG	RX FIFO write and read offset address	0x0068	RO
UART_FSM_STATUS_REG	UART transmitter and receiver status	0x006C	RO
Autobaud Register			
UART_LOWPULSE_REG	Autobaud minimum low pulse duration register	0x0028	RO
UART_HIGHPULSE_REG	Autobaud minimum high pulse duration register	0x002C	RO
UART_RXD_CNT_REG	Autobaud edge change count register	0x0030	RO
UART_POSPULSE_REG	Autobaud high pulse register	0x0070	RO
UART_NEGPULSE_REG	Autobaud low pulse register	0x0074	RO
AT Escape Sequence Selection Configuration			
UART_AT_CMD_PRECNT_REG	Pre-sequence timing configuration	0x0050	R/W
UART_AT_CMD_POSTCNT_REG	Post-sequence timing configuration	0x0054	R/W
UART_AT_CMD_GAPTOUT_REG	Timeout configuration	0x0058	R/W

Name	Description	Address	Access
UART_AT_CMD_CHAR_REG	AT escape sequence detection configuration	0x005C	R/W
Version Register			
UART_DATE_REG	UART version control register	0x007C	R/W
UART_ID_REG	UART ID register	0x0080	varies

19.7 Registers

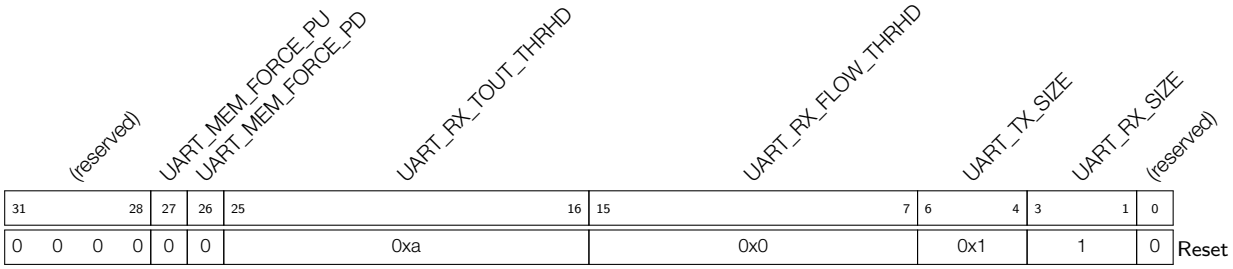
The addresses in this section are relative to UART Controller base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Register 19.1. UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE UART*n* accesses FIFO via this field. (RO)

Register 19.2. UART_MEM_CONF_REG (0x0060)



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

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

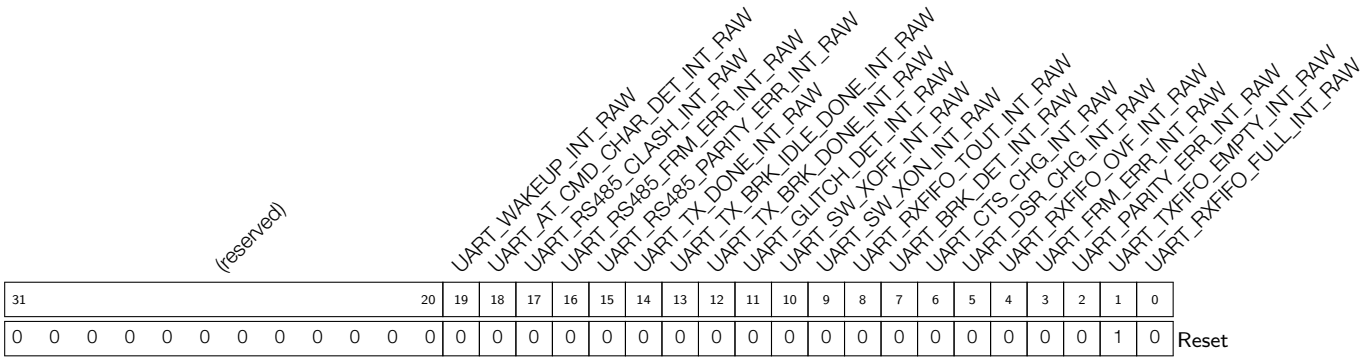
UART_RX_FLOW_THRHD This field is used to configure the maximum amount of data bytes that can be received when hardware flow control works. (R/W)

UART_RX_TOUT_THRHD This field is used to configure the threshold time that the receiver takes to receive one byte, in the unit of bit time (the time it takes to transfer one bit). The UART_RXFIFO_TOUT_INT interrupt will be triggered when the receiver takes more time to receive one byte with UART RX_TOUT_EN set to 1. (R/W)

UART_MEM_FORCE_PD Set this bit to force power down UART RAM. (R/W)

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

Register 19.3. UART_INT_RAW_REG (0x0004)



UART_RXFIFO_FULL_INT_RAW This interrupt raw bit turns to high level when the receiver receives more data than what UART_RXFIFO_FULL_THRHD specifies. (R/WTC/SS)

UART_TXFIFO_EMPTY_INT_RAW This interrupt raw bit turns to high level when the amount of data in TX FIFO is less than what UART_TXFIFO_EMPTY_THRHD specifies. (R/WTC/SS)

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

UART_FRM_ERR_INT_RAW This interrupt raw bit turns to high level when the receiver detects a framing error. (R/WTC/SS)

UART_RXFIFO_OVF_INT_RAW This interrupt raw bit turns to high level when the receiver receives more data than the capacity of RX FIFO. (R/WTC/SS)

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

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

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

UART_RXFIFO_TOUT_INT_RAW This interrupt raw bit turns to high level when the receiver takes more time than UART_RX_TOUT_THRHD to receive a byte. (R/WTC/SS)

UART_SW_XON_INT_RAW This interrupt raw bit turns to high level when the receiver receives an XON character and UART_SW_FLOW_CON_EN is set to 1. (R/WTC/SS)

UART_SW_XOFF_INT_RAW This interrupt raw bit turns to high level when the receiver receives an XOFF character and UART_SW_FLOW_CON_EN is set to 1. (R/WTC/SS)

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

Continued on the next page...

Register 19.3. UART_INT_RAW_REG (0x0004)

Continued from the previous page...

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

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

UART_TX_DONE_INT_RAW This interrupt raw bit turns to high level when the transmitter has sent out all data in FIFO. (R/WTC/SS)

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

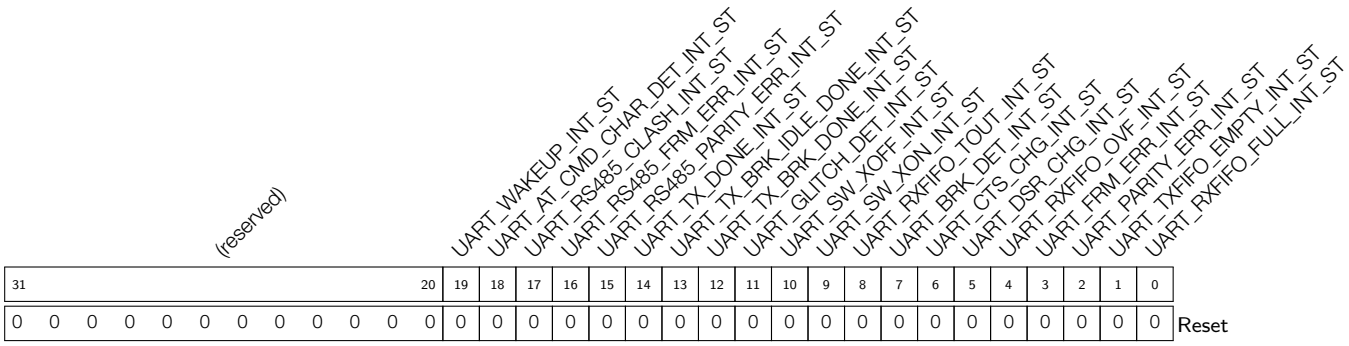
UART_RS485_FRM_ERR_INT_RAW This interrupt raw bit turns to high level when the receiver detects a framing error from the echo of the transmitter in RS485 mode. (R/WTC/SS)

UART_RS485_CLASH_INT_RAW This interrupt raw bit turns to high level when a collision is detected between the transmitter and the receiver in RS485 mode. (R/WTC/SS)

UART_AT_CMD_CHAR_DET_INT_RAW This interrupt raw bit turns to high level when the receiver detects the configured UART_AT_CMD_CHAR. (R/WTC/SS)

UART_WAKEUP_INT_RAW This interrupt raw bit turns to high level when the input RXD edge changes more times than what UART_ACTIVE_THRESHOLD specifies in Light-sleep mode. (R/WTC/SS)

Register 19.4. UART_INT_ST_REG (0x0008)



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

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

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

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

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

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

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

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

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

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

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

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

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

Continued on the next page...

Register 19.4. UART_INT_ST_REG (0x0008)

Continued from the previous page...

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

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

UART_RS485_PARITY_ERR_INT_ST This is the status bit for the UART_RS485_PARITY_ERR_INT interrupt when UART_RS485_PARITY_INT_ENA is set to 1. (RO)

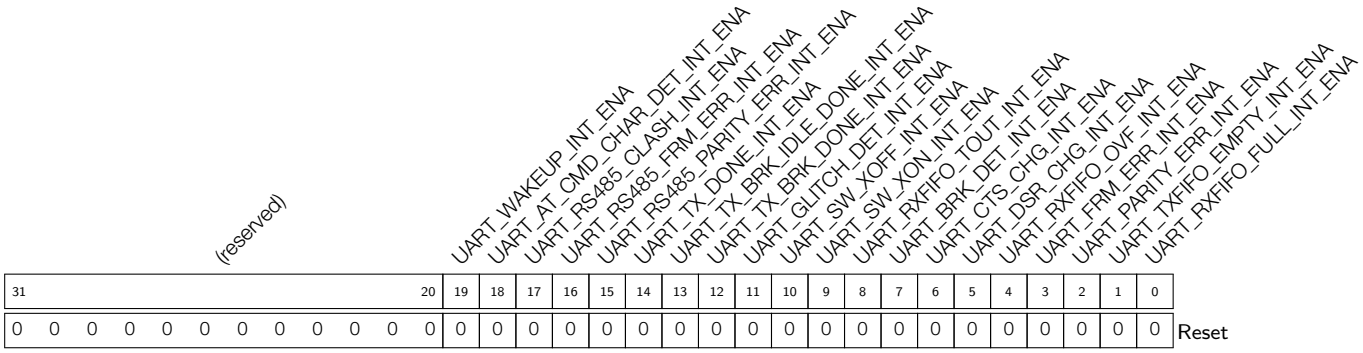
UART_RS485_FRM_ERR_INT_ST This is the status bit for the UART_RS485_FRM_ERR_INT interrupt when UART_RS485_FRM_ERR_INT_ENA is set to 1. (RO)

UART_RS485_CLASH_INT_ST This is the status bit for the UART_RS485_CLASH_INT interrupt when UART_RS485_CLASH_INT_ENA is set to 1. (RO)

UART_AT_CMD_CHAR_DET_INT_ST This is the status bit for the UART_AT_CMD_CHAR_DET_INT interrupt when UART_AT_CMD_CHAR_DET_INT_ENA is set to 1. (RO)

UART_WAKEUP_INT_ST This is the status bit for the UART_WAKEUP_INT interrupt when UART_WAKEUP_INT_ENA is set to 1. (RO)

Register 19.5. UART_INT_ENA_REG (0x000C)



UART_RXFIFO_FULL_INT_ENA This is the enable bit for the UART_RXFIFO_FULL_INT interrupt. (R/W)

UART_TXFIFO_EMPTY_INT_ENA This is the enable bit for the UART_TXFIFO_EMPTY_INT interrupt. (R/W)

UART_PARITY_ERR_INT_ENA This is the enable bit for the UART_PARITY_ERR_INT interrupt. (R/W)

UART_FRM_ERR_INT_ENA This is the enable bit for the UART_FRM_ERR_INT interrupt. (R/W)

UART_RXFIFO_OVF_INT_ENA This is the enable bit for the UART_RXFIFO_OVF_INT interrupt. (R/W)

UART_DSR_CHG_INT_ENA This is the enable bit for the UART_DSR_CHG_INT interrupt. (R/W)

UART_CTS_CHG_INT_ENA This is the enable bit for the UART_CTS_CHG_INT interrupt. (R/W)

UART_BRK_DET_INT_ENA This is the enable bit for the UART_BRK_DET_INT interrupt. (R/W)

UART_RXFIFO_TOUT_INT_ENA This is the enable bit for the UART_RXFIFO_TOUT_INT interrupt. (R/W)

UART_SW_XON_INT_ENA This is the enable bit for the UART_SW_XON_INT interrupt. (R/W)

UART_SW_XOFF_INT_ENA This is the enable bit for the UART_SW_XOFF_INT interrupt. (R/W)

UART_GLITCH_DET_INT_ENA This is the enable bit for the UART_GLITCH_DET_INT interrupt. (R/W)

UART_TX_BRK_DONE_INT_ENA This is the enable bit for the UART_TX_BRK_DONE_INT interrupt. (R/W)

UART_TX_BRK_IDLE_DONE_INT_ENA This is the enable bit for the UART_TX_BRK_IDLE_DONE_INT interrupt. (R/W)

UART_TX_DONE_INT_ENA This is the enable bit for the UART_TX_DONE_INT interrupt. (R/W)

Continued on the next page...

Register 19.5. UART_INT_ENA_REG (0x000C)

Continued from the previous page...

UART_RS485_PARITY_ERR_INT_ENA This is the enable bit for the UART_RS485_PARITY_ERR_INT interrupt. (R/W)

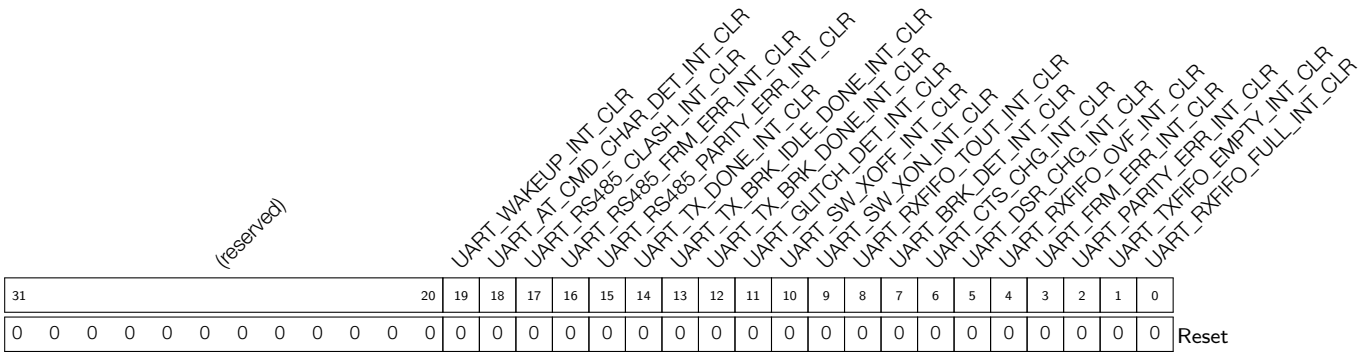
UART_RS485_FRM_ERR_INT_ENA This is the enable bit for the UART_RS485_PARITY_ERR_INT interrupt. (R/W)

UART_RS485_CLASH_INT_ENA This is the enable bit for the UART_RS485_CLASH_INT interrupt. (R/W)

UART_AT_CMD_CHAR_DET_INT_ENA This is the enable bit for the UART_AT_CMD_CHAR_DET_INT interrupt. (R/W)

UART_WAKEUP_INT_ENA This is the enable bit for the UART_WAKEUP_INT interrupt. (R/W)

Register 19.6. UART_INT_CLR_REG (0x0010)



UART_RXFIFO_FULL_INT_CLR Set this bit to clear the UART_THE RXFIFO_FULL_INT interrupt. (WT)

UART_TXFIFO_EMPTY_INT_CLR Set this bit to clear the UART_TXFIFO_EMPTY_INT interrupt. (WT)

UART_PARITY_ERR_INT_CLR Set this bit to clear the UART_PARITY_ERR_INT interrupt. (WT)

UART_FRM_ERR_INT_CLR Set this bit to clear the UART_FRM_ERR_INT interrupt. (WT)

UART_RXFIFO_OVF_INT_CLR Set this bit to clear the UART_UART_RXFIFO_OVF_INT interrupt. (WT)

UART_DSR_CHG_INT_CLR Set this bit to clear the UART_DSR_CHG_INT interrupt. (WT)

UART_CTS_CHG_INT_CLR Set this bit to clear the UART_CTS_CHG_INT interrupt. (WT)

UART_BRK_DET_INT_CLR Set this bit to clear the UART_BRK_DET_INT interrupt. (WT)

UART_RXFIFO_TOUT_INT_CLR Set this bit to clear the UART_RXFIFO_TOUT_INT interrupt. (WT)

UART_SW_XON_INT_CLR Set this bit to clear the UART_SW_XON_INT interrupt. (WT)

UART_SW_XOFF_INT_CLR Set this bit to clear the UART_SW_XOFF_INT interrupt. (WT)

UART_GLITCH_DET_INT_CLR Set this bit to clear the UART_GLITCH_DET_INT interrupt. (WT)

UART_TX_BRK_DONE_INT_CLR Set this bit to clear the UART_TX_BRK_DONE_INT interrupt. (WT)

UART_TX_BRK_IDLE_DONE_INT_CLR Set this bit to clear the UART_TX_BRK_IDLE_DONE_INT interrupt. (WT)

UART_TX_DONE_INT_CLR Set this bit to clear the UART_TX_DONE_INT interrupt. (WT)

UART_RS485_PARITY_ERR_INT_CLR Set this bit to clear the UART_RS485_PARITY_ERR_INT interrupt. (WT)

Continued on the next page...

Register 19.6. UART_INT_CLR_REG (0x0010)

Continued from the previous page...

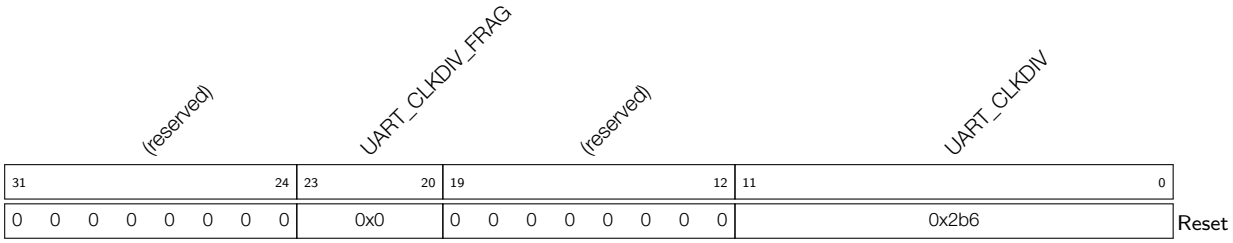
UART_RS485_FRM_ERR_INT_CLR Set this bit to clear the UART_RS485_FRM_ERR_INT interrupt. (WT)

UART_RS485_CLASH_INT_CLR Set this bit to clear the UART_RS485_CLASH_INT interrupt. (WT)

UART_AT_CMD_CHAR_DET_INT_CLR Set this bit to clear the UART_AT_CMD_CHAR_DET_INT interrupt. (WT)

UART_WAKEUP_INT_CLR Set this bit to clear the UART_WAKEUP_INT interrupt. (WT)

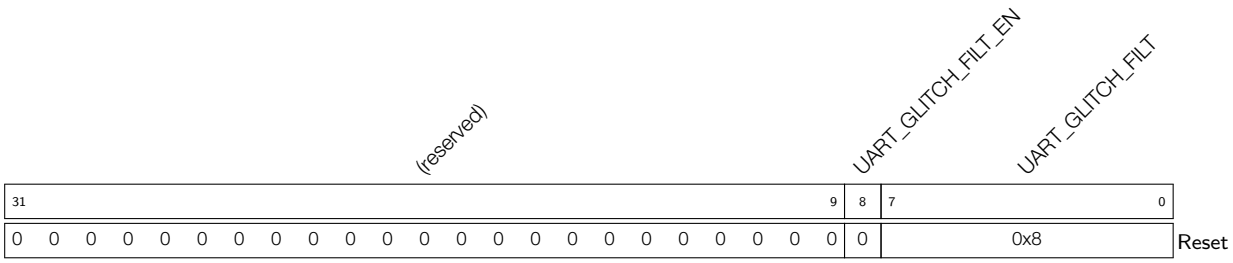
Register 19.7. UART_CLKDIV_REG (0x0014)



UART_CLKDIV The integral part of the frequency divisor. (R/W)

UART_CLKDIV_FRAG The fractional part of the frequency divisor. (R/W)

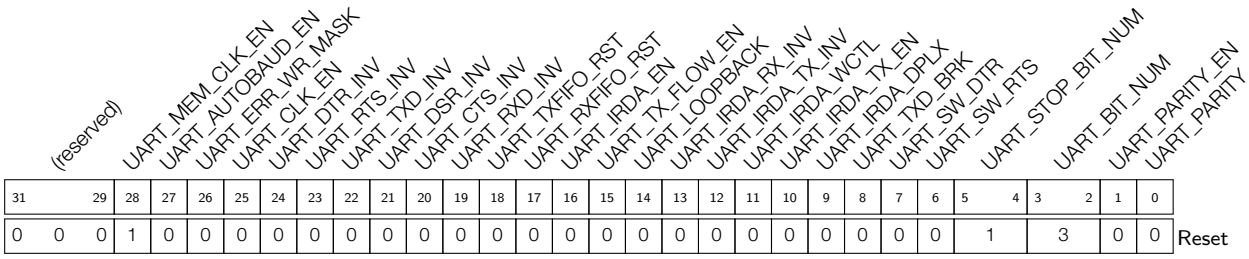
Register 19.8. UART_RX_FILT_REG (0x0018)



UART_GLITCH_FILT When input pulse width is lower than this value, the pulse is ignored. (R/W)

UART_GLITCH_FILT_EN Set this bit to enable RX signal filter. (R/W)

Register 19.9. UART_CONF0_REG (0x0020)



UART_PARITY This bit is used to configure the parity check mode. (R/W)

UART_PARITY_EN Set this bit to enable UART parity check. (R/W)

UART_BIT_NUM This field is used to set the length of data. (R/W)

UART_STOP_BIT_NUM This field is used to set the length of stop bit. (R/W)

UART_SW_RTS This bit is used to configure the software RTS signal which is used in software flow control. (R/W)

UART_SW_DTR This bit is used to configure the software DTR signal which is used in software flow control. (R/W)

UART_TXD_BRK Set this bit to enbale the transmitter to send NULL characters when the process of sending data is done. (R/W)

UART_IRDA_DPLX Set this bit to enable IrDA loopback mode. (R/W)

UART_IRDA_TX_EN This is the start enable bit for IrDA transmitter. (R/W)

UART_IRDA_WCTL 1: The IrDA transmitter's 11th bit is the same as 10th bit; 0: Set IrDA transmitter's 11th bit to 0. (R/W)

UART_IRDA_TX_INV Set this bit to invert the level of IrDA transmitter. (R/W)

UART_IRDA_RX_INV Set this bit to invert the level of IrDA receiver. (R/W)

UART_LOOPBACK Set this bit to enable UART loopback test mode. (R/W)

UART_TX_FLOW_EN Set this bit to enable flow control function for the transmitter. (R/W)

UART_IRDA_EN Set this bit to enable IrDA protocol. (R/W)

UART_RXFIFO_RST Set this bit to reset the UART RX FIFO. (R/W)

UART_TXFIFO_RST Set this bit to reset the UART TX FIFO. (R/W)

UART_RXD_INV Set this bit to invert the level value of UART RXD signal. (R/W)

UART_CTS_INV Set this bit to invert the level value of UART CTS signal. (R/W)

UART_DSR_INV Set this bit to invert the level value of UART DSR signal. (R/W)

Continued on the next page...

Register 19.9. UART_CONF0_REG (0x0020)

Continued from the previous page...

UART_TXD_INV Set this bit to invert the level value of UART TXD signal. (R/W)

UART_RTS_INV Set this bit to invert the level value of UART RTS signal. (R/W)

UART_DTR_INV Set this bit to invert the level value of UART DTR signal. (R/W)

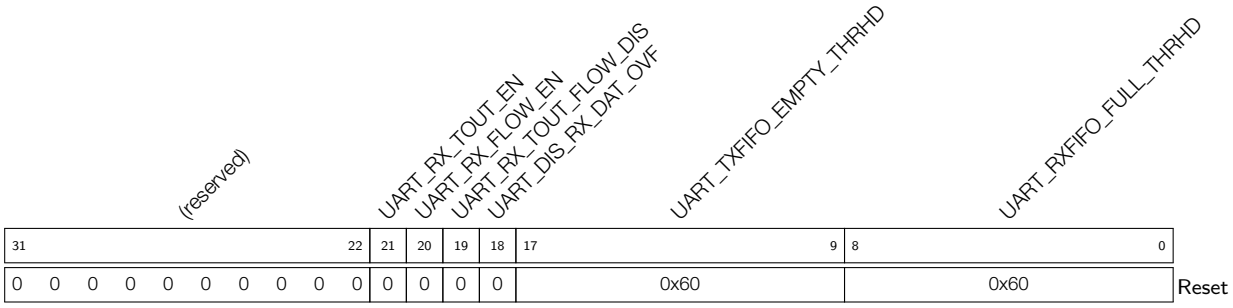
UART_CLK_EN 1: Force clock on for register; 0: Support clock only when application writes registers. (R/W)

UART_ERR_WR_MASK 1: The receiver stops storing data into FIFO when data is wrong; 0: The receiver stores the data even if the received data is wrong. (R/W)

UART_AUTOBAUD_EN This is the enable bit for baud rate detection. (R/W)

UART_MEM_CLK_EN The signal to enable UART RAM clock gating. (R/W)

Register 19.10. UART_CONF1_REG (0x0024)



UART_RXFIFO_FULL_THRHD An UART_RXFIFO_FULL_INT interrupt is generated when the receiver receives more data than the value of this field. (R/W)

UART_TXFIFO_EMPTY_THRHD An UART_TXFIFO_EMPTY_INT interrupt is generated when the number of data bytes in TX FIFO is less than the value of this field. (R/W)

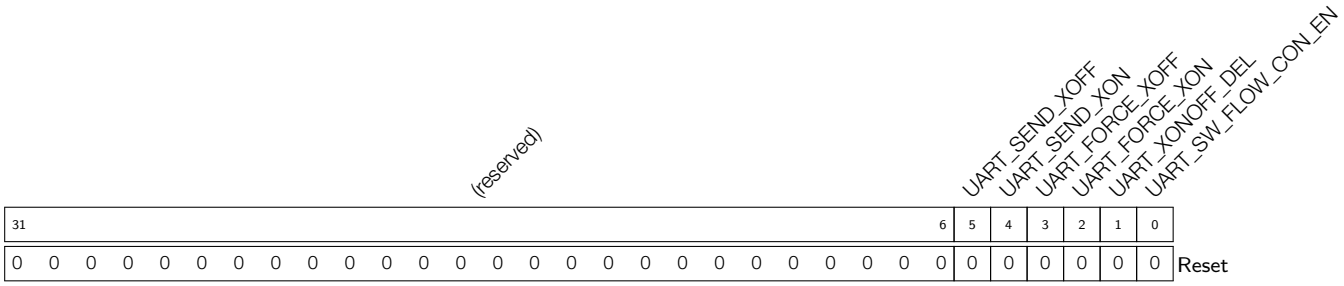
UART_DIS_RX_DAT_OVF Disable UART RX data overflow detection. (R/W)

UART_RX_TOUT_FLOW_DIS Set this bit to stop accumulating idle_cnt when hardware flow control works. (R/W)

UART_RX_FLOW_EN This is the flow enable bit for UART receiver. (R/W)

UART_RX_TOUT_EN This is the enable bit for UART receiver's timeout function. (R/W)

Register 19.11. UART_FLOW_CONF_REG (0x0034)



UART_SW_FLOW_CON_EN Set this bit to enable software flow control. When UART receives flow control characters XON or XOFF, which can be configured by UART_XON_CHAR or UART_XOFF_CHAR respectively, UART_SW_XON_INT or UART_SW_XOFF_INT interrupts can be triggered if enabled. (R/W)

UART_XONOFF_DEL Set this bit to remove flow control characters from the received data. (R/W)

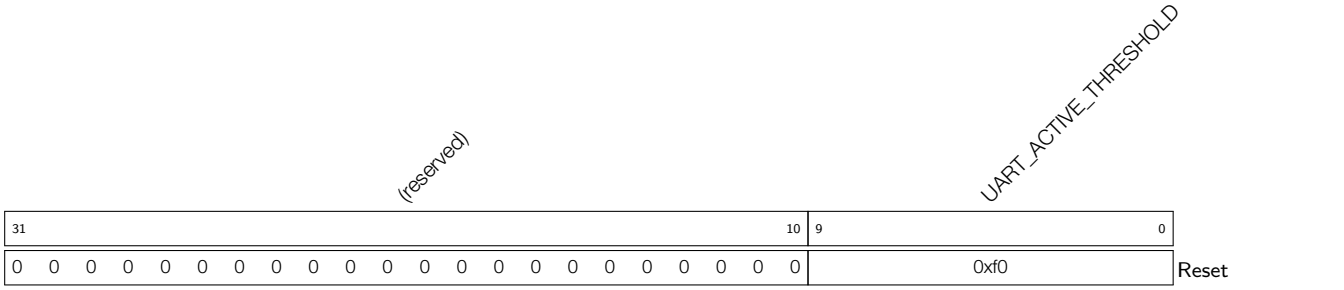
UART_FORCE_XON Set this bit to force the transmitter to send data. (R/W)

UART_FORCE_XOFF Set this bit to stop the transmitter from sending data. (R/W)

UART_SEND_XON Set this bit to send an XON character. This bit is cleared by hardware automatically. (R/W/SS/SC)

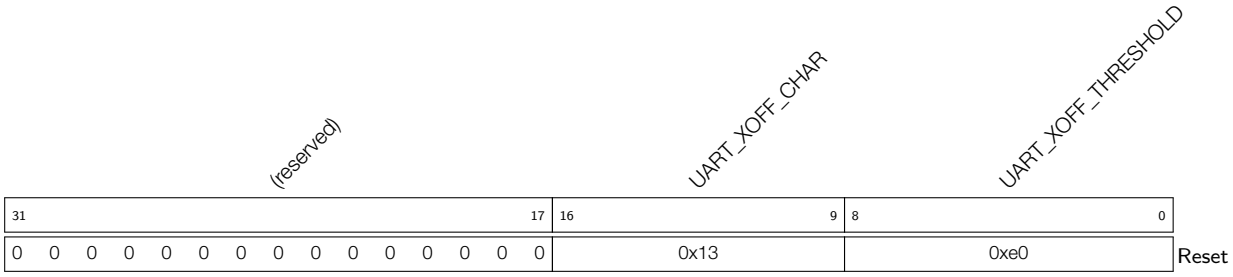
UART_SEND_XOFF Set this bit to send an XOFF character. This bit is cleared by hardware automatically. (R/W/SS/SC)

Register 19.12. UART_SLEEP_CONF_REG (0x0038)



UART_ACTIVE_THRESHOLD UART is activated from Light-sleep mode when the input RXD edge changes more times than the value of this field. (R/W)

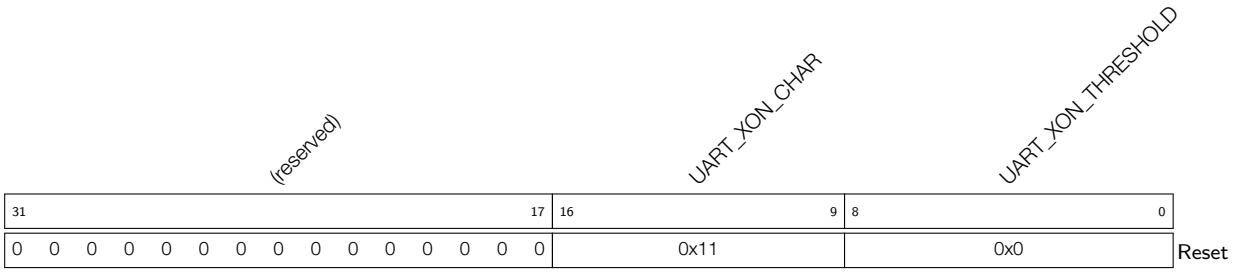
Register 19.13. UART_SWFC_CONF0_REG (0x003C)



UART_XOFF_THRESHOLD When the number of data bytes in RX FIFO is more than the value of this field with UART_SW_FLOW_CON_EN set to 1, the transmitter sends an XOFF character. (R/W)

UART_XOFF_CHAR This field stores the XOFF flow control character. (R/W)

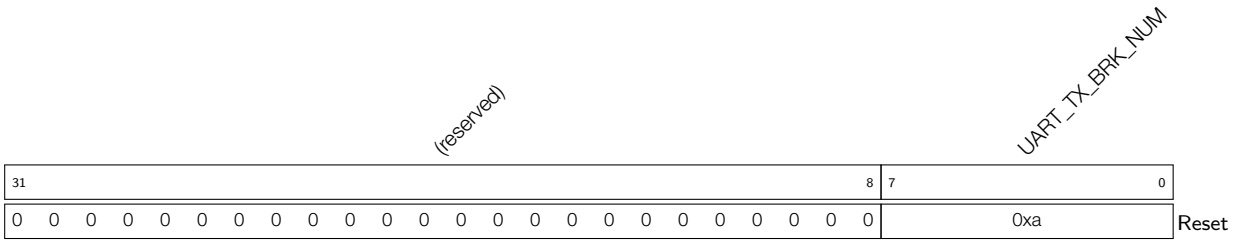
Register 19.14. UART_SWFC_CONF1_REG (0x0040)



UART_XON_THRESHOLD When the number of data bytes in RX FIFO is less than the value of this field with UART_SW_FLOW_CON_EN set to 1, the transmitter sends an XON character. (R/W)

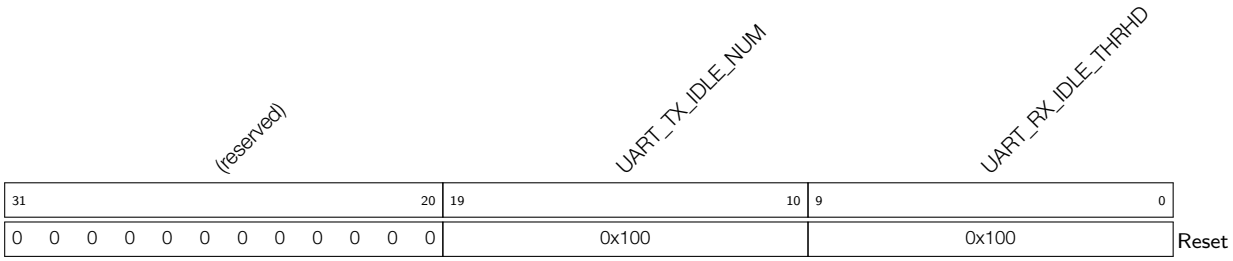
UART_XON_CHAR This field stores the XON flow control character. (R/W)

Register 19.15. UART_TXBRK_CONF_REG (0x0044)



UART_TX_BRK_NUM This field is used to configure the number of 0 to be sent after the process of sending data is done. It is active when UART_TXD_BRK is set to 1. (R/W)

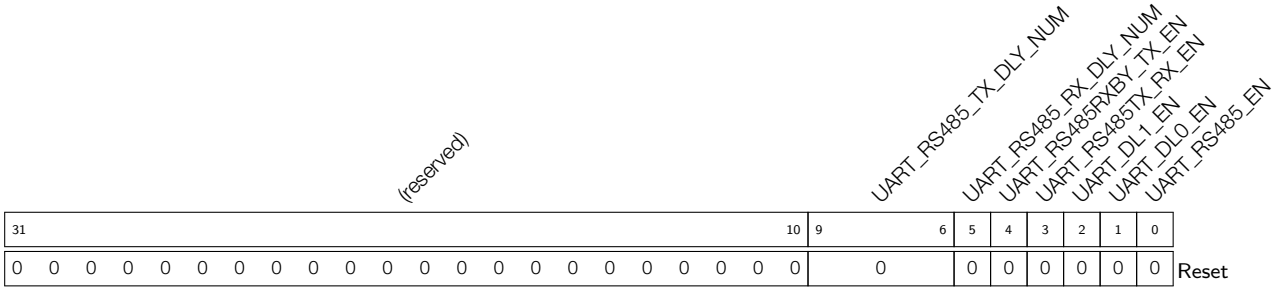
Register 19.16. UART_IDLE_CONF_REG (0x0048)



UART_RX_IDLE_THRHD A frame end signal is generated when the receiver takes more time to receive one byte data than the value of this field, in the unit of bit time (the time it takes to transfer one bit). (R/W)

UART_TX_IDLE_NUM This field is used to configure the duration time between transfers, in the unit of bit time (the time it takes to transfer one bit). (R/W)

Register 19.17. UART_RS485_CONF_REG (0x004C)



UART_RS485_EN Set this bit to choose RS485 mode. (R/W)

UART_DL0_EN Set this bit to delay the stop bit by 1 bit. (R/W)

UART_DL1_EN Set this bit to delay the stop bit by 1 bit. (R/W)

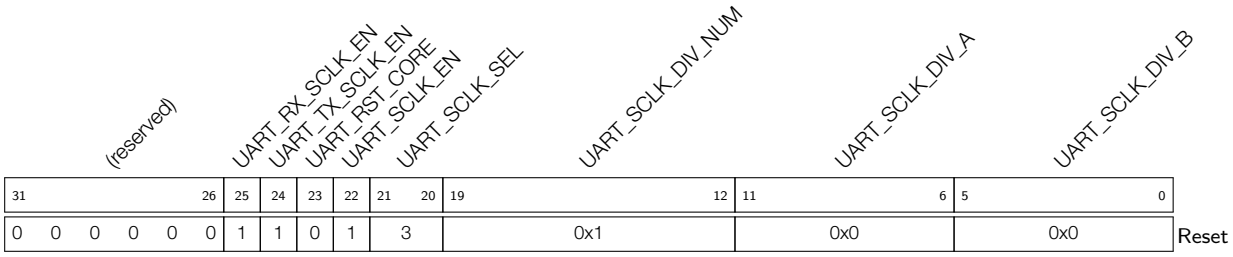
UART_RS485TX_RX_EN Set this bit to enable the receiver could receive data when the transmitter is transmitting data in RS485 mode. (R/W)

UART_RS485RXBY_TX_EN Set this bit to enable RS485 transmitter to send data when RS485 receiver line is busy. (R/W)

UART_RS485_RX_DLY_NUM This bit is used to delay the receiver's internal data signal. (R/W)

UART_RS485_TX_DLY_NUM This field is used to delay the transmitter's internal data signal. (R/W)

Register 19.18. UART_CLK_CONF_REG (0x0078)



UART_SCLK_DIV_B The denominator of the frequency divisor. (R/W)

UART_SCLK_DIV_A The numerator of the frequency divisor. (R/W)

UART_SCLK_DIV_NUM The integral part of the frequency divisor. (R/W)

UART_SCLK_SEL Selects UART clock source. 1: APB_CLK; 2: FOSC_CLK; 3: XTAL_CLK. (R/W)

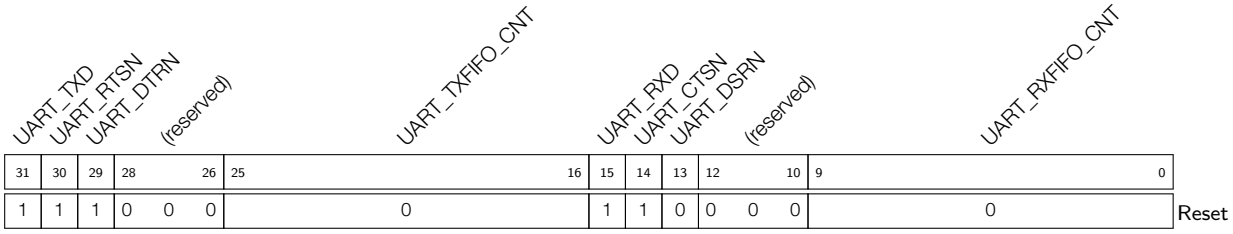
UART_SCLK_EN Set this bit to enable UART TX/RX clock. (R/W)

UART_RST_CORE Write 1 and then write 0 to this bit, to reset UART TX/RX. (R/W)

UART_TX_SCLK_EN Set this bit to enable UART TX clock. (R/W)

UART_RX_SCLK_EN Set this bit to enable UART RX clock. (R/W)

Register 19.19. UART_STATUS_REG (0x001C)



UART_RXFIFO_CNT Stores the number of valid data bytes in RX FIFO. (RO)

UART_DSRN This bit represents the level of the internal UART DSR signal. (RO)

UART_CTSN This bit represents the level of the internal UART CTS signal. (RO)

UART_RXD This bit represents the level of the internal UART RXD signal. (RO)

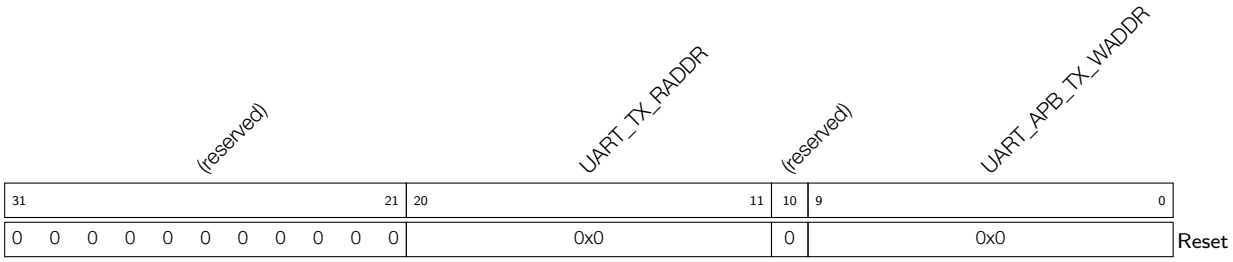
UART_TXFIFO_CNT Stores the number of data bytes in TX FIFO. (RO)

UART_DTRN This bit represents the level of the internal UART DTR signal. (RO)

UART_RTSN This bit represents the level of the internal UART RTS signal. (RO)

UART_TXD This bit represents the level of the internal UART TXD signal. (RO)

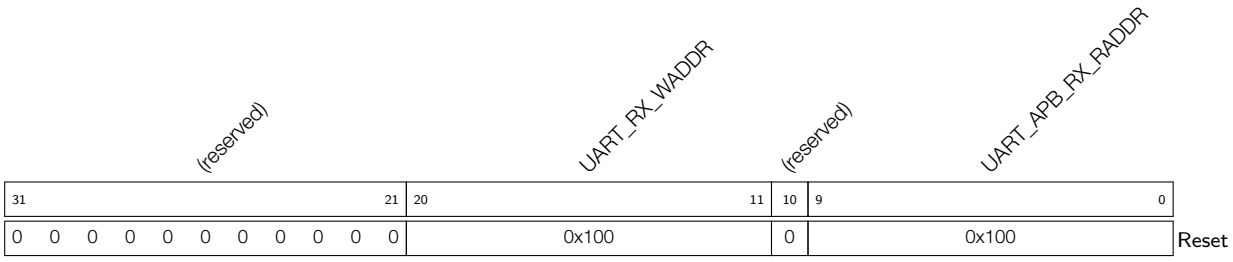
Register 19.20. UART_MEM_TX_STATUS_REG (0x0064)



UART_APB_TX_WADDR This field stores the offset address in TX FIFO when software writes TX FIFO via APB. (RO)

UART_TX_RADDR This field stores the offset address in TX FIFO when TX FSM reads data via Tx_FIFO_Ctrl. (RO)

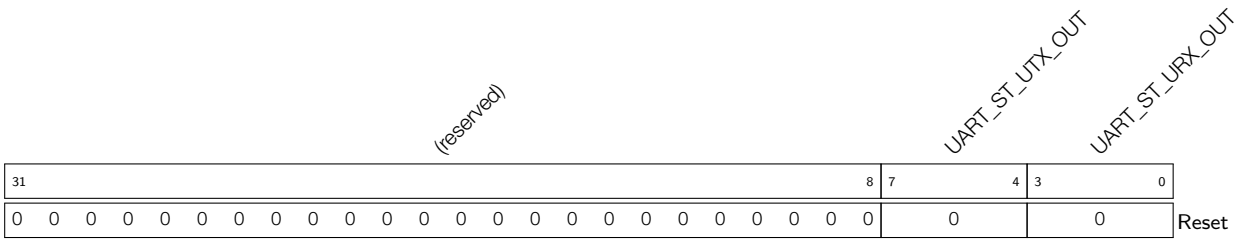
Register 19.21. UART_MEM_RX_STATUS_REG (0x0068)



UART_APB_RX_RADDR This field stores the offset address in RX FIFO when software reads data from RX FIFO via APB. UART0 is 0x100. UART1 is 0x180. (RO)

UART_RX_WADDR This field stores the offset address in RX FIFO when Rx_FIFO_Ctrl writes RX FIFO. UART0 is 0x100. UART1 is 0x180. (RO)

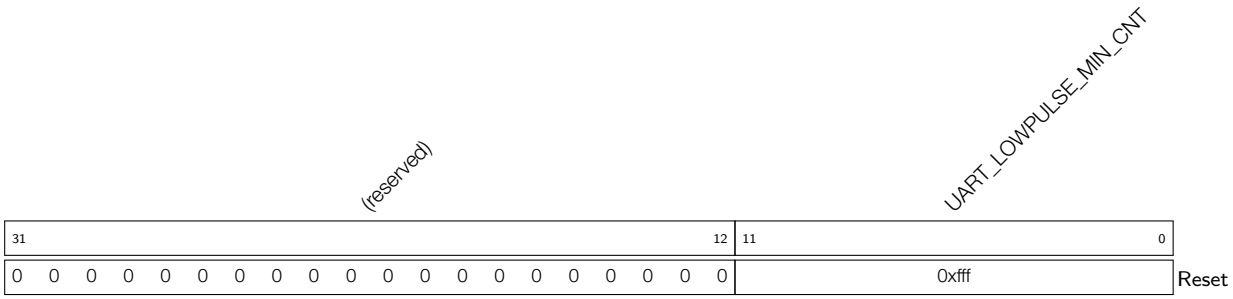
Register 19.22. UART_FSM_STATUS_REG (0x006C)



UART_ST_URX_OUT This is the status field of the receiver. (RO)

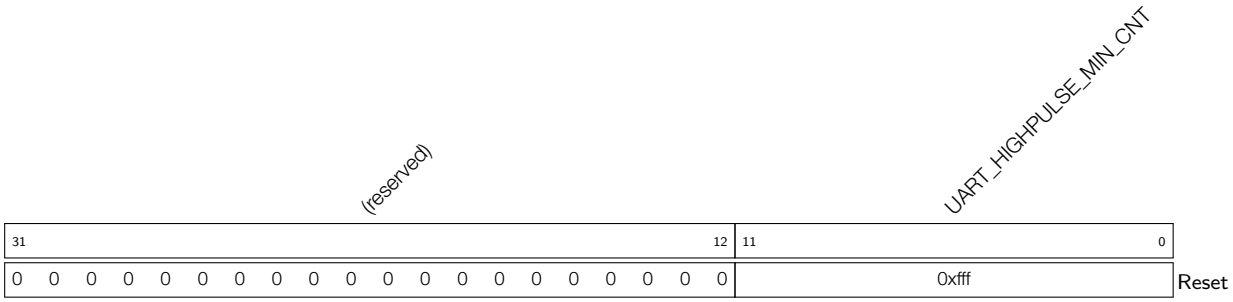
UART_ST_UTX_OUT This is the status field of the transmitter. (RO)

Register 19.23. UART_LOWPULSE_REG (0x0028)



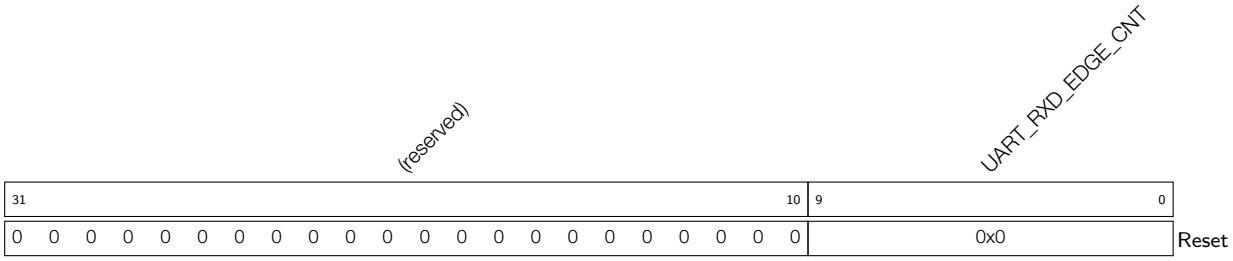
UART_LOWPULSE_MIN_CNT This field stores the value of the minimum duration time of the low level pulse, in the unit of APB_CLK cycles. It is used in baud rate detection. (RO)

Register 19.24. UART_HIGHPULSE_REG (0x002C)



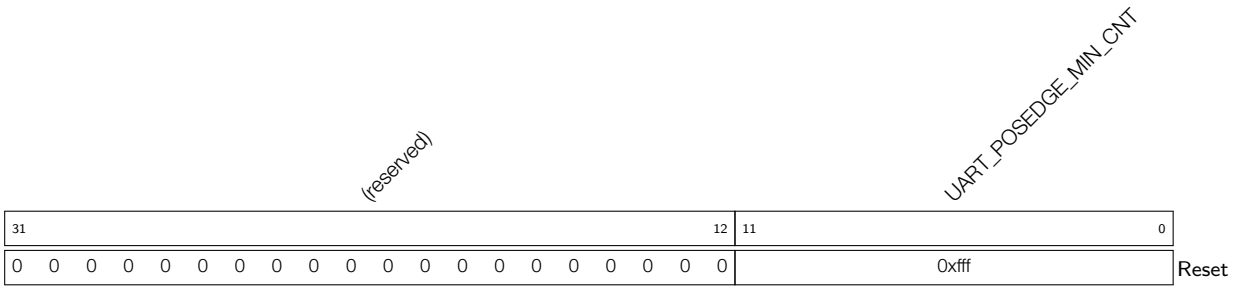
UART_HIGHPULSE_MIN_CNT This field stores the value of the maximum duration time for the high level pulse, in the unit of APB_CLK cycles. It is used in baud rate detection. (RO)

Register 19.25. UART_RXD_CNT_REG (0x0030)



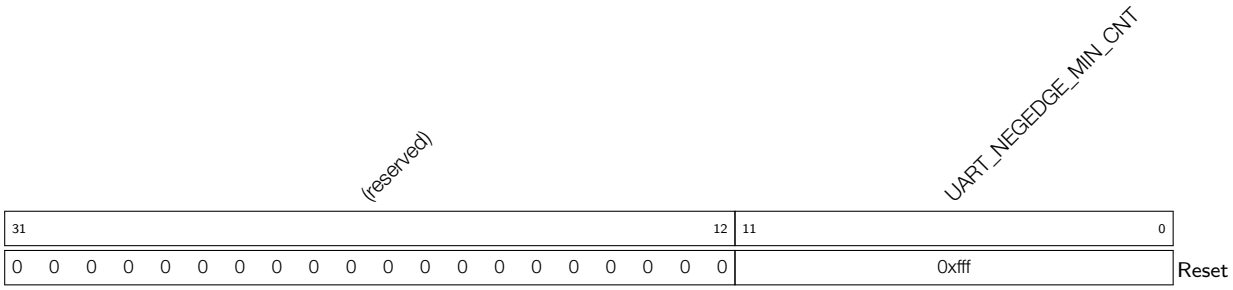
UART_RXD_EDGE_CNT This field stores the count of RXD edge change. It is used in baud rate detection. (RO)

Register 19.26. UART_POSPULSE_REG (0x0070)



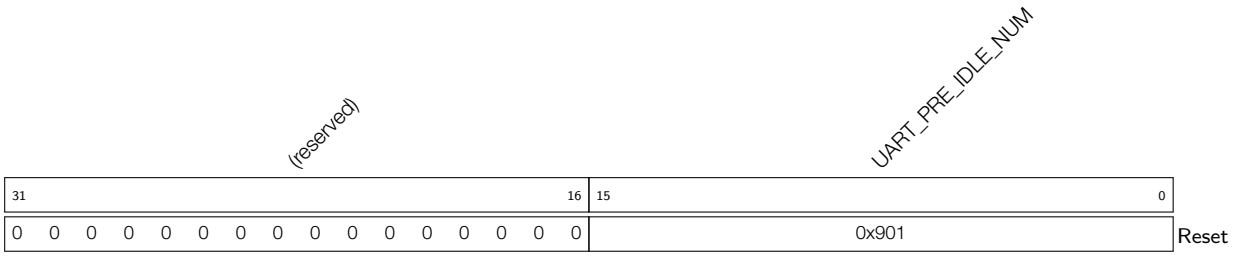
UART_POSEDGE_MIN_CNT This field stores the minimal input clock count between two positive edges. It is used in baud rate detection. (RO)

Register 19.27. UART_NEGPULSE_REG (0x0074)



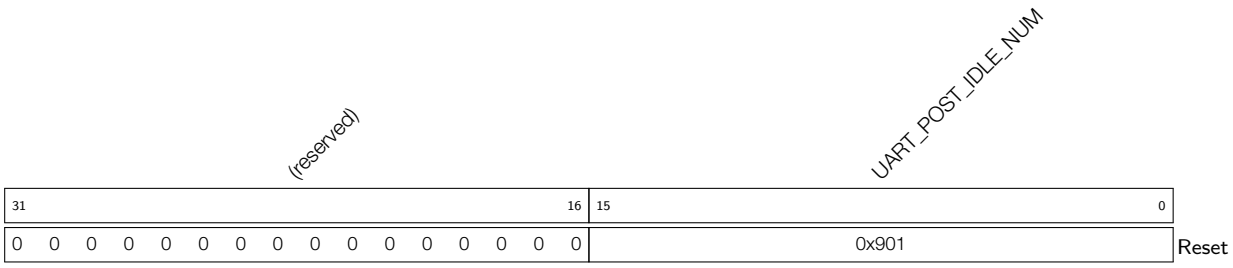
UART_NEGEDGE_MIN_CNT This field stores the minimal input clock count between two negative edges. It is used in baud rate detection. (RO)

Register 19.28. UART_AT_CMD_PRECNT_REG (0x0050)



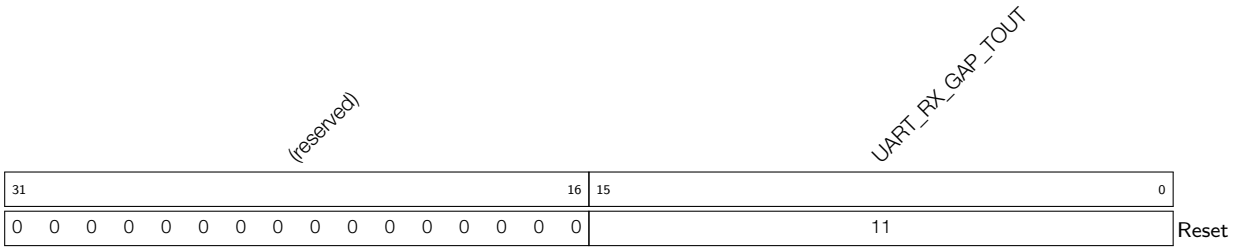
UART_PRE_IDLE_NUM This field is used to configure the idle duration time before the first AT_CMD is received by the receiver, in the unit of bit time (the time it takes to transfer one bit). (R/W)

Register 19.29. UART_AT_CMD_POSTCNT_REG (0x0054)



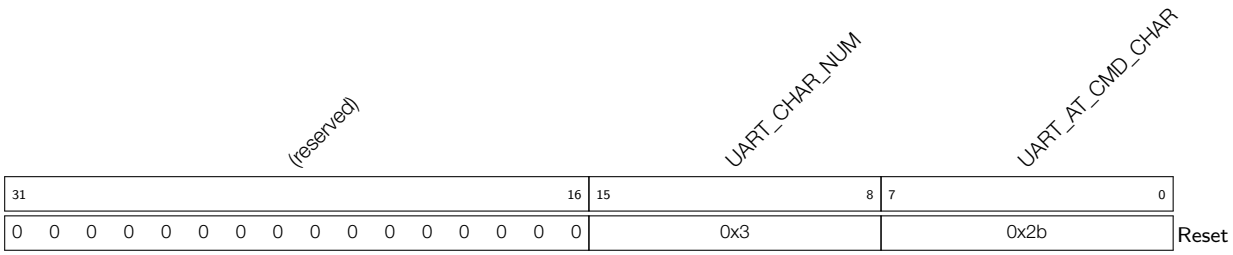
UART_POST_IDLE_NUM This field is used to configure the duration time between the last AT_CMD and the next data byte, in the unit of bit time (the time it takes to transfer one bit). (R/W)

Register 19.30. UART_AT_CMD_GAPTOUT_REG (0x0058)



UART_RX_GAP_TOUT This field is used to configure the duration time between the AT_CMD characters, in the unit of bit time (the time it takes to transfer one bit). (R/W)

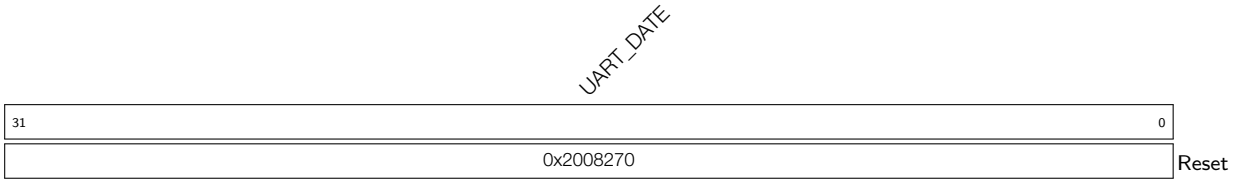
Register 19.31. UART_AT_CMD_CHAR_REG (0x005C)



UART_AT_CMD_CHAR This field is used to configure the content of AT_CMD character. (R/W)

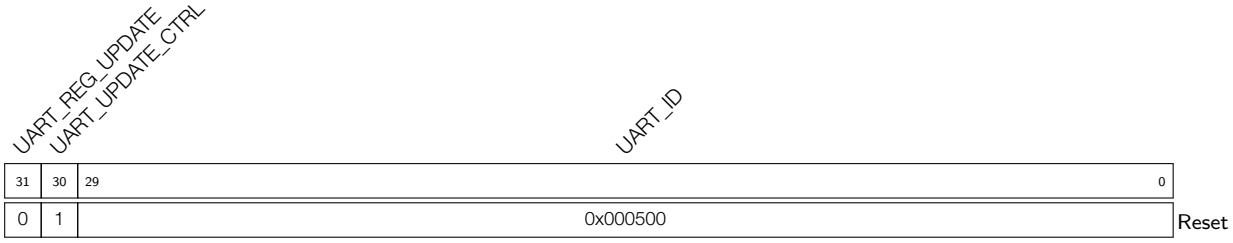
UART_CHAR_NUM This field is used to configure the number of continuous AT_CMD characterss received by the receiver. (R/W)

Register 19.32. UART_DATE_REG (0x007C)



UART_DATE This is the version control register. (R/W)

Register 19.33. UART_ID_REG (0x0080)



UART_ID This field is used to configure the UART_ID. (R/W)

UART_UPDATE_CTRL This bit is used to control register synchronization mode. This bit must be cleared before writing 1 to UART_REG_UPDATE to synchronize configured values to UART Core's clock domain. (R/W)

UART_REG_UPDATE When this bit is set to 1 by software, registers are synchronized to UART Core's clock domain. This bit is cleared by hardware after synchronization is done. (R/W/SC)

20 SPI Controller (SPI)

20.1 Overview

The Serial Peripheral Interface (SPI) is a synchronous serial interface useful for communication with external peripherals. The ESP8684 chip integrates three SPI controllers:

- SPI0,
- SPI1,
- and General Purpose SPI2 (GP-SPI2).

SPI0 and SPI1 controllers (MSPI) are primarily reserved for internal use to communicate with external flash and PSRAM memory. This chapter mainly focuses on the GP-SPI2 controller.

20.2 Glossary

To better illustrate the functions of GP-SPI2, the following terms are used in this chapter.

Master Mode	GP-SPI2 acts as an SPI master and initiates SPI transactions.
Slave Mode	GP-SPI2 acts as an SPI slave and exchanges data with its master when its CS is asserted.
MISO	Master in, slave out, data transmission from a slave to a master.
MOSI	Master out, slave in, data transmission from a master to a slave
Transaction	One instance of a master asserting a CS line, transferring data to and from a slave, and de-asserting the CS line. Transactions are atomic, which means they can never be interrupted by another transaction.
SPI Transfer	The whole process of an SPI master exchanging data with a slave. One SPI transfer consists of one or more SPI transactions.
Single Transfer	An SPI transfer that consists of only one transaction.
CPU-Controlled Transfer	A data transfer that happens between CPU buffer SPI_W0_REG ~ SPI_W15_REG and SPI peripheral.
DMA-Controlled Transfer	A data transfer that happens between DMA and SPI peripheral, controlled by the DMA engine.
Configurable Segmented Transfer	A data transfer controlled by DMA in SPI master mode. Such transfer consists of multiple transactions (segments), and each transaction can be configured independently.
Slave Segmented Transfer	A data transfer controlled by DMA in SPI slave mode. Such transfer consists of multiple transactions (segments).
Full-duplex	The sending line and receiving line between the master and the slave are independent. Sending data and receiving data happen at the same time.
Half-duplex	Only one side, the master or the slave, sends data, and the other side receives data. Sending data and receiving data can not happen simultaneously on one side.
4-line full-duplex	4-line here means: clock line, CS line, and two data lines. The two data lines can be used to send or receive data simultaneously.

4-line half-duplex	4-line here means: clock line, CS line, and two data lines. The two data lines can not be used simultaneously.
3-line half-duplex	3-line here means: clock line, CS line, and one data line. The data line is used to transmit or receive data.
1-bit SPI	In one clock cycle, one bit can be transferred.
(2-bit) Dual SPI	In one clock cycle, two bits can be transferred.
Dual Output Read	A data mode of Dual SPI. In one clock cycle, one bit of a command, or one bit of an address, or two bits of data can be transferred.
Dual I/O Read	Another data mode of Dual SPI. In one clock cycle, one bit of a command, or two bits of an address, or two bits of data can be transferred.
(4-bit) Quad SPI	In one clock cycle, four bits can be transferred.
Quad Output Read	A data mode of Quad SPI. In one clock cycle, one bit of a command, or one bit of an address, or four bits of data can be transferred.
Quad I/O Read	Another data mode of Quad SPI. In one clock cycle, one bit of a command, or four bits of an address, or four bits of data can be transferred.
QPI	In one clock cycle, four bits of a command, or four bits of an address, or four bits of data can be transferred.

20.3 Features

Some of the key features of GP-SPI2 are:

- Master and slave modes
- Half- and full-duplex communications
- CPU- and DMA-controlled transfers
- Various data modes:
  - 1-bit SPI mode
  - 2-bit Dual SPI mode
  - 4-bit Quad SPI mode
  - QPI mode
- Configurable module clock frequency:
  - Master: up to 40 MHz
  - Slave: up to 40 MHz
- Configurable data length:
  - CPU-controlled transfer in master mode or in slave mode: 1 ~ 64 B
  - DMA-controlled single transfer in master mode: 1 ~ 32 KB
  - DMA-controlled configurable segmented transfer in master mode: data length is unlimited
  - DMA-controlled single transfer or segmented transfer in slave mode: data length is unlimited

- Configurable bit read/write order
- Independent interrupts for CPU-controlled transfer and DMA-controlled transfer
- Configurable clock polarity and phase
- Four SPI clock modes: mode 0 ~ mode 3
- Six CS lines in master mode: CS0 ~ CS5
- Able to communicate with SPI devices, such as a sensor, a screen controller, as well as a flash or RAM chip

20.4 Architectural Overview

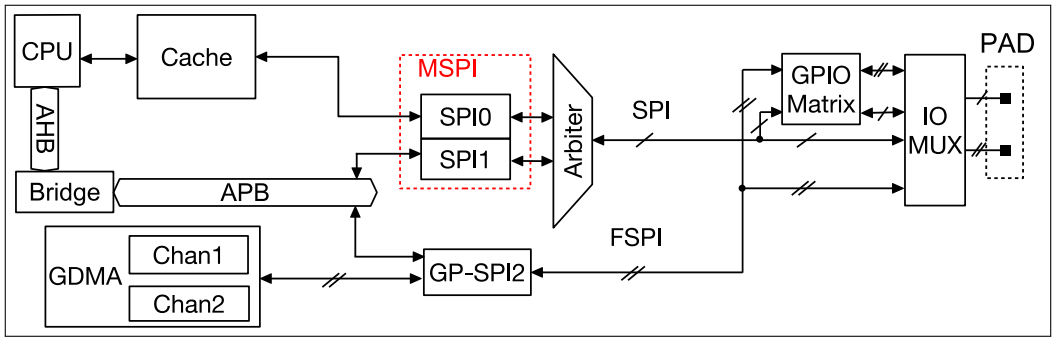


Figure 20-1. SPI Module Overview

Figure 20-1 shows an overview of SPI module. GP-SPI2 exchanges data with SPI devices by the following ways:

- CPU-controlled transfer: CPU <-> GP-SPI2 <-> SPI devices
- DMA-controlled transfer: GDMA <-> GP-SPI2 <-> SPI devices

The signals for GP-SPI2 are prefixed with "FSPI" (Fast SPI). FSPI bus signals are routed to GPIO pins via either GPIO matrix or IO MUX. For more information, see Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO MUX)*.

20.5 Functional Description

20.5.1 Data Modes

GP-SPI2 can be configured as either a master or a slave to communicate with other SPI devices in the following data modes, see Table 20-2.

Table 20-2. Data Modes Supported by GP-SPI2

Supported Mode		CMD State	Address State	Data State
1-bit SPI		1-bit	1-bit	1-bit
Dual SPI	Dual Output Read	1-bit	1-bit	2-bit
	Dual I/O Read	1-bit	2-bit	2-bit
Quad SPI	Quad Output Read	1-bit	1-bit	4-bit
	Quad I/O Read	1-bit	4-bit	4-bit
QPI		4-bit	4-bit	4-bit

For more information about the data modes used when GP-SPI2 works as a master or a slave, see Section 20.5.8 and Section 20.5.9, respectively.

20.5.2 Introduction to FSPI Bus Signals

Functional description of FSPI bus signals is shown in Table 20-3. Table 20-4 lists the signals used in various SPI modes.

Table 20-3. Functional Description of FSPI Bus Signals

FSPI Bus Signal	Function
FSPID	MOSI/SIO0 (serial data input and output, bit0)
FSPIQ	MISO/SIO1 (serial data input and output, bit1)
FSPIWP	SIO2 (serial data input and output, bit2)
FSPIHD	SIO3 (serial data input and output, bit3)
FSPICLK	Input and output clock in master/slave mode
FSPICS0	Input and output CS signal in master/slave mode
FSPICS1 ~ 5	Output CS signal in master mode

Table 20-4. Signals Used in Various SPI Modes

FSPI Signal	Master Mode						Slave Mode					
	1-bit SPI			2-bit Dual SPI	4-bit Quad SPI	QPI	1-bit SPI			2-bit Dual SPI	4-bit Quad SPI	QPI
	FD[1]	3-line HD[2]	4-line HD				FD	3-line HD	4-line HD			
FSPICLK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FSPICS0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FSPICS1	Y	Y	Y	Y	Y	Y						
FSPICS2	Y	Y	Y	Y	Y	Y						
FSPICS3	Y	Y	Y	Y	Y	Y						
FSPICS4	Y	Y	Y	Y	Y	Y						
FSPICS5	Y	Y	Y	Y	Y	Y						
FSPID	Y	Y	(Y)[3]	Y[4]	Y[5]	Y	Y	Y	(Y)[6]	Y[7]	Y[8]	Y
FSPIQ	Y		(Y)[3]	Y[4]	Y[5]	Y	Y		(Y)[6]	Y[7]	Y[8]	Y
FSPIWP					Y[5]	Y					Y[8]	Y
FSPIHD					Y[5]	Y					Y[8]	Y

[1] FD: full-duplex

[2] HD: half-duplex

[3] Only one of the two signals is used at a time.

[4] The two signals are used in parallel.

[5] The four signals are used in parallel.

[6] Only one of the two signals is used at a time.

[7] The two signals are used in parallel.

[8] The four signals are used in parallel.

20.5.3 Bit Read/Write Order Control

In master mode:

- The bit order of the command, address and data sent by the GP-SPI2 master is controlled by SPI_WR_BIT_ORDER.
- The bit order of the data received by the master is controlled by SPI_RD_BIT_ORDER.

In slave mode:

- The bit order of the data sent by the GP-SPI2 slave is controlled by SPI_WR_BIT_ORDER.
- The bit order of the command, address and data received by the slave is controlled by SPI_RD_BIT_ORDER.

Table 20-5 shows the function of SPI_RD/WR_BIT_ORDER.

GoBack

Table 20-5. Bit Order Control in GP-SPI2 Master and Slave Modes

Bit Mode	FSPI Bus Data	SPI_RD/WR_BIT_ORDER = 0 (MSB)	SPI_RD/WR_BIT_ORDER = 2 (MSB)	SPI_RD/WR_BIT_ORDER = 1 (LSB)	SPI_RD/WR_BIT_ORDER = 3 (LSB)
1-bit mode	FSPID or FSPIQ	B7->B6->B5->B4->B3->B2->B1->B0	B7->B6->B5->B4->B3->B2->B1->B0	B0->B1->B2->B3->B4->B5->B6->B7	B0->B1->B2->B3->B4->B5->B6->B7
2-bit mode	FSPIQ	B7->B5->B3->B1	B6->B4->B2->B0	B1->B3->B5->B7	B0->B2->B4->B6
	FSPID	B6->B4->B2->B0	B7->B5->B3->B1	B0->B2->B4->B6	B1->B3->B5->B7
4-bit mode	FSPIHD	B7->B3	B4->B0	B3->B7	B0->B4
	FSPIWP	B6->B2	B5->B1	B2->B6	B1->B5
	FSPIQ	B5->B1	B6->B2	B1->B5	B2->B6
	FSPID	B4->B0	B7->B3	B0->B4	B3->B7

20.5.4 Transfer Modes

GP-SPI2 supports the following transfers when working as a master or a slave.

Table 20-6. Supported Transfers in Master and Slave Modes

Mode		CPU-Controlled Single Transfer	DMA-Controlled Single Transfer	DMA-Controlled Configurable Segmented Transfer	DMA-Controlled Slave Segmented Transfer
Master	Full-Duplex	Y	Y	Y	–
	Half-Duplex	Y	Y	Y	–
Slave	Full-Duplex	Y	Y	–	Y
	Half-Duplex	Y	Y	–	Y

The following sections provide detailed information about the transfer modes listed in the table above.

20.5.5 CPU-Controlled Data Transfer

GP-SPI2 provides 16 x 32-bit data buffers, i.e., SPI_W0_REG ~ SPI_W15_REG, see Figure 20-2. CPU-controlled transfer indicates the transfer, in which the data to send is from GP-SPI2 data buffer and the received data is stored to GP-SPI2 data buffer. In such transfer, every single transaction needs to be triggered by the CPU, after its related registers are configured. For such reason, the CPU-controlled transfer is always single transfers (consisting of only one transaction). CPU-controlled mode supports full-duplex communication and half-duplex communication.

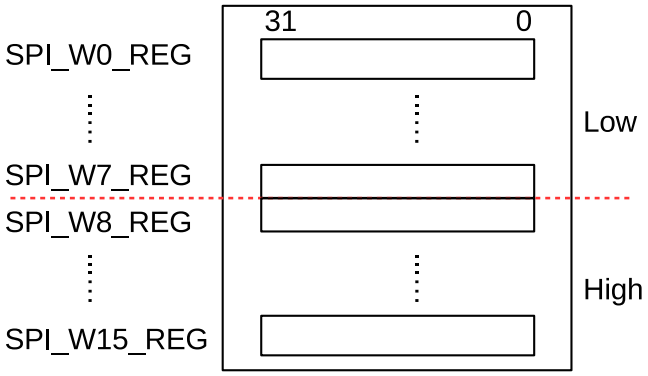


Figure 20-2. Data Buffer Used in CPU-Controlled Transfer

20.5.5.1 CPU-Controlled Master Mode

In a CPU-controlled master full-duplex or half-duplex transfer, the RX or TX data is saved to or sent from SPI_W0_REG ~ SPI_W15_REG. The bits SPI_USR_MOSI_HIGHPART and SPI_USR_MISO_HIGHPART control which buffers are used, see the list below.

- **TX data**
  - When SPI_USR_MOSI_HIGHPART is cleared, i.e. high part mode is disabled, TX data is read from SPI_W0_REG ~ SPI_W15_REG and the data address is incremented by 1 on each byte transferred. **If the data byte length is larger than 64, the data in SPI_W0_REG ~ SPI_W15_REG may be sent more than once.** Take each 256 bytes as a cycle:

* The first 64 bytes (Byte 0 ~ Byte 63) are read from SPI_W0_REG ~ SPI_W15_REG, respectively.
* Byte 64 ~ Byte 255 are read from SPI_W15_REG[31:24] repeatedly.
* Byte 256 ~ Byte 319 (the first 64 bytes in the another 256 bytes) are read from SPI_W0_REG ~ SPI_W15_REG again, respectively, same as the behaviors described above.

For instance: to send 258 bytes (Byte 0 ~ Byte 257), the data is read from the registers as follows:

* The first 64 bytes (Byte 0 ~ Byte 63) are read from SPI_W0_REG ~ SPI_W15_REG, respectively.
* Byte 64 ~ Byte 255 are read from SPI_W15_REG[31:24] repeatedly.
* The other bytes (Byte 256 and Byte 257) are read from SPI_W0_REG[7:0] and SPI_W0_REG[15:8] again, respectively. The logic is:
  - The address to read data for Byte 256 is the result of (256 % 64 = 0), i.e.,SPI_W0_REG[7:0].
  - The address to read data for Byte 257 is the result of (257 % 64 = 1), i.e., SPI_W0_REG[15:8].

– When SPI_USR_MOSI_HIGHPART is set, i.e. high part mode is enabled, TX data is read from SPI_W8_REG ~ SPI_W15_REG and the data address is incremented by 1 on each byte transferred. **If the data byte length is larger than 32, the data in SPI_W8_REG ~ SPI_W15_REG may be sent more than once.** Take each 256 bytes as a cycle:

* The first 32 bytes (Byte 0 ~ Byte 31) are read from SPI_W8_REG ~ SPI_W15_REG, respectively.
* Byte 32 ~ Byte 255 are read from SPI_W15_REG[31:24] repeatedly.
* Byte 256 ~ Byte 287 (the first 32 bytes in the another 256 bytes) are read from SPI_W8_REG ~ SPI_W15_REG again, respectively, same as the behaviors described above.

For instance: to send 258 bytes (Byte 0 ~ Byte 257), the data is read from the registers as follows:

* The first 32 bytes (Byte 0 ~ Byte 31) are read from SPI_W8_REG ~ SPI_W15_REG, respectively.
* Byte 32 ~ Byte 255 are read from SPI_W15_REG[31:24] repeatedly.
* The other bytes (Byte 256 and Byte 257) are read from SPI_W8_REG[7:0] and SPI_W8_REG[15:8] again, respectively. The logic is:
  - The address to read data for Byte 256 is the result of (256 % 32 = 0), i.e., SPI_W8_REG[7:0].
  - The address to read data for Byte 257 is the result of (257 % 32 = 1), i.e., SPI_W8_REG[15:8].

- **RX data**

– When SPI_USR_MISO_HIGHPART is cleared, i.e. high part mode is disabled, RX data is saved to SPI_W0_REG ~ SPI_W15_REG, and the data address is incremented by 1 on each byte transferred. **If the data byte length is larger than 64, the data in SPI_W0_REG ~ SPI_W15_REG may be overwritten.** Take each 256 bytes as a cycle:

* The first 64 bytes (Byte 0 ~ Byte 63) are saved to SPI_W0_REG ~ SPI_W15_REG, respectively.
* Byte 64 ~ Byte 255 are saved to SPI_W15_REG[31:24] repeatedly.
* Byte 255 ~ Byte 319 (the first 64 bytes in the another 256 bytes) are saved to SPI_W0_REG ~ SPI_W15_REG again, respectively, same as the behaviors described above.

For instance: to receive 258 bytes (Byte 0 ~ Byte 257), the data is saved to the registers as follows:

* The first 64 bytes (Byte 0 ~ Byte 63) are saved to SPI_W0_REG ~ SPI_W15_REG, respectively.
* Byte 64 ~ Byte 255 are saved to SPI_W15_REG[31:24] repeatedly.
* The other bytes (Byte 256 and Byte 257) are saved to SPI_W0_REG[7:0] and SPI_W0_REG[15:8] again, respectively. The logic is:
  - The address to save Byte 256 is the result of (256 % 64 = 0), i.e., SPI_W0_REG[7:0].
  - The address to save Byte 257 is the result of (257 % 64 = 1), i.e., SPI_W0_REG[15:8].

– When SPI_USR_MISO_HIGHPART is set, i.e. high part mode is enabled, the RX data is saved to SPI_W8_REG ~ SPI_W15_REG, and the data address is incremented by 1 on each byte transferred. **If the data byte length is larger than 32, the content of SPI_W8_REG ~ SPI_W15_REG may be overwritten.** Take each 256 bytes as a cycle:

* Byte 0 ~ Byte 31 are saved to SPI_W8_REG ~ SPI_W15_REG, respectively.
* Byte 32 ~ Byte 255 are saved to SPI_W15_REG[31:24] repeatedly.
* Byte 256 ~ Byte 287 (the first 32 bytes in the another 256 bytes) are saved to SPI_W8_REG ~ SPI_W15_REG again, respectively.

For instance: to receive 258 bytes (Byte 0 ~ Byte 257), the data is saved to the registers as follows:

* The first 32 bytes (Byte 0 ~ Byte 31) are saved to SPI_W8_REG ~ SPI_W15_REG, respectively.
* Byte 32 ~ Byte 255 are saved to SPI_W15_REG[31:24] repeatedly.
* The other bytes (Byte 256 and Byte 257) are saved to SPI_W8_REG[7:0] and SPI_W8_REG[15:8] again, respectively. The logic is:
  - The address to save Byte 256 is the result of (256 % 32 = 0), i.e., SPI_W8_REG[7:0].
  - The address to save Byte 257 is the result of (257 % 32 = 1), i.e., SPI_W8_REG[15:8].

Note:

- TX/RX data address mentioned above both are byte-addressable.
  – If high part mode is disabled, Address 0 stands for SPI_W0_REG[7:0], and Address 1 for SPI_W0_REG[15:8], and so on.
  – If high part mode is enabled, Address 0 stands for SPI_W8_REG[7:0], and Address 1 for SPI_W8_REG[15:8], and so on.

  The largest address points to SPI_W15_REG[31:24].
- To avoid any possible error in TX/RX data, such as TX data being sent more than once or RX data being overwritten, please make sure the registers are configured correctly.

20.5.5.2 CPU-Controlled Slave Mode

In a CPU-controlled slave full-duplex or half-duplex transfer, the RX data or TX data is saved to or sent from SPI_W0_REG ~ SPI_W15_REG, which are byte-addressable.

- In full-duplex communication, the address of SPI_W0_REG ~ SPI_W15_REG starts from 0 and is incremented by 1 on each byte transferred. If the data address is larger than 63, the data in SPI_W0_REG

~ SPI_W15_REG will be overwritten, same as the behaviors described in the master mode when high part mode is disabled.

- In half-duplex communication, the ADDR value in transmission format is the start address of the RX or TX data, corresponding to the registers SPI_W0_REG ~ SPI_W15_REG. The RX or TX address is incremented by 1 on each byte transferred. If the address is larger than 63 (the highest byte address, i.e. SPI_W15_REG[31:24]), the data in SPI_W8_REG ~ SPI_W15_REG will be overwritten, same as the behaviors described in the master mode when high part mode is enabled.

According to your applications, the registers SPI_W0_REG ~ SPI_W15_REG can be used as:

- data buffers only
- data buffers and status buffers
- status buffers only

20.5.6 DMA-Controlled Data Transfer

DMA-controlled transfer refers to the transfer, in which the GDMA RX module receives data and the GDMA TX module sends data. This transfer is supported both in master mode and in slave mode.

A DMA-controlled transfer can be

- a single transfer, consisting of only one transaction. GP-SPI2 supports this transfer both in master and slave modes.
- a configurable segmented transfer, consisting of several transactions (segments). GP-SPI2 supports this transfer only in master mode. For more information, see Section 20.5.8.5.
- a slave segmented transfer, consisting of several transactions (segments). GP-SPI2 supports this transfer only in slave mode. For more information, see Section 20.5.9.3.

A DMA-controlled transfer only needs to be triggered once by CPU. When such a transfer is triggered, data is transferred by the GDMA engine from or to the DMA-linked memory, without CPU operation.

DMA-controlled mode supports full-duplex communication, half-duplex communication and functions described in Section 20.5.8 and Section 20.5.9. Meanwhile, the GDMA RX module is independent from the GDMA TX module, which means that there are four kinds of full-duplex communications:

- Data is received in DMA-controlled mode and sent in DMA-controlled mode.
- Data is received in DMA-controlled mode but sent in CPU-controlled mode.
- Data is received in CPU-controlled mode but sent in DMA-controlled mode.
- Data is received in CPU-controlled mode and sent in CPU-controlled mode.

20.5.6.1 GDMA Configuration

- Select a GDMA channel*n*, and configure a GDMA TX/RX descriptor, see Chapter 2 *GDMA Controller (GDMA)*.
- Set the bit GDMA_INLINK_START_CH*n* or GDMA_OUTLINK_START_CH*n* to start GDMA RX engine and TX engine, respectively.

- Before all the GDMA TX buffer is used or the GDMA TX engine is reset, if GDMA_OUTLINK_RESTART_CH*n* is set, a new TX buffer will be added to the end of the last TX buffer in use.
- GDMA RX buffer is linked in the same way as the GDMA TX buffer, by setting GDMA_INLINK_START_CH*n* or GDMA_INLINK_RESTART_CH*n*.
- The TX and RX data lengths are determined by the configured GDMA TX and RX buffer respectively, both of which are 0 ~ 32 KB.
- Initialize GDMA inlink and outlink before GDMA starts. The bits SPI_DMA_RX_ENA and SPI_DMA_TX_ENA in register SPI_DMA_CONF_REG should be set, otherwise the read/write data will be stored to/sent from the registers SPI_W0_REG ~ SPI_W15_REG.

In master mode, if GDMA_IN_SUC_EOF_CH*n*_INT_ENA is set, then the interrupt GDMA_IN_SUC_EOF_CH*n*_INT will be triggered when one single transfer or one configurable segmented transfer is finished.

In slave mode, if GDMA_IN_SUC_EOF_CH*n*_INT_ENA is set, then the interrupt GDMA_IN_SUC_EOF_CH*n*_INT will be triggered when one of the following conditions are met.

Table 20-7. Interrupt Trigger Condition on GP-SPI2 Data Transfer in Slave Mode

Transfer Type	Control Bit[1]	Control Bit[2]	Condition
Slave Single Transfer	0	0	A single transfer is done.
	1	0	A single transfer is done. Or the length of the received data is equal to (SPI_MS_DATA_BITLEN + 1)
Slave Segmented Transfer	0	1	(CMD7 or End_SEG_TRANS) is received correctly.
	1	1	(CMD7 or End_SEG_TRANS) is received correctly. Or the length of the received data is equal to (SPI_MS_DATA_BITLEN + 1)

[1] SPI_RX_EOF_EN

[2] SPI_DMA_SLV_SEG_TRANS_EN

20.5.6.2 GDMA TX/RX Buffer Length Control

It is recommended that the length of configured GDMA TX/RX buffer is equal to the length of real transferred data.

- If the length of configured GDMA TX buffer is shorter than that of real transferred data, the extra data will be the same as the last transferred data. SPI_OUTFIFO_EMPTY_ERR_INT and GDMA_OUT_EOF_CH*n*_INT are triggered.
- If the length of configured GDMA TX buffer is longer than that of real transferred data, the TX buffer is not fully used, and the remaining buffer will be used for following transaction even if a new TX buffer is linked later. Please keep it in mind. Or save the unused data and reset DMA.
- If the length of configured GDMA RX buffer is shorter than that of real transferred data, the extra data will be lost. The interrupts SPI_INFIFO_FULL_ERR_INT and SPI_TRANS_DONE_INT are triggered. But GDMA_IN_SUC_EOF_CH*n*_INT interrupt is not generated.
- If the length of configured GDMA RX buffer is longer than that of real transferred data, the RX buffer is not fully used, and the remaining buffer is discarded. In the following transaction, a new linked buffer will be used directly.

20.5.7 Data Flow Control in GP-SPI2 Master and Slave Modes

CPU-controlled and DMA-controlled transfers are supported in GP-SPI2 master and slave modes. CPU-controlled transfer means that data is transferred between registers SPI_W0_REG ~ SPI_W15_REG and the SPI device. DMA-controlled transfer means that data is transferred between the configured GDMA TX/RX buffer and the SPI device. To select between the two transfer modes, configure SPI_DMA_RX_ENA and SPI_DMA_TX_ENA before the transfer starts.

20.5.7.1 GP-SPI2 Functional Blocks

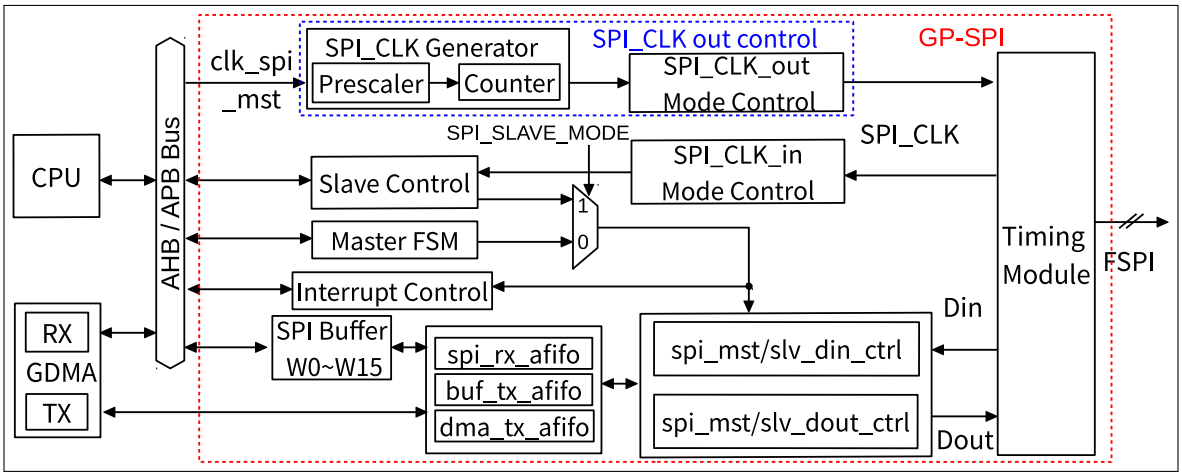


Figure 20-3. GP-SPI2 Block Diagram

Figure 20-3 shows the main functional blocks in GP-SPI2, including:

- Master FSM: all the features, supported in GP-SPI2 master mode, are controlled by this state machine together with register configuration.
- SPI Buffer: SPI_W0_REG ~ SPI_W15_REG, see Figure 20-2. The data transferred in CPU-controlled mode is prepared in this buffer.
- Timing Module: capture data on FSPI bus.
- spi_mst/slv_din_ctrl and spi_mst/slv_dout_ctrl: convert the TX/RX data into bytes.
- spi_rx_afifo: store the received data.
- buf_tx_afifo: store the data to send.
- dma_tx_afifo: store the data from GDMA.
- clk_spi_mst: this clock is the module clock of GP-SPI2 and derived from PLL_CLK. It is used in GP-SPI2 master mode, to generate SPI_CLK signal for data transfer and for slaves.
- SPI_CLK Generator: generate SPI_CLK by dividing clk_spi_mst. The divider is determined by SPI_CLKCNT_N and SPI_CLKDIV_PRE, see Section 20.7.
- SPI_CLK_out Mode Control: output the SPI_CLK signal for data transfer and for slaves.
- SPI_CLK_in Mode Control: capture the SPI_CLK signal from SPI master when GP-SPI2 works as a slave.

20.5.7.2 Data Flow Control in Master Mode

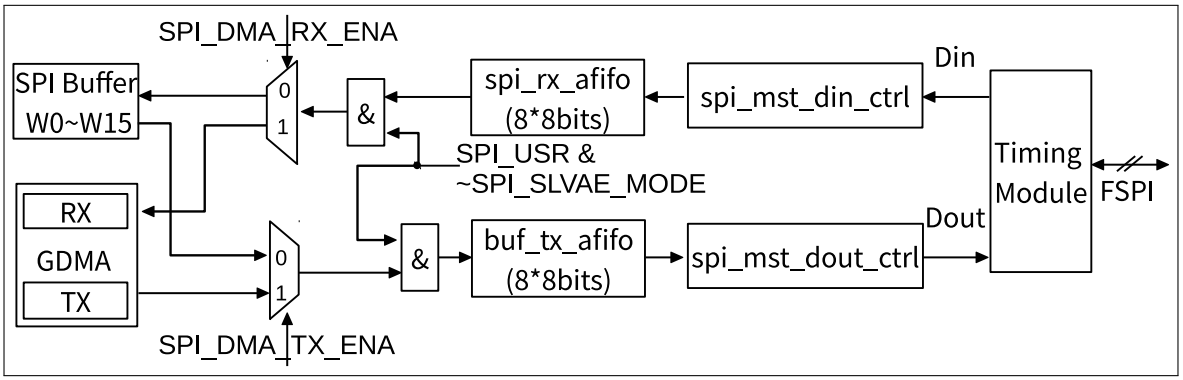


Figure 20-4. Data Flow Control in GP-SPI2 Master Mode

Figure 20-4 shows the data flow of GP-SPI2 in master mode. Its control logic is as follows:

- RX data: data in FSPI bus is captured by Timing Module, converted in units of bytes by spi_mst_din_ctrl module, then buffered in spi_rx_afifo, and finally stored in corresponding addresses according to the transfer modes.
  - CPU-controlled transfer: the data is stored to registers SPI_W0_REG ~ SPI_W15_REG.
  - DMA-controlled transfer: the data is stored to GDMA RX buffer.
- TX data: the TX data is from corresponding addresses according to transfer modes and is saved to buf_tx_afifo.
  - CPU-controlled transfer: TX data is from SPI_W0_REG ~ SPI_W15_REG.
  - DMA-controlled transfer: TX data is from GDMA TX buffer.

The data in buf_tx_afifo is sent out to Timing Module in 1/2/4-bit modes, controlled by GP-SPI2 state machine. The Timing Module can be used for timing compensation. For more information, see Section 20.8.

20.5.7.3 Data Flow Control in Slave Mode

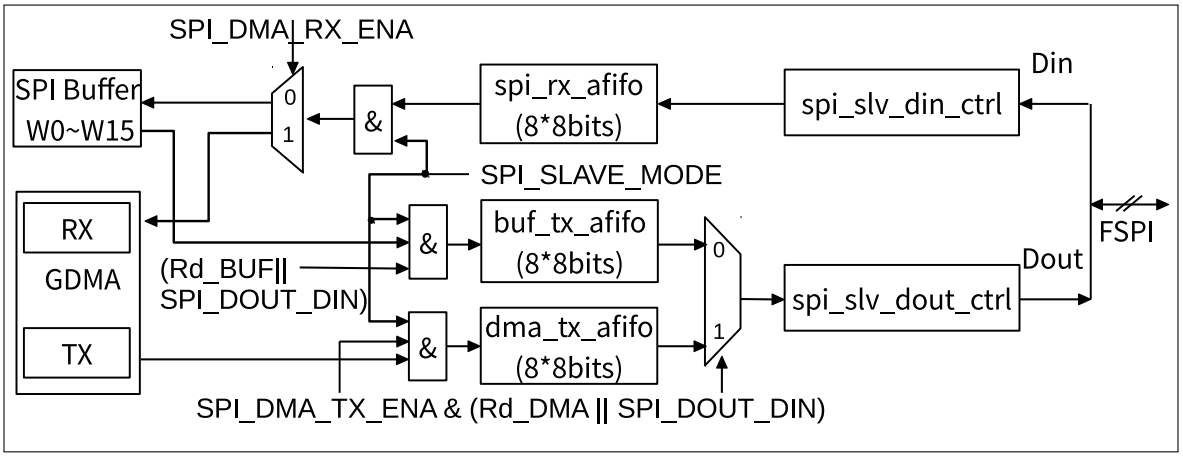


Figure 20-5. Data Flow Control in GP-SPI2 Slave Mode

Figure 20-5 shows the data flow in GP-SPI2 slave mode. Its control logic is as follows:

- In CPU/DMA-controlled full-duplex/half-duplex modes, when an external SPI master starts the SPI transfer, data on the FSPI bus is captured, converted into unit of bytes by the spi_slv_din_ctrl module, and then is stored in spi_rx_afifo.
  - In CPU-controlled full-duplex transfer, the received data in spi_rx_afifo will be later stored into registers SPI_W0_REG ~ SPI_W15_REG, successively.
  - In half-duplex Wr_BUF transfer, when the value of address (SLV_ADDR[7:0]) is received, the received data in spi_rx_afifo will be stored in the related address of registers SPI_W0_REG ~ SPI_W15_REG
  - In DMA-controlled full-duplex transfer or in half-duplex Wr_DMA transfer, the received data in spi_rx_afifo will be stored in the configured GDMA RX buffer.
- In CPU-controlled full-/half-duplex transfer, the data to send is stored in buf_tx_afifo. In DMA-controlled full-/half-duplex transfer, the data to send is stored in dma_tx_afifo. Therefore, Rd_BUF transaction controlled by CPU and Rd_DMA transaction controlled by DMA can be done in one slave segmented transfer. TX data comes from corresponding addresses according the transfer modes.
  - In CPU-controlled full-duplex transfer, when SPI_SLAVE_MODE and SPI_DOUTDIN are set and SPI_DMA_TX_ENA is cleared, the data in SPI_W0_REG ~ SPI_W15_REG will be stored into buf_tx_afifo;
  - In CPU-controlled half-duplex transfer, when SPI_SLAVE_MODE is set, SPI_DOUTDIN is cleared, Rd_BUF command and SLV_ADDR[7:0] are received, the data started from the related address of SPI_W0_REG ~ SPI_W15_REG will be stored into buf_tx_afifo;
  - In DMA-controlled full-duplex transfer, when SPI_SLAVE_MODE, SPI_DOUTDIN and SPI_DMA_TX_ENA are set, the data in the configured GDMA TX buffer will be stored into dma_tx_afifo;
  - In DMA-controlled half-duplex transfer, when SPI_SLAVE_MODE is set, SPI_DOUTDIN is cleared, and Rd_DMA command is received, the data in the configured GDMA TX buffer will be stored into dma_tx_afifo.

  The data in buf_tx_afifo or dma_tx_afifo is sent out by spi_slv_dout_ctrl module in 1/2/4-bit modes.

20.5.8 GP-SPI2 Works as a Master

GP-SPI2 can be configured as a SPI master by clearing the bit SPI_SLAVE_MODE in SPI_SLAVE_REG. In this operation mode, GP-SPI2 provides clock signal (the divided clock from GP-SPI2 module clock) and six CS lines (CS0 ~ CS5).

Note:

- The length of transferred data must be an integral multiple of byte (8 bits), otherwise the extra bits will be lost. The extra bits here means the result of total data bits mod 8.
- To transfer bits that is not an integral multiple of byte (8 bits), consider implementing it in CMD state or ADDR state.

20.5.8.1 State Machine

When GP-SPI2 works as a master, the state machine controls its various states during data transfer, including configuration (CONF), preparation (PREP), command (CMD), address (ADDR), dummy (DUMMY), data out (DOUT), and data in (DIN) states. GP-SPI2 is mainly used to access 1/2/4-bit SPI devices, such as flash and external RAM, thus the naming of GP-SPI2 states keeps consistent with the sequence naming of flash and external RAM. The meaning of each state is described as follows and Figure 20-6 shows the workflow of GP-SPI2 state machine.

1. IDLE: GP-SPI2 is not active or is in slave mode.
2. CONF: only used in DMA-controlled configurable segmented transfer. Set SPI_USR and SPI_USR_CONF to enable this state. If this state is not enabled, it means the current transfer is a single transfer.
3. PREP: prepare an SPI transaction and control SPI CS setup time. Set SPI_USR and SPI_CS_SETUP to enable this state.
4. CMD: send command sequence. Set SPI_USR and SPI_USR_COMMAND to enable this state.
5. ADDR: send address sequence. Set SPI_USR and SPI_USR_ADDR to enable this state.
6. DUMMY (wait cycle): send dummy sequence. Set SPI_USR and SPI_USR_DUMMY to enable this state.
7. DATA: transfer data.
    - DOUT: send data sequence. Set SPI_USR and SPI_USR_MOSI to enable this state.
    - DIN: receive data sequence. Set SPI_USR and SPI_USR_MISO to enable this state.
8. DONE: control SPI CS hold time. Set SPI_USR to enable this state.

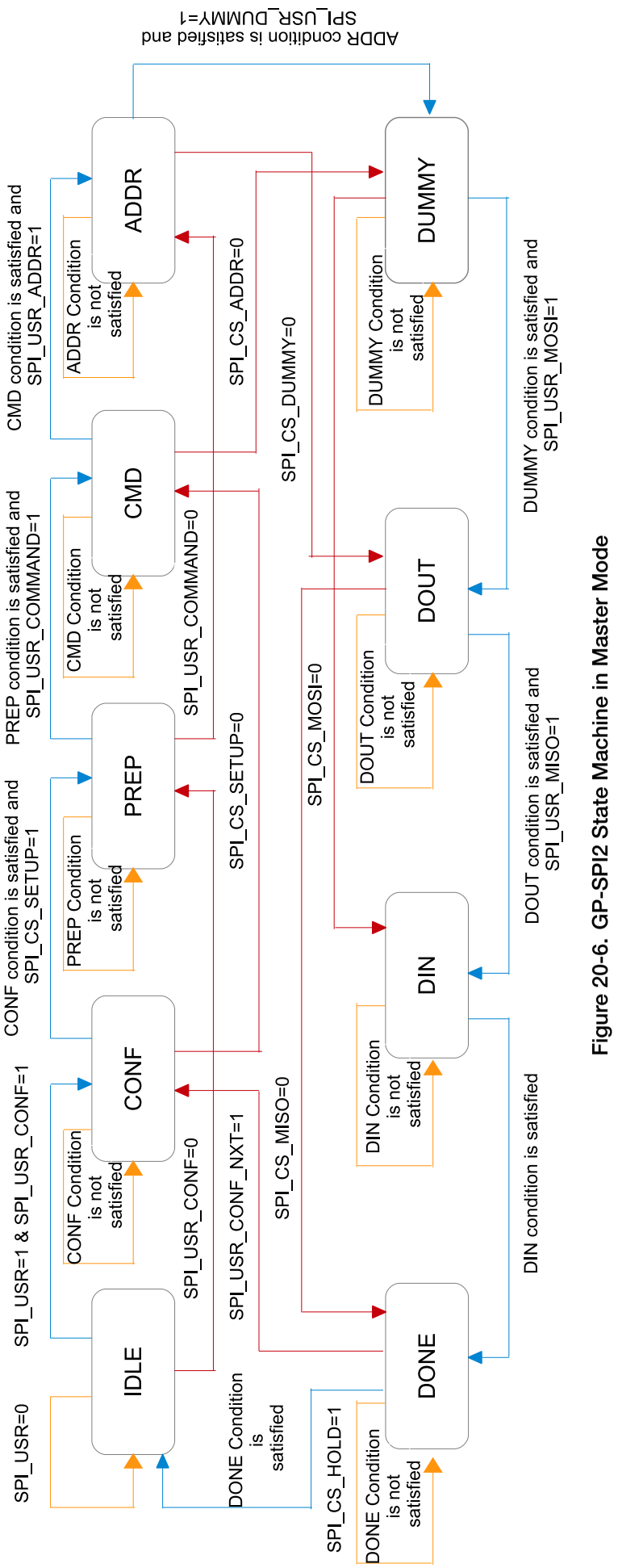


Figure 20-6. GP-SPI2 State Machine in Master Mode

Legend to state flow:

- —: indicates corresponding state condition is not satisfied; repeats current state.
- —: corresponding registers are set and conditions are satisfied; goes to next state.
- —: state registers are not set; skips one or more following states, depending on the registers of the following states are set or not.

Explanation to the conditions listed in the figure above:

- CONF condition: gpc[17:0] >= SPI_CONF_BITLEN[17:0]
- PREP condition: gpc[4:0] >= SPI_CS_SETUP_TIME[4:0]
- CMD condition: gpc[3:0] >= SPI_USR_COMMAND_BITLEN[3:0]
- ADDR condition: gpc[4:0] >= SPI_USR_ADDR_BITLEN[4:0]
- DUMMY condition: gpc[7:0] >= SPI_USR_DUMMY_CYCLELEN[7:0]
- DOUT condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DIN condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DONE condition: (gpc[4:0] >= SPI_CS_HOLD_TIME[4:0] || SPI_CS_HOLD == 1'b0)

A counter (gpc[17:0]) is used in the state machine to control the cycle length of each state. The states CONF, PREP, CMD, ADDR, DUMMY, DOUT, and DIN can be enabled or disabled independently. The cycle length of each state can also be configured independently.

20.5.8.2 Register Configuration for State and Bit Mode Control

Introduction

The registers, related to GP-SPI2 state control, are listed in Table 20-8. Users can enable QPI mode for GP-SPI2 by setting the bit SPI_QPI_MODE in register SPI_USER_REG.

Table 20-8. Registers Used for State Control in 1/2/4-bit Modes

State	Control Registers for 1-bit Mode FSPI Bus	Control Registers for 2-bit Mode FSPI Bus	Control Registers for 4-bit Mode FSPI Bus
CMD	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_DUAL SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_QUAD SPI_USR_COMMAND
ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_DUAL	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_QUAD
DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY
DIN	SPI_USR_MISO SPI_MS_DATA_BITLEN	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_DUAL	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_QUAD

Table 20-8. Registers Used for State Control in 1/2/4-bit Modes

State	Control Registers for 1-bit Mode FSPI Bus	Control Registers for 2-bit Mode FSPI Bus	Control Registers for 4-bit Mode FSPI Bus
DOUT	SPI_USR_MOSI SPI_MS_DATA_BITLEN	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_DUAL	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_QUAD

As shown in Table 20-8, the registers in each cell should be configured to set the FSPI bus to corresponding bit mode, i.e. the mode shown in the table header, at a specific state (corresponding to the first column).

Configuration

For instance, when GP-SPI2 reads data, and

- CMD is in 1-bit mode
- ADDR is in 2-bit mode
- DUMMY is 8 clock cycles
- DIN is in 4-bit mode

The register configuration can be as follows:

1. Configure CMD state related registers.
   - Configure the required command value in SPI_USR_COMMAND_VALUE.
   - Configure command bit length in SPI_USR_COMMAND_BITLEN. SPI_USR_COMMAND_BITLEN = expected bit length - 1.
   - Set SPI_USR_COMMAND.
   - Clear SPI_FCMD_DUAL and SPI_FCMD_QUAD.
2. Configure ADDR state related registers.
   - Configure the required address value in SPI_USR_ADDR_VALUE.
   - Configure address bit length in SPI_USR_ADDR_BITLEN. SPI_USR_ADDR_BITLEN = expected bit length - 1.
   - Set SPI_USR_ADDR and SPI_FADDR_DUAL.
   - Clear SPI_FADDR_QUAD.
3. Configure DUMMY state related registers.
   - Configure DUMMY cycles in SPI_USR_DUMMY_CYCLELEN. SPI_USR_DUMMY_CYCLELEN = expected clock cycles - 1.
   - Set SPI_USR_DUMMY.
4. Configure DIN state related registers.
   - Configure read data bit length in SPI_MS_DATA_BITLEN. SPI_MS_DATA_BITLEN = bit length expected - 1.
   - Set SPI_FREAD_QUAD and SPI_USR_MISO.

- Clear SPI_FREAD_DUAL.
- Configure GDMA in DMA-controlled mode. In CPU controlled mode, no action is needed.

5. Clear SPI_USR_MOSI.
6. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
7. Set SPI_USR to start GP-SPI2 transfer.

When writing data (DOUT state), SPI_USR_MOSI should be configured instead, while SPI_USR_MISO should be cleared. The output data bit length is the value of SPI_MS_DATA_BITLEN + 1. Output data should be configured in GP-SPI2 data buffer (SPI_W0_REG ~ SPI_W15_REG) in CPU-controlled mode, or GDMA TX buffer in DMA-controlled mode. The data byte order is incremented from LSB (byte 0) to MSB.

Pay special attention to the command value in SPI_USR_COMMAND_VALUE and to address value in SPI_USR_ADDR_VALUE.

The configuration of command value is as follows:

Table 20-9. Sending Sequence of Command Value

COMMAND_BITLEN[1]	COMMAND_VALUE[2]	BIT_ORDER[3]	Sending Sequence of Command Value
0 - 7	[7:0]	1	COMMAND_VALUE[COMMAND_BITLEN:0] is sent first.
		0	COMMAND_VALUE[7:7 - COMMAND_BITLEN] is sent first.
8 - 15	[15:0]	1	COMMAND_VALUE[7:0] is sent first, and then COMMAND_VALUE[COMMAND_BITLEN:8] is sent.
		0	COMMAND_VALUE[7:0] is sent first, and then COMMAND_VALUE[15:15 - COMMAND_BITLEN] is sent.

[1] SPI_USR_COMMAND_BITLEN: this field is used to configure the bit length of the command.
[2] SPI_USR_COMMAND_VALUE: command value is written into this field. For which part of this field is used, see the table above.
[3] SPI_WR_BIT_ORDER: 0: LSB first; 1: MSB first.

The configuration of address value is as follows:

Table 20-10. Sending Sequence of Address Value

ADDR_BITLEN[1]	ADDR_VALUE[2]	BIT_ORDER[3]	Sending Sequence of Address Value
0 - 7	[31:24]	1	ADDR_VALUE[ADDR_BITLEN + 24:24] is sent first.
		0	ADDR_VALUE[31:31 - ADDR_BITLEN] is sent first.
8 - 15	[31:16]	1	ADDR_VALUE[31:24] is sent first, and then ADDR_VALUE[ADDR_BITLEN + 8:16] is sent.
		0	ADDR_VALUE[31:24] is sent first, and then ADDR_VALUE[23:31 - ADDR_BITLEN] is sent.
16 - 23	[31:8]	1	ADDR_VALUE[31:16] is sent first, and then ADDR_VALUE[ADDR_BITLEN - 8:8] is sent.

		0	ADDR_VALUE[31:16] is sent first, and then ADDR_VALUE[15:31 - ADDR_BITLEN] is sent.
24 - 31	[31:0]	1	ADDR_VALUE[31:8] is sent first, and then ADDR_VALUE[ADDR_BITLEN - 24:0] is sent.
		0	ADDR_VALUE[31:8] is sent first, and then ADDR_VALUE[7:31 - ADDR_BITLEN] is sent.

[1] SPI_USR_ADDR_BITLEN: this field is used to configure the bit length of the address.

[2] SPI_USR_ADDR_VALUE: address value is written into this field. For which part of this field is used, see the table above.

[3] SPI_WR_BIT_ORDER: 0: LSB first; 1: MSB first.

20.5.8.3 Full-Duplex Communication (1-bit Mode Only)

Introduction

GP-SPI2 supports SPI full-duplex communication. In this mode, SPI master provides CLK and CS signals, exchanging data with SPI slave in 1-bit mode via MOSI (FSPID, sending) and MISO (FSPIQ, receiving) at the same time. To enable this communication mode, set the bit SPI_DOUTDIN in register SPI_USER_REG. Figure 20-7 illustrates the connection of GP-SPI2 with its slave in full-duplex communication.

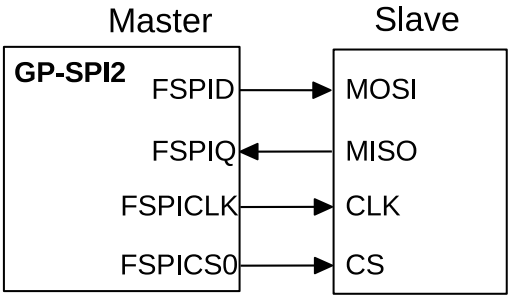


Figure 20-7. Full-Duplex Communication Between GP-SPI2 Master and a Slave

In full-duplex communication, the behavior of states CMD, ADDR, DUMMY, DOUT and DIN are configurable. Usually, the states CMD, ADDR and DUMMY are not used in this communication. The bit length of transferred data is configured in SPI_MS_DATA_BITLEN. The actual bit length used in communication equals to (SPI_MS_DATA_BITLEN + 1).

Configuration

To start a data transfer, follow the steps below:

- Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
- Configure APB clock (APB_CLK, see Chapter 6 *Reset and Clock*) and module clock (clk_spi_mst) for the GP-SPI2 module.
- Set SPI_DOUTDIN and clear SPI_SLAVE_MODE, to enable full-duplex communication in master mode.
- Configure GP-SPI2 registers listed in Table 20-8.
- Configure SPI CS setup time and hold time according to Section 20.6.
- Set the property of FSPICLK according to Section 20.7.

- Prepare data according to the selected transfer mode:
  - In CPU-controlled MOSI mode, prepare data in registers SPI_W0_REG ~ SPI_W15_REG.
  - In DMA-controlled mode,
    * configure SPI_DMA_TX_ENA/SPI_DMA_RX_ENA
    * configure GDMA TX/RX link
    * start GDMA TX/RX engine, as described in Section 20.5.6 and Section 20.5.7.
- Configure interrupts and wait for SPI slave to get ready for transfer.
- Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
- Set SPI_USR in register SPI_CMD_REG to start the transfer and wait for the configured interrupts.

20.5.8.4 Half-Duplex Communication (1/2/4-bit Mode)

Introduction

In this mode, GP-SPI2 provides CLK and CS signals. Only one side (SPI master or slave) can send data at a time, while the other side receives the data. To enable this communication mode, clear the bit SPI_DOUTDIN in register SPI_USER_REG. The standard format of SPI half-duplex communication is CMD + [ADDR +] [DUMMY +] [DOUT or DIN]. The states ADDR, DUMMY, DOUT, and DIN are optional, and can be disabled or enabled independently.

As described in Section 20.5.8.2, the properties of GP-SPI2 states: CMD, ADDR, DUMMY, DOUT and DIN, such as cycle length, value, and parallel bus bit mode, can be set independently. For the register configuration, see Table 20-8.

The detailed properties of half-duplex GP-SPI2 are as follows:

1. CMD: 0 ~ 16 bits, master output, slave input.
2. ADDR: 0 ~ 32 bits, master output, slave input.
3. DUMMY: 0 ~ 256 FSPICLK cycles, master output, slave input.
4. DOUT: 0 ~ 512 bits (64 B) in CPU-controlled mode and 0 ~ 256 Kbits (32 KB) in DMA-controlled mode, master output, slave input.
5. DIN: 0 ~ 512 bits (64 B) in CPU-controlled mode and 0 ~ 256 Kbits (32 KB) in DMA-controlled mode, master input, slave output.

Configuration

The register configuration is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure APB clock (APB_CLK) and module clock (clk_spi_mst) for the GP-SPI2 module.
3. Clear SPI_DOUTDIN and SPI_SLAVE_MODE, to enable half-duplex communication in master mode.
4. Configure GP-SPI2 registers listed in Table 20-8.
5. Configure SPI CS setup time and hold time according to Section 20.6.
6. Set the property of FSPICLK according to Section 20.7.

7. Prepare data according to the selected transfer mode:
   - In CPU-controlled MOSI mode, prepare data in registers SPI_W0_REG ~ SPI_W15_REG.
   - In DMA-controlled mode,
     - configure SPI_DMA_TX_ENA/SPI_DMA_RX_ENA
     - configure GDMA TX/RX link
     - start GDMA TX/RX engine, as described in Section 20.5.6 and Section 20.5.7.
8. Configure interrupts and wait for SPI slave to get ready for transfer.
9. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
10. Set SPI_USR in register SPI_CMD_REG to start the transfer and wait for the configured interrupts.

Application Example

The following example shows how GP-SPI2 accesses flash and external RAM in master half-duplex mode.

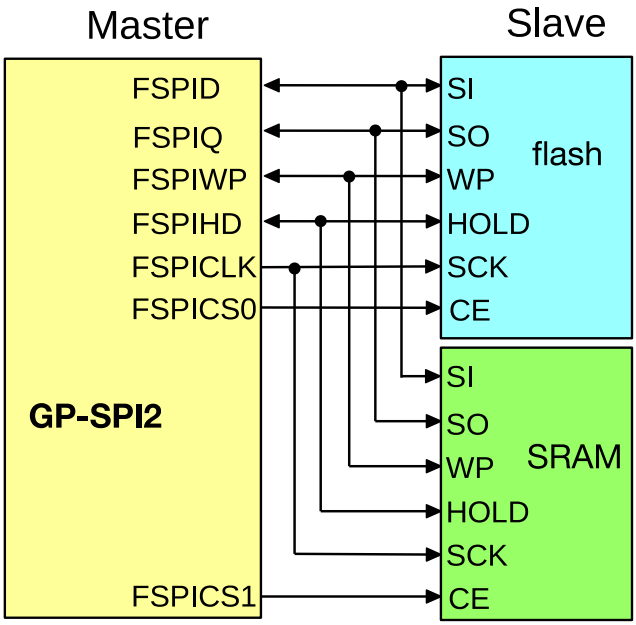


Figure 20-8. Connection of GP-SPI2 to Flash and External RAM in 4-bit Mode

Figure 20-9 indicates GP-SPI2 Quad I/O Read sequence according to standard flash specification. Other GP-SPI2 command sequences are implemented in accordance with the requirements of SPI slaves.

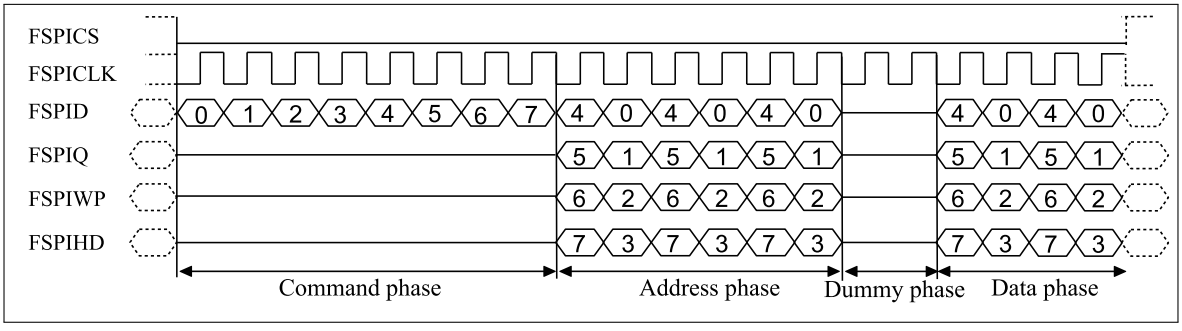


Figure 20-9. SPI Quad I/O Read Command Sequence Sent by GP-SPI2 to Flash

PRELIMINARY

20.5.8.5 DMA-Controlled Configurable Segmented Transfer

Note:
Note that there is no separate section on how to configure a single transfer in master mode, since the CONF state of a configurable segmented transfer can be skipped to implement a single transfer.

Introduction

When GP-SPI2 works as a master, it provides a feature named: configurable segmented transfer controlled by DMA.

A DMA-controlled transfer in master mode can be

- a single transfer, consisting of only one transaction;
- or a configurable segmented transfer, consisting of several transactions (segments).

In a configurable segmented transfer, the registers of each single transaction (segment) are configurable. This feature enables GP-SPI2 to do as many transactions (segments) as configured after such transfer is triggered once by the CPU. Figure 20-10 shows how this feature works.

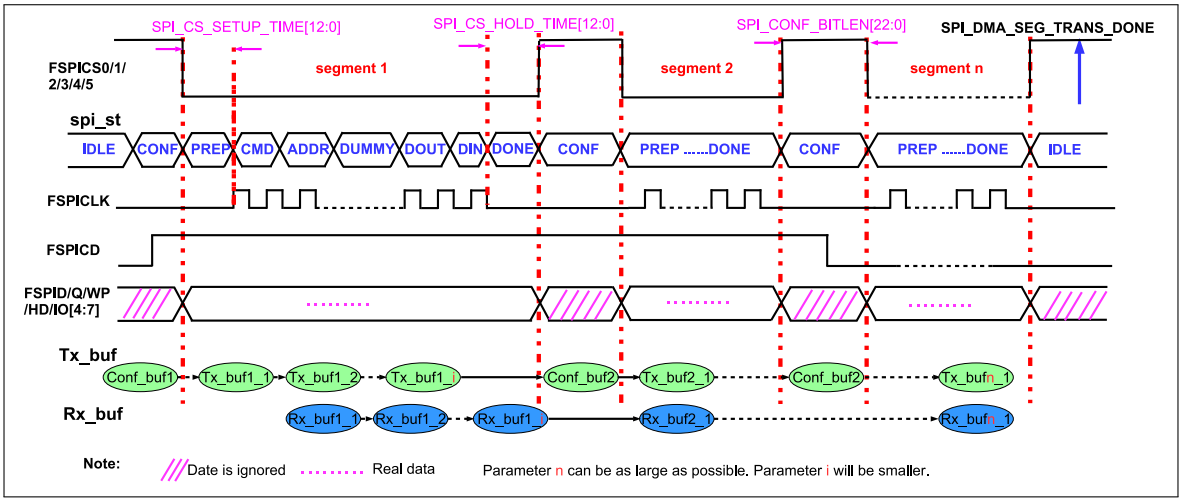


Figure 20-10. Configurable Segmented Transfer in DMA-Controlled Master Mode

As shown in Figure 20-10, the registers for one transaction (segment *n*) can be reconfigured by GP-SPI2 hardware according to the content in its Conf_buf*n* during a CONF state, before this segment starts.

It's recommended to provide separate GDMA CONF links and CONF buffers (Conf_buf*i* in Figure 20-10) for each CONF state. A GDMA TX link is used to connect all the CONF buffers and TX data buffers (Tx_buf*i* in Figure 20-10) into a chain. Hence, the behavior of the FSPI bus in each segment can be controlled independently.

For example, in a configurable segmentent transfer, its segment*i*, segment*j*, and segment*k* can be configured to full-duplex, half-duplex MISO, and half-duplex MOSI, respectively. *i*, *j*, and *k* represent different segment numbers.

Meanwhile, the state of GP-SPI2, the data length and cycle length of the FSPI bus, and the behavior of the GDMA, can be configured independently for each segment. When this whole DMA-controlled transfer (consisting of several segments) has finished, a GP-SPI2 interrupt, SPI_DMA_SEG_TRANS_DONE_INT, is triggered.

Configuration

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure APB clock (APB_CLK) and module clock (clk_spi_mst) for GP-SPI2 module.
3. Clear SPI_DOUTDIN and SPI_SLAVE_MODE, to enable half-duplex communication in master mode.
4. Configure GP-SPI2 registers listed in Table 20-8.
5. Configure SPI CS setup time and hold time according to Section 20.6.
6. Set the property of FSPICLK according to Section 20.7.
7. Prepare descriptors for GDMA CONF buffer and TX data (optional) for each segment. Chain the descriptors of CONF buffer and TX buffers of several segments into one linked list.
8. Similarly, prepare descriptors for RX buffers for each segment and chain them into one linked list.
9. Configure all the needed CONF buffers, TX buffers and RX buffers, respectively for each segment before this DMA-controlled transfer begins.
10. Point GDMA_OUTLINK_ADDR_CH*n* to the head address of the CONF and TX buffer descriptor linked list, and then set GDMA_OUTLINK_START_CH*n* to start the TX GDMA.
11. Clear the bit SPI_RX_EOF_EN in register SPI_DMA_CONF_REG. Point GDMA_INLINK_ADDR_CH*n* to the head address of the RX buffer descriptor linked list, and then set GDMA_INLINK_START_CH*n* to start the RX GDMA.
12. Set SPI_USR_CONF to enable CONF state.
13. Set SPI_DMA_SEG_TRANS_DONE_INT_ENA to enable the SPI_DMA_SEG_TRANS_DONE_INT interrupt. Configure other interrupts if needed according to Section 20.9.
14. Wait for all the slaves to get ready for transfer.
15. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST and SPI_RX_AFIFO_RST, to reset these buffers.
16. Set SPI_USR to start this DMA-controlled transfer.
17. Wait for SPI_DMA_SEG_TRANS_DONE_INT interrupt, which means this transfer has finished and the data has been stored into corresponding memory.

Configuration of CONF Buffer and Magic Value

In a configurable segmented transfer, only registers which will change from the last transaction (segment) need to be re-configured to new values in CONF state. The configuration of other registers can be skipped (i.e. kept the same) to save time and chip resources.

The first word in GDMA CONF buffer*i*, called SPI_BIT_MAP_WORD, defines whether each GP-SPI2 register is to be updated or not in segment*i*. The relation of SPI_BIT_MAP_WORD and GP-SPI2 registers to update can be seen in Bitmap (BM) Table, Table 20-11. If a bit in the BM table is set to 1, its corresponding register value will be updated in this segment. Otherwise, if some registers should be kept from being changed, the related bits should be set to 0.

Table 20-11. BM Table for CONF State

BM Bit	Register Name	BM Bit	Register Name
0	SPI_ADDR_REG	7	SPI_MISC_REG
1	SPI_CTRL_REG	8	reserved

2	SPI_CLOCK_REG	9	reserved
3	SPI_USER_REG	10	reserved
4	SPI_USER1_REG	11	SPI_DMA_CONF_REG
5	SPI_USER2_REG	12	SPI_DMA_INT_ENA_REG
6	SPI_MS_DLEN_REG	13	SPI_DMA_INT_CLR_REG

Then new values of all the registers to be modified should be placed right after SPI_BIT_MAP_WORD, in consecutive words in the CONF buffer.

To ensure the correctness of the content in each CONF buffer, the value in SPI_BIT_MAP_WORD[31:28] is used as "magic value", and will be compared with SPI_DMA_SEG_MAGIC_VALUE in register SPI_SLAVE_REG. The value of SPI_DMA_SEG_MAGIC_VALUE should be configured before this DMA-controlled transfer starts, and can not be changed during these segments.

- If SPI_BIT_MAP_WORD[31:28] == SPI_DMA_SEG_MAGIC_VALUE, this DMA-controlled transfer continues normally; the interrupt SPI_DMA_SEG_TRANS_DONE_INT is triggered at the end of this DMA-controlled transfer.
- If SPI_BIT_MAP_WORD[31:28] != SPI_DMA_SEG_MAGIC_VALUE, GP-SPI2 state (spi_st) goes back to IDLE and the transfer is ended immediately. The interrupt SPI_DMA_SEG_TRANS_DONE_INT is still triggered, with SPI_SEG_MAGIC_ERR_INT_RAW bit set to 1.

CONF Buffer Configuration Example

Table 20-12 and Table 20-13 provide an example to show how to configure a CONF buffer for a transaction (segment *i*) in which SPI_ADDR_REG, SPI_CTRL_REG, SPI_CLOCK_REG, SPI_USER_REG, SPI_USER1_REG need to be updated.

Table 20-12. An Example of CONF buffer*i* in Segment*i*

CONF buffer*i*	Note
SPI_BIT_MAP_WORD	The first word in this buffer. Its value is 0xA000001F in this example when the SPI_DMA_SEG_MAGIC_VALUE is set to 0xA. As shown in Table 20-13, bits 0, 1, 2, 3, and 4 are set, indicating the following registers will be updated.
SPI_ADDR_REG	The second word, stores the new value to SPI_ADDR_REG.
SPI_CTRL_REG	The third word, stores the new value to SPI_CTRL_REG.
SPI_CLOCK_REG	The fourth word, stores the new value to SPI_CLOCK_REG.
SPI_USER_REG	The fifth word, stores the new value to SPI_USER_REG.
SPI_USER1_REG	The sixth word, stores the new value to SPI_USER1_REG.

Table 20-13. BM Bit Value v.s. Register to Be Updated in This Example

BM Bit	Value	Register Name	BM Bit	Value	Register Name
0	1	SPI_ADDR_REG	7	0	SPI_MISC_REG
1	1	SPI_CTRL_REG	8	0	reserved
2	1	SPI_CLOCK_REG	9	0	reserved
3	1	SPI_USER_REG	10	0	reserved
4	1	SPI_USER1_REG	11	0	SPI_DMA_CONF_REG
5	0	SPI_USER2_REG	12	0	SPI_DMA_INT_ENA_REG
6	0	SPI_MS_DLEN_REG	13	0	SPI_DMA_INT_CLR_REG

Notes:

In a DMA-controlled configurable segmented transfer, please pay special attention to the following bits:

- SPI_USR_CONF: set SPI_USR_CONF before SPI_USR is set, to enable this transfer.
- SPI_USR_CONF_NXT: if segment*i* is not the final transaction of this whole DMA-controlled transfer, its SPI_USR_CONF_NXT bit should be set to 1.
- SPI_CONF_BITLEN: GP-SPI2 CS setup time and hold time are programmable independently in each segment, see Section 20.6 for detailed configuration. The CS high time in each segment is about:

$$(SPI\_CONF\_BITLEN + 5) \times T_{APB\_CLK}$$

  The CS high time in CONF state can be set from 125 μs to 6.5536 ms when $f_{\text{APB\_CLK}}$ is 40 MHz. (SPI_CONF_BITLEN + 5) will overflow from (0x40000 - SPI_CONF_BITLEN - 5) if SPI_CONF_BITLEN is larger than 0x3FFFA.

20.5.9 GP-SPI2 Works as a Slave

GP-SPI2 can be used as a slave to communicate with an SPI master. As a slave, GP-SPI2 supports 1-bit SPI, 2-bit dual SPI, 4-bit quad SPI, and QPI modes, with specific communication formats. To enable this mode, set SPI_SLAVE_MODE in register SPI_SLAVE_REG.

The CS signal must be held low during the transmission, and its falling/rising edges indicate the start/end of a single or segmented transmission. The length of transferred data must be in unit of bytes, otherwise the extra bits will be lost. The extra bits here means the result of total bits % 8.

20.5.9.1 Communication Formats

In GP-SPI2 slave mode, SPI full-duplex and half-duplex communications are available. To select from the two communications, configure SPI_DOUTDIN in register SPI_USER_REG.

Full-duplex communication means that input data and output data are transmitted simultaneously throughout the entire transaction. All bits are treated as input or output data, which means no command, address or dummy states are expected. The interrupt SPI_TRANS_DONE_INT is triggered once the transaction ends.

In half-duplex communication, the format is CMD+ADDR+DUMMY+DATA (DIN or DOUT).

- "DIN" means that an SPI master reads data from GP-SPI2.
- "DOUT" means that an SPI master writes data to GP-SPI2.

The detailed properties of each state are as follows:

1. CMD:
   - Indicate the function of SPI slave;
   - One byte from master to slave;
   - Only the values in Table 20-14 and Table 20-15 are valid;
   - Can be sent in 1-bit SPI mode or 4-bit QPI mode.
2. ADDR:
   - The address for Wr_BUF and Rd_BUF commands in CPU-controlled transfer, or placeholder bits in other transfers and can be defined by application;
   - One byte from master to slave;
   - Can be sent in 1-bit, 2-bit or 4-bit modes (according to the command).
3. DUMMY:
   - Its value is meaningless. SPI slave prepares data in this state;
   - Bit mode of FSPI bus is also meaningless here;
   - Last for eight SPI_CLK cycles.
4. DIN or DOUT:
   - Data length can be 0 ~ 64 B in CPU-controlled mode and unlimited in DMA-controlled mode;
   - Can be sent in 1-bit, 2-bit or 4-bit modes according to the CMD value.

Note:
The states of ADDR and DUMMY can never be skipped in any half-duplex communications.

When a half-duplex transaction is complete, the transferred CMD and ADDR values are latched into SPI_SLV_LAST_COMMAND and SPI_SLV_LAST_ADDR respectively. The SPI_SLV_CMD_ERR_INT_RAW will be set if the transferred CMD value is not supported by GP-SPI2 slave mode. The SPI_SLV_CMD_ERR_INT_RAW can only be cleared by software.

20.5.9.2 Supported CMD Values in Half-Duplex Communication

In half-duplex communication, the defined values of CMD determine the transfer types. Unsupported CMD values are disregarded, meanwhile the related transfer is ignored and SPI_SLV_CMD_ERR_INT_RAW is set. The transfer format is CMD (8 bits) + ADDR (8 bits) + DUMMY (8 SPI_CLK cycles) + DATA (unit in bytes). The detailed description of CMD[3:0] is as follows:

- 0x1 (Wr_BUF): CPU-controlled write mode. Master sends data and GP-SPI2 receives data. The data is stored in the related address of SPI_W0_REG ~ SPI_W15_REG.
- 0x2 (Rd_BUF): CPU-controlled read mode. Master receives the data sent by GP-SPI2. The data comes from the related address of SPI_W0_REG ~ SPI_W15_REG.
- 0x3 (Wr_DMA): DMA-controlled write mode. Master sends data and GP-SPI2 receives data. The data is stored in GP-SPI2 GDMA RX buffer.

- 0x4 (Rd_DMA): DMA-controlled read mode. Master receives the data sent by GP-SPI2. The data comes from GP-SPI2 GDMA TX buffer.
- 0x7 (CMD7): used to generate an SPI_SLV_CMD7_INT interrupt. It can also generate a GDMA_IN_SUC_EOF_CH*n*_INT interrupt in a slave segmented transfer when GDMA RX link is used. But it will not end GP-SPI2's slave segmented transfer.
- 0x8 (CMD8): only used to generate an SPI_SLV_CMD8_INT interrupt, which will not end GP-SPI2's slave segmented transfer.
- 0x9 (CMD9): only used to generate an SPI_SLV_CMD9_INT interrupt, which will not end GP-SPI2's slave segmented transfer.
- 0xA (CMDA): only used to generate an SPI_SLV_CMDA_INT interrupt, which will not end GP-SPI2's slave segmented transfer.

The detailed function of CMD7, CMD8, CMD9, and CMDA commands is reserved for user definition. These commands can be used as handshake signals, as passwords of some specific functions, as triggers of some user defined actions, and so on.

1/2/4-bit modes in states of CMD, ADDR, DATA are supported, which are determined by value of CMD[7:4]. The DUMMY state is always in 1-bit mode and lasts for eight SPI_CLK cycles. The definition of CMD[7:4] is as follows:

- 0x0: CMD, ADDR, and DATA states all are in 1-bit mode.
- 0x1: CMD and ADDR are in 1-bit mode. DATA is in 2-bit mode.
- 0x2: CMD and ADDR are in 1-bit mode. DATA is in 4-bit mode.
- 0x5: CMD is in 1-bit mode. ADDR and DATA are in 2-bit mode.
- 0xA: CMD is in 1-bit mode, ADDR and DATA are in 4-bit mode. Or in QPI mode.

In addition, if the value of CMD[7:0] is 0x05, 0xA5, 0x06, or 0xDD, DUMMY and DATA states are skipped. The definition of CMD[7:0] is as follows:

- 0x05 (End_SEG_TRANS): master sends 0x05 command to end slave segmented transfer in SPI mode.
- 0xA5 (End_SEG_TRANS): master sends 0xA5 command to end slave segmented transfer in QPI mode.
- 0x06 (En_QPI): GP-SPI2 enters QPI mode when receiving the 0x06 command and the bit SPI_QPI_MODE in register SPI_USER_REG is set.
- 0xDD (Ex_QPI): GP-SPI2 exits QPI mode when receiving the 0xDD command and the bit SPI_QPI_MODE is cleared.

All the CMD values supported by GP-SPI2 are listed in Table 20-14 and Table 20-15. Note that DUMMY state is always in 1-bit mode and lasts for eight SPI_CLK cycles.

Table 20-14. Supported CMD Values in SPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
Wr_BUF	0x01	1-bit mode	1-bit mode	1-bit mode
	0x11	1-bit mode	1-bit mode	2-bit mode
	0x21	1-bit mode	1-bit mode	4-bit mode
	0x51	1-bit mode	2-bit mode	2-bit mode

Table 20-14. Supported CMD Values in SPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
	0xA1	1-bit mode	4-bit mode	4-bit mode
Rd_BUF	0x02	1-bit mode	1-bit mode	1-bit mode
	0x12	1-bit mode	1-bit mode	2-bit mode
	0x22	1-bit mode	1-bit mode	4-bit mode
	0x52	1-bit mode	2-bit mode	2-bit mode
	0xA2	1-bit mode	4-bit mode	4-bit mode
Wr_DMA	0x03	1-bit mode	1-bit mode	1-bit mode
	0x13	1-bit mode	1-bit mode	2-bit mode
	0x23	1-bit mode	1-bit mode	4-bit mode
	0x53	1-bit mode	2-bit mode	2-bit mode
	0xA3	1-bit mode	4-bit mode	4-bit mode
Rd_DMA	0x04	1-bit mode	1-bit mode	1-bit mode
	0x14	1-bit mode	1-bit mode	2-bit mode
	0x24	1-bit mode	1-bit mode	4-bit mode
	0x54	1-bit mode	2-bit mode	2-bit mode
	0xA4	1-bit mode	4-bit mode	4-bit mode
CMD7	0x07	1-bit mode	1-bit mode	-
	0x17	1-bit mode	1-bit mode	-
	0x27	1-bit mode	1-bit mode	-
	0x57	1-bit mode	2-bit mode	-
	0xA7	1-bit mode	4-bit mode	-
CMD8	0x08	1-bit mode	1-bit mode	-
	0x18	1-bit mode	1-bit mode	-
	0x28	1-bit mode	1-bit mode	-
	0x58	1-bit mode	2-bit mode	-
	0xA8	1-bit mode	4-bit mode	-
CMD9	0x09	1-bit mode	1-bit mode	-
	0x19	1-bit mode	1-bit mode	-
	0x29	1-bit mode	1-bit mode	-
	0x59	1-bit mode	2-bit mode	-
	0xA9	1-bit mode	4-bit mode	-
CMDA	0x0A	1-bit mode	1-bit mode	-
	0x1A	1-bit mode	1-bit mode	-
	0x2A	1-bit mode	1-bit mode	-
	0x5A	1-bit mode	2-bit mode	-
	0xAA	1-bit mode	4-bit mode	-
End_SEG_TRANS	0x05	1-bit mode	-	-
En_QPI	0x06	1-bit mode	-	-

Table 20-15. Supported CMD Values in QPI Mode

Transfer Type	CMD[7:0]	CMD State	ADDR State	DATA State
Wr_BUF	0xA1	4-bit mode	4-bit mode	4-bit mode
Rd_BUF	0xA2	4-bit mode	4-bit mode	4-bit mode
Wr_DMA	0xA3	4-bit mode	4-bit mode	4-bit mode
Rd_DMA	0xA4	4-bit mode	4-bit mode	4-bit mode
CMD7	0xA7	4-bit mode	4-bit mode	-
CMD8	0xA8	4-bit mode	4-bit mode	-
CMD9	0xA9	4-bit mode	4-bit mode	-
CMDA	0xAA	4-bit mode	4-bit mode	-
End_SEG_TRANS	0xA5	4-bit mode	4-bit mode	-
Ex_QPI	0xDD	4-bit mode	4-bit mode	-

Master sends 0x06 CMD (En_QPI) to set GP-SPI2 slave to QPI mode and all the states of supported transfer will be in 4-bit mode afterwards. If 0xDD CMD (Ex_QPI) is received, GP-SPI2 slave will be back to SPI mode.

Other transfer types than described in Table 20-14 and Table 20-15 are ignored. If the transferred data is not in unit of byte, GP-SPI2 will send or receive the data in unit of byte, but the extra bits (the result of total bits mod 8) will be lost. But if the CS low time is longer than 2 APB clock (APB_CLK) cycles, SPI_TRANS_DONE_INT will be triggered. For more information on interrupts triggered at the end of transmissions, please refer to Section 20.9.

20.5.9.3 Slave Single Transfer and Slave Segmented Transfer

When GP-SPI2 works as a slave, it supports full-duplex and half-duplex communications controlled by DMA and by CPU. DMA-controlled transfer can be a single transfer, or a slave segmented transfer consisting of several transactions (segments). The CPU-controlled transfer can only be one single transfer, since each CPU-controlled transaction needs to be triggered by CPU.

In a slave segmented transfer, all transfer types listed in Table 20-14 and Table 20-15 are supported in a single transaction (segment). It means that CPU-controlled transaction and DMA-controlled transaction can be mixed in one slave segmented transfer.

It is recommended that in a slave segmented transfer:

- CPU-controlled transaction is used for handshake communication and short data transfers.
- DMA-controlled transaction is used for large data transfers.

20.5.9.4 Configuration of Slave Single Transfer

In slave mode, GP-SPI2 supports CPU/DMA-controlled full-duplex/half-duplex single transfers. The register configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure APB clock (APB_CLK).
3. Set the bit SPI_SLAVE_MODE, to enable slave mode.

4. Configure SPI_DOUTDIN:
   - 1: enable full-duplex communication.
   - 0: enable half-duplex communication.
5. Prepare data:
   - if CPU-controlled transfer mode is selected and GP-SPI2 is used to send data, then prepare data in registers SPI_W0_REG ~ SPI_W15_REG.
   - if DMA-controlled transfer mode is selected,
     - configure SPI_DMA_TX_ENA/SPI_DMA_RX_ENA and SPI_RX_EOF_EN.
     - configure GDMA TX/RX link.
     - start GDMA TX/RX engine, as described in Section 20.5.6 and Section 20.5.7.
6. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST to reset these buffers.
7. Clear SPI_DMA_SLV_SEG_TRANS_EN in register SPI_DMA_CONF_REG to enable slave single transfer mode.
8. Set SPI_TRANS_DONE_INT_ENA in SPI_DMA_INT_ENA_REG and wait for the interrupt SPI_TRANS_DONE_INT. In DMA-controlled mode, it is recommended to wait for the interrupt GDMA_IN_SUC_EOF_CH*n*_INT when GDMA RX buffer is used, which means that data has been stored in the related memory. Other interrupts described in Section 20.9 are optional.

20.5.9.5 Configuration of Slave Segmented Transfer in Half-Duplex

GDMA must be used in this mode. The register configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure APB clock (APB_CLK).
3. Set SPI_SLAVE_MODE to enable slave mode.
4. Clear SPI_DOUTDIN to enable half-duplex communication.
5. Prepare data in registers SPI_W0_REG ~ SPI_W15_REG, if needed.
6. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST and SPI_RX_AFIFO_RST to reset these buffers.
7. Set bits SPI_DMA_RX_ENA and SPI_DMA_TX_ENA. Clear the bit SPI_RX_EOF_EN. Configure GDMA TX/RX link and start GDMA TX/RX engine, as shown in Section 20.5.6 and Section 20.5.7.
8. Set SPI_DMA_SLV_SEG_TRANS_EN in SPI_DMA_CONF_REG to enable slave segmented transfer.
9. Set SPI_DMA_SEG_TRANS_DONE_INT_ENA in SPI_DMA_INT_ENA_REG and wait for the interrupt SPI_DMA_SEG_TRANS_DONE_INT, which means that the segmented transfer has finished and data has been put into the related memory. Other interrupts described in Section 20.9 are optional.

When End_SEG_TRANS (0x05 in SPI mode, 0xA5 in QPI mode) is received by GP-SPI2, this slave segmented transfer is ended and the interrupt SPI_DMA_SEG_TRANS_DONE_INT is triggered.

20.5.9.6 Configuration of Slave Segmented Transfer in Full-Duplex

GDMA must be used in this mode. In such transfer, the data is transferred from and to the GDMA buffer. The interrupt GDMA_IN_SUC_EOF_CH*n*
_INT is triggered when the transfer ends. The configuration procedure is as follows:

1. Configure the IO path via IO MUX or GPIO matrix between GP-SPI2 and an external SPI device.
2. Configure APB clock (APB_CLK).
3. Set SPI_SLAVE_MODE and SPI_DOUTDIN, to enable full-duplex communication in slave mode.
4. Set SPI_DMA_AFIFO_RST, SPI_BUF_AFIFO_RST, and SPI_RX_AFIFO_RST, to reset these buffers.
5. Set SPI_DMA_TX_ENA/SPI_DMA_RX_ENA. Configure GDMA TX/RX link and start GDMA TX/RX engine, as shown in Section 20.5.6 and Section 20.5.7.
6. Set the bit SPI_RX_EOF_EN in register SPI_DMA_CONF_REG. Configure SPI_MS_DATA_BITLEN[17:0] in register SPI_MS_DLEN_REG to the byte length of the received DMA data.
7. Set SPI_DMA_SLV_SEG_TRANS_EN in SPI_DMA_CONF_REG to enable slave segmented transfer mode.
8. Set GDMA_IN_SUC_EOF_CH*n*_INT_ENA and wait for the interrupt GDMA_IN_SUC_EOF_CH*n*_INT.

20.6 CS Setup Time and Hold Time Control

SPI bus CS (SPI_CS) setup time and hold time are very important to meet the timing requirements of various SPI devices (e.g. flash or PSRAM).

CS setup time is the time between the CS falling edge and the first latch edge of SPI bus CLK (SPI_CLK). The first latch edge for mode 0 and mode 3 is rising edge, and falling edge for mode 2 and mode 4.

CS hold time is the time between the last latch edge of SPI_CLK and the CS rising edge.

In slave mode, the CS setup time and hold time should be longer than 0.5 x T_SPI_CLK, otherwise the SPI transfer may be incorrect. T_SPI_CLK is one cycle of SPI_CLK.

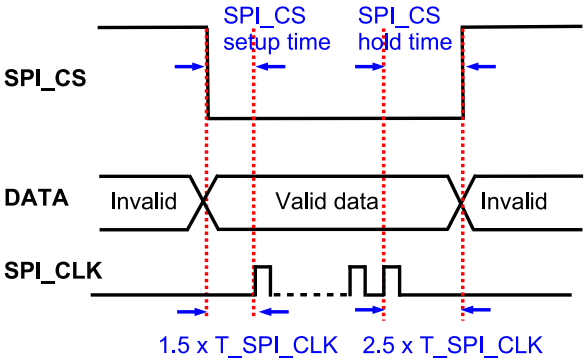
In master mode, set the CS setup time by specifying SPI_CS_SETUP in SPI_USER_REG and SPI_CS_SETUP_TIME in SPI_USER1_REG:

- If SPI_CS_SETUP is cleared, the SPI CS setup time is 0.5 x T_SPI_CLK.
- If SPI_CS_SETUP is set, the SPI CS setup time is (SPI_CS_SETUP_TIME + 1.5) x T_SPI_CLK.

Set the CS hold time by specifying SPI_CS_HOLD in SPI_USER_REG and SPI_CS_HOLD_TIME in SPI_USER1_REG:

- If SPI_CS_HOLD is cleared, the SPI CS hold time is 0.5 x T_SPI_CLK;
- If SPI_CS_HOLD is set, the SPI CS hold time is (SPI_CS_HOLD_TIME + 1.5) x T_SPI_CLK.

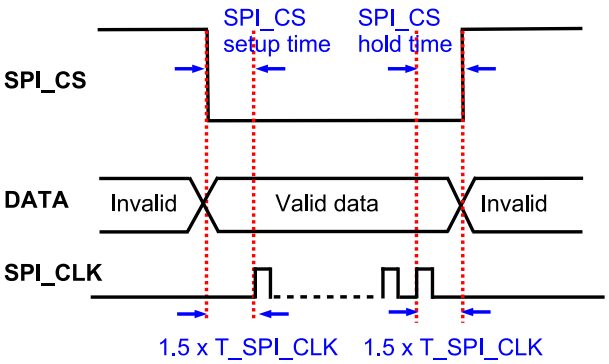
Figure 20-11 and Figure 20-12 show the recommended CS timing and register configuration to access external RAM and flash.



Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 1.

Figure 20-11. Recommended CS Timing and Settings When Accessing External RAM



Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 0.

Figure 20-12. Recommended CS Timing and Settings When Accessing Flash

20.7 GP-SPI2 Clock Control

GP-SPI2 has the following clocks:

- clk_spi_mst: module clock of GP-SPI2, derived from PLL_CLK. Used in GP-SPI2 master mode, to generate SPI_CLK signal for data transfer and for slaves.
- SPI_CLK: output clock in master mode.
- APB_CLK: clock for register configuration.

In master mode, the maximum output clock frequency of GP-SPI2 is $f_{clk\_spi\_mst}$. To have slower frequencies, the output clock frequency can be divided as follows:

$$f_{SPI\_CLK} = \frac{f_{clk\_spi\_mst}}{(SPI\_CLKCNT\_N + 1)(SPI\_CLKDIV\_PRE + 1)}$$

The divider is configured by SPI_CLKCNT_N and SPI_CLKDIV_PRE in register SPI_CLOCK_REG. When the bit SPI_CLK_EQU_SYSCLK in register SPI_CLOCK_REG is set to 1, the output clock frequency of GP-SPI2 will be $f_{clk\_spi\_mst}$. For other integral clock divisions, SPI_CLK_EQU_SYSCLK should be set to 0.

In slave mode, the supported input clock frequency (f_{SPI_CLK}) of GP-SPI2 is:

$$f_{SPI\_CLK} <= 40MHz$$

20.7.1 Clock Phase and Polarity

SPI protocol has four clock modes, modes 0 ~ 3, see Figure 20-13 and Figure 20-14 (excerpted from SPI protocol):

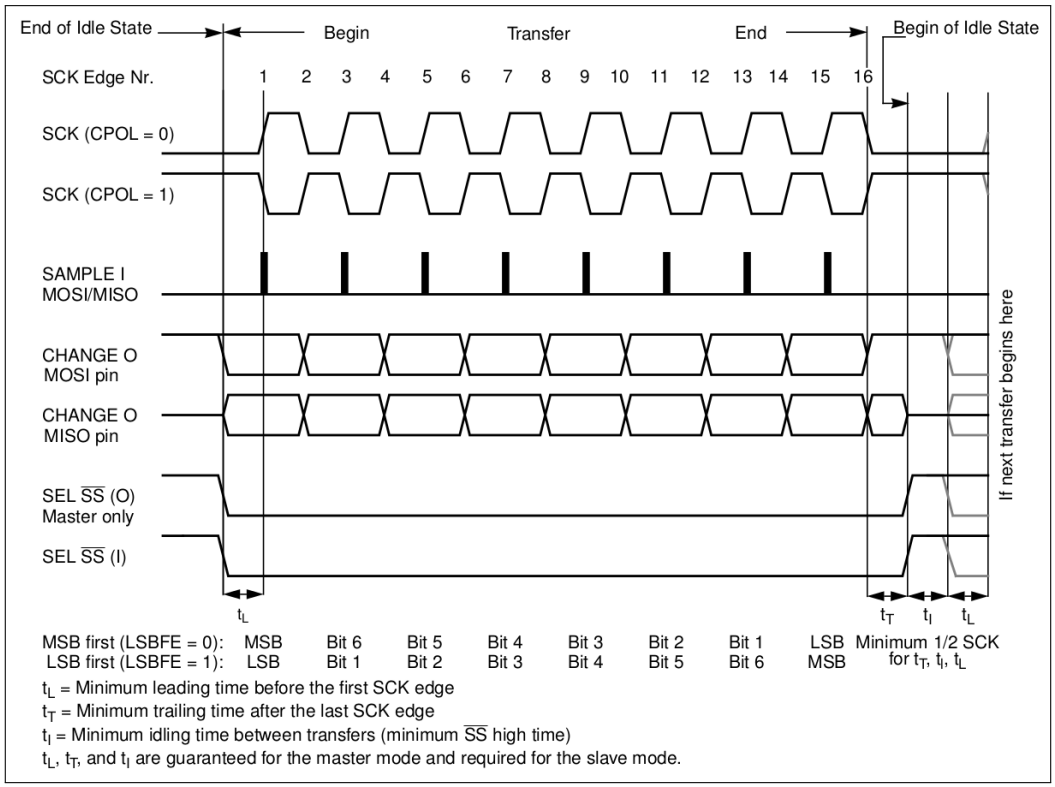


Figure 20-13. SPI Clock Mode 0 or 2

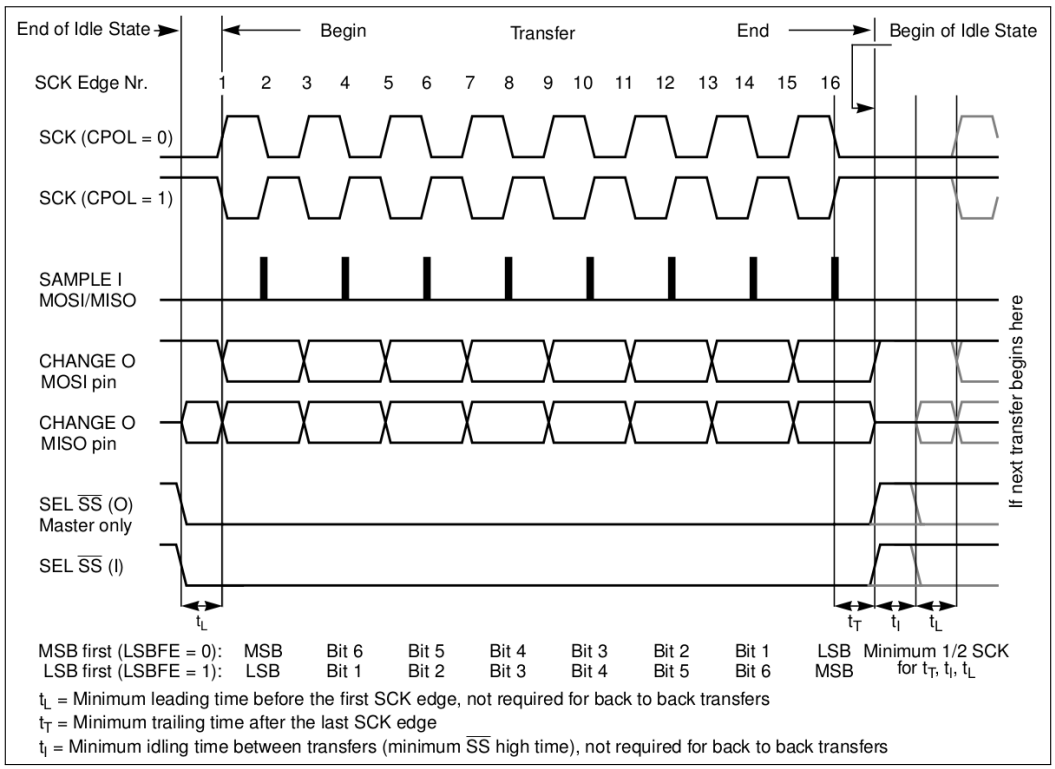


Figure 20-14. SPI Clock Mode 1 or 3

1. Mode 0: CPOL = 0, CPHA = 0; SCK is 0 when the SPI is in idle state; data is changed on the negative edge of SCK and sampled on the positive edge. The first data is shifted out before the first negative edge of SCK.
2. Mode 1: CPOL = 0, CPHA = 1; SCK is 0 when the SPI is in idle state; data is changed on the positive edge of SCK and sampled on the negative edge.
3. Mode 2: CPOL = 1, CPHA = 0; SCK is 1 when the SPI is in idle state; data is changed on the positive edge of SCK and sampled on the negative edge. The first data is shifted out before the first positive edge of SCK.
4. Mode 3: CPOL = 1, CPHA = 1; SCK is 1 when the SPI is in idle state; data is changed on the negative edge of SCK and sampled on the positive edge.

20.7.2 Clock Control in Master Mode

The four clock modes 0 ~ 3 are supported in GP-SPI2 master mode. The polarity and phase of GP-SPI2 clock are controlled by the bit SPI_CK_IDLE_EDGE in register SPI_MISC_REG and the bit SPI_CK_OUT_EDGE in register SPI_USER_REG. The register configuration for SPI clock modes 0 ~ 3 is provided in Table 20-16, and can be changed according to the path delay in the application.

Table 20-16. Clock Phase and Polarity Configuration in Master Mode

Control Bit	Mode 0	Mode 1	Mode 2	Mode 3
SPI_CK_IDLE_EDGE	0	0	1	1
SPI_CK_OUT_EDGE	0	1	1	0

SPI_CLK_MODE is used to select the number of rising edges of SPI_CLK, when SPI_CS raises high, to be 0, 1, 2 or SPI_CLK always on.

Note:
When SPI_CLK_MODE is configured to 1 or 2, the bit SPI_CS_HOLD must be set and the value of SPI_CS_HOLD_TIME should be larger than 1.

20.7.3 Clock Control in Slave Mode

GP-SPI2 slave mode also supports clock modes 0 ~ 3. The polarity and phase are configured by the bits SPI_TSCK_I_EDGE and SPI_RSCK_I_EDGE in register SPI_USER_REG. The output edge of data is controlled by SPI_CLK_MODE_13 in register SPI_SLAVE_REG. The detailed register configuration is shown in Table 20-17:

Table 20-17. Clock Phase and Polarity Configuration in Slave Mode

Control Bit	Mode 0	Mode 1	Mode 2	Mode 3
SPI_TSCK_I_EDGE	0	1	1	0
SPI_RSCK_I_EDGE	0	1	1	0
SPI_CLK_MODE_13	0	1	0	1

20.8 GP-SPI2 Timing Compensation

The I/O lines are mapped via GPIO matrix or IO MUX. But there is no timing adjustment in IO MUX. The input data and output data can be delayed for 1 or 2 APB_CLK cycles at the rising or falling edge in GPIO matrix. For detailed register configuration, see Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO MUX)*.

In GP-SPI2 slave mode, if the bit SPI_RSCK_DATA_OUT in register SPI_SLAVE_REG is set to 1, the output data is sent at latch edge, which is half an SPI clock cycle earlier. This can be used for slave mode timing compensation.

20.9 Interrupts

Interrupt Summary

GP-SPI2 provides an SPI interface interrupt SPI_INT. When an SPI transfer ends, an interrupt is generated in GP-SPI2.

- SPI_DMA_INFIFO_FULL_ERR_INT: triggered when GDMA RX FIFO length is shorter than the real transferred data length.
- SPI_DMA_OUTFIFO_EMPTY_ERR_INT: triggered when GDMA TX FIFO length is shorter than the real transferred data length.
- SPI_SLV_EX_QPI_INT: triggered when Ex_QPI is received correctly in GP-SPI2 slave mode and the SPI transfer ends.
- SPI_SLV_EN_QPI_INT: triggered when En_QPI is received correctly in GP-SPI2 slave mode and the SPI transfer ends.
- SPI_SLV_CMD7_INT: triggered when CMD7 is received correctly in GP-SPI2 slave mode and the SPI transfer ends.
- SPI_SLV_CMD8_INT: triggered when CMD8 is received correctly in GP-SPI2 slave mode and the SPI transfer ends.

- SPI_SLV_CMD9_INT: triggered when CMD9 is received correctly in GP-SPI2 slave mode and the SPI transfer ends.
- SPI_SLV_CMDA_INT: triggered when CMDA is received correctly in GP-SPI2 slave mode and the SPI transfer ends.
- SPI_SLV_RD_DMA_DONE_INT: triggered at the end of Rd_DMA transfer in slave mode.
- SPI_SLV_WR_DMA_DONE_INT: triggered at the end of Wr_DMA transfer in slave mode.
- SPI_SLV_RD_BUF_DONE_INT: triggered at the end of Rd_BUF transfer in slave mode.
- SPI_SLV_WR_BUF_DONE_INT: triggered at the end of Wr_BUF transfer in slave mode.
- SPI_TRANS_DONE_INT: triggered at the end of SPI bus transfer in both master and slave modes.
- SPI_DMA_SEG_TRANS_DONE_INT: triggered at the end of End_SEG_TRANS transfer in GP-SPI2 slave segmented transfer mode or at the end of configurable segmented transfer in master mode.
- SPI_SEG_MAGIC_ERR_INT: triggered when a Magic error occurs in CONF buffer during configurable segmented transfer in master mode.
- SPI_MST_RX_AFIFO_WFULL_ERR_INT: triggered by RX AFIFO write-full error in GP-SPI2 master mode.
- SPI_MST_TX_AFIFO_REMPTY_ERR_INT: triggered by TX AFIFO read-empty error in GP-SPI2 master mode.
- SPI_SLV_CMD_ERR_INT: triggered when a received command value is not supported in GP-SPI2 slave mode.
- SPI_APP2_INT: used and triggered by software. Only used for user defined function.
- SPI_APP1_INT: used and triggered by software. Only used for user defined function.

Interrupts Used in Master and Slave Modes

Table 20-18 and Table 20-19 show the interrupts used in GP-SPI2 master and slave modes. Set the interrupt enable bit SPI_*_INT_ENA in SPI_DMA_INT_ENA_REG and wait for the SPI_INT interrupt. When the transfer ends, the related interrupt is triggered and should be cleared by software before the next transfer.

Table 20-18. GP-SPI2 Master Mode Interrupts

Transfer Type	Communication Mode	Controlled by	Interrupt
Single Transfer	Full-duplex	DMA	GDMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	Half-duplex MOSI Mode	DMA	SPI_TRANS_DONE_INT
		CPU	SPI_TRANS_DONE_INT
	Half-duplex MISO Mode	DMA	GDMA_IN_SUC_EOF_CH*n*_INT
		CPU	SPI_TRANS_DONE_INT
Configurable Segmented Transfer	Full-duplex	DMA	SPI_DMA_SEG_TRANS_DONE_INT [3]
		CPU	Not supported
	Half-duplex MOSI Mode	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	Not supported
	Half-duplex MISO	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	Not supported

Continued on the next page

Table 20-18 – Continued from the previous page

Transfer Type	Communication Mode	Controlled by	Interrupt

[1] If GDMA_IN_SUC_EOF_CH*n*_INT is triggered, it means all the RX data of GP-SPI2 has been stored in the RX buffer, and the TX data has been transferred to the slave.

[2] SPI_TRANS_DONE_INT is triggered when CS is high, which indicates that master has completed the data exchange in SPI_W0_REG ~ SPI_W15_REG with slave in this mode.

[3] If SPI_DMA_SEG_TRANS_DONE_INT is triggered, it means that the whole configurable segmented transfer (consisting of several segments) has finished, i.e. the RX data has been stored in the RX buffer completely and all the TX data has been sent out.

Table 20-19. GP-SPI2 Slave Mode Interrupts

Transfer Type	Communication Mode	Controlled by	Interrupt
Single Transfer	Full-duplex	DMA	GDMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	Half-duplex MOSI Mode	DMA (Wr_DMA)	GDMA_IN_SUC_EOF_CH*n*_INT[3]
		CPU (Wr_BUF)	SPI_TRANS_DONE_INT[4]
	Half-duplex MISO Mode	DMA (Rd_DMA)	SPI_TRANS_DONE_INT[5]
		CPU (Rd_BUF)	SPI_TRANS_DONE_INT[6]
Slave Segmented Transfer	Full-duplex	DMA	GDMA_IN_SUC_EOF_CH*n*_INT[7]
		CPU	Not supported[8]
	Half-duplex MOSI Mode	DMA (Wr_DMA)	SPI_DMA_SEG_TRANS_DONE_INT[9]
		CPU (Wr_BUF)	Not supported[10]
	Half-duplex MISO Mode	DMA (Rd_DMA)	SPI_DMA_SEG_TRANS_DONE_INT[11]
		CPU (Rd_BUF)	Not supported[12]

[1] If GDMA_IN_SUC_EOF_CH*n*_INT is triggered, it means all the RX data has been stored in the RX buffer, and the TX data has been sent to the slave.

[2] SPI_TRANS_DONE_INT is triggered when CS is high, which indicates that master has completed the data exchange in SPI_W0_REG ~ SPI_W15_REG with slave in this mode.

[3] SPI_SLV_WR_DMA_DONE_INT just means that the transmission on the SPI bus is done, but can not ensure that all the push data has been stored in the RX buffer. For this reason, GDMA_IN_SUC_EOF_CH*n*_INT is recommended.

[4] Or wait for SPI_SLV_WR_BUF_DONE_INT.

[5] Or wait for SPI_SLV_RD_DMA_DONE_INT.

[6] Or wait for SPI_SLV_RD_BUF_DONE_INT.

[7] Slave should set the total read data byte length in SPI_MS_DATA_BITLEN before the transfer begins. Set SPI_RX_EOF_EN to 1 before the end of the interrupt program.

[8] Master and slave should define a method to end the segmented transfer, such as via GPIO interrupt.

[9] Master sends End_SEG_TRAN to end the segmented transfer or slave sets the total read data byte length in SPI_MS_DATA_BITLEN and waits for GDMA_IN_SUC_EOF_CH*n*_INT.

[10] Half-duplex Wr_BUF single transfer can be used in a slave segmented transfer.

[11] Master sends End_SEG_TRAN to end the segmented transfer.

[12] Half-duplex Rd_BUF single transfer can be used in a slave segmented transfer.

20.10 Register Summary

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

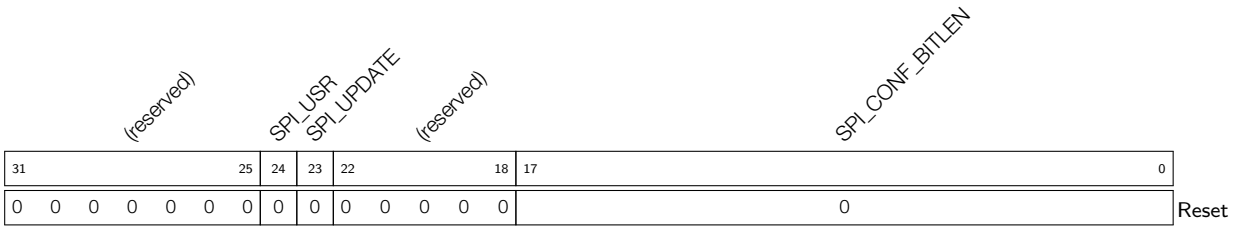
Name	Description	Address	Access
User-defined control registers			
SPI_CMD_REG	Command control register	0x0000	varies
SPI_ADDR_REG	Address value register	0x0004	R/W
SPI_USER_REG	SPI USER control register	0x0010	varies
SPI_USER1_REG	SPI USER control register 1	0x0014	R/W
SPI_USER2_REG	SPI USER control register 2	0x0018	R/W
Control and configuration registers			
SPI_CTRL_REG	SPI control register	0x0008	R/W
SPI_MS_DLEN_REG	SPI data bit length control register	0x001C	R/W
SPI_MISC_REG	SPI MISC register	0x0020	R/W
SPI_DMA_CONF_REG	SPI DMA control register	0x0030	varies
SPI_SLAVE_REG	SPI slave control register	0x00E0	varies
SPI_SLAVE1_REG	SPI slave control register 1	0x00E4	R/W/SS
Clock control registers			
SPI_CLOCK_REG	SPI clock control register	0x000C	R/W
SPI_CLK_GATE_REG	SPI module clock and register clock control	0x00E8	R/W
Interrupt registers			
SPI_DMA_INT_ENA_REG	SPI DMA interrupt enable register	0x0034	R/W
SPI_DMA_INT_CLR_REG	SPI DMA interrupt clear register	0x0038	WT
SPI_DMA_INT_RAW_REG	SPI DMA interrupt raw register	0x003C	varies
SPI_DMA_INT_ST_REG	SPI DMA interrupt status register	0x0040	RO
SPI_DMA_INT_SET_REG	SPI DMA interrupt software set register	0x0044	RO
CPU-controlled data buffer			
SPI_W0_REG	SPI CPU-controlled buffer 0	0x0098	R/W/SS
SPI_W1_REG	SPI CPU-controlled buffer 1	0x009C	R/W/SS
SPI_W2_REG	SPI CPU-controlled buffer 2	0x00A0	R/W/SS
SPI_W3_REG	SPI CPU-controlled buffer 3	0x00A4	R/W/SS
SPI_W4_REG	SPI CPU-controlled buffer 4	0x00A8	R/W/SS
SPI_W5_REG	SPI CPU-controlled buffer 5	0x00AC	R/W/SS
SPI_W6_REG	SPI CPU-controlled buffer 6	0x00B0	R/W/SS
SPI_W7_REG	SPI CPU-controlled buffer 7	0x00B4	R/W/SS
SPI_W8_REG	SPI CPU-controlled buffer 8	0x00B8	R/W/SS
SPI_W9_REG	SPI CPU-controlled buffer 9	0x00BC	R/W/SS
SPI_W10_REG	SPI CPU-controlled buffer 10	0x00C0	R/W/SS
SPI_W11_REG	SPI CPU-controlled buffer 11	0x00C4	R/W/SS
SPI_W12_REG	SPI CPU-controlled buffer 12	0x00C8	R/W/SS
SPI_W13_REG	SPI CPU-controlled buffer 13	0x00CC	R/W/SS
SPI_W14_REG	SPI CPU-controlled buffer 14	0x00D0	R/W/SS
SPI_W15_REG	SPI CPU-controlled buffer 15	0x00D4	R/W/SS

Name	Description	Address	Access
Version register			
SPI_DATE_REG	Version control register	0x00F0	R/W

20.11 Registers

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

Register 20.1. SPI_CMD_REG (0x0000)

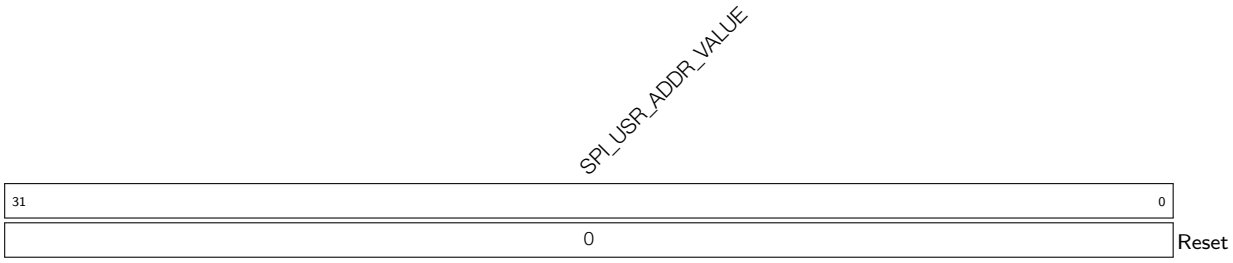


SPI_CONF_BITLEN Define the SPI CLK cycles of SPI CONF state. Can be configured in CONF state. (R/W)

SPI_UPDATE Set this bit to synchronize SPI registers from APB clock domain into SPI module clock domain. This bit is only used in SPI master mode. (WT)

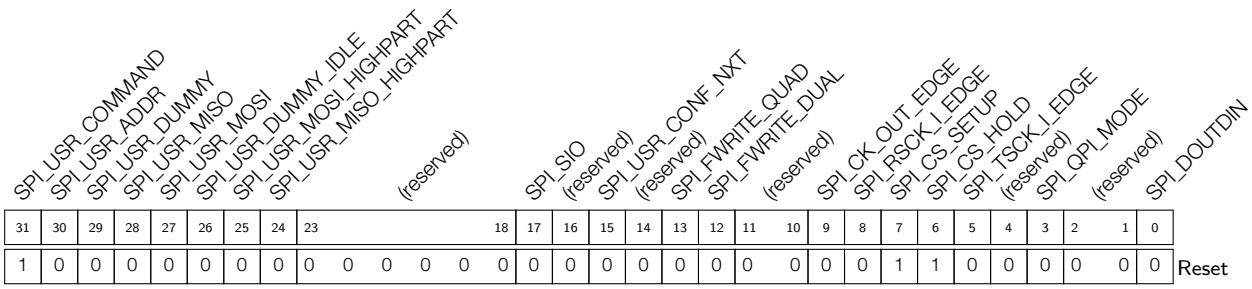
SPI_USR User-defined command enable. An SPI operation will be triggered when the bit is set. The bit will be cleared once the operation done. 1: enable; 0: disable. Can not be changed by CONF_buf. (R/W/SC)

Register 20.2. SPI_ADDR_REG (0x0004)



SPI_USR_ADDR_VALUE Address to slave. Can be configured in CONF state. (R/W)

Register 20.3. SPI_USER_REG (0x0010)



SPI_DOUTDIN Set the bit to enable full duplex communication. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_QPI_MODE 1: Enable QPI mode. 0: Disable QPI mode. This configuration is applicable when the SPI controller works as master or slave. Can be configured in CONF state. (R/W/SS/SC)

SPI_TSCK_I_EDGE In slave mode, this bit can be used to change the polarity of TSCK. 0: TSCK = SPI_CK_I. 1: TSCK = !SPI_CK_I. (R/W)

SPI_CS_HOLD Keep SPI CS low when SPI is in DONE state. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_CS_SETUP Enable SPI CS when SPI is in prepare (PREP) state. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_RSCK_I_EDGE In slave mode, this bit can be used to change the polarity of RSCK. 0: RSCK = !SPI_CK_I. 1: RSCK = SPI_CK_I. (R/W)

SPI_CK_OUT_EDGE This bit together with SPI_CK_IDLE_EDGE is used to control SPI clock mode. Can be configured in CONF state. For more information, see Section 20.7.2. (R/W)

SPI_FWRITE_DUAL In write operations, read-data phase is in 2-bit mode. Can be configured in CONF state. (R/W)

SPI_FWRITE_QUAD In write operations, read-data phase is in 4-bit mode. Can be configured in CONF state. (R/W)

SPI_USR_CONF_NXT Enable the CONF state for the next transaction (segment) in a configurable segmented transfer. Can be configured in CONF state. (R/W)

- If this bit is set, it means this configurable segmented transfer will continue its next transaction (segment).
- If this bit is cleared, it means this transfer will end after the current transaction (segment) is finished. Or this is not a configurable segmented transfer.

SPI_SIO Set the bit to enable 3-line half-duplex communication, where MOSI and MISO signals share the same pin. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_USR_MISO_HIGHPART In read-data phase, only access to high-part of the buffers: SPI_W8_REG ~ SPI_W15_REG. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

Continued on the next page...

Register 20.3. SPI_USER_REG (0x0010)

Continued from the previous page...

SPI_USR_MOSI_HIGHPART In write-data phase, only access to high-part of the buffers: SPI_W8_REG ~ SPI_W15_REG. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_USR_DUMMY_IDLE If this bit is set, SPI clock is disabled in DUMMY state. Can be configured in CONF state. (R/W)

SPI_USR_MOSI Set this bit to enable the write-data (DOUT) state of an operation. Can be configured in CONF state. (R/W)

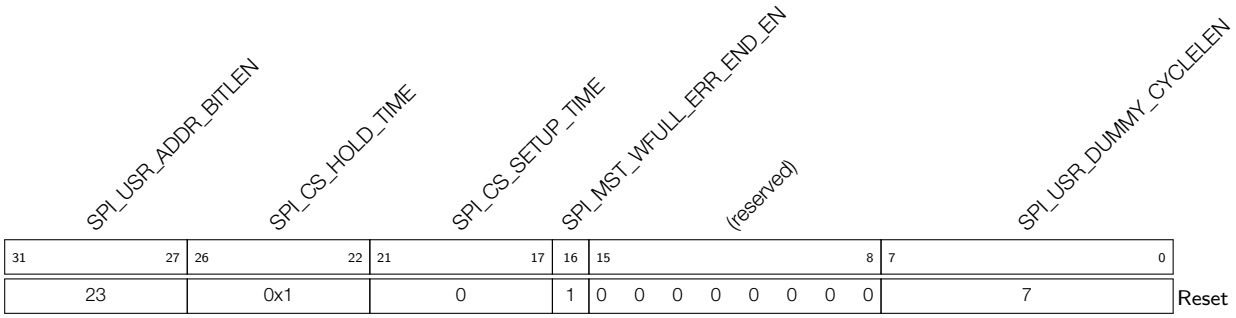
SPI_USR_MISO Set this bit to enable the read-data (DIN) state of an operation. Can be configured in CONF state. (R/W)

SPI_USR_DUMMY Set this bit to enable the DUMMY state of an operation. Can be configured in CONF state. (R/W)

SPI_USR_ADDR Set this bit to enable the address (ADDR) state of an operation. Can be configured in CONF state. (R/W)

SPI_USR_COMMAND Set this bit to enable the command (CMD) state of an operation. Can be configured in CONF state. (R/W)

Register 20.4. SPI_USER1_REG (0x0014)



SPI_USR_DUMMY_CYCLELEN The length of DUMMY state, in unit of SPI_CLK cycles. This value is (the expected cycle number - 1). Can be configured in CONF state. (R/W)

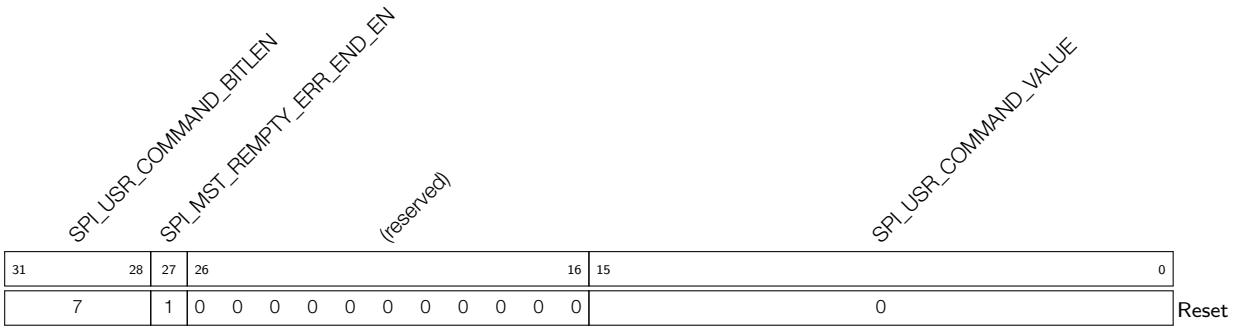
SPI_MST_WFULL_ERR_END_EN 1: SPI transfer is ended when SPI RX AFIFO wfull error occurs in GP-SPI2 master full-/half-duplex modes. 0: SPI transfer is not ended when SPI RX AFIFO wfull error occurs in GP-SPI2 master full-/half-duplex modes. (R/W)

SPI_CS_SETUP_TIME The length of prepare (PREP) state, in unit of SPI_CLK cycles. This value is equal to the expected cycles -1. This field is used together with SPI_CS_SETUP. Can be configured in CONF state. (R/W)

SPI_CS_HOLD_TIME Delay cycles of CS pin, in units of SPI_CLK cycles. This field is used together with SPI_CS_HOLD. Can be configured in CONF state. (R/W)

SPI_USR_ADDR_BITLEN The bit length in address state. This value is (expected bit number - 1). Can be configured in CONF state. (R/W)

Register 20.5. SPI_USER2_REG (0x0018)

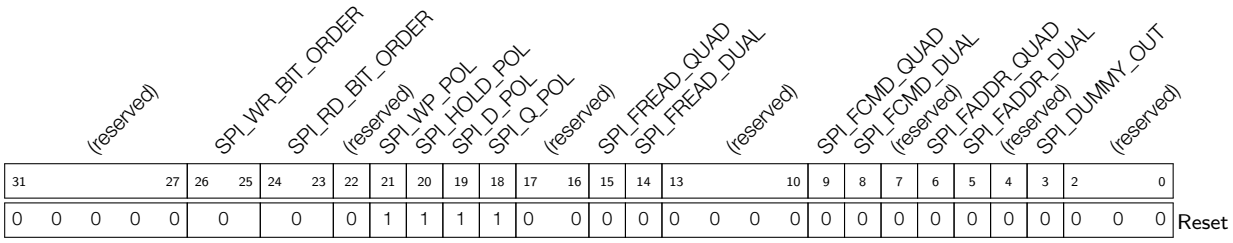


SPI_USR_COMMAND_VALUE The value of command. Can be configured in CONF state. (R/W)

SPI_MST_REMPTY_ERR_END_EN 1: SPI transfer is ended when SPI TX AFIFO read empty error occurs in GP-SPI2 master full-/half-duplex modes. 0: SPI transfer is not ended when SPI TX AFIFO read empty error occurs in GP-SPI2 master full-/half-duplex modes. (R/W)

SPI_USR_COMMAND_BITLEN The bit length of command state. This value is (expected bit number - 1). Can be configured in CONF state. (R/W)

Register 20.6. SPI_CTRL_REG (0x0008)



SPI_DUMMY_OUT 0: In the dummy phase, the FSPI bus signals are not output. 1: In the dummy phase, the FSPI bus signals are output. Can be configured in CONF state. (R/W)

SPI_FADDR_DUAL Apply 2-bit mode during address (ADDR) state 1: enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_FADDR_QUAD Apply 4-bit mode during address (ADDR) state. 1:enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_FCMD_DUAL Apply 2-bit mode during command (CMD) state. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_FCMD_QUAD Apply 4-bit mode during command (CMD) state. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_FREAD_DUAL In read operations, read-data (DIN) state is in 2-bit mode. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_FREAD_QUAD In read operations, read-data (DIN) state is in 4-bit mode. 1: enable; 0: disable. Can be configured in CONF state. (R/W)

SPI_Q_POL This bit is used to set MISO line polarity. 1: high; 0: low. Can be configured in CONF state. (R/W)

SPI_D_POL This bit is used to set MOSI line polarity. 1: high; 0: low. Can be configured in CONF state. (R/W)

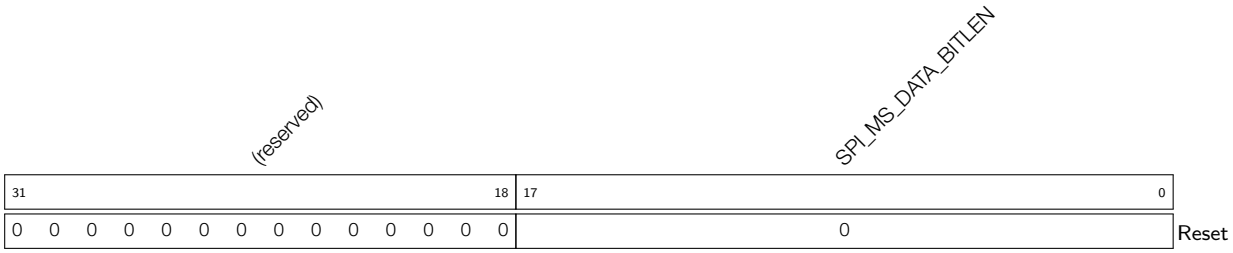
SPI_HOLD_POL This bit is used to set SPI_HOLD output value when SPI is in idle. 1: output high; 0: output low. Can be configured in CONF state. (R/W)

SPI_WP_POL This bit is to set the output value of write-protect signal when SPI is in idle. 1: output high; 0: output low. Can be configured in CONF state. (R/W)

SPI_RD_BIT_ORDER In read-data (MISO) state, 1: LSB first; 0: MSB first. Can be configured in CONF state. (R/W)

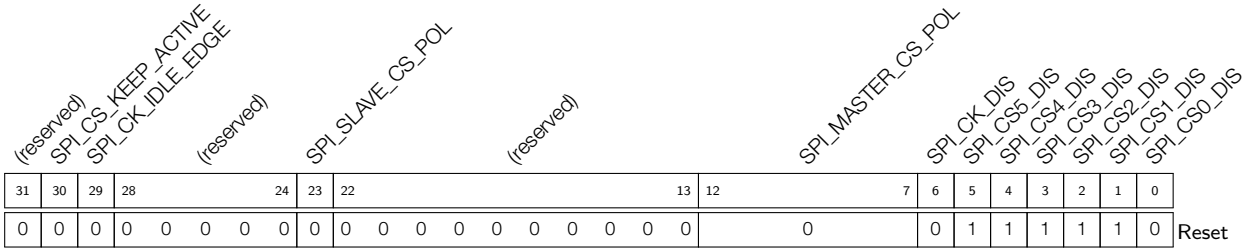
SPI_WR_BIT_ORDER In command (CMD), address (ADDR), and write-data (MOSI) states, 1: LSB first; 0: MSB first. Can be configured in CONF state. (R/W)

Register 20.7. SPI_MS_DLEN_REG (0x001C)



SPI_MS_DATA_BITLEN The value of this field is the configured SPI transmission data bit length in master mode DMA controlled transfer or CPU controlled transfer. The value is also the configured bit length in slave mode DMA RX controlled transfer. The register value shall be (bit_num - 1). Can be configured in CONF state. (R/W)

Register 20.8. SPI_MISC_REG (0x0020)



SPI_CS0_DIS SPI CS0 pin enable bit. 1: disable CS0, 0: SPI_CS0 signal is from/to CS0 pin. Can be configured in CONF state. (R/W)

SPI_CS1_DIS SPI CS1 pin enable bit. 1: disable CS1, 0: SPI_CS1 signal is from/to CS1 pin. Can be configured in CONF state. (R/W)

SPI_CS2_DIS SPI CS2 pin enable bit. 1: disable CS2, 0: SPI_CS2 signal is from/to CS2 pin. Can be configured in CONF state. (R/W)

SPI_CS3_DIS SPI CS3 pin enable bit. 1: disable CS3, 0: SPI_CS3 signal is from/to CS3 pin. Can be configured in CONF state. (R/W)

SPI_CS4_DIS SPI CS4 pin enable bit. 1: disable CS4, 0: SPI_CS4 signal is from/to CS4 pin. Can be configured in CONF state. (R/W)

SPI_CS5_DIS SPI CS5 pin enable bit. 1: disable CS5, 0: SPI_CS5 signal is from/to CS5 pin. Can be configured in CONF state. (R/W)

SPI_CK_DIS 1: disable SPI_CLK output. 0: enable SPI_CLK output. Can be configured in CONF state. (R/W)

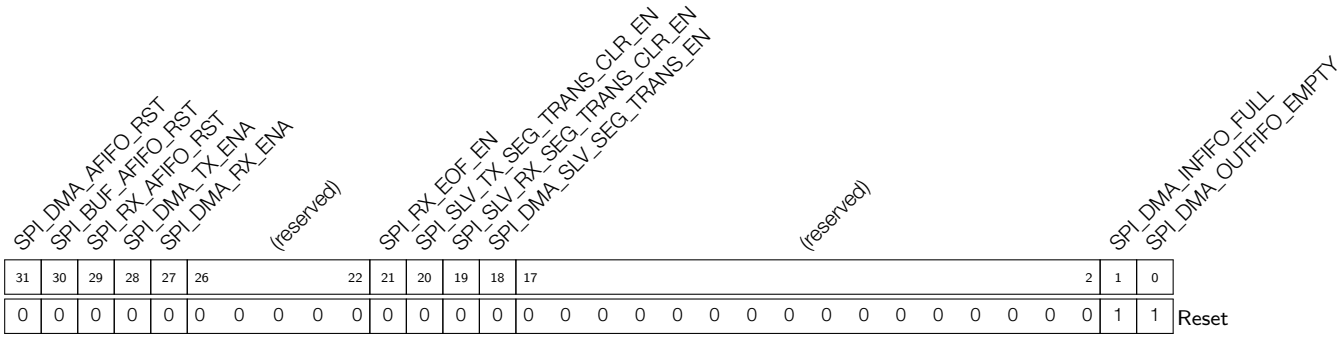
SPI_MASTER_CS_POL SPI_MASTER_CS_POL[*i*] configures the polarity of SPI CS*i* (*i* is from 0 ~ 5) line in master mode. 0: CS*i* is low active. 1: CS*i* is high active. Can be configured in CONF state. (R/W)

SPI_SLAVE_CS_POL Configure SPI slave input CS polarity. 1: invert. 0: not change. Can be configured in CONF state. (R/W)

SPI_CK_IDLE_EDGE 1: SPI_CLK line is high when GP-SPI2 is in idle. 0: SPI_CLK line is low when GP-SPI2 is in idle. Can be configured in CONF state. (R/W)

SPI_CS_KEEP_ACTIVE SPI CS line keeps low when the bit is set. Can be configured in CONF state. (R/W)

Register 20.9. SPI_DMA_CONF_REG (0x0030)



SPI_DMA_OUTFIFO_EMPTY Records the status of DMA TX FIFO. 1: DMA TX FIFO is not ready for sending data. 0: DMA TX FIFO is ready for sending data. (RO)

SPI_DMA_INFIFO_FULL Records the status of DMA RX FIFO. 1: DMA RX FIFO is not ready for receiving data. 0: DMA RX FIFO is ready for receiving data. (RO)

SPI_DMA_SLV_SEG_TRANS_EN 1: enable DMA-controlled segmented transfer in slave half-duplex mode. 0: disable. (R/W)

SPI_SLV_RX_SEG_TRANS_CLR_EN In slave segmented transfer, if the size of the DMA RX buffer is smaller than the size of the received data, 1: the data in all the following Wr_DMA transactions will not be received; 0: the data in this Wr_DMA transaction will not be received, but in the following transactions, (R/W)

- if the size of DMA RX buffer is not 0, the data in following Wr_DMA transactions will be received.
- if the size of DMA RX buffer is 0, the data in following Wr_DMA transactions will not be received.

SPI_SLV_TX_SEG_TRANS_CLR_EN In slave segmented transfer, if the size of the DMA TX buffer is smaller than the size of the transmitted data, (R/W)

- 1: the data in the following transactions will not be updated, i.e. the old data is transmitted repeatedly.
- 0: the data in this transaction will not be updated. But in the following transactions,
  - if new data is filled in DMA TX FIFO, new data will be transmitted.
  - if no new data is filled in DMA TX FIFO, no new data will be transmitted.

SPI_RX_EOF_EN 1: In a DAM-controlled transfer, if the bit number of transferred data is equal to (SPI_MS_DATA_BITLEN + 1), then GDMA_IN_SUC_EOF_CH*n*_INT_RAW will be set by hardware. 0: GDMA_IN_SUC_EOF_CH*n*_INT_RAW is set by SPI_TRANS_DONE_INT event in a single transfer, or by an SPI_DMA_SEG_TRANS_DONE_INT event in a segmented transfer. (R/W)

SPI_DMA_RX_ENA Set this bit to enable SPI DMA controlled receive data mode. (R/W)

SPI_DMA_TX_ENA Set this bit to enable SPI DMA controlled send data mode. (R/W)

Continued on the next page...

Register 20.9. SPI_DMA_CONF_REG (0x0030)

Continued from the previous page...

SPI_RX_AFIFO_RST Set this bit to reset spi_rx_afifo as shown in Figure 20-4 and in Figure 20-5. spi_rx_afifo is used to receive data in SPI master and slave transfer. (WT)

SPI_BUF_AFIFO_RST Set this bit to reset buf_tx_afifo as shown in Figure 20-4 and in Figure 20-5. buf_tx_afifo is used to send data out in CPU-controlled master and slave transfer. (WT)

SPI_DMA_AFIFO_RST Set this bit to reset dma_tx_afifo as shown in Figure 20-4 and in Figure 20-5. dma_tx_afifo is used to send data out in DMA-controlled slave transfer. (WT)

Register 20.10. SPI_SLAVE_REG (0x00E0)

(reserved)	SPI_USR_CONF	SPI_SOFT_RESET	SPI_SLAVE_MODE	SPI_DMA_SEG_MAGIC_VALUE	(reserved)	SPI_SLV_WRBUF_BITLEN_EN	SPI_SLV_RDBUF_BITLEN_EN	SPI_SLV_WRDMA_BITLEN_EN	SPI_SLV_RDDMA_BITLEN_EN	(reserved)	SPI_RSCK_DATA_OUT	SPI_CLK_MODE_13	SPI_CLK_MODE	
31 29	28	27	26	25 22	21 12	11	10	9	8	7 4	3	2	1 0	
0 0 0	0	0	0	10	0 0 0 0 0 0 0 0 0 0	0	0	0	0	0 0 0 0	0	0	0	Reset

SPI_CLK_MODE SPI clock mode control bits. Can be configured in CONF state. (R/W)

- 0: SPI clock is off when CS becomes inactive.
- 1: SPI clock is delayed one cycle after CS becomes inactive.
- 2: SPI clock is delayed two cycles after CS becomes inactive.
- 3: SPI clock is always on.

SPI_CLK_MODE_13 Configure clock mode. (R/W)

- 1: support SPI clock mode 1 or 3. See Table 20-17.
- 0: support SPI clock mode 0 or 2. See Table 20-17.

SPI_RSCK_DATA_OUT Save half a cycle when TSCK is the same as RSCK. 1: output data at RSCK rising edge. 0: output data at TSCK rising edge. (R/W)

SPI_SLV_RDDMA_BITLEN_EN If this bit is set, SPI_SLV_DATA_BITLEN is used to store the data bit length of Rd_DMA transfer (R/W)

SPI_SLV_WRDMA_BITLEN_EN If this bit is set, SPI_SLV_DATA_BITLEN is used to store the data bit length of Wr_DMA transfer. (R/W)

SPI_SLV_RDBUF_BITLEN_EN If this bit is set, SPI_SLV_DATA_BITLEN is used to store data bit length of Rd_BUF transfer. (R/W)

SPI_SLV_WRBUF_BITLEN_EN If this bit is set, SPI_SLV_DATA_BITLEN is used to store data bit length of Wr_BUF transfer. (R/W)

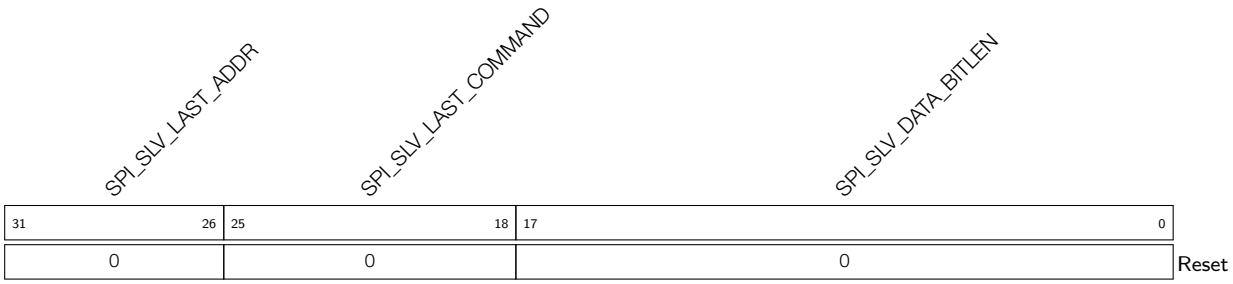
SPI_DMA_SEG_MAGIC_VALUE Configure the magic value of BM table in DMA-controlled configurable segmented transfer. (R/W)

SPI_SLAVE_MODE Set SPI work mode. 1: slave mode. 0: master mode. (R/W)

SPI_SOFT_RESET Software reset enable bit. If this bit is set, the SPI clock line, CS line, and data line are reset. Can be configured in CONF state. (WT)

SPI_USR_CONF 1: enable the CONF state of current DMA-controlled configurable segmented transfer, which means a configurable segmented transfer is started. 0: the current transfer is not a configurable segmented transfer. (R/W)

Register 20.11. SPI_SLAVE1_REG (0x00E4)

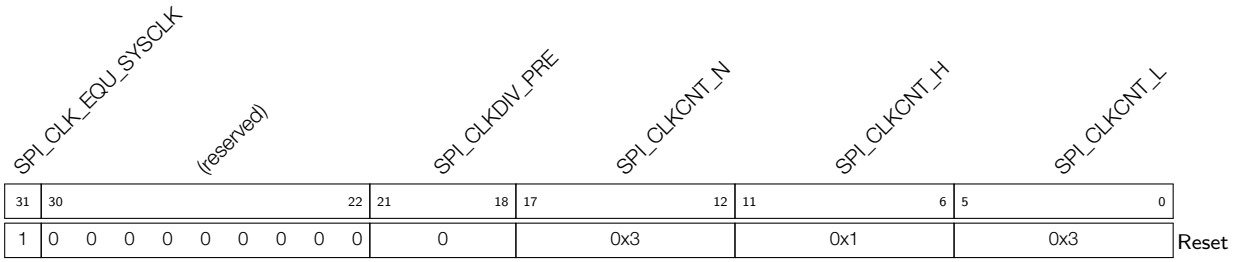


SPI_SLV_DATA_BITLEN Configure the transferred data bit length in SPI slave full-/half-duplex modes. (R/W/SS)

SPI_SLV_LAST_COMMAND In slave mode, it is the value of command. (R/W/SS)

SPI_SLV_LAST_ADDR In slave mode, it is the value of address. (R/W/SS)

Register 20.12. SPI_CLOCK_REG (0x000C)



SPI_CLKCNT_L In master mode, this field must be equal to SPI_CLKCNT_N. In slave mode, it must be 0. Can be configured in CONF state. (R/W)

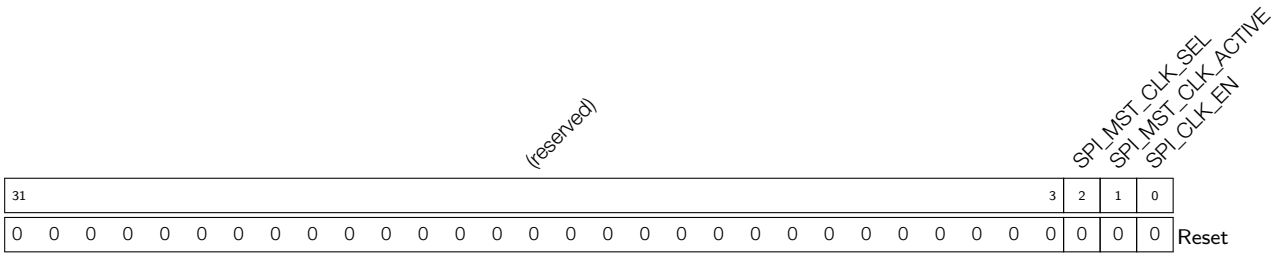
SPI_CLKCNT_H In master mode, this field is used to configure the duty cycle of SPI_CLK (high level). It's recommended to configure this value to floor((SPI_CLKCNT_N + 1)/2 - 1). floor() here is to round a number down, e.g., floor(2.2) = 2. In slave mode, it must be 0. Can be configured in CONF state. (R/W)

SPI_CLKCNT_N In master mode, this is the divider of SPI_CLK. So SPI_CLK frequency is f_{apb_clk}/(SPI_CLKDIV_PRE + 1)/(SPI_CLKCNT_N + 1). Can be configured in CONF state. (R/W)

SPI_CLKDIV_PRE In master mode, this is the pre-divider of SPI_CLK. Can be configured in CONF state. (R/W)

SPI_CLK_EQU_SYSCLK In master mode, 1: SPI_CLK is equal to APB_CLK. 0: SPI_CLK is divided from APB_CLK. Can be configured in CONF state. (R/W)

Register 20.13. SPI_CLK_GATE_REG (0x00E8)



SPI_CLK_EN Set this bit to enable clock gate. (R/W)

SPI_MST_CLK_ACTIVE Set this bit to power on the SPI module clock. (R/W)

SPI_MST_CLK_SEL This bit is used to select SPI module clock source in master mode. 1: PLL_F80M_CLK. 0: XTAL_CLK. (R/W)

Register 20.14. SPI_DMA_INT_ENA_REG (0x0034)

Bits	Field	Reset
31:21	(reserved)	0 0 0 0 0 0 0 0 0 0 0
20	SPI_APP1_INT_ENA	0
19	SPI_APP2_INT_ENA	0
18	SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA	0
17	SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA	0
16	SPI_SLV_CMD_ERR_INT_ENA	0
15	(reserved)	0
14	SPI_SEG_MAGIC_ERR_INT_ENA	0
13	SPI_DMA_SEG_TRANS_DONE_INT_ENA	0
12	SPI_TRANS_DONE_INT_ENA	0
11	SPI_SLV_WR_BUF_DONE_INT_ENA	0
10	SPI_SLV_RD_BUF_DONE_INT_ENA	0
9	SPI_SLV_WR_DMA_DONE_INT_ENA	0
8	SPI_SLV_RD_DMA_DONE_INT_ENA	0
7	SPI_SLV_CMDA_INT_ENA	0
6	SPI_SLV_CMD9_INT_ENA	0
5	SPI_SLV_CMD8_INT_ENA	0
4	SPI_SLV_CMD7_INT_ENA	0
3	SPI_SLV_EN_QPI_INT_ENA	0
2	SPI_SLV_EX_QPI_INT_ENA	0
1	SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA	0
0	SPI_DMA_INFIFO_FULL_ERR_INT_ENA	0

SPI_DMA_INFIFO_FULL_ERR_INT_ENA The enable bit for SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (R/W)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA The enable bit for SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (R/W)

SPI_SLV_EX_QPI_INT_ENA The enable bit for SPI_SLV_EX_QPI_INT interrupt. (R/W)

SPI_SLV_EN_QPI_INT_ENA The enable bit for SPI_SLV_EN_QPI_INT interrupt. (R/W)

SPI_SLV_CMD7_INT_ENA The enable bit for SPI_SLV_CMD7_INT interrupt. (R/W)

SPI_SLV_CMD8_INT_ENA The enable bit for SPI_SLV_CMD8_INT interrupt. (R/W)

SPI_SLV_CMD9_INT_ENA The enable bit for SPI_SLV_CMD9_INT interrupt. (R/W)

SPI_SLV_CMDA_INT_ENA The enable bit for SPI_SLV_CMDA_INT interrupt. (R/W)

SPI_SLV_RD_DMA_DONE_INT_ENA The enable bit for SPI_SLV_RD_DMA_DONE_INT interrupt. (R/W)

SPI_SLV_WR_DMA_DONE_INT_ENA The enable bit for SPI_SLV_WR_DMA_DONE_INT interrupt. (R/W)

SPI_SLV_RD_BUF_DONE_INT_ENA The enable bit for SPI_SLV_RD_BUF_DONE_INT interrupt. (R/W)

SPI_SLV_WR_BUF_DONE_INT_ENA The enable bit for SPI_SLV_WR_BUF_DONE_INT interrupt. (R/W)

SPI_TRANS_DONE_INT_ENA The enable bit for SPI_TRANS_DONE_INT interrupt. (R/W)

SPI_DMA_SEG_TRANS_DONE_INT_ENA The enable bit for SPI_DMA_SEG_TRANS_DONE_INT interrupt. (R/W)

SPI_SEG_MAGIC_ERR_INT_ENA The enable bit for SPI_SEG_MAGIC_ERR_INT interrupt. (R/W)

Continued on the next page...

Register 20.14. SPI_DMA_INT_ENA_REG (0x0034)

Continued from the previous page...

SPI_SLV_CMD_ERR_INT_ENA The enable bit for SPI_SLV_CMD_ERR_INT interrupt. (R/W)

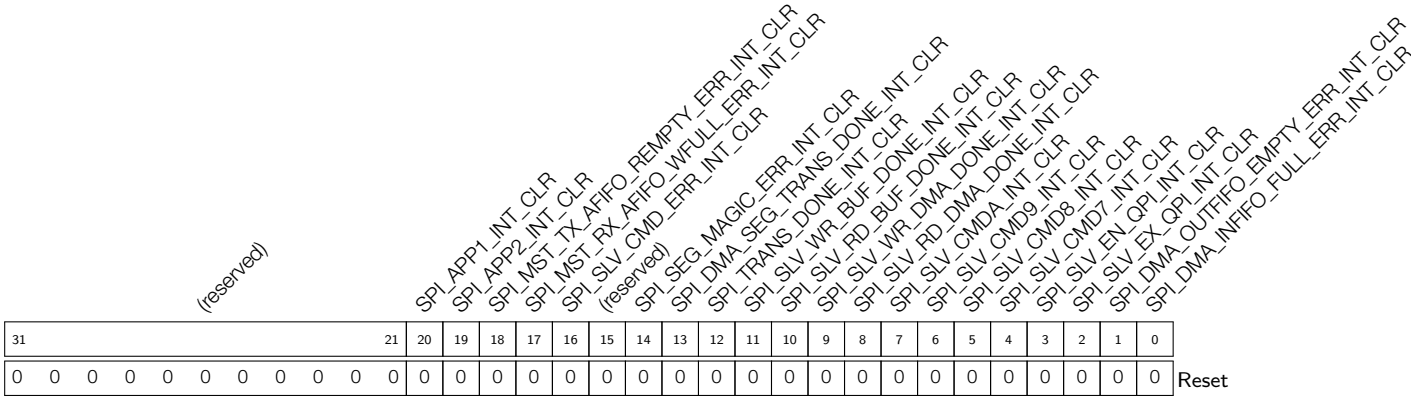
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA The enable bit for SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (R/W)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA The enable bit for SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (R/W)

SPI_APP2_INT_ENA The enable bit for SPI_APP2_INT interrupt. (R/W)

SPI_APP1_INT_ENA The enable bit for SPI_APP1_INT interrupt. (R/W)

Register 20.15. SPI_DMA_INT_CLR_REG (0x0038)



SPI_DMA_INFIFO_FULL_ERR_INT_CLR The clear bit for SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR The clear bit for SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (WT)

SPI_SLV_EX_QPI_INT_CLR The clear bit for SPI_SLV_EX_QPI_INT interrupt. (WT)

SPI_SLV_EN_QPI_INT_CLR The clear bit for SPI_SLV_EN_QPI_INT interrupt. (WT)

SPI_SLV_CMD7_INT_CLR The clear bit for SPI_SLV_CMD7_INT interrupt. (WT)

SPI_SLV_CMD8_INT_CLR The clear bit for SPI_SLV_CMD8_INT interrupt. (WT)

SPI_SLV_CMD9_INT_CLR The clear bit for SPI_SLV_CMD9_INT interrupt. (WT)

SPI_SLV_CMDA_INT_CLR The clear bit for SPI_SLV_CMDA_INT interrupt. (WT)

SPI_SLV_RD_DMA_DONE_INT_CLR The clear bit for SPI_SLV_RD_DMA_DONE_INT interrupt. (WT)

SPI_SLV_WR_DMA_DONE_INT_CLR The clear bit for SPI_SLV_WR_DMA_DONE_INT interrupt. (WT)

SPI_SLV_RD_BUF_DONE_INT_CLR The clear bit for SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

SPI_SLV_WR_BUF_DONE_INT_CLR The clear bit for SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

SPI_TRANS_DONE_INT_CLR The clear bit for SPI_TRANS_DONE_INT interrupt. (WT)

SPI_DMA_SEG_TRANS_DONE_INT_CLR The clear bit for SPI_DMA_SEG_TRANS_DONE_INT interrupt. (WT)

SPI_SEG_MAGIC_ERR_INT_CLR The clear bit for SPI_SEG_MAGIC_ERR_INT interrupt. (WT)

Continued on the next page...

Register 20.15. SPI_DMA_INT_CLR_REG (0x0038)

Continued from the previous page...

SPI_SLV_CMD_ERR_INT_CLR The clear bit for SPI_SLV_CMD_ERR_INT interrupt. (WT)

SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR The clear bit for SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR The clear bit for SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

SPI_APP2_INT_CLR The clear bit for SPI_APP2_INT interrupt. (WT)

SPI_APP1_INT_CLR The clear bit for SPI_APP1_INT interrupt. (WT)

Register 20.16. SPI_DMA_INT_RAW_REG (0x003C)

Bits	Field	Reset
31:21	(reserved)	0 0 0 0 0 0 0 0 0 0 0
20	SPI_APP1_INT_RAW	0
19	SPI_APP2_INT_RAW	0
18	SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW	0
17	SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW	0
16	SPI_SLV_CMD_ERR_INT_RAW	0
15	(reserved)	0
14	SPI_SEG_MAGIC_ERR_INT_RAW	0
13	SPI_DMA_SEG_TRANS_DONE_INT_RAW	0
12	SPI_TRANS_DONE_INT_RAW	0
11	SPI_SLV_WR_BUF_DONE_INT_RAW	0
10	SPI_SLV_RD_BUF_DONE_INT_RAW	0
9	SPI_SLV_WR_DMA_DONE_INT_RAW	0
8	SPI_SLV_RD_DMA_DONE_INT_RAW	0
7	SPI_SLV_CMDA_INT_RAW	0
6	SPI_SLV_CMD9_INT_RAW	0
5	SPI_SLV_CMD8_INT_RAW	0
4	SPI_SLV_CMD7_INT_RAW	0
3	SPI_SLV_EN_QPI_INT_RAW	0
2	SPI_SLV_EX_QPI_INT_RAW	0
1	SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW	0
0	SPI_DMA_INFIFO_FULL_ERR_INT_RAW	0

SPI_DMA_INFIFO_FULL_ERR_INT_RAW The raw bit for SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (R/W/WTC/SS)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW The raw bit for SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (R/W/WTC/SS)

SPI_SLV_EX_QPI_INT_RAW The raw bit for SPI_SLV_EX_QPI_INT interrupt. (R/W/WTC/SS)

SPI_SLV_EN_QPI_INT_RAW The raw bit for SPI_SLV_EN_QPI_INT interrupt. (R/W/WTC/SS)

SPI_SLV_CMD7_INT_RAW The raw bit for SPI_SLV_CMD7_INT interrupt. (R/W/WTC/SS)

SPI_SLV_CMD8_INT_RAW The raw bit for SPI_SLV_CMD8_INT interrupt. (R/W/WTC/SS)

SPI_SLV_CMD9_INT_RAW The raw bit for SPI_SLV_CMD9_INT interrupt. (R/W/WTC/SS)

SPI_SLV_CMDA_INT_RAW The raw bit for SPI_SLV_CMDA_INT interrupt. (R/W/WTC/SS)

SPI_SLV_RD_DMA_DONE_INT_RAW The raw bit for SPI_SLV_RD_DMA_DONE_INT interrupt. (R/W/WTC/SS)

SPI_SLV_WR_DMA_DONE_INT_RAW The raw bit for SPI_SLV_WR_DMA_DONE_INT interrupt. (R/W/WTC/SS)

SPI_SLV_RD_BUF_DONE_INT_RAW The raw bit for SPI_SLV_RD_BUF_DONE_INT interrupt. (R/W/WTC/SS)

SPI_SLV_WR_BUF_DONE_INT_RAW The raw bit for SPI_SLV_WR_BUF_DONE_INT interrupt. (R/W/WTC/SS)

SPI_TRANS_DONE_INT_RAW The raw bit for SPI_TRANS_DONE_INT interrupt. (R/W/WTC/SS)

Continued on the next page...

Register 20.16. SPI_DMA_INT_RAW_REG (0x003C)

Continued from the previous page...

SPI_DMA_SEG_TRANS_DONE_INT_RAW The raw bit for SPI_DMA_SEG_TRANS_DONE_INT interrupt. (R/W/WTC/SS)

SPI_SEG_MAGIC_ERR_INT_RAW The raw bit for SPI_SEG_MAGIC_ERR_INT interrupt. (R/W/WTC/SS)

SPI_SLV_CMD_ERR_INT_RAW The raw bit for SPI_SLV_CMD_ERR_INT interrupt. (R/W/WTC/SS)

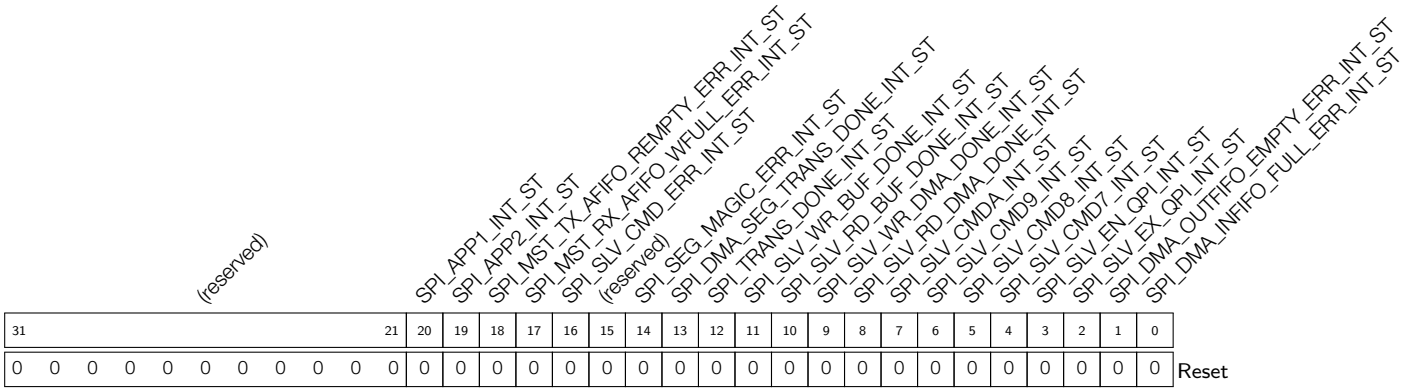
SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW The raw bit for SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (R/W/WTC/SS)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW The raw bit for SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (R/W/WTC/SS)

SPI_APP2_INT_RAW The raw bit for SPI_APP2_INT interrupt. The value is only controlled by the application. (R/W/WTC)

SPI_APP1_INT_RAW The raw bit for SPI_APP1_INT interrupt. The value is only controlled by the application. (R/W/WTC)

Register 20.17. SPI_DMA_INT_ST_REG (0x0040)



SPI_DMA_INFIFO_FULL_ERR_INT_ST The status bit for SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (RO)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ST The status bit for SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (RO)

SPI_SLV_EX_QPI_INT_ST The status bit for SPI_SLV_EX_QPI_INT interrupt. (RO)

SPI_SLV_EN_QPI_INT_ST The status bit for SPI_SLV_EN_QPI_INT interrupt. (RO)

SPI_SLV_CMD7_INT_ST The status bit for SPI_SLV_CMD7_INT interrupt. (RO)

SPI_SLV_CMD8_INT_ST The status bit for SPI_SLV_CMD8_INT interrupt. (RO)

SPI_SLV_CMD9_INT_ST The status bit for SPI_SLV_CMD9_INT interrupt. (RO)

SPI_SLV_CMDA_INT_ST The status bit for SPI_SLV_CMDA_INT interrupt. (RO)

SPI_SLV_RD_DMA_DONE_INT_ST The status bit for SPI_SLV_RD_DMA_DONE_INT interrupt. (RO)

SPI_SLV_WR_DMA_DONE_INT_ST The status bit for SPI_SLV_WR_DMA_DONE_INT interrupt. (RO)

SPI_SLV_RD_BUF_DONE_INT_ST The status bit for SPI_SLV_RD_BUF_DONE_INT interrupt. (RO)

SPI_SLV_WR_BUF_DONE_INT_ST The status bit for SPI_SLV_WR_BUF_DONE_INT interrupt. (RO)

SPI_TRANS_DONE_INT_ST The status bit for SPI_TRANS_DONE_INT interrupt. (RO)

SPI_DMA_SEG_TRANS_DONE_INT_ST The status bit for SPI_DMA_SEG_TRANS_DONE_INT interrupt. (RO)

SPI_SEG_MAGIC_ERR_INT_ST The status bit for SPI_SEG_MAGIC_ERR_INT interrupt. (RO)

SPI_SLV_CMD_ERR_INT_ST The status bit for SPI_SLV_CMD_ERR_INT interrupt. (RO)

Continued on the next page...

Register 20.17. SPI_DMA_INT_ST_REG (0x0040)

Continued from the previous page...

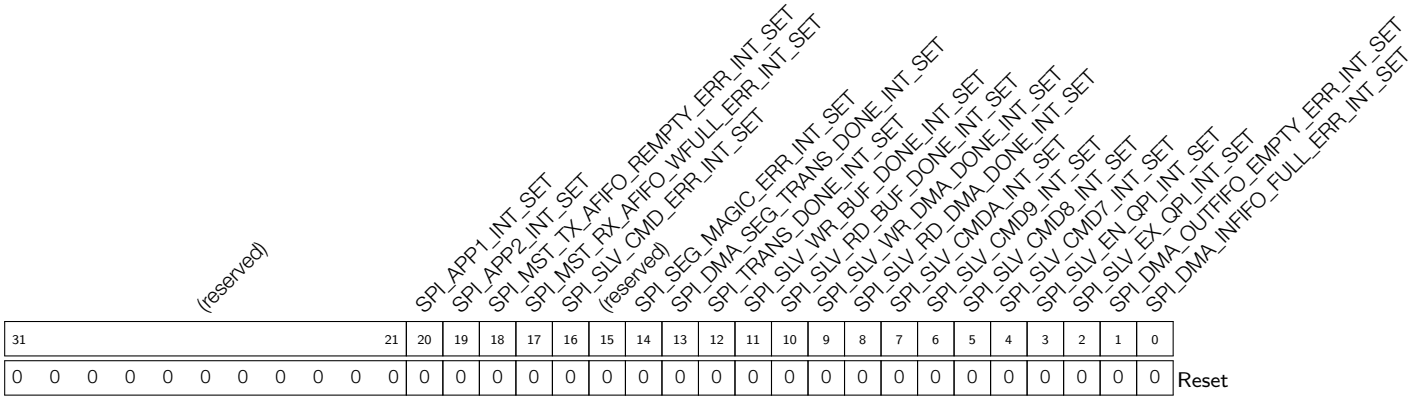
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST The status bit for SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (RO)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST The status bit for SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (RO)

SPI_APP2_INT_ST The status bit for SPI_APP2_INT interrupt. (RO)

SPI_APP1_INT_ST The status bit for SPI_APP1_INT interrupt. (RO)

Register 20.18. SPI_DMA_INT_SET_REG (0x0044)



SPI_DMA_INFIFO_FULL_ERR_INT_SET The software set bit for SPI_DMA_INFIFO_FULL_ERR_INT interrupt. (WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET The software set bit for SPI_DMA_OUTFIFO_EMPTY_ERR_INT interrupt. (WT)

SPI_SLV_EX_QPI_INT_SET The software set bit for SPI_SLV_EX_QPI_INT interrupt. (WT)

SPI_SLV_EN_QPI_INT_SET The software set bit for SPI_SLV_EN_QPI_INT interrupt. (WT)

SPI_SLV_CMD7_INT_SET The software set bit for SPI_SLV_CMD7_INT interrupt. (WT)

SPI_SLV_CMD8_INT_SET The software set bit for SPI_SLV_CMD8_INT interrupt. (WT)

SPI_SLV_CMD9_INT_SET The software set bit for SPI_SLV_CMD9_INT interrupt. (WT)

SPI_SLV_CMDA_INT_SET The software set bit for SPI_SLV_CMDA_INT interrupt. (WT)

Continued on the next page...

Register 20.18. SPI_DMA_INT_SET_REG (0x0044)

Continued from the previous page...

SPI_SLV_RD_DMA_DONE_INT_SET The software set bit for SPI_SLV_RD_DMA_DONE_INT interrupt. (WT)

SPI_SLV_WR_DMA_DONE_INT_SET The software set bit for SPI_SLV_WR_DMA_DONE_INT interrupt. (WT)

SPI_SLV_RD_BUF_DONE_INT_SET The software set bit for SPI_SLV_RD_BUF_DONE_INT interrupt. (WT)

SPI_SLV_WR_BUF_DONE_INT_SET The software set bit for SPI_SLV_WR_BUF_DONE_INT interrupt. (WT)

SPI_TRANS_DONE_INT_SET The software set bit for SPI_TRANS_DONE_INT interrupt. (WT)

SPI_DMA_SEG_TRANS_DONE_INT_SET The software set bit for SPI_DMA_SEG_TRANS_DONE_INT interrupt. (WT)

SPI_SEG_MAGIC_ERR_INT_SET The software set bit for SPI_SEG_MAGIC_ERR_INT interrupt. (WT)

SPI_SLV_CMD_ERR_INT_SET The software set bit for SPI_SLV_CMD_ERR_INT interrupt. (WT)

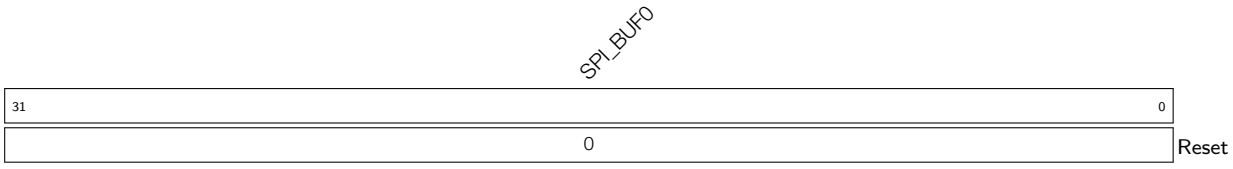
SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET The software set bit for SPI_MST_RX_AFIFO_WFULL_ERR_INT interrupt. (WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET The software set bit for SPI_MST_TX_AFIFO_REMPTY_ERR_INT interrupt. (WT)

SPI_APP2_INT_SET The software set bit for SPI_APP2_INT interrupt. (WT)

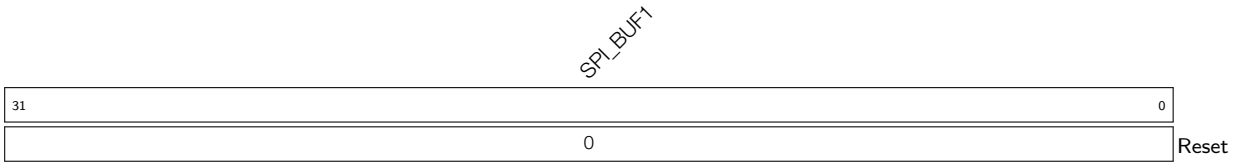
SPI_APP1_INT_SET The software set bit for SPI_APP1_INT interrupt. (WT)

Register 20.19. SPI_W0_REG (0x0098)



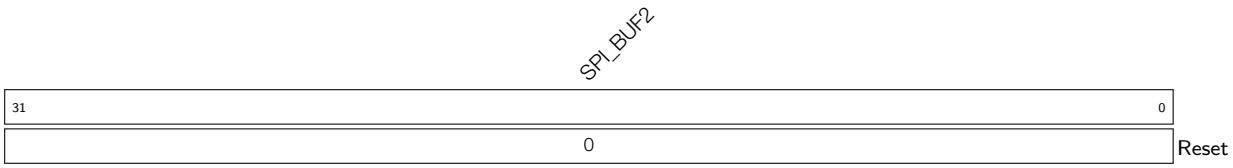
SPI_BUF0 32-bit data buffer 0. (R/W/SS)

Register 20.20. SPI_W1_REG (0x009C)



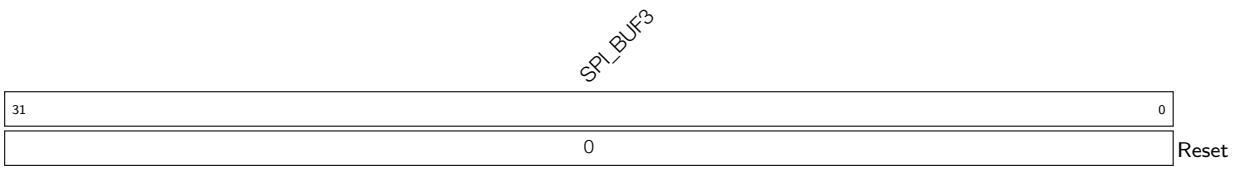
SPI_BUF1 32-bit data buffer 1. (R/W/SS)

Register 20.21. SPI_W2_REG (0x00A0)



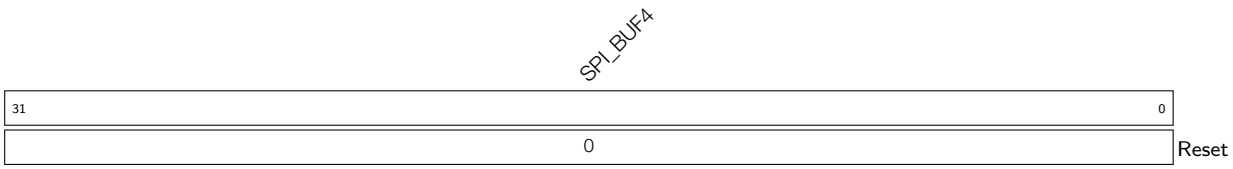
SPI_BUF2 32-bit data buffer 2. (R/W/SS)

Register 20.22. SPI_W3_REG (0x00A4)



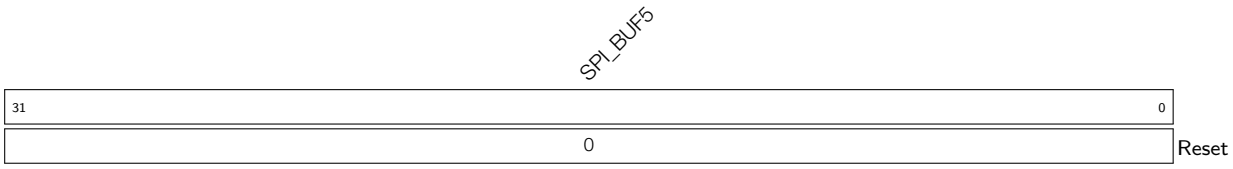
SPI_BUF3 32-bit data buffer 3. (R/W/SS)

Register 20.23. SPI_W4_REG (0x00A8)



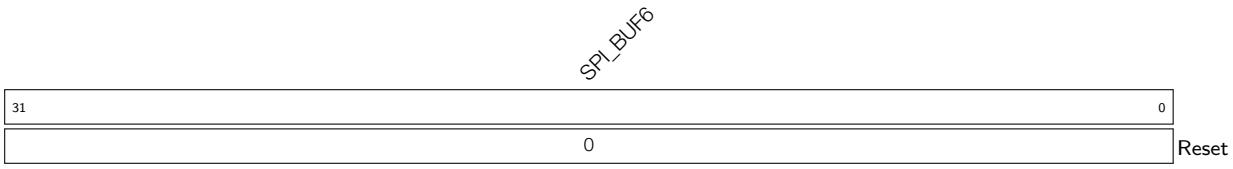
SPI_BUF4 32-bit data buffer 4. (R/W/SS)

Register 20.24. SPI_W5_REG (0x00AC)



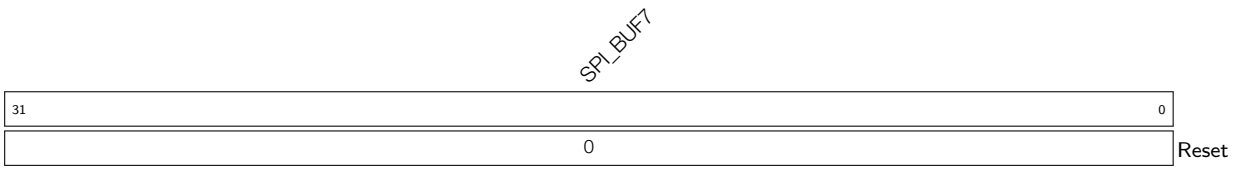
SPI_BUF5 32-bit data buffer 5. (R/W/SS)

Register 20.25. SPI_W6_REG (0x00B0)



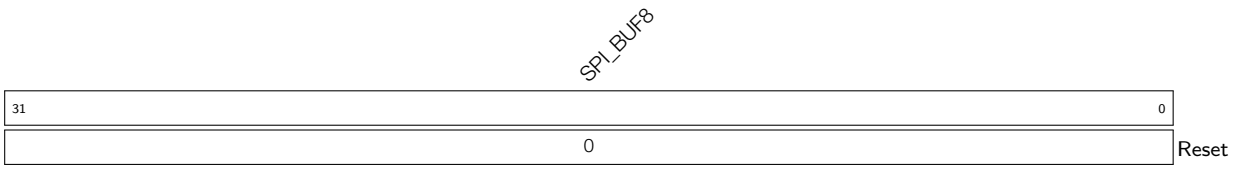
SPI_BUF6 32-bit data buffer 6. (R/W/SS)

Register 20.26. SPI_W7_REG (0x00B4)



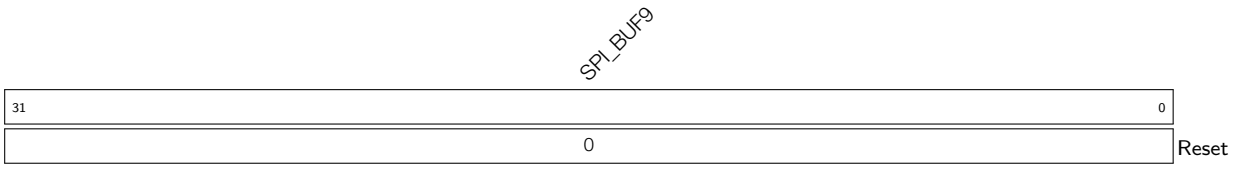
SPI_BUF7 32-bit data buffer 7. (R/W/SS)

Register 20.27. SPI_W8_REG (0x00B8)



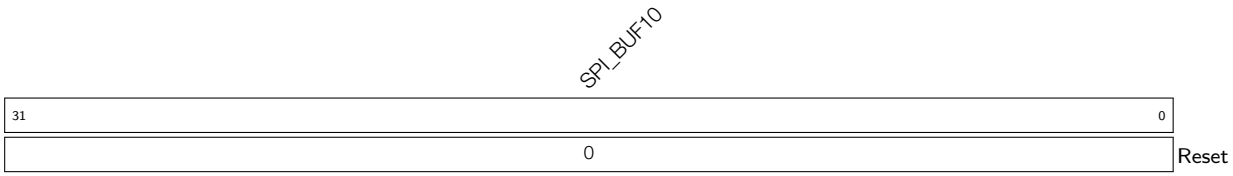
SPI_BUF8 32-bit data buffer 8. (R/W/SS)

Register 20.28. SPI_W9_REG (0x00BC)



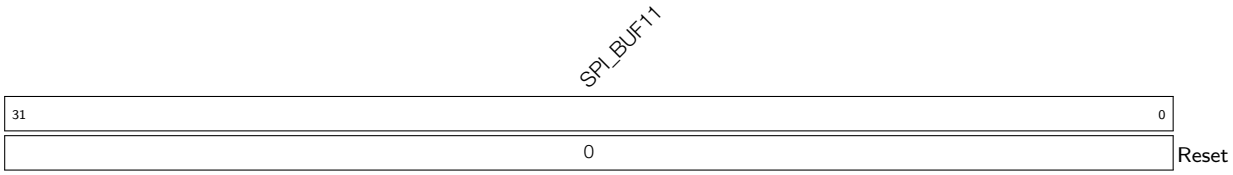
SPI_BUF9 32-bit data buffer 9. (R/W/SS)

Register 20.29. SPI_W10_REG (0x00C0)



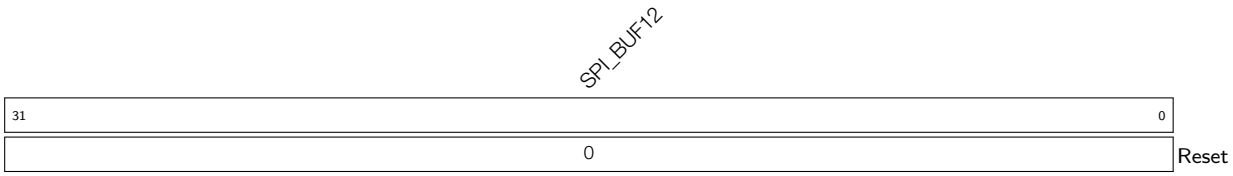
SPI_BUF10 32-bit data buffer 10. (R/W/SS)

Register 20.30. SPI_W11_REG (0x00C4)



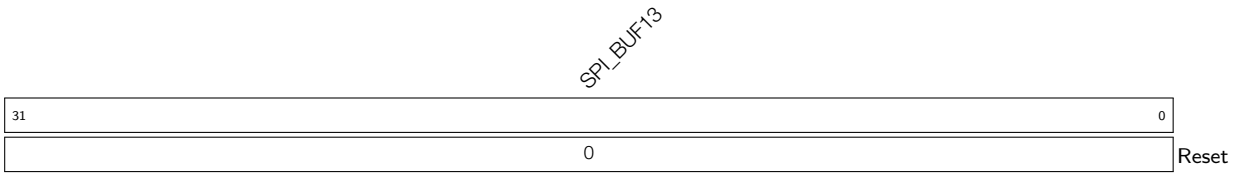
SPI_BUF11 32-bit data buffer 11. (R/W/SS)

Register 20.31. SPI_W12_REG (0x00C8)



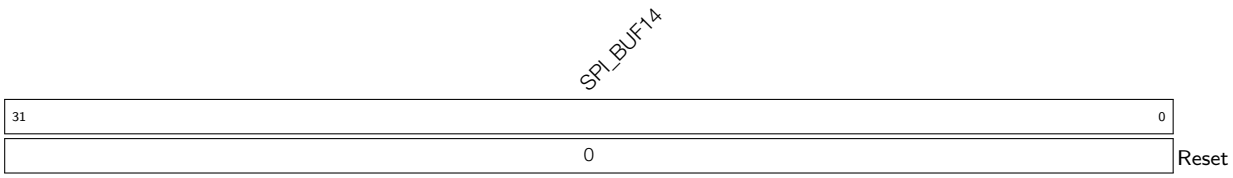
SPI_BUF12 32-bit data buffer 12. (R/W/SS)

Register 20.32. SPI_W13_REG (0x00CC)



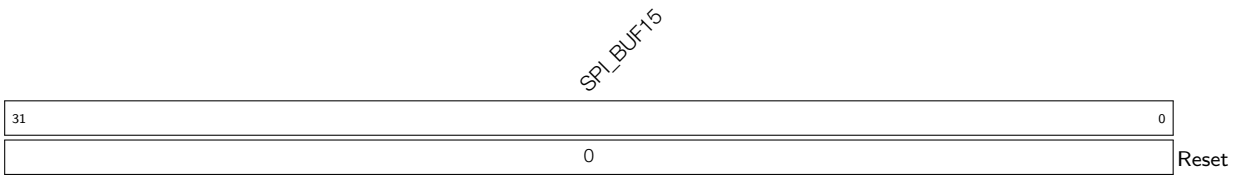
SPI_BUF13 32-bit data buffer 13. (R/W/SS)

Register 20.33. SPI_W14_REG (0x00D0)



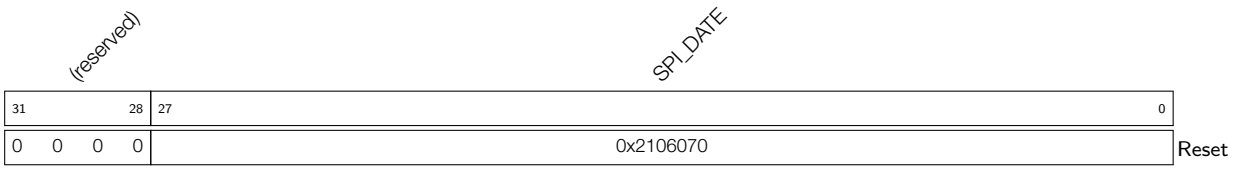
SPI_BUF14 32-bit data buffer 14. (R/W/SS)

Register 20.34. SPI_W15_REG (0x00D4)



SPI_BUF15 32-bit data buffer 15. (R/W/SS)

Register 20.35. SPI_DATE_REG (0x00F0)



SPI_DATE Version control register. (R/W)

21 I2C Master Controller (I2C)

The I2C (Inter-Integrated Circuit) bus allows ESP8684 to communicate with multiple external devices. These external devices can share one bus.

ESP8684 provides one I2C controller operating in master mode.

21.1 Overview

I2C is a two-wire bus, consisting of a serial data line (SDA) and a serial clock line (SCL). Both SDA and SCL lines are open-drain. The I2C bus can be connected to a single or multiple master devices and a single or multiple slave devices. However, only one master device can access a slave at a time via the bus.

The master initiates communication by generating a START condition: pulling the SDA line low while SCL is high, and sending nine clock pulses via SCL. The first eight pulses are used to transmit a 7-bit address followed by a read/write ($R/\overline{W}$) bit. If the address of an I2C slave matches the 7-bit address transmitted, this matching slave can respond by pulling SDA low on the ninth clock pulse. The master can send data to the slave according to the $R/\overline{W}$ bit. Whether to terminate the data transfer or not is determined by the logic level of the acknowledge (ACK) bit. During data transfer, SDA changes only when SCL is low. Once finishing communication, the master sends a STOP condition: pulling SDA up while SCL is high. If a master both reads and writes data in one transfer, then it should send a RSTART condition, a slave address and a $R/\overline{W}$ bit before changing its operation. The RSTART condition is used to change the transfer direction and the mode of the devices (master mode or slave mode).

21.2 Features

The I2C master controller has the following features:

- Master mode only
- Communication between multiple masters
- Standard mode (100 Kbit/s)
- Fast mode (400 Kbit/s)
- 7-bit and 10-bit slave addressing
- Continuous data transfer achieved by pulling SCL low
- Programmable digital noise filtering
- Double addressing mode, which uses slave address and slave memory or register address

21.3 I2C Architecture

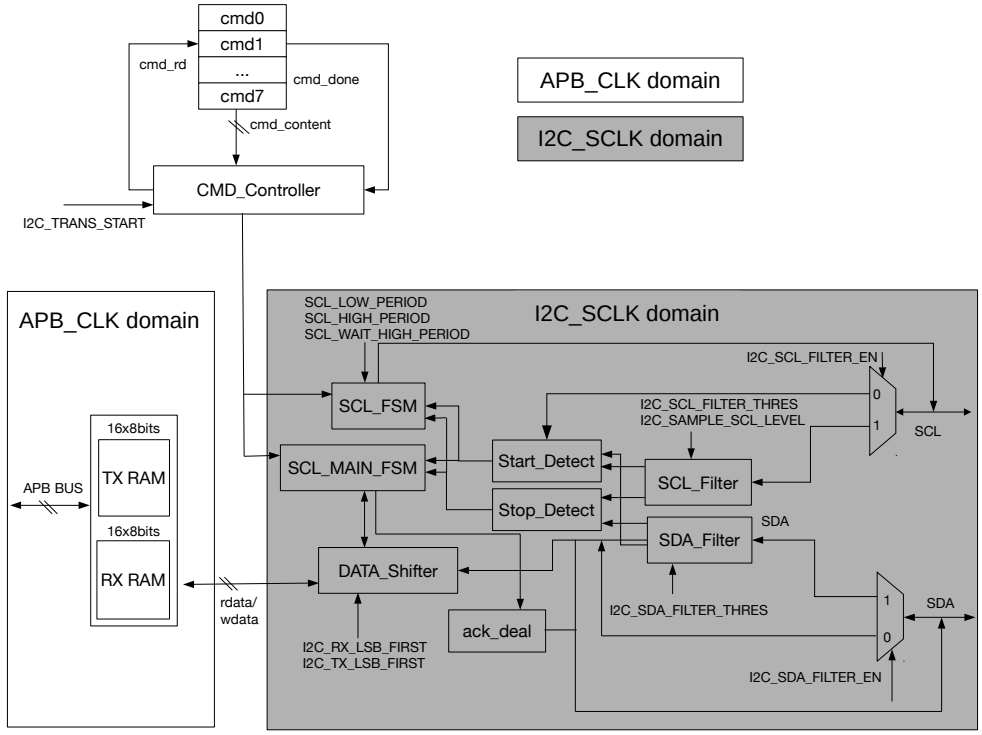


Figure 21-1. I2C Master Architecture

Figure 21-1 shows the architecture of an I2C master. The I2C master controller has the following main parts:

- transmit and receive memory (TX/RX RAM)
- command controller (CMD_Controller)
- SCL clock controller (SCL_FSM)
- SDA data controller (SCL_MAIN_FSM)
- serial/parallel data converter (DATA_Shifter)
- filter for SCL (SCL_Filter)
- filter for SDA (SDA_Filter)
- ACK bit controller (ACK_deal)

Besides, the I2C master controller also has a clock module which generates I2C clocks, and a synchronization module which synchronizes the APB bus and the I2C master controller.

The clock module is used to select clock sources, turn on and off clocks, and divide clocks. SCL_Filter and SDA_Filter remove noises on SCL input signals and SDA input signals respectively. The synchronization module synchronizes signal transfer between different clock domains.

Figure 21-2 and Figure 21-3 are the timing diagram and corresponding parameters of the I2C protocol. SCL_FSM generates the SCL clock timing sequence conforming to the I2C protocol.

SCL_MAIN_FSM controls the execution of I2C commands and the sequence of the SDA line. Also, it controls the ACK_deal module to generate ACK bit or detect the level of ACK bit on SDA line. CMD_Controller is used for an I2C master to generate (R)START, STOP, WRITE, READ and END commands. TX RAM and RX RAM store data to be transmitted and data received respectively. DATA_Shifter shifts data between serial and parallel form.

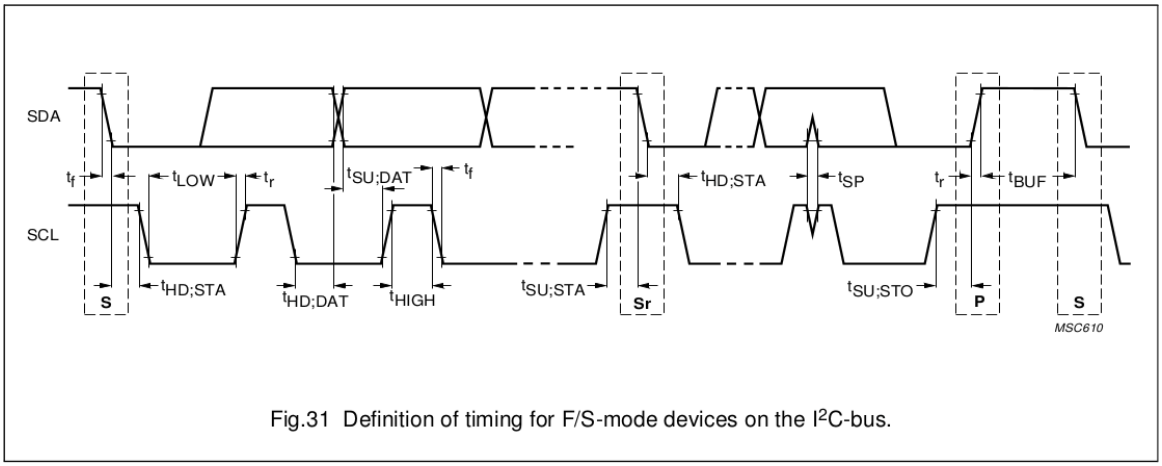


Fig.31 Definition of timing for F/S-mode devices on the I²C-bus.

Figure 21-2. I2C Protocol Timing (Cited from Fig. 31 in The I2C-bus specification Version 2.1)

PARAMETER	SYMBOL	STANDARD-MODE		FAST-MODE		UNIT
		MIN.	MAX.	MIN.	MAX.	
SCL clock frequency	f_{SCL}	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	4.0	–	0.6	–	µs
LOW period of the SCL clock	t_{LOW}	4.7	–	1.3	–	µs
HIGH period of the SCL clock	t_{HIGH}	4.0	–	0.6	–	µs
Set-up time for a repeated START condition	$t_{SU;STA}$	4.7	–	0.6	–	µs
Data hold time: for CBUS compatible masters (see NOTE, Section 10.1.3) for I²C-bus devices	$t_{HD;DAT}$	 5.0 0(2)	 – 3.45(3)	 – 0(2)	 – 0.9(3)	 µs µs
Data set-up time	$t_{SU;DAT}$	250	–	100(4)	–	ns
Rise time of both SDA and SCL signals	t_r	–	1000	20 + 0.1C_b(5)	300	ns
Fall time of both SDA and SCL signals	t_f	–	300	20 + 0.1C_b(5)	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	–	0.6	–	µs
Bus free time between a STOP and START condition	t_{BUF}	4.7	–	1.3	–	µs

Figure 21-3. I2C Timing Parameters (Cited from Table 5 in The I2C-bus specification Version 2.1)

21.4 Functional Description

Note that operations may differ between the I2C master controller in ESP8684 and other masters or slaves on the bus. Please refer to datasheets of individual I2C devices for specific information.

21.4.1 Clock Configuration

Registers, TX RAM, and RX RAM are configured and accessed in the APB_CLK clock domain. The main logic of the I2C master controller, including SCL_FSM, SCL_MAIN_FSM, SCL_FILTER, SDA_FILTER, and

DATA_SHIFTER, are in the I2C_SCLK clock domain.

You can choose the clock source for I2C_SCLK from XTAL_CLK or FOSC_CLK via I2C_SCLK_SEL. When I2C_SCLK_SEL is cleared, the clock source is XTAL_CLK. When I2C_SCLK_SEL is set, the clock source is FOSC_CLK. The clock source is enabled by configuring I2C_SCLK_ACTIVE as high level, which passes through a fractional divider to generate I2C_SCLK according to the following equation:

$$Divisor = I2C\_SCLK\_DIV\_NUM + 1 + \frac{I2C\_SCLK\_DIV\_A}{I2C\_SCLK\_DIV\_B}$$

The frequency of XTAL_CLK is 40 MHz, while the frequency of FOSC_CLK is 17.5 MHz. Limited by timing parameters, the derived clock I2C_SCLK should operate at a frequency 20 timers larger than SCL's frequency.

21.4.2 SCL and SDA Noise Filtering

SCL_Filter and SDA_Filter modules are identical and are used to filter signal noises on SCL and SDA, respectively. These filters can be enabled or disabled by configuring I2C_SCL_FILTER_EN and I2C_SDA_FILTER_EN.

Take SCL_Filter as an example. When enabled, SCL_Filter samples input signals on the SCL line continuously. These input signals are valid only if they remain unchanged for consecutive I2C_SCL_FILTER_THRES I2C_SCLK clock cycles. Given that only valid input signals can pass through the filter, SCL_Filter can remove glitches whose pulse width is shorter than I2C_SCL_FILTER_THRES I2C_SCLK clock cycles, while SDA_Filter can remove glitches whose pulse width is shorter than I2C_SDA_FILTER_THRES I2C_SCLK clock cycles.

21.4.3 Generating SCL Pulses in Idle State

Usually when the I2C bus is idle, the SCL line is held high. The I2C master controller in ESP8684 can be programmed to generate SCL pulses in idle state. If the I2C_SCL_RST_SLV_EN bit is set, hardware will send I2C_SCL_RST_SLV_NUM SCL pulses. When software reads 0 in I2C_SCL_RST_SLV_EN (this bit is cleared automatically by hardware), set I2C_CONF_UPGATE to stop this function.

21.4.4 Synchronization

I2C registers are configured in APB_CLK domain, whereas the I2C master controller is configured in asynchronous I2C_SCLK domain. Therefore, before being used by the I2C master controller, register values should be synchronized by first writing configuration registers and then writing 1 to I2C_CONF_UPGATE. Registers that need synchronization are listed in Table 21-1.

Table 21-1. I2C Registers that Need Synchronization

Register	Parameter	Address
I2C_CTR_REG	I2C_SLV_TX_AUTO_START_EN	0x0004
	I2C_SDA_FORCE_OUT	
	I2C_SCL_FORCE_OUT	
	I2C_SAMPLE_SCL_LEVEL	
	I2C_RX_FULL_ACK_LEVEL	
	I2C_MS_MODE	
	I2C_TX_LSB_FIRST	
	I2C_RX_LSB_FIRST	
	I2C_ARBITRATION_EN	

I2C_TO_REG	I2C_TIME_OUT_EN	0x000C
	I2C_TIME_OUT_VALUE	
I2C_SCL_SP_CONF_REG	I2C_SDA_PD_EN	0x0080
	I2C_SCL_PD_EN	
	I2C_SCL_RST_SLV_NUM	
	I2C_SCL_RST_SLV_EN	
I2C_SCL_LOW_PERIOD_REG	I2C_SCL_LOW_PERIOD	0x0000
I2C_SCL_HIGH_PERIOD_REG	I2C_WAIT_HIGH_PERIOD	0x0038
	I2C_HIGH_PERIOD	
I2C_SDA_HOLD_REG	I2C_SDA_HOLD_TIME	0x0030
I2C_SDA_SAMPLE_REG	I2C_SDA_SAMPLE_TIME	0x0034
I2C_SCL_START_HOLD_REG	I2C_SCL_START_HOLD_TIME	0x0040
I2C_SCL_RSTART_SETUP_REG	I2C_SCL_RSTART_SETUP_TIME	0x0044
I2C_SCL_STOP_HOLD_REG	I2C_SCL_STOP_HOLD_TIME	0x0048
I2C_SCL_STOP_SETUP_REG	I2C_SCL_STOP_SETUP_TIME	0x004C
I2C_SCL_ST_TIME_OUT_REG	I2C_SCL_ST_TO_I2C	0x0078
I2C_SCL_MAIN_ST_TIME_OUT_REG	I2C_SCL_MAIN_ST_TO_I2C	0x007C
I2C_FILTER_CFG_REG	I2C_SCL_FILTER_EN	0x0050
	I2C_SCL_FILTER_THRES	
	I2C_SDA_FILTER_EN	
	I2C_SDA_FILTER_THRES	

21.4.5 Open-Drain Output

SCL and SDA output drivers must be configured as open drain. There are two ways to achieve this:

1. Set I2C_SCL_FORCE_OUT and I2C_SDA_FORCE_OUT, and configure GPIO_PIN*n*_PAD_DRIVER for corresponding SCL and SDA pads as open-drain.
2. Clear I2C_SCL_FORCE_OUT and I2C_SDA_FORCE_OUT.

Because these lines are configured as open-drain, the low-to-high transition time of each line is longer, determined together by the pull-up resistor and the line capacitance. The output duty cycle of I2C is limited by the SDA and SCL line's pull-up speed, mainly SCL's speed.

In addition, when I2C_SCL_FORCE_OUT and I2C_SCL_PD_EN are set to 1, SCL can be forced low; when I2C_SDA_FORCE_OUT and I2C_SDA_PD_EN are set to 1, SDA can be forced low.

21.4.6 Timing Parameter Configuration

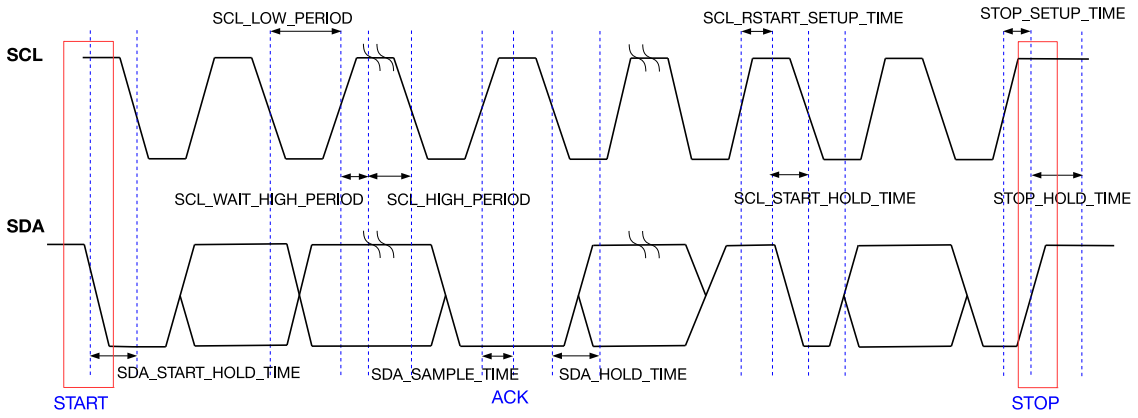


Figure 21-4. I2C Timing Diagram

Figure 21-4 shows the timing diagram of an I2C master. This figure also specifies registers used to configure the START bit, STOP bit, data hold time, data sample time, waiting time on the rising SCL edge, etc. Timing parameters are calculated as follows in I2C_SCLK clock cycles:

1. $t_{LOW} = (I2C\_SCL\_LOW\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
2. $t_{HIGH} = (I2C\_SCL\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
3. $t_{SU:STA} = (I2C\_SCL\_RSTART\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
4. $t_{HD:STA} = (I2C\_SCL\_START\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
5. $t_{r} = (I2C\_SCL\_WAIT\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
6. $t_{SU:STO} = (I2C\_SCL\_STOP\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
7. $t_{BUF} = (I2C\_SCL\_STOP\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
8. $t_{HD:DAT} = (I2C\_SDA\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
9. $t_{SU:DAT} = (I2C\_SCL\_LOW\_PERIOD - I2C\_SDA\_HOLD\_TIME) \cdot T_{I2C\_SCLK}$

Timing registers are:

1. I2C_SCL_START_HOLD_TIME: Specifies the interval between pulling SDA low and pulling SCL low when the master generates a START condition. This interval is (I2C_SCL_START_HOLD_TIME +1) in I2C_SCLK cycles.
2. I2C_SCL_LOW_PERIOD: Specifies the low period of SCL. This period lasts (I2C_SCL_LOW_PERIOD + 1) in I2C_SCLK cycles. However, it could be extended when SCL is pulled low by peripheral devices or by an END command executed by the I2C master controller, or when the clock is stretched.
3. I2C_SCL_WAIT_HIGH_PERIOD: Specifies time for SCL to go high in I2C_SCLK cycles. Please make sure that SCL could be pulled high within this time period. Otherwise, the high period of SCL may be incorrect.
4. I2C_SCL_HIGH_PERIOD: Specifies the high period of SCL in I2C_SCLK cycles. When SCL goes high within (I2C_SCL_WAIT_HIGH_PERIOD + 1) in I2C_SCLK cycles, its frequency is:

$$f_{scl} = \frac{f_{\text{I2C\_SCLK}}}{\text{I2C\_SCL\_LOW\_PERIOD} + \text{I2C\_SCL\_HIGH\_PERIOD} + \text{I2C\_SCL\_WAIT\_HIGH\_PERIOD}+3}$$

5. I2C_SDA_SAMPLE_TIME: Specifies the interval between the rising edge of SCL and the level sampling time of SDA. It is advised to set a value in the middle of SCL's high period, so as to correctly sample the level of SCL.
6. I2C_SDA_HOLD_TIME: Specifies the interval between changing the SDA output level and the falling edge of SCL.

Timing parameters limits corresponding register configuration.

1. $\frac{f_{I2C\_SCLK}}{f_{SCL}} > 20$
2. $3 \times f_{I2C\_SCLK} \leq (I2C\_SDA\_HOLD\_TIME - 4) \times f_{APB\_CLK}$
3. I2C_SDA_HOLD_TIME + I2C_SCL_START_HOLD_TIME > SDA_FILTER_THRES + 3
4. I2C_SCL_WAIT_HIGH_PERIOD < I2C_SDA_SAMPLE_TIME < I2C_SCL_HIGH_PERIOD
5. I2C_SDA_SAMPLE_TIME < I2C_SCL_WAIT_HIGH_PERIOD + I2C_SCL_START_HOLD_TIME + I2C_SCL_RSTART_SETUP_TIME

21.4.7 Timeout Control

The I2C master controller has three types of timeout control, namely timeout control for SCL_FSM, for SCL_MAIN_FSM, and for the SCL line. The first two are always enabled, while enabling the third is configurable.

When SCL_FSM remains unchanged for more than $2^{I2C\_SCL\_ST\_TO\_I2C}$ clock cycles, an I2C_SCL_ST_TO_INT interrupt is triggered, and then SCL_FSM goes to idle state. The value of I2C_SCL_ST_TO_I2C should be less than or equal to 22, which means SCL_FSM could remain unchanged for 2^{22} I2C_SCLK clock cycles at most before the interrupt is generated.

When SCL_MAIN_FSM remains unchanged for more than $2^{I2C\_SCL\_MAIN\_ST\_TO\_I2C}$ clock cycles, an I2C_SCL_MAIN_ST_TO_INT interrupt is triggered, and then SCL_MAIN_FSM goes to idle state. The value of I2C_SCL_MAIN_ST_TO_I2C should be less than or equal to 22, which means SCL_MAIN_FSM could remain unchanged for 2^{22} I2C_SCLK clock cycles at most before the interrupt is generated.

Timeout control for SCL is enabled by setting I2C_TIME_OUT_EN. When the level of SCL remains unchanged for more than I2C_TIME_OUT_VALUE clock cycles, an I2C_TIME_OUT_INT interrupt is triggered, and then the I2C bus goes to idle state.

21.4.8 Command Configuration

The CMD_Controller of the I2C master reads commands from 8 sequential command registers and controls SCL_FSM and SCL_MAIN_FSM accordingly.

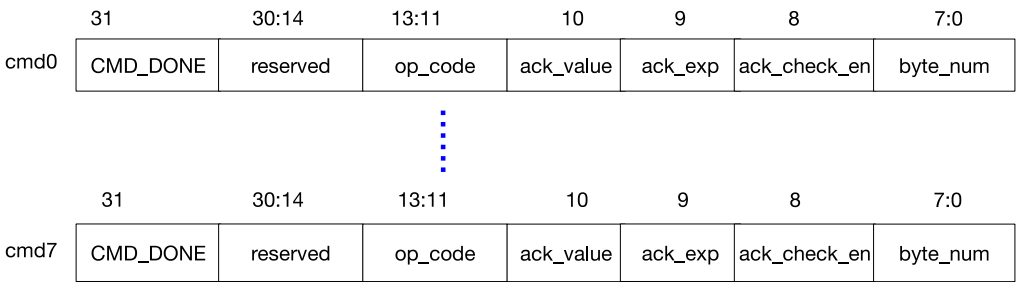


Figure 21-5. Structure of I2C Command Registers

Figure 21-5 illustrates the structure of command registers. Fields of command registers are:

1. CMD_DONE: Indicates that a command has been executed. After each command has been executed, the CMD_DONE bit in the corresponding command register is set to 1 by hardware. By reading this bit, software can tell if the command has been executed. When writing new commands, this bit must be cleared by software.
2. op_code: Indicates the command. The I2C master controller supports five commands:
   - RSTART: op_code = 6. The I2C master controller sends a START bit or a RSTART bit defined by the I2C protocol.
   - WRITE: op_code = 1. The I2C master controller sends a slave address, a register address (only in double addressing mode) and data to the slave.
   - READ: op_code = 3. The I2C master controller reads data from the slave.
   - STOP: op_code = 2. The I2C master controller sends a STOP bit defined by the I2C protocol. This code also indicates that the command sequence has been executed, and the CMD_Controller stops reading commands. After restarted by software, the CMD_Controller resumes reading commands from command register 0.
   - END: op_code = 4. The I2C master controller pulls the SCL line low and suspends I2C communication. This code also indicates that the command sequence has completed, and the CMD_Controller stops executing commands. Once software refreshes data in command registers and the RAM, the CMD_Controller can be restarted to execute commands from command register 0 again.
3. ack_value: Used to configure the level of the ACK bit sent by the I2C master controller during a read operation. This bit is ignored in RSTART, STOP, END and WRITE conditions.
4. ack_exp: Used to configure the level of the ACK bit expected by the I2C master controller during a write operation. This bit is ignored during RSTART, STOP, END and READ conditions.
5. ack_check_en: Used to enable the I2C master controller during a write operation to check whether the ACK level sent by the slave matches ack_exp in the command. If this bit is set and the level received does not match ack_exp in the WRITE command, the master will generate an I2C_NACK_INT interrupt and a STOP condition for data transfer. If this bit is cleared, the controller will not check the ACK level sent by the slave. This bit is ignored during RSTART, STOP, END and READ conditions.
6. byte_num: Specifies the length of data (in bytes) to be read or written. Can range from 1 to 255 bytes. This bit is ignored during RSTART, STOP and END conditions.

Each command sequence is executed starting from command register 0 and terminated by a STOP or an END. Therefore, there must be a STOP or an END command in one command sequence.

A complete data transfer on the I2C bus should be initiated by a START and terminated by a STOP. The transfer process may be completed using multiple sequences, each one separated by an END command. Each sequence may differ in the direction of data transfer, clock frequency, slave addresses, data length, etc. This allows efficient use of available peripheral RAM and also achieves more flexible I2C communication.

21.4.9 TX/RX RAM Data Storage

Both TX RAM and RX RAM are 16 × 8 bits, and can be accessed in FIFO or non-FIFO mode. If I2C_NONFIFO_EN bit is cleared, both RAMs are accessed in FIFO mode; if I2C_NONFIFO_EN bit is set, both

RAMs are accessed in non-FIFO mode.

TX RAM stores data that the I2C master controller needs to send. During communication, when the I2C master controller needs to send data (except acknowledgement bits), it reads data from TX RAM and sends them sequentially via SDA. All data must be stored in TX RAM in the order they will be sent to slaves. The data stored in TX RAM include slave addresses, read/write bits, register addresses (only in double addressing mode) and data to be sent.

TX RAM can be read and written by the CPU. The CPU writes to TX RAM either in FIFO mode or in non-FIFO mode (direct address). In FIFO mode, the CPU writes to TX RAM via the fixed address I2C_DATA_REG, with addresses for writing in TX RAM incremented automatically by hardware. In non-FIFO mode, the CPU accesses TX RAM directly via address fields (I2C Base Address + 0x100) ~(I2C Base Address + 0x17C). Each byte in TX RAM occupies an entire word in the address space. Therefore, the address of the first byte is (I2C Base Address + 0x100), the second byte is (I2C Base Address + 0x104), the third byte is (I2C Base Address + 0x108), and so on. The CPU can only read TX RAM via direct addresses. Addresses for reading TX RAM are the same with addresses for writing TX RAM.

RX RAM stores data the I2C master controller receives during communication. Values of RX RAM can be read by software after I2C communication completes.

RX RAM can only be read by the CPU. The CPU reads RX RAM either in FIFO mode or in non-FIFO mode (direct address). In FIFO mode, the CPU reads RX RAM via the fixed address I2C_DATA_REG, with addresses for reading RX RAM incremented automatically by hardware. In non-FIFO mode, the CPU accesses TX RAM directly via address fields (I2C Base Address + 0x180) ~(I2C Base Address + 0x1FC). Each byte in RX RAM occupies an entire word in the address space. Therefore, the address of the first byte is I2C Base Address + 0x180, the second byte is I2C Base Address + 0x184, the third byte is I2C Base Address + 0x188 and so on.

In FIFO mode, TX RAM of a master may wrap around to send data larger than the FIFO depth (for ESP8684 the depth is 16 bytes). Set I2C_FIFO_PRT_EN. If the size of data to be sent is smaller than I2C_TXFIFO_WM_THRHD, an I2C_TXFIFO_WM_INT interrupt is generated. After receiving the interrupt, software continues writing to I2C_DATA_REG. Please ensure that software writes to or refreshes TX RAM before the master sends data, otherwise it may result in unpredictable consequences.

In FIFO mode, RX RAM of a slave may also wrap around to receive data larger than the FIFO depth (for ESP8684 the depth is 16 bytes). Set I2C_FIFO_PRT_EN and clear I2C_RX_FULL_ACK_LEVEL. If data already received (to be overwritten) is larger than I2C_RXFIFO_WM_THRHD, an I2C_RXFIFO_WM_INT interrupt is generated. After receiving the interrupt, software continues reading from I2C_DATA_REG.

21.4.10 Data Conversion

DATA_Shifter is used for serial/parallel conversion, converting byte data in TX RAM to an outgoing serial bitstream or an incoming serial bitstream to byte data in RX RAM. I2C_RX_LSB_FIRST and I2C_TX_LSB_FIRST can be used to select LSB- or MSB-first storage and transmission of data.

21.4.11 Addressing Mode

Besides 7-bit addressing, the ESP8684 I2C also supports 10-bit addressing and double addressing. 10-bit addressing can be mixed with 7-bit addressing.

Define the slave address as SLV_ADDR. In 7-bit addressing mode, the slave address is SLV_ADDR[6:0]; in 10-bit addressing mode, the slave address is SLV_ADDR[9:0].

In 7-bit addressing mode, the master only needs to send one byte of address, which comprises SLV_ADDR[6:0] and a $R/\overline{W}$ bit. In 7-bit addressing mode, there is a special case called general call addressing (broadcast). When the master sends the general call address (0x00) and the $R/\overline{W}$ bit is 0, slaves that support general call addressing respond to the master regardless of their own address.

In 10-bit addressing mode, the master needs to send two bytes of address. The first byte is slave_addr_first_7bits followed by a $R/\overline{W}$ bit, and slave_addr_first_7bits should be configured as (0x78 | SLV_ADDR[9:8]). The second byte is slave_addr_second_byte, which should be configured as SLV_ADDR[7:0]. Since a 10-bit slave address has one more byte than a 7-bit address, byte_num of the WRITE command and the number of bytes in the RAM increase by one.

Some I2C slaves support double addressing, where the first address is the address of an I2C slave, and the second one is the slave's memory address. ESP8684 I2C also supports double addressing.

21.4.12 Starting of the I2C Master Controller

To start the I2C master controller, after configuring the controller to master mode (I2C_MS_MODE) and command registers, write 1 to I2C_TRANS_START in order that the master starts to parse and execute command sequences. The master always executes a command sequence starting from command register 0 to a STOP or an END at the end. To execute another command sequence starting from command register 0, refresh commands by writing 1 again to I2C_TRANS_START.

21.5 Programming Example

This section provides programming examples for typical communication scenarios. ESP8684 has one I2C master controller. For the convenience of description, the I2C master in subsequent figures is ESP8684's I2C master controller, and the I2C slave are controllers compliant with The I2C-bus specification Version 2.1 and have corresponding functions. I2C master is referred to as $I2C_{master}$, and I2C slave is referred to as $I2C_{slave}$.

21.5.1 $I2C_{master}$ Writes to $I2C_{slave}$ with a 7-bit Address in One Command Sequence

21.5.1.1 Introduction

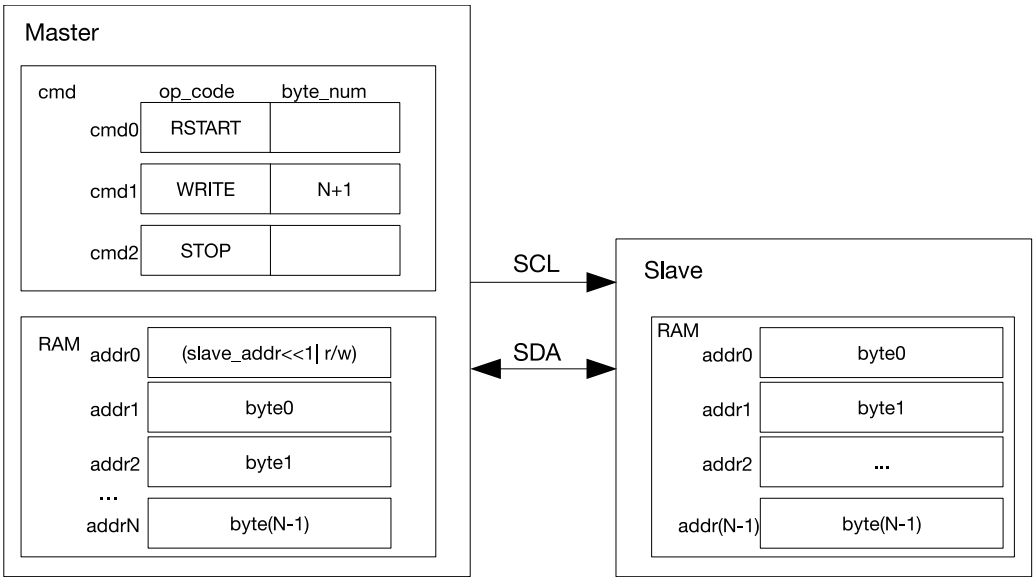


Figure 21-6. $I2C_{master}$ Writing to $I2C_{slave}$ with a 7-bit Address

Figure 21-6 shows how $I2C_{master}$ writes N bytes of data to $I2C_{slave}$'s registers and RAM using 7-bit addressing. As shown in figure 21-6 , the first byte in the RAM of $I2C_{master}$ is a 7-bit $I2C_{slave}$ address followed by a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit is 0, it indicates a WRITE operation. The remaining bytes are used to store data ready for transfer. The cmd box contains related command sequences.

After the command sequence is configured and data in RAM is ready, $I2C_{master}$ enables the controller and initiates data transfer by setting the I2C_TRANS_START bit. The controller has four steps to take:

1. Wait for SCL to go high, to avoid SCL being used by other masters or slaves.
2. Execute a RSTART command and send a START bit.
3. Execute a WRITE command by taking N+1 bytes from the RAM in order and send them to $I2C_{slave}$ in the same order. The first byte is the address of $I2C_{slave}$.
4. Send a STOP. Once the $I2C_{master}$ transfers a STOP bit, an I2C_TRANS_COMPLETE_INT interrupt is generated.

21.5.1.2 Configuration Example

1. Configure the timing parameter registers of $I2C_{master}$ and $I2C_{slave}$ according to Section 21.4.6. Adjust the timing of $I2C_{slave}$ according to its manual.
2. Set I2C_MS_MODE to 1.
3. Write 1 to I2C_CONF_UPGATE to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command register	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2	STOP	—	—	—	—

5. Write $I2C_{slave}$ address and data to be sent to TX RAM of $I2C_{master}$ in either FIFO mode or non-FIFO mode according to Section 21.4.9.
6. Set the address of $I2C_{slave}$ via I2C_SLAVE_ADDR[7:0].
7. Write 1 to I2C_TRANS_START to start transfer, and enable $I2C_{slave}$.
8. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address. When ack_check_en in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as a matching slave by default.
   - Match: If the received ACK value matches ack_exp (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT interrupt and stops data transfer.
9. $I2C_{master}$ sends data, and checks ACK value or not according to ack_check_en.
10. If data to be sent (N) is larger than the depth of TX FIFO, TX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 21.4.9.

11. After data transfer completes, I2C$_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT interrupt.

21.5.2 I2C$_{master}$ Writes to I2C$_{slave}$ with a 10-bit Address in One Command Sequence

21.5.2.1 Introduction

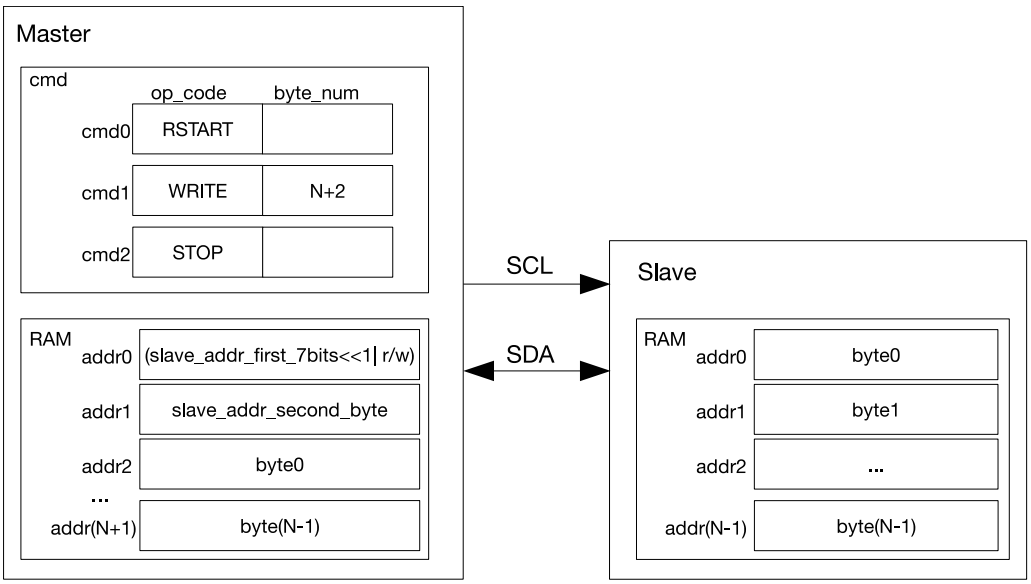


Figure 21-7. I2C$_{master}$ Writing to a Slave with a 10-bit Address

Figure 21-7 shows how I2C$_{master}$ writes N bytes of data using 10-bit addressing to an I2C slave. The configuration and transfer process is similar to what is described in 21.5.1, except that a 10-bit I2C$_{slave}$ address is formed from two bytes. Since a 10-bit I2C$_{slave}$ address has one more byte than a 7-bit I2C$_{slave}$ address, byte_num and length of data in TX RAM increase by 1 accordingly.

21.5.2.2 Configuration Example

1. Set I2C_MS_MODE to 1.
2. Write 1 to I2C_CONF_UPGATE to synchronize registers.
3. Configure command registers of I2C$_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2	STOP	—	—	—	—

4. Set the address of I2C$_{slave}$ via I2C_SLAVE_ADDR[9:0].
5. Write I2C$_{slave}$ address and data to be sent to TX RAM of I2C$_{master}$. The first byte of I2C$_{slave}$ address comprises ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit. The second byte of I2C$_{slave}$ address is I2C_SLAVE_ADDR[7:0]. These two bytes are followed by data to be sent in FIFO or non-FIFO mode.
6. Write 1 to I2C_CONF_UPGATE to synchronize registers.

7. Write 1 to I2C_TRANS_START to start transfer, and enable $I2C_{slave}$.
8. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address. When ack_check_en in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as matching slave by default.
   - Match: If the received ACK value matches ack_exp (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT interrupt and stops data transfer.
9. $I2C_{master}$ sends data, and checks ACK value or not according to ack_check_en.
10. If data to be sent is larger than the depth of TX FIFO, TX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 21.4.9.
11. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT interrupt.

21.5.3 $I2C_{master}$ Writes to $I2C_{slave}$ with Two 7-bit Addresses in One Command Sequence

21.5.3.1 Introduction

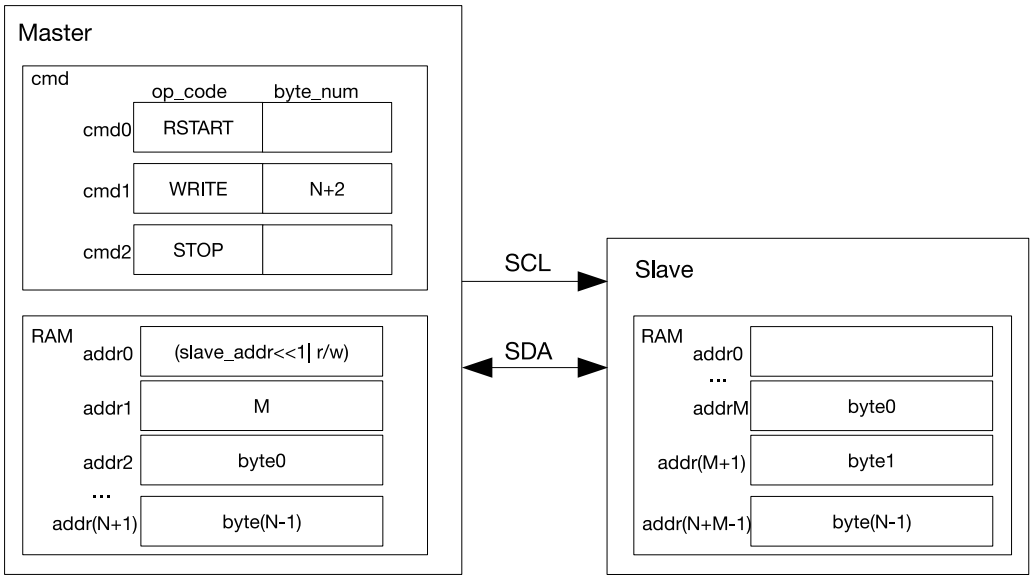


Figure 21-8. $I2C_{master}$ Writing to $I2C_{slave}$ with Two 7-bit Addresses

Figure 21-8 shows how $I2C_{master}$ writes N bytes of data to $I2C_{slave}$'s registers or RAM using 7-bit double addressing. The configuration and transfer process is similar to what is described in Section 21.5.1, except that in 7-bit double addressing mode $I2C_{master}$ sends two 7-bit addresses. The first address is the address of an I2C slave, and the second one is $I2C_{slave}$'s memory address (i.e. addrM in Figure 21-8 on the right). When using double addressing, RAM must be accessed in non-FIFO mode. The I2C slave put received byte0 ~ byte(N-1) into its registers or RAM in an order staring from addrM.

21.5.3.2 Configuration Example

1. Choose an $I2C_{slave}$ that supports double addressing mode and enable this mode.
2. Set I2C_MS_MODE to 1.
3. Write 1 to I2C_CONF_UPGATE to synchronize registers.
4. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2	STOP	—	—	—	—

5. Write $I2C_{slave}$ address and data to be sent to TX RAM of $I2C_{master}$ in FIFO or non-FIFO mode.
6. Set the address of $I2C_{slave}$ via I2C_SLAVE_ADDR[7:0].
7. Write 1 to I2C_CONF_UPGATE to synchronize registers.
8. Write 1 to I2C_TRANS_START to start transfer, and enable $I2C_{slave}$.

9. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address. When ack_check_en in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as matching slave by default.
    - Match: If the received ACK value matches ack_exp (the expected ACK value), $I2C_{master}$ continues data transfer.
    - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT interrupt and stops data transfer.
10. $I2C_{slave}$ receives the RX RAM address sent by $I2C_{master}$ and adds the offset.
11. $I2C_{master}$ sends data, and checks ACK value or not according to ack_check_en.
12. If data to be sent is larger than the depth of TX FIFO, TX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 21.4.9.
13. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT interrupt.

21.5.4 I2C$_{master}$ Writes to I2C$_{slave}$ with a 7-bit Address in Multiple Command Sequences

21.5.4.1 Introduction

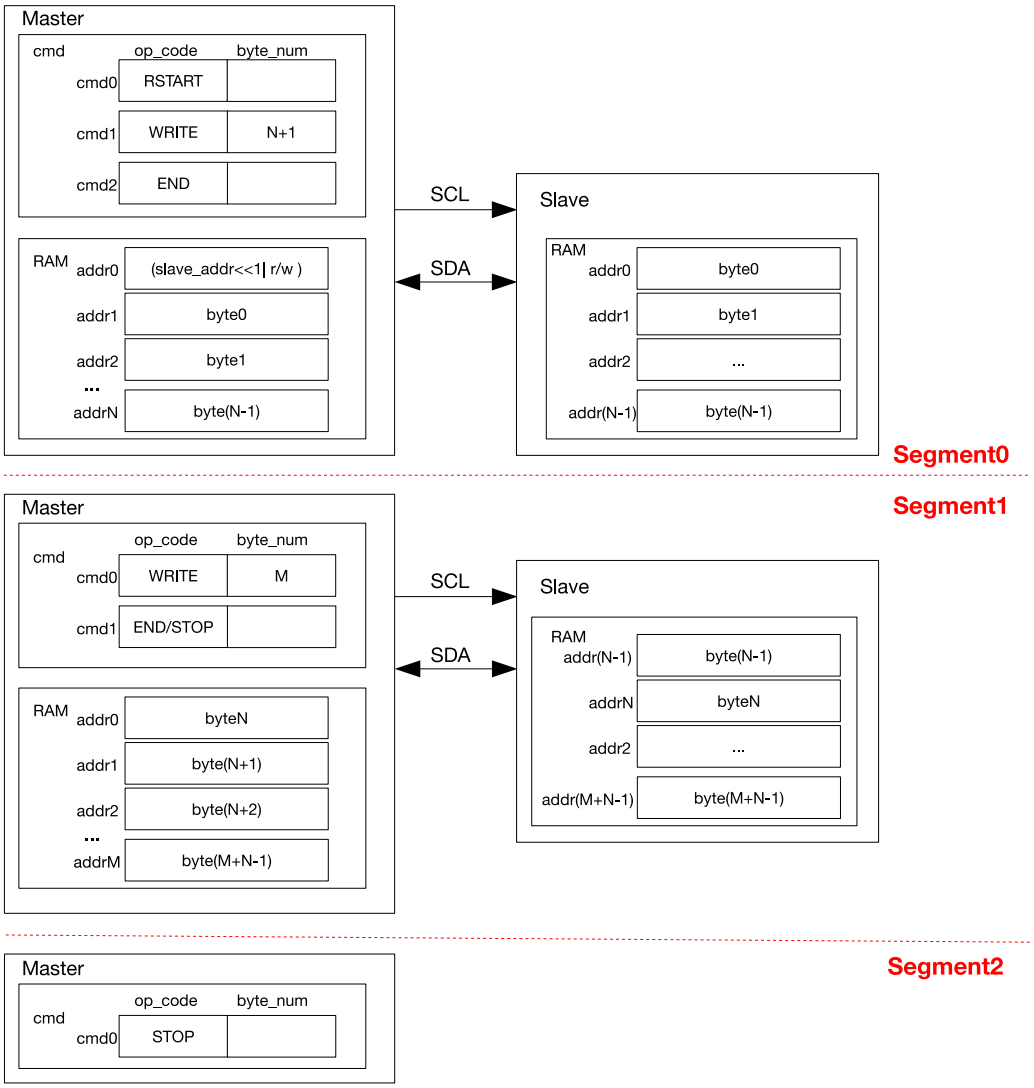


Figure 21-9. I2C$_{master}$ Writing to I2C$_{slave}$ with a 7-bit Address in Multiple Sequences

Given that the I2C RAM holds only 16 bytes, when data are too large to be processed even by the wrapped RAM, it is advised to transmit them in multiple command sequences by adding an END command at the end of every sequence. When the controller executes this END command to pull SCL low, software refreshes command sequence registers and RAM for next the transfer.

Figure 21-9 shows how I2C$_{master}$ writes to an I2C slave in two or three segments as an example. For the first segment, the CMD_Controller registers are configured as shown in Segment0. Once data in I2C$_{master}$'s RAM is ready and I2C_TRANS_START is set, I2C$_{master}$ initiates data transfer. After executing the END command, I2C$_{master}$ turns off the SCL clock and pulls SCL low to reserve the bus. Meanwhile, the controller generates an I2C_END_DETECT_INT interrupt.

For the second segment, after detecting the I2C_END_DETECT_INT interrupt, software refreshes the CMD_Controller registers, reloads the RAM and clears this interrupt, as shown in Segment1. If cmd1 in the second segment is a STOP, then data is transmitted to I2C$_{slave}$ in two segments. I2C$_{master}$ resumes data transfer

after I2C_TRANS_START is set, and terminates the transfer by sending a STOP bit.

For the third segment, after the second data transfer finishes and an I2C_END_DETECT_INT is detected, the CMD_Controller registers of $I2C_{master}$ are configured as shown in Segment2. Once I2C_TRANS_START is set, $I2C_{master}$ generates a STOP bit and terminates the transfer.

Note that other $I2C_{master}$ devices will not transact on the bus between two segments. The bus is only released after a STOP signal is sent. The I2C master controller can be reset by setting I2C_FSM_RST field at any time. This field will later be cleared automatically by hardware.

21.5.4.2 Configuration Example

1. Set I2C_MS_MODE to 1.
2. Write 1 to I2C_CONF_UPGATE to synchronize registers.
3. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2	END	—	—	—	—

4. Write $I2C_{slave}$ address and data to be sent to TX RAM of $I2C_{master}$ in either FIFO mode or non-FIFO mode according to Section 21.4.9.
5. Set the address of $I2C_{slave}$ via I2C_SLAVE_ADDR[7:0].
6. Write 1 to I2C_CONF_UPGATE to synchronize registers.
7. Write 1 to I2C_TRANS_START to start transfer, and enable $I2C_{slave}$.
8. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address. When ack_check_en in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as matching slave by default.
   - Match: If the received ACK value matches ack_exp (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT interrupt and stops data transfer.
9. $I2C_{master}$ sends data, and checks ACK value or not according to ack_check_en.
10. After the I2C_END_DETECT_INT interrupt is generated, set I2C_END_DETECT_INT_CLR to 1 to clear this interrupt.
11. Update $I2C_{master}$'s command registers.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	WRITE	ack_value	ack_exp	1	M
I2C_COMMAND1	END/STOP	—	—	—	—

12. Write M bytes of data to be sent to TX RAM of $I2C_{master}$ in FIFO or non-FIFO mode.

13. Write 1 to I2C_TRANS_START bit to start transfer and repeat step 9.
14. If the command is a STOP, I2C stops transfer and generates an I2C_TRANS_COMPLETE_INT interrupt.
15. If the command is an END, repeat step 10.
16. Update $I2C_{master}$'s command registers.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND1	STOP	—	—	—	—

17. Write 1 to I2C_TRANS_START bit to start transfer.
18. $I2C_{master}$ executes the STOP command and generates an I2C_TRANS_COMPLETE_INT interrupt.

21.5.5 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in One Command Sequence

21.5.5.1 Introduction

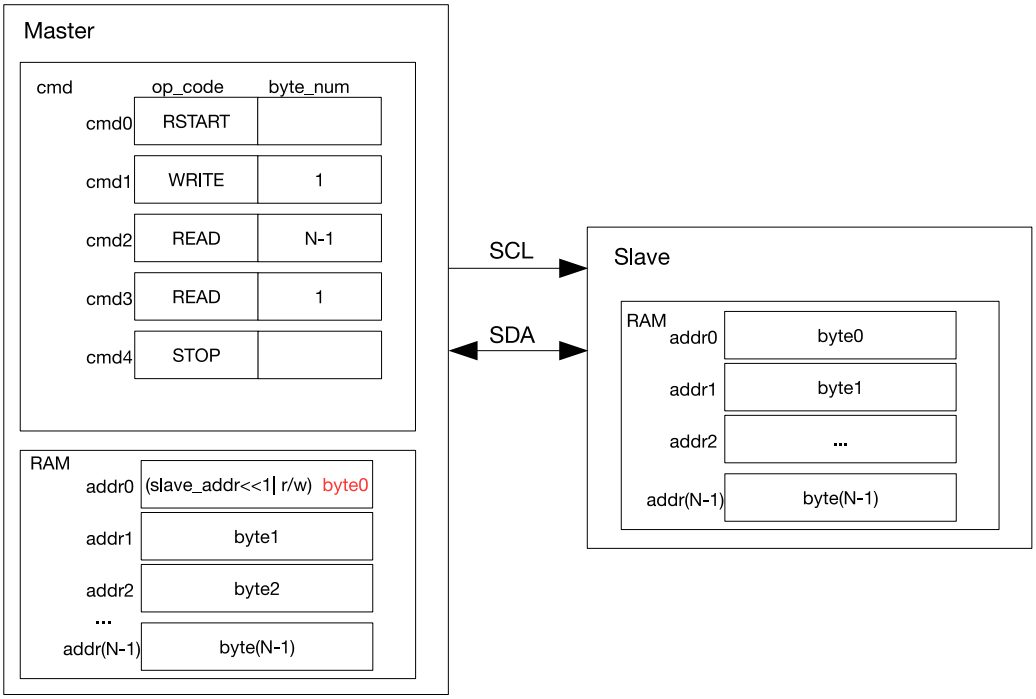


Figure 21-10. $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address

Figure 21-10 shows how $I2C_{master}$ reads N bytes of data from an I2C slave's registers or RAM using 7-bit addressing. cmd1 is a WRITE command, and when this command is executed, $I2C_{master}$ sends $I2C_{slave}$ address. The byte sent comprises a 7-bit $I2C_{slave}$ address and a $R/\overline{W}$ bit. When the $R/\overline{W}$ bit is 1, it indicates a READ operation. If the address of an I2C slave matches the sent address, this matching slave starts sending data to $I2C_{master}$. $I2C_{master}$ generates acknowledgements according to ack_value defined in the READ command upon receiving a byte.

As illustrated in Figure 21-10, $I2C_{master}$ executes two READ commands: it generates ACKs for (N-1) bytes of data in cmd2, and a NACK for the last byte of data in cmd3. This configuration may be changed as required. $I2C_{master}$ writes received data into the controller RAM from addr0, whose original content (a $I2C_{slave}$ address and a $R/\overline{W}$ bit) is overwritten by byte0 marked red in Figure 21-10.

21.5.5.2 Configuration Example

1. Set I2C_MS_MODE to 1.
2. Write 1 to I2C_CONF_UPGATE to synchronize registers.
3. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	0	0	1	1
I2C_COMMAND2	READ	0	0	1	N-1
I2C_COMMAND3	READ	1	0	1	1
I2C_COMMAND4	STOP	—	—	—	—

4. Write $I2C_{slave}$ address to TX RAM of $I2C_{master}$ in either FIFO mode or non-FIFO mode according to Section 21.4.9.
5. Set the address of $I2C_{slave}$ via I2C_SLAVE_ADDR[7:0].
6. Write 1 to I2C_CONF_UPGATE to synchronize registers.
7. Write 1 to I2C_TRANS_START bit to start transfer, and enable $I2C_{slave}$.
8. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address. When ack_check_en in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as matching slave by default.
   - Match: If the received ACK value matches ack_exp (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT interrupt and stops data transfer.
9. $I2C_{slave}$ sends data, and $I2C_{master}$ sends ACK value according to ack_check_en in the READ command.
10. If data to be received (N) is larger than the depth of RX FIFO, RX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 21.4.9.
11. After $I2C_{master}$ has received the last byte of data, set ack_value to 1. $I2C_{slave}$ will stop transfer once receiving the I2C_NACK_INT interrupt.
12. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT interrupt.

21.5.6 $I2C_{master}$ Reads $I2C_{slave}$ with a 10-bit Address in One Command Sequence

21.5.6.1 Introduction

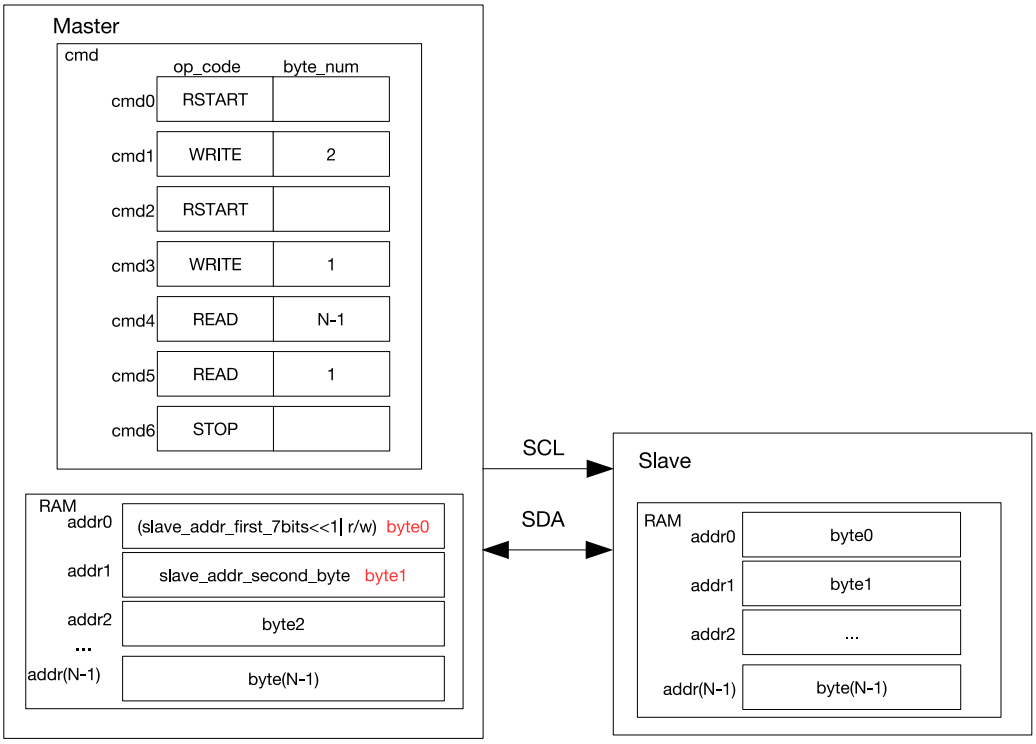


Figure 21-11. $I2C_{master}$ Reading $I2C_{slave}$ with a 10-bit Address

Figure 21-11 shows how $I2C_{master}$ reads data from an I2C slave's registers or RAM using 10-bit addressing. Unlike 7-bit addressing, in 10-bit addressing the WRITE command of the $I2C_{master}$ is formed from two bytes, and correspondingly TX RAM of this master stores a 10-bit address of two bytes. The $R/\overline{W}$ bit in the first byte is 0, which indicates a WRITE operation. After a RSTART condition, $I2C_{master}$ sends the first byte of address again to read data from $I2C_{slave}$, but the $R/\overline{W}$ bit is 1, which indicates a READ operation. The two address bytes can be configured as described in Section 21.5.2.

21.5.6.2 Configuration Example

1. Set I2C_MS_MODE to 1.
2. Write 1 to I2C_CONF_UPGATE to synchronize registers.
3. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	0	0	1	2
I2C_COMMAND2	RSTART	—	—	—	—
I2C_COMMAND3	WRITE	0	0	1	1
I2C_COMMAND4	READ	0	0	1	N-1
I2C_COMMAND5	READ	1	0	1	1
I2C_COMMAND6	STOP	—	—	—	—

4. Set the 10-bit address of $I2C_{slave}$ via I2C_SLAVE_ADDR[9:0].
5. Write $I2C_{slave}$ address and data to be sent to TX RAM of $I2C_{master}$ in either FIFO or non-FIFO mode. The first byte of address comprises ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit, which is 1 and indicates a WRITE operation. The second byte of address is I2C_SLAVE_ADDR[7:0]. The third byte is ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit, which is 1 and indicates a READ operation.
6. Write 1 to I2C_CONF_UPGATE to synchronize registers.
7. Write 1 to I2C_TRANS_START to start transfer, and enable $I2C_{slave}$.
8. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address. When ack_check_en in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as matching slave by default.
    - Match: If the received ACK value matches ack_exp (the expected ACK value), $I2C_{master}$ continues data transfer.
    - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT interrupt and stops data transfer.
9. $I2C_{master}$ sends a RSTART and the third byte in TX RAM, which is ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a $R/\overline{W}$ bit that indicates READ.
10. $I2C_{slave}$ repeats step 8. If its address matches the address sent by $I2C_{master}$, $I2C_{slave}$ proceed on to the next steps.
11. $I2C_{slave}$ sends data, and $I2C_{master}$ sends ACK value according to ack_check_en in the READ command.
12. If data to be received (N) is larger than the depth of RX FIFO, RX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 21.4.9.
13. After $I2C_{master}$ has received the last byte of data, set ack_value to 1. $I2C_{slave}$ will stop transfer once receiving the I2C_NACK_INT interrupt.
14. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT interrupt.

21.5.7 I2C$_{master}$ Reads I2C$_{slave}$ with Two 7-bit Addresses in One Command Sequence

21.5.7.1 Introduction

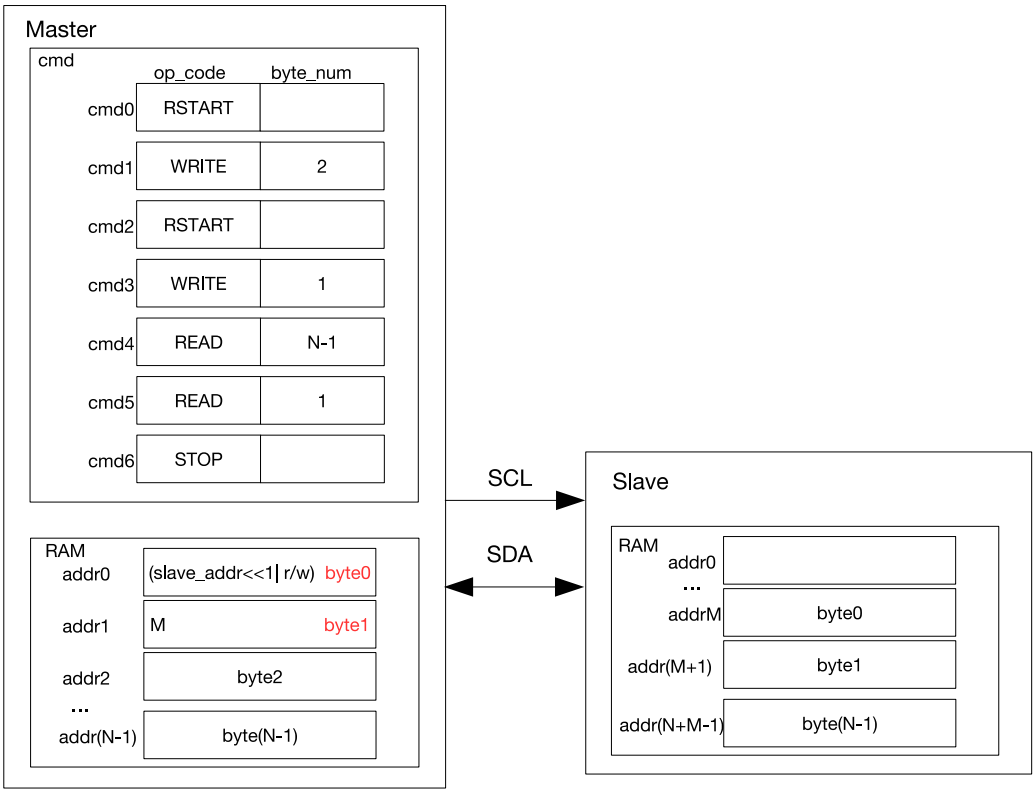


Figure 21-12. I2C$_{master}$ Reading N Bytes of Data from addrM of I2C$_{slave}$ with a 7-bit Address

Figure 21-12 shows how I2C$_{master}$ reads data from specified addresses in I2C slave's registers or RAM. I2C$_{master}$ sends two bytes of addresses: the first byte is a 7-bit I2C$_{slave}$ address followed by a $R/\overline{W}$ bit, which is 0 and indicates a WRITE; the second byte is I2C$_{slave}$'s memory address addrM. After a RSTART condition, I2C$_{master}$ sends the first byte of address again, but the $R/\overline{W}$ bit is 1 which indicates a READ. Then, I2C$_{master}$ reads data starting from addrM.

21.5.7.2 Configuration Example

1. Set I2C_MS_MODE to 1.
2. Write 1 to I2C_CONF_UPGATE to synchronize registers.
3. Choose an I2C$_{slave}$ that supports double addressing mode and enable this mode.
4. Configure command registers of I2C$_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	0	0	1	2
I2C_COMMAND2	RSTART	—	—	—	—
I2C_COMMAND3	WRITE	0	0	1	1
I2C_COMMAND4	READ	0	0	1	N-1

I2C_COMMAND5	READ	1	0	1	1
I2C_COMMAND6	STOP	—	—	—	—

5. Set the address of $I2C_{slave}$ via I2C_SLAVE_ADDR[7:0].
6. Write $I2C_{slave}$ address and data to be sent to TX RAM of $I2C_{master}$ in either FIFO or non-FIFO mode according to Section 21.4.9. The first byte of address comprises (I2C_SLAVE_ADDR[6:0])«1) and a $R/\overline{W}$ bit, which is 0 and indicates a WRITE. The second byte of address is memory address M of $I2C_{slave}$. The third byte is (I2C_SLAVE_ADDR[6:0])«1) and a $R/\overline{W}$ bit, which is 1 and indicates a READ.
7. Write 1 to I2C_CONF_UPGATE to synchronize registers.
8. Write 1 to I2C_TRANS_START to start transfer, and enable $I2C_{slave}$.
9. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address. When ack_check_en in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as matching slave by default.
   - Match: If the received ACK value matches ack_exp (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT interrupt and stops data transfer.
10. $I2C_{slave}$ receives memory address sent by $I2C_{master}$ and adds the offset.
11. $I2C_{master}$ sends a RSTART and the third byte in TX RAM, which is ((0x78 | I2C_SLAVE_ADDR[9:8])«1) and a R bit.
12. $I2C_{slave}$ repeats step 9. If its address matches the address sent by $I2C_{master}$, $I2C_{slave}$ proceed on to the next steps.
13. $I2C_{slave}$ sends data, and $I2C_{master}$ sends ACK value according to ack_check_en in the READ command.
14. If data to be received (N) is larger than the depth of RX FIFO, RX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 21.4.9.
15. After $I2C_{master}$ has received the last byte of data, set ack_value to 1. $I2C_{slave}$ will stop transfer once receiving the I2C_NACK_INT interrupt.
16. After data transfer completes, $I2C_{master}$ executes the STOP command, and generates an I2C_TRANS_COMPLETE_INT interrupt.

21.5.8 $I2C_{master}$ Reads $I2C_{slave}$ with a 7-bit Address in Multiple Command Sequences

21.5.8.1 Introduction

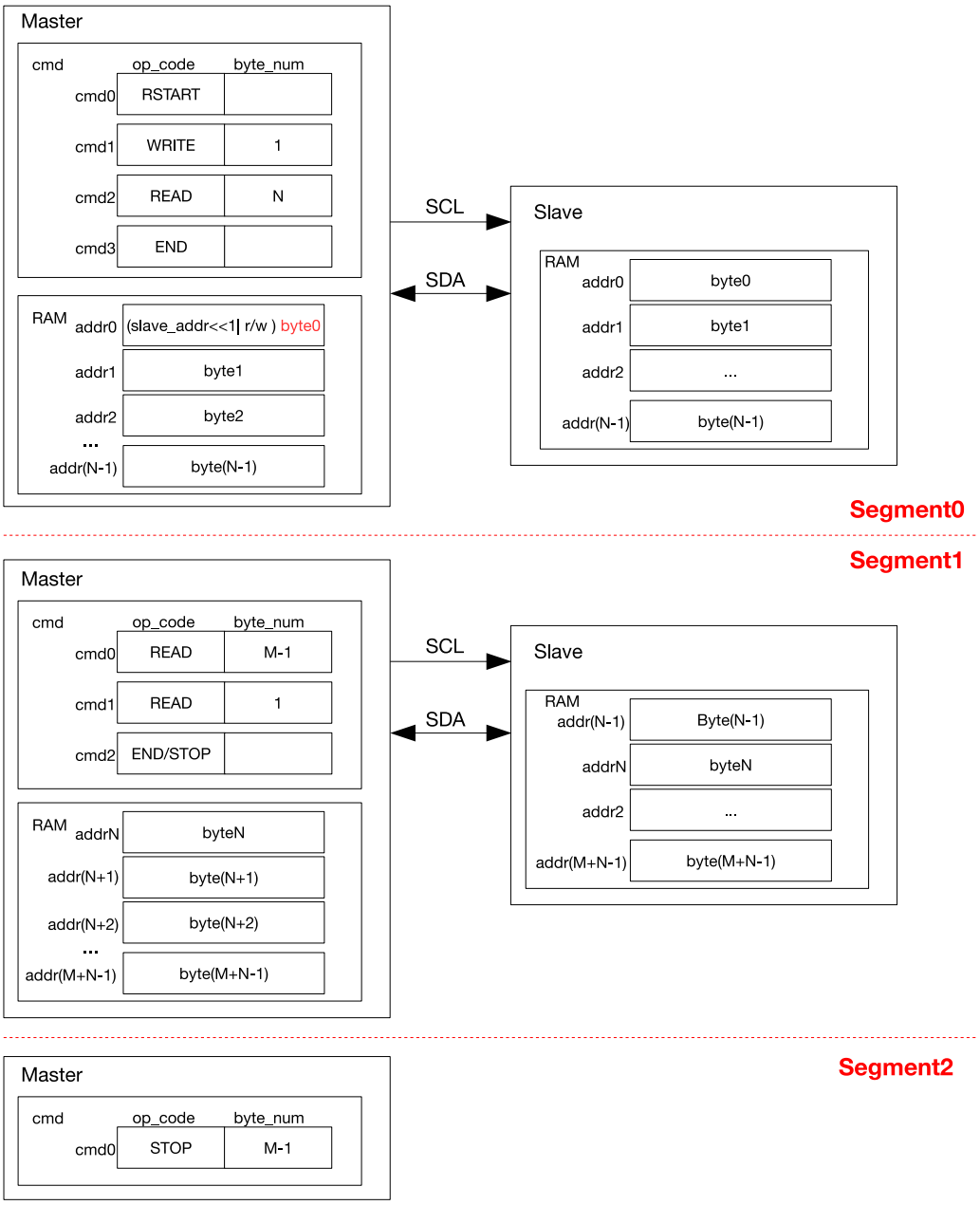


Figure 21-13. $I2C_{master}$ Reading $I2C_{slave}$ with a 7-bit Address in Segments

Figure 21-13 shows how $I2C_{master}$ reads (N+M) bytes of data from an I2C slave in two/three segments separated by END commands. Configuration procedures are described as follows:

1. The procedures for Segment0 is similar to Figure 21-10, except that the last command is an END.
2. Prepare the data that $I2C_{slave}$ will send, and set I2C_TRANS_START (master) to start data transfer. After executing the END command, $I2C_{master}$ refreshes command registers and the RAM as shown in Segment1, and clears the corresponding I2C_END_DETECT_INT interrupt. If cmd2 in Segment1 is a STOP, then data is read from $I2C_{slave}$ in two segments. $I2C_{master}$ resumes data transfer by setting I2C_TRANS_START and terminates the transfer by sending a STOP bit.

3. If cmd2 in Segment1 is an END, then data is read from $I2C_{slave}$ in three segments. After the second data transfer finishes and an I2C_END_DETECT_INT interrupt is detected, the cmd box is configured as shown in Segment2. Once I2C_TRANS_START(master) is set, $I2C_{master}$ terminates the transfer by sending a STOP bit.

21.5.8.2 Configuration Example

1. Set I2C_MS_MODE to 1.
2. Write 1 to I2C_CONF_UPGATE to synchronize registers.
3. Configure command registers of $I2C_{master}$.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	0	0	1	1
I2C_COMMAND2	READ	0	0	1	N
I2C_COMMAND3	END	—	—	—	—

4. Write $I2C_{slave}$ address to TX RAM of $I2C_{master}$ in FIFO or non-FIFO mode.
5. Set the address of $I2C_{slave}$ via I2C_SLAVE_ADDR[7:0].
6. Write 1 to I2C_CONF_UPGATE to synchronize registers.
7. Write 1 to I2C_TRANS_START to start transfer, and enable $I2C_{slave}$.
8. $I2C_{slave}$ compares the slave address sent by $I2C_{master}$ with its own address. When ack_check_en in $I2C_{master}$'s WRITE command is 1, $I2C_{master}$ checks ACK value each time it sends a byte. When ack_check_en is 0, $I2C_{master}$ does not check ACK value and take $I2C_{slave}$ as matching slave by default.
   - Match: If the received ACK value matches ack_exp (the expected ACK value), $I2C_{master}$ continues data transfer.
   - Not match: If the received ACK value does not match ack_exp, $I2C_{master}$ generates an I2C_NACK_INT interrupt and stops data transfer.
9. $I2C_{slave}$ sends data, and $I2C_{master}$ sends ACK value according to ack_check_en in the READ command.
10. If data to be received (N) is larger than the depth of RX FIFO, RX RAM of $I2C_{master}$ may wrap around in FIFO mode. For details, please refer to Section 21.4.9.
11. Once finishing reading data in the first READ command, $I2C_{master}$ executes the END command and triggers an I2C_END_DETECT_INT interrupt, which is cleared by setting I2C_END_DETECT_INT_CLR to 1.
12. Update $I2C_{master}$'s command registers using one of the following two methods:

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0	READ	ack_value	ack_exp	1	M
I2C_COMMAND1	END	—	—	—	—

Or

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num

I2C_COMMAND0	READ	0	0	1	M-1
I2C_COMMAND1	READ	1	0	1	1
I2C_COMMAND2	STOP	—	—	—	—

13. Prepare data that $I2C_{slave}$ will send.
14. Write 1 to I2C_TRANS_START bit to start transfer and repeat step 9.
15. If the last command is a STOP, then set ack_value to 1 after $I2C_{master}$ has received the last byte of data. $I2C_{slave}$ stops transfer upon the I2C_NACK_INT interrupt. $I2C_{master}$ executes the STOP command to stop transfer and generates an I2C_TRANS_COMPLETE_INT interrupt.
16. If the last command is an END, then repeat step 11 and proceed on to the next steps.
17. Update $I2C_{master}$'s command registers.

Command registers	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND1	STOP	—	—	—	—

18. Write 1 to I2C_TRANS_START bit to start transfer.
19. $I2C_{master}$ executes the STOP command to stop transfer, and generates an I2C_TRANS_COMPLETE_INT interrupt.

21.6 Interrupts

- I2C_DET_START_INT: Triggered when the master or the slave detects a START bit.
- I2C_SCL_MAIN_ST_TO_INT: Triggered when the main state machine SCL_MAIN_FSM remains unchanged for over I2C_SCL_MAIN_ST_TO_I2C[23:0] clock cycles.
- I2C_SCL_ST_TO_INT: Triggered when the state machine SCL_FSM remains unchanged for over I2C_SCL_ST_TO_I2C[23:0] clock cycles.
- I2C_RXFIFO_UDF_INT: Triggered when the I2C master controller reads RX FIFO via the APB bus, but RX FIFO is empty.
- I2C_TXFIFO_OVF_INT: Triggered when the I2C master controller writes TX FIFO via the APB bus, but TX FIFO is full.
- I2C_NACK_INT: Triggered when the ACK value received by the master is not as expected, or when the ACK value received by the slave is 1.
- I2C_TRANS_START_INT: Triggered when the I2C master controller sends a START bit.
- I2C_TIME_OUT_INT: Triggered when SCL stays high or low for more than I2C_TIME_OUT_VALUE clock cycles during data transfer.
- I2C_TRANS_COMPLETE_INT: Triggered when the I2C master controller detects a STOP bit.
- I2C_MST_TXFIFO_UDF_INT: Triggered when TX FIFO of the master underflows.
- I2C_ARBITRATION_LOST_INT: Triggered when the SDA's output value does not match its input value while the master's SCL is high.

- I2C_BYTE_TRANS_DONE_INT: Triggered when the I2C master controller sends or receives a byte.
- I2C_END_DETECT_INT: Triggered when op_code of the master indicates an END command and an END condition is detected.
- I2C_RXFIFO_OVF_INT: Triggered when RX FIFO of the I2C master controller overflows.
- I2C_TXFIFO_WM_INT: I2C TX FIFO watermark interrupt. Triggered when I2C_FIFO_PRT_EN is 1 and the pointers of TX FIFO are less than I2C_TXFIFO_WM_THRHD[4:0].
- I2C_RXFIFO_WM_INT: I2C RX FIFO watermark interrupt. Triggered when I2C_FIFO_PRT_EN is 1 and the pointers of RX FIFO are greater than I2C_RXFIFO_WM_THRHD[4:0].

21.7 Register Summary

The addresses in this section are relative to I2C Master controller base address provided in Table 3-3 in Chapter 3 *System and Memory*.

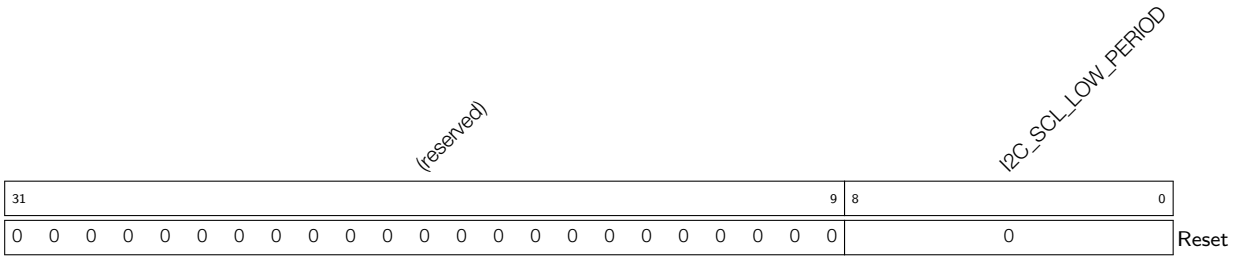
Name	Description	Address	Access
Timing registers			
I2C_SCL_LOW_PERIOD_REG	Configures the low level width of SCL	0x0000	R/W
I2C_SDA_HOLD_REG	Configures the hold time after a falling SCL edge	0x0030	R/W
I2C_SDA_SAMPLE_REG	Configures the sample time after a rising SCL edge	0x0034	R/W
I2C_SCL_HIGH_PERIOD_REG	Configures the high level width of SCL	0x0038	R/W
I2C_SCL_START_HOLD_REG	Configures the delay between the SDA and SCL falling edge for a START condition	0x0040	R/W
I2C_SCL_RSTART_SETUP_REG	Configures the delay between the rising edge of SCL and the falling edge of SDA	0x0044	R/W
I2C_SCL_STOP_HOLD_REG	Configures the delay after the SCL clock edge for a STOP condition	0x0048	R/W
I2C_SCL_STOP_SETUP_REG	Configures the delay between the SDA and SCL rising edge for a STOP condition	0x004C	R/W
I2C_SCL_ST_TIME_OUT_REG	SCL status timeout register	0x0078	R/W
I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL main status timeout register	0x007C	R/W
Configuration registers			
I2C_CTR_REG	Transmission configuration register	0x0004	varies
I2C_TO_REG	Timeout control register	0x000C	R/W
I2C_FIFO_CONF_REG	FIFO configuration register	0x0018	R/W
I2C_FILTER_CFG_REG	SCL and SDA filter configuration register	0x0050	R/W
I2C_CLK_CONF_REG	I2C clock configuration register	0x0054	R/W
I2C_SCL_SP_CONF_REG	Power configuration register	0x0080	varies
Status registers			
I2C_SR_REG	Describes I2C work status	0x0008	RO
I2C_FIFO_ST_REG	FIFO status register	0x0014	RO
I2C_DATA_REG	Read/write FIFO register	0x001C	R/W
Interrupt registers			
I2C_INT_RAW_REG	Raw interrupt status	0x0020	R/SS/WTC
I2C_INT_CLR_REG	Interrupt clear bits	0x0024	WT
I2C_INT_ENA_REG	Interrupt enable bits	0x0028	R/W
I2C_INT_STATUS_REG	Status of captured I2C communication events	0x002C	RO
Command registers			
I2C_COMD0_REG	I2C command register 0	0x0058	varies
I2C_COMD1_REG	I2C command register 1	0x005C	varies
I2C_COMD2_REG	I2C command register 2	0x0060	varies
I2C_COMD3_REG	I2C command register 3	0x0064	varies
I2C_COMD4_REG	I2C command register 4	0x0068	varies

Name	Description	Address	Access
I2C_COMD5_REG	I2C command register 5	0x006C	varies
I2C_COMD6_REG	I2C command register 6	0x0070	varies
I2C_COMD7_REG	I2C command register 7	0x0074	varies
Address registers			
I2C_TXFIFO_START_ADDR_REG	I2C TX FIFO base address register	0x0100	HRO
I2C_RXFIFO_START_ADDR_REG	I2C RX FIFO base address register	0x0180	HRO
Version register			
I2C_DATE_REG	Version control register	0x00F8	R/W

21.8 Registers

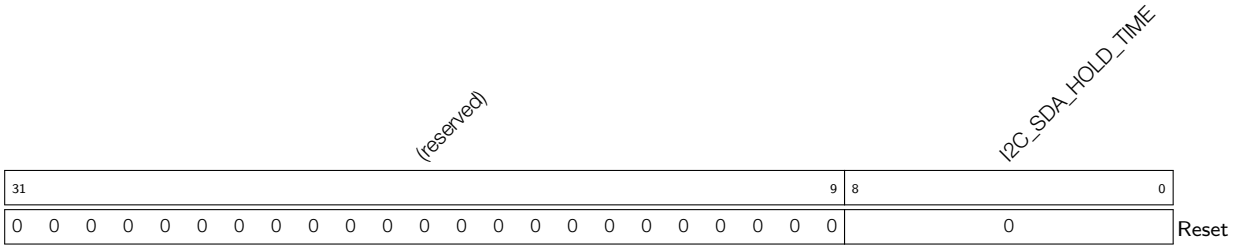
The addresses in this section are relative to I2C Master controller base address provided in Table 3-3 in Chapter 3 *System and Memory*.

Register 21.1. I2C_SCL_LOW_PERIOD_REG (0x0000)



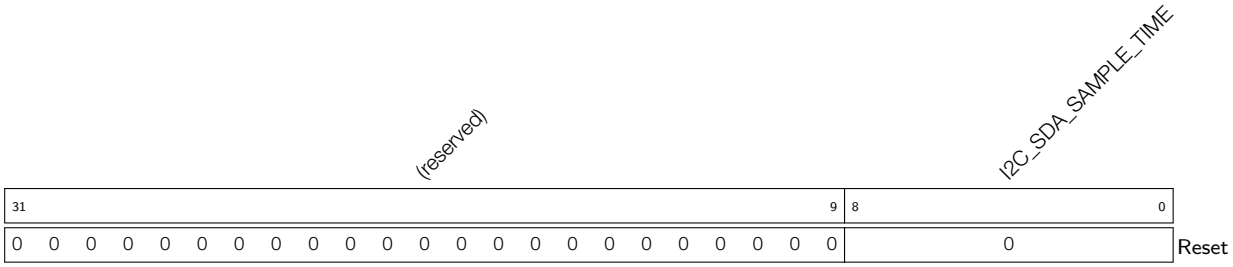
I2C_SCL_LOW_PERIOD This field is used to configure how long SCL remains low, in I2C module clock cycles. (R/W)

Register 21.2. I2C_SDA_HOLD_REG (0x0030)



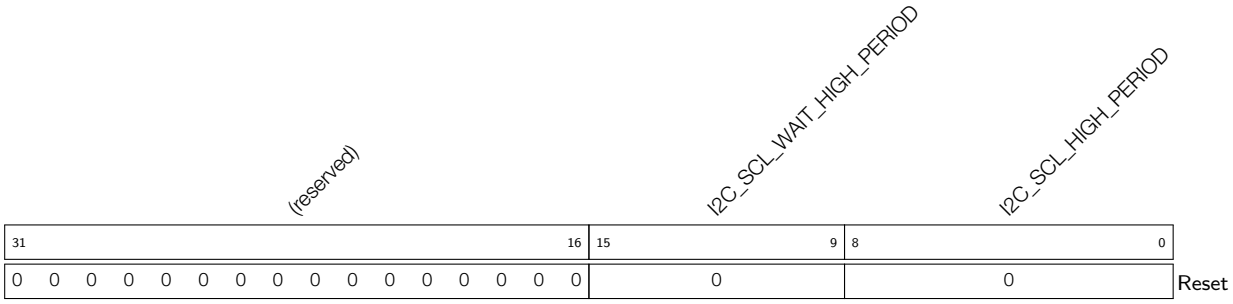
I2C_SDA_HOLD_TIME This field is used to configure the time to hold the data after the falling edge of SCL, in I2C module clock cycles. (R/W)

Register 21.3. I2C_SDA_SAMPLE_REG (0x0034)



I2C_SDA_SAMPLE_TIME This field is used to configure how long SDA is sampled, in I2C module clock cycles. (R/W)

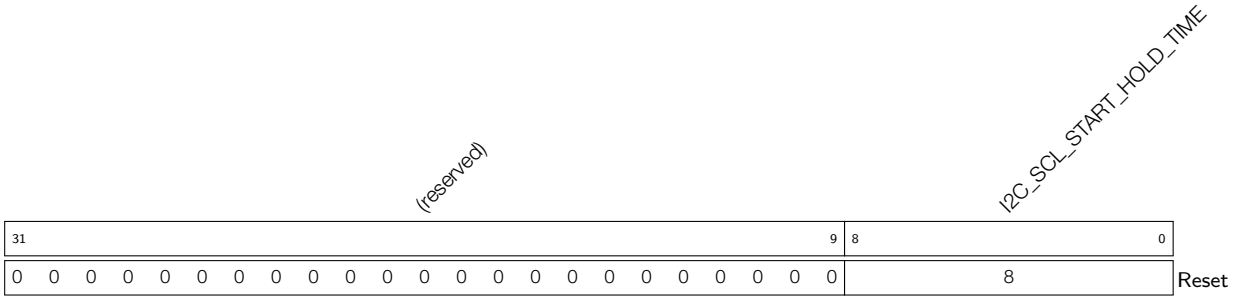
Register 21.4. I2C_SCL_HIGH_PERIOD_REG (0x0038)



I2C_SCL_HIGH_PERIOD This field is used to configure how long SCL remains high, in I2C module clock cycles. (R/W)

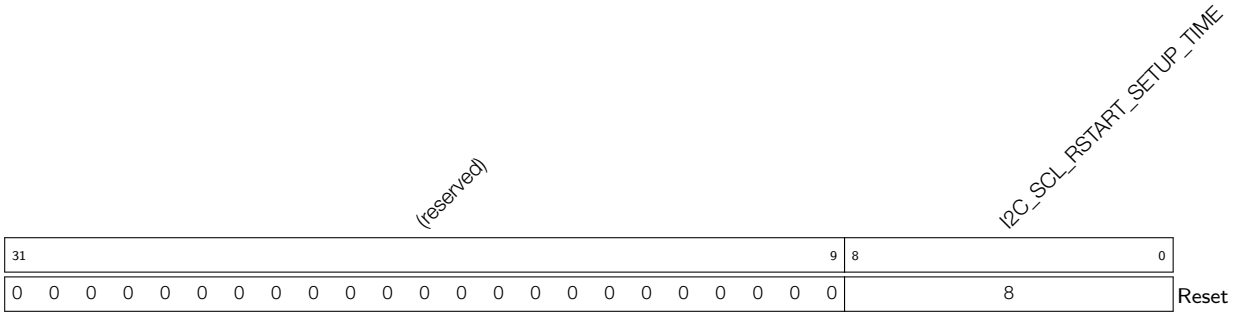
I2C_SCL_WAIT_HIGH_PERIOD This field is used to configure the SCL_FSM's waiting period for SCL high level, in I2C module clock cycles. (R/W)

Register 21.5. I2C_SCL_START_HOLD_REG (0x0040)



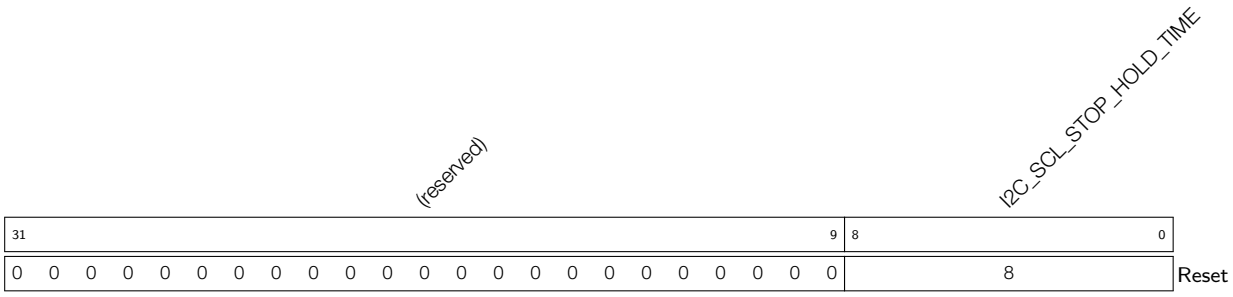
I2C_SCL_START_HOLD_TIME This field is used to configure the time between the falling edge of SDA and the falling edge of SCL for a START condition, in I2C module clock cycles. (R/W)

Register 21.6. I2C_SCL_RSTART_SETUP_REG (0x0044)



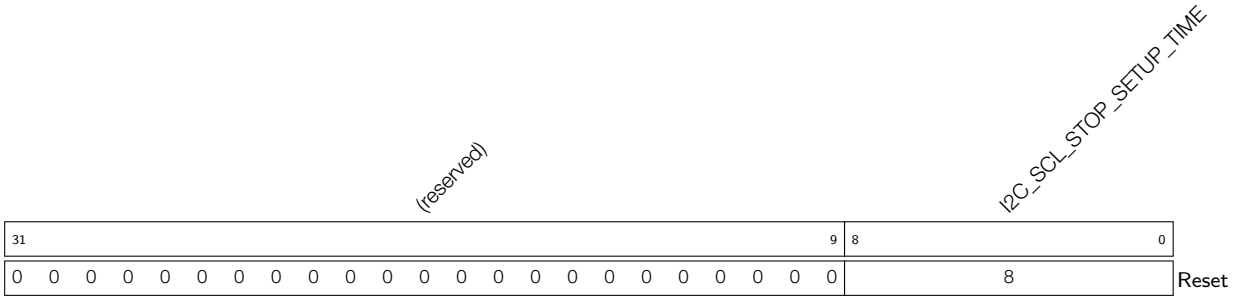
I2C_SCL_RSTART_SETUP_TIME This field is used to configure the time between the rising edge of SCL and the falling edge of SDA for a RSTART condition, in I2C module clock cycles. (R/W)

Register 21.7. I2C_SCL_STOP_HOLD_REG (0x0048)



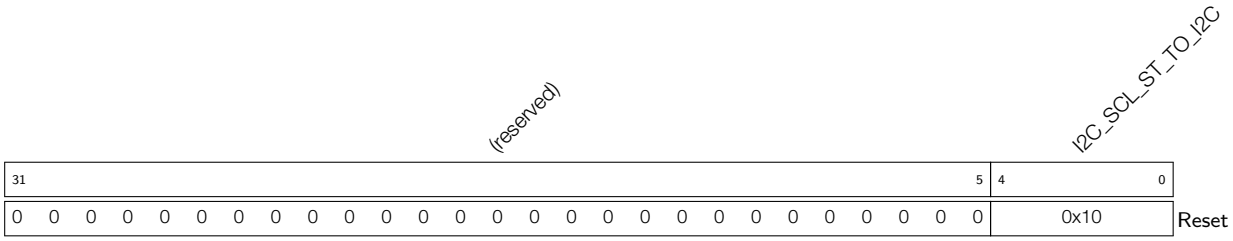
I2C_SCL_STOP_HOLD_TIME This field is used to configure the delay after the STOP condition, in I2C module clock cycles. (R/W)

Register 21.8. I2C_SCL_STOP_SETUP_REG (0x004C)



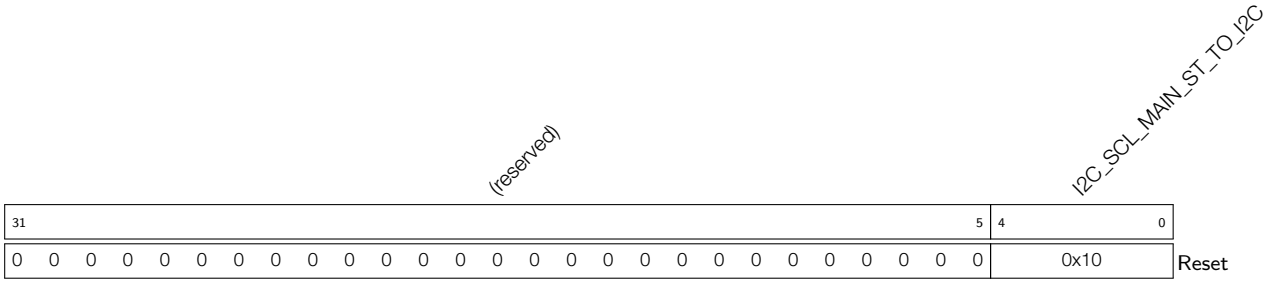
I2C_SCL_STOP_SETUP_TIME This field is used to configure the time between the rising edge of SCL and the rising edge of SDA, in I2C module clock cycles. (R/W)

Register 21.9. I2C_SCL_ST_TIME_OUT_REG (0x0078)



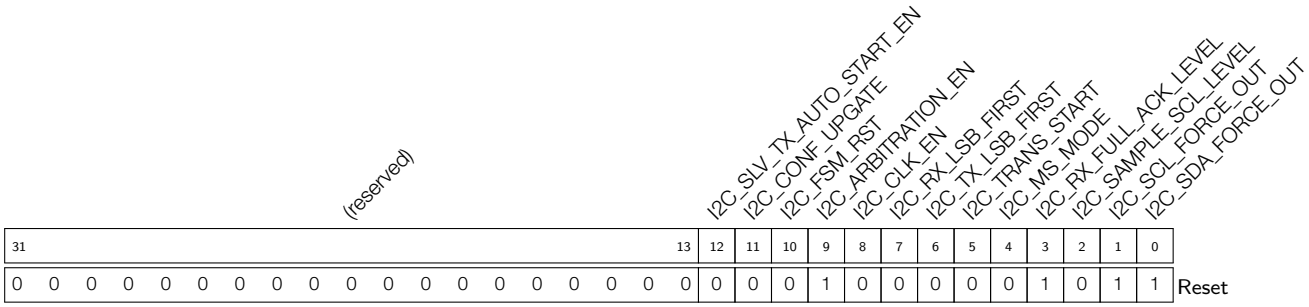
I2C_SCL_ST_TO_I2C The maximum time that SCL_FSM remains unchanged. It should be no more than 23. (R/W)

Register 21.10. I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)



I2C_SCL_MAIN_ST_TO_I2C The maximum time that SCL_MAIN_FSM remains unchanged. It should be no more than 23. (R/W)

Register 21.11. I2C_CTR_REG (0x0004)



I2C_SDA_FORCE_OUT 0: direct output; 1: open-drain output. (R/W)

I2C_SCL_FORCE_OUT 0: direct output; 1: open-drain output. (R/W)

I2C_SAMPLE_SCL_LEVEL This bit is used to select the sampling mode. 0: samples SDA data on the SCL high level; 1: samples SDA data on the SCL low level. (R/W)

I2C_RX_FULL_ACK_LEVEL This bit is used to configure the ACK value that need to be sent by master when I2C_RXFIFO_CNT has reached the threshold. (R/W)

I2C_MS_MODE Set this bit to configure the I2C master controller as an I2C Master. Clear this bit to make the I2C master controller non-operational. (R/W)

I2C_TRANS_START Set this bit to start sending the data in TX FIFO. (WT)

I2C_TX_LSB_FIRST This bit is used to control the order to send data. 0: sends data from the most significant bit; 1: sends data from the least significant bit. (R/W)

I2C_RX_LSB_FIRST This bit is used to control the order to receive data. 0: receives data from the most significant bit; 1: receives data from the least significant bit. (R/W)

I2C_CLK_EN Reserved. (R/W)

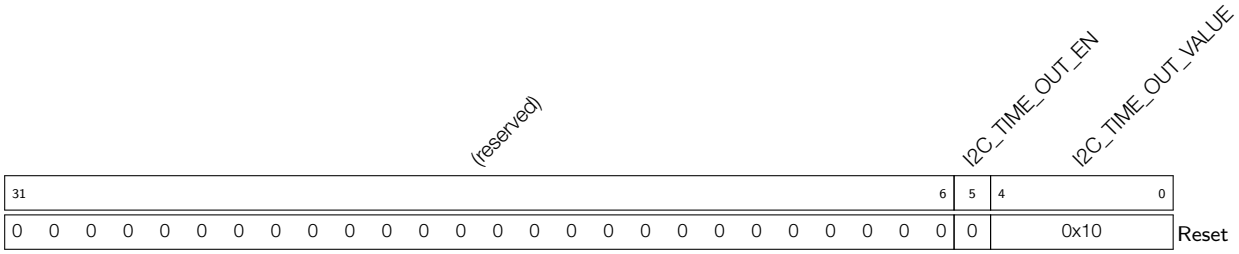
I2C_ARBITRATION_EN This is the enable bit for I2C bus arbitration function. (R/W)

I2C_FSM_RST This bit is used to reset the SCL_FSM. (WT)

I2C_CONF_UPGATE Synchronization bit. (WT)

I2C_SLV_TX_AUTO_START_EN This is the enable bit for slave to send data automatically. (R/W)

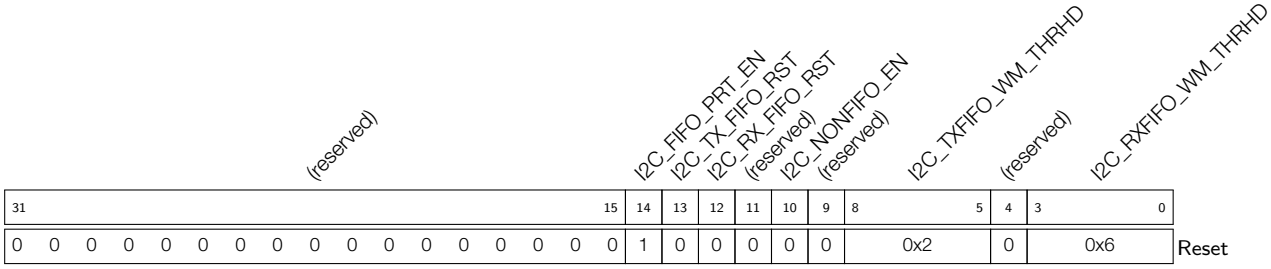
Register 21.12. I2C_TO_REG (0x000C)



I2C_TIME_OUT_VALUE This field is used to configure the timeout for receiving a data bit in APB clock cycles. (R/W)

I2C_TIME_OUT_EN This is the enable bit for timeout control. (R/W)

Register 21.13. I2C_FIFO_CONF_REG (0x0018)



I2C_RXFIFO_WM_THRHD The watermark threshold of RX FIFO in non-FIFO mode. When I2C_FIFO_PRT_EN is 1 and RX FIFO counter is bigger than I2C_RXFIFO_WM_THRHD[3:0], I2C_RXFIFO_WM_INT_RAW bit is valid. (R/W)

I2C_TXFIFO_WM_THRHD The watermark threshold of TX FIFO in non-FIFO mode. When I2C_FIFO_PRT_EN is 1 and TX FIFO counter is smaller than I2C_TXFIFO_WM_THRHD[3:0], I2C_TXFIFO_WM_INT_RAW bit is valid. (R/W)

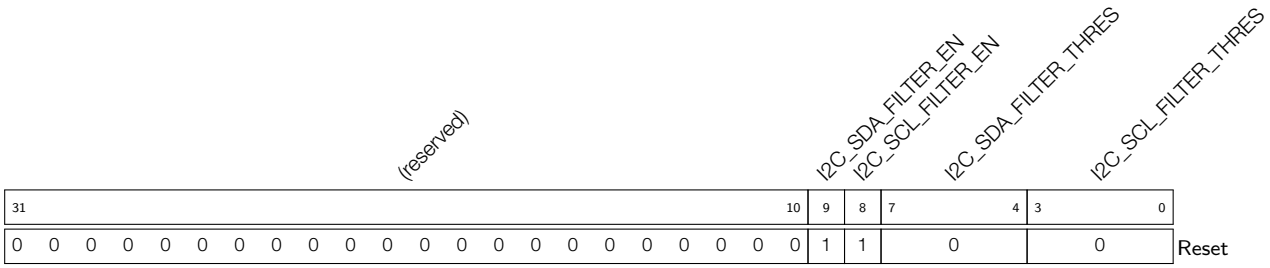
I2C_NONFIFO_EN Set this bit to enable APB non-FIFO mode. (R/W)

I2C_RX_FIFO_RST Set this bit to reset RX FIFO. (R/W)

I2C_TX_FIFO_RST Set this bit to reset TX FIFO. (R/W)

I2C_FIFO_PRT_EN The control enable bit of FIFO pointer in non-FIFO mode. This bit controls the valid bits and TX/RX FIFO overflow, underflow, full and empty interrupts. (R/W)

Register 21.14. I2C_FILTER_CFG_REG (0x0050)



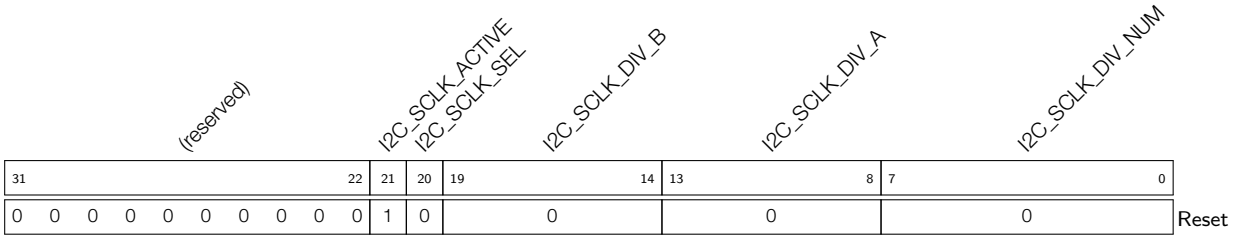
I2C_SCL_FILTER_THRES When a pulse on the SCL input has smaller width than the value of this field in I2C module clock cycles, the I2C master controller ignores that pulse. (R/W)

I2C_SDA_FILTER_THRES When a pulse on the SDA input has smaller width than the value of this field in I2C module clock cycles, the I2C master controller ignores that pulse. (R/W)

I2C_SCL_FILTER_EN This is the filter enable bit for SCL. (R/W)

I2C_SDA_FILTER_EN This is the filter enable bit for SDA. (R/W)

Register 21.15. I2C_CLK_CONF_REG (0x0054)



I2C_SCLK_DIV_NUM The integral part of the divisor. (R/W)

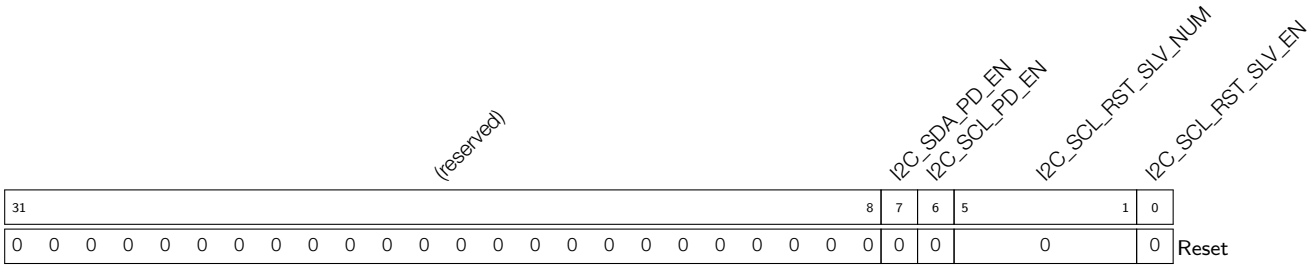
I2C_SCLK_DIV_A The numerator of the divisor's fractional part. (R/W)

I2C_SCLK_DIV_B The denominator of the divisor's fractional part. (R/W)

I2C_SCLK_SEL The clock selection bit for the I2C master controller. 0: XTAL_CLK; 1: FOSC_CLK. (R/W)

I2C_SCLK_ACTIVE The clock switch bit for the I2C master controller. (R/W)

Register 21.16. I2C_SCL_SP_CONF_REG (0x0080)



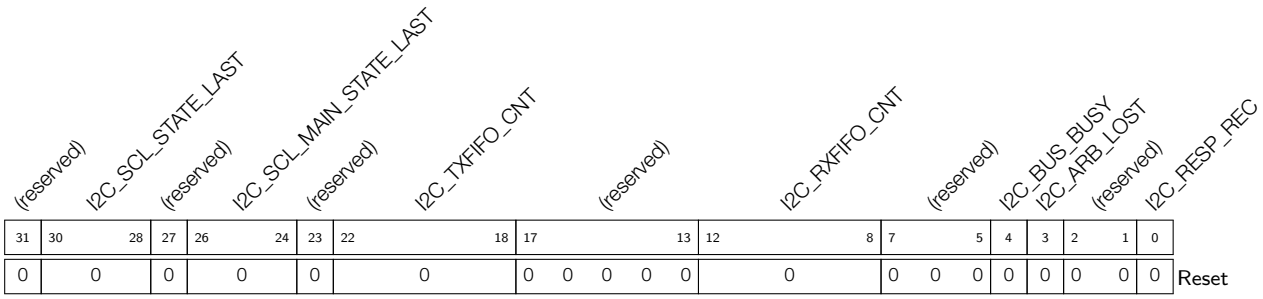
I2C_SCL_RST_SLV_EN When the master is idle, set this bit to send out SCL pulses. The number of pulses equals to I2C_SCL_RST_SLV_NUM[4:0]. (R/W/SC)

I2C_SCL_RST_SLV_NUM Configures the pulses of SCL generated. Valid when I2C_SCL_RST_SLV_EN is 1. (R/W)

I2C_SCL_PD_EN The power down enable bit for the I2C output SCL line. 0: Not power down; 1: Power down. Set I2C_SCL_FORCE_OUT and I2C_SCL_PD_EN to 1 to stretch SCL low. (R/W)

I2C_SDA_PD_EN The power down enable bit for the I2C output SDA line. 0: Not power down; 1: Power down. Set I2C_SDA_FORCE_OUT and I2C_SDA_PD_EN to 1 to stretch SDA low. (R/W)

Register 21.17. I2C_SR_REG (0x0008)



I2C_RESP_REC The received ACK value. 0: ACK; 1: NACK. (RO)

I2C_ARB_LOST When the I2C master controller loses control of the SCL line, this bit changes to 1. (RO)

I2C_BUS_BUSY 0: The I2C bus is in idle state; 1: The I2C bus is busy transferring data. (RO)

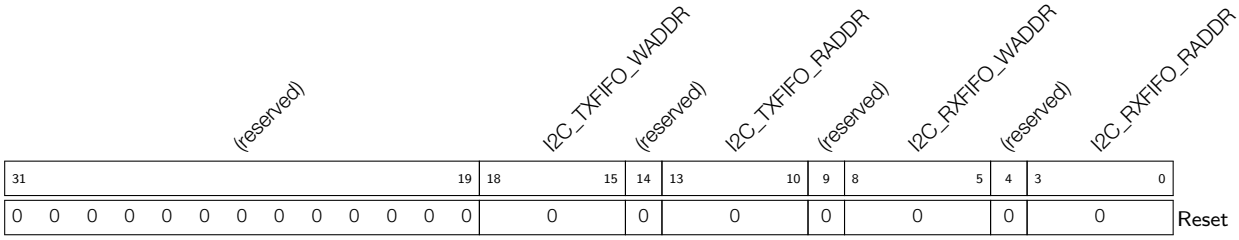
I2C_RXFIFO_CNT This field represents the number of data bytes to be sent. (RO)

I2C_TXFIFO_CNT This field stores the number of data bytes received in RAM. (RO)

I2C_SCL_MAIN_STATE_LAST This field indicates the status of the state machine. 0: idle; 1: address shift; 2: ACK address; 3: receive data; 4: transmit data; 5: send ACK; 6: wait for ACK. (RO)

I2C_SCL_STATE_LAST This field indicates the status of the state machine used to produce SCL. 0: idle; 1: start; 2: falling edge; 3: low; 4: rising edge; 5: high; 6: stop. (RO)

Register 21.18. I2C_FIFO_ST_REG (0x0014)



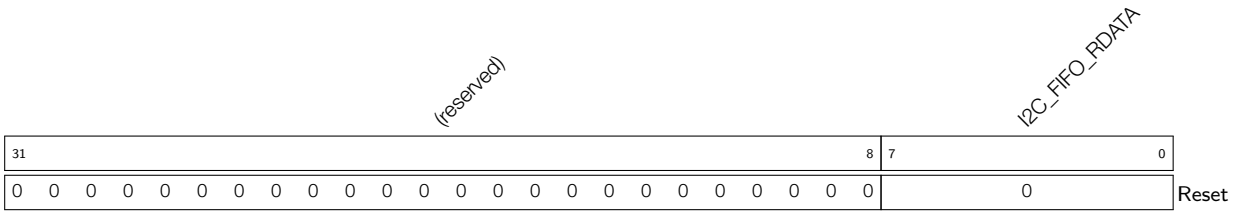
I2C_RXFIFO_RADDR This is the offset address of the APB reading from RX FIFO. (RO)

I2C_RXFIFO_WADDR This is the offset address of the I2C master controller receiving data and writing to RX FIFO. (RO)

I2C_TXFIFO_RADDR This is the offset address of the I2C master controller reading from TX FIFO. (RO)

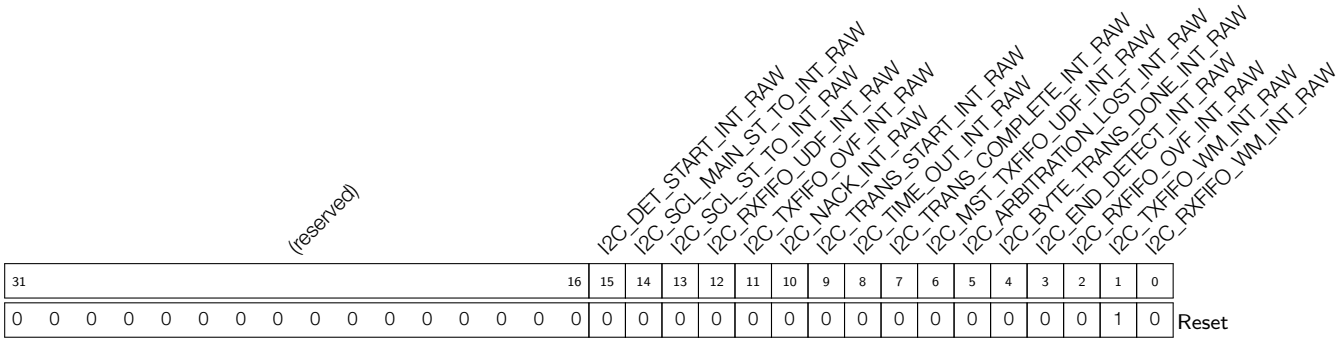
I2C_TXFIFO_WADDR This is the offset address of APB bus writing to TX FIFO. (RO)

Register 21.19. I2C_DATA_REG (0x001C)



I2C_FIFO_RDATA This field is used to read data from RX FIFO, or write data to TX FIFO. (R/W)

Register 21.20. I2C_INT_RAW_REG (0x0020)



I2C_RXFIFO_WM_INT_RAW The raw interrupt bit for the I2C_RXFIFO_WM_INT interrupt. (R/SS/WTC)

I2C_TXFIFO_WM_INT_RAW The raw interrupt bit for the I2C_TXFIFO_WM_INT interrupt. (R/SS/WTC)

I2C_RXFIFO_OVF_INT_RAW The raw interrupt bit for the I2C_RXFIFO_OVF_INT interrupt. (R/SS/WTC)

I2C_END_DETECT_INT_RAW The raw interrupt bit for the I2C_END_DETECT_INT interrupt. (R/SS/WTC)

I2C_BYTE_TRANS_DONE_INT_RAW The raw interrupt bit for the I2C_END_DETECT_INT interrupt. (R/SS/WTC)

I2C_ARBITRATION_LOST_INT_RAW The raw interrupt bit for the I2C_ARBITRATION_LOST_INT interrupt. (R/SS/WTC)

I2C_MST_TXFIFO_UDF_INT_RAW The raw interrupt bit for the I2C_TRANS_COMPLETE_INT interrupt. (R/SS/WTC)

I2C_TRANS_COMPLETE_INT_RAW The raw interrupt bit for the I2C_TRANS_COMPLETE_INT interrupt. (R/SS/WTC)

I2C_TIME_OUT_INT_RAW The raw interrupt bit for the I2C_TIME_OUT_INT interrupt. (R/SS/WTC)

I2C_TRANS_START_INT_RAW The raw interrupt bit for the I2C_TRANS_START_INT interrupt. (R/SS/WTC)

I2C_NACK_INT_RAW The raw interrupt bit for the I2C_SLAVE_STRETCH_INT interrupt. (R/SS/WTC)

I2C_TXFIFO_OVF_INT_RAW The raw interrupt bit for the I2C_TXFIFO_OVF_INT interrupt. (R/SS/WTC)

I2C_RXFIFO_UDF_INT_RAW The raw interrupt bit for the I2C_RXFIFO_UDF_INT interrupt. (R/SS/WTC)

Continued on the next page...

Register 21.20. I2C_INT_RAW_REG (0x0020)

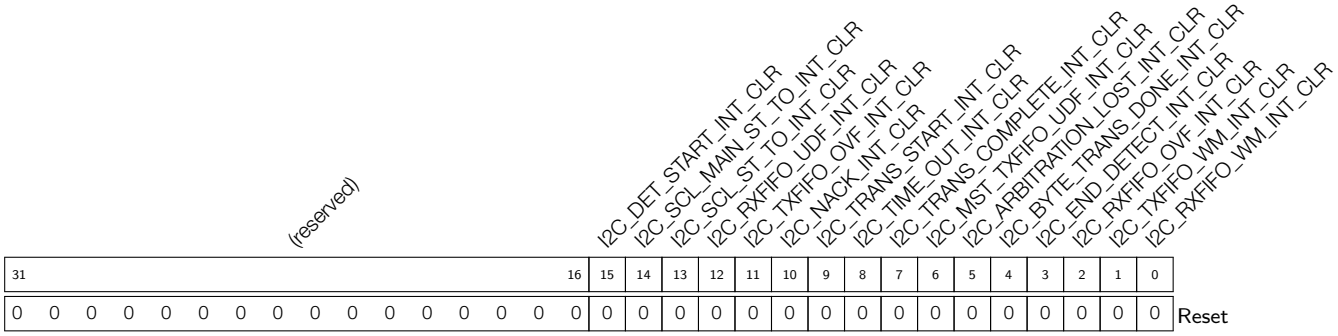
Continued from the previous page...

I2C_SCL_ST_TO_INT_RAW The raw interrupt bit for the I2C_SCL_ST_TO_INT interrupt. (R/SS/WTC)

I2C_SCL_MAIN_ST_TO_INT_RAW The raw interrupt bit for the I2C_SCL_MAIN_ST_TO_INT interrupt. (R/SS/WTC)

I2C_DET_START_INT_RAW The raw interrupt bit for the I2C_DET_START_INT interrupt. (R/SS/WTC)

Register 21.21. I2C_INT_CLR_REG (0x0024)



I2C_RXFIFO_WM_INT_CLR Set this bit to clear the I2C_RXFIFO_WM_INT interrupt. (WT)

I2C_TXFIFO_WM_INT_CLR Set this bit to clear the I2C_TXFIFO_WM_INT interrupt. (WT)

I2C_RXFIFO_OVF_INT_CLR Set this bit to clear the I2C_RXFIFO_OVF_INT interrupt. (WT)

I2C_END_DETECT_INT_CLR Set this bit to clear the I2C_END_DETECT_INT interrupt. (WT)

I2C_BYTE_TRANS_DONE_INT_CLR Set this bit to clear the I2C_END_DETECT_INT interrupt. (WT)

I2C_ARBITRATION_LOST_INT_CLR Set this bit to clear the I2C_ARBITRATION_LOST_INT interrupt. (WT)

I2C_MST_TXFIFO_UDF_INT_CLR Set this bit to clear the I2C_TRANS_COMPLETE_INT interrupt. (WT)

I2C_TRANS_COMPLETE_INT_CLR Set this bit to clear the I2C_TRANS_COMPLETE_INT interrupt. (WT)

I2C_TIME_OUT_INT_CLR Set this bit to clear the I2C_TIME_OUT_INT interrupt. (WT)

I2C_TRANS_START_INT_CLR Set this bit to clear the I2C_TRANS_START_INT interrupt. (WT)

I2C_NACK_INT_CLR Set this bit to clear the I2C_SLAVE_STRETCH_INT interrupt. (WT)

I2C_TXFIFO_OVF_INT_CLR Set this bit to clear the I2C_TXFIFO_OVF_INT interrupt. (WT)

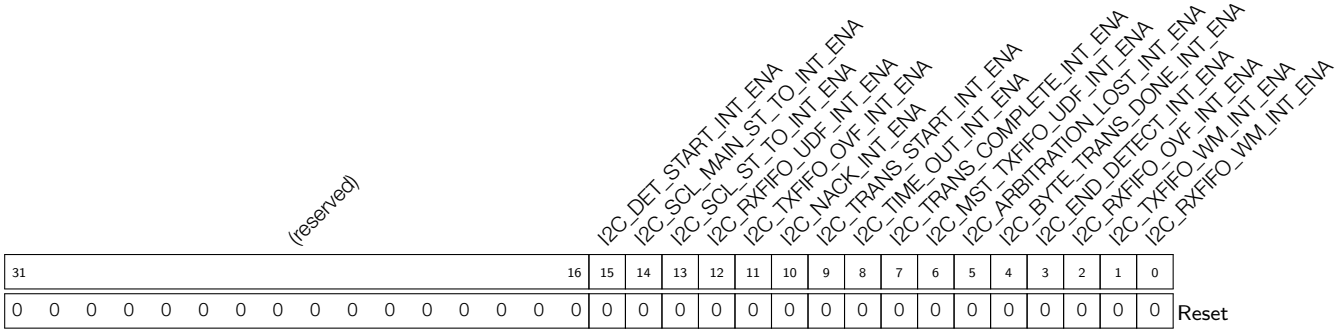
I2C_RXFIFO_UDF_INT_CLR Set this bit to clear the I2C_RXFIFO_UDF_INT interrupt. (WT)

I2C_SCL_ST_TO_INT_CLR Set this bit to clear the I2C_SCL_ST_TO_INT interrupt. (WT)

I2C_SCL_MAIN_ST_TO_INT_CLR Set this bit to clear the I2C_SCL_MAIN_ST_TO_INT interrupt. (WT)

I2C_DET_START_INT_CLR Set this bit to clear the I2C_DET_START_INT interrupt. (WT)

Register 21.22. I2C_INT_ENA_REG (0x0028)



I2C_RXFIFO_WM_INT_ENA The interrupt enable bit for the I2C_RXFIFO_WM_INT interrupt. (R/W)

I2C_TXFIFO_WM_INT_ENA The interrupt enable bit for the I2C_TXFIFO_WM_INT interrupt. (R/W)

I2C_RXFIFO_OVF_INT_ENA The interrupt enable bit for the I2C_RXFIFO_OVF_INT interrupt. (R/W)

I2C_END_DETECT_INT_ENA The interrupt enable bit for the I2C_END_DETECT_INT interrupt. (R/W)

I2C_BYTE_TRANS_DONE_INT_ENA The interrupt enable bit for the I2C_BYTE_TRANS_DONE_INT interrupt. (R/W)

I2C_ARBITRATION_LOST_INT_ENA The interrupt enable bit for the I2C_ARBITRATION_LOST_INT interrupt. (R/W)

I2C_MST_TXFIFO_UDF_INT_ENA The interrupt enable bit for the I2C_TRANS_COMPLETE_INT interrupt. (R/W)

I2C_TRANS_COMPLETE_INT_ENA The interrupt enable bit for the I2C_TRANS_COMPLETE_INT interrupt. (R/W)

I2C_TIME_OUT_INT_ENA The interrupt enable bit for the I2C_TIME_OUT_INT interrupt. (R/W)

I2C_TRANS_START_INT_ENA The interrupt enable bit for the I2C_TRANS_START_INT interrupt. (R/W)

I2C_NACK_INT_ENA The interrupt enable bit for the I2C_SLAVE_STRETCH_INT interrupt. (R/W)

I2C_TXFIFO_OVF_INT_ENA The interrupt enable bit for the I2C_TXFIFO_OVF_INT interrupt. (R/W)

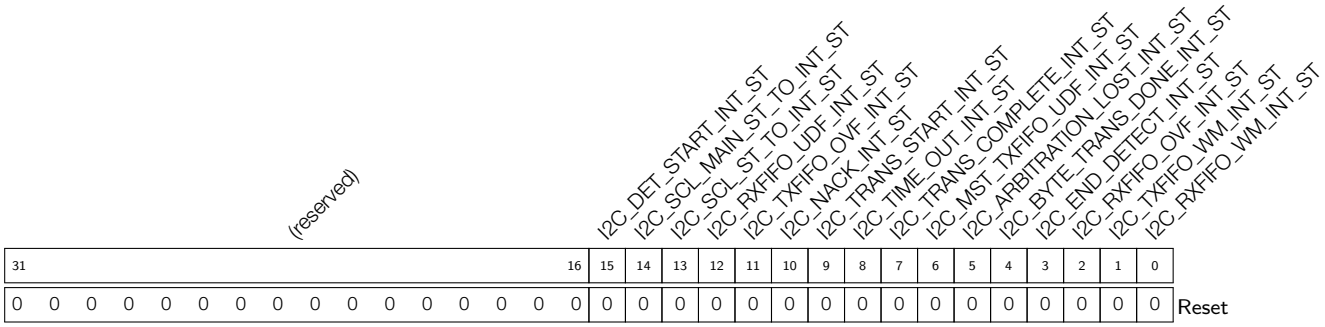
I2C_RXFIFO_UDF_INT_ENA The interrupt enable bit for the I2C_RXFIFO_UDF_INT interrupt. (R/W)

I2C_SCL_ST_TO_INT_ENA The interrupt enable bit for the I2C_SCL_ST_TO_INT interrupt. (R/W)

I2C_SCL_MAIN_ST_TO_INT_ENA The interrupt enable bit for the I2C_SCL_MAIN_ST_TO_INT interrupt. (R/W)

I2C_DET_START_INT_ENA The interrupt enable bit for the I2C_DET_START_INT interrupt. (R/W)

Register 21.23. I2C_INT_STATUS_REG (0x002C)



I2C_RXFIFO_WM_INT_ST The masked interrupt status bit for the I2C_RXFIFO_WM_INT interrupt. (RO)

I2C_TXFIFO_WM_INT_ST The masked interrupt status bit for the I2C_TXFIFO_WM_INT interrupt. (RO)

I2C_RXFIFO_OVF_INT_ST The masked interrupt status bit for the I2C_RXFIFO_OVF_INT interrupt. (RO)

I2C_END_DETECT_INT_ST The masked interrupt status bit for the I2C_END_DETECT_INT interrupt. (RO)

I2C_BYTE_TRANS_DONE_INT_ST The masked interrupt status bit for the I2C_END_DETECT_INT interrupt. (RO)

I2C_ARBITRATION_LOST_INT_ST The masked interrupt status bit for the I2C_ARBITRATION_LOST_INT interrupt. (RO)

I2C_MST_TXFIFO_UDF_INT_ST The masked interrupt status bit for the I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_TRANS_COMPLETE_INT_ST The masked interrupt status bit for the I2C_TRANS_COMPLETE_INT interrupt. (RO)

I2C_TIME_OUT_INT_ST The masked interrupt status bit for the I2C_TIME_OUT_INT interrupt. (RO)

I2C_TRANS_START_INT_ST The masked interrupt status bit for the I2C_TRANS_START_INT interrupt. (RO)

I2C_NACK_INT_ST The masked interrupt status bit for the I2C_SLAVE_STRETCH_INT interrupt. (RO)

I2C_TXFIFO_OVF_INT_ST The masked interrupt status bit for the I2C_TXFIFO_OVF_INT interrupt. (RO)

I2C_RXFIFO_UDF_INT_ST The masked interrupt status bit for the I2C_RXFIFO_UDF_INT interrupt. (RO)

Continued on the next page...

Register 21.23. I2C_INT_STATUS_REG (0x002C)

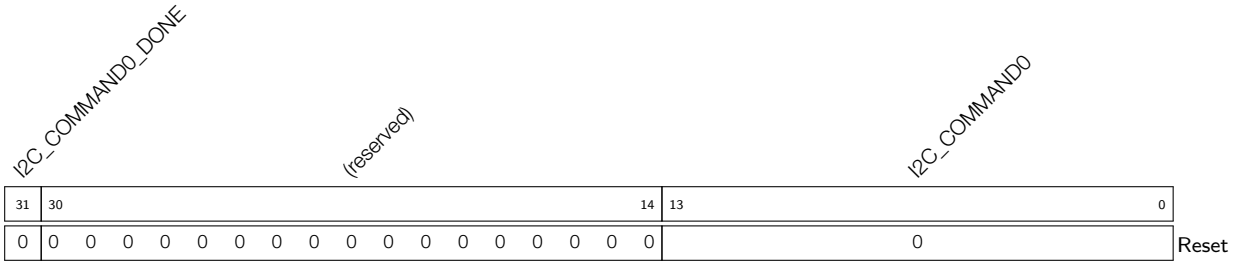
Continued from the previous page...

I2C_SCL_ST_TO_INT_ST The masked interrupt status bit for the I2C_SCL_ST_TO_INT interrupt. (RO)

I2C_SCL_MAIN_ST_TO_INT_ST The masked interrupt status bit for the I2C_SCL_MAIN_ST_TO_INT interrupt. (RO)

I2C_DET_START_INT_ST The masked interrupt status bit for the I2C_DET_START_INT interrupt. (RO)

Register 21.24. I2C_COMD0_REG (0x0058)



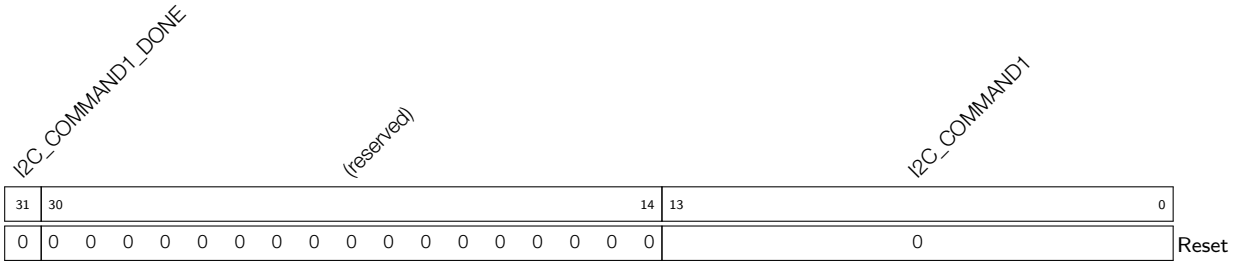
I2C_COMMAND0 This is the content of command register 0. It consists of three parts:

- op_code is the command. 0: RSTART; 1: WRITE; 2: READ; 3: STOP; 4: END.
- Byte_num represents the number of bytes that need to be sent or received.
- ack_check_en, ack_exp and ack are used to control the ACK bit. For more information, see Section 21.4.8.

(R/W)

I2C_COMMAND0_DONE When command 0 has been executed, this bit changes to high level. (R/W/SS)

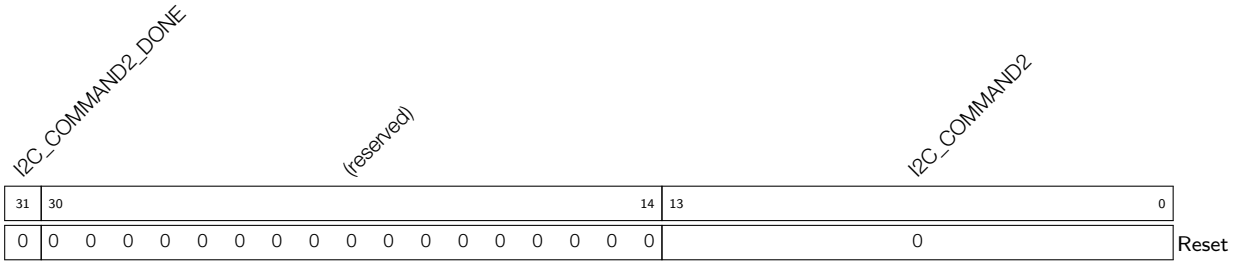
Register 21.25. I2C_COMD1_REG (0x005C)



I2C_COMMAND1 This is the content of command register 1. It is the same as that of I2C_COMMAND0. (R/W)

I2C_COMMAND1_DONE When command 1 has been executed, this bit changes to high level. (R/W/SS)

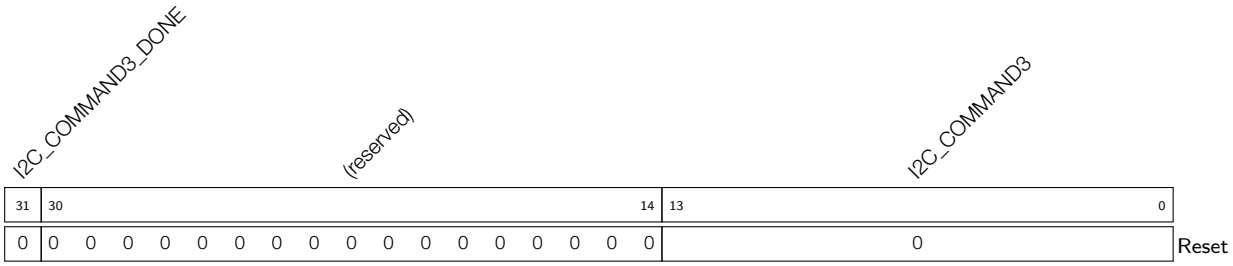
Register 21.26. I2C_COMD2_REG (0x0060)



I2C_COMMAND2 This is the content of command register 2. It is the same as that of I2C_COMMAND0. (R/W)

I2C_COMMAND2_DONE When command 2 has been executed, this bit changes to high Level. (R/W/SS)

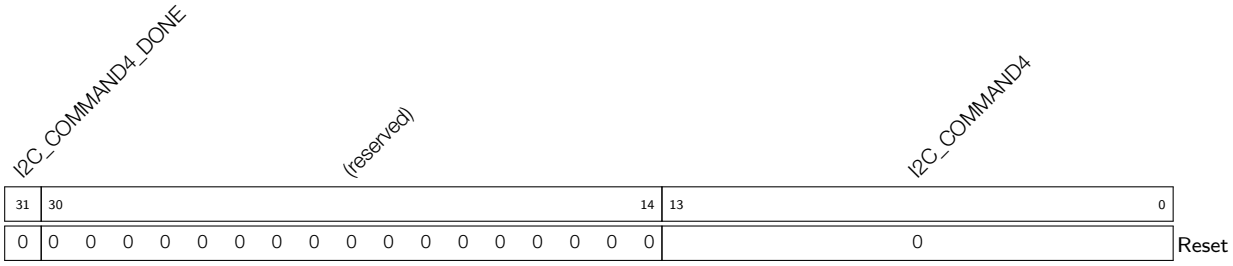
Register 21.27. I2C_COMD3_REG (0x0064)



I2C_COMMAND3 This is the content of command register 3. It is the same as that of I2C_COMMAND0. (R/W)

I2C_COMMAND3_DONE When command 3 has been executed, this bit changes to high level. (R/W/SS)

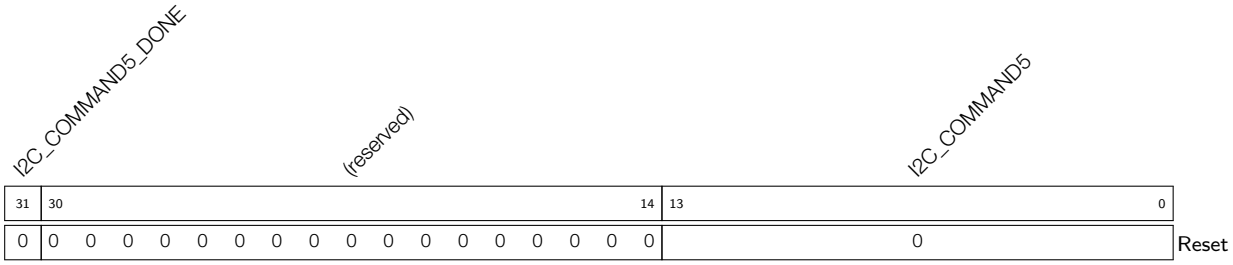
Register 21.28. I2C_COMD4_REG (0x0068)



I2C_COMMAND4 This is the content of command register 4. It is the same as that of I2C_COMMAND0. (R/W)

I2C_COMMAND4_DONE When command 4 has been executed, this bit changes to high level. (R/W/SS)

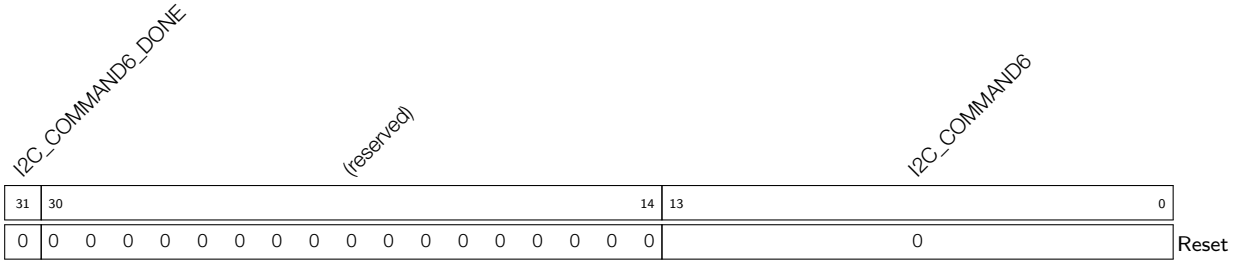
Register 21.29. I2C_COMD5_REG (0x006C)



I2C_COMMAND5 This is the content of command register 5. It is the same as that of I2C_COMMAND0. (R/W)

I2C_COMMAND5_DONE When command 5 has been executed, this bit changes to high level. (R/W/SS)

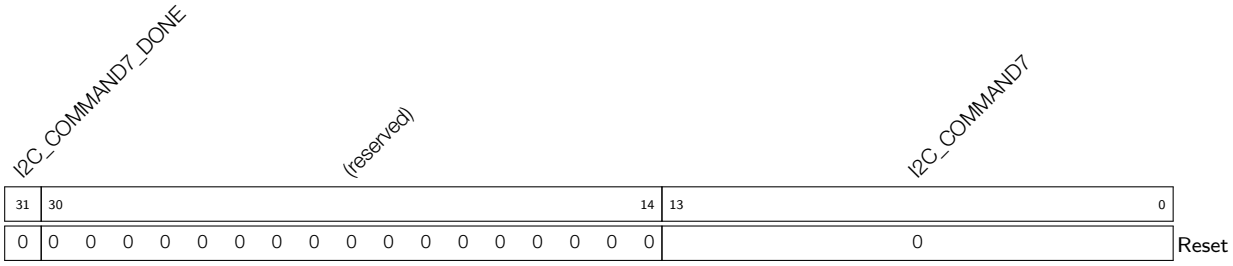
Register 21.30. I2C_COMD6_REG (0x0070)



I2C_COMMAND6 This is the content of command register 6. It is the same as that of I2C_COMMAND0. (R/W)

I2C_COMMAND6_DONE When command 6 has been executed, this bit changes to high level. (R/W/SS)

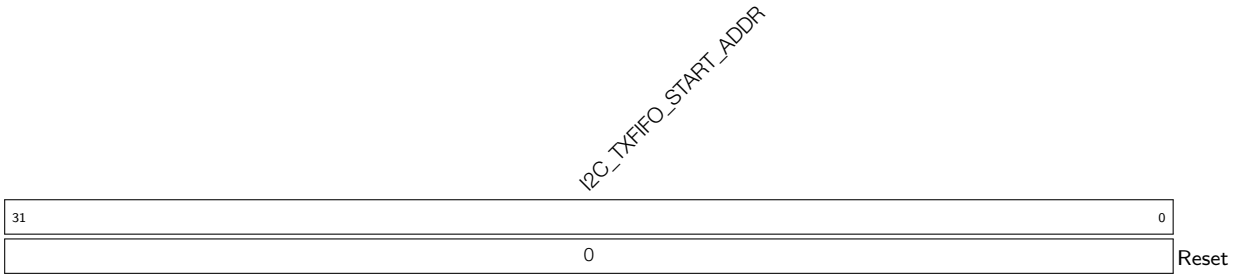
Register 21.31. I2C_COMD7_REG (0x0074)



I2C_COMMAND7 This is the content of command register 7. It is the same as that of I2C_COMMAND0. (R/W)

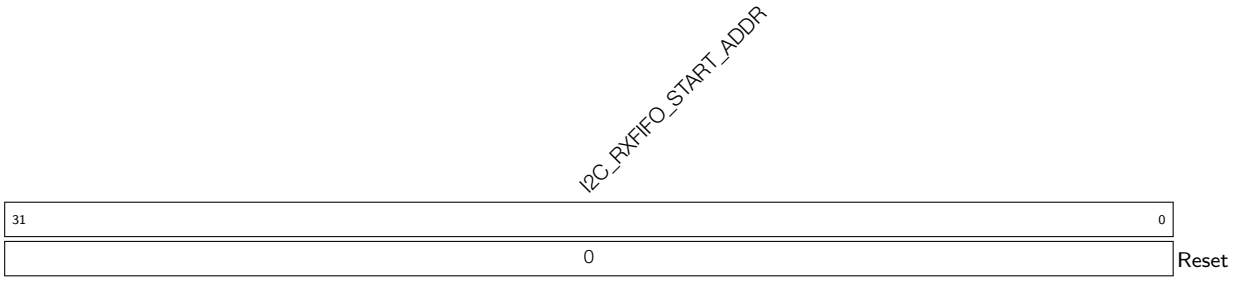
I2C_COMMAND7_DONE When command 7 has been executed, this bit changes to high level. (R/W/SS)

Register 21.32. I2C_TXFIFO_START_ADDR_REG (0x0100)



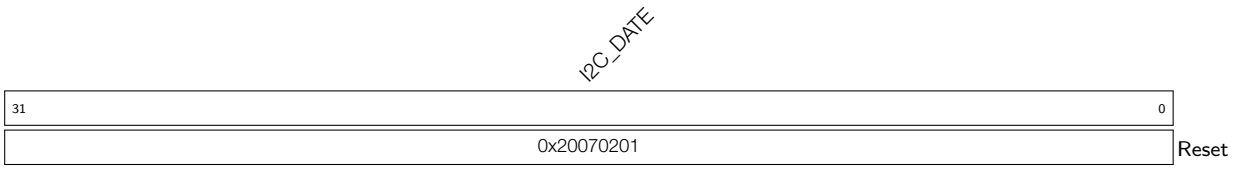
I2C_TXFIFO_START_ADDR This is the start address of I2C TX FIFO. (HRO)

Register 21.33. I2C_RXFIFO_START_ADDR_REG (0x0180)



I2C_RXFIFO_START_ADDR This is the start address of I2C RX FIFO. (HRO)

Register 21.34. I2C_DATE_REG (0x00F8)



I2C_DATE This is the version control register. (R/W)

22 LED PWM Controller (LEDC)

22.1 Overview

The LED PWM Controller is a peripheral designed to generate PWM signals for LED control. It has specialized features such as automatic duty cycle fading. However, the LED PWM Controller can also be used to generate PWM signals for other purposes.

22.2 Features

The LED PWM Controller has the following features:

- Six independent PWM generators (i.e. six channels)
- Maximum PWM duty cycle resolution: 14 bits
- Adjustable phase and duty cycle of PWM signal output
- PWM duty cycle dithering
- Automatic duty cycle fading — gradual increase/decrease of a PWM's duty cycle without interference from the processor. An interrupt will be generated upon fade completion
- PWM signal output in low-power mode (Light-sleep mode)
- Three clock sources that can be divided:
  - PLL_60M_CLK
  - FOSC_CLK
  - XTAL_CLK
- Four independent timers that support fractional division

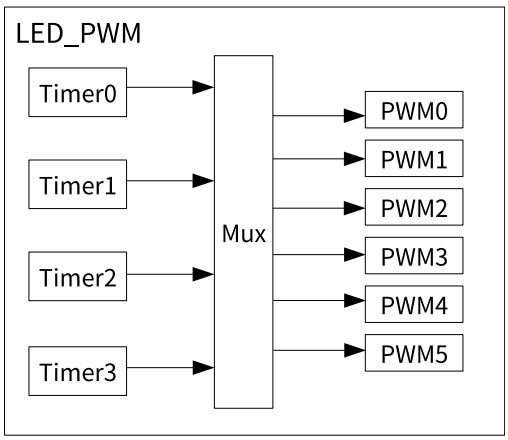


Figure 22-1. LED PWM Architecture

Note that the four timers are identical regarding their features and operation. The following sections refer to the timers collectively as Timer*x* (where *x* ranges from 0 to 3). Likewise, the six PWM generators are also identical in features and operation, and thus are collectively referred to as PWM*n* (where *n* ranges from 0 to 5).

22.3 Functional Description

22.3.1 Architecture

Figure 22-1 shows the architecture of the LED PWM Controller.

The four timers can be independently configured (i.e. each has a configurable clock divider, and counter overflow value) and each internally maintains a timebase counter (i.e. a counter that counts on cycles of a reference clock). Each PWM generator selects one of the timers and uses the timer's counter value as a reference to generate its PWM signal.

Figure 22-2 illustrates the main functional blocks of the timer and the PWM generator.

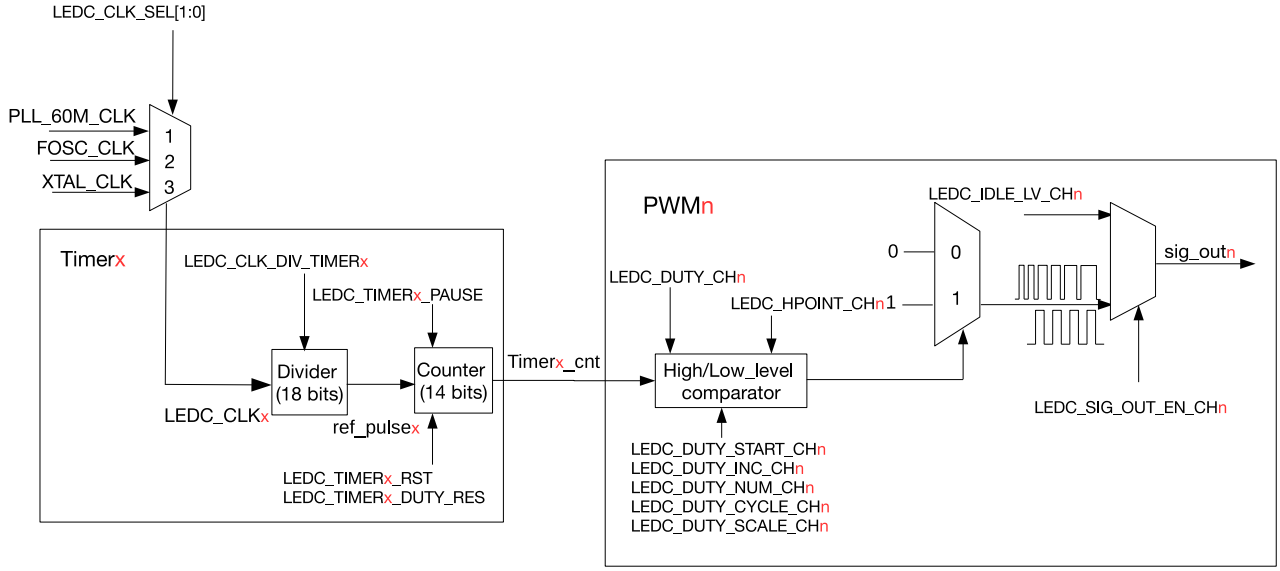


Figure 22-2. LED PWM Generator Diagram

22.3.2 Timers

Each timer in LED PWM Controller internally maintains a timebase counter. Referring to Figure 22-2, 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.

22.3.2.1 Clock Source

LED PWM registers configured by software are clocked by APB_CLK. For more information about APB_CLK, see Chapter 6 *Reset and Clock*. To use the LED PWM 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 13-1 in Chapter 13 *System Registers (SYSTEM)*.

Timers in the LED PWM Controller choose their common clock source from one of the following clock signals: PLL_60M_CLK, FOSC_CLK and XTAL_CLK (see Chapter 6 *Reset and Clock* for more details about each clock signal). The procedure for selecting a clock source signal for LEDC_CLK*x* is described below:

- PLL_60M_CLK: Set LEDC_CLK_SEL[1:0] to 1
- FOSC_CLK: Set LEDC_CLK_SEL[1:0] to 2
- XTAL_CLK: Set LEDC_CLK_SEL[1:0] to 3

The LEDC_CLK*x* signal will then be passed through the clock divider.

22.3.2.2 Clock Divider Configuration

The LEDC_CLK*x* signal is passed through a clock divider to generate the ref_pulse*x* signal for the counter. The frequency of ref_pulse*x* is equal to the frequency of LEDC_CLK*x* divided by the LEDC_CLK_DIV_TIMER*x* divider value (see Figure 22-2).

The LEDC_CLK_DIV_TIMER*x* divider value is a fractional clock divider. Thus, it supports non-integer divider values for finer granularity of available frequencies. LEDC_CLK_DIV_TIMER*x* is configured 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_TIMER*x* (i.e. LEDC_TIMER*x*_CONF_REG[21:12])
- The fractional part *B* corresponds to the least significant 8 bits of LEDC_CLK_DIV_TIMER*x* (i.e. LEDC_TIMER*x*_CONF_REG[11:4])

When the fractional part *B* is zero, LEDC_CLK_DIV_TIMER*x* is equivalent to an integer divider value (i.e. an integer prescaler). In other words, a ref_pulse*x* clock pulse is generated after every *A* number of LEDC_CLK*x* clock pulses.

However, when *B* is nonzero, LEDC_CLK_DIV_TIMER*x* becomes a non-integer divider value. The clock divider implements non-integer frequency division by alternating between *A* and (*A*+1) LEDC_CLK*x* clock pulses per ref_pulse*x* clock pulse. This will result in the average frequency of ref_pulse*x* clock pulse being the desired frequency (i.e. the non-integer divided frequency). For every 256 ref_pulse*x* clock pulses:

- a number of *B* ref_pulse*x* clock pulses will have duration of (*A*+1) LEDC_CLK*x* clock pulses
- a number of (256-*B*) ref_pulse*x* clock pulses will have duration of *A* LEDC_CLK*x* clock pulses
- the ref_pulse*x* clock pulses with duration of (*A*+1) pulses are evenly distributed amongst those with duration of *A* pulses

Figure 22-3 illustrates the relation between LEDC_CLK*x* clock pulses and ref_pulse*x* clock pulses when dividing by a non-integer LEDC_CLK_DIV_TIMER*x*.

To change the timer's clock divider value at runtime, first set the LEDC_CLK_DIV_TIMER*x* field, and then set the LEDC_TIMER*x*_PARA_UP field to apply the new configuration. This will cause the newly configured values to take effect upon the next overflow of the counter. The LEDC_TIMER*x*_PARA_UP field will be automatically cleared by hardware.

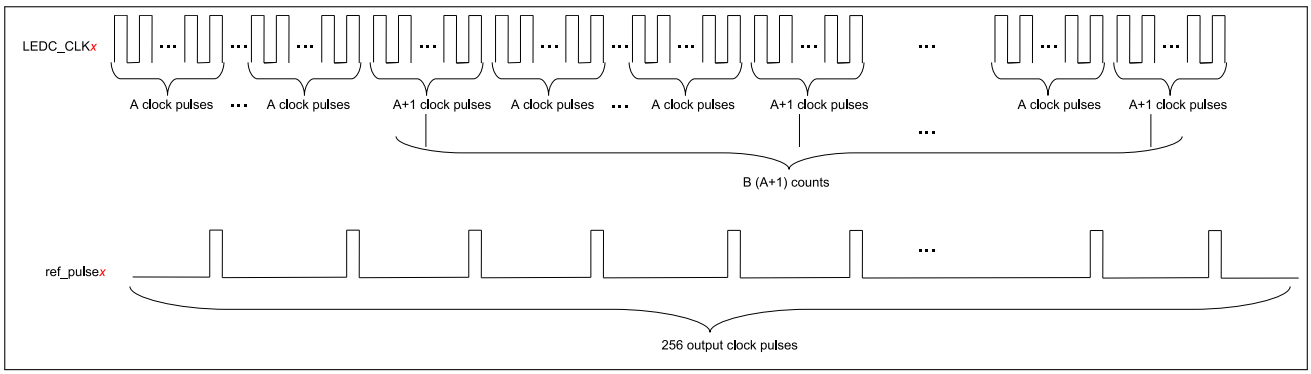


Figure 22-3. Frequency Division When LEDC_CLK_DIV_TIMERx is a Non-Integer Value

22.3.2.3 14-bit Counter

Each timer contains a 14-bit timebase counter that uses ref_pulsex as its reference clock (see Figure 22-2). The LEDC_TIMERx_DUTY_RES field configures the overflow value of this 14-bit counter. Hence, the maximum resolution of the PWM duty cycle is 14 bits. The counter counts up to ($2^{LEDC\_TIMERx\_DUTY\_RES} - 1$), overflows and begins counting from 0 again. The counter's value can be read, reset, and suspended by software.

The counter can trigger LEDC_TIMERx_OVF_INT interrupt (generated automatically by hardware without configuration) every time the counter overflows. It can also be configured to trigger LEDC_OVF_CNT_CHn_INT interrupt after the counter overflows ($LEDC\_OVF\_NUM\_CHn + 1$) times. To configure LEDC_OVF_CNT_CHn_INT interrupt, please:

1. Configure LEDC_TIMER_SEL_CHn as the counter for the PWM generator
2. Enable the counter by setting LEDC_OVF_CNT_EN_CHn
3. Set LEDC_OVF_NUM_CHn to the number of counter overflows to generate an interrupt, minus 1
4. Enable the overflow interrupt by setting LEDC_OVF_CNT_CHn_INT_ENA
5. Set LEDC_TIMERx_DUTY_RES to enable the timer and wait for a LEDC_OVF_CNT_CHn_INT interrupt

Referring to Figure 22-2, the frequency of a PWM generator output signal (sig_outn) is dependent on the frequency of the timer's clock source (LEDC_CLKx), the clock divider value (LEDC_CLK_DIV_TIMERx), and the range of the counter (LEDC_TIMERx_DUTY_RES):

$$f_{\text{PWM}} = \frac{f_{\text{LEDC\_CLKx}}}{\text{LEDC\_CLK\_DIV\_TIMERx} \cdot 2^{\text{LEDC\_TIMERx\_DUTY\_RES}}}$$

To change the overflow value at runtime, first set the LEDC_TIMERx_DUTY_RES field, and then set the LEDC_TIMERx_PARA_UP field. This will cause the newly configured values to take effect upon the next overflow of the counter. If LEDC_OVF_CNT_EN_CHn field is reconfigured, LEDC_PARA_UP_CHn should be set to apply the new configuration. In summary, these configuration values need to be updated by setting LEDC_TIMERx_PARA_UP or LEDC_PARA_UP_CHn. LEDC_TIMERx_PARA_UP and LEDC_PARA_UP_CHn will be automatically cleared by hardware.

22.3.3 PWM Generators

To generate a PWM signal, a PWM generator (PWMn) selects a timer (Timerx). Each PWM generator can be configured separately by setting LEDC_TIMER_SEL_CHn to use one of four timers to generate the PWM

output.

As shown in Figure 22-2, 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 22-4 illustrates how Hpoint*n* or Lpoint*n* are used to generate a fixed duty cycle PWM output signal.

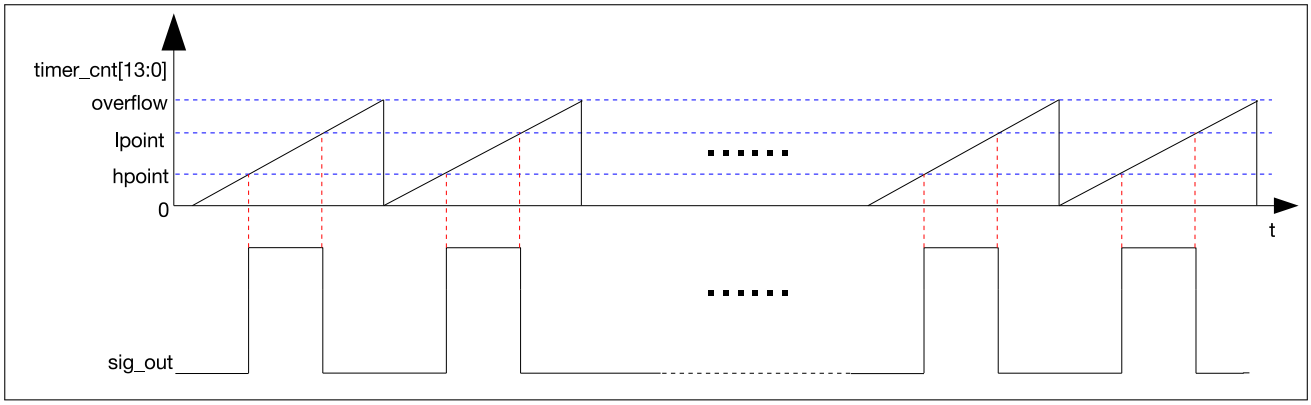


Figure 22-4. LED_PWM Output Signal Diagram

For a particular PWM generator (PWM*n*), its Hpoint*n* is sampled from the LEDC_HPOINT_CH*n* field each time the selected timer's counter overflows. Likewise, Lpoint*n* is also sampled on every counter overflow and is calculated from the sum of the LEDC_DUTY_CH*n*[18:4] and LEDC_HPOINT_CH*n* fields. By setting Hpoint*n* and Lpoint*n* via the LEDC_HPOINT_CH*n* and LEDC_DUTY_CH*n*[18:4] fields, the relative phase and duty cycle of the PWM output can be set.

The PWM output signal (sig_out*n*) is enabled by setting LEDC_SIG_OUT_EN_CH*n*. When LEDC_SIG_OUT_EN_CH*n* is cleared, PWM signal output is disabled, and the output signal (sig_out*n*) will output a constant level as specified by LEDC_IDLE_LV_CH*n*.

The bits LEDC_DUTY_CH*n*[3:0] are used to dither the duty cycles of the PWM output signal (sig_out*n*) by periodically altering the duty cycle of sig_out*n*. When LEDC_DUTY_CH*n*[3:0] is set to a non-zero value, then for every 16 cycles of sig_out*n*, LEDC_DUTY_CH*n*[3:0] of those cycles will have PWM pulses that are one timer tick longer than the other (16- LEDC_DUTY_CH*n*[3:0]) cycles. For instance, if LEDC_DUTY_CH*n*[18:4] is set to 10 and LEDC_DUTY_CH*n*[3:0] is set to 5, then 5 of 16 cycles will have a PWM pulse with a duty value of 11 and the rest of the 16 cycles will have a PWM pulse with a duty value of 10. The average duty cycle after 16 cycles is 10.3125.

If fields LEDC_TIMER_SEL_CH*n*, LEDC_HPOINT_CH*n*, LEDC_DUTY_CH*n*[18:4] and LEDC_SIG_OUT_EN_CH*n* are reconfigured, LEDC_PARA_UP_CH*n* must be set to apply the new configuration. This will cause the newly configured values to take effect upon the next overflow of the counter. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

22.3.4 Duty Cycle Fading

The PWM generators can fade the duty cycle of a PWM output signal (i.e. gradually change the duty cycle from one value to another). If Duty Cycle Fading is enabled, the value of Lpoint*n* will be incremented/decremented after a fixed number of counter overflows has occured. Figure 22-5 illustrates Duty Cycle Fading.

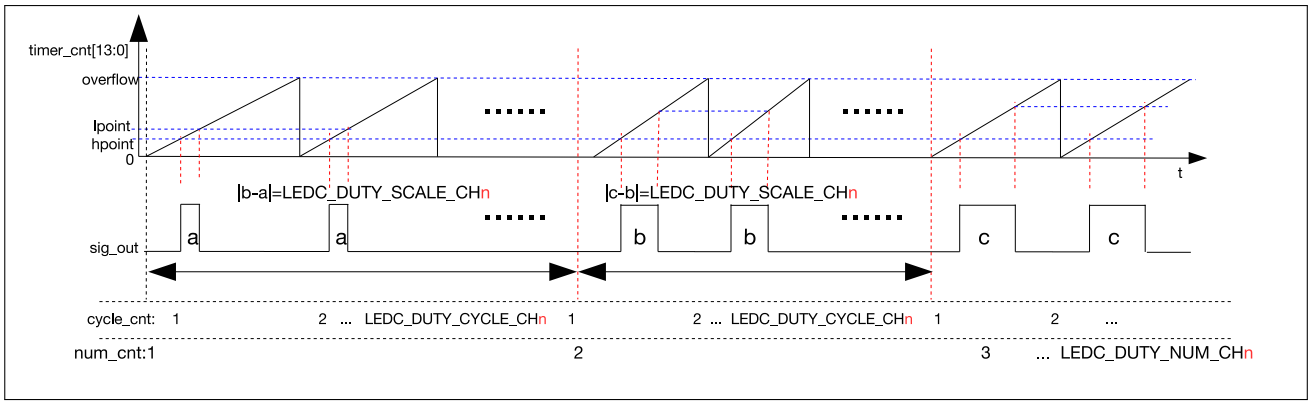


Figure 22-5. Output Signal Diagram of Fading Duty Cycle

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.

22.3.5 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.

22.4 Register Summary

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

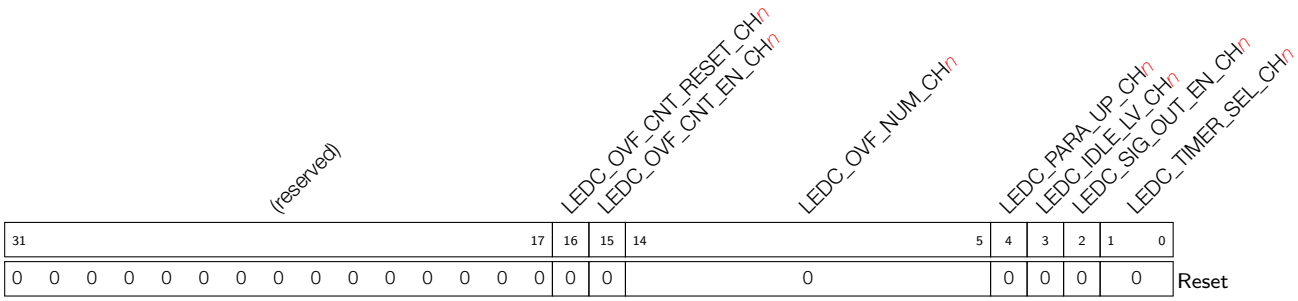
Name	Description	Address	Access
Configuration Register			
LEDC_CH0_CONF0_REG	Configuration register 0 for channel 0	0x0000	varies
LEDC_CH0_CONF1_REG	Configuration register 1 for channel 0	0x000C	varies
LEDC_CH1_CONF0_REG	Configuration register 0 for channel 1	0x0014	varies
LEDC_CH1_CONF1_REG	Configuration register 1 for channel 1	0x0020	varies
LEDC_CH2_CONF0_REG	Configuration register 0 for channel 2	0x0028	varies
LEDC_CH2_CONF1_REG	Configuration register 1 for channel 2	0x0034	varies
LEDC_CH3_CONF0_REG	Configuration register 0 for channel 3	0x003C	varies
LEDC_CH3_CONF1_REG	Configuration register 1 for channel 3	0x0048	varies
LEDC_CH4_CONF0_REG	Configuration register 0 for channel 4	0x0050	varies
LEDC_CH4_CONF1_REG	Configuration register 1 for channel 4	0x005C	varies
LEDC_CH5_CONF0_REG	Configuration register 0 for channel 5	0x0064	varies
LEDC_CH5_CONF1_REG	Configuration register 1 for channel 5	0x0070	varies
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
Duty Cycle Register			
LEDC_CH0_DUTY_REG	Initial duty cycle for channel 0	0x0008	R/W
LEDC_CH0_DUTY_R_REG	Current duty cycle for channel 0	0x0010	RO
LEDC_CH1_DUTY_REG	Initial duty cycle for channel 1	0x001C	R/W
LEDC_CH1_DUTY_R_REG	Current duty cycle for channel 1	0x0024	RO
LEDC_CH2_DUTY_REG	Initial duty cycle for channel 2	0x0030	R/W
LEDC_CH2_DUTY_R_REG	Current duty cycle for channel 2	0x0038	RO
LEDC_CH3_DUTY_REG	Initial duty cycle for channel 3	0x0044	R/W
LEDC_CH3_DUTY_R_REG	Current duty cycle for channel 3	0x004C	RO
LEDC_CH4_DUTY_REG	Initial duty cycle for channel 4	0x0058	R/W
LEDC_CH4_DUTY_R_REG	Current duty cycle for channel 4	0x0060	RO
LEDC_CH5_DUTY_REG	Initial duty cycle for channel 5	0x006C	R/W
LEDC_CH5_DUTY_R_REG	Current duty cycle for channel 5	0x0074	RO
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

Name	Description	Address	Access
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	R/WTC/SS
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	WT
Version Register			
LEDC_DATE_REG	Version control register	0x00FC	R/W

22.5 Registers

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

Register 22.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14n*)**



LEDC_TIMER_SEL_CH*n* This field is used to select one of timers for channel *n*.

0: select Timer 0; 1: select Timer 1; 2: select Timer 2; 3: select Timer 3 (R/W)

LEDC_SIG_OUT_EN_CH*n* Set this bit to enable signal output on channel *n*. (R/W)

LEDC_IDLE_LV_CH*n* This bit is used to control the output value when channel *n* is inactive (when LEDC_SIG_OUT_EN_CH*n* is 0). (R/W)

LEDC_PARA_UP_CH*n* This bit is used to update the listed fields for channel *n*, and will be automatically cleared by hardware. (WT)

- LEDC_HPOINT_CH*n*
- LEDC_DUTY_START_CH*n*
- LEDC_SIG_OUT_EN_CH*n*
- LEDC_TIMER_SEL_CH*n*
- LEDC_DUTY_NUM_CH*n*
- LEDC_DUTY_CYCLE_CH*n*
- LEDC_DUTY_SCALE_CH*n*
- LEDC_DUTY_INC_CH*n*
- LEDC_OVF_CNT_EN_CH*n*

LEDC_OVF_NUM_CH*n* This field is used to configure the number of counter overflows to generate an interrupt minus 1. The LEDC_OVF_CNT_CH*n*_INT interrupt will be triggered when channel *n* overflows for (LEDC_OVF_NUM_CH*n* + 1) times. (R/W)

Continued on the next page...

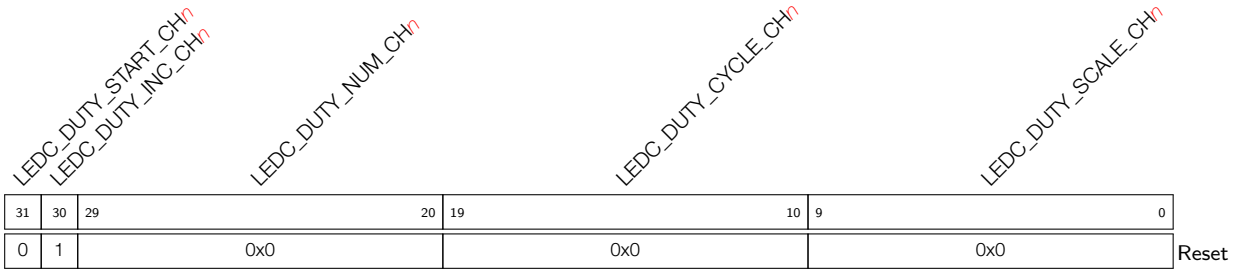
Register 22.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14n*)**

Continued from the previous page...

LEDC_OVF_CNT_EN_CH*n* This bit is used to enable the counter that counts the number of times when the timer selected by channel *n* overflows. (R/W)

LEDC_OVF_CNT_RESET_CH*n* Set this bit to reset the timer-overflow counter of channel *n*. (WT)

Register 22.2. LEDC_CH*n*_CONF1_REG (*n*: 0-5) (0x000C+0x14n*)**



LEDC_DUTY_SCALE_CH*n* This field configures the step size of the duty cycle change during fading. (R/W)

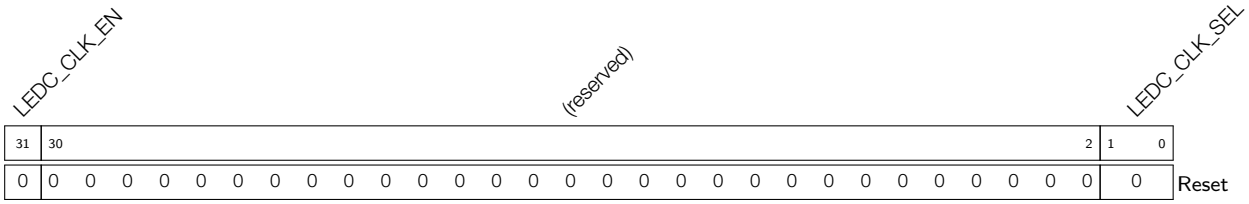
LEDC_DUTY_CYCLE_CH*n* The duty will change every LEDC_DUTY_CYCLE_CH*n* on channel *n*. (R/W)

LEDC_DUTY_NUM_CH*n* This field sets the maximum number of increments/decrements before duty cycle fading stops. (R/W)

LEDC_DUTY_INC_CH*n* This bit determines whether the duty cycle of the output signal on channel *n* increases or decreases. 1: Increase; 0: Decrease. (R/W)

LEDC_DUTY_START_CH*n* If this bit is set to 1, other configured fields in LEDC_CH*n*_CONF1_REG will take effect upon the next timer overflow.(R/W/SC)

Register 22.3. LEDC_CONF_REG (0x00D0)



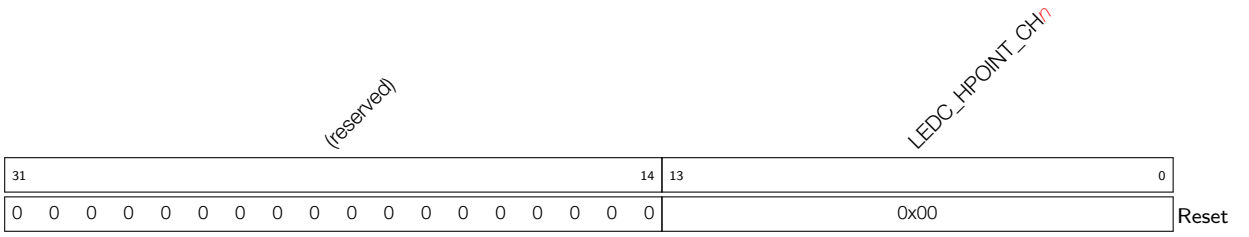
LEDC_CLK_SEL This field is used to select the common clock source for all the 4 timers.

1: PLL_60M_CLK; 2: FOSC_CLK; 3: XTAL_CLK. (R/W)

LEDC_CLK_EN This bit is used to control the clock.

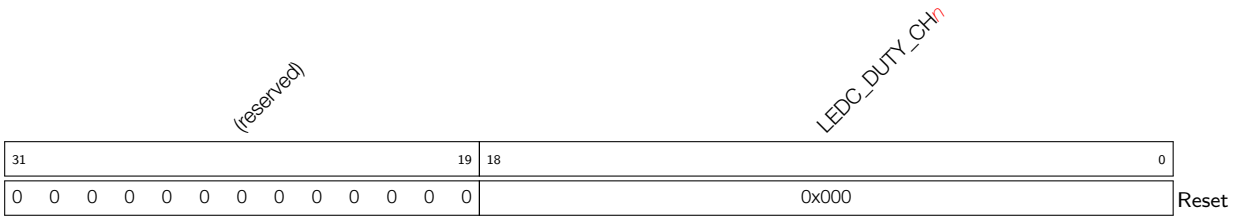
1: Force clock on for register. 0: Support clock only when application writes registers. (R/W)

Register 22.4. LEDC_CH*n*_HPOINT_REG (*n*: 0-5) (0x0004+0x14n*)**



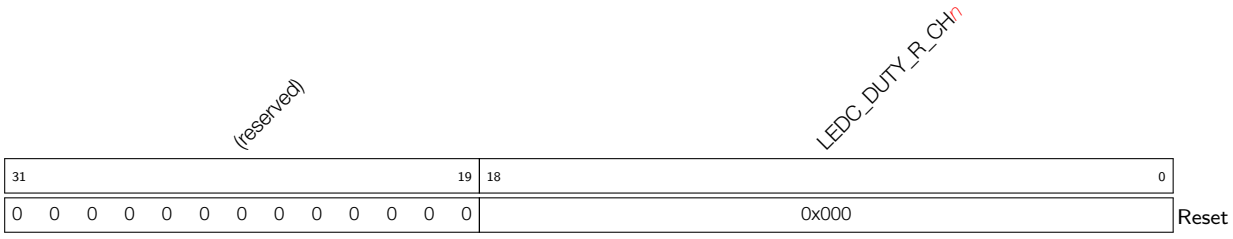
LEDC_HPOINT_CH*n* The output value changes to high when the selected timer for this channel has reached the value specified by this field. (R/W)

Register 22.5. LEDC_CH*n*_DUTY_REG (*n*: 0-5) (0x0008+0x14n*)**



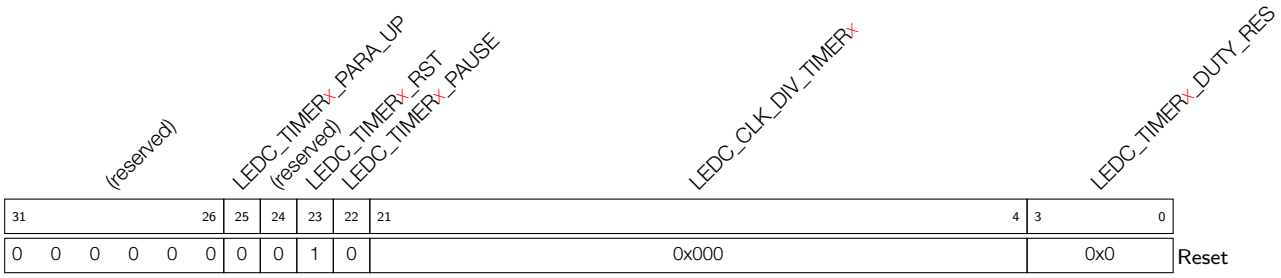
LEDC_DUTY_CH*n* This field is used to change the output duty by controlling the Lpoint. The output value turns to low when the selected timer for this channel has reached the Lpoint. (R/W)

Register 22.6. LEDC_CH*n*_DUTY_R_REG (*n*: 0-5) (0x0010+0x14n*)**



LEDC_DUTY_R_CH*n* This field stores the current duty cycle of the output signal on channel *n*. (RO)

Register 22.7. LEDC_TIMER*x*_CONF_REG (*x*: 0-3) (0x00A0+0x8x*)**



LEDC_TIMER*x*_DUTY_RES This field is used to control the range of the counter in timer *x*. (R/W)

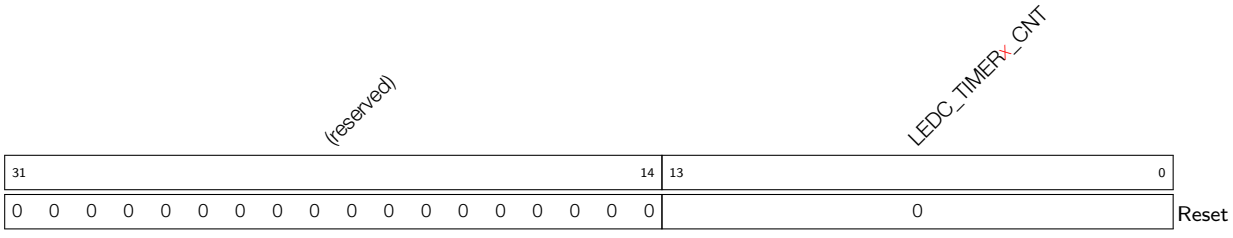
LEDC_CLK_DIV_TIMER*x* This field is used to configure the divisor for the divider in timer *x*. The least significant eight bits represent the fractional part. (R/W)

LEDC_TIMER*x*_PAUSE This bit is used to suspend the counter in timer *x*. (R/W)

LEDC_TIMER*x*_RST This bit is used to reset timer *x*. The counter will show 0 after reset. (R/W)

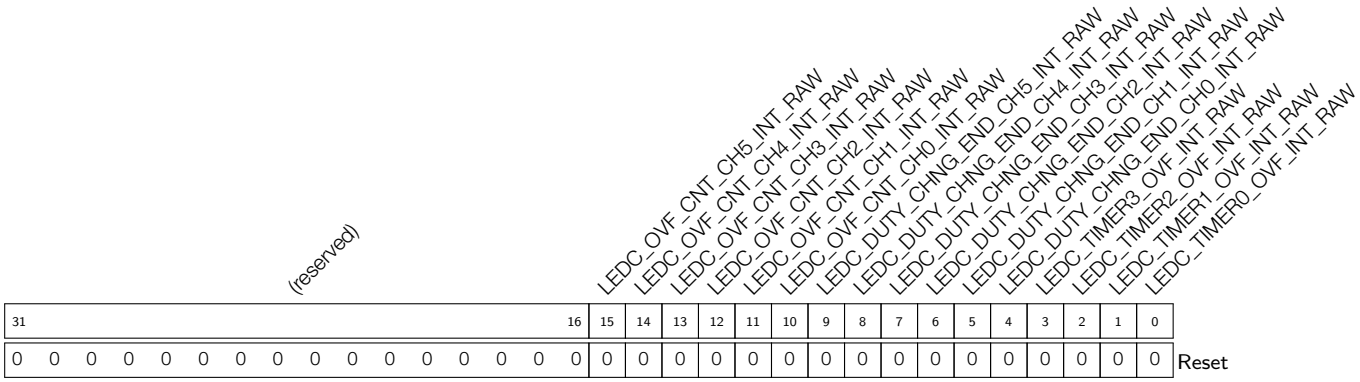
LEDC_TIMER*x*_PARA_UP Set this bit to update LEDC_CLK_DIV_TIMER*x* and LEDC_TIMER*x*_DUTY_RES. (WT)

Register 22.8. LEDC_TIMER*x*_VALUE_REG (*x*: 0-3) (0x00A4+0x8x*)**



LEDC_TIMER*x*_CNT This field stores the current counter value of timer *x*. (RO)

Register 22.9. LEDC_INT_RAW_REG (0x00C0)

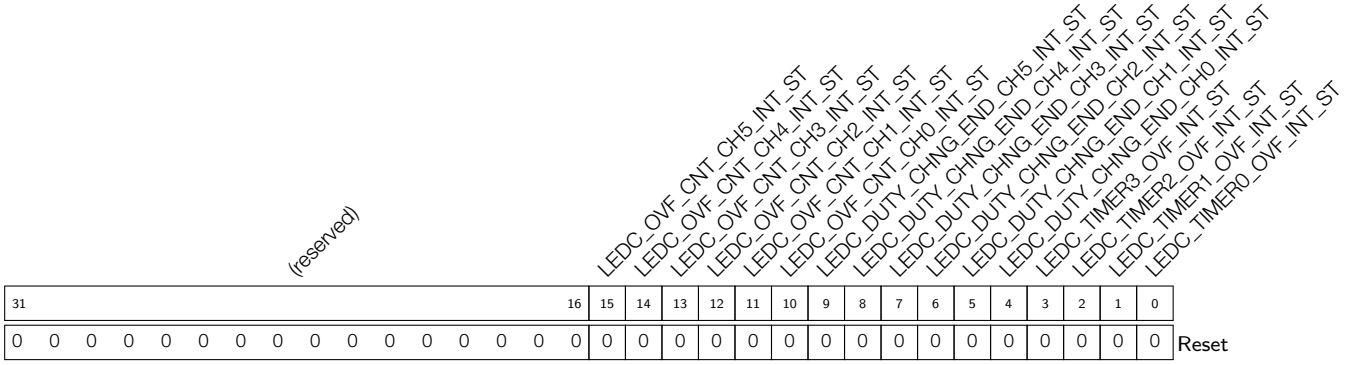


LEDC_TIMER*x*_OVF_INT_RAW The raw interrupt status of LEDC_TIMER*x*_OVF_INT. (R/WTC/SS)

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

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

Register 22.10. LEDC_INT_ST_REG (0x00C4)

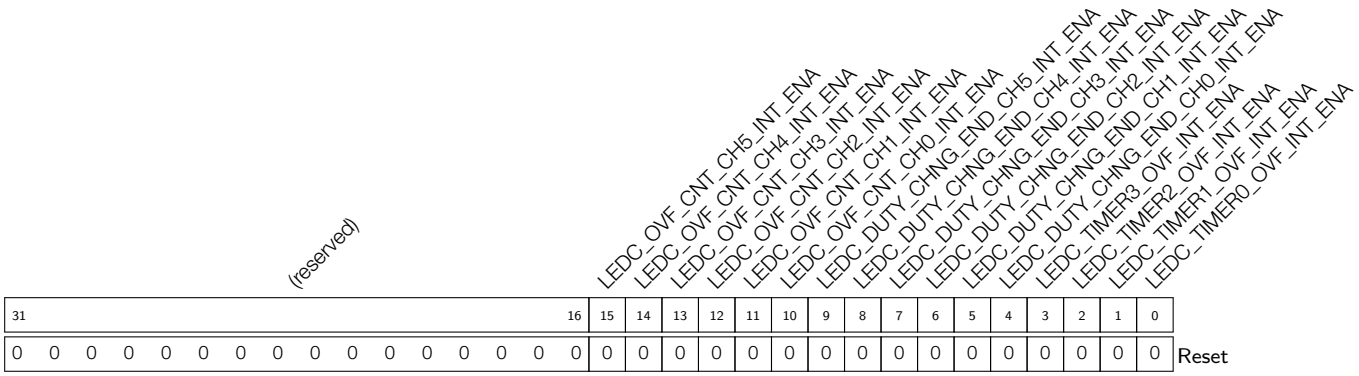


LEDC_TIMER*x*_OVF_INT_ST This is the masked interrupt status bit for the LEDC_TIMER*x*_OVF_INT interrupt when LEDC_TIMER*x*_OVF_INT_ENA is set to 1. (RO)

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_ENA is 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 22.11. LEDC_INT_ENA_REG (0x00C8)

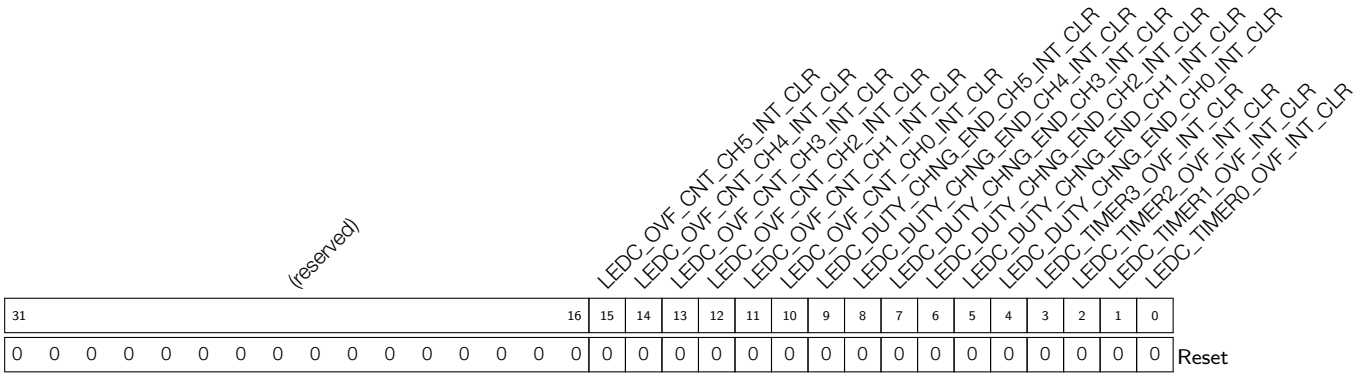


LEDC_TIMER*x*_OVF_INT_ENA The interrupt enable bit for the LEDC_TIMER*x*_OVF_INT interrupt. (R/W)

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 22.12. LEDC_INT_CLR_REG (0x00CC)

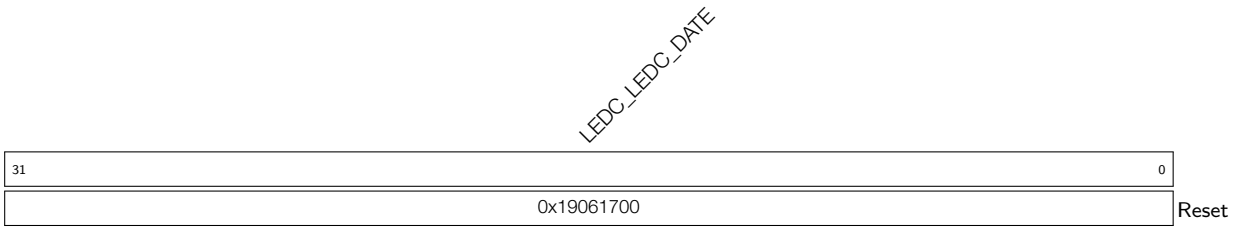


LEDC_TIMER*x*_OVF_INT_CLR Set this bit to clear the LEDC_TIMER*x*_OVF_INT interrupt. (WT)

LEDC_DUTY_CHNG_END_CH*n*_INT_CLR Set this bit to clear the LEDC_DUTY_CHNG_END_CH*n*_INT interrupt. (WT)

LEDC_OVF_CNT_CH*n*_INT_CLR Set this bit to clear the LEDC_OVF_CNT_CH*n*_INT interrupt. (WT)

Register 22.13. LEDC_DATE_REG (0x00FC)



LEDC_LEDC_DATE This is the version control register. (R/W)

23 On-Chip Sensor and Analog Signal Processing

23.1 Overview

ESP8684 provides the following analog signal processing peripheral and on-chip sensor:

- One 12-bit Successive Approximation ADC (SAR ADC) for measuring analog signals from five channels.
- One temperature sensor for measuring the internal temperature of the ESP8684 chip.

23.2 SAR ADC

23.2.1 Overview

ESP8684 integrates one 12-bit SAR ADC, which is able to measure analog signals from up to five pins. The SAR ADC is managed by DIG ADC controller, which drives Digital_Reader to sample channel voltages of SAR ADC. This controller supports multi-channel scanning and threshold monitoring.

23.2.2 Features

SAR ADC has the following features:

- One ADC Reader module (Digital_Reader) to read sampling results
- 12-bit sampling resolution
- Able to sample the analog voltages from up to five pins
- One DIG ADC controller
  - Provides separate control modules for one-time sampling and multi-channel scanning
  - Supports one-time sampling and multi-channel scanning working simultaneously
  - User-defined scanning sequence in multi-channel scanning mode
  - Provides two filters with configurable filter coefficient
  - Supports threshold monitoring

23.2.3 Functional Description

The major components of SAR ADC and their interconnections are shown in Figure 23-1.

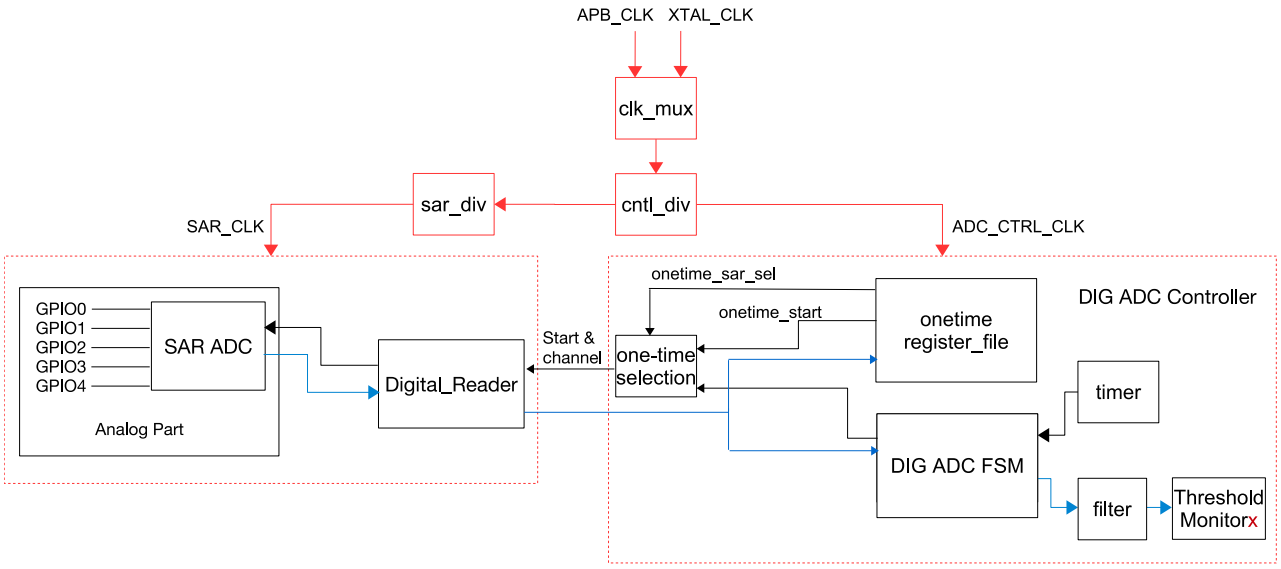


Figure 23-1. SAR ADC Function Overview

As shown in Figure 23-1, the SAR ADC module consists of the following components:

- SAR ADC: measures voltages from up to five channels.
- Clock management: selects clock sources and their dividers:
  - Clock sources: can be APB_CLK or XTAL_CLK.
  - Divided Clocks:
    * SAR_CLK: operating clock for SAR ADC and Digital_Reader. Note that the divider (sar_div) of SAR_CLK must be no less than 2.
    * ADC_CTRL_CLK: operating clock for DIG ADC FSM.
- Digital_Reader (driven by DIG ADC FSM): reads data from SAR ADC.
- DIG ADC FSM: generates the signals required throughout the ADC sampling process.
- Threshold monitor*x*: threshold monitor 1 and threshold monitor 2. The monitor*x* will trigger an interrupt when the sampled value is greater than the pre-set high threshold or less than the pre-set low threshold.

The following sections describe the individual components in details.

23.2.3.1 Input Signals

In order to sample an analog signal, the SAR ADC must first select the analog pin to measure via an internal multiplexer. A summary of all the analog signals that may be sent to the SAR ADC module for processing are presented in Table 23-1.

Table 23-1. SAR ADC Input Signals

Signal	Channel
GPIO0	0
GPIO1	1
GPIO2	2
GPIO3	3
GPIO4	4

23.2.3.2 ADC Conversion and Attenuation

When the SAR ADC converts an analog voltage, the resolution (12-bit) of the conversion spans voltage range from 0 mV to V_{ref}. V_{ref} is the SAR ADC's internal reference voltage. The output value of the conversion (data) is mapped to analog voltage V_{data} using the following formula:

$$V_{data} = \frac{V_{ref}}{4095} \times data$$

In order to convert voltages larger than V_{ref}, input signals can be attenuated before being input into the SAR ADCs. The attenuation can be configured to 0 dB, 2.5 dB, 6 dB, and 10 dB.

23.2.3.3 DIG ADC Controller

The clock of the DIG ADC controller is quite fast, thus the sample rate is high. For more information, see Section ADC Characteristics in ESP8684 Series Datasheet.

This controller supports:

- up to 12-bit sampling resolution
- one-time sampling triggered by software
- multi-channel scanning triggered by the timer

The configuration of a one-time sampling triggered by the software is as follows:

- Set APB_SARADC1_ONETIME_SAMPLE to enable the one-time sampling function of the SAR ADC.
- Configure APB_SARADC_ONETIME_CHANNEL to select one channel to sample.
- Configure APB_SARADC_ONETIME_ATTEN to set attenuation.
- Configure APB_SARADC_ONETIME_START to start this one-time sampling.
- On completion of sampling, APB_SARADC_ADC1_DONE_INT_RAW interrupt is generated. Software can use this interrupt to initiate reading of the sampled values from APB_SARADC_ADC1_DATA.

If the timer-triggered multi-channel scanning is selected, follow the configuration below. Note that in this mode, the scan sequence is performed according to the configuration entered into pattern table.

- Configure APB_SARADC_TIMER_TARGET to set the trigger target for DIG ADC timer. When the timer counting reaches two times of the pre-configured cycle number, a sampling operation is triggered. For the working clock of the timer, see Section 23.2.3.4.
- Configure APB_SARADC_TIMER_EN to enable the timer.

- When the timer times out, DIG ADC FSM starts sampling according to the pattern table;
- **An interrupt is triggered once the scanning is completed. The software needs to read the sampled data from corresponding registers, otherwise, the sampled data will be directly discarded after passing through the threshold monitor.**

23.2.3.4 DIG ADC Clock

Two clocks can be used as the working clock of DIG ADC controller, depending on the configuration of APB_SARADC_CLK_SEL:

- 1: Select the clock (ADC_CTRL_CLK) divided from XTAL_CLK.
- 0: Select APB_CLK.

If ADC_CTRL_CLK is selected, users can configure the divider by APB_SARADC_CLKM_DIV_NUM.

Note that due to speed limits of SAR ADC, the operating clock of Digital_Reader and SAR ADC is SAR_CLK, the frequency of which affects the sampling precision. The lower the frequency, the higher the precision. SAR_CLK is divided from ADC_CTRL_CLK. The divider coefficient is configured by APB_SARADC_SAR_CLK_DIV.

The ADC needs 25 SAR_CLK clock cycles per sample, so the maximum sampling rate is limited by the SAR_CLK frequency. For more information about clocks, see Chapter 6 *Reset and Clock*.

23.2.3.5 DIG ADC FSM

Overview

Figure 23-2 shows the diagram of DIG ADC FSM.

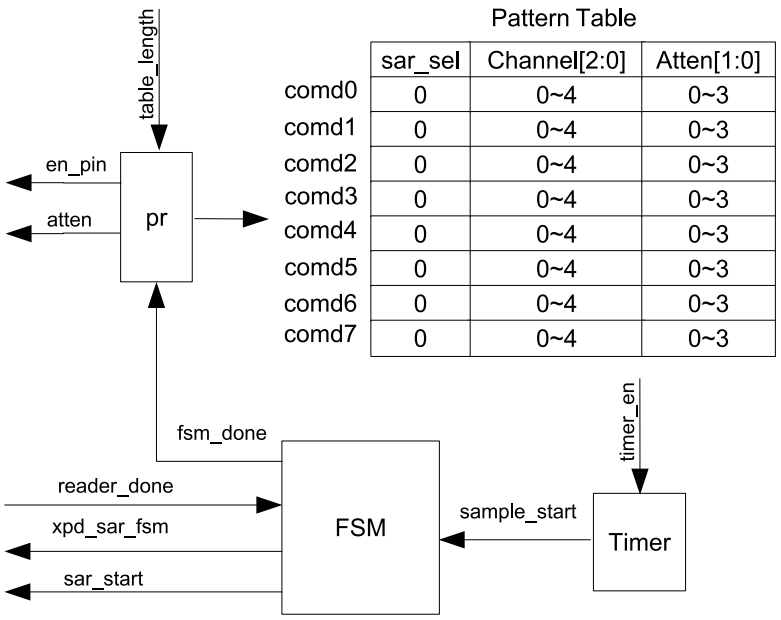


Figure 23-2. Diagram of DIG ADC FSM

Wherein:

- Timer: a dedicated timer for DIG ADC controller, to generate a sample_start signal.

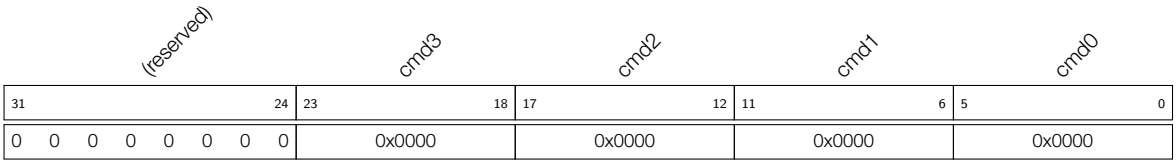
- pr: the pointer to pattern table entries. FSM sends out corresponding signals based on the configuration of the pattern table entry that the pointer points to.

The execution process is as follows:

- Configure APB_SARADC_TIMER_EN to enable the DIG ADC timer. The timeout event of this timer triggers a sample_start signal. This signal drives the FSM module to start sampling.
- When the FSM module receives the sample_start signal, it starts the following operations:
  - Power up SAR ADC.
  - Select SAR ADC as the working ADC, configure the ADC channel and attenuation, based on the pattern table entry that the current pr points to.
  - According to the configuration information, output the corresponding en_pad and atten signals to the analog side.
  - Initiate the sar_start signal and start sampling.
- When the FSM receives the reader_done signal from ADC Reader (Digital_Reader), it will
  - stop sampling,
  - the data is discarded after passing through the filter and the threshold monitor, see Figure 23-1),
  - update the pattern table pointer (pr) and wait for the next sampling. Note that if the pointer (pr) is smaller than APB_SARADC_SAR_PATT_LEN (table_length), then pr = pr + 1, otherwise, pr is cleared.

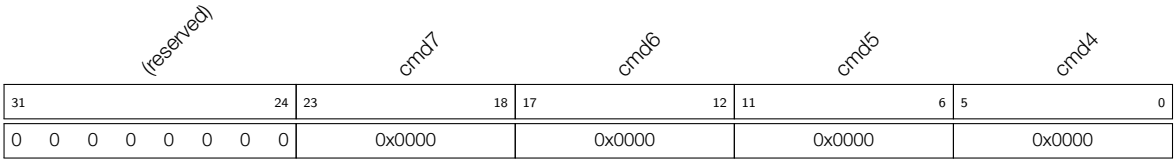
Pattern Table

There is one pattern table in the controller, consisting of the APB_SARADC_SAR_PATT_TAB1_REG and APB_SARADC_SAR_PATT_TAB2_REG registers, see Figure 23-3 and Figure 23-4:



cmd *x* represents pattern table entries. *x* here is the index, 0 ~ 3.

Figure 23-3. APB_SARADC_SAR_PATT_TAB1_REG and Pattern Table Entry 0 - Entry 3



cmd *x* represents pattern table entries. *x* here is the index, 4 ~ 7.

Figure 23-4. APB_SARADC_SAR_PATT_TAB2_REG and Pattern Table Entry 4 - Entry 7

Each register consists of four 6-bit pattern table entries. Each entry is composed of three fields that contain working ADC, ADC channel and attenuation information, as shown in Table 23-5.

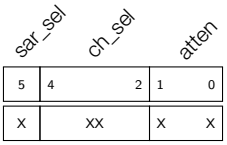


Figure 23-5. Pattern Table Entry

atten Attenuation. 0: 0 dB; 1: 2.5 dB; 2: 6 dB; 3: 10 dB.

ch_sel ADC channel, see Table 23-1.

sar_sel Working ADC. 0: SAR ARC. ESP8684 provides only one SAR ADC, therefore, this value is always 0.

Configuration of multi-channel scanning

In this example, two channels are selected for multi-channel scanning:

- Channel 2, with the attenuation of 10 dB
- Channel 0, with the attenuation of 2.5 dB

The detailed configuration is as follows:

- Configure the first pattern table entry (cmd0):

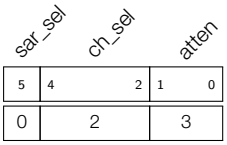


Figure 23-6. cmd0 Configuration

atten write the value of 3 to this field, to set the attenuation to 10 dB.

ch_sel write the value of 2 to this field, to select channel 2 (see Table 23-1).

sar_sel Configure this bit to 0.

- Configure the second pattern table entry (cmd1):

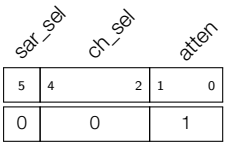


Figure 23-7. cmd1 Configuration

atten write the value of 1 to this field, to set the attenuation to 2.5 dB.

ch_sel write the value of 0 to this field, to select channel 0 (see Table 23-1).

sar_sel Configure this bit to 0.

- Configure APB_SARADC_SAR_PATT_LEN to 1, i.e., set pattern table length to (this value + 1 = 2). Then pattern table entries cmd0 and cmd1 will be used.
- Enable the timer, then DIG ADC controller starts scanning the two channels in cycles, as configured in the pattern table entries.

23.2.3.6 ADC Filters

The DIG ADC controller provides two filters for automatic filtering of sampled ADC data. Both filters can be configured to any channel of the SAR ADC and then filter the sampled data for the target channel. The filter's formula is shown below:

$$data_{cur} = \frac{(k-1)data_{prev}}{k} + \frac{data_{in}}{k} - 0.5$$

- $data_{cur}$: the filtered data value.
- $data_{in}$: the sampled data value from the ADC.
- $data_{prev}$: the last filtered data value.
- k: the filter coefficient.

The filters are configured as follows:

- Configure APB_SARADC_FILTER_CHANNEL*x* to select the ADC channel for filter *x*;
- Configure APB_SARADC_FILTER_FACTOR*x* to set the coefficient for filter *x*;

Note that *x* is used here as the placeholder of filter index. 0: filter 0; 1: filter 1.

23.2.3.7 Threshold Monitoring

DIG ADC controller contains two threshold monitors that can be configured to monitor on any channel of the SAR ADC. A high threshold interrupt is triggered when the ADC sample value is larger than the pre-configured high threshold, and a low threshold interrupt is triggered if the sample value is lower than the pre-configured low threshold.

The configuration of threshold monitoring is as follows:

- Set APB_SARADC_THRES*x*_EN to enable threshold monitor *x*.
- Configure APB_SARADC_THRES*x*_LOW to set a low threshold;
- Configure APB_SARADC_THRES*x*_HIGH to set a high threshold;
- Configure APB_SARADC_THRES*x*_CHANNEL to select the channel to monitor.

Note that *x* is used here as the placeholder of monitor index. 0: monitor 0; 1: monitor 1.

23.3 Temperature Sensor

23.3.1 Overview

ESP8684 provides a temperature sensor to monitor temperature changes inside the chip in real time.

23.3.2 Features

The temperature sensor has the following features:

- Supports software triggering and, once triggered, the data can be read continuously
- Configurable temperature offset based on the environment, to improve the accuracy
- Adjustable measurement range

23.3.3 Functional Description

The temperature sensor can be started by software as follows:

- Set APB_SARADC_TSENS_PU to power up the temperature sensor;
- Wait for APB_SARADC_TSENS_XPD_WAIT clock cycles till the reset of temperature sensor is released, the sensor starts measuring the temperature;
- If this is the first time to start the temperature sensor, wait the sensor to get started up (about 100 μs). Then, the temperature data can be read continuously from APB_SARADC_TSENS_OUT.

The actual temperature (°C) can be obtained by converting the output of temperature sensor via the following formula:

$$T(°C) = 0.4386 * VALUE - 27.88 * offset - 20.52$$

VALUE in the formula is the output of the temperature sensor, and the offset is determined by the temperature offset TSENS_DAC. Users can set I2C register I2C_SARADC_TSENS_ADC to configure TSENS_DAC according to the actual environment (the temperature range) and Table 23-2.

Table 23-2. Temperature Offset

TSENS_DAC	Temperature Offset (°C)	Measurement Range (°C)
5	-2	50 ~ 125
13 or 7	-1	20 ~ 100
15	0	-10 ~ 80
11 or 14	1	-30 ~ 50
10	2	-40 ~ 20

23.4 Interrupts

- APB_SARADC_ADC1_DONE_INT: triggered when SAR ADC completes one data conversion.
- APB_SARADC_THRES*x*_HIGH_INT: triggered when the sampling value is higher than the high threshold of monitor *x*.
- APB_SARADC_THRES*x*_LOW_INT: triggered when the sampling value is lower than the low threshold of monitor *x*.

23.5 Register Summary

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

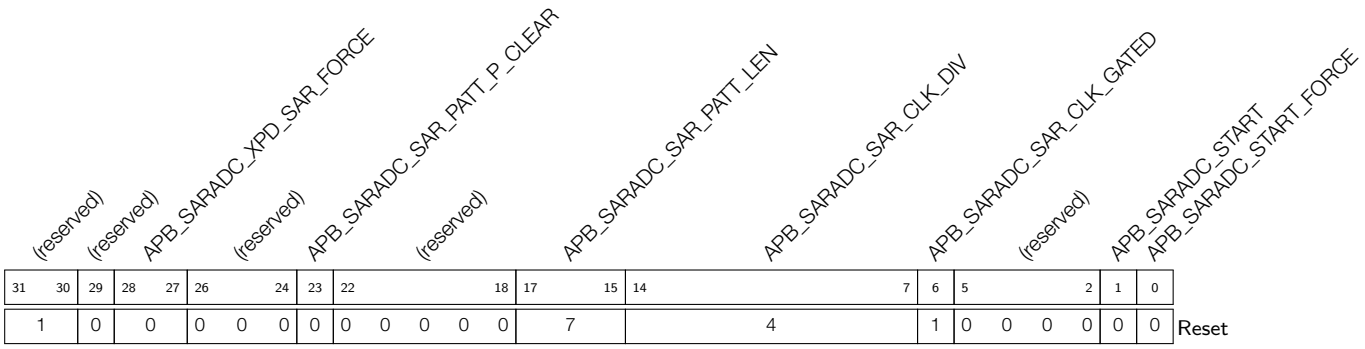
Name	Description	Address	Access
Configuration registers			
APB_SARADC_CTRL_REG	Configuration register for SAR ADC FSM	0x0000	R/W
APB_SARADC_CTRL2_REG	Configuration register for SAR ADC FSM sampling	0x0004	R/W
APB_SARADC_FILTER_CTRL1_REG	Configuration register 1 for filter	0x0008	R/W

Name	Description	Address	Access
APB_SARADC_SAR_PATT_TAB1_REG	Pattern table register 1	0x0018	R/W
APB_SARADC_SAR_PATT_TAB2_REG	Pattern table register 2	0x001C	R/W
APB_SARADC_ONETIME_SAMPLE_REG	Configuration register for one-time sampling	0x0020	R/W
APB_SARADC_FILTER_CTRL0_REG	Configuration register 0 for filter	0x0028	R/W
APB_SARADC_1_DATA_STATUS_REG	SAR ADC sampling data register	0x002C	RO
APB_SARADC_THRES0_CTRL_REG	Sampling threshold control register 0	0x0034	R/W
APB_SARADC_THRES1_CTRL_REG	Sampling threshold control register 1	0x0038	R/W
APB_SARADC_THRES_CTRL_REG	Sampling threshold enable register	0x003C	R/W
APB_SARADC_INT_ENA_REG	Enable register of SAR ADC interrupts	0x0040	R/W
APB_SARADC_INT_RAW_REG	Raw register of SAR ADC interrupts	0x0044	RO
APB_SARADC_INT_ST_REG	State register of SAR ADC interrupts	0x0048	RO
APB_SARADC_INT_CLR_REG	Clear register of SAR ADC interrupts	0x004C	WO
APB_SARADC_DMA_CONF_REG	DMA configuration register for SAR ADC	0x0050	R/W
APB_SARADC_APB_ADC_CLKM_CONF_REG	SAR ADC clock control register	0x0054	R/W
APB_SARADC_APB_TSENS_CTRL_REG	Temperature sensor control register 1	0x0058	varies
APB_SARADC_APB_TSENS_CTRL2_REG	Temperature sensor control register 2	0x005C	R/W
Version register			
APB_SARADC_APB_CTRL_DATE_REG	Version control register	0x03FC	R/W

23.6 Register

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

Register 23.1. APB_SARADC_CTRL_REG (0x0000)



APB_SARADC_START_FORCE 0: select FSM to start SAR ADC. 1: select software to start SAR ADC. (R/W)

APB_SARADC_START Write 1 here to start the SAR ADC by software. Valid only when APB_SARADC_START_FORCE = 1. (R/W)

APB_SARADC_SAR_CLK_GATED 0: SAR ADC clock is always on. 1: SAR ADC clock is turned off when SAR ADC is in idle. (R/W)

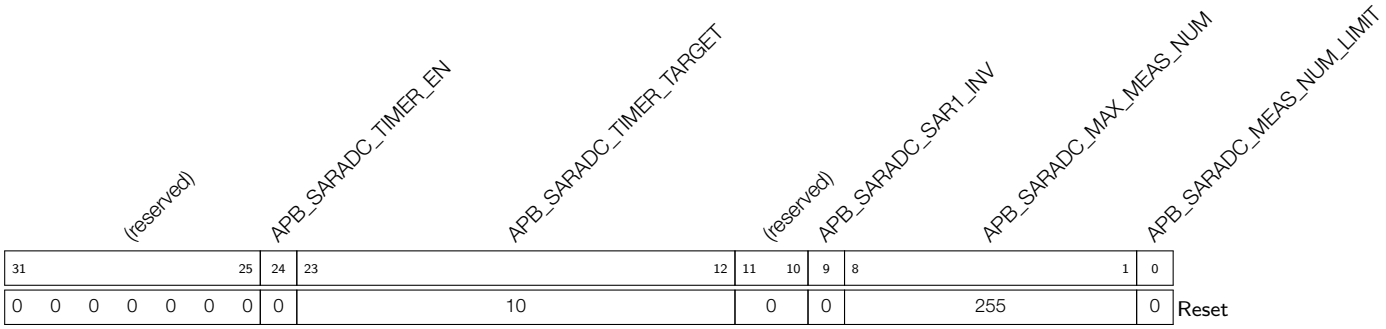
APB_SARADC_SAR_CLK_DIV SAR ADC clock divider. This value should be no less than 2. (R/W)

APB_SARADC_SAR_PATT_LEN Configure how many pattern table entries will be used. If this field is set to 1, then pattern table entries (cmd0) and (cmd1) will be used. (R/W)

APB_SARADC_SAR_PATT_P_CLEAR Clear the pointer of pattern table entry for DIG ADC controller. (R/W)

APB_SARADC_XPD_SAR_FORCE Force select XPD SAR. (R/W)

Register 23.2. APB_SARADC_CTRL2_REG (0x0004)



APB_SARADC_MEAS_NUM_LIMIT Enable the limitation of SAR ADC maximum conversion times. Valid only when the timer is used to control SAR ADC. (R/W)

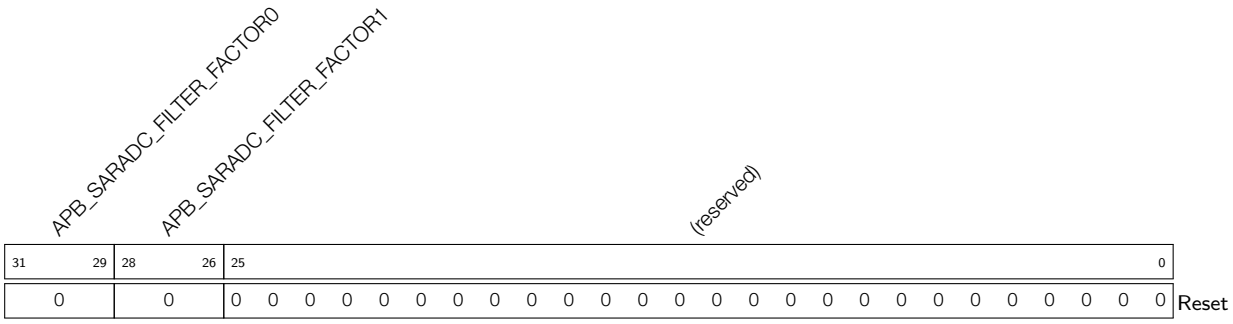
APB_SARADC_MAX_MEAS_NUM The SAR ADC maximum conversion times. (R/W)

APB_SARADC_SAR1_INV Write 1 here to invert the data of SAR ADC. (R/W)

APB_SARADC_TIMER_TARGET Set SAR ADC timer target. (R/W)

APB_SARADC_TIMER_EN Enable SAR ADC timer trigger. (R/W)

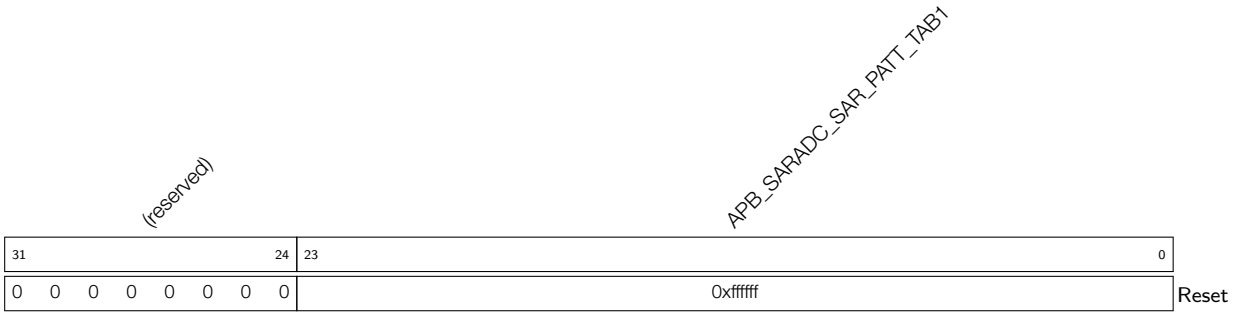
Register 23.3. APB_SARADC_FILTER_CTRL1_REG (0x0008)



APB_SARADC_FILTER_FACTOR1 The filter coefficient for SAR ADC filter 1. (R/W)

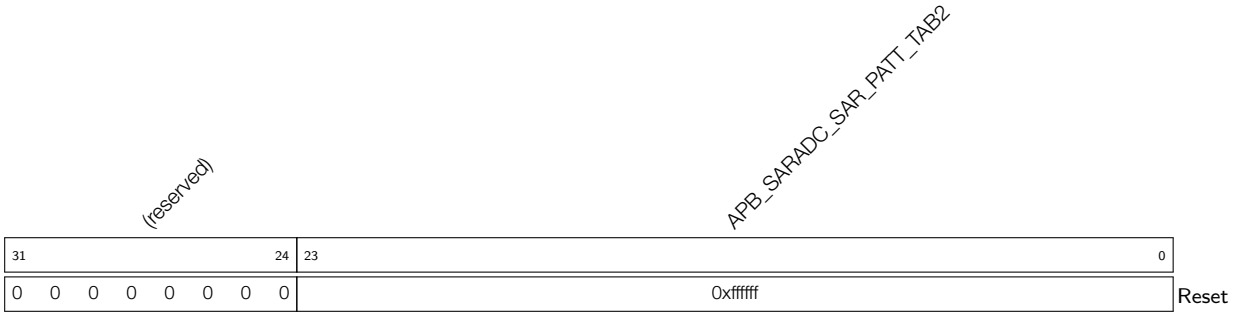
APB_SARADC_FILTER_FACTOR0 The filter coefficient for SAR ADC filter 0. (R/W)

Register 23.4. APB_SARADC_SAR_PATT_TAB1_REG (0x0018)



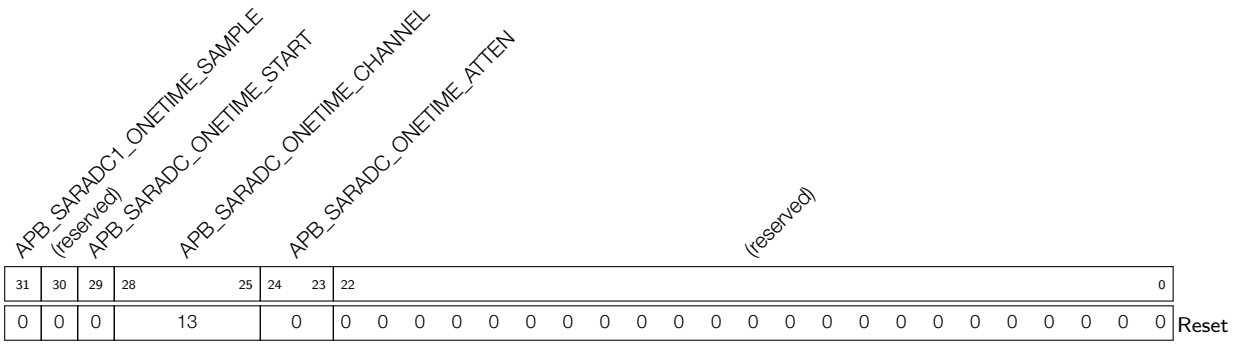
APB_SARADC_SAR_PATT_TAB1 Pattern table entries 0 ~ 3 (each entry is six bits). (R/W)

Register 23.5. APB_SARADC_SAR_PATT_TAB2_REG (0x001C)



APB_SARADC_SAR_PATT_TAB2 Pattern table entries 4 ~ 7 (each entry is six bits). (R/W)

Register 23.6. APB_SARADC_ONETIME_SAMPLE_REG (0x0020)



APB_SARADC_ONETIME_ATTEN Configure the attenuation for a one-time sampling. (R/W)

APB_SARADC_ONETIME_CHANNEL Configure the channel for a one-time sampling. (R/W)

APB_SARADC_ONETIME_START Start SAR ADC one-time sampling. (R/W)

APB_SARADC1_ONETIME_SAMPLE Enable SAR ADC one-time sampling. (R/W)

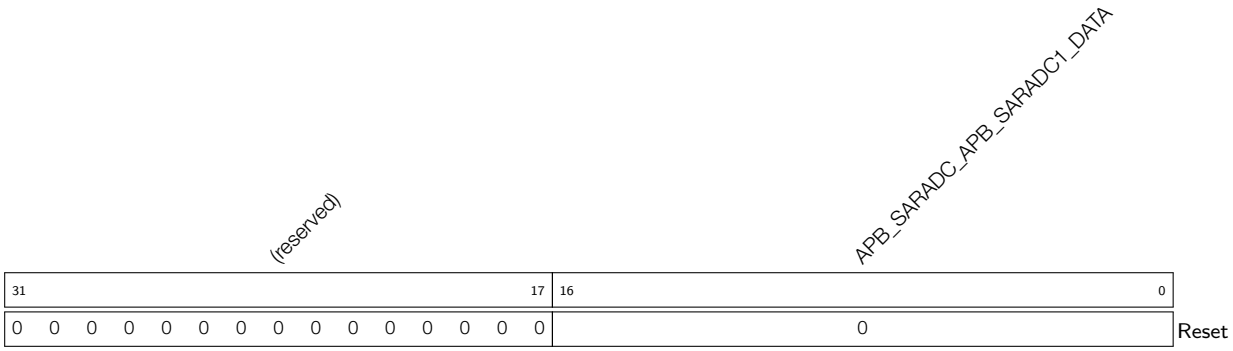
Register 23.7. APB_SARADC_FILTER_CTRL0_REG (0x0028)

APB_SARADC_FILTER_CHANNEL1 The filter channel for SAR ADC filter 1. (R/W)

APB_SARADC_FILTER_CHANNEL0 The filter channel for SAR ADC filter 0. (R/W)

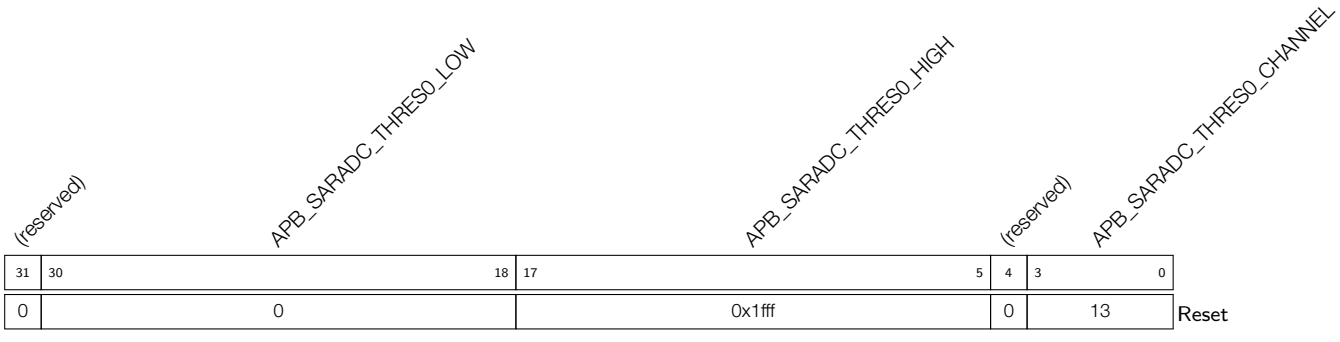
APB_SARADC_FILTER_RESET Reset SAR ADC filter. (R/W)

Register 23.8. APB_SARADC_1_DATA_STATUS_REG (0x002C)



APB_SARADC_ADC1_DATA SAR ADC conversion data. (RO)

Register 23.9. APB_SARADC_THRES0_CTRL_REG (0x0034)

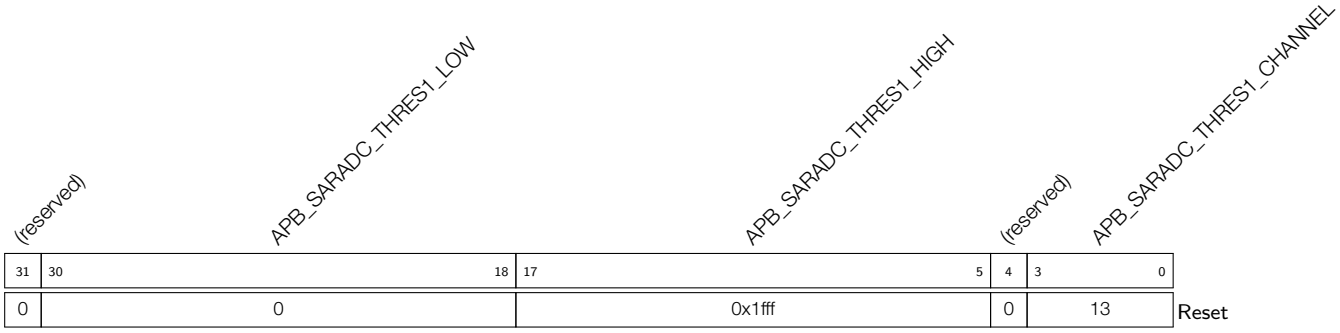


APB_SARADC_THRES0_CHANNEL The channel for SAR ADC monitor 0. (R/W)

APB_SARADC_THRES0_HIGH The high threshold for SAR ADC monitor 0. (R/W)

APB_SARADC_THRES0_LOW The low threshold for SAR ADC monitor 0. (R/W)

Register 23.10. APB_SARADC_THRES1_CTRL_REG (0x0038)

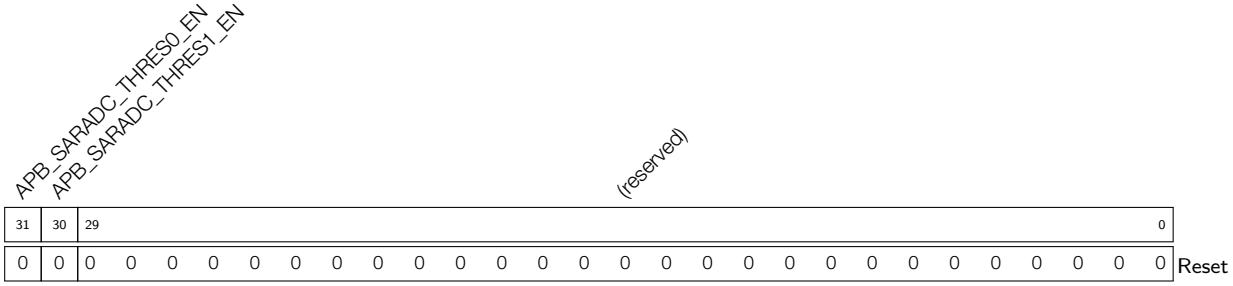


APB_SARADC_THRES1_CHANNEL The channel for SAR ADC monitor 1. (R/W)

APB_SARADC_THRES1_HIGH The high threshold for SAR ADC monitor 1. (R/W)

APB_SARADC_THRES1_LOW The low threshold for SAR ADC monitor 1. (R/W)

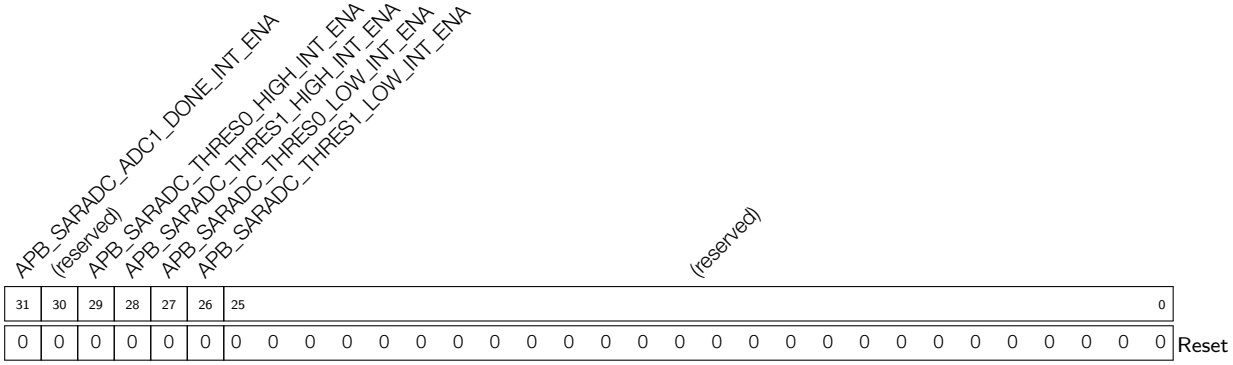
Register 23.11. APB_SARADC_THRES_CTRL_REG (0x003C)



APB_SARADC_THRES1_EN Enable threshold monitor 1. (R/W)

APB_SARADC_THRES0_EN Enable threshold monitor 0. (R/W)

Register 23.12. APB_SARADC_INT_ENA_REG (0x0040)



APB_SARADC_THRES1_LOW_INT_ENA Enable bit of APB_SARADC_THRES1_LOW_INT interrupt. (R/W)

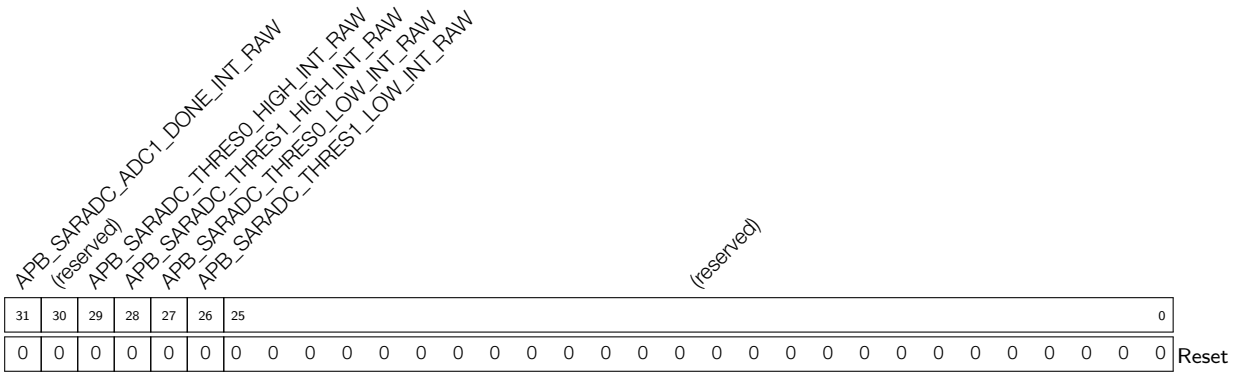
APB_SARADC_THRES0_LOW_INT_ENA Enable bit of APB_SARADC_THRES0_LOW_INT interrupt. (R/W)

APB_SARADC_THRES1_HIGH_INT_ENA Enable bit of APB_SARADC_THRES1_HIGH_INT interrupt. (R/W)

APB_SARADC_THRES0_HIGH_INT_ENA Enable bit of APB_SARADC_THRES0_HIGH_INT interrupt. (R/W)

APB_SARADC_ADC1_DONE_INT_ENA Enable bit of APB_SARADC_ADC1_DONE_INT interrupt. (R/W)

Register 23.13. APB_SARADC_INT_RAW_REG (0x0044)



APB_SARADC_THRES1_LOW_INT_RAW Raw bit of APB_SARADC_THRES1_LOW_INT interrupt. (RO)

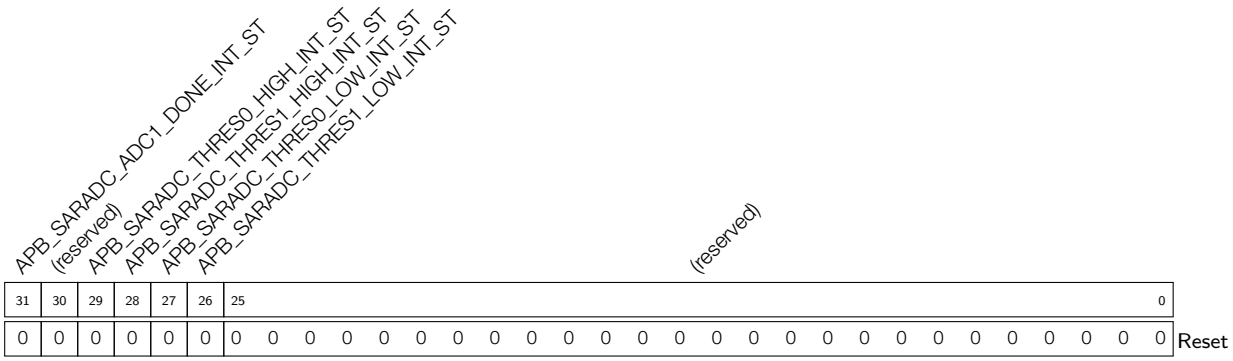
APB_SARADC_THRES0_LOW_INT_RAW Raw bit of APB_SARADC_THRES0_LOW_INT interrupt. (RO)

APB_SARADC_THRES1_HIGH_INT_RAW Raw bit of APB_SARADC_THRES1_HIGH_INT interrupt. (RO)

APB_SARADC_THRES0_HIGH_INT_RAW Raw bit of APB_SARADC_THRES0_HIGH_INT interrupt. (RO)

APB_SARADC_ADC1_DONE_INT_RAW Raw bit of APB_SARADC_ADC1_DONE_INT interrupt. (RO)

Register 23.14. APB_SARADC_INT_ST_REG (0x0048)



APB_SARADC_THRES1_LOW_INT_ST Status of APB_SARADC_THRES1_LOW_INT interrupt. (RO)

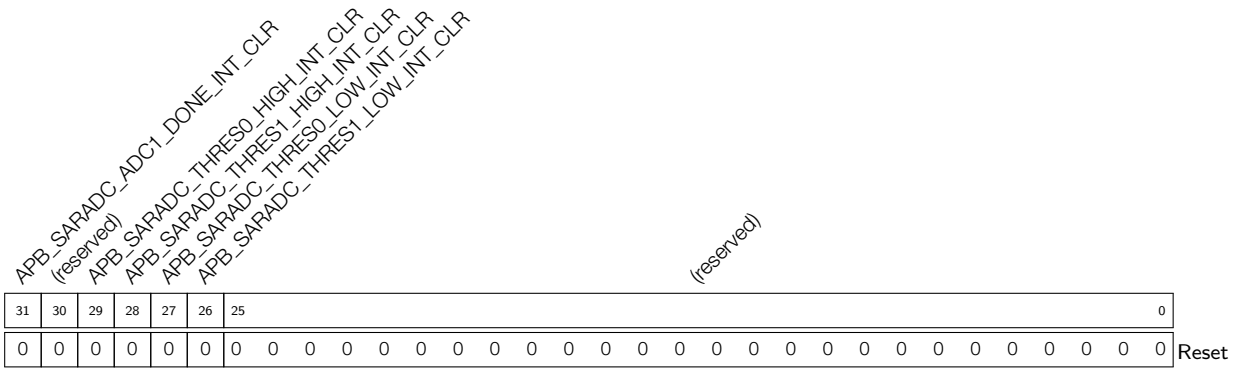
APB_SARADC_THRES0_LOW_INT_ST Status of APB_SARADC_THRES0_LOW_INT interrupt. (RO)

APB_SARADC_THRES1_HIGH_INT_ST Status of APB_SARADC_THRES1_HIGH_INT interrupt. (RO)

APB_SARADC_THRES0_HIGH_INT_ST Status of APB_SARADC_THRES0_HIGH_INT interrupt. (RO)

APB_SARADC_ADC1_DONE_INT_ST Status of APB_SARADC_ADC1_DONE_INT interrupt. (RO)

Register 23.15. APB_SARADC_INT_CLR_REG (0x004C)



APB_SARADC_THRES1_LOW_INT_CLR Clear bit of APB_SARADC_THRES1_LOW_INT interrupt. (WO)

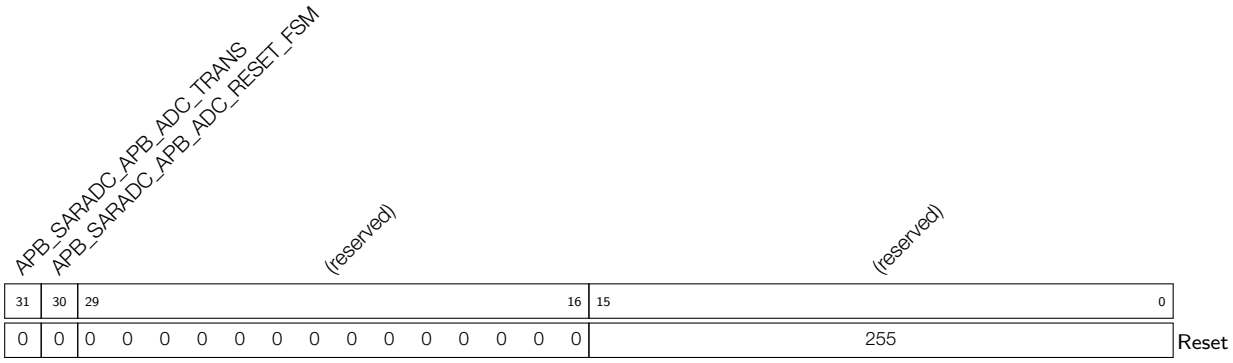
APB_SARADC_THRES0_LOW_INT_CLR Clear bit of APB_SARADC_THRES0_LOW_INT interrupt. (WO)

APB_SARADC_THRES1_HIGH_INT_CLR Clear bit of APB_SARADC_THRES1_HIGH_INT interrupt. (WO)

APB_SARADC_THRES0_HIGH_INT_CLR Clear bit of APB_SARADC_THRES0_HIGH_INT interrupt. (WO)

APB_SARADC_ADC1_DONE_INT_CLR Clear bit of APB_SARADC_ADC1_DONE_INT interrupt. (WO)

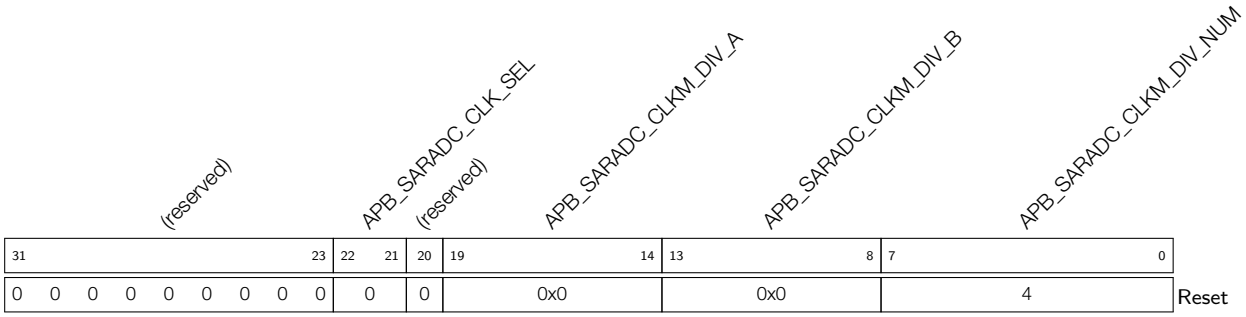
Register 23.16. APB_SARADC_DMA_CONF_REG (0x0050)



APB_SARADC_APB_ADC_RESET_FSM Reset DIG ADC controller status. (R/W)

APB_SARADC_APB_ADC_TRANS ESP8684 does not support DMA. This bit must be set to 0. Data is discarded after passing through the filter and the threshold monitor. (R/W)

Register 23.17. APB_SARADC_APB_ADC_CLKM_CONF_REG (0x0054)



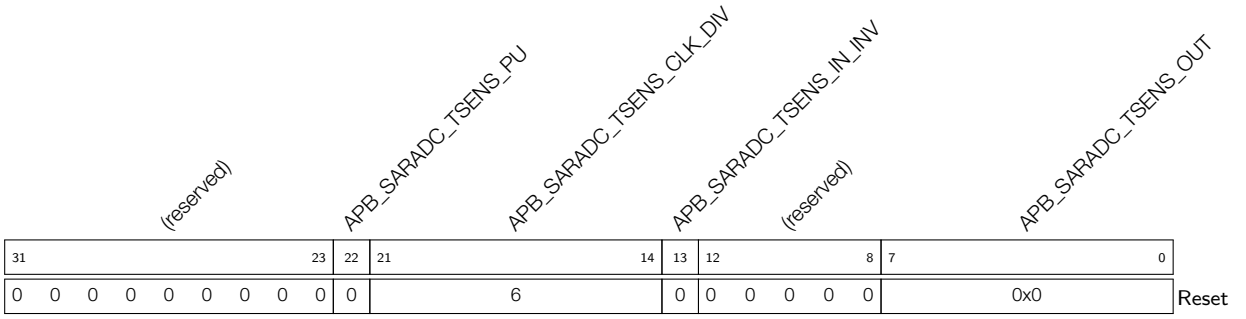
APB_SARADC_CLKM_DIV_NUM The integer part of ADC clock divider. Divider value = APB_SARADC_CLKM_DIV_NUM + APB_SARADC_CLKM_DIV_B/APB_SARADC_CLKM_DIV_A. (R/W)

APB_SARADC_CLKM_DIV_B The numerator value of fractional clock divider. (R/W)

APB_SARADC_CLKM_DIV_A The denominator value of fractional clock divider. (R/W)

APB_SARADC_CLK_SEL 0: Use APB_CLK as clock source, 1: use divided-down XTAL_CLK as clock source. (R/W)

Register 23.18. APB_SARADC_APB_TSENS_CTRL_REG (0x0058)



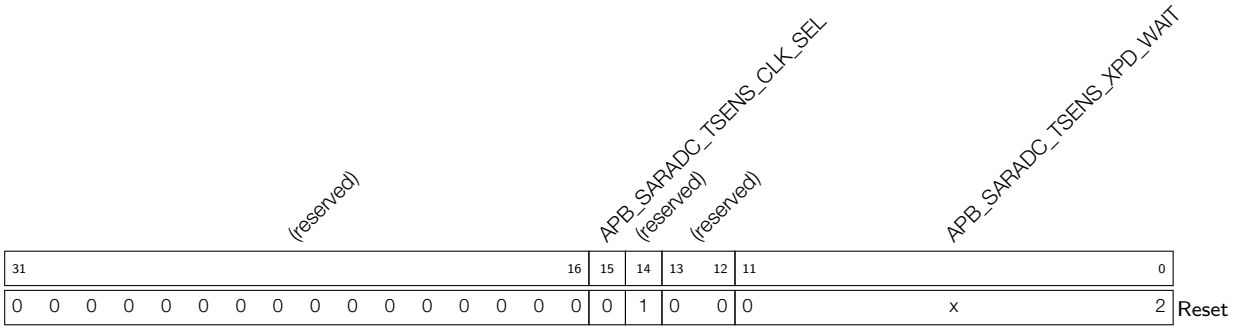
APB_SARADC_TSENS_OUT Temperature sensor data out. (RO)

APB_SARADC_TSENS_IN_INV Invert temperature sensor input value. (R/W)

APB_SARADC_TSENS_CLK_DIV Temperature sensor clock divider. (R/W)

APB_SARADC_TSENS_PU Temperature sensor power up. (R/W)

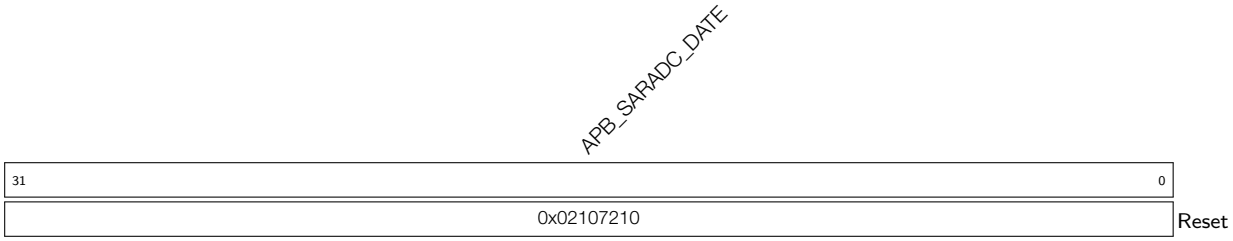
Register 23.19. APB_SARADC_APB_TSENS_CTRL2_REG (0x005C)



APB_SARADC_TSENS_XPD_WAIT The wait time before temperature sensor is powered up. (R/W)

APB_SARADC_TSENS_CLK_SEL Choose working clock for temperature sensor. 0: RC_FAST_CLK. 1: XTAL_CLK. (R/W)

Register 23.20. APB_SARADC_APB_CTRL_DATE_REG (0x03FC)



APB_SARADC_DATE Version register. (R/W)

24 Related Documentation and Resources

Related Documentation

- ESP8684 Series Datasheet – Specifications of the ESP8684 hardware.
- ESP8684 Hardware Design Guidelines – Guidelines on how to integrate the ESP8684 into your hardware product.
- *Certificates*
  https://espressif.com/en/support/documents/certificates
- *Documentation Updates and Update Notification Subscription*
  https://espressif.com/en/support/download/documents

Developer Zone

- ESP-IDF Programming Guide for ESP8684 – Extensive documentation for the ESP-IDF development framework.
- *ESP-IDF* and other development frameworks on GitHub.
  https://github.com/espressif
- *ESP32 BBS Forum* – Engineer-to-Engineer (E2E) Community for Espressif products where you can post questions, share knowledge, explore ideas, and help solve problems with fellow engineers.
  https://esp32.com/
- *The ESP Journal* – Best Practices, Articles, and Notes from Espressif folks.
  https://blog.espressif.com/
- See the tabs *SDKs and Demos*, *Apps*, *Tools*, *AT Firmware*.
  https://espressif.com/en/support/download/sdks-demos

Products

- *ESP8684 Series SoCs* – Browse through all ESP8684 SoCs.
  https://espressif.com/en/products/socs?id=ESP8684
- *ESP8684 Series Modules* – Browse through all ESP8684-based modules.
  https://espressif.com/en/products/modules?id=ESP8684
- *ESP8684 Series DevKits* – Browse through all ESP8684-based devkits.
  https://espressif.com/en/products/devkits?id=ESP8684
- *ESP Product Selector* – Find an Espressif hardware product suitable for your needs by comparing or applying filters.
  https://products.espressif.com/#/product-selector?language=en

Contact Us

- See the tabs *Sales Questions*, *Technical Enquiries*, *Circuit Schematic & PCB Design Review*, *Get Samples* (Online stores), *Become Our Supplier*, *Comments & Suggestions*.
  https://espressif.com/en/contact-us/sales-questions

Glossary

Abbreviations for Peripherals

AES	AES (Advanced Encryption Standard) Accelerator
BOOTCTRL	Chip Boot Control
DS	Digital Signature
DMA	DMA (Direct Memory Access) Controller
eFuse	eFuse Controller
HMAC	HMAC (Hash-based Message Authentication Code) Accelerator
I2C	I2C (Inter-Integrated Circuit) Controller
I2S	I2S (Inter-IC Sound) Controller
LEDC	LED Control PWM (Pulse Width Modulation)
MCPWM	Motor Control PWM (Pulse Width Modulation)
PCNT	Pulse Count Controller
RNG	Random Number Generator
RSA	RSA (Rivest Shamir Adleman) Accelerator
SDHOST	SD/MMC Host Controller
SHA	SHA (Secure Hash Algorithm) Accelerator
SPI	SPI (Serial Peripheral Interface) Controller
SYSTIMER	System Timer
TIMG	Timer Group
TWAI	Two-wire Automotive Interface
UART	UART (Universal Asynchronous Receiver-Transmitter) Controller
ULP Coprocessor	Ultra-low-power Coprocessor
USB OTG	USB On-The-Go
WDT	Watchdog Timers

Abbreviations 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
2022-10-27	v0.3	Added the following chapters: • 2 *GDMA Controller (GDMA)* • 9 *Low-power Management (RTC_CNTL)* • 20 *SPI Controller (SPI)* • 23 *On-Chip Sensor and Analog Signal Processing* Updated the following chapters: • 18 *Random Number Generator (RNG)* • 14 *Debug Assistant (ASSIST_DEBUG)*
2022-07-14	v0.2	Added the following chapters: • 4 *eFuse Controller (eFuse)* • 15 *ECC Hardware Accelerator (ECC)* Updated the following chapters: • 1 *ESP-RISC-V CPU* • 5 *IO MUX and GPIO Matrix (GPIO, IO MUX)* • 6 *Reset and Clock*
2022-05-18	v0.1	Preliminary release

www.espressif.com

Disclaimer and Copyright Notice