ESP32-C3

Technical Reference Manual

Pre-release v0.2

Espressif Systems

About This Manual

The **ESP32-C3 Technical Reference Manual** is addressed to application developers. The manual provides detailed and complete information on how to use the ESP32-C3 memory and peripherals.

For pin definition, electrical characteristics, and package information, please see *ESP32-C3 Datasheet*.

Document Updates

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

Revision History

For revision history of this document, please refer to the last page.

Documentation Change Notification

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

Certification

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

Contents

List of Tables

List of Figures

1 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.

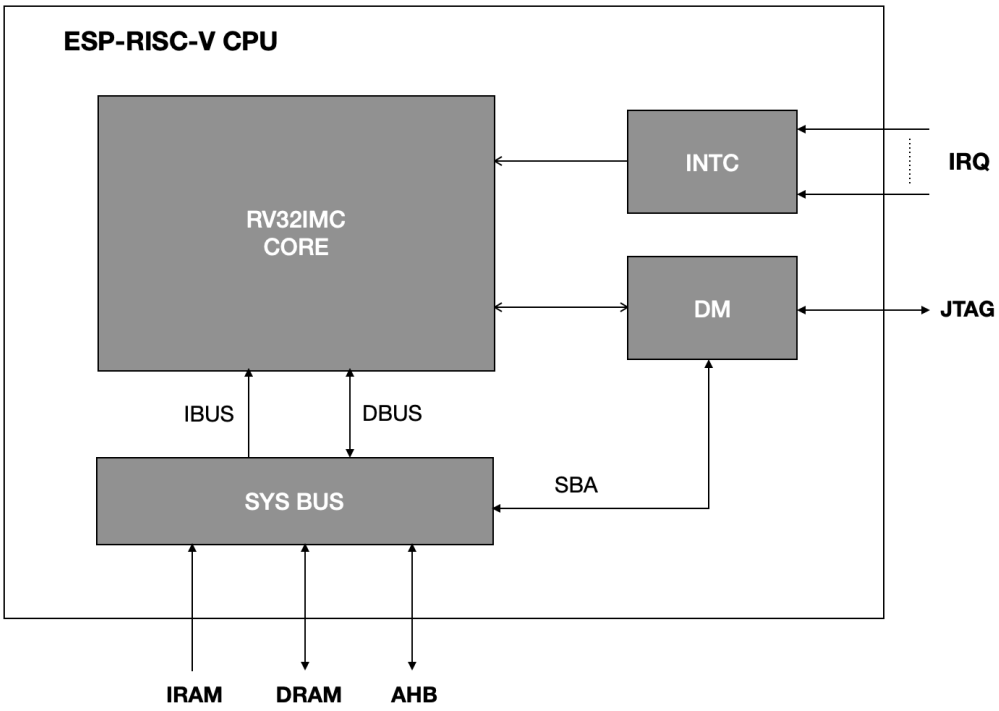


Figure 1-1. CPU Block Diagram

1.2 Features

- Operating clock frequency up to 160 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 8 breakpoints/watchpoints
- Physical memory protection (PMP) for up to 16 configurable 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
Physical Memory Protection (PMP) CSRs			
pmpcfg0	Physical memory protection configuration	0x3A0	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

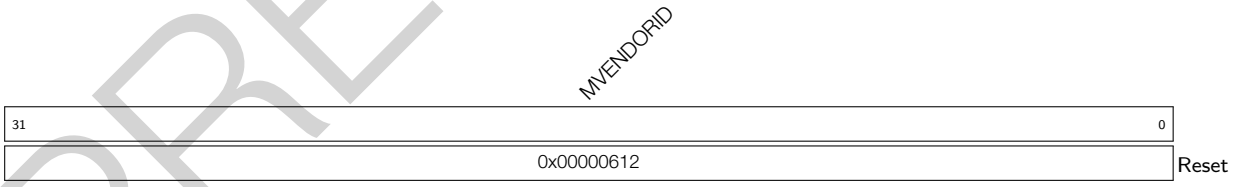
[3] External interrupt IDs reflected in mcause include even those IDs which have been reserved by RISC-V standard for core internal sources.

Name	Description	Address	Access
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 register	0x3B0	R/W
pmpaddr1	Physical memory protection address register	0x3B1	R/W
....			
pmpaddr15	Physical memory protection address register	0x3BF	R/W
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

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

1.4.2 Register Description

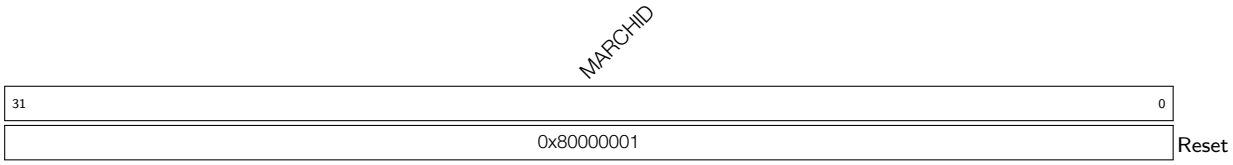
Register 1.1. mvendorid (0xF11)



MVENDORID Vendor ID. (RO)

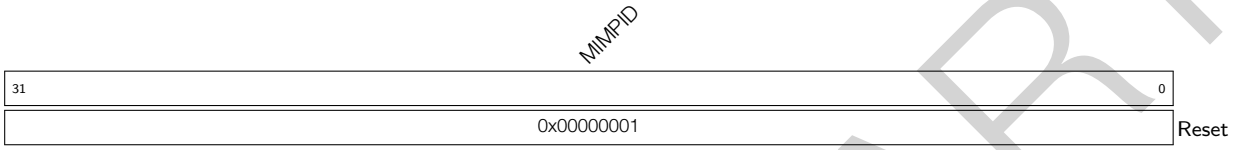
[4] These custom machine-mode CSRs have been implemented in the address space reserved by RISC-V standard for custom use

Register 1.2. marchid (0xF12)



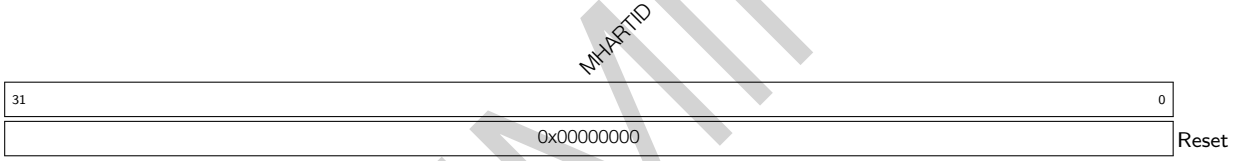
MARCHID Architecture ID. (RO)

Register 1.3. mimpid (0xF13)



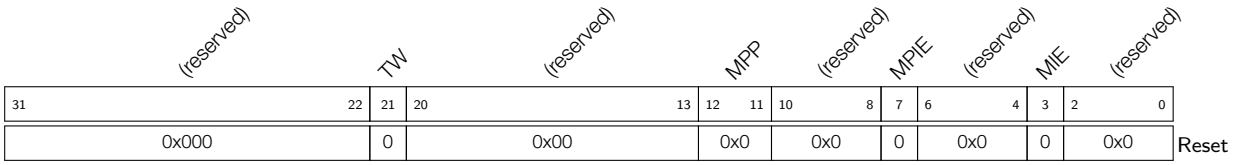
MIMPID Implementation ID. (RO)

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:

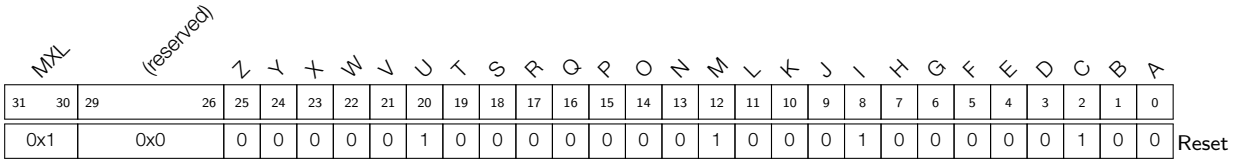
- 0x0: User mode
- 0x3: Machine mode

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 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)

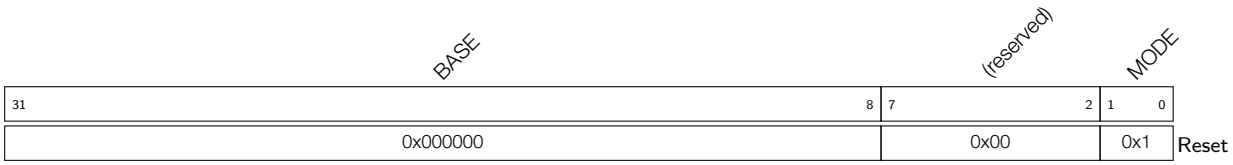
D Double-precision floating-point extension = 0. (RO)

C Compressed Extension = 1. (RO)

B Reserved = 0. (RO)

A Atomic Extension = 0. (RO)

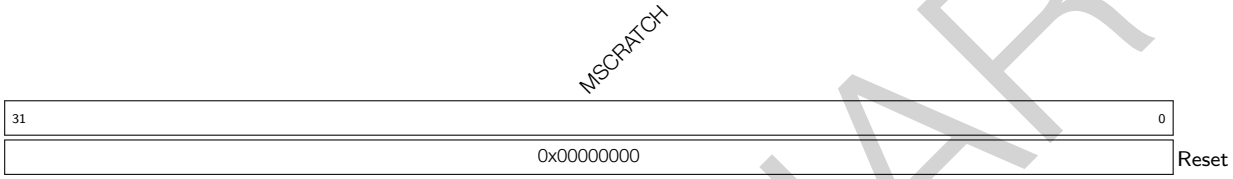
Register 1.7. mtvec (0x305)



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

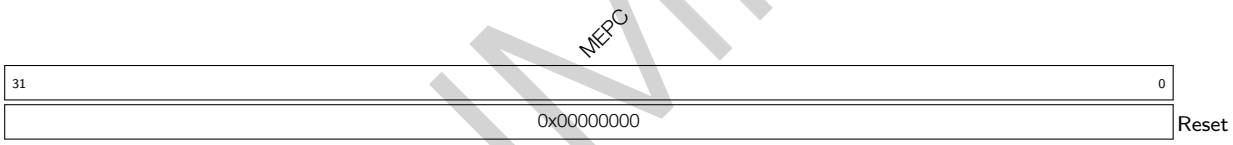
BASE Higher 24 bits of trap vector base address aligned to 256 bytes. (R/W)

Register 1.8. mscratch (0x340)



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

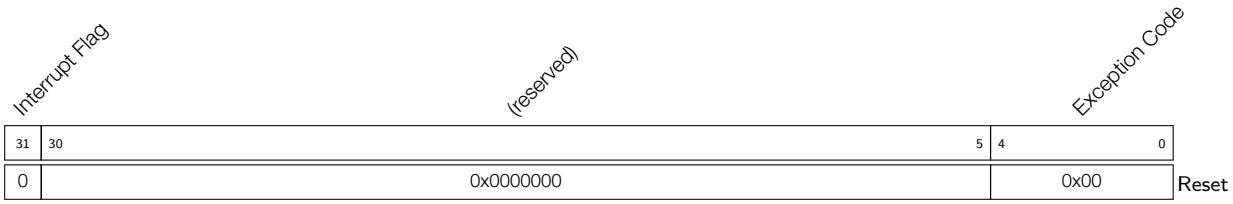
Register 1.9. mepc (0x341)



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

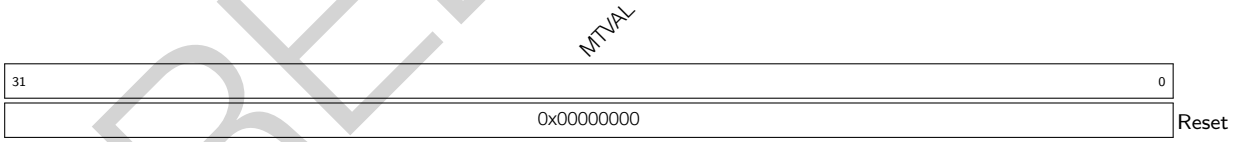
Note : Exception ID 0x0 (instruction access misaligned) is not present because CPU always masks the lowest bit of the address during instruction fetch.

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)

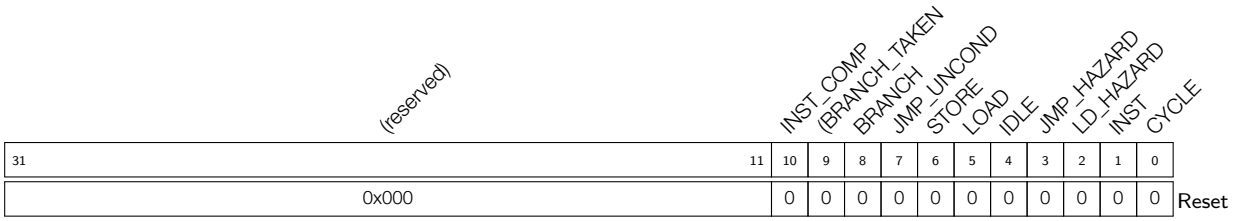
This is automatically updated with an exception dependent data which may be useful for handling that exception.

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)

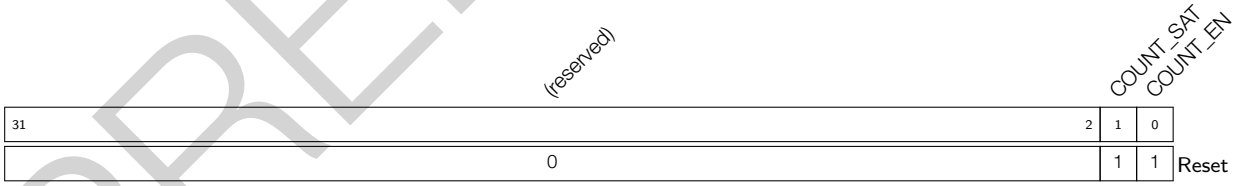
LD_HAZARD Count Load Hazards. (R/W)

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, then counter increments by one only.

Register 1.13. mpcmr (0x7E1)



COUNT_SAT Counter Saturation Control. (R/W)

Possible values:

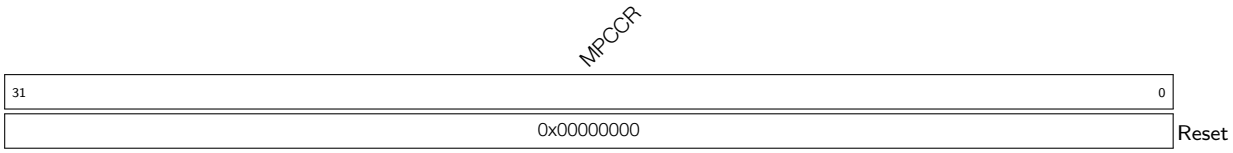
- 0: Overflow on maximum value
- 1: Halt on maximum value

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)

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 supports:

- 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 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 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 INT_PRIORITY_*n*_REG for a particular ID *n* in range (1-31).
   - Enabled interrupts with priorities zero or less than the threshold value in 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 INT_EIP_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 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 INT_ENABLE_REG and then toggling same bit in 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 INT_EIP_REG.

- If an interrupt is visible in INT_EIP_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 INT_EIP_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 INT_TYPE_REG
3. set the priority by writing a value to INT_PRIORITY_*n*_REG in range 1(lowest) to 15 (highest)

4. set the *n*th bit of 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 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 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 INT_EIP_REG
3. set/unset the *n*th bit of 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 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 3-4 in Chapter 3 *System and Memory*.

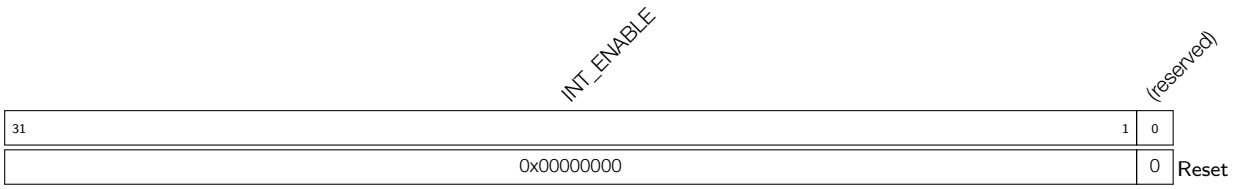
Name	Description	Address	Access
INT_ENABLE_REG	Enables assertion of interrupt to the CPU	0x0104	R/W
INT_TYPE_REG	Specify interrupt type as level/edge	0x0108	R/W
INT_CLEAR_REG	Write to clear "pulse" type interrupts	0x010C	R/W
INT_EIP_REG	External/peripheral interrupt pending status to CPU	0x0110	RO
INT_PRIORITY_1_REG	Priority setting for interrupt ID=1	0x0118	R/W
INT_PRIORITY_2_REG	Priority setting for interrupt ID=2	0x011C	R/W
INT_PRIORITY_3_REG	Priority setting for interrupt ID=3	0x0120	R/W

Name	Description	Address	Access
INT_PRIORITY_4_REG	Priority setting for interrupt ID=4	0x0124	R/W
INT_PRIORITY_5_REG	Priority setting for interrupt ID=5	0x0128	R/W
INT_PRIORITY_6_REG	Priority setting for interrupt ID=6	0x012C	R/W
INT_PRIORITY_7_REG	Priority setting for interrupt ID=7	0x0130	R/W
INT_PRIORITY_8_REG	Priority setting for interrupt ID=8	0x0134	R/W
INT_PRIORITY_9_REG	Priority setting for interrupt ID=9	0x0138	R/W
INT_PRIORITY_10_REG	Priority setting for interrupt ID=10	0x013C	R/W
INT_PRIORITY_11_REG	Priority setting for interrupt ID=11	0x0140	R/W
INT_PRIORITY_12_REG	Priority setting for interrupt ID=12	0x0144	R/W
INT_PRIORITY_13_REG	Priority setting for interrupt ID=13	0x0148	R/W
INT_PRIORITY_14_REG	Priority setting for interrupt ID=14	0x014C	R/W
INT_PRIORITY_15_REG	Priority setting for interrupt ID=15	0x0150	R/W
INT_PRIORITY_16_REG	Priority setting for interrupt ID=16	0x0154	R/W
INT_PRIORITY_17_REG	Priority setting for interrupt ID=17	0x0158	R/W
INT_PRIORITY_18_REG	Priority setting for interrupt ID=18	0x015C	R/W
INT_PRIORITY_19_REG	Priority setting for interrupt ID=19	0x0160	R/W
INT_PRIORITY_20_REG	Priority setting for interrupt ID=20	0x0164	R/W
INT_PRIORITY_21_REG	Priority setting for interrupt ID=21	0x0168	R/W
INT_PRIORITY_22_REG	Priority setting for interrupt ID=22	0x016C	R/W
INT_PRIORITY_23_REG	Priority setting for interrupt ID=23	0x0170	R/W
INT_PRIORITY_24_REG	Priority setting for interrupt ID=24	0x0174	R/W
INT_PRIORITY_25_REG	Priority setting for interrupt ID=25	0x0178	R/W
INT_PRIORITY_26_REG	Priority setting for interrupt ID=26	0x017C	R/W
INT_PRIORITY_27_REG	Priority setting for interrupt ID=27	0x0180	R/W
INT_PRIORITY_28_REG	Priority setting for interrupt ID=28	0x0184	R/W
INT_PRIORITY_29_REG	Priority setting for interrupt ID=29	0x0188	R/W
INT_PRIORITY_30_REG	Priority setting for interrupt ID=30	0x018C	R/W
INT_PRIORITY_31_REG	Priority setting for interrupt ID=31	0x0190	R/W
INT_THRESH_REG	Priority threshold setting for interrupt assertion to CPU	0x0194	R/W

1.5.5 Register Description

The addresses in this section are relative to Interrupt Controller base address provided in Table 3-4 in Chapter 3 *System and Memory*.

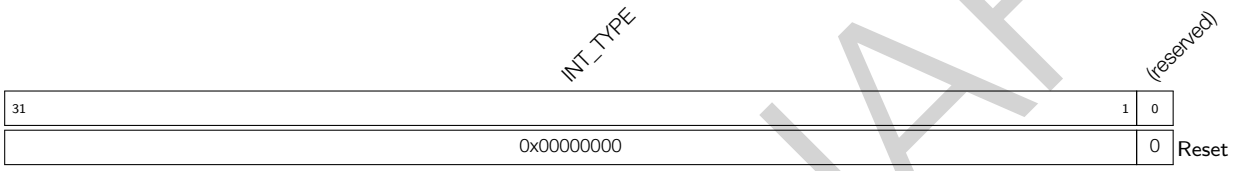
Register 1.15. INT_ENABLE_REG (0x0104)



INT_ENABLE[*n*] Setting *n*th bit enables assertion of *n*th interrupt to the CPU. (R/W)

- 0: Disabled;
- 1: Enabled;

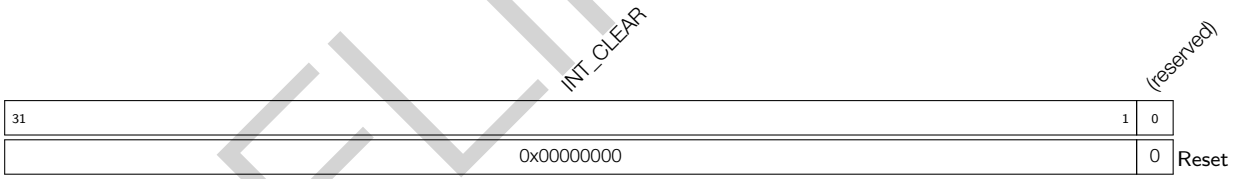
Register 1.16. INT_TYPE_REG (0x0108)



INT_TYPE[*n*] Setting *n*th bit enables capturing the rising edge of *n*th interrupt. (R/W)

- 0: Level type (signal level detection);
- 1: Pulse type (rising edge detection);

Register 1.17. INT_CLEAR_REG (0x010C)

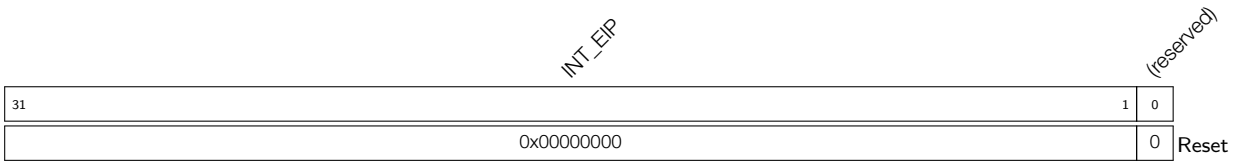


INT_CLEAR[*n*] Set *n*th bit to clear pending status of the *n*th interrupt. (R/W)

This is only useful for "pulse" type interrupts, since "level" type interrupts must be cleared at source. Note that the set bit must be manually toggled back to 0 afterwards.

- 0: Don't care;
- 1: Clear pending status;

Register 1.18. INT_EIP_REG (0x0110)

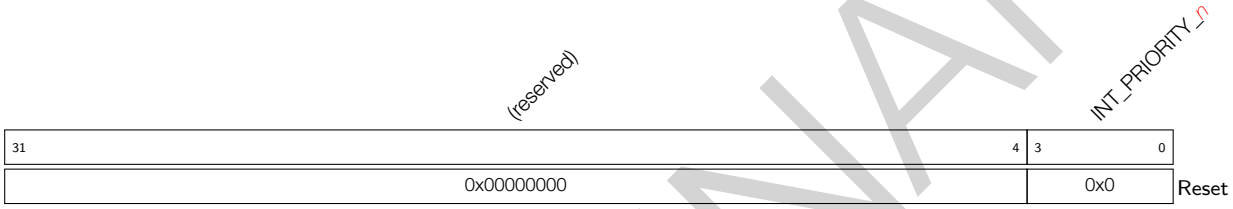


INT_EIP[*n*] Read *n*th bit to get the pending status of *n*th interrupt to CPU. (RO)

Only enabled and above threshold interrupts are reflected here.

- 0: Not pending
- 1: Pending

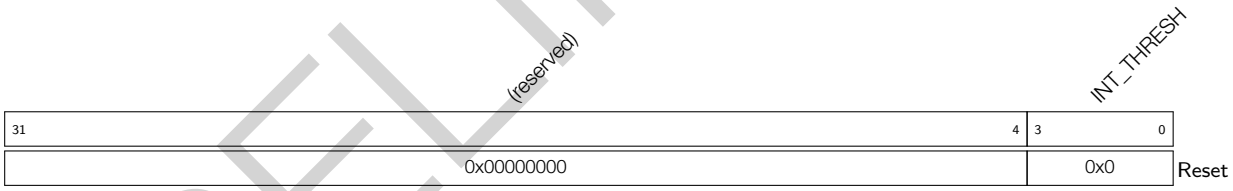
Register 1.19. INT_PRIORITY_*n*_REG (*n*: 1-31) (0x0114+4n*)**



INT_PRIORITY_*n* Writing a 4-bit value to *n*th register configures priority of *n*th interrupt. (R/W)

Note : Interrupts with 0 priority are masked regardless of threshold value.

Register 1.20. INT_THRESH_REG (0x0194)



INT_THRESH Writing a 4-bit value configures the global priority threshold for all interrupts. (R/W)

All interrupts with priority lower than the threshold are masked.

Note : Interrupts with 0 priority are masked regardless of threshold value.

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.

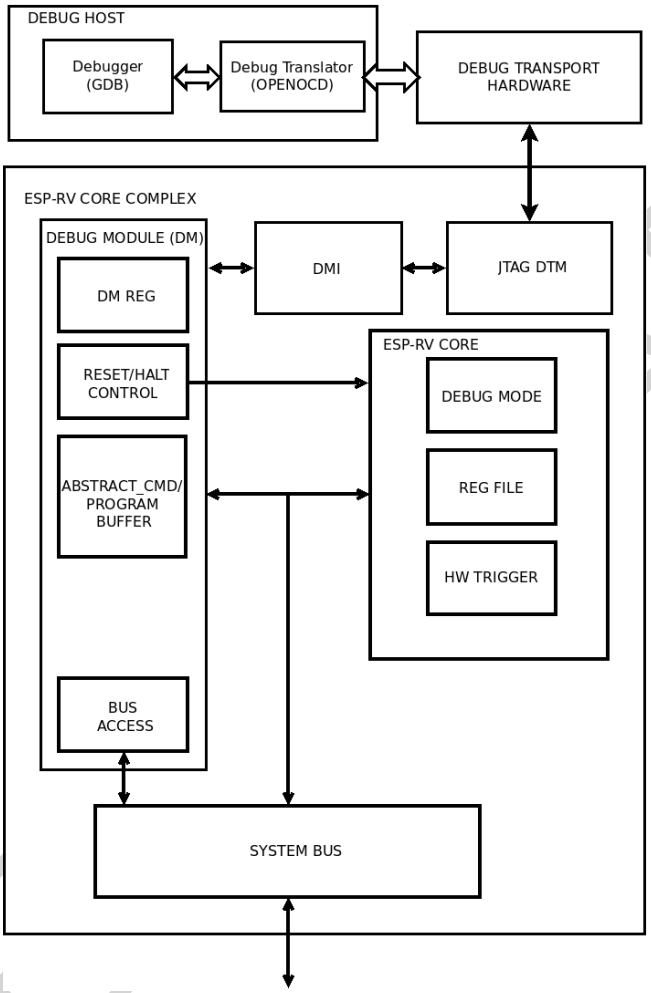


Figure 1-2. Debug System Overview

The user interacts with the Debug Host (eg. laptop), which is running a debugger (eg. gdb). The debugger communicates with a Debug Translator (eg. OpenOCD, which may include a hardware driver) to communicate with Debug Transport Hardware (eg. 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 8 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 supports below features.

- Provides necessary information about the implementation to the debugger.
- Allows the CPU core to be halted and resumed.
- CPU core registers (including CSR's) can be read/written by debugger.
- CPU can be debugged from the first instruction executed after reset.
- CPU core can be reset through debugger.
- CPU can be halted on software breakpoint (planted breakpoint instruction).
- Hardware single-stepping.
- Execute arbitrary instructions in the halted CPU by means of the program buffer. 16-word program buffer is supported.
- System bus access is supported through word aligned address access.
- Supports eight Hardware Triggers (can be used as breakpoints/watchpoints) as described in Section 1.7.

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

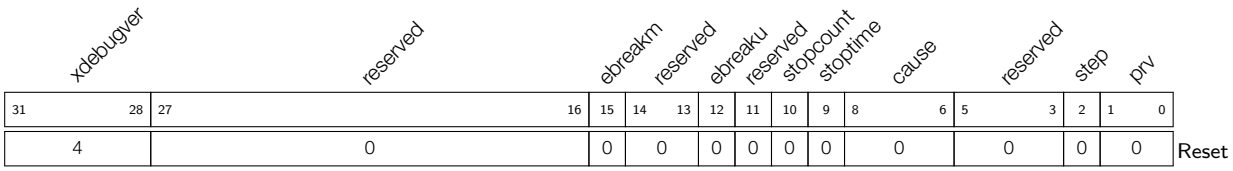
.

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 1.21. 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)

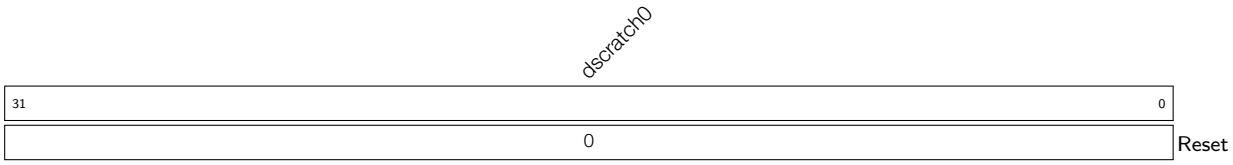
prv Contains the privilege level the core was operating in when Debug Mode was entered. A debugger can change this value to change the core's privilege level when exiting Debug Mode. Only **0x3** (machine mode) and **0x0**(user mode) are supported.

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

Register 1.22. 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 1.23. dscratch0 (0x7B2)



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

Register 1.24. 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 includes the following features:

- 8 independent trigger units
- each unit can be configured for matching the address of program counter or load-store accesses
- can preempt execution by causing breakpoint exception
- can halt execution and transfer control to debugger
- support 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 1-6) 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 1-6. 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.

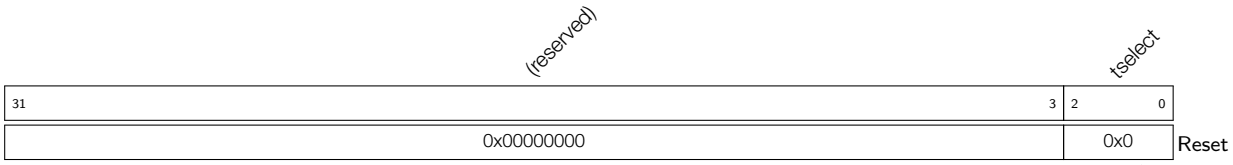
1.7.4 Register Summary

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

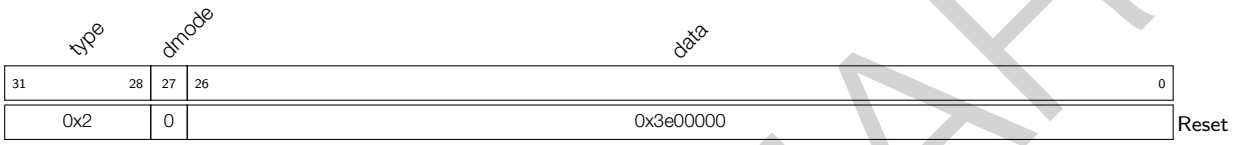
1.7.5 Register Description

Register 1.25. tselect (0x7A0)



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

Register 1.26. 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 *)

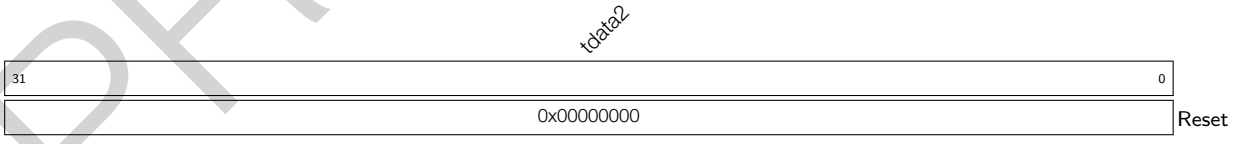
- 0: Both Debug and M-mode can write the tdata1 and tdata2 registers at the selected tselect.
- 1: Only Debug Mode can write the tdata1 and tdata2 registers at the selected tselect. Writes from other modes are ignored.

** 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.

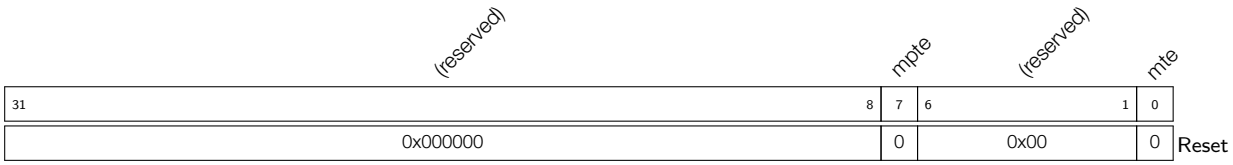
Register 1.27. tdata2 (0x7A2)



tdata2 Abstract tdata2 content. (R/W)

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

Register 1.28. 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 1.29. mcontrol (0x7A1)

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

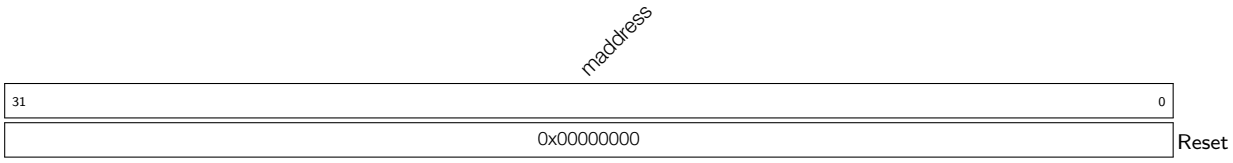
u Set this for enabling selected trigger to operate in user mode. (R/W)

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 1.30. 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. However it is not fully compliant to the Physical Memory Protection (PMP) description specified in **RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10**. Details of existing non-conformance are provided in next 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 PMP unit can be used to restrict access to physical memory. It supports 16 regions and a minimum granularity of 4 bytes. Below are the current non-conformance with PMP description from RISC-V Privilege specifications:

- Static priority i.e. overlapping regions are not supported
- Maximum supported NAPOT range is 1 GB

As per RISC-V Privilege specifications, PMP entries should be statically prioritized and the lowest-numbered PMP entry that matches any address byte of an access will determine whether that access succeeds or fails. This means, when any address matches more than one PMP entry i.e. overlapping regions among different PMP entries, lowest number PMP entry will decide whether such address access will succeed or fail.

However, RISC-V CPU PMP unit in ESP32-C3 does not implement static priority. So, software should make sure that all enabled PMP entries are programmed with unique regions i.e. without any region overlap among them. If software still tries to program multiple PMP entries with overlapping region having contradicting permissions, then access will succeed if it matches at least one of enabled PMP entries. An exception will be generated, if access matches none of the enabled PMP entries.

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 16 pmpcfgX and pmpaddrX registers (refer Register Summary).

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 pmpcfg and pmpaddr registers (refer Register Summary) 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 deafult. 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.

Name	Description	Address	Access
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 register.	0x3B0	R/W
pmpaddr1	Physical memory protection address register.	0x3B1	R/W
pmpaddr2	Physical memory protection address register.	0x3B2	R/W
pmpaddr3	Physical memory protection address register.	0x3B3	R/W
pmpaddr4	Physical memory protection address register.	0x3B4	R/W
pmpaddr5	Physical memory protection address register.	0x3B5	R/W
pmpaddr6	Physical memory protection address register.	0x3B6	R/W
pmpaddr7	Physical memory protection address register.	0x3B7	R/W
pmpaddr8	Physical memory protection address register.	0x3B8	R/W
pmpaddr9	Physical memory protection address register.	0x3B9	R/W
pmpaddr10	Physical memory protection address register.	0x3BA	R/W
pmpaddr11	Physical memory protection address register.	0x3BB	R/W
pmpaddr12	Physical memory protection address register.	0x3BC	R/W
pmpaddr13	Physical memory protection address register.	0x3BD	R/W
pmpaddr14	Physical memory protection address register.	0x3BE	R/W
pmpaddr15	Physical memory protection address register.	0x3BF	R/W

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 ESP32-C3 has six independent channels, i.e. three transmit channels and three receive channels. These six channels are shared by peripherals with GDMA feature, namely SPI2, UHCI0 (UART0/UART1), I2S, AES, SHA, and ADC. Users can assign the six channels to any of these peripherals. UART0 and UART1 use UHCI0 together.

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

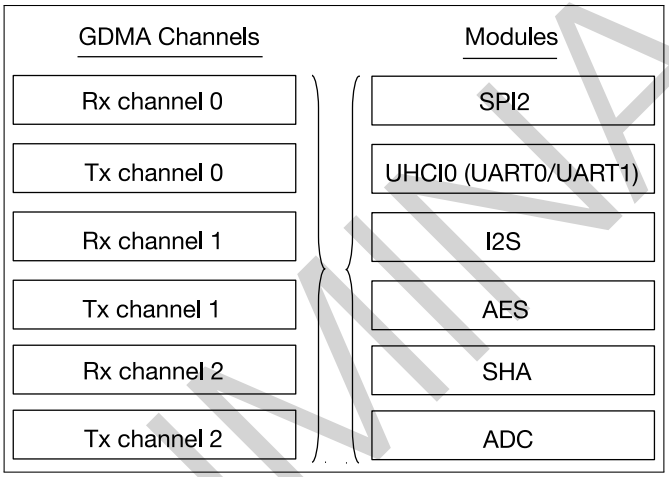


Figure 2-1. Modules with GDMA Feature and GDMA Channels

2.2 Features

The GDMA controller has the following features:

- AHB bus architecture
- 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 384 KB at most in internal RAM
- Three transmit channels and three receive channels
- Software-configurable selection of peripheral requesting its service
- Fixed channel priority and round-robin channel arbitration

2.3 Architecture

In ESP32-C3, 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.

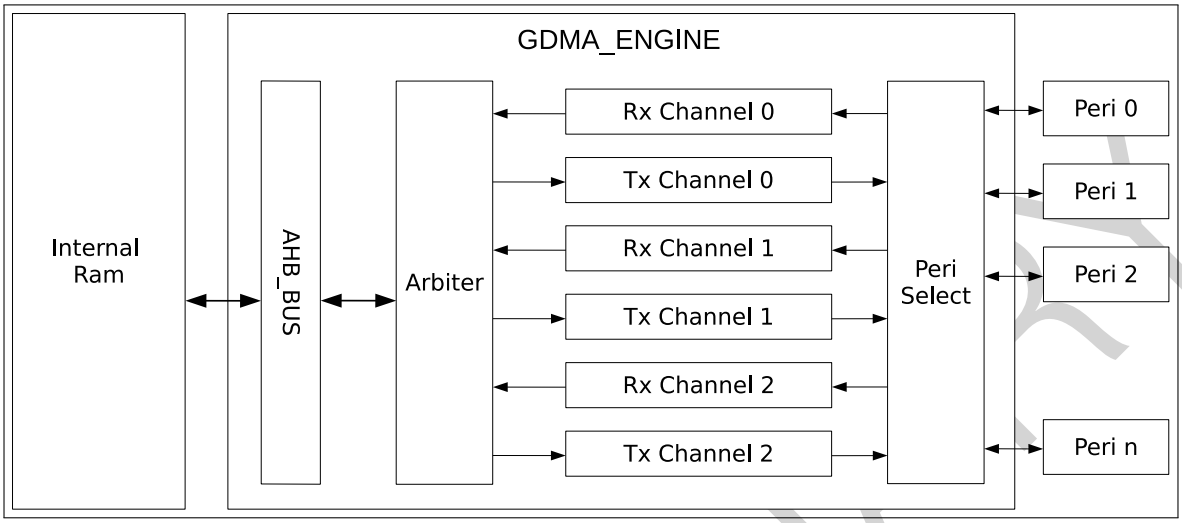


Figure 2-2. GDMA Engine Architecture

The GDMA has six independent channels, i.e. three transmit channels and three receive channels. 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*n* and inlink*n*, where *n* indicates the channel number (ranging from 0 to 2). 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 Linked List

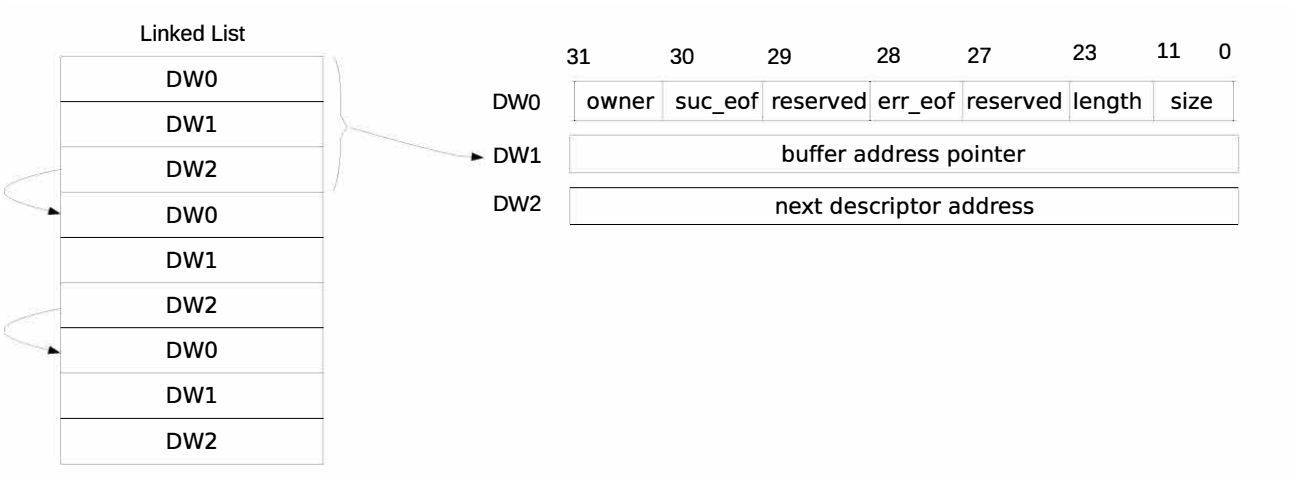


Figure 2-3. Structure of a Linked List

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_CH*n* is set to 1. Software can disable automatic clearing by hardware by setting GDMA_OUT_LOOP_TEST_CH*n* or GDMA_IN_LOOP_TEST_CH*n* bit. When 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 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.
  This bit is used only when UHCI0 uses GDMA to receive data. When an error is detected in the received packet, this bit in the receive descriptor is set to 1 by hardware.
- Reserved (DW0) [27:24]: Reserved.
- Length (DW0) [23:12]: Specifies the number of valid bytes in the buffer that this descriptor points to. This field in a transmit descriptor is written by software and indicates how many bytes can be read from the buffer; this field in a receive descriptor is written by hardware automatically and indicates how many valid bytes have been stored into the buffer.

- Size (DW0) [11:0]: Specifies the size of the buffer that this descriptor points to.
- 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 (suc_eof = 1), this value is 0. This field can only point to internal RAM.

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

2.4.2 Peripheral-to-Memory and Memory-to-Peripheral Data Transfer

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*n*, whereas a receive channel transfers data received by a peripheral to the specified memory location via an inlink*n*.

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 a channel is connected to a peripheral, the rest channels can not be connected to that peripheral.

Table 2-1. Selecting Peripherals via Register Configuration

GDMA_IN_PERI_SEL_CH*n* GDMA_OUT_PERI_SEL_CH*n*	Peripheral
0	SPI2
1	Reserved
2	UHCI0
3	I2S
4	Reserved
5	Reserved
6	AES
7	SHA
8	ADC

2.4.3 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_CH*n*, which connects the output of transmit channel *n* to the input of receive channel *n*. Note that a transmit channel is only connected to the receive channel with the same number (*n*).

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_CH*n* field with address of the first receive descriptor, and sets GDMA_INLINK_START_CH*n* bit to enable GDMA. When the GDMA controller transmits data, software loads an outlink, prepares data to be transmitted, configures GDMA_OUTLINK_ADDR_CH*n* field with address of the first transmit descriptor, and sets GDMA_OUTLINK_START_CH*n* bit to enable GDMA. GDMA_INLINK_START_CH*n* bit and GDMA_OUTLINK_START_CH*n* 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 by clearing the EOF bit of the final descriptor in the existing list and setting its next descriptor address pointer field (DW2) 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_CH*n* bit or GDMA_OUTLINK_RESTART_CH*n* 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.

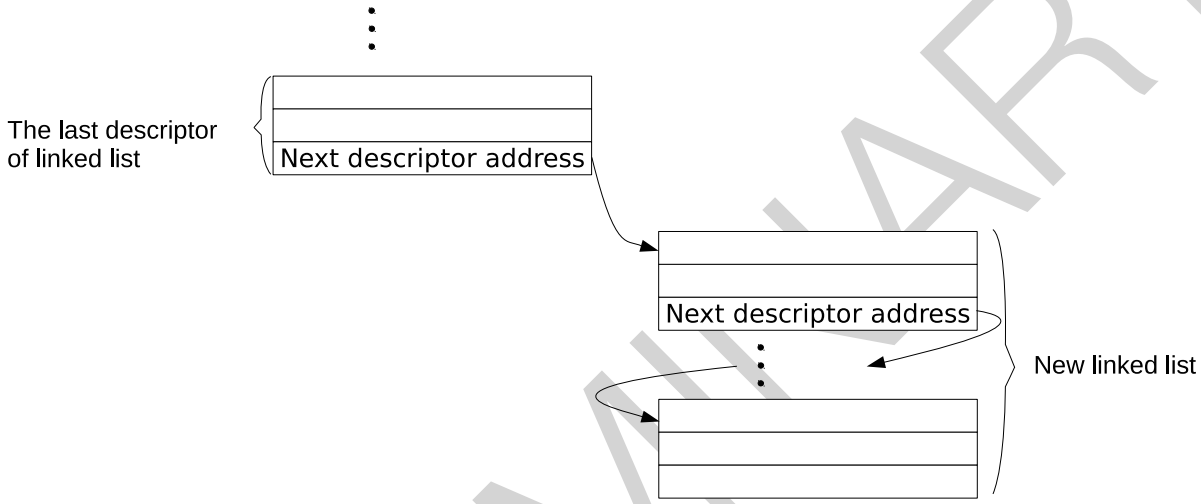


Figure 2-4. Relationship among Linked Lists

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_CH*n*_INT or GDMA_OUT_DSCR_ERR_CH*n*_INT), and the channel will get stuck and stop working.

The checks performed on descriptors are:

- Owner bit check when GDMA_IN_CHECK_OWNER_CH*n* or GDMA_OUT_CHECK_OWNER_CH*n* is set to 1. If the owner bit is 0, the buffer is accessed by the CPU. In this case, the owner bit fails the check. The owner bit will not be checked if GDMA_IN_CHECK_OWNER_CH*n* or GDMA_OUT_CHECK_OWNER_CH*n* is 0;
- Buffer address pointer (DW1) check. If the buffer address pointer points to 0x3FC80000 ~ 0x3FCDFFFF (please refer to Section 2.4.7), it passes the check.

After software detects a descriptor error interrupt, it must reset the corresponding channel, and enable GDMA by setting GDMA_OUTLINK_START_CH*n* or GDMA_INLINK_START_CH*n* bit.

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 file) flags to indicate the completion of data transfer.

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

Before the GDMA controller receives data, GDMA_IN_SUC_EOF_CH*n*_INT_ENA bit should be set to enable GDMA_IN_SUC_EOF_CH*n*_INT interrupt. If data has been received successfully, a GDMA_IN_SUC_EOF_CH*n*_INT interrupt is generated. In addition, when GDMA channel is connected to UHCI0, the GDMA controller also supports GDMA_IN_ERR_CH*n*_EOF_INT interrupt. This interrupt is enabled by setting GDMA_IN_ERR_EOF_CH*n*_INT_ENA bit, and it indicates that a data packet has been received with errors.

When detecting a GDMA_OUT_TOTAL_EOF_CH*n*_INT or a GDMA_IN_SUC_EOF_CH*n*_INT interrupt, software can record the value of GDMA_OUT_EOF_DES_ADDR_CH*n* or GDMA_IN_SUC_EOF_DES_ADDR_CH*n* 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 channels of GDMA can access 0x3FC80000 ~ 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 channels by setting GDMA_IN_DATA_BURST_EN_CH*n*, and enabled for transmit channels by setting GDMA_OUT_DATA_BURST_EN_CH*n*.

Table 2-2. Descriptor Field Alignment Requirements

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

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, so that requests from low-priority peripherals can be responded to.

2.4.9 Bandwidth

As an AHB master, the GDMA controller accesses memory via the AHB bus. Without regard to other AHB masters such as Wi-Fi, the total bandwidth supported by GDMA is calculated as:
All channels in burst mode: 8/5*fhclk MB/s;
All channels not in burst mode: 4/3*fhclk MB/s;
where fhclk is the frequency of AHB clock fixed at 80 MHz. The total bandwidth according to formulas above is listed in Table 2-3:

Table 2-3. Total Bandwidth Supported by GDMA

fpclk	All Channels NOT in Burst Mode	All Channels in Burst Mode
80 MHz	106.6 MB/s	128 MB/s

Please note that since the GDMA controller transfers data via linked list descriptors, the data transfer volume includes the number of bytes these descriptors have. The transfer efficiency corresponding to one descriptor is length/(length + 12), where length is the field in the descriptor, and 12 is the number of bytes a descriptor has. Therefore, applications with multiple linked list descriptors should increase length of each descriptor for higher transfer efficiency, which can be 99.7% at most.

When allocating bandwidth to a peripheral, software can estimate the bandwidth occupied by this peripheral according to:

T*(length + 12)/length

where T stands for the throughput of this peripheral.

2.5 GDMA Interrupts

- GDMA_OUT_TOTAL_EOF_CH*n*_INT: Triggered when all data corresponding to a linked list (including multiple descriptors) has been sent via transmit channel *n*.
- GDMA_IN_DSCR_EMPTY_CH*n*_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 *n*.
- GDMA_OUT_DSCR_ERR_CH*n*_INT: Triggered when an error is detected in a transmit descriptor on transmit channel *n*.
- GDMA_IN_DSCR_ERR_CH*n*_INT: Triggered when an error is detected in a receive descriptor on receive channel *n*.
- GDMA_OUT_EOF_CH*n*_INT: Triggered when EOF in a transmit descriptor is 1 and data corresponding to this descriptor has been sent via transmit channel *n*. If GDMA_OUT_EOF_MODE_CH*n* 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_CH*n* is 1, this interrupt is triggered when the last byte of data is taken from GDMA's transmit channel.

- GDMA_OUT_DONE_CH*n*_INT: Triggered when all data corresponding to a transmit descriptor has been sent via transmit channel *n*.
- GDMA_IN_ERR_EOF_CH*n*_INT: Triggered when an error is detected in the data packet received via receive channel *n*. This interrupt is used only for UHCI0 peripheral (UART0 or UART1).
- GDMA_IN_SUC_EOF_CH*n*_INT: Triggered when a data packet has been received via receive channel *n*.
- GDMA_IN_DONE_CH*n*_INT: Triggered when all data corresponding to a receive descriptor has been received via receive channel *n*.

2.6 Programming Procedures

2.6.1 Programming Procedures for GDMA's Transmit Channel

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

1. Set GDMA_OUT_RST_CH*n* 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_CH*n* with address of the first transmit descriptor;
3. Configure GDMA_PERI_OUT_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 2-1;
4. Set GDMA_OUTLINK_START_CH*n* to enable GDMA's transmit channel for data transfer;
5. Configure and enable the corresponding peripheral (SPI2, UHCI0 (UART 0 or UART 1), I2S, AES, SHA, and ADC). See details in individual chapters of these peripherals;
6. Wait for GDMA_OUT_EOF_CH*n*_INT interrupt, which indicates the completion of data transfer.

2.6.2 Programming Procedures for GDMA's Receive Channel

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

1. Set GDMA_IN_RST_CH*n* 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_CH*n* with address of the first receive descriptor;
3. Configure GDMA_PERI_IN_SEL_CH*n* with the value corresponding to the peripheral to be connected, as shown in Table 2-1;
4. Set GDMA_INLINK_START_CH*n* to enable GDMA receive channel for data transfer;
5. Configure and enable the corresponding peripheral (SPI2, UHCI0 (UART 0 or UART 1), I2S, AES, SHA, and ADC). See details in individual chapters of these peripherals;
6. Wait for GDMA_IN_SUC_EOF_CH*n*_INT interrupt, which indicates that a data packet has been received.

2.6.3 Programming Procedures 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_CH*n* 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_CH*n* 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_CH*n* with address of the first transmit descriptor;
4. Load an inlink, and configure GDMA_INLINK_ADDR_CH*n* with address of the first receive descriptor;
5. Set GDMA_MEM_TRANS_EN_CH*n* to enable memory-to-memory transfer;
6. Set GDMA_OUTLINK_START_CH*n* to enable GDMA's transmit channel for data transfer;
7. Set GDMA_INLINK_START_CH*n* to enable GDMA receive channel for data transfer;
8. Wait for GDMA_IN_SUC_EOF_CH*n*_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 base address provided in Table 3-4 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
GDMA_INT_RAW_CH1_REG	Raw status interrupt of Rx channel 1	0x0010	R/WTC/SS
GDMA_INT_ST_CH1_REG	Masked interrupt of Rx channel 1	0x0014	RO
GDMA_INT_ENA_CH1_REG	Interrupt enable bits of Rx channel 1	0x0018	R/W
GDMA_INT_CLR_CH1_REG	Interrupt clear bits of Rx channel 1	0x001C	WT
GDMA_INT_RAW_CH2_REG	Raw status interrupt of Rx channel 2	0x0020	R/WTC/SS
GDMA_INT_ST_CH2_REG	Masked interrupt of Rx channel 2	0x0024	RO
GDMA_INT_ENA_CH2_REG	Interrupt enable bits of Rx channel 2	0x0028	R/W
GDMA_INT_CLR_CH2_REG	Interrupt clear bits of Rx channel 2	0x002C	WT
Configuration Register			
GDMA_MISC_CONF_REG	MISC register	0x0044	R/W
Version Registers			
GDMA_DATE_REG	Version control register	0x0048	R/W
Configuration Registers			
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
GDMA_IN_CONF0_CH1_REG	Configuration register 0 of Rx channel 1	0x0130	R/W
GDMA_IN_CONF1_CH1_REG	Configuration register 1 of Rx channel 1	0x0134	R/W
GDMA_IN_POP_CH1_REG	Pop control register of Rx channel 1	0x013C	varies
GDMA_IN_LINK_CH1_REG	Link descriptor configuration and control register of Rx channel 1	0x0140	varies
GDMA_OUT_CONF0_CH1_REG	Configuration register 0 of Tx channel 1	0x0190	R/W
GDMA_OUT_CONF1_CH1_REG	Configuration register 1 of Tx channel 1	0x0194	R/W
GDMA_OUT_PUSH_CH1_REG	Push control register of Tx channel 1	0x019C	varies
GDMA_OUT_LINK_CH1_REG	Link descriptor configuration and control register of Tx channel 1	0x01A0	varies
GDMA_IN_CONF0_CH2_REG	Configuration register 0 of Rx channel 2	0x01F0	R/W

Name	Description	Address	Access
GDMA_IN_CONF1_CH2_REG	Configuration register 1 of Rx channel 2	0x01F4	R/W
GDMA_IN_POP_CH2_REG	Pop control register of Rx channel 2	0x01FC	varies
GDMA_IN_LINK_CH2_REG	Link descriptor configuration and control register of Rx channel 2	0x0200	varies
GDMA_OUT_CONF0_CH2_REG	Configuration register 0 of Tx channel 2	0x0250	R/W
GDMA_OUT_CONF1_CH2_REG	Configuration register 1 of Tx channel 2	0x0254	R/W
GDMA_OUT_PUSH_CH2_REG	Push control register of Tx channel 2	0x025C	varies
GDMA_OUT_LINK_CH2_REG	Link descriptor configuration and control register of Tx channel 2	0x0260	varies
Status Registers			
GDMA_INFIFO_STATUS_CH0_REG	Receive 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	Transmit 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
GDMA_OUT_DSCR_BF1_CH0_REG	The second-to-last inlink descriptor address of Tx channel 0	0x00F8	RO
GDMA_INFIFO_STATUS_CH1_REG	Receive FIFO status of Rx channel 1	0x0138	RO
GDMA_IN_STATE_CH1_REG	Receive status of Rx channel 1	0x0144	RO
GDMA_IN_SUC_EOF_DES_ADDR_CH1_REG	Inlink descriptor address when EOF occurs of Rx channel 1	0x0148	RO
GDMA_IN_ERR_EOF_DES_ADDR_CH1_REG	Inlink descriptor address when errors occur of Rx channel 1	0x014C	RO
GDMA_IN_DSCR_CH1_REG	Current inlink descriptor address of Rx channel 1	0x0150	RO
GDMA_IN_DSCR_BF0_CH1_REG	The last inlink descriptor address of Rx channel 1	0x0154	RO

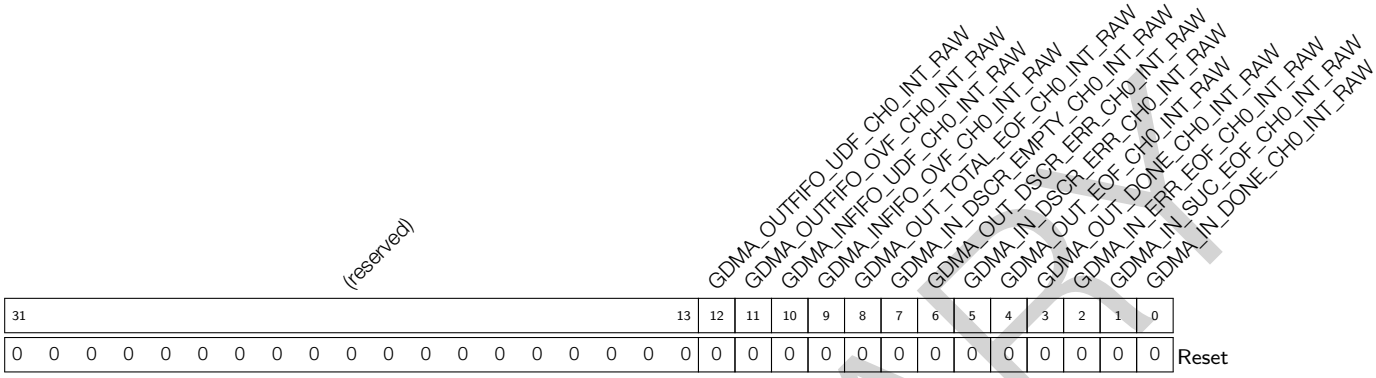
Name	Description	Address	Access
GDMA_IN_DSCR_BF1_CH1_REG	The second-to-last inlink descriptor address of Rx channel 1	0x0158	RO
GDMA_OUTFIFO_STATUS_CH1_REG	Transmit FIFO status of Tx channel 1	0x0198	RO
GDMA_OUT_STATE_CH1_REG	Transmit status of Tx channel 1	0x01A4	RO
GDMA_OUT_EOF_DES_ADDR_CH1_REG	Outlink descriptor address when EOF occurs of Tx channel 1	0x01A8	RO
GDMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	The last outlink descriptor address when EOF occurs of Tx channel 1	0x01AC	RO
GDMA_OUT_DSCR_CH1_REG	Current inlink descriptor address of Tx channel 1	0x01B0	RO
GDMA_OUT_DSCR_BF0_CH1_REG	The last inlink descriptor address of Tx channel 1	0x01B4	RO
GDMA_OUT_DSCR_BF1_CH1_REG	The second-to-last inlink descriptor address of Tx channel 1	0x01B8	RO
GDMA_INFIFO_STATUS_CH2_REG	Receive FIFO status of Rx channel 2	0x01F8	RO
GDMA_IN_STATE_CH2_REG	Receive status of Rx channel 2	0x0204	RO
GDMA_IN_SUC_EOF_DES_ADDR_CH2_REG	Inlink descriptor address when EOF occurs of Rx channel 2	0x0208	RO
GDMA_IN_ERR_EOF_DES_ADDR_CH2_REG	Inlink descriptor address when errors occur of Rx channel 2	0x020C	RO
GDMA_IN_DSCR_CH2_REG	Current inlink descriptor address of Rx channel 2	0x0210	RO
GDMA_IN_DSCR_BF0_CH2_REG	The last inlink descriptor address of Rx channel 2	0x0214	RO
GDMA_IN_DSCR_BF1_CH2_REG	The second-to-last inlink descriptor address of Rx channel 2	0x0218	RO
GDMA_OUTFIFO_STATUS_CH2_REG	Transmit FIFO status of Tx channel 2	0x0258	RO
GDMA_OUT_STATE_CH2_REG	Transmit status of Tx channel 2	0x0264	RO
GDMA_OUT_EOF_DES_ADDR_CH2_REG	Outlink descriptor address when EOF occurs of Tx channel 2	0x0268	RO
GDMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	The last outlink descriptor address when EOF occurs of Tx channel 2	0x026C	RO
GDMA_OUT_DSCR_CH2_REG	Current inlink descriptor address of Tx channel 2	0x0270	RO
GDMA_OUT_DSCR_BF0_CH2_REG	The last inlink descriptor address of Tx channel 2	0x0274	RO
GDMA_OUT_DSCR_BF1_CH2_REG	The second-to-last inlink descriptor address of Tx channel 2	0x0278	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
GDMA_IN_PRI_CH1_REG	Priority register of Rx channel 1	0x015C	R/W
GDMA_OUT_PRI_CH1_REG	Priority register of Tx channel 1	0x01BC	R/W
GDMA_IN_PRI_CH2_REG	Priority register of Rx channel 2	0x021C	R/W

Name	Description	Address	Access
GDMA_OUT_PRI_CH2_REG	Priority register of Tx channel 2	0x027C	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
GDMA_IN_PERI_SEL_CH1_REG	Peripheral selection of Rx channel 1	0x0160	R/W
GDMA_OUT_PERI_SEL_CH1_REG	Peripheral selection of Tx channel 1	0x01C0	R/W
GDMA_IN_PERI_SEL_CH2_REG	Peripheral selection of Rx channel 2	0x0220	R/W
GDMA_OUT_PERI_SEL_CH2_REG	Peripheral selection of Tx channel 2	0x0280	R/W

2.8 Registers

The addresses in this section are relative to GDMA base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Register 2.1. GDMA_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0000+16n*)**



GDMA_IN_DONE_CH*n*_INT_RAW The raw interrupt bit turns to high level when the last data pointed by one inlink descriptor has been received for Rx channel 0. (R/WTC/SS)

GDMA_IN_SUC_EOF_CH*n*_INT_RAW The raw interrupt bit turns to high level when the last data pointed by one inlink descriptor has been received for Rx channel 0. For UHCI0, the raw interrupt bit turns to high level when the last data pointed by one inlink descriptor has been received and no data error is detected for Rx channel 0. (R/WTC/SS)

GDMA_IN_ERR_EOF_CH*n*_INT_RAW The raw interrupt bit turns to high level when data error is detected only in the case that the peripheral is UHCI0 for Rx channel 0. For other peripherals, this raw interrupt is reserved. (R/WTC/SS)

GDMA_OUT_DONE_CH*n*_INT_RAW The raw interrupt bit turns to high level when the last data pointed by one outlink descriptor has been transmitted to peripherals for Tx channel 0. (R/WTC/SS)

GDMA_OUT_EOF_CH*n*_INT_RAW The raw interrupt bit turns to high level when the last data pointed by one outlink descriptor has been read from memory for Tx channel 0. (R/WTC/SS)

GDMA_IN_DSCR_ERR_CH*n*_INT_RAW The raw interrupt bit turns to high level when detecting inlink descriptor error, including owner error, the second and third word error of inlink descriptor for Rx channel 0. (R/WTC/SS)

GDMA_OUT_DSCR_ERR_CH*n*_INT_RAW The raw interrupt bit turns to high level when detecting outlink descriptor error, including owner error, the second and third word error of outlink descriptor for Tx channel 0. (R/WTC/SS)

Continued on the next page...

Register 2.1. GDMA_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0000+16n*)**

Continued from the previous page...

GDMA_IN_DSCR_EMPTY_CH*n*_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_CH*n*_INT_RAW The raw interrupt bit turns to high level when data corresponding a outlink (includes one link descriptor or few link descriptors) is transmitted out for Tx channel 0. (R/WTC/SS)

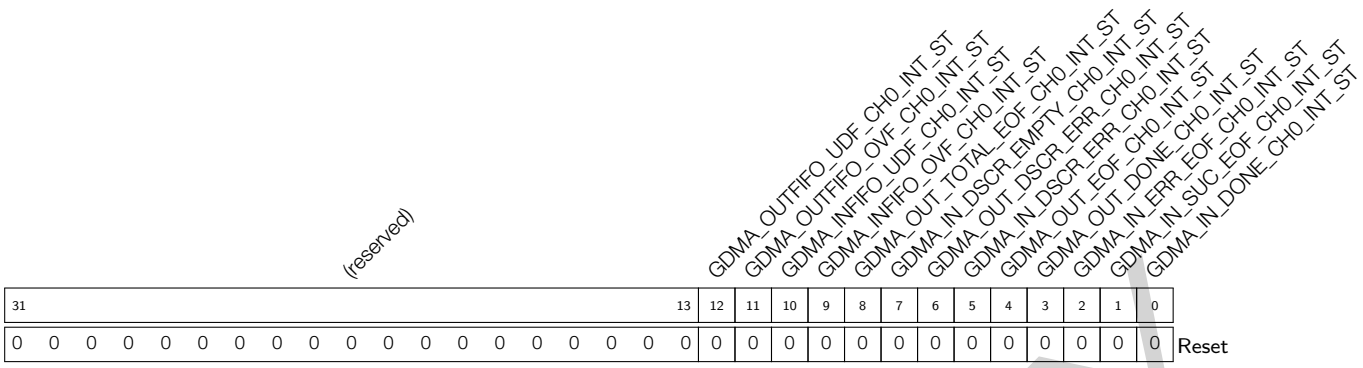
GDMA_INFIFO_OVF_CH*n*_INT_RAW This raw interrupt bit turns to high level when level 1 fifo of Rx channel 0 is overflow. (R/WTC/SS)

GDMA_INFIFO_UDF_CH*n*_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_CH*n*_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_CH*n*_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_CH*n*_REG (*n*: 0-2) (0x0004+16n*)**



GDMA_IN_DONE_CH*n*_INT_ST The raw interrupt status bit for the IN_DONE_CH_INT interrupt. (RO)

GDMA_IN_SUC_EOF_CH*n*_INT_ST The raw interrupt status bit for the IN_SUC_EOF_CH_INT interrupt. (RO)

GDMA_IN_ERR_EOF_CH*n*_INT_ST The raw interrupt status bit for the IN_ERR_EOF_CH_INT interrupt. (RO)

GDMA_OUT_DONE_CH*n*_INT_ST The raw interrupt status bit for the OUT_DONE_CH_INT interrupt. (RO)

GDMA_OUT_EOF_CH*n*_INT_ST The raw interrupt status bit for the OUT_EOF_CH_INT interrupt. (RO)

GDMA_IN_DSCR_ERR_CH*n*_INT_ST The raw interrupt status bit for the IN_DSCR_ERR_CH_INT interrupt. (RO)

GDMA_OUT_DSCR_ERR_CH*n*_INT_ST The raw interrupt status bit for the OUT_DSCR_ERR_CH_INT interrupt. (RO)

GDMA_IN_DSCR_EMPTY_CH*n*_INT_ST The raw interrupt status bit for the IN_DSCR_EMPTY_CH_INT interrupt. (RO)

GDMA_OUT_TOTAL_EOF_CH*n*_INT_ST The raw interrupt status bit for the OUT_TOTAL_EOF_CH_INT interrupt. (RO)

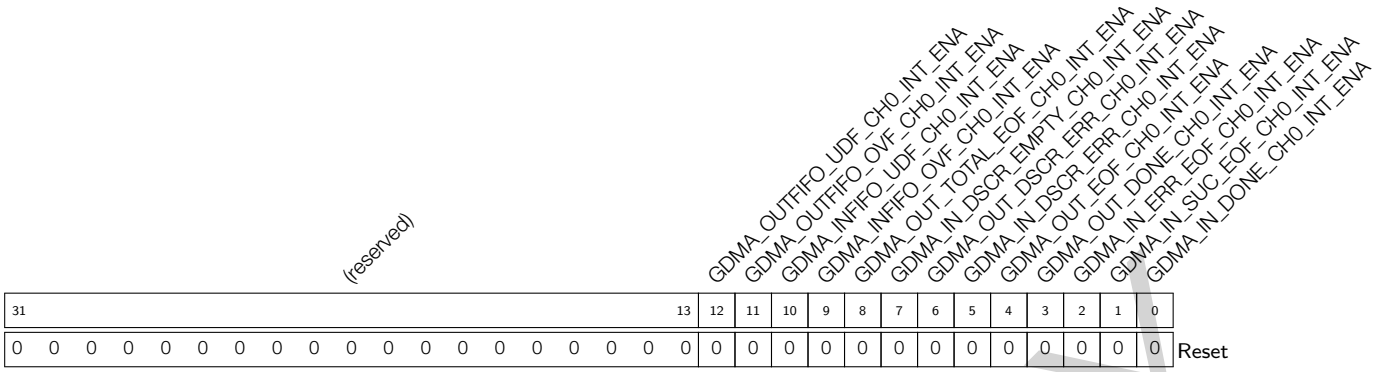
GDMA_INFIFO_OVF_CH*n*_INT_ST The raw interrupt status bit for the INFIFO_OVF_L1_CH_INT interrupt. (RO)

GDMA_INFIFO_UDF_CH*n*_INT_ST The raw interrupt status bit for the INFIFO_UDF_L1_CH_INT interrupt. (RO)

GDMA_OUTFIFO_OVF_CH*n*_INT_ST The raw interrupt status bit for the OUTFIFO_OVF_L1_CH_INT interrupt. (RO)

GDMA_OUTFIFO_UDF_CH*n*_INT_ST The raw interrupt status bit for the OUTFIFO_UDF_L1_CH_INT interrupt. (RO)

Register 2.3. GDMA_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0008+16n*)**



GDMA_IN_DONE_CH*n*_INT_ENA The interrupt enable bit for the IN_DONE_CH_INT interrupt. (R/W)

GDMA_IN_SUC_EOF_CH*n*_INT_ENA The interrupt enable bit for the IN_SUC_EOF_CH_INT interrupt. (R/W)

GDMA_IN_ERR_EOF_CH*n*_INT_ENA The interrupt enable bit for the IN_ERR_EOF_CH_INT interrupt. (R/W)

GDMA_OUT_DONE_CH*n*_INT_ENA The interrupt enable bit for the OUT_DONE_CH_INT interrupt. (R/W)

GDMA_OUT_EOF_CH*n*_INT_ENA The interrupt enable bit for the OUT_EOF_CH_INT interrupt. (R/W)

GDMA_IN_DSCR_ERR_CH*n*_INT_ENA The interrupt enable bit for the IN_DSCR_ERR_CH_INT interrupt. (R/W)

GDMA_OUT_DSCR_ERR_CH*n*_INT_ENA The interrupt enable bit for the OUT_DSCR_ERR_CH_INT interrupt. (R/W)

GDMA_IN_DSCR_EMPTY_CH*n*_INT_ENA The interrupt enable bit for the IN_DSCR_EMPTY_CH_INT interrupt. (R/W)

GDMA_OUT_TOTAL_EOF_CH*n*_INT_ENA The interrupt enable bit for the OUT_TOTAL_EOF_CH_INT interrupt. (R/W)

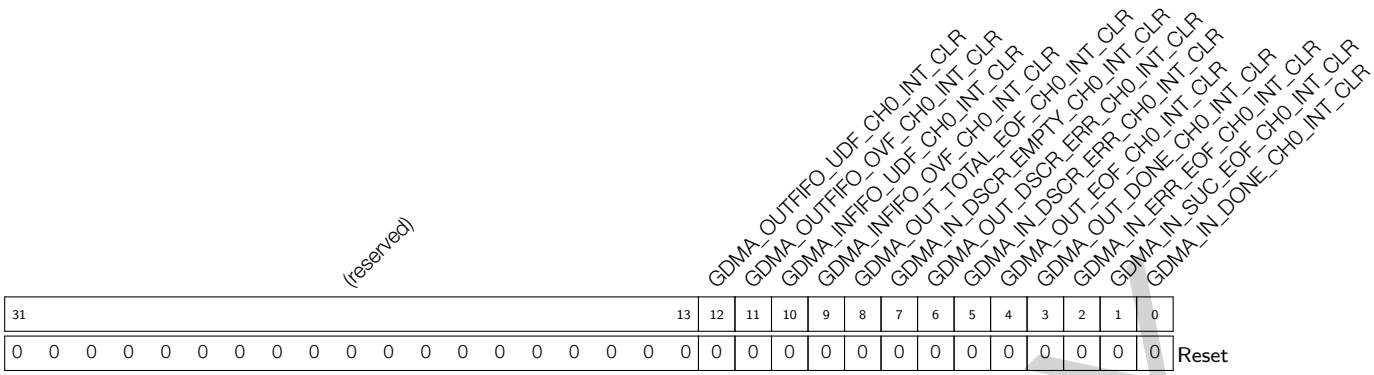
GDMA_INFIFO_OVF_CH*n*_INT_ENA The interrupt enable bit for the INFIFO_OVF_L1_CH_INT interrupt. (R/W)

GDMA_INFIFO_UDF_CH*n*_INT_ENA The interrupt enable bit for the INFIFO_UDF_L1_CH_INT interrupt. (R/W)

GDMA_OUTFIFO_OVF_CH*n*_INT_ENA The interrupt enable bit for the OUTFIFO_OVF_L1_CH_INT interrupt. (R/W)

GDMA_OUTFIFO_UDF_CH*n*_INT_ENA The interrupt enable bit for the OUTFIFO_UDF_L1_CH_INT interrupt. (R/W)

Register 2.4. GDMA_INT_CLR_CH*n*_REG (*n*: 0-2) (0x000C+16n*)**



GDMA_IN_DONE_CH*n*_INT_CLR Set this bit to clear the IN_DONE_CH_INT interrupt. (WT)

GDMA_IN_SUC_EOF_CH*n*_INT_CLR Set this bit to clear the IN_SUC_EOF_CH_INT interrupt. (WT)

GDMA_IN_ERR_EOF_CH*n*_INT_CLR Set this bit to clear the IN_ERR_EOF_CH_INT interrupt. (WT)

GDMA_OUT_DONE_CH*n*_INT_CLR Set this bit to clear the OUT_DONE_CH_INT interrupt. (WT)

GDMA_OUT_EOF_CH*n*_INT_CLR Set this bit to clear the OUT_EOF_CH_INT interrupt. (WT)

GDMA_IN_DSCR_ERR_CH*n*_INT_CLR Set this bit to clear the IN_DSCR_ERR_CH_INT interrupt. (WT)

GDMA_OUT_DSCR_ERR_CH*n*_INT_CLR Set this bit to clear the OUT_DSCR_ERR_CH_INT interrupt. (WT)

GDMA_IN_DSCR_EMPTY_CH*n*_INT_CLR Set this bit to clear the IN_DSCR_EMPTY_CH_INT interrupt. (WT)

GDMA_OUT_TOTAL_EOF_CH*n*_INT_CLR Set this bit to clear the OUT_TOTAL_EOF_CH_INT interrupt. (WT)

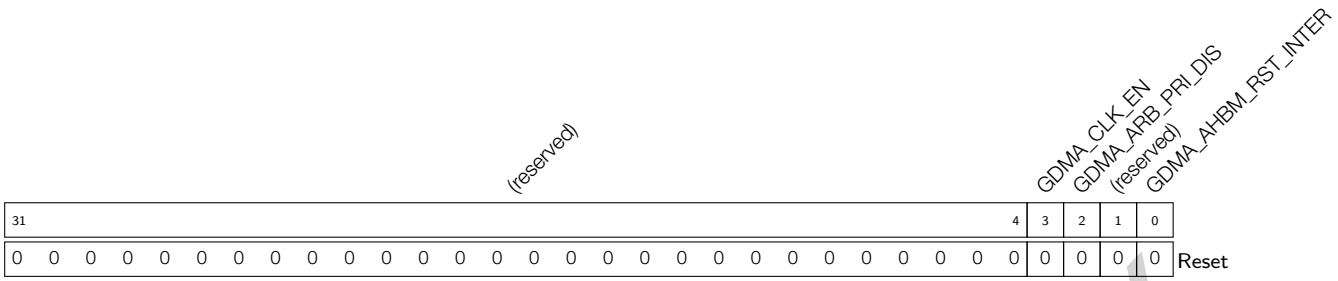
GDMA_INFIFO_OVF_CH*n*_INT_CLR Set this bit to clear the INFIFO_OVF_L1_CH_INT interrupt. (WT)

GDMA_INFIFO_UDF_CH*n*_INT_CLR Set this bit to clear the INFIFO_UDF_L1_CH_INT interrupt. (WT)

GDMA_OUTFIFO_OVF_CH*n*_INT_CLR Set this bit to clear the OUTFIFO_OVF_L1_CH_INT interrupt. (WT)

GDMA_OUTFIFO_UDF_CH*n*_INT_CLR Set this bit to clear the OUTFIFO_UDF_L1_CH_INT interrupt. (WT)

Register 2.5. GDMA_MISC_CONF_REG (0x0044)

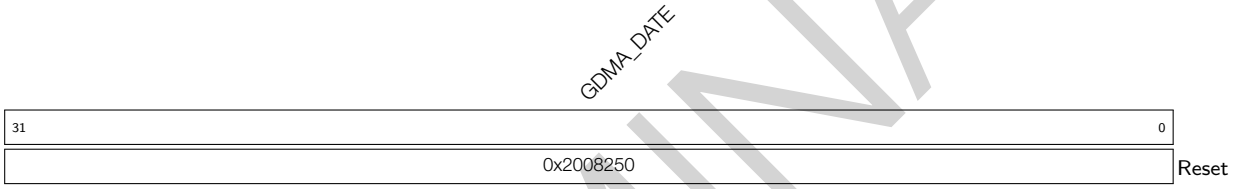


GDMA_AHBM_RST_INTER Set this bit, then clear this bit to reset the internal ahb FSM. (R/W)

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

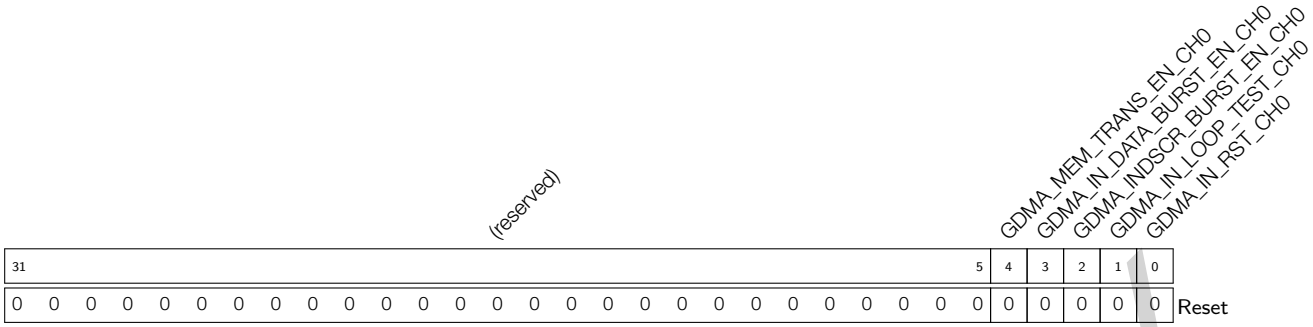
GDMA_CLK_EN reg_clk_en (R/W)

Register 2.6. GDMA_DATE_REG (0x0048)



GDMA_DATE register version. (R/W)

Register 2.7. GDMA_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0070+192n*)**



GDMA_IN_RST_CH*n* This bit is used to reset DMA channel 0 Rx FSM and Rx FIFO pointer. (R/W)

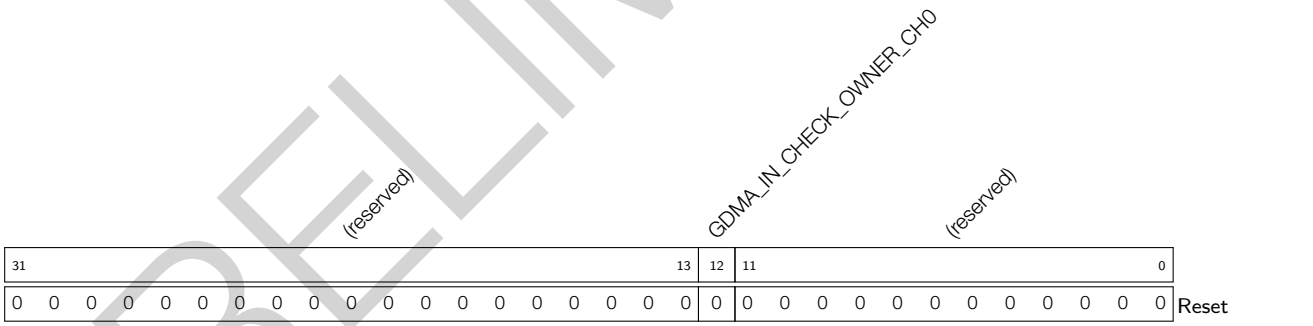
GDMA_IN_LOOP_TEST_CH*n* This bit is used to fill the owner bit of inlink descriptor by hardware of inlink descriptor. (R/W)

GDMA_INDSCR_BURST_EN_CH*n* Set this bit to 1 to enable INCR burst transfer for Rx channel 0 reading link descriptor when accessing internal SRAM. (R/W)

GDMA_IN_DATA_BURST_EN_CH*n* Set this bit to 1 to enable INCR burst transfer for Rx channel 0 receiving data when accessing internal SRAM. (R/W)

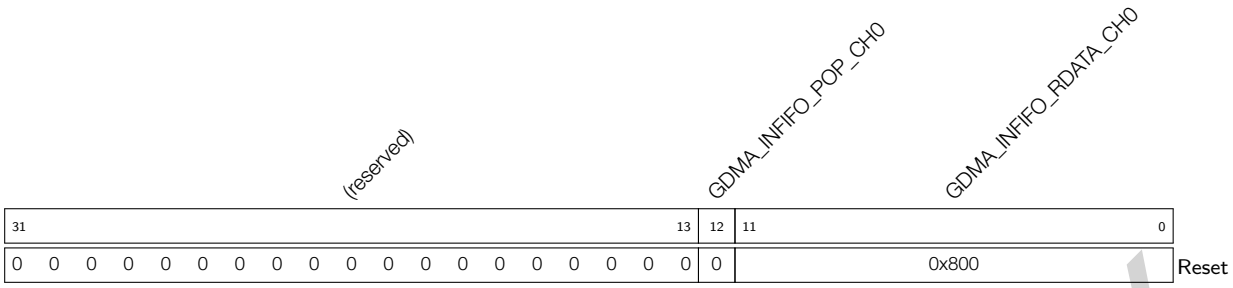
GDMA_MEM_TRANS_EN_CH*n* Set this bit 1 to enable automatic transmitting data from memory to memory via DMA. (R/W)

Register 2.8. GDMA_IN_CONF1_CH*n*_REG (*n*: 0-2) (0x0074+192n*)**



GDMA_IN_CHECK_OWNER_CH*n* Set this bit to enable checking the owner attribute of the link descriptor. (R/W)

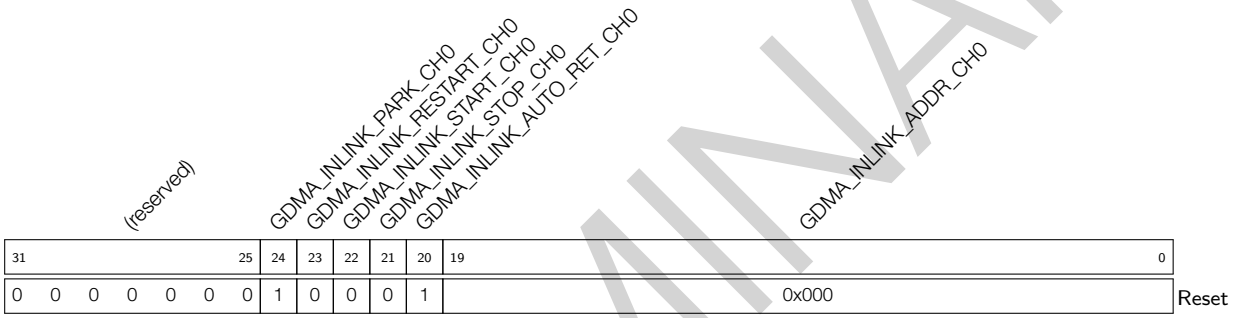
Register 2.9. GDMA_IN_POP_CH*n*_REG (*n*: 0-2) (0x007C+192n*)**



GDMA_INFIFO_RDATA_CH*n* This register stores the data popping from DMA FIFO. (RO)

GDMA_INFIFO_POP_CH*n* Set this bit to pop data from DMA FIFO. (R/W/SC)

Register 2.10. GDMA_IN_LINK_CH*n*_REG (*n*: 0-2) (0x0080+192n*)**



GDMA_INLINK_ADDR_CH*n* This register stores the 20 least significant bits of the first inlink descriptor's address. (R/W)

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

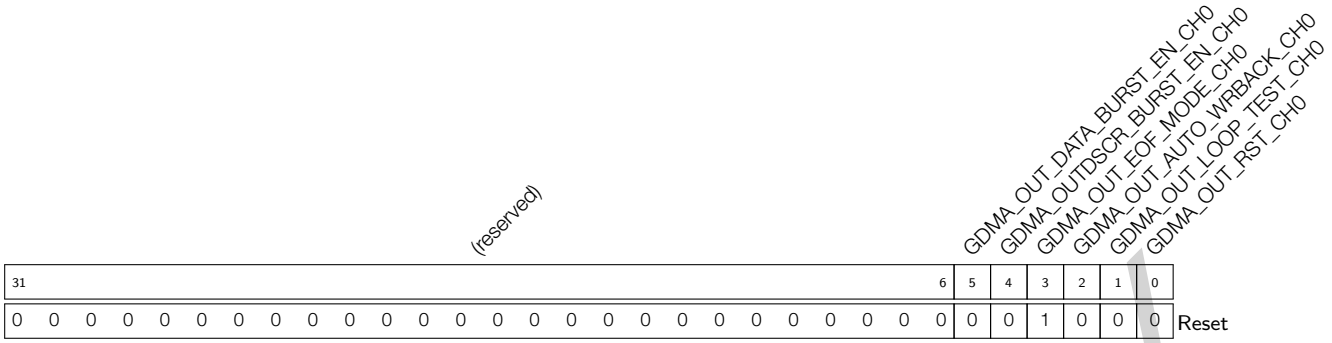
GDMA_INLINK_STOP_CH*n* Set this bit to stop dealing with the inlink descriptors. (R/W/SC)

GDMA_INLINK_START_CH*n* Set this bit to start dealing with the inlink descriptors. (R/W/SC)

GDMA_INLINK_RESTART_CH*n* Set this bit to mount a new inlink descriptor. (R/W/SC)

GDMA_INLINK_PARK_CH*n* 1: the inlink descriptor's FSM is in idle state. 0: the inlink descriptor's FSM is working. (RO)

Register 2.11. GDMA_OUT_CONF0_CH*n*_REG (*n*: 0-2) (0x00D0+192n*)**



GDMA_OUT_RST_CH*n* This bit is used to reset DMA channel 0 Tx FSM and Tx FIFO pointer. (R/W)

GDMA_OUT_LOOP_TEST_CH*n* reserved (R/W)

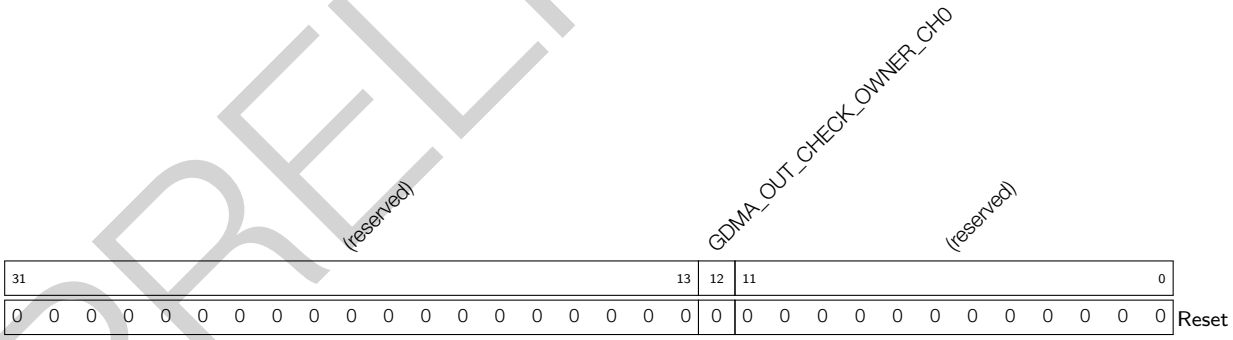
GDMA_OUT_AUTO_WRBACK_CH*n* Set this bit to enable automatic outlink-writeback when all the data in tx buffer has been transmitted. (R/W)

GDMA_OUT_EOF_MODE_CH*n* 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 DMA (R/W)

GDMA_OUTDSCR_BURST_EN_CH*n* Set this bit to 1 to enable INCR burst transfer for Tx channel 0 reading link descriptor when accessing internal SRAM. (R/W)

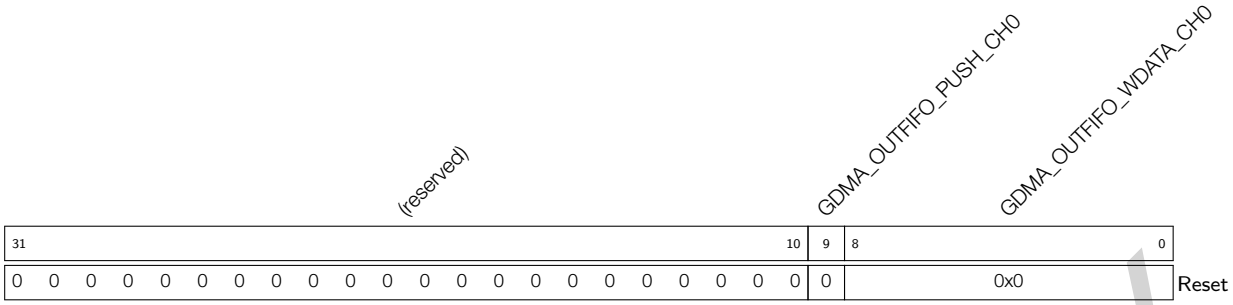
GDMA_OUT_DATA_BURST_EN_CH*n* Set this bit to 1 to enable INCR burst transfer for Tx channel 0 transmitting data when accessing internal SRAM. (R/W)

Register 2.12. GDMA_OUT_CONF1_CH*n*_REG (*n*: 0-2) (0x00D4+192n*)**



GDMA_OUT_CHECK_OWNER_CH*n* Set this bit to enable checking the owner attribute of the link descriptor. (R/W)

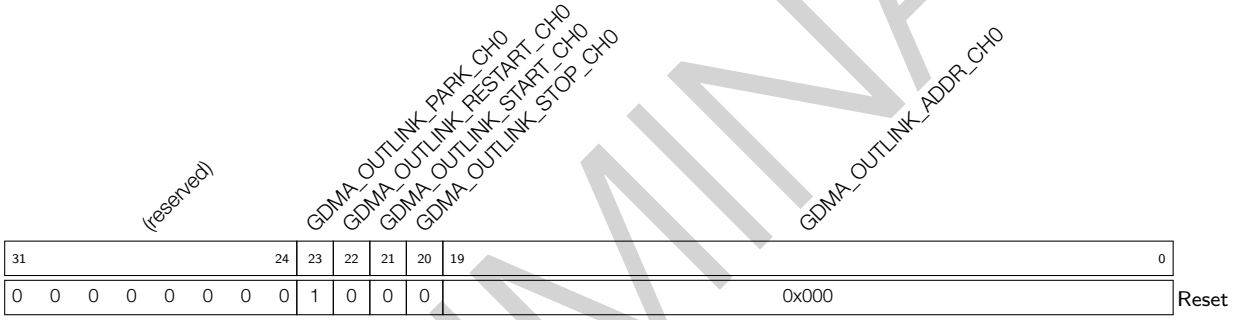
Register 2.13. GDMA_OUT_PUSH_CH*n*_REG (*n*: 0-2) (0x00DC+192n*)**



GDMA_OUTFIFO_WDATA_CH*n* This register stores the data that need to be pushed into DMA FIFO. (R/W)

GDMA_OUTFIFO_PUSH_CH*n* Set this bit to push data into DMA FIFO. (R/W/SC)

Register 2.14. GDMA_OUT_LINK_CH*n*_REG (*n*: 0-2) (0x00E0+192n*)**



GDMA_OUTLINK_ADDR_CH*n* This register stores the 20 least significant bits of the first outlink descriptor's address. (R/W)

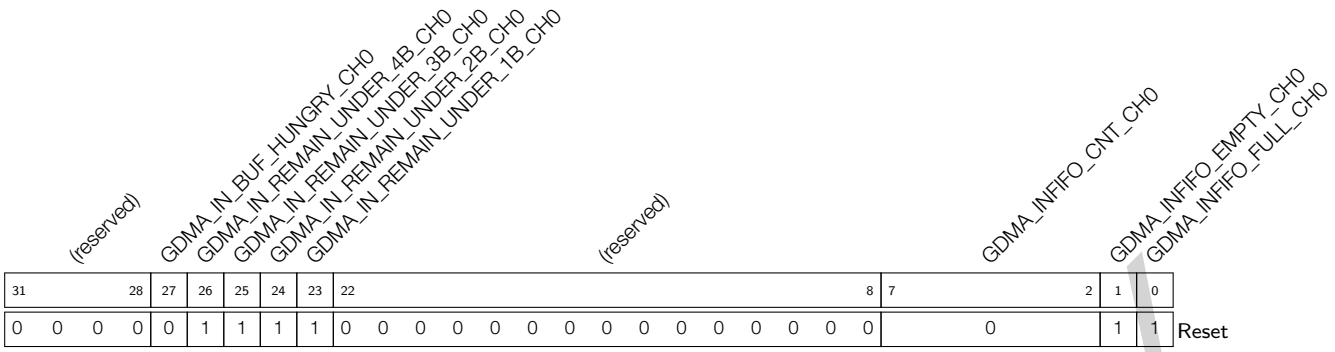
GDMA_OUTLINK_STOP_CH*n* Set this bit to stop dealing with the outlink descriptors. (R/W/SC)

GDMA_OUTLINK_START_CH*n* Set this bit to start dealing with the outlink descriptors. (R/W/SC)

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

GDMA_OUTLINK_PARK_CH*n* 1: the outlink descriptor's FSM is in idle state. 0: the outlink descriptor's FSM is working. (RO)

Register 2.15. GDMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0078+192n*)**



GDMA_INFIFO_FULL_CH*n* L1 Rx FIFO full signal for Rx channel 0. (RO)

GDMA_INFIFO_EMPTY_CH*n* L1 Rx FIFO empty signal for Rx channel 0. (RO)

GDMA_INFIFO_CNT_CH*n* The register stores the byte number of the data in L1 Rx FIFO for Rx channel 0. (RO)

GDMA_IN_REMAIN_UNDER_1B_CH*n* reserved (RO)

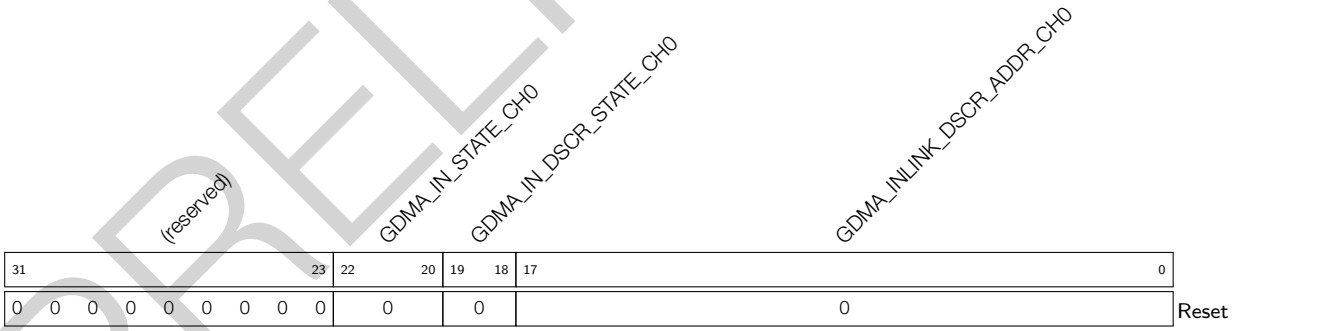
GDMA_IN_REMAIN_UNDER_2B_CH*n* reserved (RO)

GDMA_IN_REMAIN_UNDER_3B_CH*n* reserved (RO)

GDMA_IN_REMAIN_UNDER_4B_CH*n* reserved (RO)

GDMA_IN_BUF_HUNGRY_CH*n* reserved (RO)

Register 2.16. GDMA_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0084+192n*)**

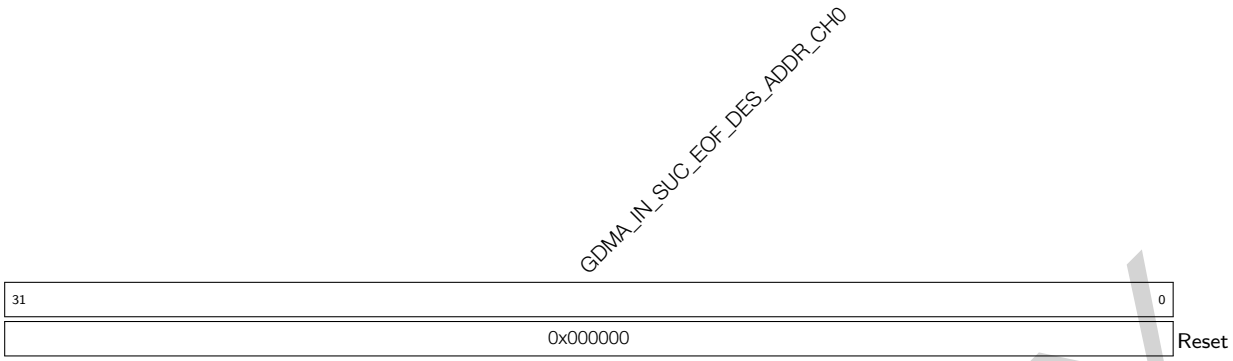


GDMA_INLINK_DSCR_ADDR_CH*n* This register stores the current inlink descriptor's address. (RO)

GDMA_IN_DSCR_STATE_CH*n* reserved (RO)

GDMA_IN_STATE_CH*n* reserved (RO)

Register 2.17. GDMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0088+192n*)**

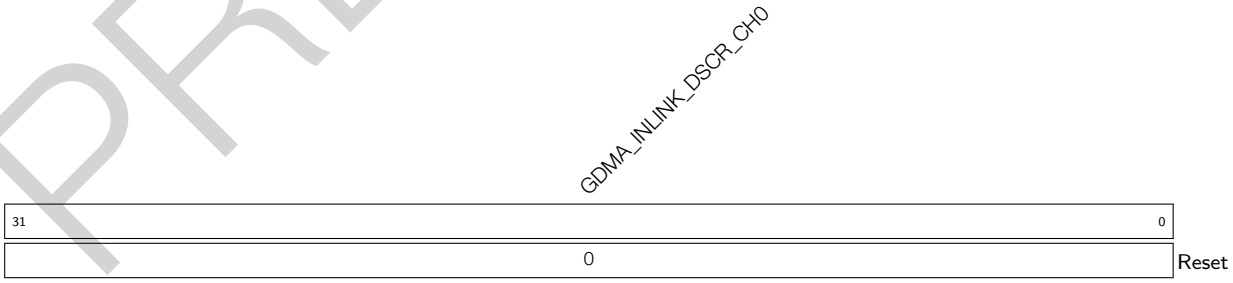


GDMA_IN_SUC_EOF_DES_ADDR_CH*n* This register stores the address of the inlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 2.18. GDMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x008C+192n*)**

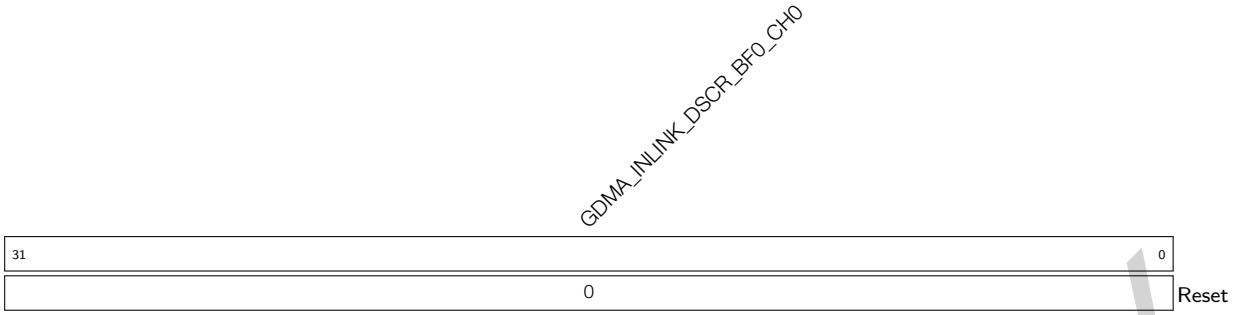
GDMA_IN_ERR_EOF_DES_ADDR_CH*n* This register stores the address of the inlink descriptor when there are some errors in current receiving data. Only used when peripheral is UHCI0. (RO)

Register 2.19. GDMA_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0090+192n*)**



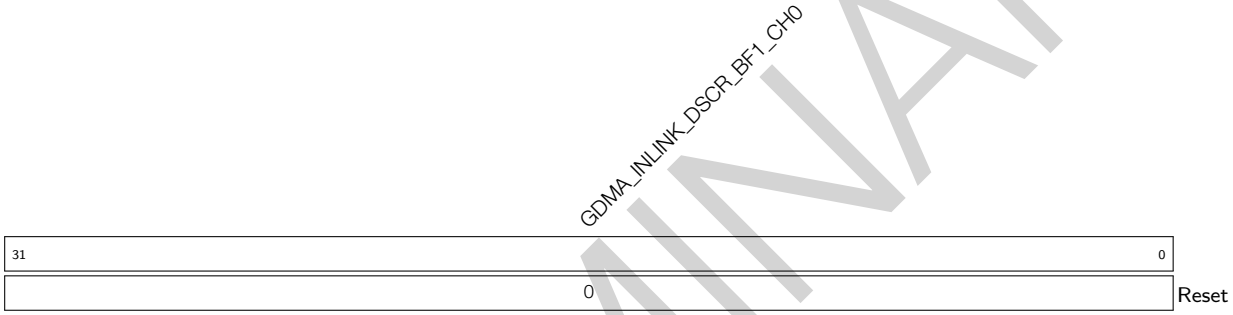
GDMA_INLINK_DSCR_CH*n* The address of the current inlink descriptor x. (RO)

Register 2.20. GDMA_IN_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0094+192n*)**



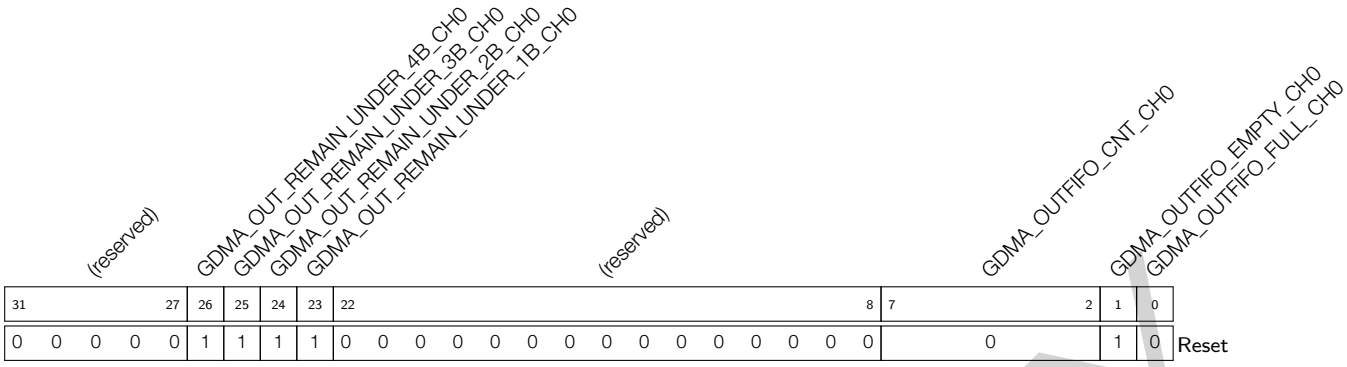
GDMA_INLINK_DSCR_BF0_CH*n* The address of the last inlink descriptor x-1. (RO)

Register 2.21. GDMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0098+192n*)**



GDMA_INLINK_DSCR_BF1_CH*n* The address of the second-to-last inlink descriptor x-2. (RO)

Register 2.22. GDMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x00D8+192n*)**



GDMA_OUTFIFO_FULL_CH*n* L1 Tx FIFO full signal for Tx channel 0. (RO)

GDMA_OUTFIFO_EMPTY_CH*n* L1 Tx FIFO empty signal for Tx channel 0. (RO)

GDMA_OUTFIFO_CNT_CH*n* The register stores the byte number of the data in L1 Tx FIFO for Tx channel 0. (RO)

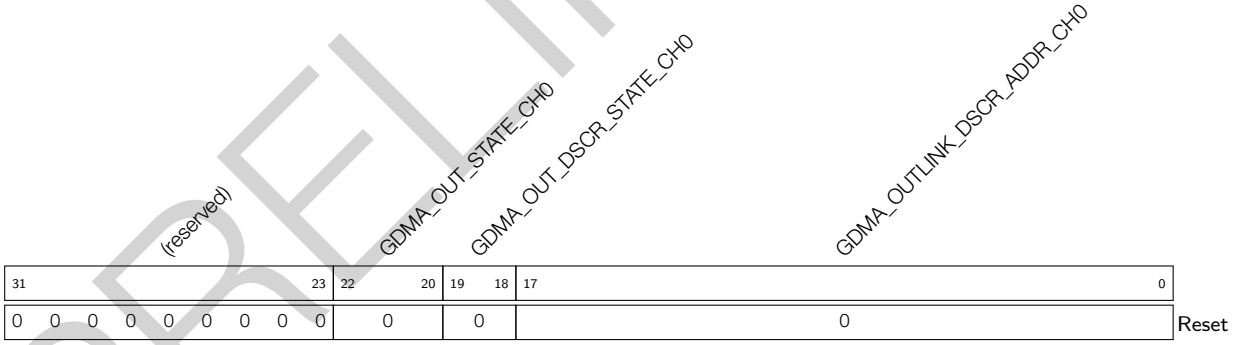
GDMA_OUT_REMAIN_UNDER_1B_CH*n* reserved (RO)

GDMA_OUT_REMAIN_UNDER_2B_CH*n* reserved (RO)

GDMA_OUT_REMAIN_UNDER_3B_CH*n* reserved (RO)

GDMA_OUT_REMAIN_UNDER_4B_CH*n* reserved (RO)

Register 2.23. GDMA_OUT_STATE_CH*n*_REG (*n*: 0-2) (0x00E4+192n*)**

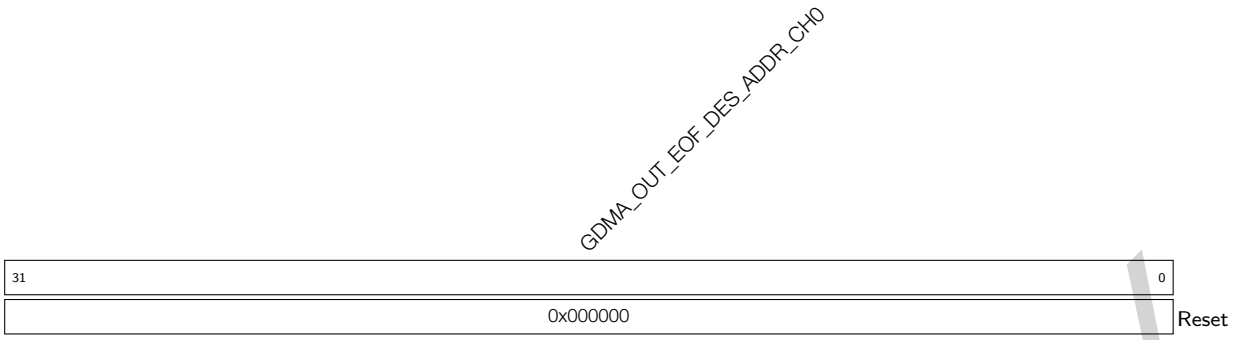


GDMA_OUTLINK_DSCR_ADDR_CH*n* This register stores the current outlink descriptor's address. (RO)

GDMA_OUT_DSCR_STATE_CH*n* reserved (RO)

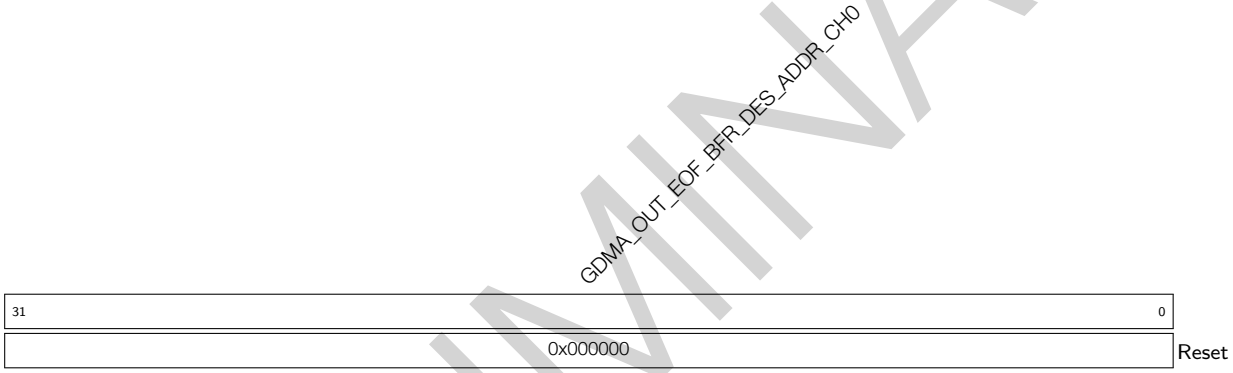
GDMA_OUT_STATE_CH*n* reserved (RO)

Register 2.24. GDMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00E8+192n*)**



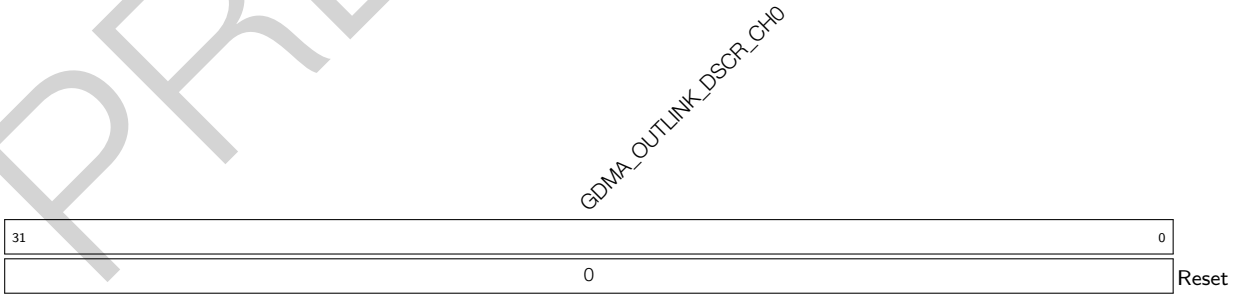
GDMA_OUT_EOF_DES_ADDR_CH*n* This register stores the address of the outlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 2.25. GDMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00EC+192n*)**



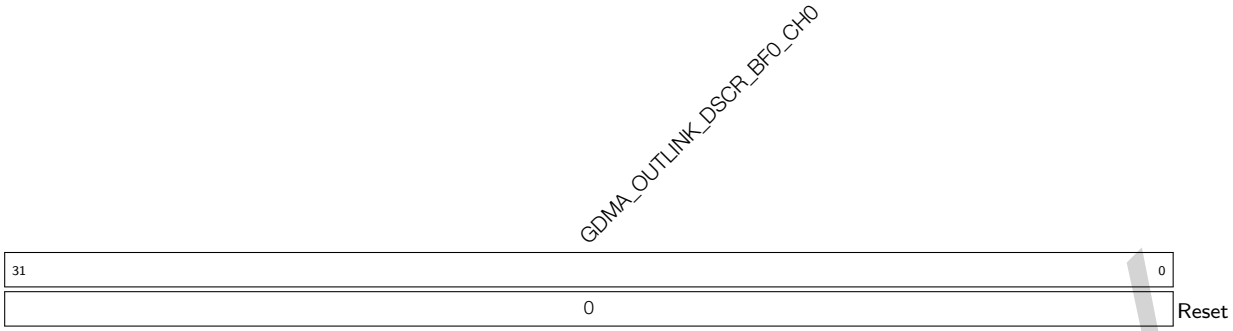
GDMA_OUT_EOF_BFR_DES_ADDR_CH*n* This register stores the address of the outlink descriptor before the last outlink descriptor. (RO)

Register 2.26. GDMA_OUT_DSCR_CH*n*_REG (*n*: 0-2) (0x00F0+192n*)**



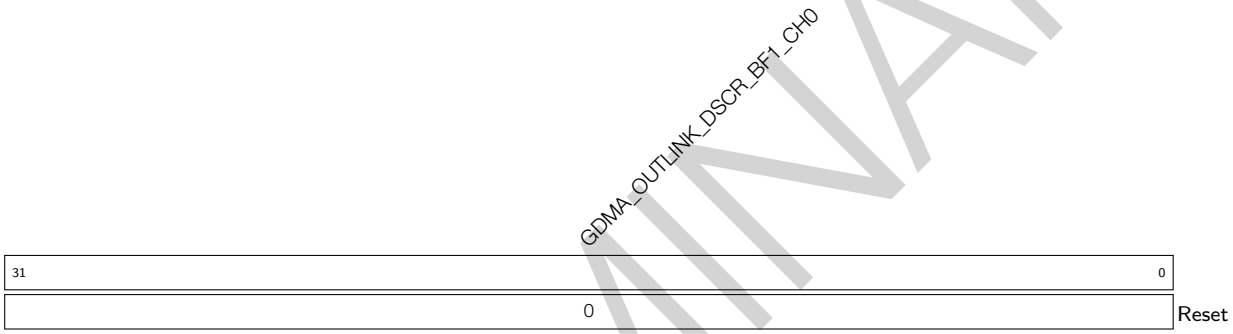
GDMA_OUTLINK_DSCR_CH*n* The address of the current outlink descriptor y. (RO)

Register 2.27. GDMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x00F4+192n*)**



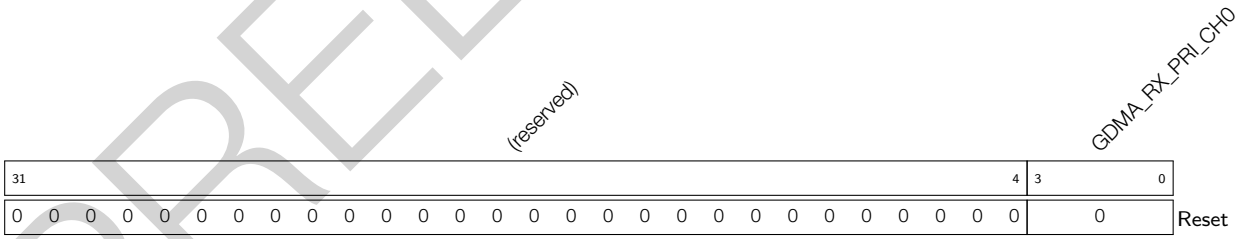
GDMA_OUTLINK_DSCR_BF0_CH*n* The address of the last outlink descriptor y-1. (RO)

Register 2.28. GDMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x00F8+192n*)**



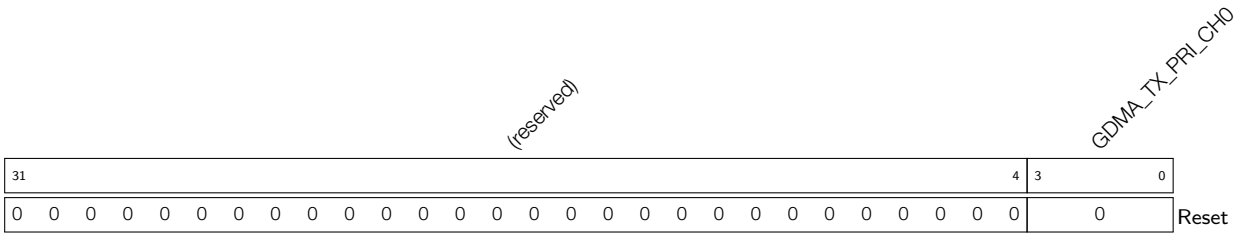
GDMA_OUTLINK_DSCR_BF1_CH*n* The address of the second-to-last inlink descriptor x-2. (RO)

Register 2.29. GDMA_IN_PRI_CH*n*_REG (*n*: 0-2) (0x009C+192n*)**



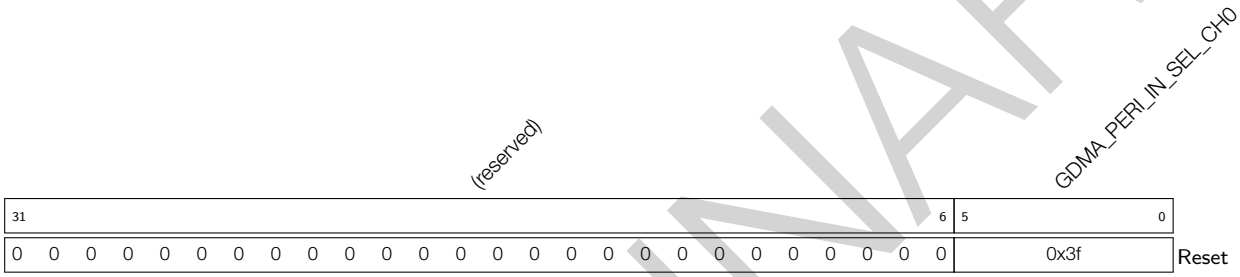
GDMA_RX_PRI_CH*n* The priority of Rx channel 0. The larger of the value, the higher of the priority. (R/W)

Register 2.30. GDMA_OUT_PRI_CH*n*_REG (*n*: 0-2) (0x00FC+192n*)**



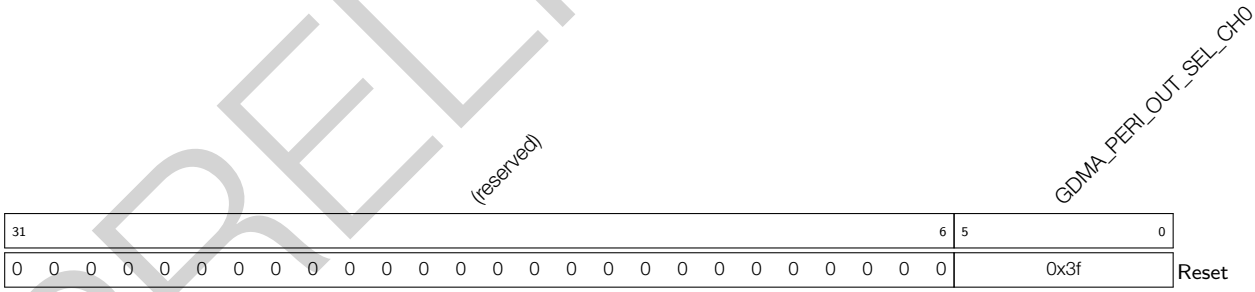
GDMA_TX_PRI_CH*n* The priority of Tx channel 0. The larger of the value, the higher of the priority. (R/W)

Register 2.31. GDMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x00A0+192n*)**



GDMA_PERI_IN_SEL_CH*n* This register is used to select peripheral for Rx channel 0. 0:SPI2. 1: reserved. 2: UHCI0. 3: I2S. 4: reserved. 5: reserved. 6: AES. 7: SHA. 8: ADC. (R/W)

Register 2.32. GDMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x0100+192n*)**



GDMA_PERI_OUT_SEL_CH*n* This register is used to select peripheral for Tx channel 0. 0:SPI2. 1: reserved. 2: UHCI0. 3: I2S. 4: reserved. 5: reserved. 6: AES. 7: SHA. 8: ADC. (R/W)

3 System and Memory

3.1 Overview

The ESP32-C3 is an ultra-low-power and highly-integrated system with a 32-bit RISC-V single-core processor with a four-stage pipeline that operates at up to 160 MHz. All internal memory, external memory, and peripherals are located on the CPU buses.

3.2 Features

- **Address Space**
  - 792 KB of internal memory address space accessed from the instruction bus
  - 552 KB of internal memory address space accessed from the data bus
  - 836 KB of peripheral address space
  - 8 MB of external memory virtual address space accessed from the instruction bus
  - 8 MB of external memory virtual address space accessed from the data bus
  - 384 KB of internal DMA address space
- **Internal Memory**
  - 384 KB of Internal ROM
  - 400 KB of Internal SRAM
  - 8 KB of RTC Memory
- **External Memory**
  - Supports up to 16 MB external flash
- **Peripheral Space**
  - 35 modules/peripherals in total
- **GDMA**
  - 7 GDMA-supported modules/peripherals

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

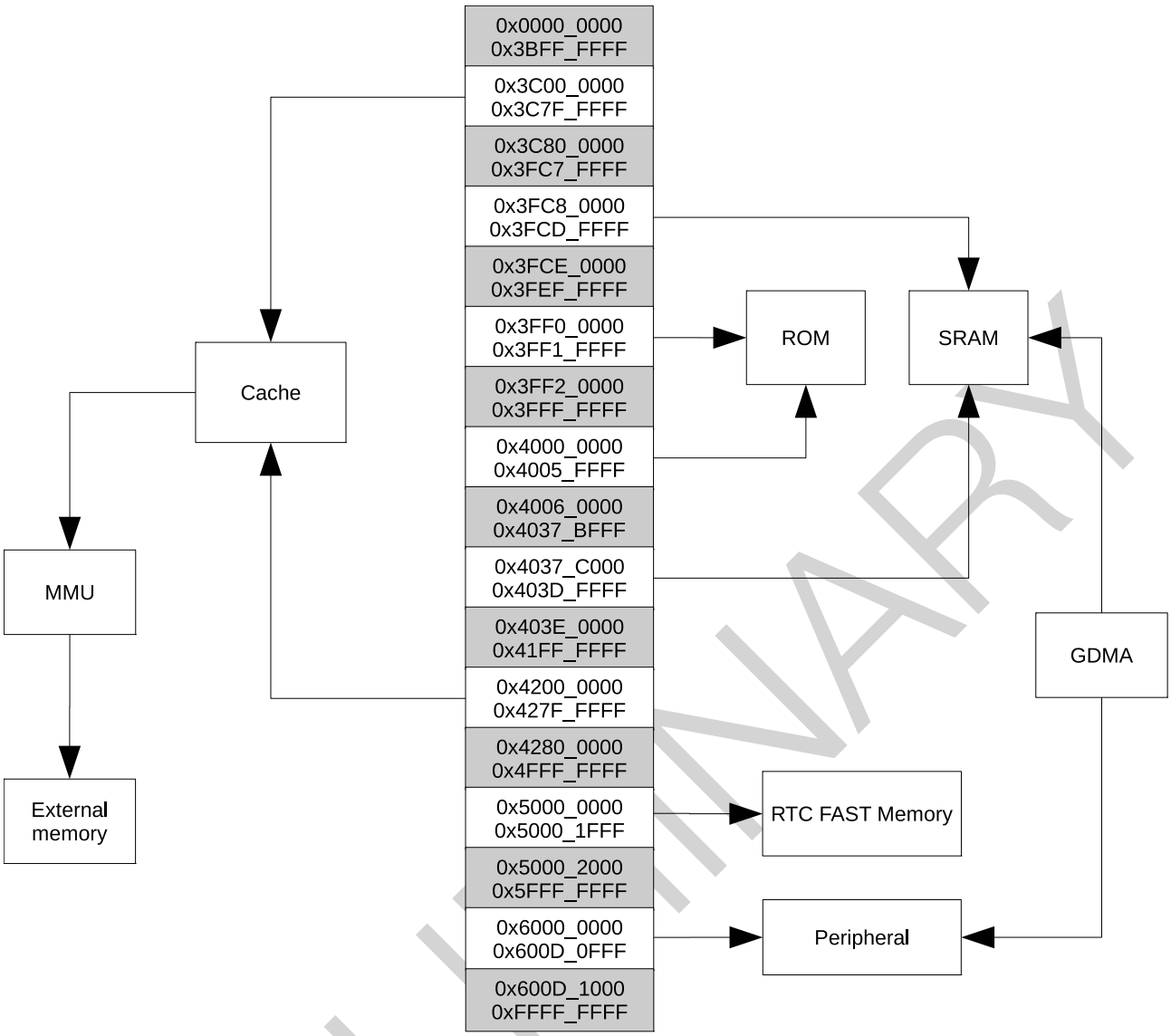


Figure 3-1. System Structure and Address Mapping

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 shared by the data bus and the instruction bus.

Both data bus and instruction bus are little-endian. The CPU can access data via the data bus using single-byte, double-byte, 4-byte alignment. The CPU can also access data via the instruction bus, but only in 4-byte aligned

manner.

The CPU can:

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

Table 3-1 lists the address ranges on the data bus and instruction bus and their corresponding target memory.

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

Table 3-1. Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
	0x0000_0000	0x3BFF_FFFF		Reserved
Data bus	0x3C00_0000	0x3C7F_FFFF	8 MB	External memory
	0x3C80_0000	0x3FC7_FFFF		Reserved
Data bus	0x3FC8_0000	0x3FCD_FFFF	384 KB	Internal memory
	0x3FCE_0000	0x3FEF_FFFF		Reserved
Data bus	0x3FF0_0000	0x3FF1_FFFF	128 KB	Internal memory
	0x3FF2_0000	0x3FFF_FFFF		Reserved
Instruction bus	0x4000_0000	0x4005_FFFF	384 KB	Internal memory
	0x4006_0000	0x4037_BFFF		Reserved
Instruction bus	0x4037_C000	0x403D_FFFF	400 KB	Internal memory
	0x403E_0000	0x41FF_FFFF		Reserved
Instruction bus	0x4200_0000	0x427F_FFFF	8 MB	External memory
	0x4280_0000	0x4FFF_FFFF		Reserved
Data/Instruction bus	0x5000_0000	0x5000_1FFF	8 KB	Internal memory
	0x5000_2000	0x5FFF_FFFF		Reserved
Data/Instruction bus	0x6000_0000	0x600D_0FFF	836 KB	Peripherals
	0x600D_1000	0xFFFF_FFFF		Reserved

3.3.2 Internal Memory

The ESP32-C3 consists of the following three types of internal memory:

- Internal ROM (384 KB): The Internal ROM of the ESP32-C3 is a Mask ROM, meaning it is strictly read-only and cannot be reprogrammed. Internal ROM contains the ROM code (software instructions and some software read-only data) of some low level system software.
- Internal SRAM (400 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.

- RTC Memory (8 KB): The RTC (Real Time Clock) memory implemented as Static RAM (SRAM) thus is volatile. However, RTC memory has the added feature of being persistent in deep sleep (i.e., the RTC memory retains its values throughout deep sleep).
  - RTC FAST Memory (8 KB): RTC FAST memory can only be accessed by the CPU and can be generally used to store instructions and data that needs to persist across a deep sleep.

Based on the three different types of internal memory described above, the internal memory of the ESP32-C3 is split into three segments: Internal ROM (384 KB), Internal SRAM (400 KB), RTC FAST Memory (8 KB).

However, within each segment, there may be different bus access restrictions (e.g., some parts of the segment may only be accessible by the CPU's Data bus). Therefore, each some segments are also further divided into parts. Table 3-2 describes each part of internal memory and their address ranges on the data bus and/or instruction bus.

Table 3-2. Internal Memory Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
Data bus	0x3FF0_0000	0x3FF1_FFFF	128 KB	Internal ROM 1
	0x3FC8_0000	0x3FCD_FFFF	384 KB	Internal SRAM 1
Instruction bus	0x4000_0000	0x4003_FFFF	256 KB	Internal ROM 0
	0x4004_0000	0x4005_FFFF	128 KB	Internal ROM 1
	0x4037_C000	0x4037_FFFF	16 KB	Internal SRAM 0
	0x4038_0000	0x403D_FFFF	384 KB	Internal SRAM 1
Data/Instruction bus	0x5000_0000	0x5000_1FFF	8 KB	RTC FAST Memory

Note:

All of the internal memories are managed by Permission Control module. An internal memory can only be accessed when it is allowed by Permission Control, then the internal memory can be available to the CPU.

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 3-2.

2. Internal ROM 1

Internal ROM 1 is a 128 KB, read-only memory space, addressed by the CPU through the instruction bus via 0x4004_0000 ~ 0x4005_FFFF or through the data bus via 0x3FF0_0000 ~ 0x3FF1_FFFF in the same order, as shown in Table 3-2.

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 (the same ordering applies for Internal SRAM 1).

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 3-2.

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

4. Internal SRAM 1

Internal SRAM 1 is a 384 KB, read-and-write memory space, addressed by the CPU through the data bus or instruction bus, in the same order, via the ranges described in Table 3-2.

5. RTC FAST Memory

RTC FAST Memory is a 8 KB, read-and-write SRAM, addressed by the CPU through the data/instruction bus via the shared address 0x5000_0000 ~ 0x5000_1FFF, as described in Table 3-2.

3.3.3 External Memory

ESP32-C3 supports SPI, Dual SPI, Quad SPI, and QPI interfaces that allow connection to multiple external flash. It supports hardware manual encryption and automatic decryption based on XTS_AES 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 ESP32-C3 can address up to 16 MB external flash.

Using the cache, ESP32-C3 is able to support the following address space mappings. Note that the instruction bus address space (8MB) and the data bus address space (8 MB) is always shared.

- Up to 8 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 8 MB data bus (read-only) address space can be mapped into the external flash. The mapped address space is organized as individual 64-KB blocks.

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

Table 3-3. External Memory Address Mapping

Bus Type	Boundary Address		Size	Target
	Low Address	High Address		
Data bus (read-only)	0x3C00_0000	0x3C7F_FFFF	8 MB	Uniform Cache
Instruction bus	0x4200_0000	0x427F_FFFF	8 MB	Uniform Cache

Note:

Only if the CPU obtains permission for accessing the external memory, can it be responded for memory access.

3.3.3.2 Cache

As shown in Figure 3-2, ESP32-C3 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.

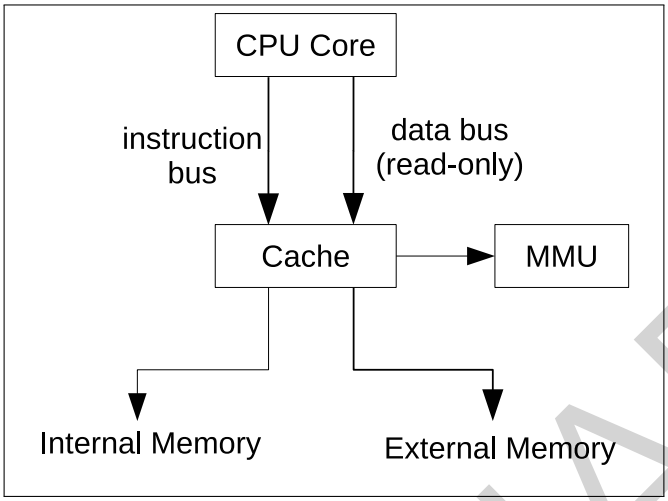


Figure 3-2. Cache Structure

3.3.3.3 Cache Operations

ESP32-C3 cache support 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. 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 ESP32-C3 can provide DMA (Direct Memory Access) services including:

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

GDMA uses the same addresses as the data bus to read and write Internal SRAM 1. Specifically, GDMA uses address range 0x3FC8_0000 ~ 0x3FCD_FFFF to access Internal SRAM 1. Note that GDMA cannot access the internal memory occupied by the cache.

There are 7 peripherals/modules that can work together with GDMA. As shown in Figure 3-3, these 7 vertical lines in turn correspond to these 7 peripherals/modules with GDMA function, the horizontal line represents a certain channel of GDMA (can be any channel), and the intersection of the vertical line and the horizontal line indicates that a peripheral/module has the ability to access the corresponding channel of GDMA. If there are multiple intersections on the same line, it means that these peripherals/modules cannot enable the GDMA function at the same time.

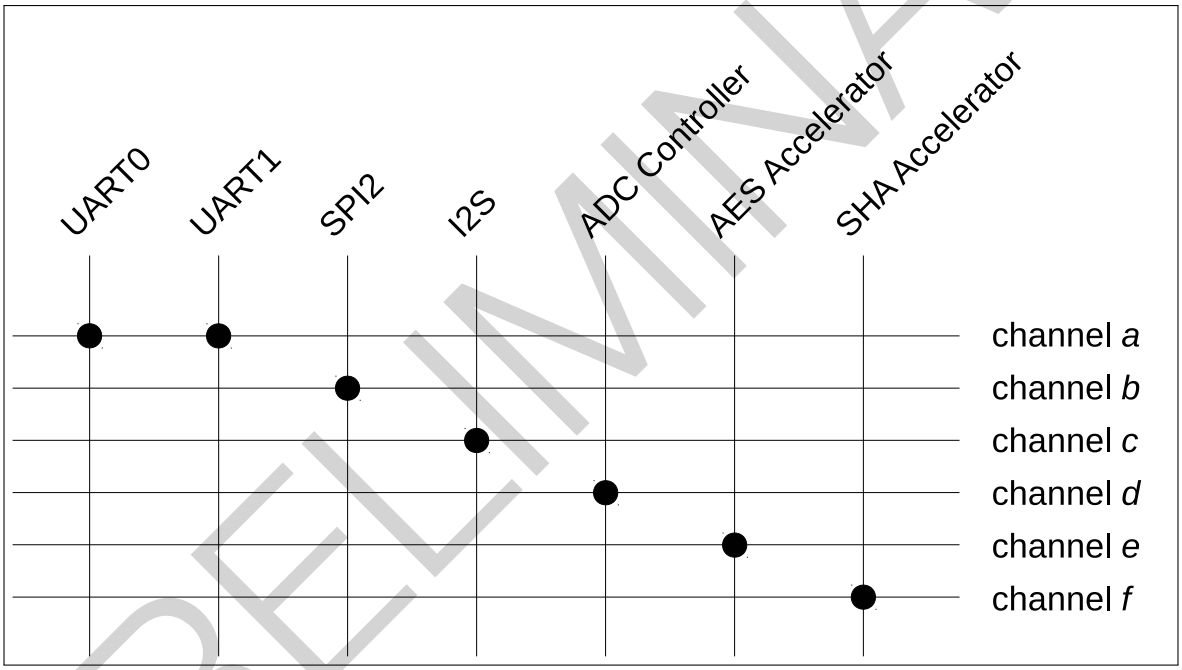


Figure 3-3. Peripherals/modules that can work with GDMA

These peripherals/modules can access any memory available to GDMA.

Note:

When accessing a memory via GDMA, a corresponding access permission is needed, otherwise this access may fail.

3.3.5 Modules/Peripherals

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

3.3.5.1 Module/Peripheral Address Mapping

Table 3-4 lists all the modules/peripherals and their respective address ranges. Note that the address space of specific modules/peripherals is defined by "Boundary Address" (including both Low Address and High Address).

Table 3-4. Module/Peripheral Address Mapping

Target	Boundary Address		Size	Notes
	Low Address	High Address		
UART Controller 0	0x6000_0000	0x6000_0FFF	4 KB	
Reserved	0x6000_1000	0x6000_1FFF		
SPI Controller 1	0x6000_2000	0x6000_2FFF	4 KB	
SPI Controller 0	0x6000_3000	0x6000_3FFF	4 KB	
GPIO	0x6000_4000	0x6000_4FFF	4 KB	
Reserved	0x6000_5000	0x6000_6FFF		
TIMER	0x6000_7000	0x6000_7FFF	4 KB	
Low-Power Management	0x6000_8000	0x6000_8FFF	4 KB	
IO MUX	0x6000_9000	0x6000_9FFF	4 KB	
Reserved	0x6000_A000	0x6000_FFFF		
UART Controller 1	0x6001_0000	0x6001_0FFF	4 KB	
Reserved	0x6001_1000	0x6001_2FFF		
I2C Controller	0x6001_3000	0x6001_3FFF	4 KB	
UHCI0	0x6001_4000	0x6001_4FFF	4 KB	
Reserved	0x6001_5000	0x6001_5FFF		
Remote Control Peripheral	0x6001_6000	0x6001_6FFF	4 KB	
Reserved	0x6001_7000	0x6001_8FFF		
LED Control PWM	0x6001_9000	0x6001_9FFF	4 KB	
eFuse Controller	0x6001_A000	0x6001_AFFF	4 KB	
Reserved	0x6001_B000	0x6001_EFFF		
Timer Group 0	0x6001_F000	0x6001_FFFF	4 KB	
Timer Group 1	0x6002_0000	0x6002_0FFF	4 KB	
Reserved	0x6002_1000	0x6002_2FFF		
System Timer	0x6002_3000	0x6002_3FFF	4 KB	
SPI Controller 2	0x6002_4000	0x6002_4FFF	4 KB	
Reserved	0x6002_5000	0x6002_5FFF		
APB Controller	0x6002_6000	0x6002_6FFF	4 KB	
Reserved	0x6002_7000	0x6002_AFFF		
Two-wire Automotive Interface	0x6002_B000	0x6002_BFFF	4 KB	
Reserved	0x6002_C000	0x6002_CFFF		
I2S Controller	0x6002_D000	0x6002_DFFF	4 KB	
Reserved	0x6002_E000	0x6003_9FFF		
AES Accelerator	0x6003_A000	0x6003_AFFF	4 KB	
SHA Accelerator	0x6003_B000	0x6003_BFFF	4 KB	
RSA Accelerator	0x6003_C000	0x6003_CFFF	4 KB	
Digital Signature	0x6003_D000	0x6003_DFFF	4 KB	

Target	Boundary Address		Size	Notes
	Low Address	High Address		
HMAC Accelerator	0x6003_E000	0x6003_EFFF	4 KB	
GDMA Controller	0x6003_F000	0x6003_FFFF	4 KB	
ADC Controller	0x6004_0000	0x6004_0FFF	4 KB	
Reserved	0x6004_1000	0x6002_FFFF		
USB Serial/JTAG Controller	0x6004_3000	0x6004_3FFF	4 KB	
Reserved	0x6004_4000	0x600B_FFFF		
System Registers	0x600C_0000	0x600C_0FFF	4 KB	
Sensitive Register	0x600C_1000	0x600C_1FFF	4 KB	
Interrupt Matrix	0x600C_2000	0x600C_2FFF	4 KB	
Reserved	0x600C_3000	0x600C_3FFF		
Configure Cache	0x600C_4000	0x600C_BFFF	32 KB	
External Memory Encryption and Decryption	0x600C_C000	0x600C_CFFF	4 KB	
Reserved	0x600C_D000	0x600C_DFFF		
Assist Debug	0x600C_E000	0x600C_EFFF	4 KB	
Reserved	0x600C_F000	0x600C_FFFF		
World Controller	0x600D_0000	0x600D_0FFF	4 KB	

4 eFuse Controller (EFUSE)

4.1 Overview

ESP32-C3 contains a 4096-bit eFuse controller to store parameters. 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 software configurations. Some of these parameters can be read by software using the eFuse controller, while some can be directly used by hardware modules.

4.2 Features

- 4096-bit One-time programmable storage
- Programmable write-protection
- Programmable read-protection against software
- Various hardware encoding schemes against data corruption

4.3 Functional Description

4.3.1 Structure

eFuse data is organized in 11 blocks (BLOCK0 ~ BLOCK10).

BLOCK0, which holds most parameters, has 9 bits that can only be used by hardware and are invisible to software, and 61 further bits are reserved for future use.

Table 4-1 lists all the parameters in BLOCK0 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 disable the writing of other parameters, while **EFUSE_RD_DIS** is used to disable software from reading BLOCK4 ~ BLOCK10. 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 eFuse BLOCK0

Parameters	Offset	Bit Width	Hardware Use	Write-Protect Bits in EFUSE_WR_DIS	Description
EFUSE_WR_DIS	0	32	Y	N/A	Disable writing of individual eFuses.
EFUSE_RD_DIS	32	7	Y	0	Disable software from reading eFuse blocks BLOCK4 ~ 10.
EFUSE_DIS_ICACHE	40	1	Y	2	Disable ICache.
EFUSE_DIS_USB_JTAG	41	1	Y	2	Disable usb-to-jtag function.
EFUSE_DIS_DOWNLOAD_ICACHE	42	1	Y	2	Disable ICache in Download mode.
EFUSE_DIS_USB_DEVICE	43	1	Y	2	Disable USB device peripheral.
EFUSE_DIS_FORCE_DOWNLOAD	44	1	Y	2	Disable chip from force-entering Download mode.

Parameters	Offset	Bit Width	Hardware Use	Write-Protect Bits in EFUSE_WR_DIS	Description
EFUSE_DIS_TWAI	46	1	Y	2	Disable TWAI Controller.
EFUSE_JTAG_SEL_ENABLE	47	1	Y	2	Set 1 enables strap pin (GPIO 10) to select usb-to-jtag function or use jtag directly (When GPIO 10 is 1, it means usb-to-jtag function is select; while 0 means using jtag directly).
EFUSE_SOFT_DIS_JTAG	48	3	Y	31	Disable JTAG by programming 1 to odd number of bits. JTAG can be re-enabled via HMAC peripheral.
EFUSE_DIS_PAD_JTAG	51	1	Y	2	Hardware Disable JTAG permanently.
EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT	52	1	Y	2	Disable flash encryption in Download boot mode.
EFUSE_USB_EXCHG_PINS	57	1	Y	30	Exchange USB D+/D- pins.
EFUSE_VDD_SPI_AS_GPIO	58	1	N	30	Set this parameter to 1 to override the function of the VDD SPI pin and use it as a normal GPIO pin instead.
EFUSE_WDT_DELAY_SEL	80	2	Y	3	Select RTC WDT timeout threshold.
EFUSE_SPI_BOOT_CRYPT_CNT	82	3	Y	4	Enable SPI boot encryption and decryption. This feature is enabled when odd number of bits are set in this parameter, disabled otherwise.
EFUSE_SECURE_BOOT_KEY_REVOKE0	85	1	N	5	Revoke the first secure boot key when enabled.
EFUSE_SECURE_BOOT_KEY_REVOKE1	86	1	N	6	Revoke the second secure boot key when enabled.
EFUSE_SECURE_BOOT_KEY_REVOKE2	87	1	N	7	Revoke the third secure boot key when enabled.
EFUSE_KEY_PURPOSE_0	88	4	Y	8	Key0 purpose, see Table 4-2.
EFUSE_KEY_PURPOSE_1	92	4	Y	9	Key1 purpose, see Table 4-2.
EFUSE_KEY_PURPOSE_2	96	4	Y	10	Key2 purpose, see Table 4-2.
EFUSE_KEY_PURPOSE_3	100	4	Y	11	Key3 purpose, see Table 4-2.
EFUSE_KEY_PURPOSE_4	104	4	Y	12	Key4 purpose, see Table 4-2.
EFUSE_KEY_PURPOSE_5	108	4	Y	13	Key5 purpose, see Table 4-2.
EFUSE_SECURE_BOOT_EN	116	1	N	15	Enable secure boot.
EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE	117	1	N	16	Enable aggressive Secure boot key revocation mode.

Parameters	Offset	Bit Width	Hardware Use	Write-Protect Bits in EFUSE_WR_DIS	Description
EFUSE_FLASH_TPUW	124	4	N	18	Configure flash startup delay after SoC being powered up (the unit is ms/2). When the value is 15, delay will be 7.5 ms.
EFUSE_DIS_DOWNLOAD_MODE	128	1	N	18	Disable all download boot modes.
EFUSE_USB_PRINT_CHANNEL	130	1	N	18	Set this parameter to 1, the usb print function will be disabled.
EFUSE_DIS_USB_DOWNLOAD_MODE	132	1	N	18	Disable the USB OTG download feature in UART download boot mode.
EFUSE_ENABLE_SECURITY_DOWNLOAD	133	1	N	18	Enable UART secure download mode (read/write flash only).
EFUSE_UART_PRINT_CONTROL	134	2	N	18	Set UART boot message output mode. 2'b00: Force print; 2'b01: Low-level print, controlled by GPIO 8; 2'b10: High-level print, controlled by GPIO 8; 2'b11: Print force disabled.
EFUSE_FORCE_SEND_RESUME	141	1	N	18	Force ROM code to send an SPI flash resume command during SPI boot.
EFUSE_SECURE_VERSION	142	16	N	18	Secure version (used by ESP-IDF anti-rollback feature).

Table 4-2 lists all key purpose and their values. Setting the eFuse parameter EFUSE_KEY_PURPOSE_*n* declares the purpose of KEY*n* (*n*: 0 ~ 5).

Table 4-2. Secure Key Purpose Values

Key Purpose Values	Purposes
0	For users (software-only)
1	Reserved
2	XTS_AES_256_KEY_1 (flash/SRAM encryption and decryption)
3	XTS_AES_256_KEY_2 (flash/SRAM encryption and decryption)
4	XTS_AES_128_KEY (flash/SRAM encryption and decryption)
5	HMAC Downstream mode (both JTAG and DS)
6	JTAG in HMAC Downstream mode
7	Digital Signature peripheral in HMAC Downstream mode
8	HMAC Upstream mode
9	SECURE_BOOT_DIGEST0 (secure boot key digest)
10	SECURE_BOOT_DIGEST1 (secure boot key digest)
11	SECURE_BOOT_DIGEST2 (secure boot key digest)

Table 4-3 provides the details of parameters in BLOCK1 ~ BLOCK10.

Table 4-3. Parameters in BLOCK1 to BLOCK10

BLOCK	Parameters	Bit Width	Hardware Use	Write-Protect Bits in EFUSE_WR_DIS	Software Read-Protect Bits in EFUSE_RD_DIS	Description
BLOCK1	EFUSE_MAC	48	N	20	N/A	MAC address
	EFUSE_SPI_PAD_CONFIGURE	[0:5]	N	20	N/A	CLK
		[6:11]	N	20	N/A	Q (D1)
		[12:17]	N	20	N/A	D (D0)
		[18:23]	N	20	N/A	CS
		[24:29]	N	20	N/A	HD (D3)
		[30:35]	N	20	N/A	WP (D2)
		[36:41]	N	20	N/A	DQS
		[42:47]	N	20	N/A	D4
		[48:53]	N	20	N/A	D5
		[54:59]	N	20	N/A	D6
		[60:65]	N	20	N/A	D7
	EFUSE_SYS_DATA_PART0	78	N	20	N/A	System data
BLOCK2	EFUSE_SYS_DATA_PART1	256	N	21	N/A	System data
BLOCK3	EFUSE_USR_DATA	256	N	22	N/A	User data
BLOCK4	EFUSE_KEY0_DATA	256	Y	23	0	KEY0 or user data
BLOCK5	EFUSE_KEY1_DATA	256	Y	24	1	KEY1 or user data
BLOCK6	EFUSE_KEY2_DATA	256	Y	25	2	KEY2 or user data
BLOCK7	EFUSE_KEY3_DATA	256	Y	26	3	KEY3 or user data
BLOCK8	EFUSE_KEY4_DATA	256	Y	27	4	KEY4 or user data
BLOCK9	EFUSE_KEY5_DATA	256	Y	28	5	KEY5 or user data
BLOCK10	EFUSE_SYS_DATA_PART2	256	N	29	6	System data

Among these blocks, BLOCK4 ~ 9 stores KEY0 ~ 5, respectively. Up to six 256-bit keys can be written into eFuse. Whenever a key is written, its purpose value should also be written (see table 4-2). For example, when a key for the JTAG function in HMAC Downstream mode is written to KEY3 (i.e., BLOCK7), its key purpose value 6 should also be written to EFUSE_KEY_PURPOSE_3.

BLOCK1 ~ BLOCK10 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 the new values would take effect.

Column “Write-Protect Bits in EFUSE_WR_DIS” in Table 4-1 and Table 4-3 list 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 remain 1.

4.3.1.2 EFUSE_RD_DIS

Only parameters in BLCOK4 ~ BLOCK10 may be read-protected against software reads, as shown in column "Software Read-Protect Bits in EFUSE_RD_DIS" of Table 4-3. After EFUSE_RD_DIS has been programmed, execute an eFuse read operation so the new values would take effect.

If a bit in EFUSE_RD_DIS is 0, it means that its parameters are not read-protected against software; if a bit in EFUSE_RD_DIS is 1, it means that its parameters are read-protected against software.

Other parameters that are not in BLOCK4 ~ BLOCK10 can always be read by software.

However, even if BLOCK4 ~ BLOCK10 are set to be read-protected, they can still be read by hardware modules, if the EFUSE_KEY_PURPOSE_*n* bit is set accordingly.

4.3.1.3 Data Storage

Internally, eFuses use 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 software.

BLOCK1 ~ BLOCK10 use RS (44, 32) coding scheme that supports up to 5 bytes of automatic error correction. The primitive polynomial of RS (44, 32) is $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.

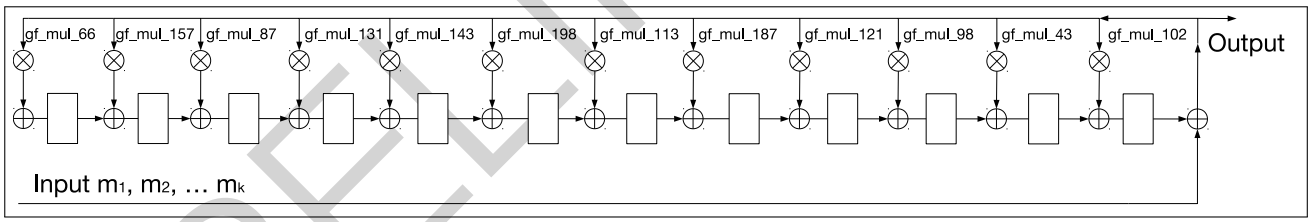


Figure 4-1. Shift Register Circuit

The shift register circuit that generates the check code is shown in Figure 4-1, 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.

Software must encode the 32-byte parameter using RS (44, 32) to generate a 12-byte check code, and then program the parameter and the check code into eFuse at the same time. The eFuse controller will automatically process decoding and error correction 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.

4.3.2 Software Programming of Parameters

The eFuse controller can only program eFuse parameters in one block at a time. BLOCK0 ~ BLOCK10 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

When EFUSE_BLK_NUM is set to 0, BLOCK0 will be programmed. Register EFUSE_PGM_DATA0_REG stores EFUSE_WR_DIS. Registers EFUSE_PGM_DATA1_REG ~ EFUSE_PGM_DATA5_REG store the information of parameters to be programmed. Note that 7 bits can only be used by hardware and must always be set to 0. The specific bits are:

- EFUSE_PGM_DATA1_REG[24:21]
- EFUSE_PGM_DATA1_REG[31:27]

Data in registers EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG and EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG are ignored when programming BLOCK0.

Programming BLOCK1

When EFUSE_BLK_NUM is set to 1, registers EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA5_REG store the BLOCK1 parameters to be programmed. Registers EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_DATA2_REG store the corresponding RS check codes. Data in registers EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG are ignored when programming BLOCK1, and the RS check codes will be calculated with these bits all treated as 0.

Programming BLOCK2 ~ 10

When EFUSE_BLK_NUM is set to 2 ~ 10, registers EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG store the parameters to be programmed to this block. Registers EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG store the corresponding RS check codes.

Programming process

The process of programming parameters is as follows:

1. Configure the value of parameter EFUSE_BLK_NUM to determine the block to be programmed.
2. Write 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. Configure the field EFUSE_OP_CODE of register EFUSE_CONF_REG to 0x5A5A.
5. Configure the field EFUSE_PGM_CMD of register EFUSE_CMD_REG to 1.
6. Poll register EFUSE_CMD_REG until software reads 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.

Limitations

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. The programming of parameters controlled by a certain bit of EFUSE_WR_DIS, and the programming of the bit itself can even be completed at the same time. Repeated programming of already programmed bits is strictly forbidden, otherwise, programming errors will occur.

BLOCK1 cannot be programmed by users as it has been programmed at manufacturing.

BLOCK2 ~ 10 can only be programmed once. Repeated programming is not allowed.

4.3.3 Software Reading of Parameters

Software cannot read eFuse bits directly. The eFuse Controller hardware reads all eFuse bits and stores the results to their corresponding registers in its memory space. Then, software can read eFuse bits by reading the registers that start with EFUSE_RD_. Details are provided in Table 4-4.

Table 4-4. Registers Information

BLOCK	Read Registers	Registers When Programming This Block
0	EFUSE_RD_WR_DIS_REG	EFUSE_PGM_DATA0_REG
0	EFUSE_RD_REPEAT_DATA0 ~ 4_REG	EFUSE_PGM_DATA1 ~ 5_REG
1	EFUSE_RD_MAC_SPI_SYS_0 ~ 5_REG	EFUSE_PGM_DATA0 ~ 5_REG
2	EFUSE_RD_SYS_DATA_PART1_0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
3	EFUSE_RD_USR_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
4-9	EFUSE_RD_KEY*n*_DATA0 ~ 7_REG (*n*: 0 ~ 5)	EFUSE_PGM_DATA0 ~ 7_REG
10	EFUSE_RD_SYS_DATA_PART2_0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG

Updating eFuse read registers

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

1. Configure the field EFUSE_OP_CODE in register EFUSE_CONF_REG to 0x5AA5.
2. Configure the field EFUSE_READ_CMD in register EFUSE_CMD_REG to 1.
3. Poll register EFUSE_CMD_REG until software reads 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. Software reads the values of each parameter from memory.

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

Error detection

Error record registers allow software to detect if there are any inconsistencies in the stored backup eFuse parameters.

Registers EFUSE_RD_REPEAT_ERR0 ~ 3_REG indicate 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).

Registers EFUSE_RD_RS_ERR0 ~ 1_REG store the number of corrected bytes as well as the result of RS decoding during eFuse reading BLOCK1 ~ BLOCK10.

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

Identifying program/read operation

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

- Method one:
  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 two:
  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 interrupt signals, see Chapter 8 *Interrupt Matrix (INTERRUPT) [to be added later]*.
  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.

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-out time for VDDQ): The value of this parameter should be larger than 10 μs.

Table 4-5. 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 The Use of Parameters 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 4-3, specifically those marked with “Y” in columns “Hardware Use”. Software 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 reading 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 3-4 in Chapter 3 *System and Memory*.

07

Name	Description	Address	Access
PGM Data Register			
EFUSE_PGM_DATA0_REG	Register 0 that stores data to be programmed	0x0000	R/W
EFUSE_PGM_DATA1_REG	Register 1 that stores data to be programmed	0x0004	R/W
EFUSE_PGM_DATA2_REG	Register 2 that stores data to be programmed	0x0008	R/W
EFUSE_PGM_DATA3_REG	Register 3 that stores data to be programmed	0x000C	R/W
EFUSE_PGM_DATA4_REG	Register 4 that stores data to be programmed	0x0010	R/W
EFUSE_PGM_DATA5_REG	Register 5 that stores data to be programmed	0x0014	R/W
EFUSE_PGM_DATA6_REG	Register 6 that stores data to be programmed	0x0018	R/W
EFUSE_PGM_DATA7_REG	Register 7 that stores data to be programmed	0x001C	R/W
EFUSE_PGM_CHECK_VALUE0_REG	Register 0 that stores the RS code to be programmed	0x0020	R/W
EFUSE_PGM_CHECK_VALUE1_REG	Register 1 that stores the RS code to be programmed	0x0024	R/W
EFUSE_PGM_CHECK_VALUE2_REG	Register 2 that stores the RS code to be programmed	0x0028	R/W
Read Data Register			
EFUSE_RD_WR_DIS_REG	BLOCK0 data register 0	0x002C	RO
EFUSE_RD_REPEAT_DATA0_REG	BLOCK0 data register 1	0x0030	RO
EFUSE_RD_REPEAT_DATA1_REG	BLOCK0 data register 2	0x0034	RO
EFUSE_RD_REPEAT_DATA2_REG	BLOCK0 data register 3	0x0038	RO
EFUSE_RD_REPEAT_DATA3_REG	BLOCK0 data register 4	0x003C	RO
EFUSE_RD_REPEAT_DATA4_REG	BLOCK0 data register 5	0x0040	RO
EFUSE_RD_MAC_SPI_SYS_0_REG	BLOCK1 data register 0	0x0044	RO
EFUSE_RD_MAC_SPI_SYS_1_REG	BLOCK1 data register 1	0x0048	RO
EFUSE_RD_MAC_SPI_SYS_2_REG	BLOCK1 data register 2	0x004C	RO
EFUSE_RD_MAC_SPI_SYS_3_REG	BLOCK1 data register 3	0x0050	RO
EFUSE_RD_MAC_SPI_SYS_4_REG	BLOCK1 data register 4	0x0054	RO
EFUSE_RD_MAC_SPI_SYS_5_REG	BLOCK1 data register 5	0x0058	RO
EFUSE_RD_SYS_PART1_DATA0_REG	Register 0 of BLOCK2 (system)	0x005C	RO
EFUSE_RD_SYS_PART1_DATA1_REG	Register 1 of BLOCK2 (system)	0x0060	RO
EFUSE_RD_SYS_PART1_DATA2_REG	Register 2 of BLOCK2 (system)	0x0064	RO
EFUSE_RD_SYS_PART1_DATA3_REG	Register 3 of BLOCK2 (system)	0x0068	RO
EFUSE_RD_SYS_PART1_DATA4_REG	Register 4 of BLOCK2 (system)	0x006C	RO
EFUSE_RD_SYS_PART1_DATA5_REG	Register 5 of BLOCK2 (system)	0x0070	RO
EFUSE_RD_SYS_PART1_DATA6_REG	Register 6 of BLOCK2 (system)	0x0074	RO
EFUSE_RD_SYS_PART1_DATA7_REG	Register 7 of BLOCK2 (system)	0x0078	RO
EFUSE_RD_USR_DATA0_REG	Register 0 of BLOCK3 (user)	0x007C	RO
EFUSE_RD_USR_DATA1_REG	Register 1 of BLOCK3 (user)	0x0080	RO

Name	Description	Address	Access
EFUSE_RD_USR_DATA2_REG	Register 2 of BLOCK3 (user)	0x0084	RO
EFUSE_RD_USR_DATA3_REG	Register 3 of BLOCK3 (user)	0x0088	RO
EFUSE_RD_USR_DATA4_REG	Register 4 of BLOCK3 (user)	0x008C	RO
EFUSE_RD_USR_DATA5_REG	Register 5 of BLOCK3 (user)	0x0090	RO
EFUSE_RD_USR_DATA6_REG	Register 6 of BLOCK3 (user)	0x0094	RO
EFUSE_RD_USR_DATA7_REG	Register 7 of BLOCK3 (user)	0x0098	RO
EFUSE_RD_KEY0_DATA0_REG	Register 0 of BLOCK4 (KEY0)	0x009C	RO
EFUSE_RD_KEY0_DATA1_REG	Register 1 of BLOCK4 (KEY0)	0x00A0	RO
EFUSE_RD_KEY0_DATA2_REG	Register 2 of BLOCK4 (KEY0)	0x00A4	RO
EFUSE_RD_KEY0_DATA3_REG	Register 3 of BLOCK4 (KEY0)	0x00A8	RO
EFUSE_RD_KEY0_DATA4_REG	Register 4 of BLOCK4 (KEY0)	0x00AC	RO
EFUSE_RD_KEY0_DATA5_REG	Register 5 of BLOCK4 (KEY0)	0x00B0	RO
EFUSE_RD_KEY0_DATA6_REG	Register 6 of BLOCK4 (KEY0)	0x00B4	RO
EFUSE_RD_KEY0_DATA7_REG	Register 7 of BLOCK4 (KEY0)	0x00B8	RO
EFUSE_RD_KEY1_DATA0_REG	Register 0 of BLOCK5 (KEY1)	0x00BC	RO
EFUSE_RD_KEY1_DATA1_REG	Register 1 of BLOCK5 (KEY1)	0x00C0	RO
EFUSE_RD_KEY1_DATA2_REG	Register 2 of BLOCK5 (KEY1)	0x00C4	RO
EFUSE_RD_KEY1_DATA3_REG	Register 3 of BLOCK5 (KEY1)	0x00C8	RO
EFUSE_RD_KEY1_DATA4_REG	Register 4 of BLOCK5 (KEY1)	0x00CC	RO
EFUSE_RD_KEY1_DATA5_REG	Register 5 of BLOCK5 (KEY1)	0x00D0	RO
EFUSE_RD_KEY1_DATA6_REG	Register 6 of BLOCK5 (KEY1)	0x00D4	RO
EFUSE_RD_KEY1_DATA7_REG	Register 7 of BLOCK5 (KEY1)	0x00D8	RO
EFUSE_RD_KEY2_DATA0_REG	Register 0 of BLOCK6 (KEY2)	0x00DC	RO
EFUSE_RD_KEY2_DATA1_REG	Register 1 of BLOCK6 (KEY2)	0x00E0	RO
EFUSE_RD_KEY2_DATA2_REG	Register 2 of BLOCK6 (KEY2)	0x00E4	RO
EFUSE_RD_KEY2_DATA3_REG	Register 3 of BLOCK6 (KEY2)	0x00E8	RO
EFUSE_RD_KEY2_DATA4_REG	Register 4 of BLOCK6 (KEY2)	0x00EC	RO
EFUSE_RD_KEY2_DATA5_REG	Register 5 of BLOCK6 (KEY2)	0x00F0	RO
EFUSE_RD_KEY2_DATA6_REG	Register 6 of BLOCK6 (KEY2)	0x00F4	RO
EFUSE_RD_KEY2_DATA7_REG	Register 7 of BLOCK6 (KEY2)	0x00F8	RO
EFUSE_RD_KEY3_DATA0_REG	Register 0 of BLOCK7 (KEY3)	0x00FC	RO
EFUSE_RD_KEY3_DATA1_REG	Register 1 of BLOCK7 (KEY3)	0x0100	RO
EFUSE_RD_KEY3_DATA2_REG	Register 2 of BLOCK7 (KEY3)	0x0104	RO
EFUSE_RD_KEY3_DATA3_REG	Register 3 of BLOCK7 (KEY3)	0x0108	RO
EFUSE_RD_KEY3_DATA4_REG	Register 4 of BLOCK7 (KEY3)	0x010C	RO
EFUSE_RD_KEY3_DATA5_REG	Register 5 of BLOCK7 (KEY3)	0x0110	RO
EFUSE_RD_KEY3_DATA6_REG	Register 6 of BLOCK7 (KEY3)	0x0114	RO
EFUSE_RD_KEY3_DATA7_REG	Register 7 of BLOCK7 (KEY3)	0x0118	RO
EFUSE_RD_KEY4_DATA0_REG	Register 0 of BLOCK8 (KEY4)	0x011C	RO
EFUSE_RD_KEY4_DATA1_REG	Register 1 of BLOCK8 (KEY4)	0x0120	RO
EFUSE_RD_KEY4_DATA2_REG	Register 2 of BLOCK8 (KEY4)	0x0124	RO
EFUSE_RD_KEY4_DATA3_REG	Register 3 of BLOCK8 (KEY4)	0x0128	RO
EFUSE_RD_KEY4_DATA4_REG	Register 4 of BLOCK8 (KEY4)	0x012C	RO

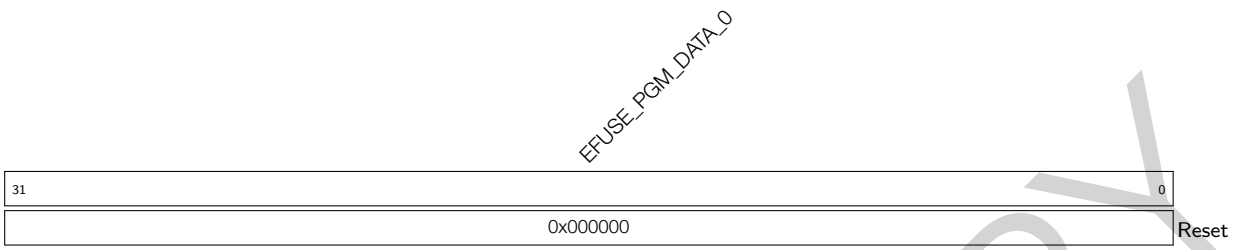
Name	Description	Address	Access
EFUSE_RD_KEY4_DATA5_REG	Register 5 of BLOCK8 (KEY4)	0x0130	RO
EFUSE_RD_KEY4_DATA6_REG	Register 6 of BLOCK8 (KEY4)	0x0134	RO
EFUSE_RD_KEY4_DATA7_REG	Register 7 of BLOCK8 (KEY4)	0x0138	RO
EFUSE_RD_KEY5_DATA0_REG	Register 0 of BLOCK9 (KEY5)	0x013C	RO
EFUSE_RD_KEY5_DATA1_REG	Register 1 of BLOCK9 (KEY5)	0x0140	RO
EFUSE_RD_KEY5_DATA2_REG	Register 2 of BLOCK9 (KEY5)	0x0144	RO
EFUSE_RD_KEY5_DATA3_REG	Register 3 of BLOCK9 (KEY5)	0x0148	RO
EFUSE_RD_KEY5_DATA4_REG	Register 4 of BLOCK9 (KEY5)	0x014C	RO
EFUSE_RD_KEY5_DATA5_REG	Register 5 of BLOCK9 (KEY5)	0x0150	RO
EFUSE_RD_KEY5_DATA6_REG	Register 6 of BLOCK9 (KEY5)	0x0154	RO
EFUSE_RD_KEY5_DATA7_REG	Register 7 of BLOCK9 (KEY5)	0x0158	RO
EFUSE_RD_SYS_PART2_DATA0_REG	Register 0 of BLOCK10 (system)	0x015C	RO
EFUSE_RD_SYS_PART2_DATA1_REG	Register 1 of BLOCK10 (system)	0x0160	RO
EFUSE_RD_SYS_PART2_DATA2_REG	Register 2 of BLOCK10 (system)	0x0164	RO
EFUSE_RD_SYS_PART2_DATA3_REG	Register 3 of BLOCK10 (system)	0x0168	RO
EFUSE_RD_SYS_PART2_DATA4_REG	Register 4 of BLOCK10 (system)	0x016C	RO
EFUSE_RD_SYS_PART2_DATA5_REG	Register 5 of BLOCK10 (system)	0x0170	RO
EFUSE_RD_SYS_PART2_DATA6_REG	Register 6 of BLOCK10 (system)	0x0174	RO
EFUSE_RD_SYS_PART2_DATA7_REG	Register 7 of BLOCK10 (system)	0x0178	RO
Report Register			
EFUSE_RD_REPEAT_ERR0_REG	Programming error record register 0 of BLOCK0	0x017C	RO
EFUSE_RD_REPEAT_ERR1_REG	Programming error record register 1 of BLOCK0	0x0180	RO
EFUSE_RD_REPEAT_ERR2_REG	Programming error record register 2 of BLOCK0	0x0184	RO
EFUSE_RD_REPEAT_ERR3_REG	Programming error record register 3 of BLOCK0	0x0188	RO
EFUSE_RD_REPEAT_ERR4_REG	Programming error record register 4 of BLOCK0	0x0190	RO
EFUSE_RD_RS_ERR0_REG	Programming error record register 0 of BLOCK1 ~ 10	0x01C0	RO
EFUSE_RD_RS_ERR1_REG	Programming error record register 1 of BLOCK1 ~ 10	0x01C4	RO
Configuration Register			
EFUSE_CLK_REG	eFuse clock configuration register	0x01C8	R/W
EFUSE_CONF_REG	eFuse operation mode configuration register	0x01CC	R/W
EFUSE_CMD_REG	eFuse command register	0x01D4	varies
EFUSE_DAC_CONF_REG	Controls the eFuse programming voltage	0x01E8	R/W
EFUSE_RD_TIM_CONF_REG	Configures read timing parameters	0x01EC	R/W
EFUSE_WR_TIM_CONF1_REG	Configuration register 1 of eFuse programming timing parameters	0x01F4	R/W
EFUSE_WR_TIM_CONF2_REG	Configuration register 2 of eFuse programming timing parameters	0x01F8	R/W
Status Register			
EFUSE_STATUS_REG	eFuse status register	0x01D0	RO
Interrupt Register			
EFUSE_INT_RAW_REG	eFuse raw interrupt register	0x01D8	R/WC/SS

Name	Description	Address	Access
EFUSE_INT_ST_REG	eFuse interrupt status register	0x01DC	RO
EFUSE_INT_ENA_REG	eFuse interrupt enable register	0x01E0	R/W
EFUSE_INT_CLR_REG	eFuse interrupt clear register	0x01E4	WO
Version Register			
EFUSE_DATE_REG	Version control register	0x01FC	R/W

4.5 Registers

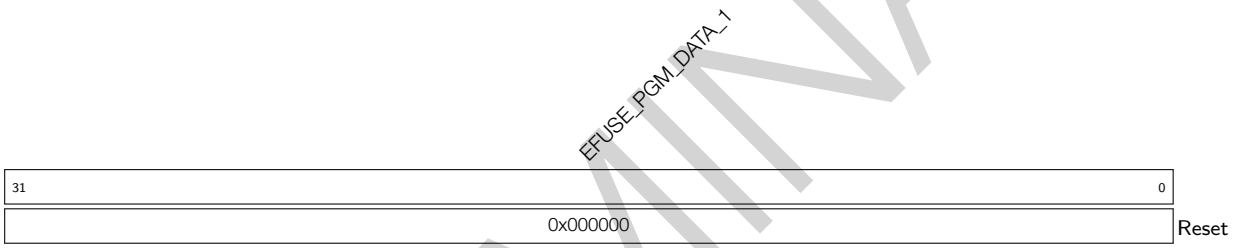
The addresses in this section are relative to eFuse Controller base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Register 4.1. EFUSE_PGM_DATA0_REG (0x0000)



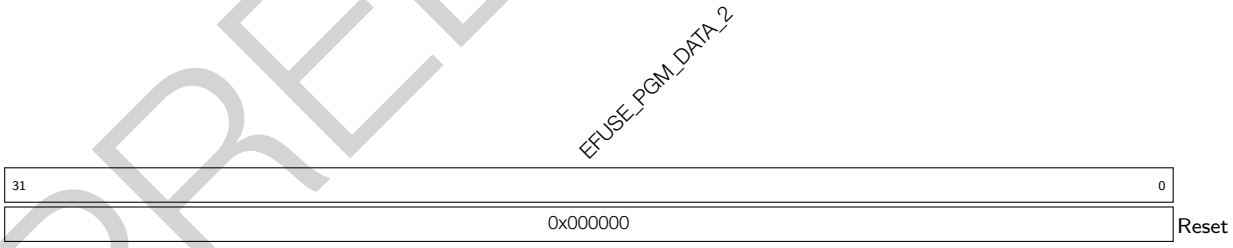
EFUSE_PGM_DATA_0 The content of the 0th 32-bit data to be programmed. (R/W)

Register 4.2. EFUSE_PGM_DATA1_REG (0x0004)



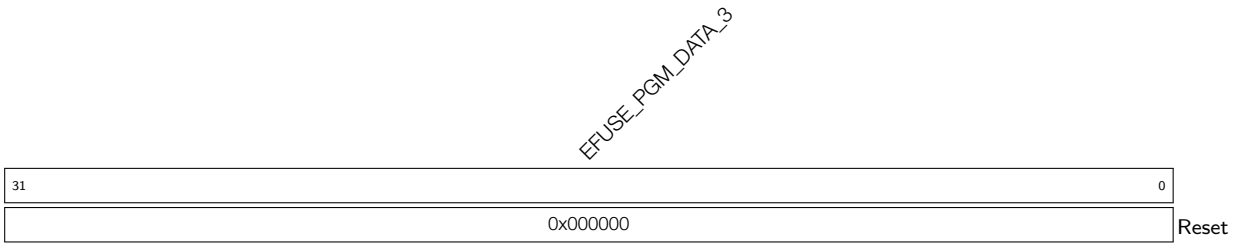
EFUSE_PGM_DATA_1 The content of the 1st 32-bit data to be programmed. (R/W)

Register 4.3. EFUSE_PGM_DATA2_REG (0x0008)



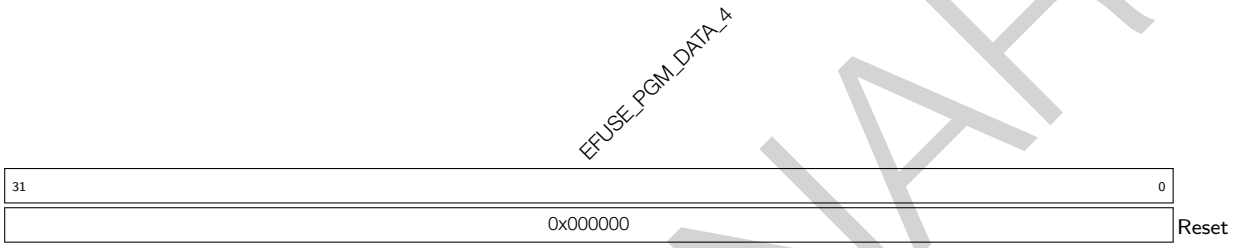
EFUSE_PGM_DATA_2 The content of the 2nd 32-bit data to be programmed. (R/W)

Register 4.4. EFUSE_PGM_DATA3_REG (0x000C)



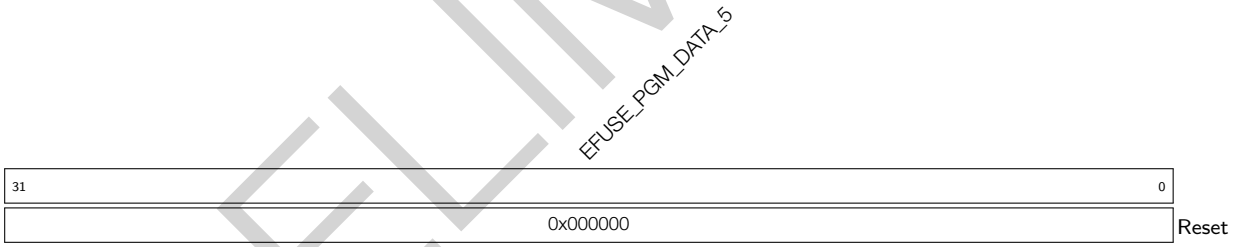
EFUSE_PGM_DATA_3 The content of the 3rd 32-bit data to be programmed. (R/W)

Register 4.5. EFUSE_PGM_DATA4_REG (0x0010)



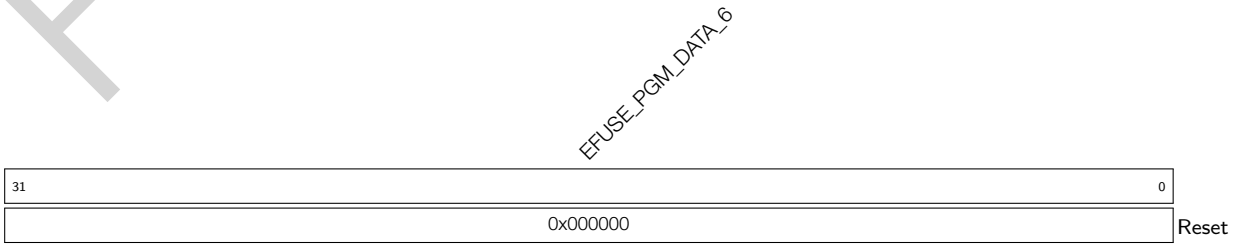
EFUSE_PGM_DATA_4 The content of the 4th 32-bit data to be programmed. (R/W)

Register 4.6. EFUSE_PGM_DATA5_REG (0x0014)



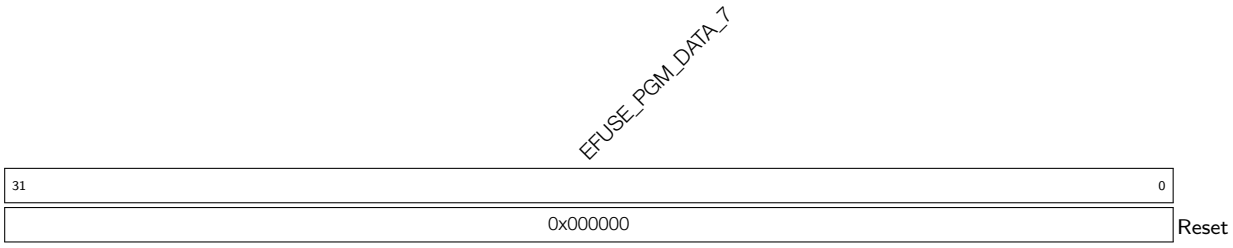
EFUSE_PGM_DATA_5 The content of the 5th 32-bit data to be programmed. (R/W)

Register 4.7. EFUSE_PGM_DATA6_REG (0x0018)



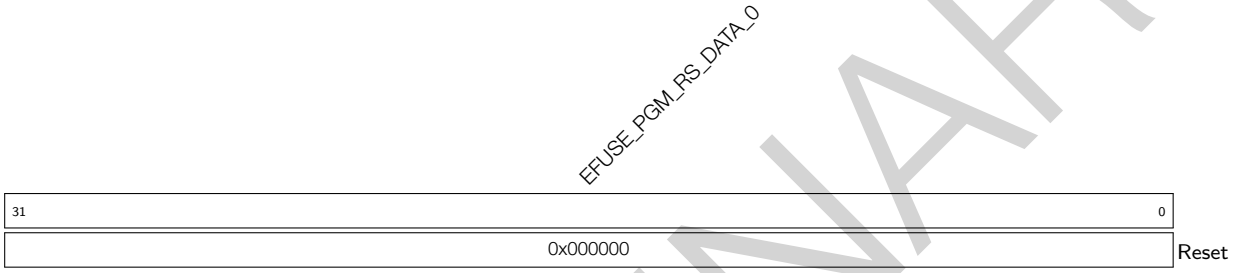
EFUSE_PGM_DATA_6 The content of the 6th 32-bit data to be programmed. (R/W)

Register 4.8. EFUSE_PGM_DATA7_REG (0x001C)



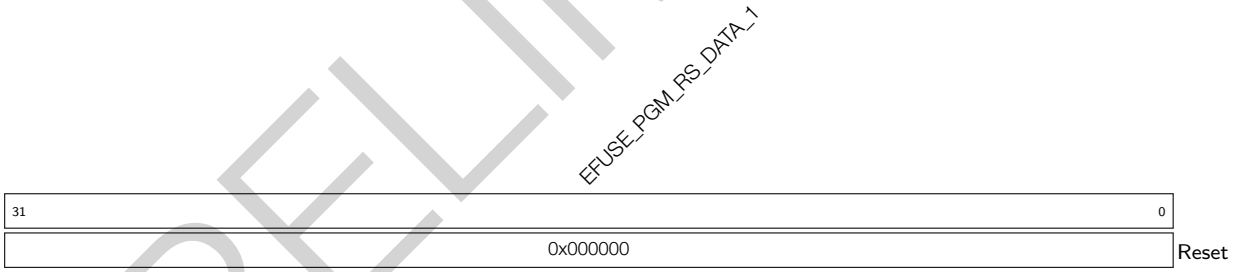
EFUSE_PGM_DATA_7 The content of the 7th 32-bit data to be programmed. (R/W)

Register 4.9. EFUSE_PGM_CHECK_VALUE0_REG (0x0020)



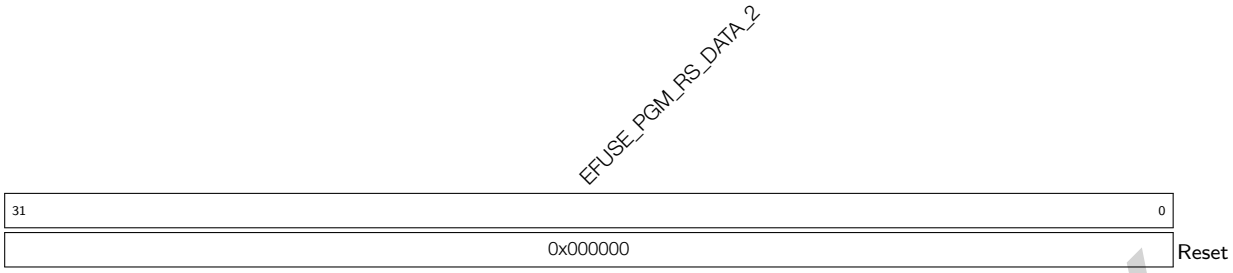
EFUSE_PGM_RS_DATA_0 The content of the 0th 32-bit RS code to be programmed. (R/W)

Register 4.10. EFUSE_PGM_CHECK_VALUE1_REG (0x0024)



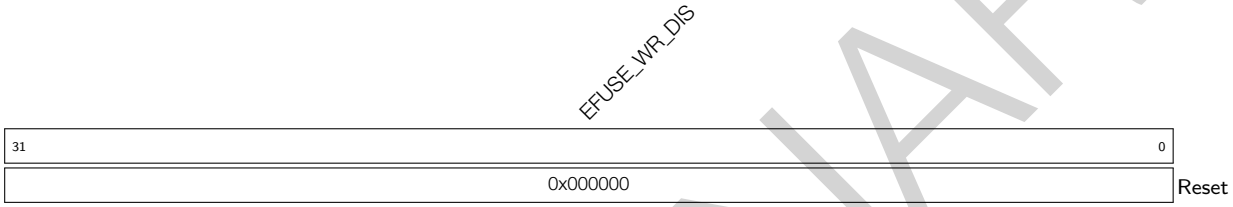
EFUSE_PGM_RS_DATA_1 The content of the 1st 32-bit RS code to be programmed. (R/W)

Register 4.11. EFUSE_PGM_CHECK_VALUE2_REG (0x0028)



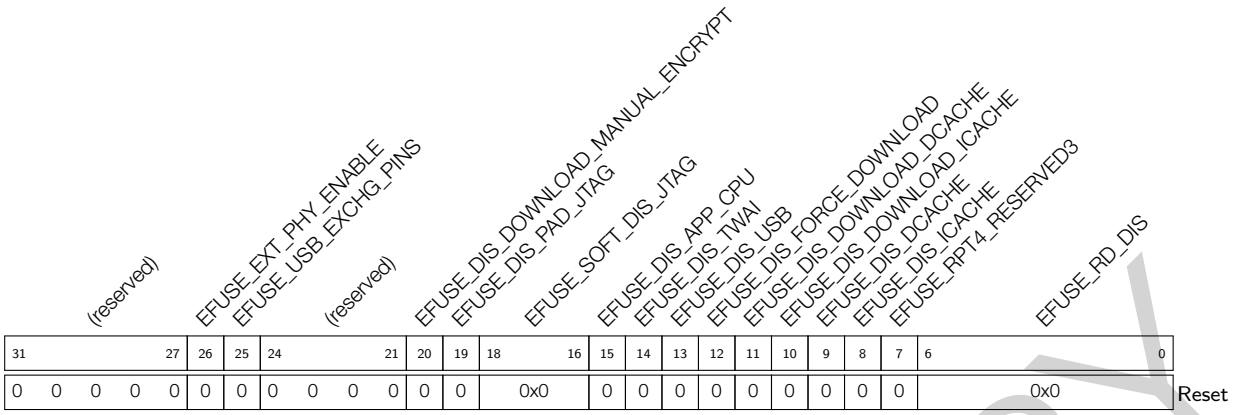
EFUSE_PGM_RS_DATA_2 The content of the 2nd 32-bit RS code to be programmed. (R/W)

Register 4.12. EFUSE_RD_WR_DIS_REG (0x002C)



EFUSE_WR_DIS Disable programming of individual eFuses. (RO)

Register 4.13. EFUSE_RD_REPEAT_DATA0_REG (0x0030)



EFUSE_RD_DIS Set this bit to disable reading from BlOCK4 ~ 10. (RO)

EFUSE_RPT4_RESERVED3 Reserved (used for four backups method). (RO)

EFUSE_DIS_ICACHE Set this bit to disable Icache. (RO)

EFUSE_DIS_DCACHE Set this bit to disable Dcache. (RO)

EFUSE_DIS_DOWNLOAD_ICACHE Set this bit to disable Icache in download mode (boot_mode[3:0] is 0, 1, 2, 3, 6, 7). (RO)

EFUSE_DIS_DOWNLOAD_DCACHE Set this bit to disable Dcache in download mode (boot_mode[3:0] is 0, 1, 2, 3, 6, 7). (RO)

EFUSE_DIS_FORCE_DOWNLOAD Set this bit to disable the function that forces chip into download mode. (RO)

EFUSE_DIS_USB Set this bit to disable USB function. (RO)

EFUSE_DIS_TWAI Set this bit to disable TWAI function. (RO)

EFUSE_DIS_APP_CPU Disable app cpu. (RO)

EFUSE_SOFT_DIS_JTAG Set these bits to disable JTAG in the soft way (odd number 1 means disable). JTAG can be enabled in HMAC module. (RO)

EFUSE_DIS_PAD_JTAG Set this bit to disable JTAG in the hard way. JTAG is disabled permanently. (RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT Set this bit to disable flash encryption when in download boot modes. (RO)

EFUSE_USB_EXCHG_PINS Set this bit to exchange USB D+ and D- pins. (RO)

EFUSE_EXT_PHY_ENABLE Set this bit to enable external PHY. (RO)

Register 4.14. EFUSE_RD_REPEAT_DATA1_REG (0x0034)

EFUSE_VDD_SPI_XPD SPI regulator power up signal. (RO)

EFUSE_VDD_SPI_TIEH SPI regulator output is short connected to VDD3P3_RTC_IO. (RO)

EFUSE_VDD_SPI_FORCE Set this bit and force to use the configuration of eFuse to configure VDD_SPI. (RO)

EFUSE_WDT_DELAY_SEL Selects RTC watchdog timeout threshold, in unit of slow clock cycle. 00: 40000, 01: 80000, 10: 160000, 11:320000. (RO)

EFUSE_SPI_BOOT_CRYPT_CNT Set this bit to enable SPI boot encrypt/decrypt. Odd number of 1: enable. even number of 1: disable. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0 Set this bit to enable revoking first secure boot key. (RO)

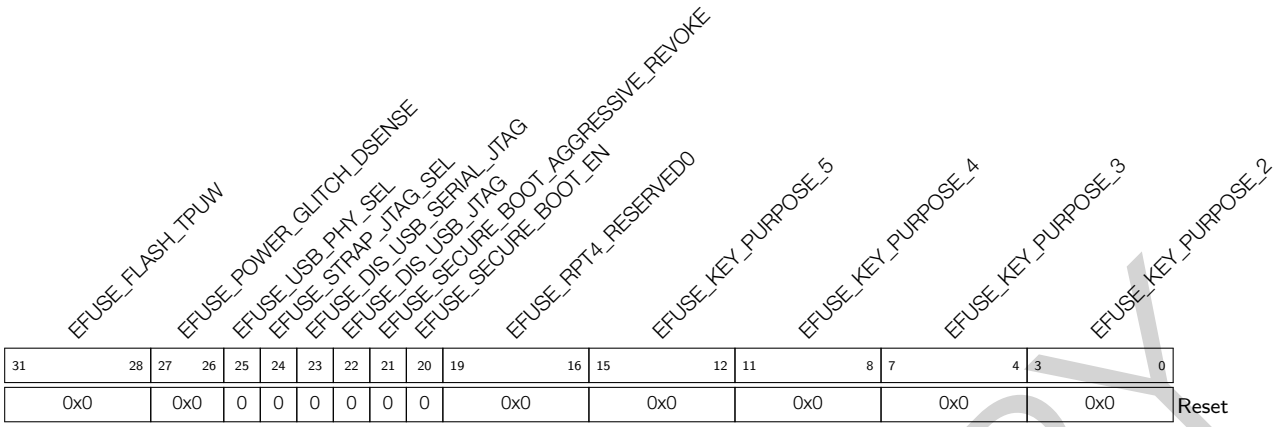
EFUSE_SECURE_BOOT_KEY_REVOKE1 Set this bit to enable revoking second secure boot key. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2 Set this bit to enable revoking third secure boot key. (RO)

EFUSE_KEY_PURPOSE_0 Purpose of Key0. (RO)

EFUSE_KEY_PURPOSE_1 Purpose of Key1. (RO)

Register 4.15. EFUSE_RD_REPEAT_DATA2_REG (0x0038)



EFUSE_KEY_PURPOSE_2 Purpose of Key2. (RO)

EFUSE_KEY_PURPOSE_3 Purpose of Key3. (RO)

EFUSE_KEY_PURPOSE_4 Purpose of Key4. (RO)

EFUSE_KEY_PURPOSE_5 Purpose of Key5. (RO)

EFUSE_RPT4_RESERVED0 Reserved (used for four backups method). (RO)

EFUSE_SECURE_BOOT_EN Set this bit to enable secure boot. (RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE Set this bit to enable revoking aggressive secure boot. (RO)

EFUSE_DIS_USB_JTAG Set this bit to disable function of usb switch to jtag in module of usb_serial_jtag device. (RO)

EFUSE_DIS_USB_SETIAL_JTAG Set this bit to disable usb_serial_jtag module. (RO)

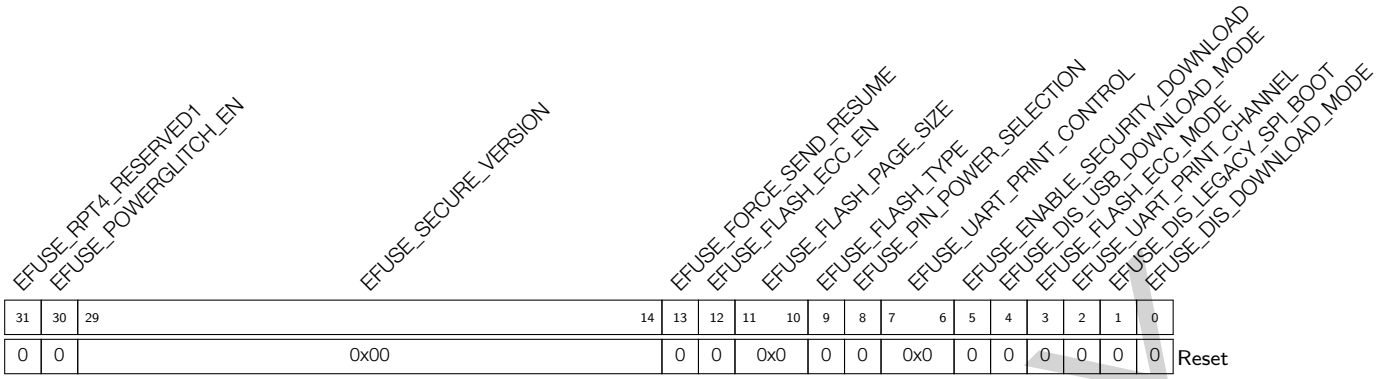
EFUSE_STRAP_JTAG_SEL Set this bit to enable selection between usb_to_jtag and pad_to_jtag through strapping gpio10 when both reg_dis_usb_jtag and reg_dis_pad_jtag are equal to 0. (RO)

EFUSE_USB_PHY_SEL This bit is used to switch internal PHY and external PHY for USB OTG and USB Device. 0: internal PHY is assigned to USB Device while external PHY is assigned to USB OTG. 1: internal PHY is assigned to USB OTG while external PHY is assigned to USB Device. (RO)

EFUSE_POWER_GLITCH_DSENSE Sample delay configuration of power glitch. (RO)

EFUSE_FLASH_TPUW Configures flash waiting time after power-up, in unit of ms. If the value is less than 15, the waiting time is the configurable value. Otherwise, the waiting time is twice the configurable value. (RO)

Register 4.16. EFUSE_RD_REPEAT_DATA3_REG (0x003C)



EFUSE_DIS_DOWNLOAD_MODE Set this bit to disable download mode (boot_mode[3:0] = 0, 1, 2, 3, 6, 7). (RO)

EFUSE_DIS_LEGACY_SPI_BOOT Set this bit to disable Legacy SPI boot mode (boot_mode[3:0] = 4). (RO)

EFUSE_UART_PRINT_CHANNEL Selects the default UART print channel. 0: UART0. 1: UART1. (RO)

EFUSE_FLASH_ECC_MODE Set ECC mode in ROM, 0: ROM would Enable Flash ECC 16-to-18 byte mode. 1: ROM would use 16-to-17 byte mode. (RO)

EFUSE_DIS_USB_DOWNLOAD_MODE Set this bit to disable UART download mode through USB. (RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD Set this bit to enable secure UART download mode. (RO)

EFUSE_UART_PRINT_CONTROL Set the default UART boot message output mode. 00: Enabled. 01: Enabled when GPIO8 is low at reset. 10: Enabled when GPIO8 is high at reset. 11:disabled. (RO)

EFUSE_PIN_POWER_SELECTION GPIO33 ~ GPIO37 power supply selection in ROM code. 0: VDD3P3_CPU. 1: VDD_SPI. (RO)

EFUSE_FLASH_TYPE Set the maximum lines of SPI flash. 0: four lines. 1: eight lines. (RO)

EFUSE_FLASH_PAGE_SIZE Set Flash page size. (RO)

EFUSE_FLASH_ECC_EN Set 1 to enable ECC for flash boot. (RO)

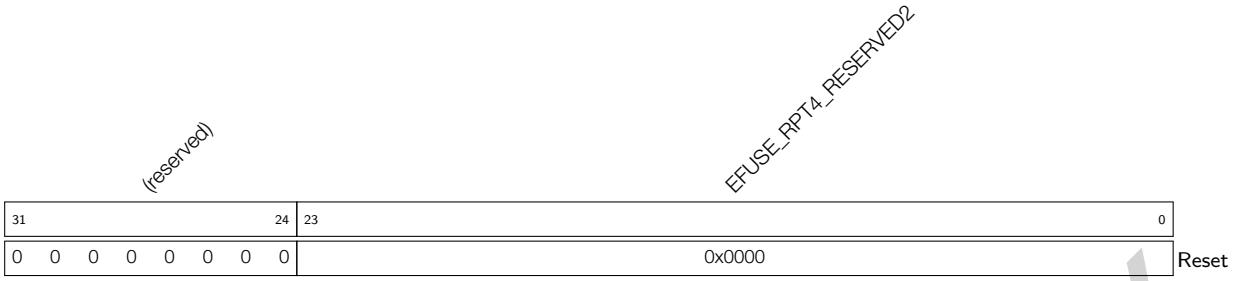
EFUSE_FORCE_SEND_RESUME Set this bit to force ROM code to send a resume command during SPI boot. (RO)

EFUSE_SECURE_VERSION Secure version (used by ESP-IDF anti-rollback feature). (RO)

EFUSE_POWERGLITCH_EN Set this bit to enable power glitch function. (RO)

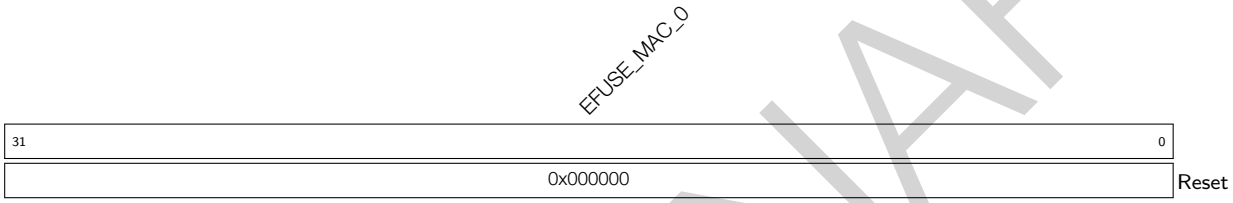
EFUSE_RPT4_RESERVED1 Reserved (used for four backups method). (RO)

Register 4.17. EFUSE_RD_REPEAT_DATA4_REG (0x0040)



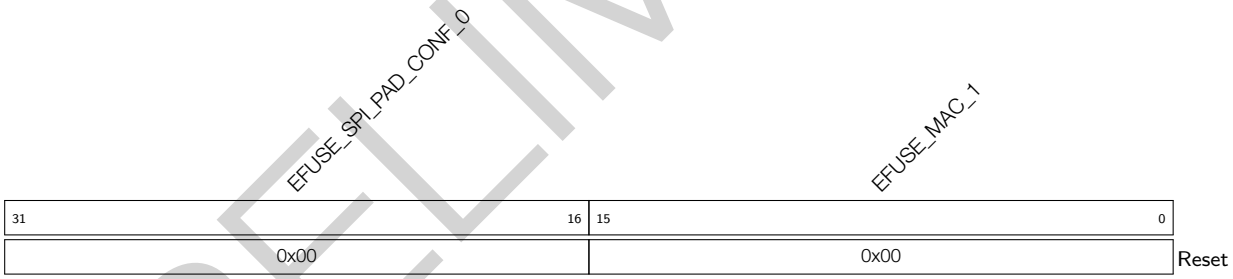
EFUSE_RPT4_RESERVED2 Reserved (used for four backups method). (RO)

Register 4.18. EFUSE_RD_MAC_SPI_SYS_0_REG (0x0044)



EFUSE_MAC_0 Stores the low 32 bits of MAC address. (RO)

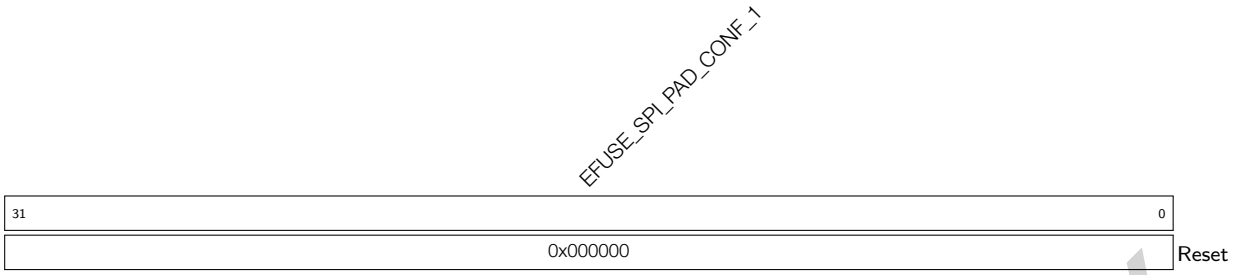
Register 4.19. EFUSE_RD_MAC_SPI_SYS_1_REG (0x0048)



EFUSE_MAC_1 Stores the high 16 bits of MAC address. (RO)

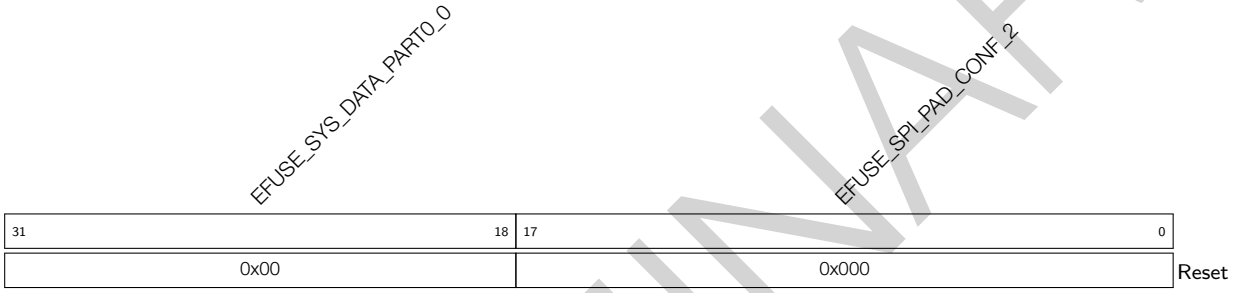
EFUSE_SPI_PAD_CONF_0 Stores the zeroth part of SPI_PAD_CONF. (RO)

Register 4.20. EFUSE_RD_MAC_SPI_SYS_2_REG (0x004C)



EFUSE_SPI_PAD_CONF_1 Stores the first part of SPI_PAD_CONF. (RO)

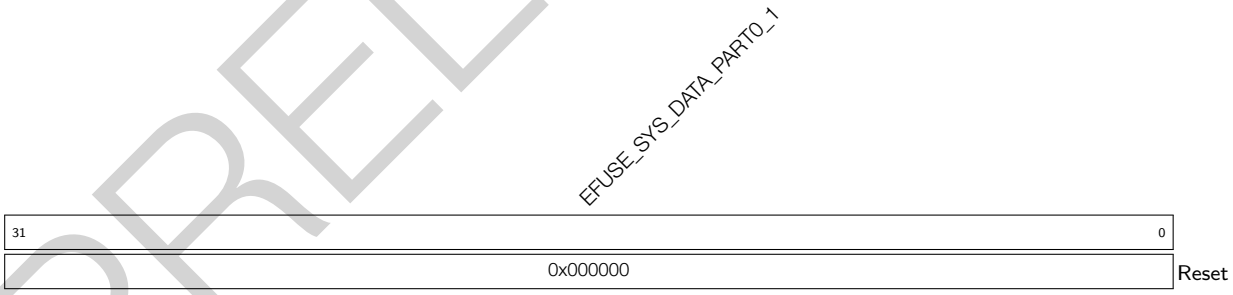
Register 4.21. EFUSE_RD_MAC_SPI_SYS_3_REG (0x0050)



EFUSE_SPI_PAD_CONF_2 Stores the second part of SPI_PAD_CONF. (RO)

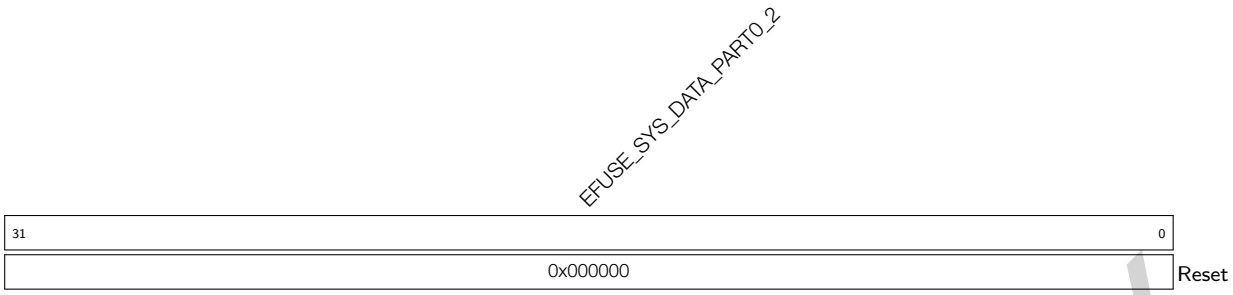
EFUSE_SYS_DATA_PART0_0 Stores the fist 14 bits of the zeroth part of system data. (RO)

Register 4.22. EFUSE_RD_MAC_SPI_SYS_4_REG (0x0054)



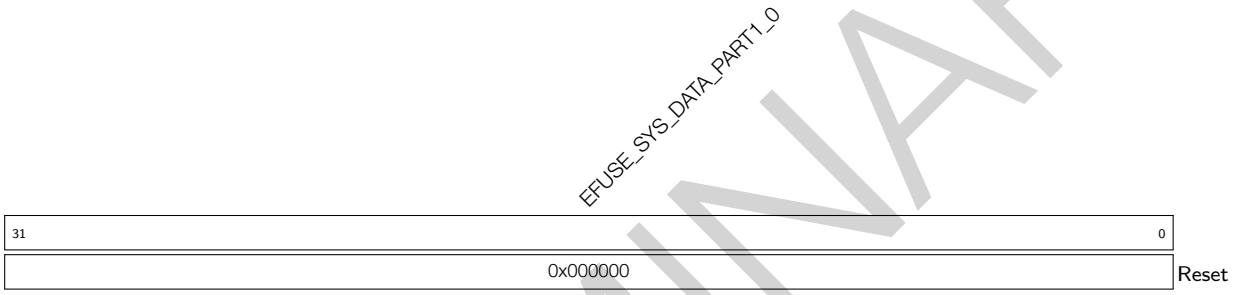
EFUSE_SYS_DATA_PART0_1 Stores the fist 32 bits of the zeroth part of system data. (RO)

Register 4.23. EFUSE_RD_MAC_SPI_SYS_5_REG (0x0058)



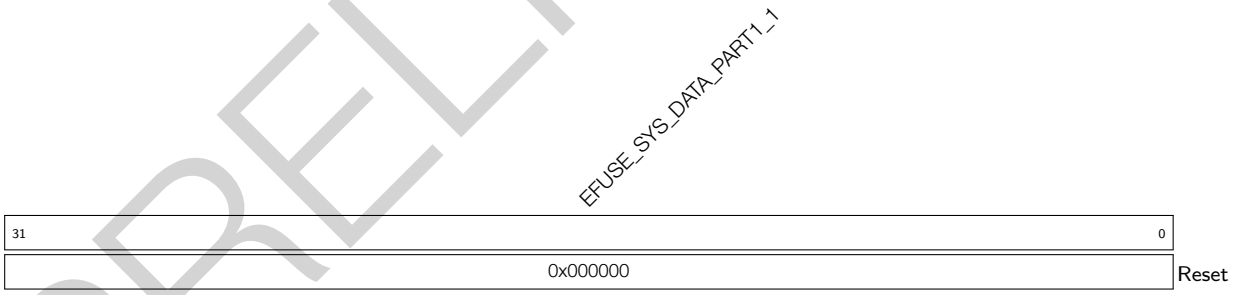
EFUSE_SYS_DATA_PART0_2 Stores the second 32 bits of the zeroth part of system data. (RO)

Register 4.24. EFUSE_RD_SYS_PART1_DATA0_REG (0x005C)



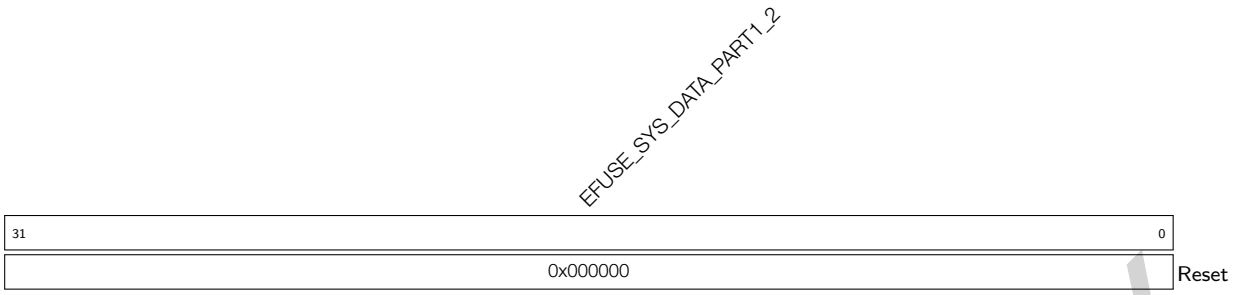
EFUSE_SYS_DATA_PART1_0 Stores the zeroth 32 bits of the first part of system data. (RO)

Register 4.25. EFUSE_RD_SYS_PART1_DATA1_REG (0x0060)



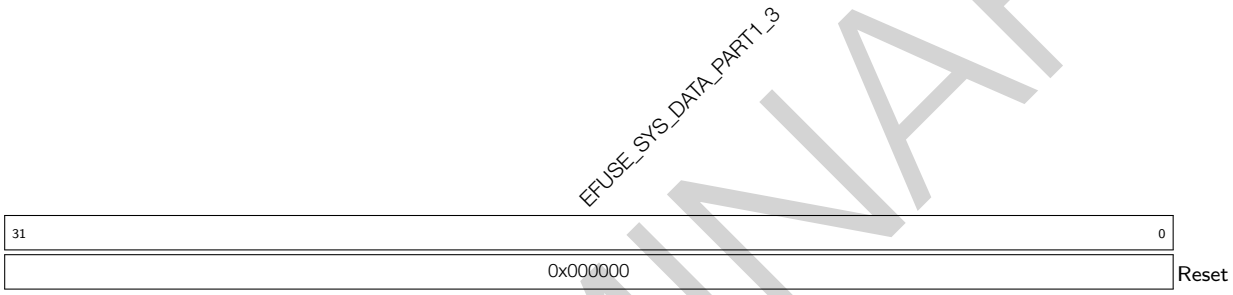
EFUSE_SYS_DATA_PART1_1 Stores the first 32 bits of the first part of system data. (RO)

Register 4.26. EFUSE_RD_SYS_PART1_DATA2_REG (0x0064)



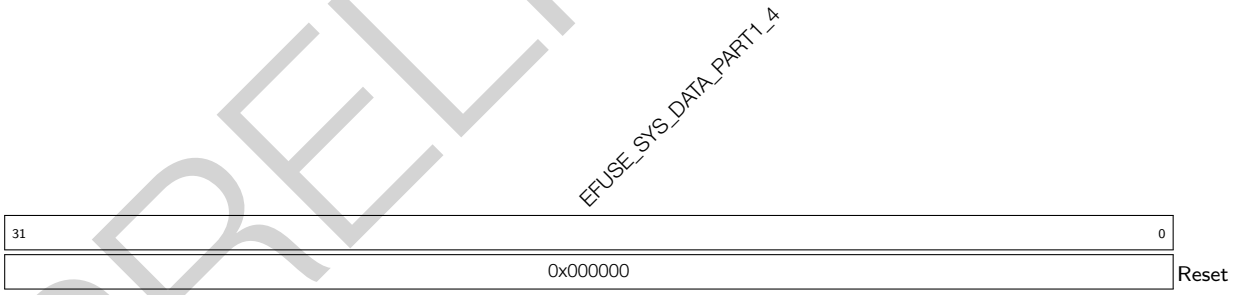
EFUSE_SYS_DATA_PART1_2 Stores the second 32 bits of the first part of system data. (RO)

Register 4.27. EFUSE_RD_SYS_PART1_DATA3_REG (0x0068)



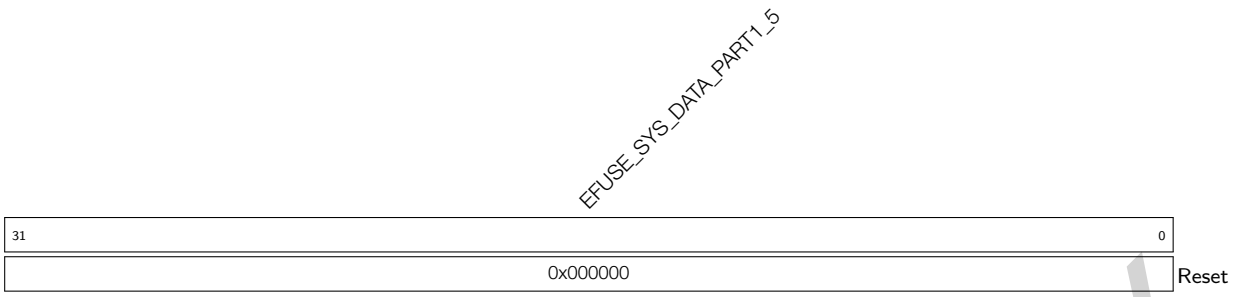
EFUSE_SYS_DATA_PART1_3 Stores the third 32 bits of the first part of system data. (RO)

Register 4.28. EFUSE_RD_SYS_PART1_DATA4_REG (0x006C)



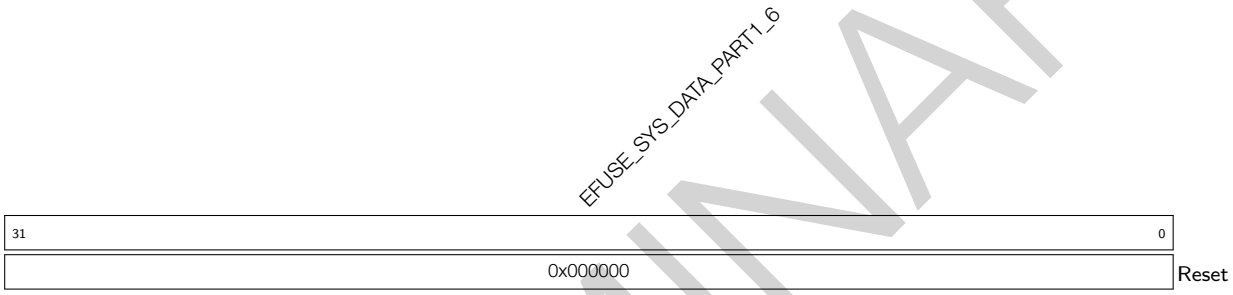
EFUSE_SYS_DATA_PART1_4 Stores the fourth 32 bits of the first part of system data. (RO)

Register 4.29. EFUSE_RD_SYS_PART1_DATA5_REG (0x0070)



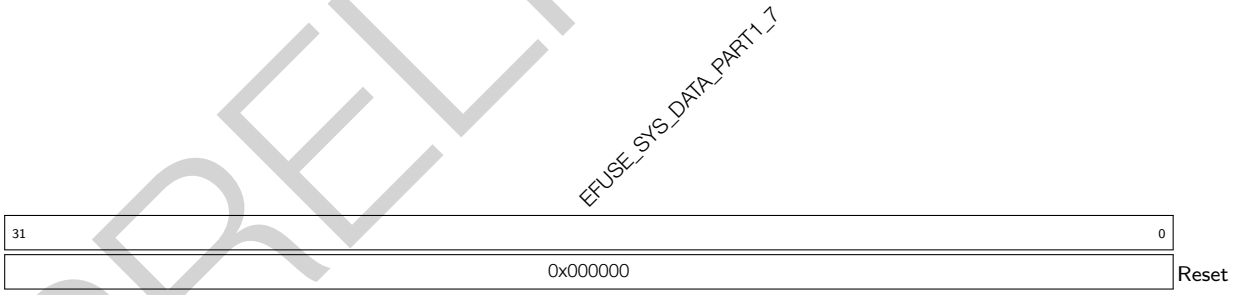
EFUSE_SYS_DATA_PART1_5 Stores the fifth 32 bits of the first part of system data. (RO)

Register 4.30. EFUSE_RD_SYS_PART1_DATA6_REG (0x0074)



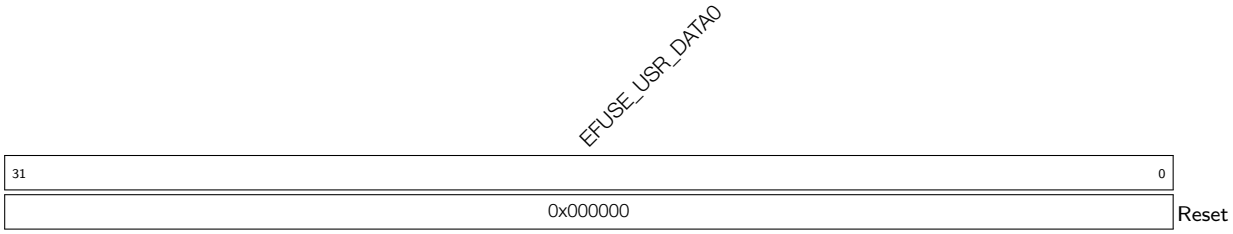
EFUSE_SYS_DATA_PART1_6 Stores the sixth 32 bits of the first part of system data. (RO)

Register 4.31. EFUSE_RD_SYS_PART1_DATA7_REG (0x0078)



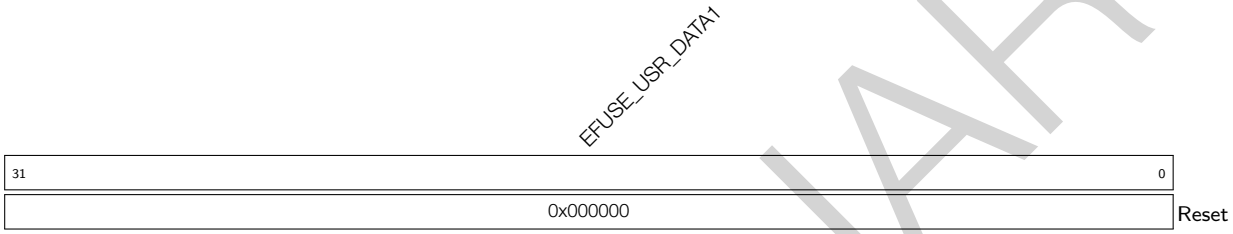
EFUSE_SYS_DATA_PART1_7 Stores the seventh 32 bits of the first part of system data. (RO)

Register 4.32. EFUSE_RD_USR_DATA0_REG (0x007C)



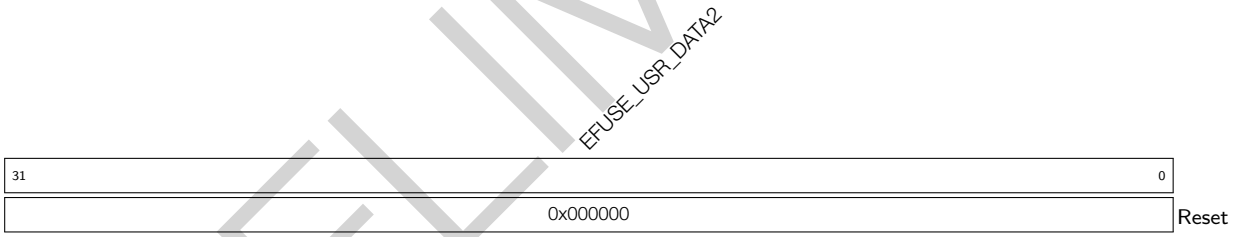
EFUSE_USR_DATA0 Stores the zeroth 32 bits of BLOCK3 (user). (RO)

Register 4.33. EFUSE_RD_USR_DATA1_REG (0x0080)



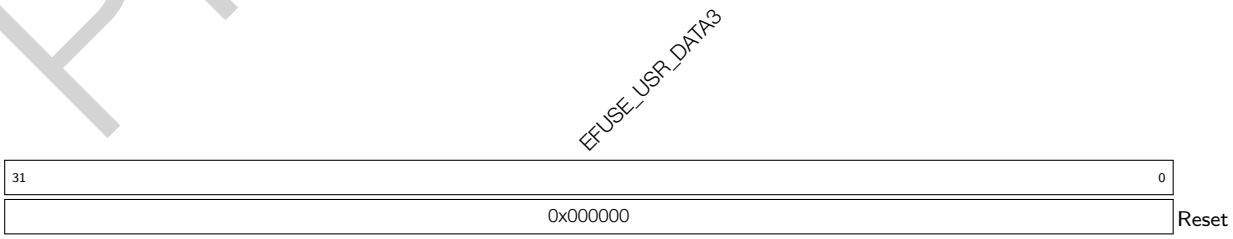
EFUSE_USR_DATA1 Stores the first 32 bits of BLOCK3 (user). (RO)

Register 4.34. EFUSE_RD_USR_DATA2_REG (0x0084)



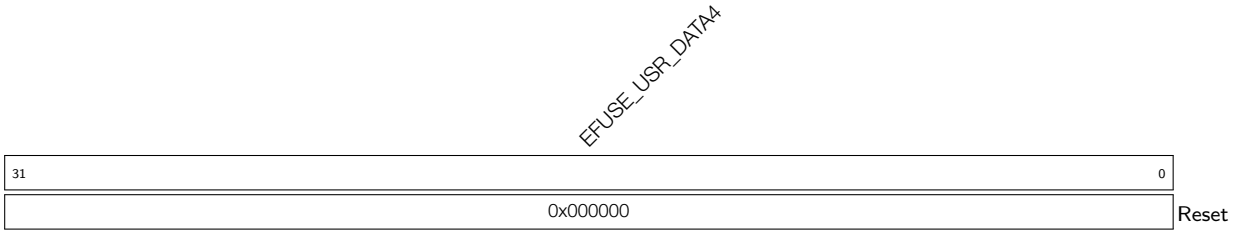
EFUSE_USR_DATA2 Stores the second 32 bits of BLOCK3 (user). (RO)

Register 4.35. EFUSE_RD_USR_DATA3_REG (0x0088)



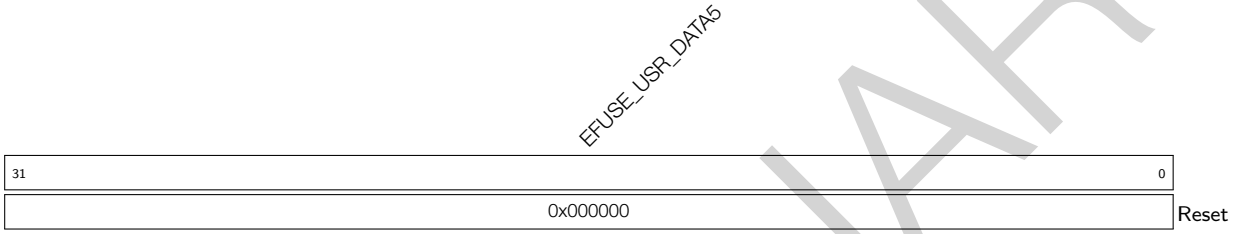
EFUSE_USR_DATA3 Stores the third 32 bits of BLOCK3 (user). (RO)

Register 4.36. EFUSE_RD_USR_DATA4_REG (0x008C)



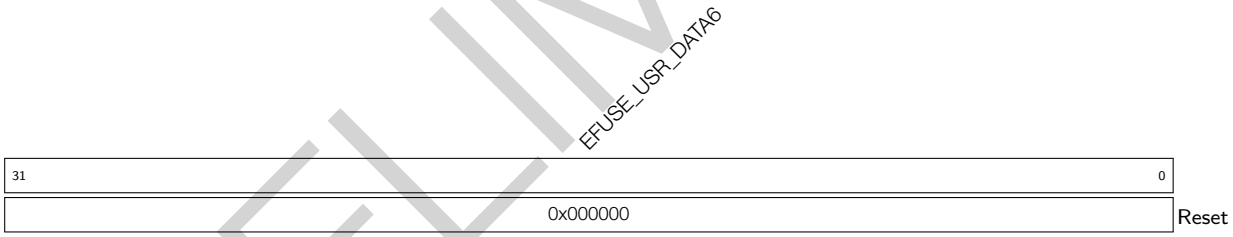
EFUSE_USR_DATA4 Stores the fourth 32 bits of BLOCK3 (user). (RO)

Register 4.37. EFUSE_RD_USR_DATA5_REG (0x0090)



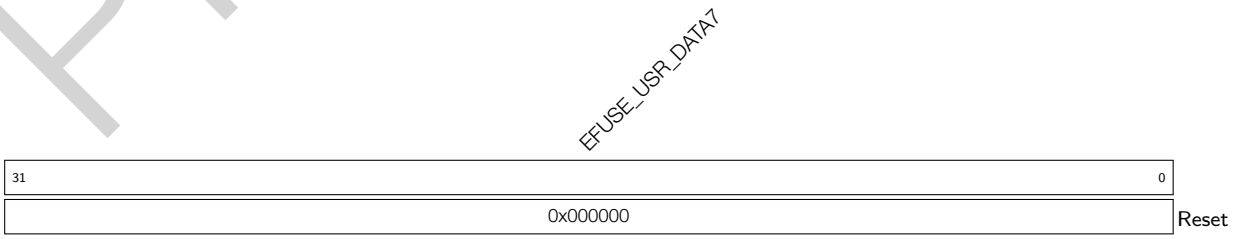
EFUSE_USR_DATA5 Stores the fifth 32 bits of BLOCK3 (user). (RO)

Register 4.38. EFUSE_RD_USR_DATA6_REG (0x0094)



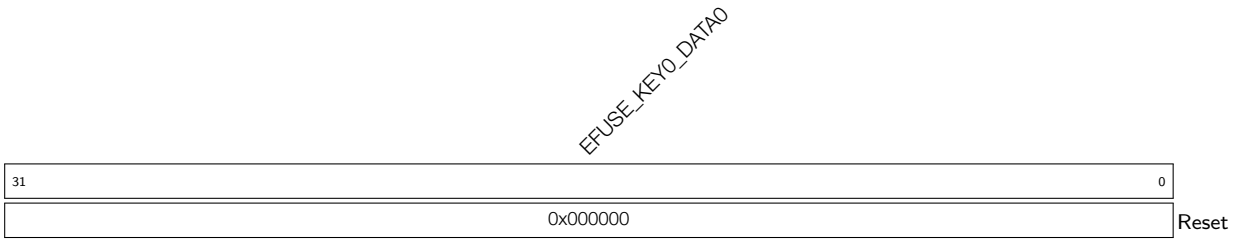
EFUSE_USR_DATA6 Stores the sixth 32 bits of BLOCK3 (user). (RO)

Register 4.39. EFUSE_RD_USR_DATA7_REG (0x0098)



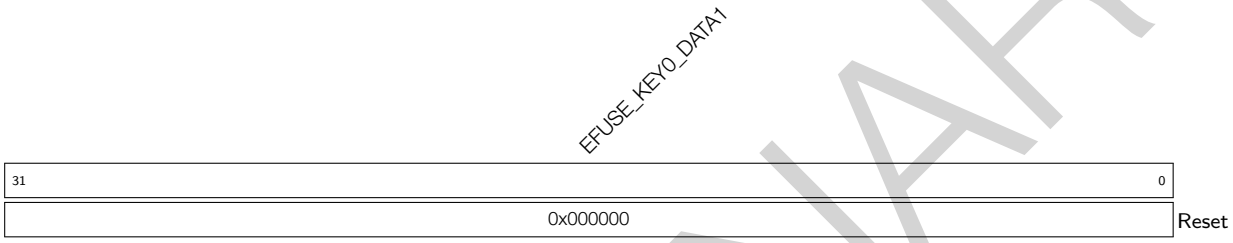
EFUSE_USR_DATA7 Stores the seventh 32 bits of BLOCK3 (user). (RO)

Register 4.40. EFUSE_RD_KEY0_DATA0_REG (0x009C)



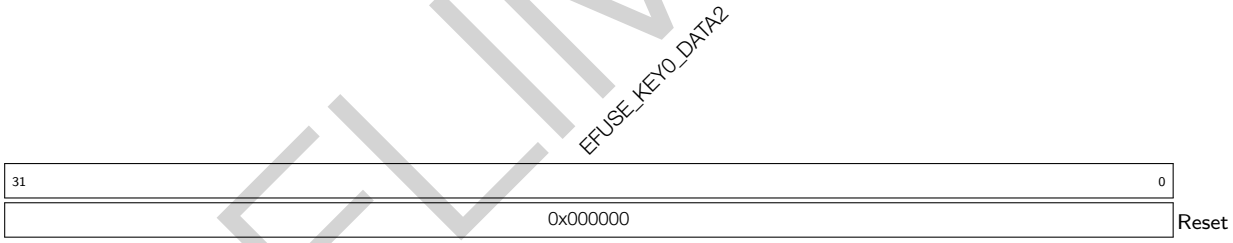
EFUSE_KEY0_DATA0 Stores the zeroth 32 bits of KEY0. (RO)

Register 4.41. EFUSE_RD_KEY0_DATA1_REG (0x00A0)



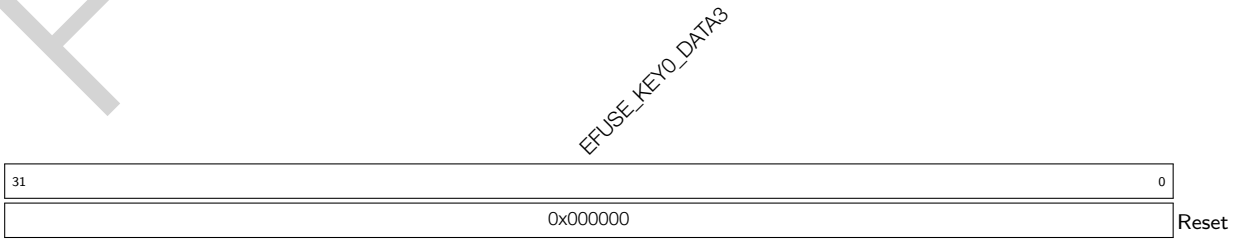
EFUSE_KEY0_DATA1 Stores the first 32 bits of KEY0. (RO)

Register 4.42. EFUSE_RD_KEY0_DATA2_REG (0x00A4)



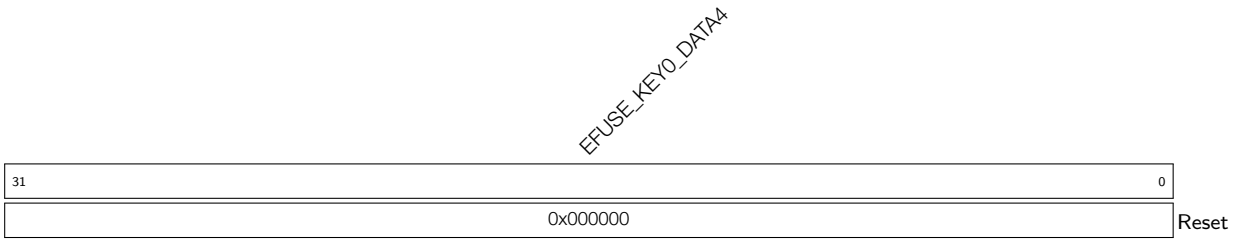
EFUSE_KEY0_DATA2 Stores the second 32 bits of KEY0. (RO)

Register 4.43. EFUSE_RD_KEY0_DATA3_REG (0x00A8)



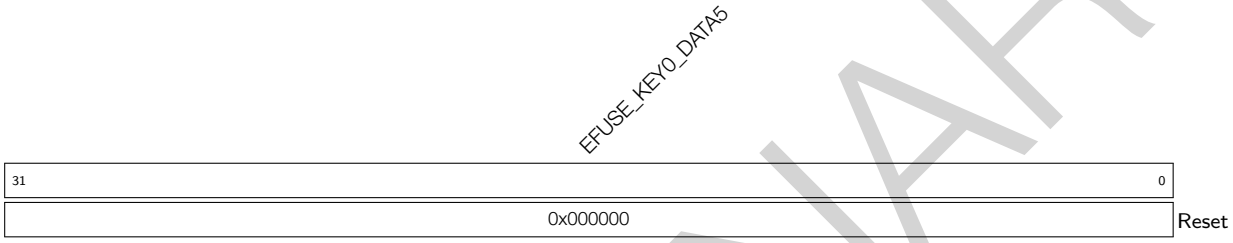
EFUSE_KEY0_DATA3 Stores the third 32 bits of KEY0. (RO)

Register 4.44. EFUSE_RD_KEY0_DATA4_REG (0x00AC)



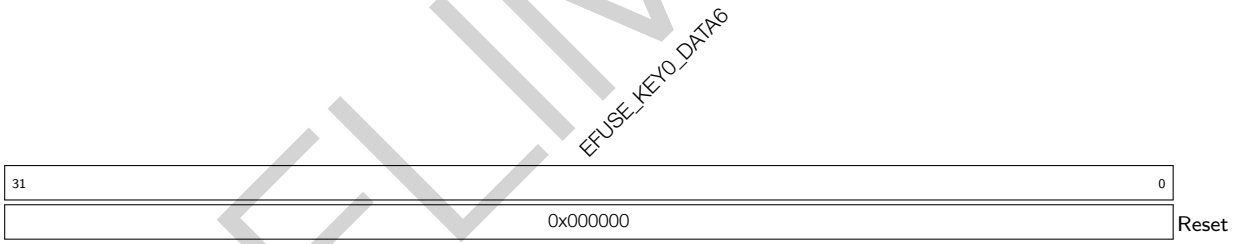
EFUSE_KEY0_DATA4 Stores the fourth 32 bits of KEY0. (RO)

Register 4.45. EFUSE_RD_KEY0_DATA5_REG (0x00B0)



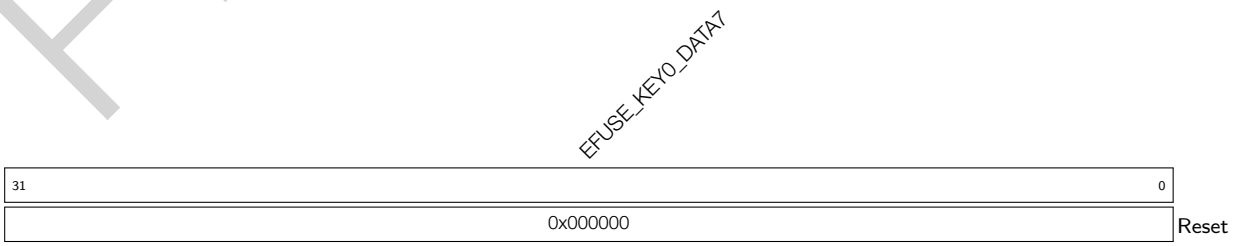
EFUSE_KEY0_DATA5 Stores the fifth 32 bits of KEY0. (RO)

Register 4.46. EFUSE_RD_KEY0_DATA6_REG (0x00B4)



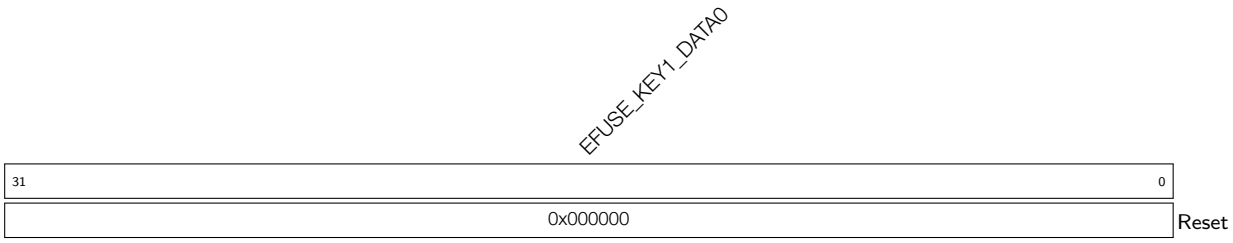
EFUSE_KEY0_DATA6 Stores the sixth 32 bits of KEY0. (RO)

Register 4.47. EFUSE_RD_KEY0_DATA7_REG (0x00B8)



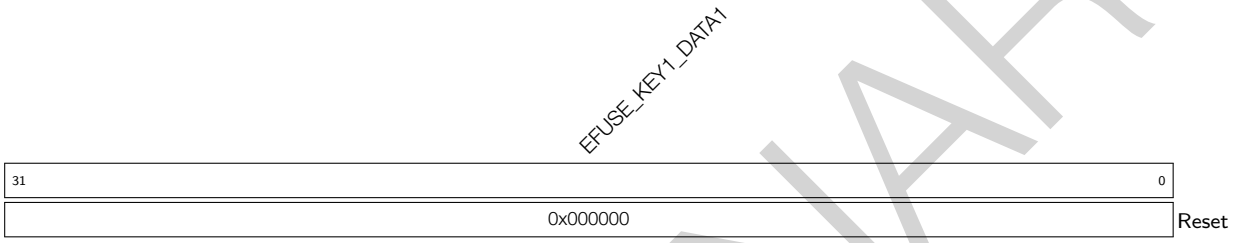
EFUSE_KEY0_DATA7 Stores the seventh 32 bits of KEY0. (RO)

Register 4.48. EFUSE_RD_KEY1_DATA0_REG (0x00BC)



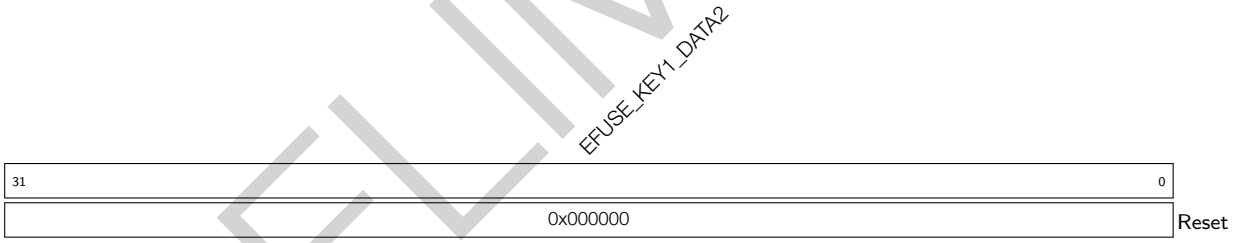
EFUSE_KEY1_DATA0 Stores the zeroth 32 bits of KEY1. (RO)

Register 4.49. EFUSE_RD_KEY1_DATA1_REG (0x00C0)



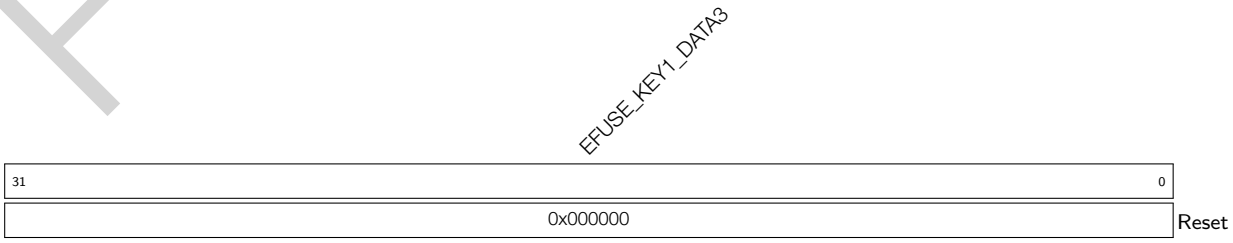
EFUSE_KEY1_DATA1 Stores the first 32 bits of KEY1. (RO)

Register 4.50. EFUSE_RD_KEY1_DATA2_REG (0x00C4)



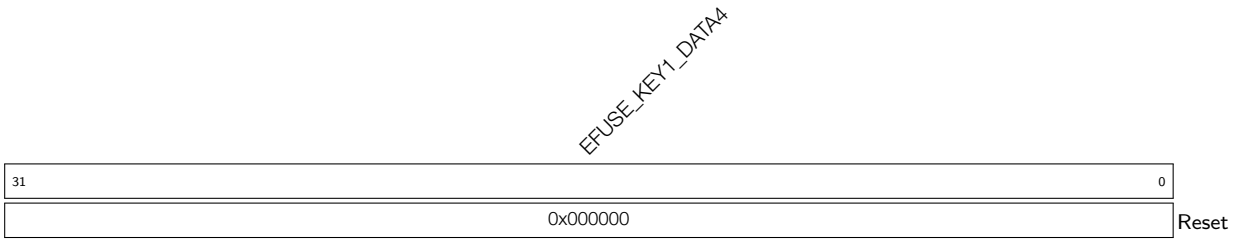
EFUSE_KEY1_DATA2 Stores the second 32 bits of KEY1. (RO)

Register 4.51. EFUSE_RD_KEY1_DATA3_REG (0x00C8)



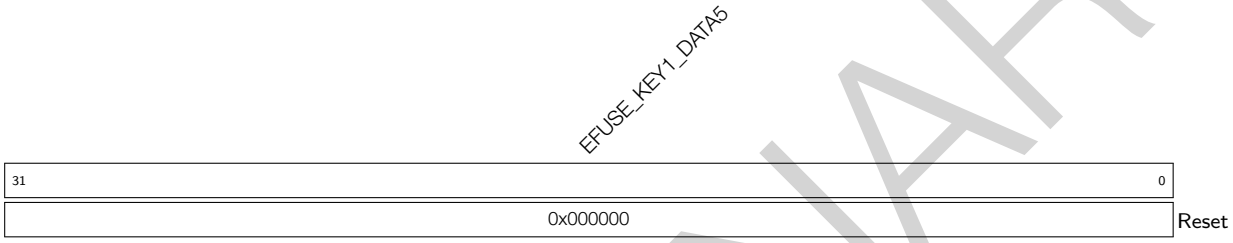
EFUSE_KEY1_DATA3 Stores the third 32 bits of KEY1. (RO)

Register 4.52. EFUSE_RD_KEY1_DATA4_REG (0x00CC)



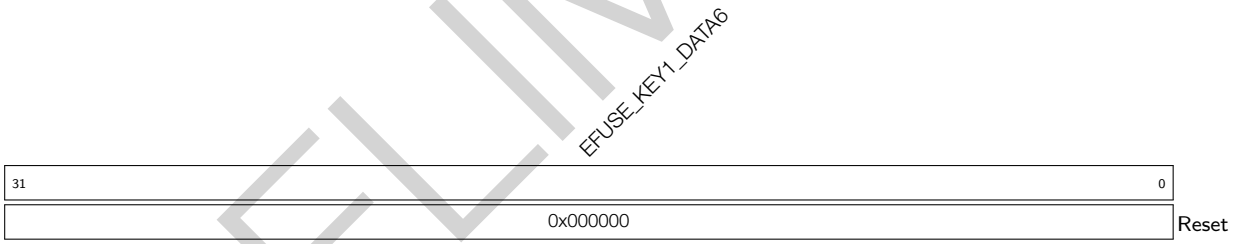
EFUSE_KEY1_DATA4 Stores the fourth 32 bits of KEY1. (RO)

Register 4.53. EFUSE_RD_KEY1_DATA5_REG (0x00D0)



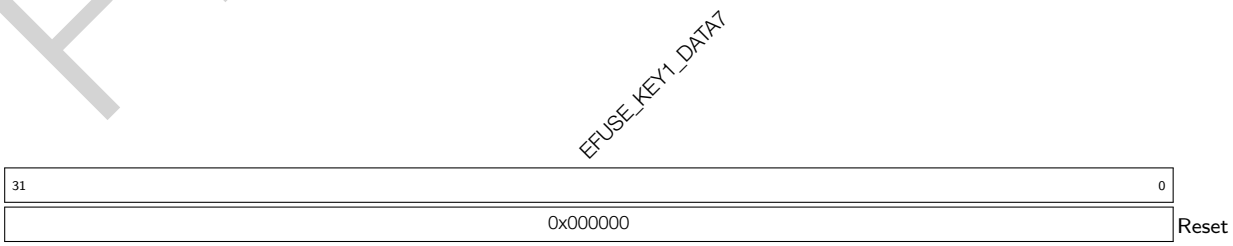
EFUSE_KEY1_DATA5 Stores the fifth 32 bits of KEY1. (RO)

Register 4.54. EFUSE_RD_KEY1_DATA6_REG (0x00D4)



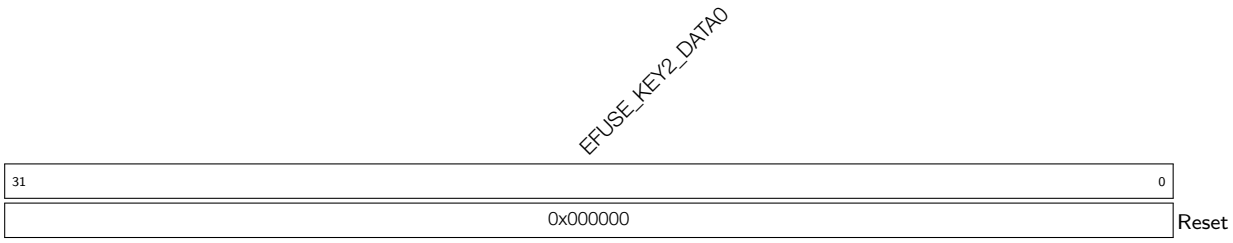
EFUSE_KEY1_DATA6 Stores the sixth 32 bits of KEY1. (RO)

Register 4.55. EFUSE_RD_KEY1_DATA7_REG (0x00D8)



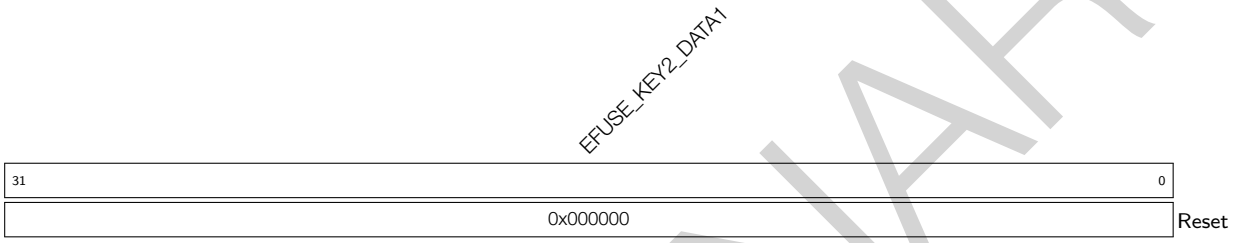
EFUSE_KEY1_DATA7 Stores the seventh 32 bits of KEY1. (RO)

Register 4.56. EFUSE_RD_KEY2_DATA0_REG (0x00DC)



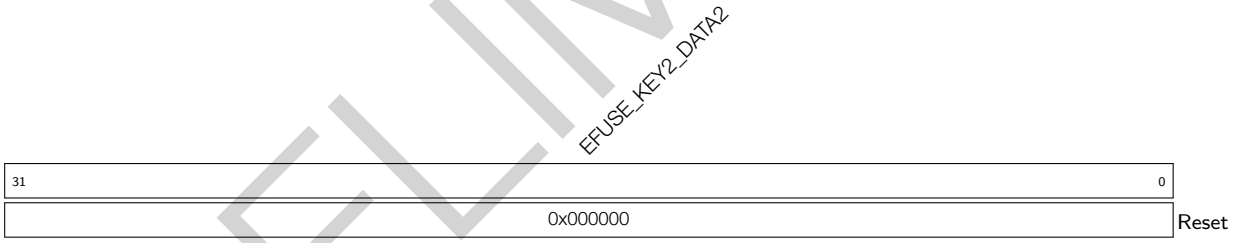
EFUSE_KEY2_DATA0 Stores the zeroth 32 bits of KEY2. (RO)

Register 4.57. EFUSE_RD_KEY2_DATA1_REG (0x00E0)



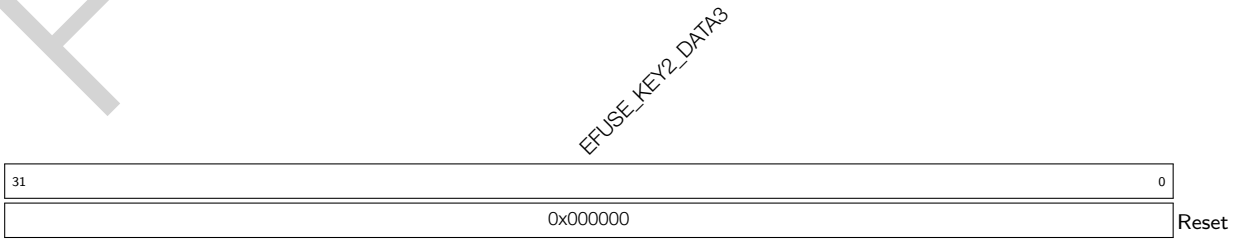
EFUSE_KEY2_DATA1 Stores the first 32 bits of KEY2. (RO)

Register 4.58. EFUSE_RD_KEY2_DATA2_REG (0x00E4)



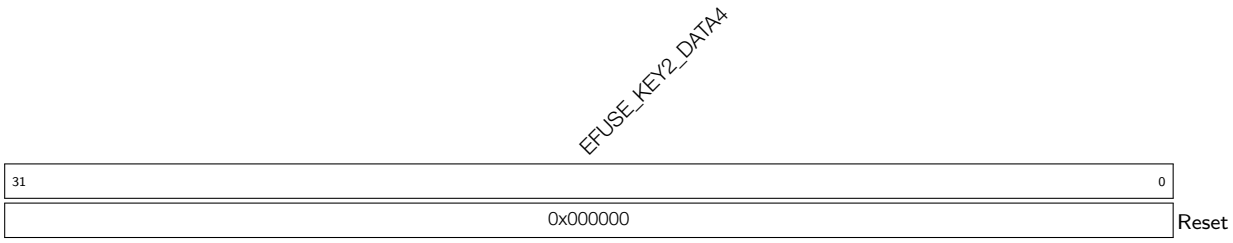
EFUSE_KEY2_DATA2 Stores the second 32 bits of KEY2. (RO)

Register 4.59. EFUSE_RD_KEY2_DATA3_REG (0x00E8)



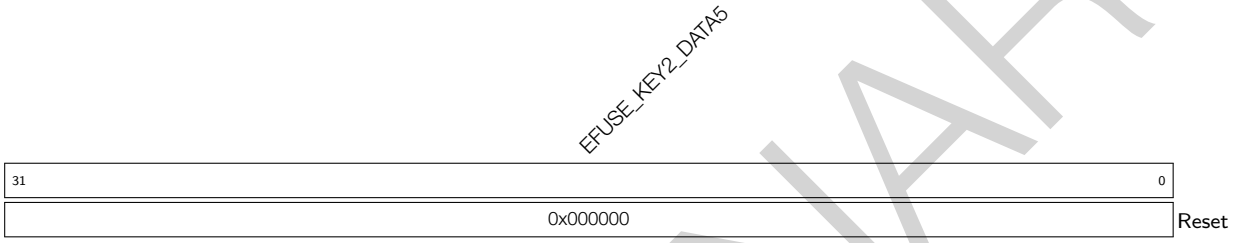
EFUSE_KEY2_DATA3 Stores the third 32 bits of KEY2. (RO)

Register 4.60. EFUSE_RD_KEY2_DATA4_REG (0x00EC)



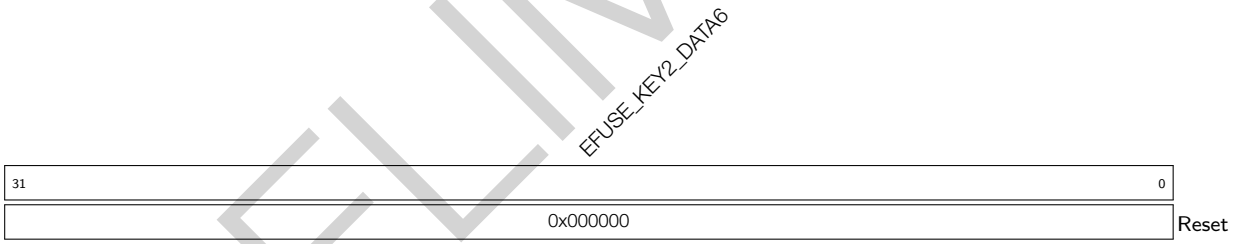
EFUSE_KEY2_DATA4 Stores the fourth 32 bits of KEY2. (RO)

Register 4.61. EFUSE_RD_KEY2_DATA5_REG (0x00F0)



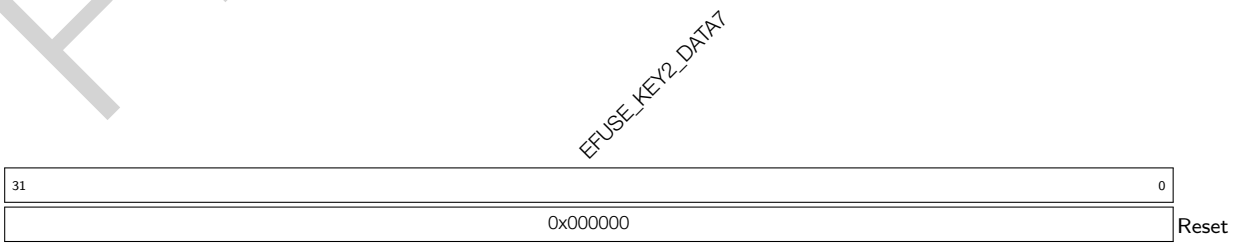
EFUSE_KEY2_DATA5 Stores the fifth 32 bits of KEY2. (RO)

Register 4.62. EFUSE_RD_KEY2_DATA6_REG (0x00F4)



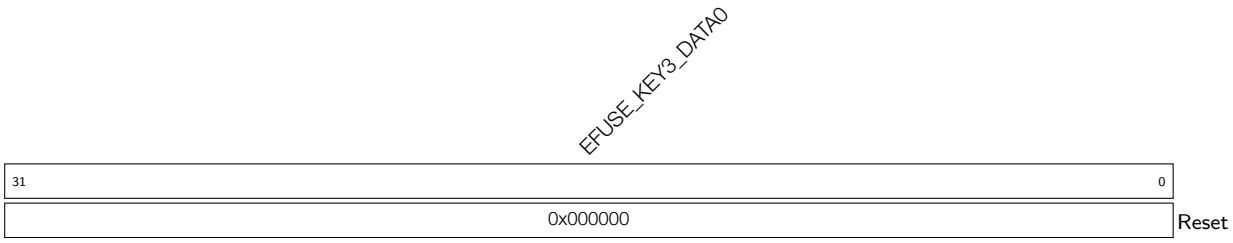
EFUSE_KEY2_DATA6 Stores the sixth 32 bits of KEY2. (RO)

Register 4.63. EFUSE_RD_KEY2_DATA7_REG (0x00F8)



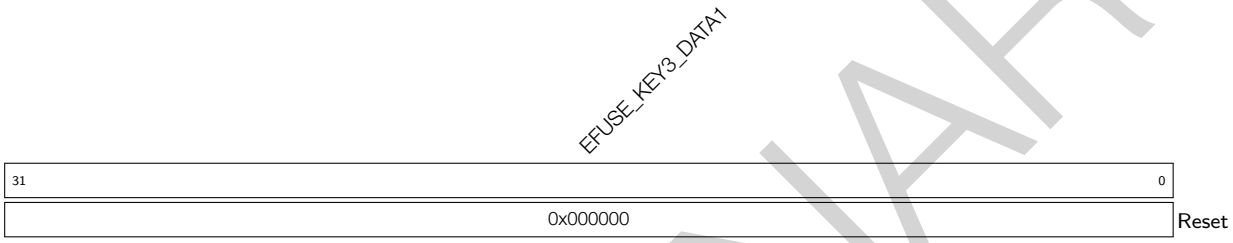
EFUSE_KEY2_DATA7 Stores the seventh 32 bits of KEY2. (RO)

Register 4.64. EFUSE_RD_KEY3_DATA0_REG (0x00FC)



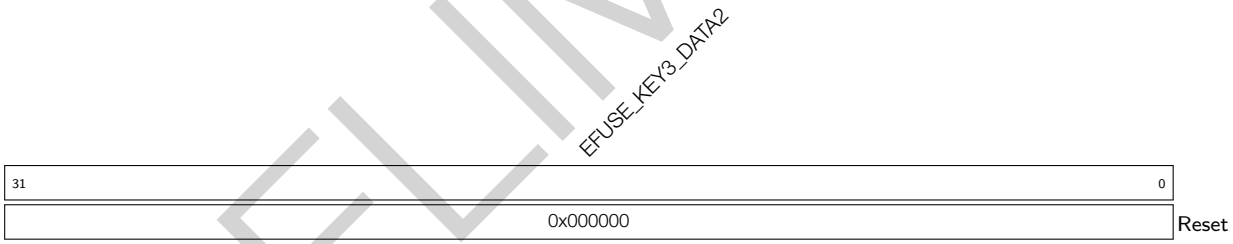
EFUSE_KEY3_DATA0 Stores the zeroth 32 bits of KEY3. (RO)

Register 4.65. EFUSE_RD_KEY3_DATA1_REG (0x0100)



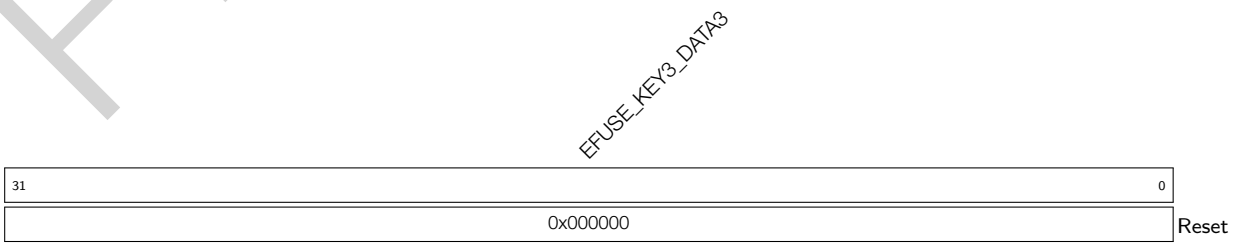
EFUSE_KEY3_DATA1 Stores the first 32 bits of KEY3. (RO)

Register 4.66. EFUSE_RD_KEY3_DATA2_REG (0x0104)



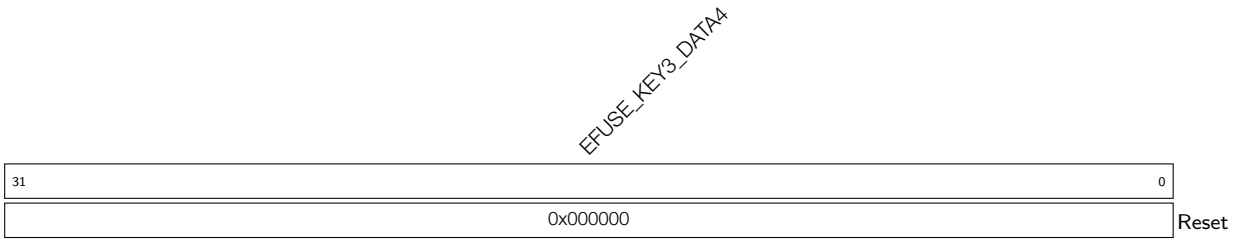
EFUSE_KEY3_DATA2 Stores the second 32 bits of KEY3. (RO)

Register 4.67. EFUSE_RD_KEY3_DATA3_REG (0x0108)



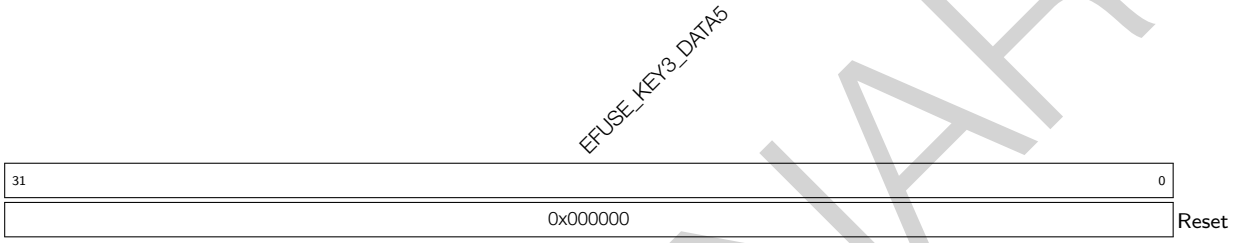
EFUSE_KEY3_DATA3 Stores the third 32 bits of KEY3. (RO)

Register 4.68. EFUSE_RD_KEY3_DATA4_REG (0x010C)



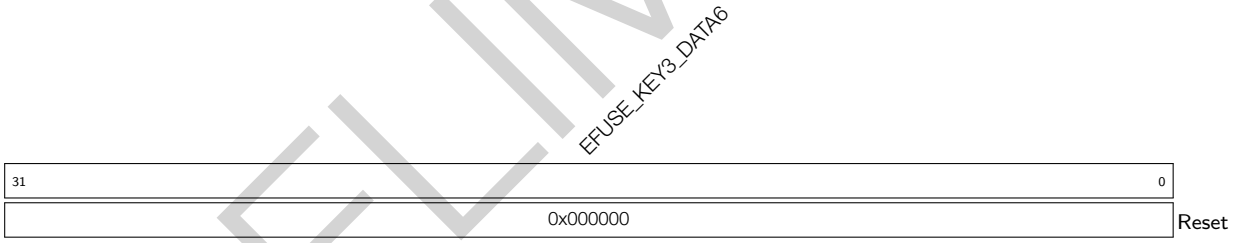
EFUSE_KEY3_DATA4 Stores the fourth 32 bits of KEY3. (RO)

Register 4.69. EFUSE_RD_KEY3_DATA5_REG (0x0110)



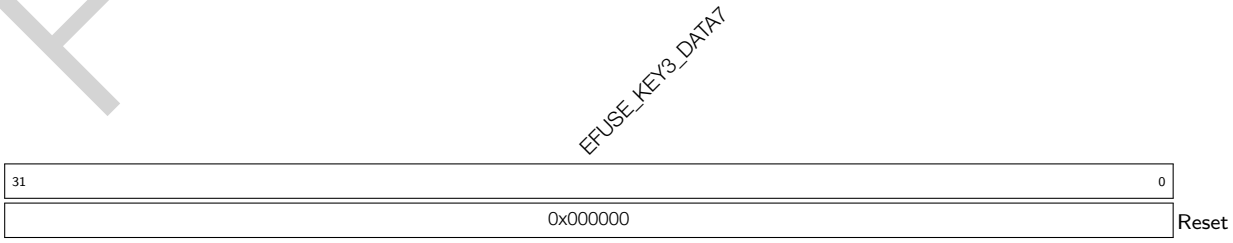
EFUSE_KEY3_DATA5 Stores the fifth 32 bits of KEY3. (RO)

Register 4.70. EFUSE_RD_KEY3_DATA6_REG (0x0114)



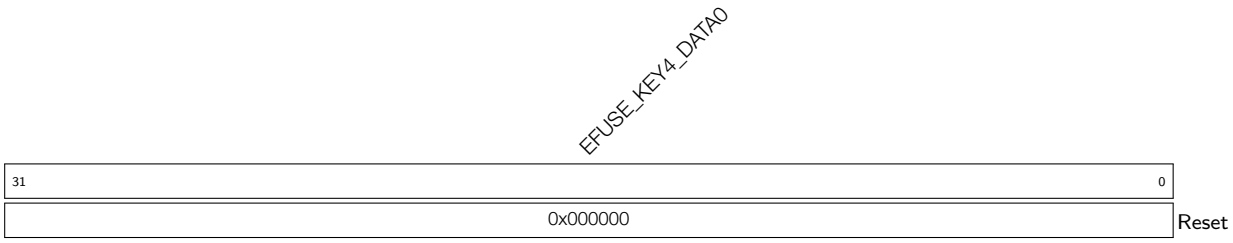
EFUSE_KEY3_DATA6 Stores the sixth 32 bits of KEY3. (RO)

Register 4.71. EFUSE_RD_KEY3_DATA7_REG (0x0118)



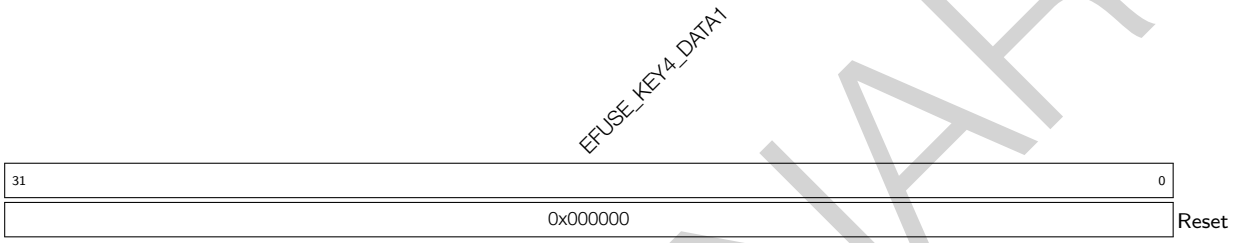
EFUSE_KEY3_DATA7 Stores the seventh 32 bits of KEY3. (RO)

Register 4.72. EFUSE_RD_KEY4_DATA0_REG (0x011C)



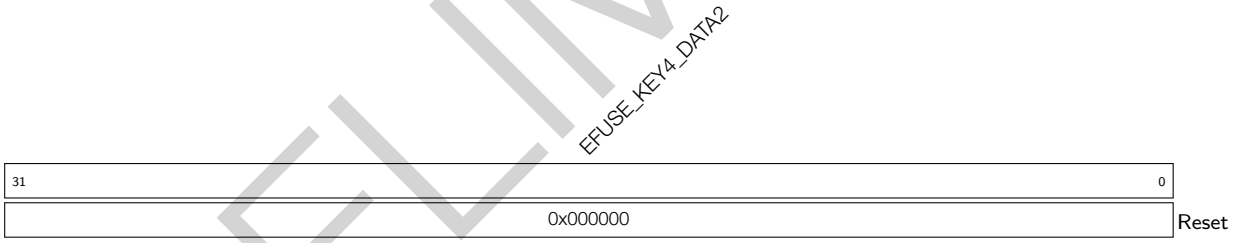
EFUSE_KEY4_DATA0 Stores the zeroth 32 bits of KEY4. (RO)

Register 4.73. EFUSE_RD_KEY4_DATA1_REG (0x0120)



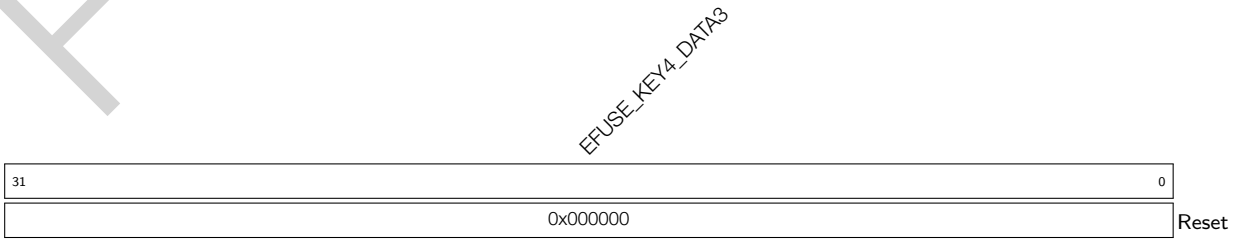
EFUSE_KEY4_DATA1 Stores the first 32 bits of KEY4. (RO)

Register 4.74. EFUSE_RD_KEY4_DATA2_REG (0x0124)



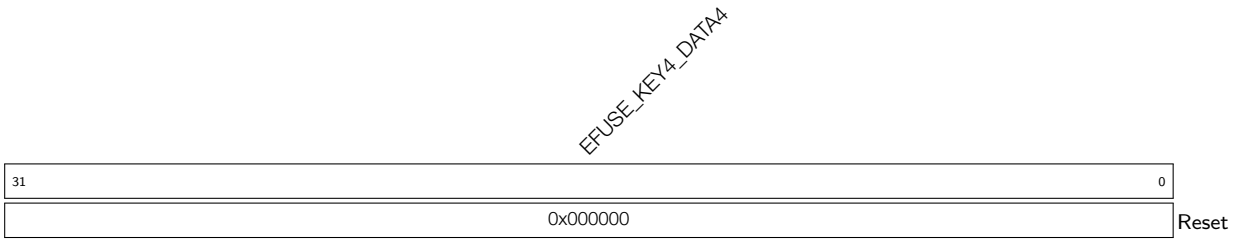
EFUSE_KEY4_DATA2 Stores the second 32 bits of KEY4. (RO)

Register 4.75. EFUSE_RD_KEY4_DATA3_REG (0x0128)



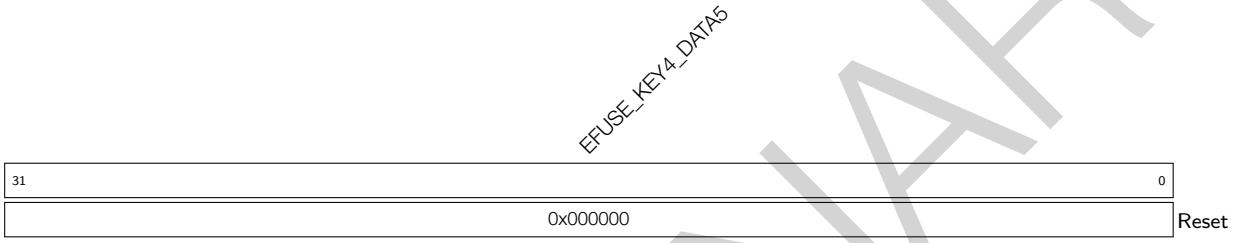
EFUSE_KEY4_DATA3 Stores the third 32 bits of KEY4. (RO)

Register 4.76. EFUSE_RD_KEY4_DATA4_REG (0x012C)



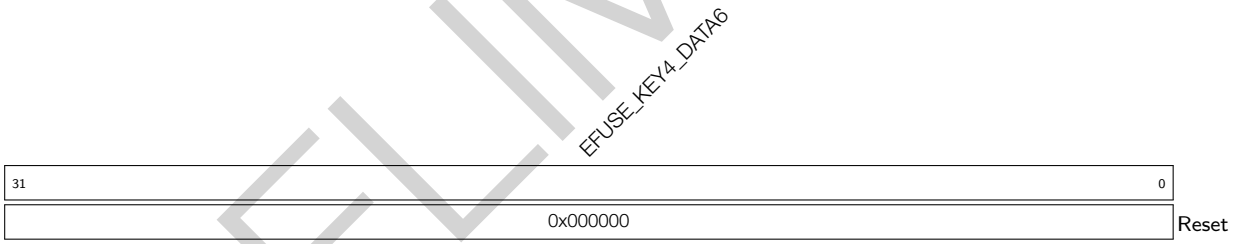
EFUSE_KEY4_DATA4 Stores the fourth 32 bits of KEY4. (RO)

Register 4.77. EFUSE_RD_KEY4_DATA5_REG (0x0130)



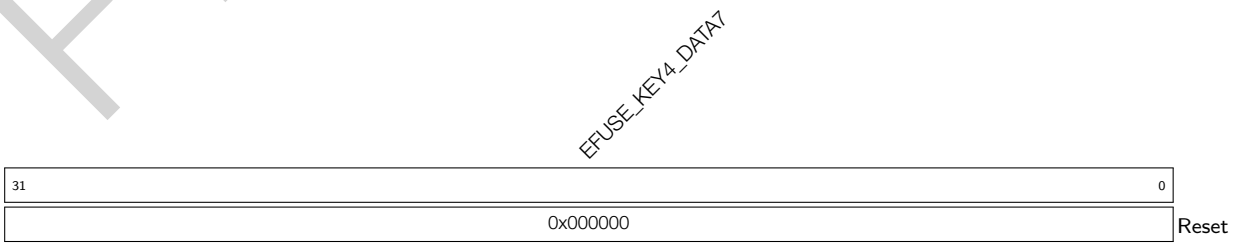
EFUSE_KEY4_DATA5 Stores the fifth 32 bits of KEY4. (RO)

Register 4.78. EFUSE_RD_KEY4_DATA6_REG (0x0134)



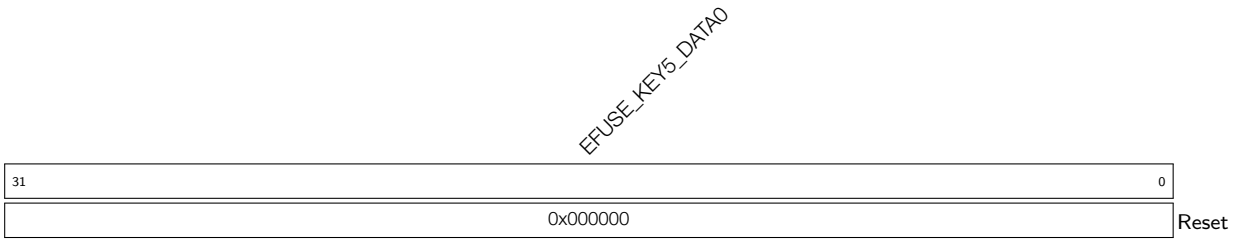
EFUSE_KEY4_DATA6 Stores the sixth 32 bits of KEY4. (RO)

Register 4.79. EFUSE_RD_KEY4_DATA7_REG (0x0138)



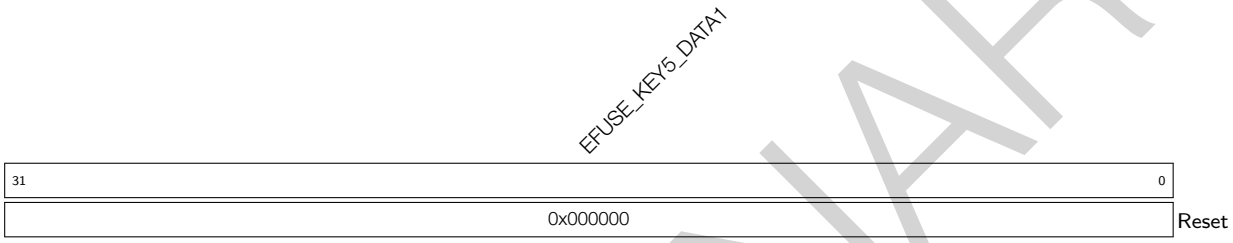
EFUSE_KEY4_DATA7 Stores the seventh 32 bits of KEY4. (RO)

Register 4.80. EFUSE_RD_KEY5_DATA0_REG (0x013C)



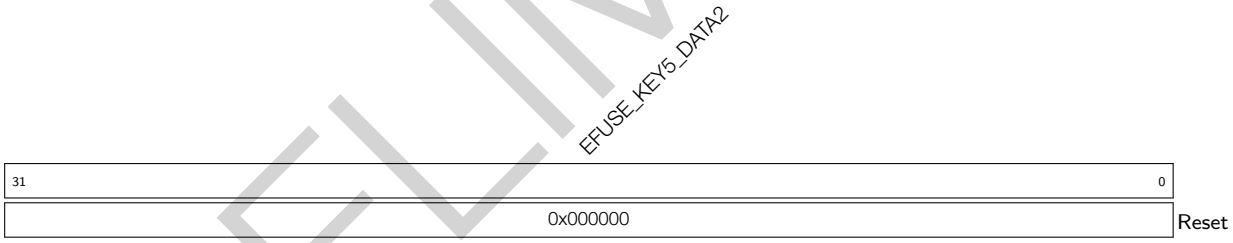
EFUSE_KEY5_DATA0 Stores the zeroth 32 bits of KEY5. (RO)

Register 4.81. EFUSE_RD_KEY5_DATA1_REG (0x0140)



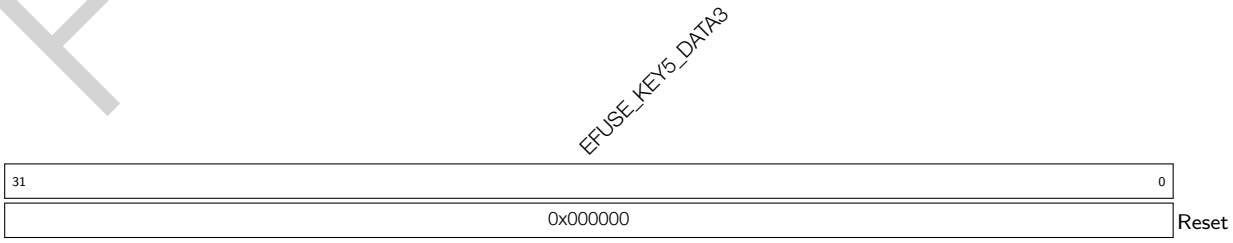
EFUSE_KEY5_DATA1 Stores the first 32 bits of KEY5. (RO)

Register 4.82. EFUSE_RD_KEY5_DATA2_REG (0x0144)



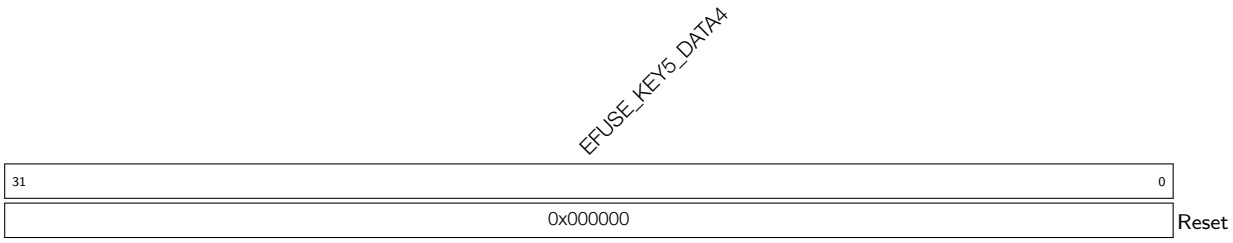
EFUSE_KEY5_DATA2 Stores the second 32 bits of KEY5. (RO)

Register 4.83. EFUSE_RD_KEY5_DATA3_REG (0x0148)



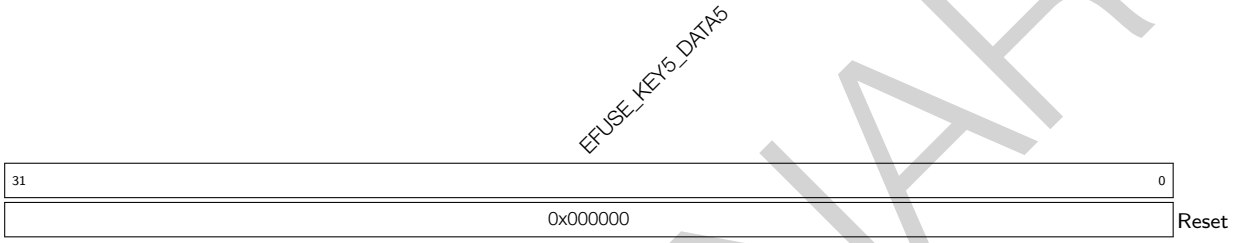
EFUSE_KEY5_DATA3 Stores the third 32 bits of KEY5. (RO)

Register 4.84. EFUSE_RD_KEY5_DATA4_REG (0x014C)



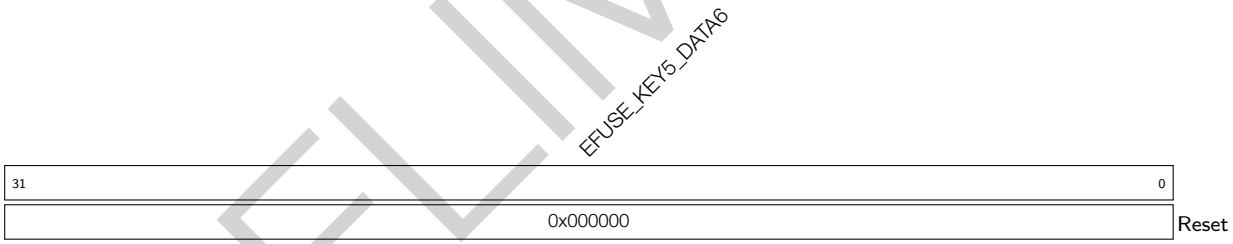
EFUSE_KEY5_DATA4 Stores the fourth 32 bits of KEY5. (RO)

Register 4.85. EFUSE_RD_KEY5_DATA5_REG (0x0150)



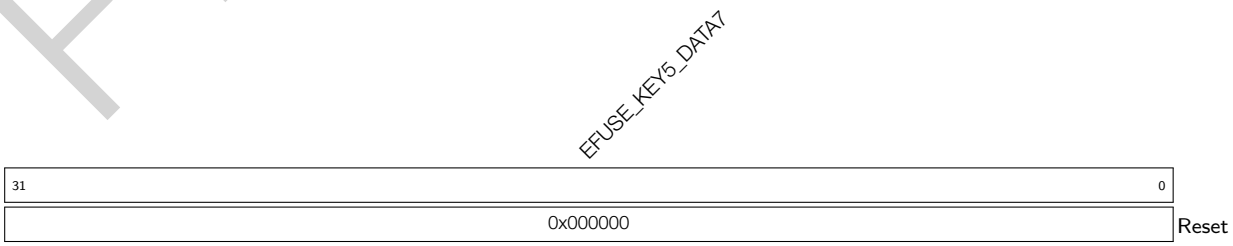
EFUSE_KEY5_DATA5 Stores the fifth 32 bits of KEY5. (RO)

Register 4.86. EFUSE_RD_KEY5_DATA6_REG (0x0154)



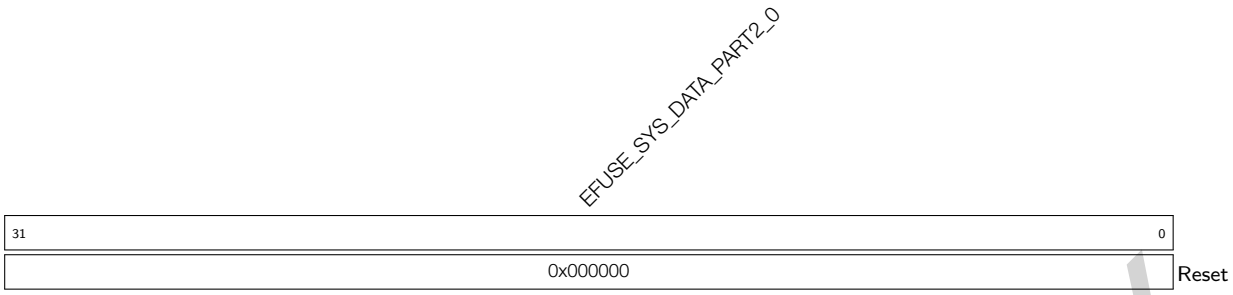
EFUSE_KEY5_DATA6 Stores the sixth 32 bits of KEY5. (RO)

Register 4.87. EFUSE_RD_KEY5_DATA7_REG (0x0158)



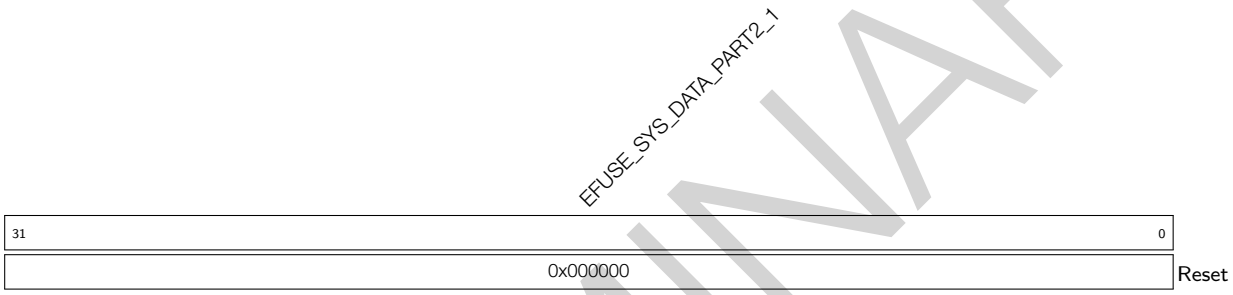
EFUSE_KEY5_DATA7 Stores the seventh 32 bits of KEY5. (RO)

Register 4.88. EFUSE_RD_SYS_PART2_DATA0_REG (0x015C)



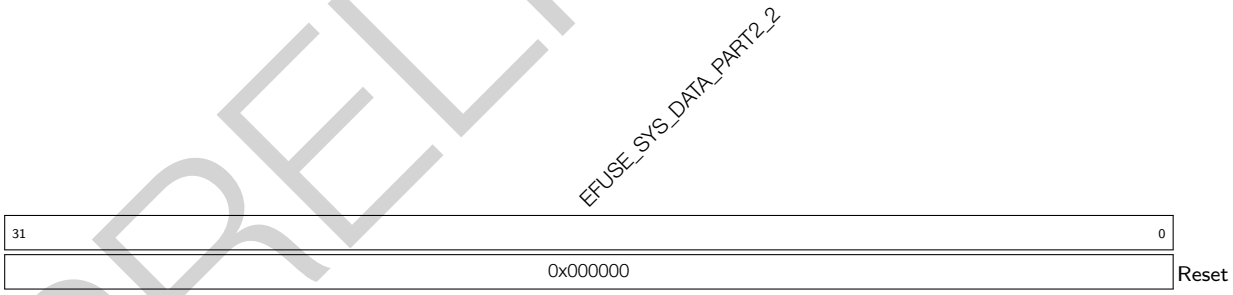
EFUSE_SYS_DATA_PART2_0 Stores the 0th 32 bits of the 2nd part of system data. (RO)

Register 4.89. EFUSE_RD_SYS_PART2_DATA1_REG (0x0160)



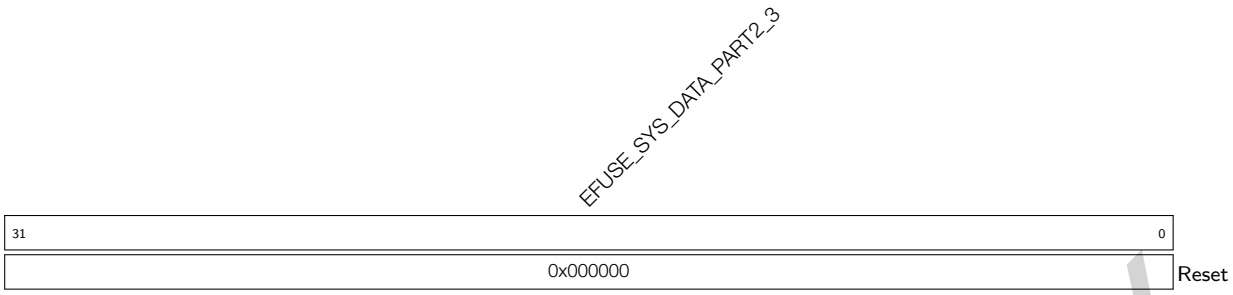
EFUSE_SYS_DATA_PART2_1 Stores the 1st 32 bits of the 2nd part of system data. (RO)

Register 4.90. EFUSE_RD_SYS_PART2_DATA2_REG (0x0164)



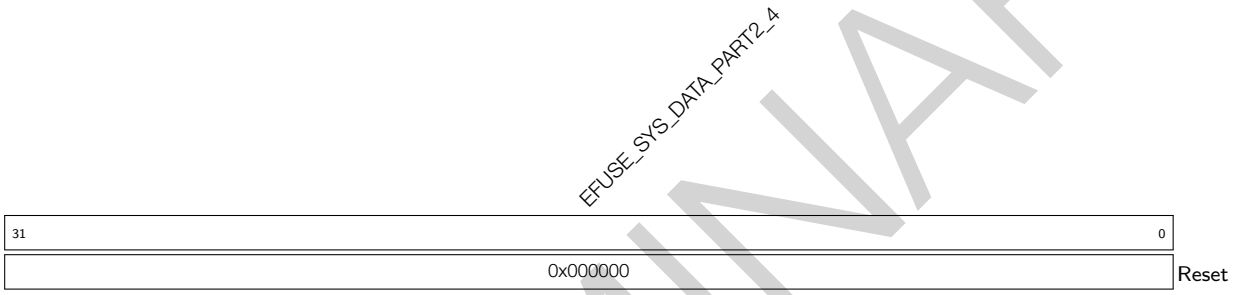
EFUSE_SYS_DATA_PART2_2 Stores the 2nd 32 bits of the 2nd part of system data. (RO)

Register 4.91. EFUSE_RD_SYS_PART2_DATA3_REG (0x0168)



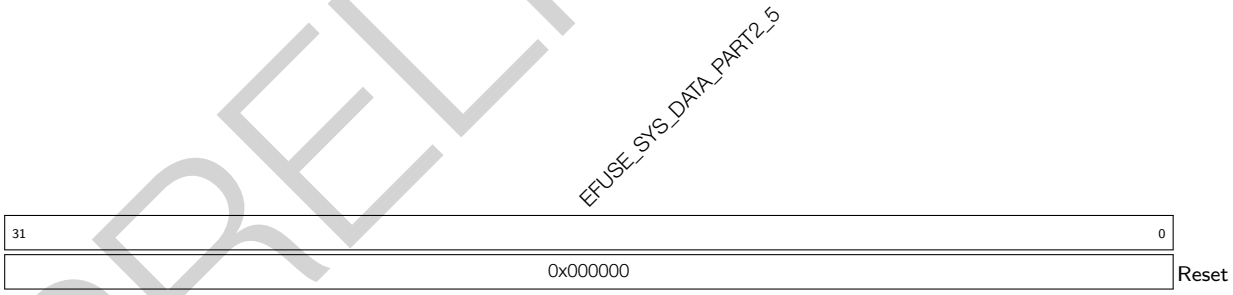
EFUSE_SYS_DATA_PART2_3 Stores the 3rd 32 bits of the 2nd part of system data. (RO)

Register 4.92. EFUSE_RD_SYS_PART2_DATA4_REG (0x016C)



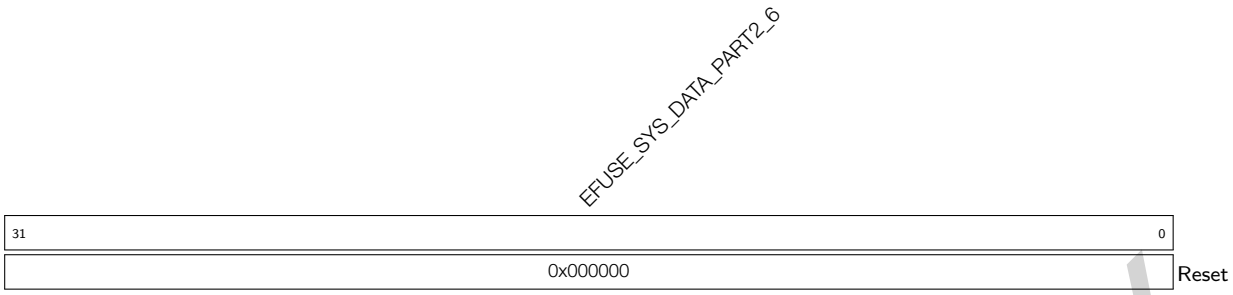
EFUSE_SYS_DATA_PART2_4 Stores the 4th 32 bits of the 2nd part of system data. (RO)

Register 4.93. EFUSE_RD_SYS_PART2_DATA5_REG (0x0170)



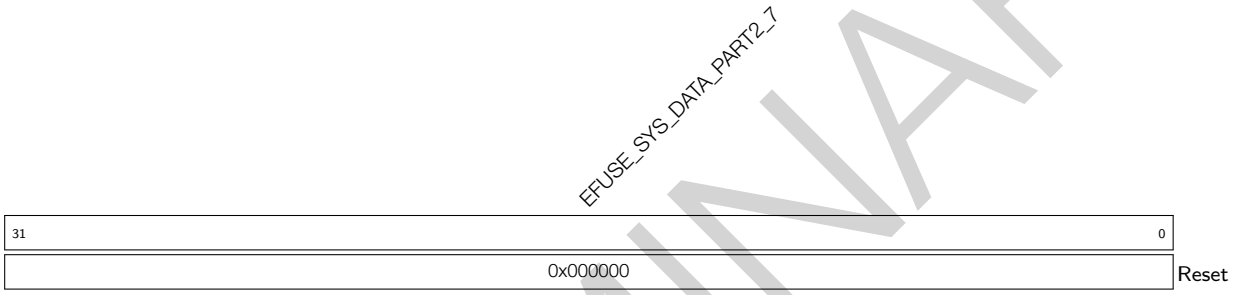
EFUSE_SYS_DATA_PART2_5 Stores the 5th 32 bits of the 2nd part of system data. (RO)

Register 4.94. EFUSE_RD_SYS_PART2_DATA6_REG (0x0174)



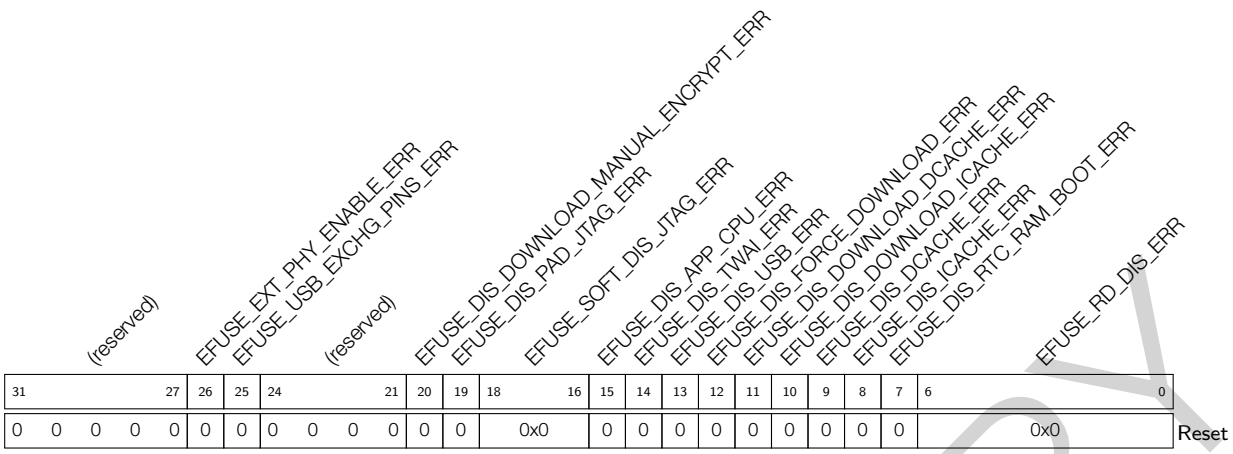
EFUSE_SYS_DATA_PART2_6 Stores the 6th 32 bits of the 2nd part of system data. (RO)

Register 4.95. EFUSE_RD_SYS_PART2_DATA7_REG (0x0178)



EFUSE_SYS_DATA_PART2_7 Stores the 7th 32 bits of the 2nd part of system data. (RO)

Register 4.96. EFUSE_RD_REPEAT_ERR0_REG (0x017C)



EFUSE_RD_DIS_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_RTC_RAM_BOOT_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_ICACHE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_DCACHE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_DOWNLOAD_ICACHE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_DOWNLOAD_DCACHE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_FORCE_DOWNLOAD_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_USB_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_TWAI_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_APP_CPU_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_SOFT_DIS_JTAG_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_PAD_JTAG_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

Continued on the next page...

Register 4.96. EFUSE_RD_REPEAT_ERR0_REG (0x017C)

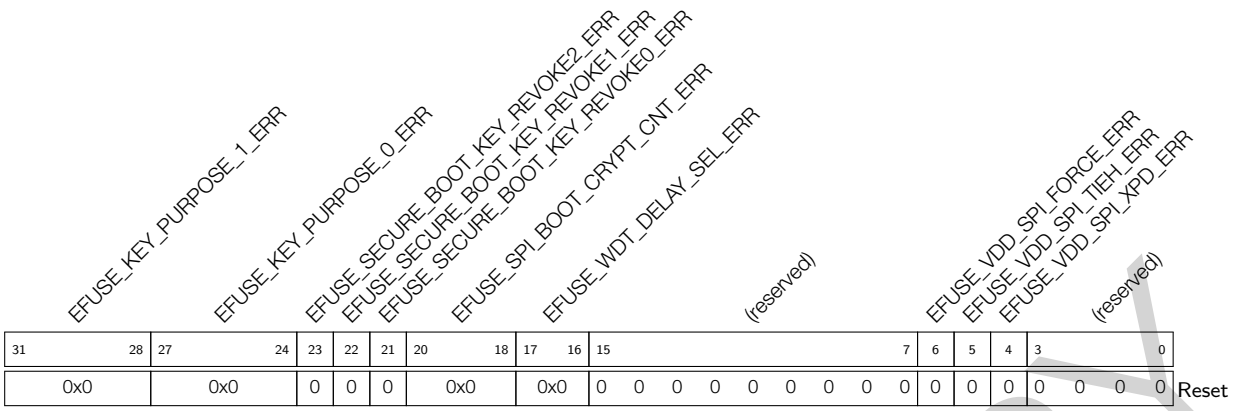
Continued from the previous page...

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_USB_EXCHG_PINS_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_EXT_PHY_ENABLE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

Register 4.97. EFUSE_RD_REPEAT_ERR1_REG (0x0180)



EFUSE_VDD_SPI_XPD_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_VDD_SPI_TIEH_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_VDD_SPI_FORCE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_WDT_DELAY_SEL_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_SPI_BOOT_CRYPT_CNT_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

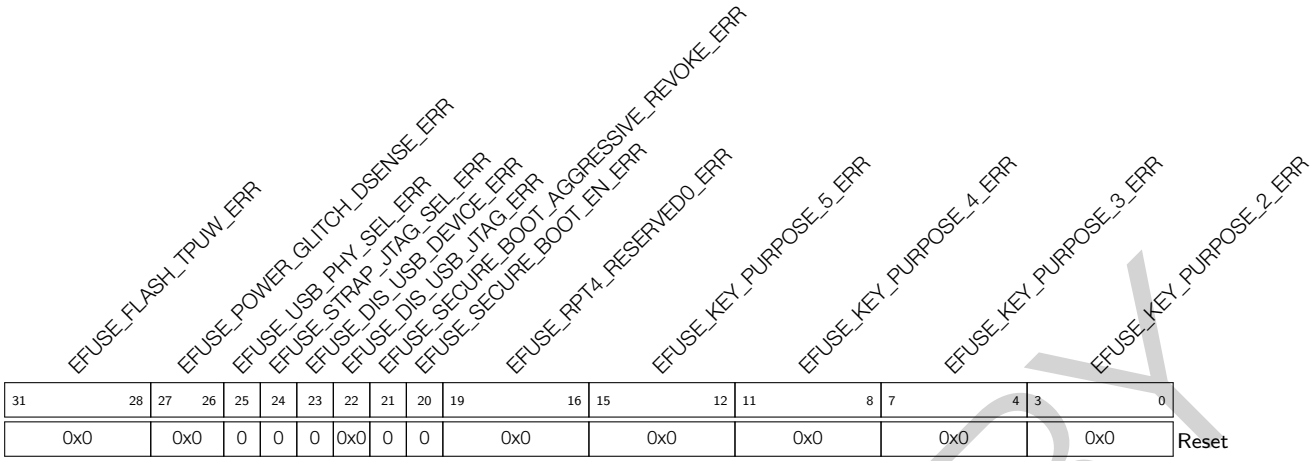
EFUSE_SECURE_BOOT_KEY_REVOKE1_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_KEY_PURPOSE_0_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_KEY_PURPOSE_1_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

Register 4.98. EFUSE_RD_REPEAT_ERR2_REG (0x0184)



EFUSE_KEY_PURPOSE_2_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_KEY_PURPOSE_3_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_KEY_PURPOSE_4_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_KEY_PURPOSE_5_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_RPT4_RESERVED0_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_SECURE_BOOT_EN_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_USB_JTAG_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_USB_DEVICE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

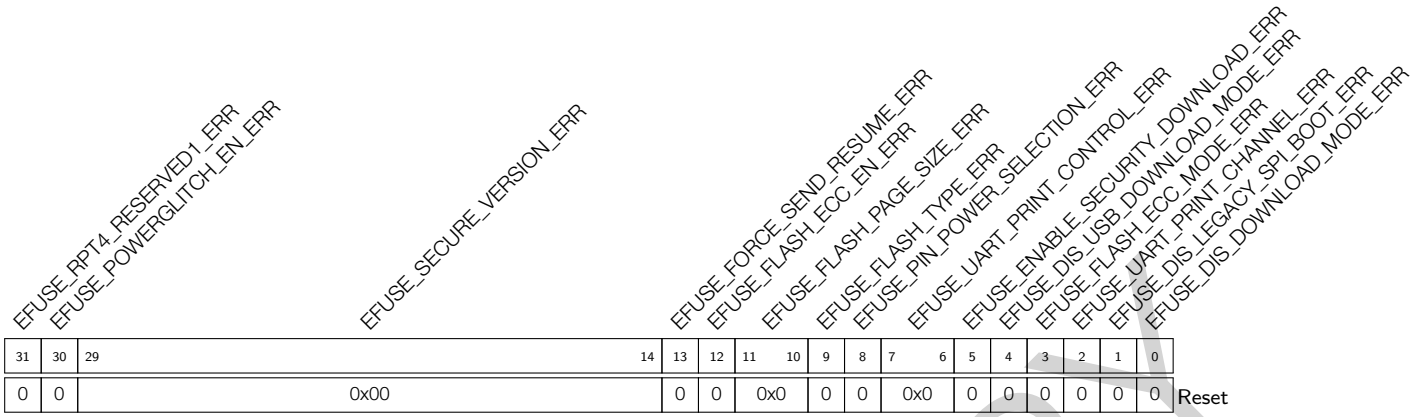
EFUSE_STRAP_JTAG_SEL_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_USB_PHY_SEL_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_POWER_GLITCH_DSENSE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_FLASH_TPUW_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

Register 4.99. EFUSE_RD_REPEAT_ERR3_REG (0x0188)



EFUSE_DIS_DOWNLOAD_MODE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_LEGACY_SPI_BOOT_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_UART_PRINT_CHANNEL_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_FLASH_ECC_MODE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_DIS_USB_DOWNLOAD_MODE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_UART_PRINT_CONTROL_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_PIN_POWER_SELECTION_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_FLASH_TYPE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_FLASH_PAGE_SIZE_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_FLASH_ECC_EN_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

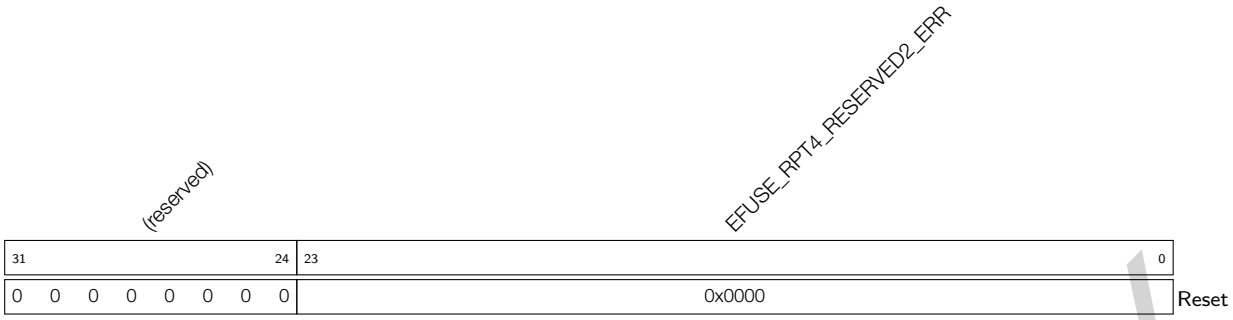
EFUSE_FORCE_SEND_RESUME_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_SECURE_VERSION_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

EFUSE_POWERGLITCH_EN_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

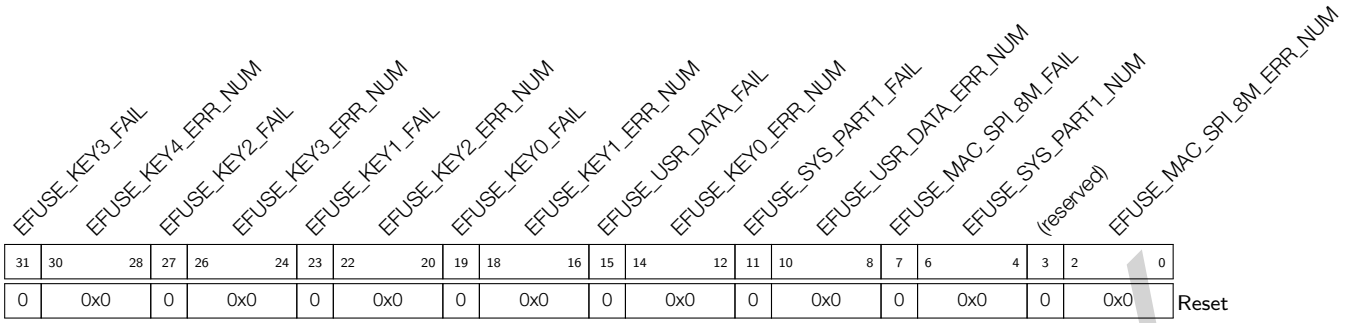
EFUSE_RPT4_RESERVED1_ERR Reserved. (RO)

Register 4.100. EFUSE_RD_REPEAT_ERR4_REG (0x0190)



EFUSE_RPT4_RESERVED2_ERR If any bits in this filed are 1, then it indicates a programming error. (RO)

Register 4.101. EFUSE_RD_RS_ERR0_REG (0x01C0)



EFUSE_MAC_SPI_8M_ERR_NUM The value of this signal means the number of error bytes. (RO)

EFUSE_SYS_PART1_NUM The value of this signal means the number of error bytes. (RO)

EFUSE_MAC_SPI_8M_FAIL 0: Means no failure and that the data of MAC_SPI_8M is reliable 1: Means that programming data of MAC_SPI_8M failed and the number of error bytes is over 6. (RO)

EFUSE_USR_DATA_ERR_NUM The value of this signal means the number of error bytes. (RO)

EFUSE_SYS_PART1_FAIL 0: Means no failure and that the data of system part1 is reliable 1: Means that programming data of system part1 failed and the number of error bytes is over 6. (RO)

EFUSE_KEY0_ERR_NUM The value of this signal means the number of error bytes. (RO)

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

EFUSE_KEY1_ERR_NUM The value of this signal means the number of error bytes. (RO)

EFUSE_KEY0_FAIL 0: Means no failure and that the data of key0 is reliable 1: Means that programming key0 failed and the number of error bytes is over 6. (RO)

EFUSE_KEY2_ERR_NUM The value of this signal means the number of error bytes. (RO)

EFUSE_KEY1_FAIL 0: Means no failure and that the data of key1 is reliable 1: Means that programming key1 failed and the number of error bytes is over 6. (RO)

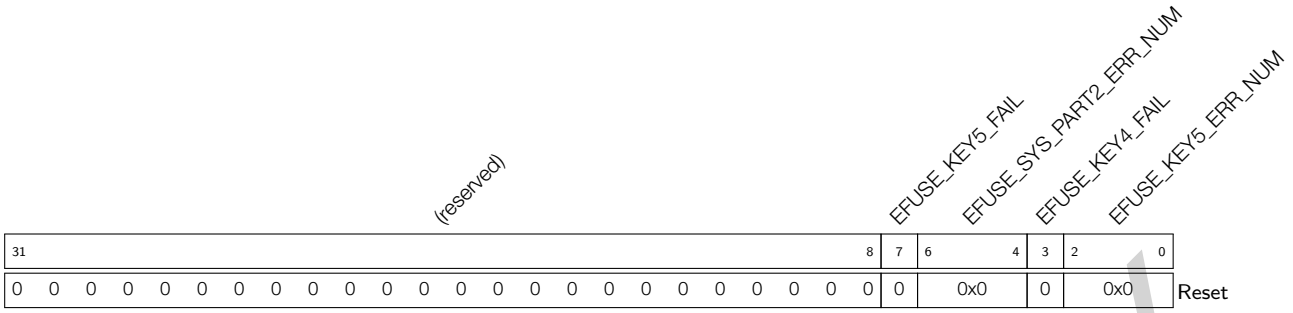
EFUSE_KEY3_ERR_NUM The value of this signal means the number of error bytes. (RO)

EFUSE_KEY2_FAIL 0: Means no failure and that the data of key2 is reliable 1: Means that programming key2 failed and the number of error bytes is over 6. (RO)

EFUSE_KEY4_ERR_NUM The value of this signal means the number of error bytes. (RO)

EFUSE_KEY3_FAIL 0: Means no failure and that the data of key3 is reliable 1: Means that programming key3 failed and the number of error bytes is over 6. (RO)

Register 4.102. EFUSE_RD_RS_ERR1_REG (0x01C4)



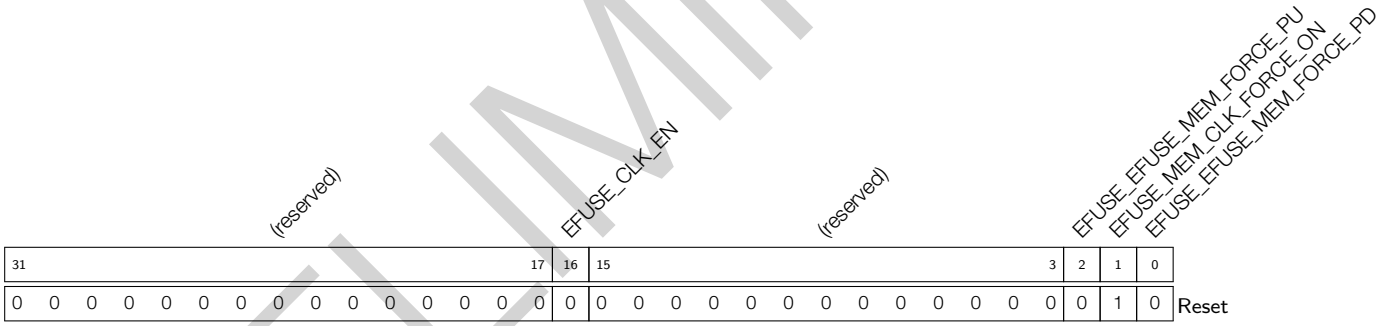
EFUSE_KEY5_ERR_NUM The value of this signal means the number of error bytes. (RO)

EFUSE_KEY4_FAIL 0: Means no failure and that the data of KEY4 is reliable 1: Means that programming KEY4 data failed and the number of error bytes is over 6. (RO)

EFUSE_SYS_PART2_ERR_NUM The value of this signal means the number of error bytes. (RO)

EFUSE_KEY5_FAIL 0: Means no failure and that the data of KEY5 is reliable 1: Means that programming KEY5 data failed and the number of error bytes is over 6. (RO)

Register 4.103. EFUSE_CLK_REG (0x01C8)



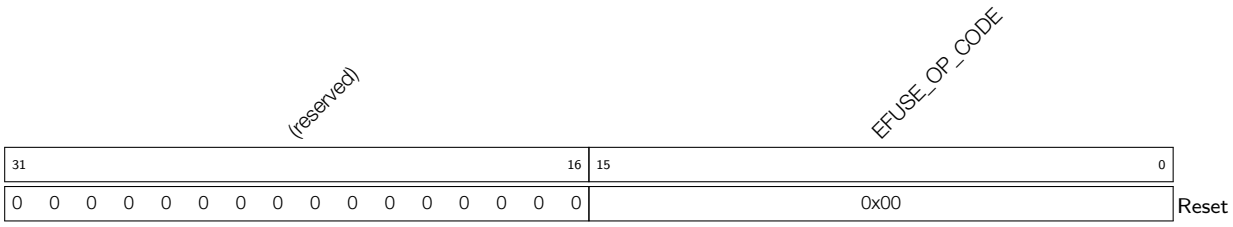
EFUSE_EFUSE_MEM_FORCE_PD Set this bit to force eFuse SRAM into power-saving mode. (R/W)

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

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

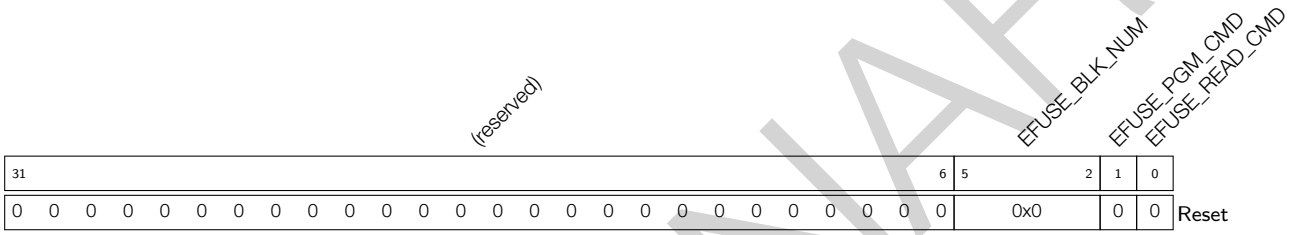
EFUSE_CLK_EN Set this bit and force to enable clock signal of eFuse memory. (R/W)

Register 4.104. EFUSE_CONF_REG (0x01CC)



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

Register 4.105. EFUSE_CMD_REG (0x01D4)

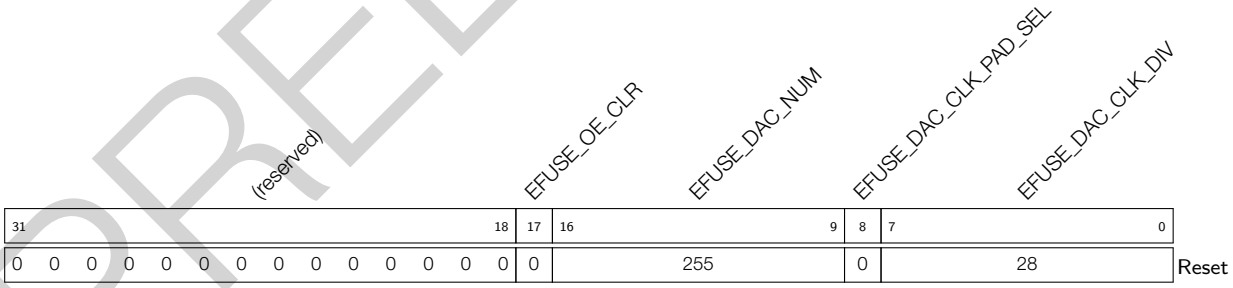


EFUSE_READ_CMD Set this bit to send read command. (R/WS/SC)

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

EFUSE_BLK_NUM The serial number of the block to be programmed. Value 0 ~ 10 corresponds to block number 0 ~ 10, respectively. (R/W)

Register 4.106. EFUSE_DAC_CONF_REG (0x01E8)



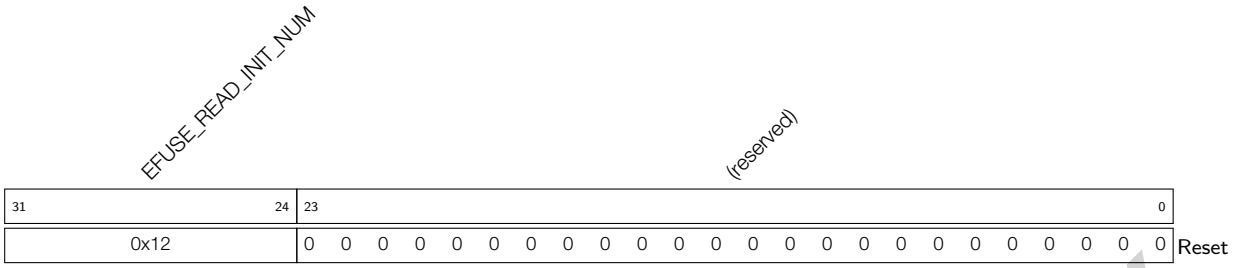
EFUSE_DAC_CLK_DIV Controls the division factor of the rising clock of the programming voltage. (R/W)

EFUSE_DAC_CLK_PAD_SEL Don't care. (R/W)

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

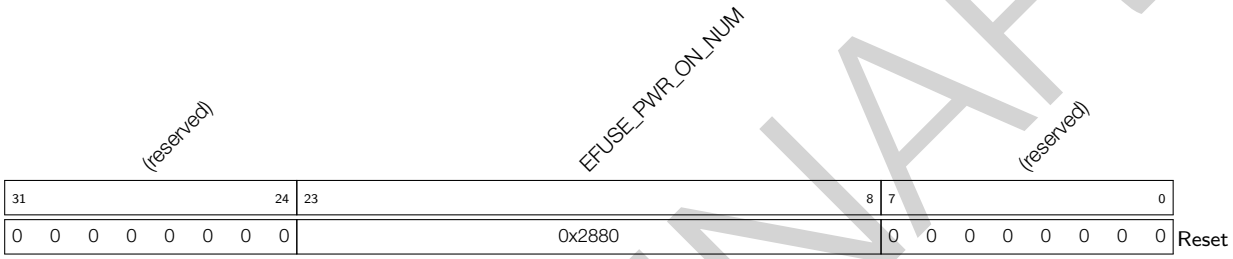
EFUSE_OE_CLR Reduces the power supply of the programming voltage. (R/W)

Register 4.107. EFUSE_RD_TIM_CONF_REG (0x01EC)



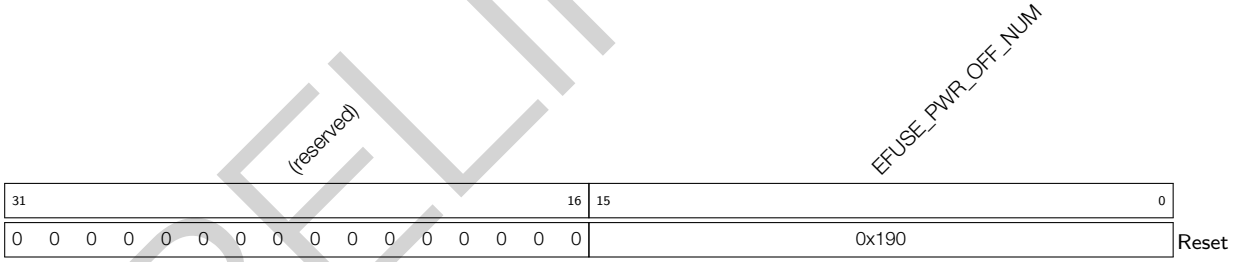
EFUSE_READ_INIT_NUM Configures the initial read time of eFuse. (R/W)

Register 4.108. EFUSE_WR_TIM_CONF1_REG (0x01F4)



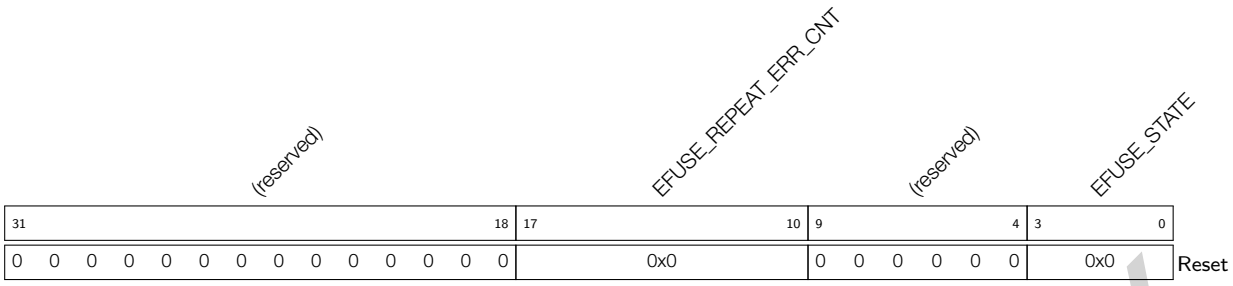
EFUSE_PWR_ON_NUM Configures the power up time for VDDQ. (R/W)

Register 4.109. EFUSE_WR_TIM_CONF2_REG (0x01F8)



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

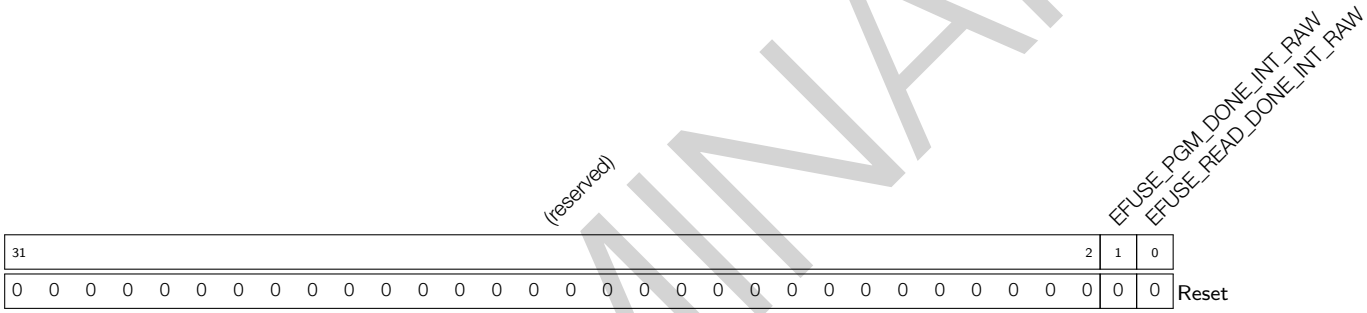
Register 4.110. EFUSE_STATUS_REG (0x01D0)



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

EFUSE_REPEAT_ERR_CNT Indicates the number of error bits during programming BLOCK0. (RO)

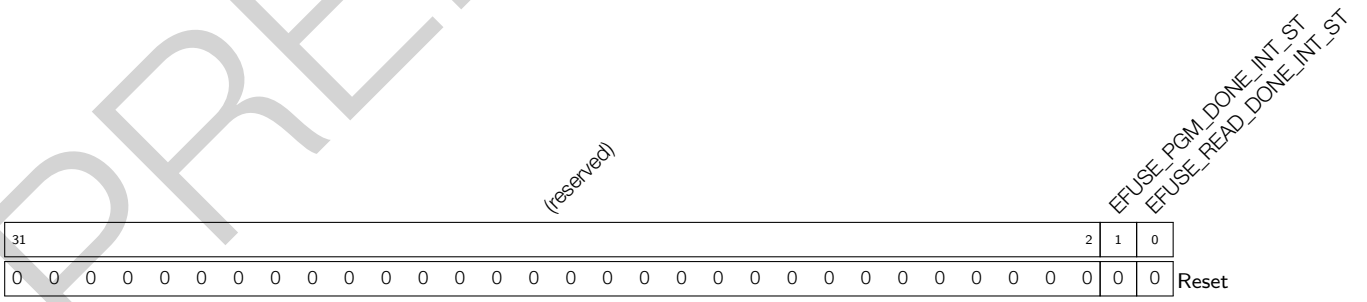
Register 4.111. EFUSE_INT_RAW_REG (0x01D8)



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

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

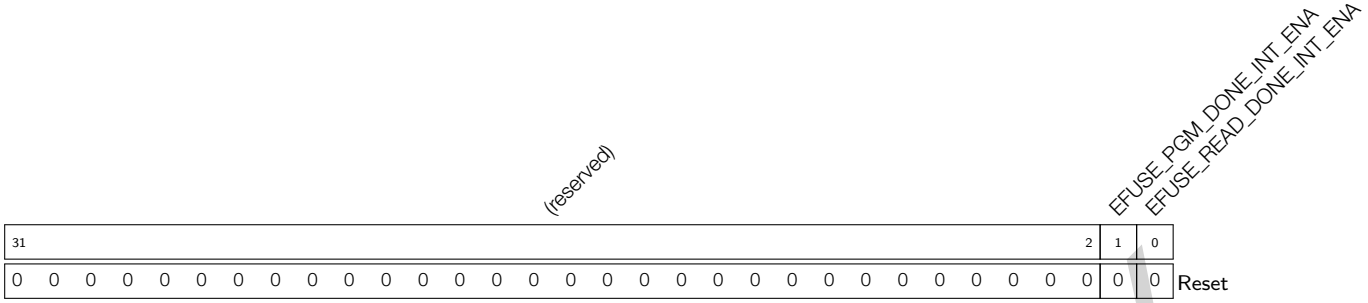
Register 4.112. EFUSE_INT_ST_REG (0x01DC)



EFUSE_READ_DONE_INT_ST The status signal for read_done interrupt. (RO)

EFUSE_PGM_DONE_INT_ST The status signal for pgm_done interrupt. (RO)

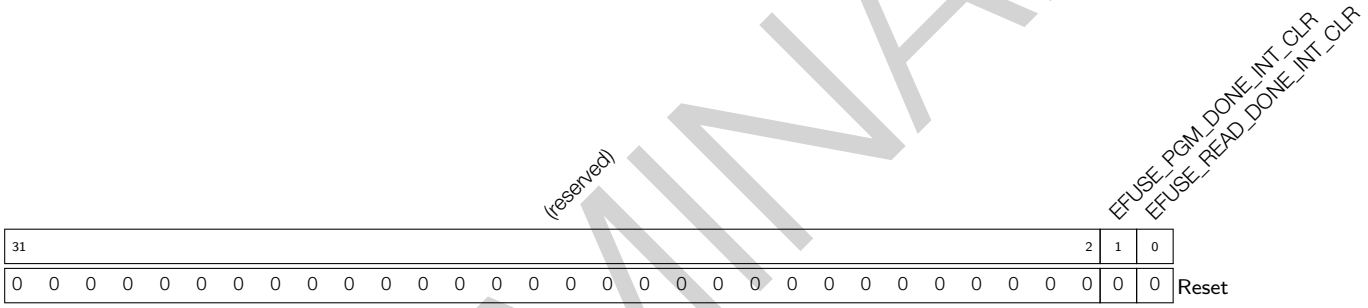
Register 4.113. EFUSE_INT_ENA_REG (0x01E0)



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

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

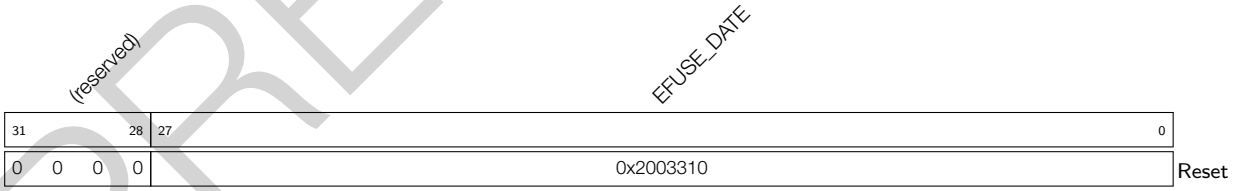
Register 4.114. EFUSE_INT_CLR_REG (0x01E4)



EFUSE_READ_DONE_INT_CLR The clear signal for read_done interrupt. (WO)

EFUSE_PGM_DONE_INT_CLR The clear signal for pgm_done interrupt. (WO)

Register 4.115. EFUSE_DATE_REG (0x01FC)



EFUSE_DATE Stores eFuse version. (R/W)

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

5.1 Overview

The ESP32-C3 chip features 22 physical GPIO pins. Each pin can be used as a general-purpose I/O, or be connected to an internal peripheral signal. Through GPIO matrix 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 that the GPIO pins are numbered from 0 ~ 21.

5.2 Features

GPIO Matrix Features

- A full-switching matrix between the peripheral input/output signals and the pins. Control signals: DRV, IE, OE, WPU, WPD.
- 49 peripheral input signals can be sourced from the input of any GPIO pins. Control signals: SIG_IN_SEL, IE, etc.
- The output of any GPIO pins can be from any of the 125 peripheral output signals. Control signals: SIG_OUT_SEL, OE, etc.
- Supports signal synchronization for peripheral inputs based on APB clock bus.
- Provides input signal filter.
- Supports sigma delta modulated output.
- Supports GPIO simple input and output.

IO MUX Features

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

5.3 Architectural Overview

This section provides an overview to the architecture of IO MUX and GPIO matrix with the following figures:

- Figure 5-1 shows the general work flow of IO MUX and GPIO matix.
- Figure 5-2 shows in details how IO MUX and GPIO matrix route signals from pins to peripherals, and from peripherals to pins.
- Figure 5-3 shows the interface logic for a GPIO pin.

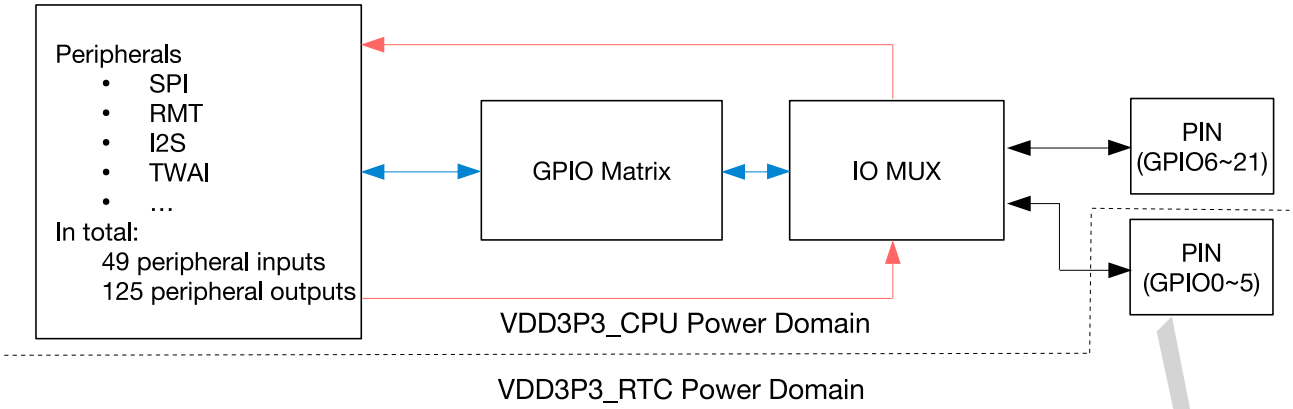


Figure 5-1. Diagram of IO MUX and GPIO Matrix

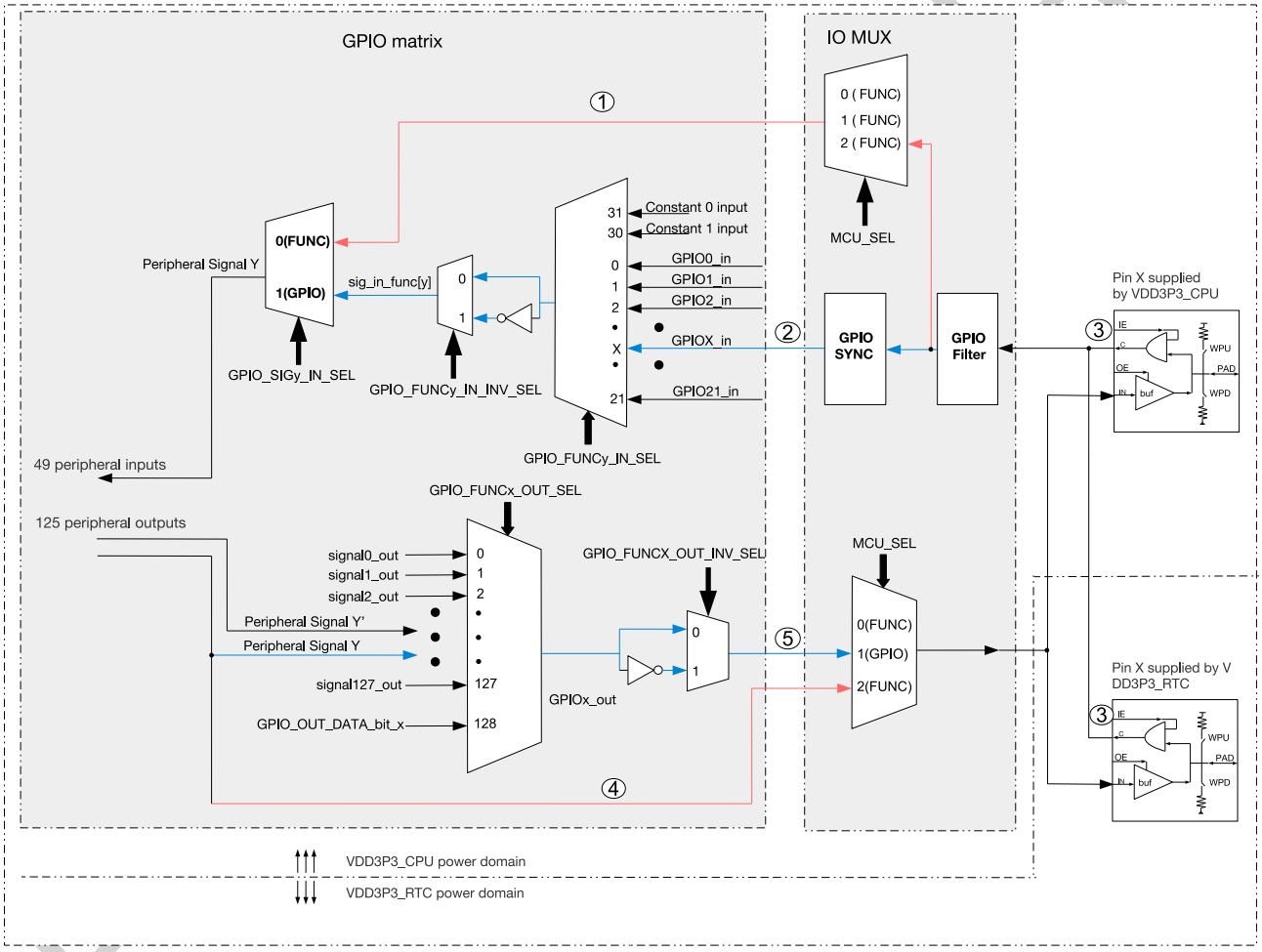


Figure 5-2. Architecture of IO MUX and GPIO Matirx

1. Only part of peripheral input signals (Y: 0 ~ 3, 6 ~ 7, 9 ~ 10, 63 ~ 68) can bypass GPIO matrix. The other input signals can only be routed to peripherals via GPIO matrix.
2. There are only 22 inputs from GPIO SYNC to GPIO matrix, since ESP32-C3 provides 22 GPIO pins in total.
3. The pins supplied by VDD3P3_CPU or by VDD3P3_RTC are controlled by the signals: IE, OE, WPU, and WPD.
4. Only part of peripheral outputs (0 ~ 6, 63~ 68) can be routed to pins bypassing GPIO matrix. See Table 5-1.

5. There are only 22 outputs (GPIO pin X: 0 ~ 21) from GPIO matrix to IO MUX.

Figure 5-3 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 22 GPIO pins and can be controlled using IE, OE, WPU, and WPD signals.

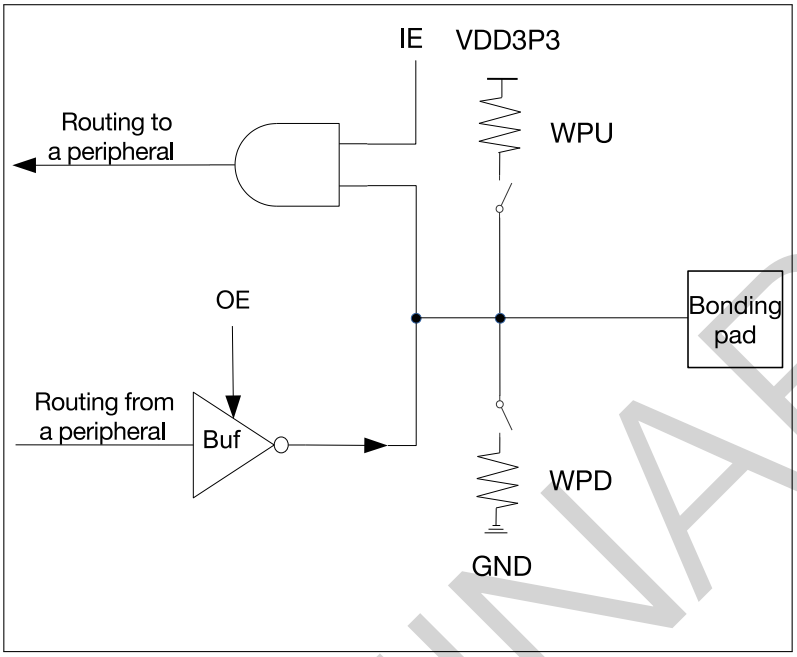


Figure 5-3. Internal Structure of a Pad

Note:

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

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 22 GPIOs (0 ~ 21), see Table 5-1. Meanwhile, register corresponding to the peripheral signal should be set to receive input signal via GPIO matrix.

5.4.2 Signal Synchronization

When signals are directed from pins using GPIO matrix, the signals will be synchronized to the APB bus clock by GPIO SYNC hardware, then go to GPIO matrix. This synchronization applies to all GPIO matrix signals but does not apply when using the IO MUX, see Figure 5-2.

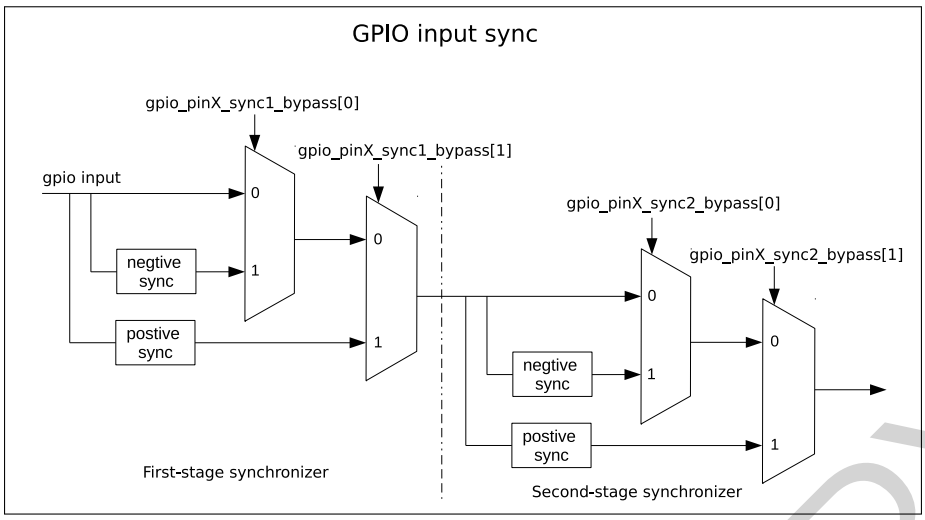


Figure 5-4. GPIO Input Synchronized on APB Clock Rising Edge or on Falling Edge

Figure 5-4 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.

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

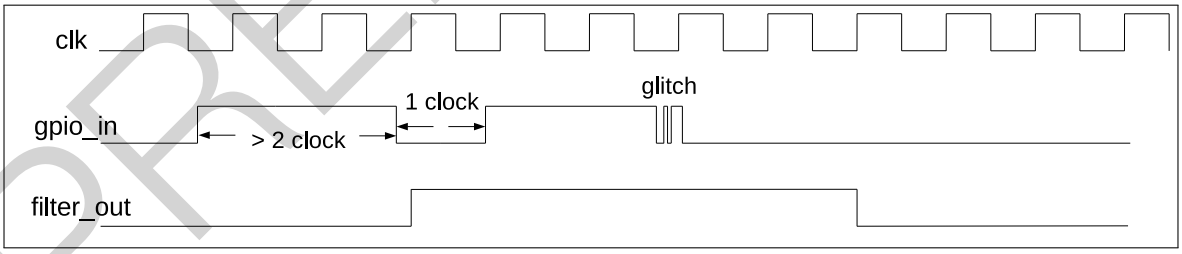


Figure 5-5. Filter Timing of GPIO Input Signals

3. Synchronize GPIO input. To do so, please set GPIO_PIN*x*_REG corresponding to GPIO pin *X* as follows:
   - Set GPIO_PIN*x*_SYNC1_BYPASS to enable input signal synchronized on rising edge or on falling edge in the first clock, see Figure 5-4.
   - Set GPIO_PIN*x*_SYNC2_BYPASS to enable input signal synchronized on rising edge or on falling edge in the second clock, see Figure 5-4.

4. Configure IO MUX register to enable pin input. For this end, please set IO_MUX_GPIO*x*_REG corresponding to GPIO pin *X* as follows:
    - Set IO_MUX_GPIO*x*_FUN_IE to enable input[2].
    - Set or clear IO_MUX_GPIO*x*_FUN_WPU and IO_MUX_GPIO*x*_FUN_WPD, as desired, to enable or disable pull-up and pull-down resistors.

For example, to connect I2S MCLK input signal[3] (I2S_MCLK_in, signal index 12) to GPIO7, please follow the steps below. Note that GPIO7 is also named as MTDO pin.

1. Set GPIO_SIG12_IN_SEL in register GPIO_FUNC12_IN_SEL_CFG_REG to enable peripheral signal input via GPIO matrix.
2. Set GPIO_FUNC12_IN_SEL in register GPIO_FUNC12_IN_SEL_CFG_REG to 7.
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_FUNC*y*_IN_INV_SEL.
3. It is possible to have a peripheral read a constantly low or constantly high input value without connecting this input to a pin. This can be done by selecting a special GPIO_FUNC*y*_IN_SEL input, instead of a GPIO number:
    - When GPIO_FUNC*y*_IN_SEL is set to 0x1F, input signal is always 0.
    - When GPIO_FUNC*y*_IN_SEL is set to 0x1E, input signal is always 1.

5.4.4 Simple GPIO Input

GPIO_IN_REG holds the input values of each GPIO pin. The input value of any GPIO pin can be read at any time without configuring GPIO matrix for a particular peripheral signal. However, it is necessary to enable the input via IO MUX by setting IO_MUX_GPIO*x*_FUN_IE bit in register IO_MUX_GPIO*x*_REG corresponding to pin *X*, as mentioned in Section 5.4.2.

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 (0 ~ 59, 63~ 127) to one of the 22 GPIOs (0 ~ 21). See Table 5-1.

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) which are not connected to any peripheral, but to the input signals (97 ~ 100) directly. These can be used to input a signal from one GPIO pin and output directly to another GPIO pin.

5.5.2 Functional Description

Some of the 125 output signals (0 ~ 59, 63~ 127) can be set to go through GPIO matrix into IO MUX and then to a pin. Figure 5-2 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 5-1), clear GPIO_FUNC*x*
     _OEN_SEL bit instead.
   - Clear 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 value)
     - 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. 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[21] correspond to GPIO0 ~ GPIO21, and GPIO_OUT_REG[25:22] 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.

5.5.4 Sigma Delta Modulated Output (SDM)

5.5.4.1 Functional Description

Four out of the 125 peripheral outputs (output index: 55 ~ 58) support 1-bit second-order sigma delta modulation. By default output is enabled for these four channels. This modulator can also output PDM (pulse density modulation) signal with configurable duty cycle. The transfer function of this second-order SDM modulator is:

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

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

- Set GPIOSD_FUNCTION_CLK_EN to enable the modulator clock.
- Configure register GPIOSD_SD*n*_PRESCALE (*n* is 0 ~ 3 for four channels).

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

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

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

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

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

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

5.5.4.2 SDM Configuration

The configuration of SDM is shown below:

- Route one of SDM outputs to a pin via GPIO matrix, see Section 5.5.2.
- Enable the modulator clock by setting the register GPIOSD_FUNCTION_CLK_EN.
- Configure the divider value by setting the register GPIOSD_SD*n*_PRESCALE.
- Configure the duty cycle of SDM output signal by setting the register GPIOSD_SD*n*_IN.

5.6 Direct Input and Output via IO MUX

5.6.1 Overview

Some high-speed signals (SPI and JTAG) can bypass GPIO matrix for better high-frequency digital performance. In this case, IO MUX is used to connect these pins directly to 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 5.11.
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. For the list of pin functions, please refer to Section 5.11.

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 ESP32-C3 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 ESP32-C3 pins.

5.8 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 ~21), 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.

5.9 Power Supplies and Management of GPIO Pins

5.9.1 Power Supplies of GPIO Pins

For more information on the power supply for GPIO pins, please refer to Pin Definition in *ESP32-C3 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.

5.9.2 Power Supply Management

Each ESP32-C3 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

5.10 Peripheral Signal List

Table 5-1 shows the peripheral input/output signals via GPIO matrix.

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

- GPIO_FUNC*n*_OEN_SEL = 1: the output enable is controlled by the corresponding bit *n* of GPIO_ENABLE_REG:
  - GPIO_ENABLE_REG = 0: output is disabled;
  - GPIO_ENABLE_REG = 1: output is enabled;
- GPIO_FUNC*n*_OEN_SEL = 0: use the output enable signal from peripheral, for example SPIQ_oe in the column "Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0" of Table 5-1. Note that the signals such as SPIQ_oe can be 1 (1'd1) or 0 (1'd0), depending on the configuration of corresponding peripherals. If it's 1'd1 in the "Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0", it indicates that once the register GPIO_FUNC*n*_OEN_SEL is cleared, the output signal is always enabled by default.

Note:

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

- For input signals, only 0 ~ 3, 6 ~ 19, 28 ~ 35, 45, 51 ~ 54, 63 ~ 68, 74, 77 ~ 80, 97 ~ 100 are valid.
- For output signals, only 0 ~ 39, 45 ~ 59, 63 ~ 127 are valid.

Table 5-1. Peripheral Signals via GPIO Matrix

Signal No.	Input Signal	Default value	Direct Input through IO MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output through 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	yes	U0RTS_out	1'd1	no
8	U0DSR_in	0	no	U0DTR_out	1'd1	no
9	U1RXD_in	0	yes	U1TXD_out	1'd1	no
10	U1CTS_in	0	yes	U1RTS_out	1'd1	no
11	U1DSR_in	0	no	U1DTR_out	1'd1	no
12	I2S_MCLK_in	0	no	I2S_MCLK_out	1'd1	no
13	I2SO_BCK_in	0	no	I2SO_BCK_out	1'd1	no
13	I2SO_WS_in	0	no	I2SO_WS_out	1'd1	no
15	I2SI_SD_in	0	no	I2SO_SD_out	1'd1	no
16	I2SI_BCK_in	0	no	I2SI_BCK_out	1'd1	no
17	I2SI_WS_in	0	no	I2SI_WS_out	1'd1	no
18	gpio_bt_priority	0	no	gpio_wlan_prio	1'd1	no
19	gpio_bt_active	0	no	gpio_wlan_active	1'd1	no
20	-	-	-	cpu_test_bu0	1'd1	no
21	-	-	-	cpu_test_bu1	1'd1	no
22	-	-	-	cpu_test_bu2	1'd1	no
23	-	-	-	cpu_test_bu3	1'd1	no

Signal No.	Input Signal	Default value	Direct Input through IO_MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output through IO_MUX
24	-	-	-	cpu_test_bu4	1'd1	no
25	-	-	-	cpu_test_bu5	1'd1	no
26	-	-	-	cpu_test_bu6	1'd1	no
27	-	-	-	cpu_test_bu7	1'd1	no
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	-	-	-	usb_jtag_tck	1'd1	no
37	-	-	-	usb_jtag_tms	1'd1	no
38	-	-	-	usb_jtag_tdi	1'd1	no
39	-	-	-	usb_jtag_tdo	1'd1	no
40	-	-	-	-	1'd1	no
41	-	-	-	-	1'd1	no
42	-	-	-	-	1'd1	no
43	-	-	-	-	1'd1	no
44	-	-	-	-	1'd1	no
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

Signal No.	Input Signal	Default value	Direct Input through IO_MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output through IO_MUX
50	-	-	-	ledc_ls_sig_out5	1'd1	no
51	rmt_sig_in0	0	no	rmt_sig_out0	1'd1	no
52	rmt_sig_in1	0	no	rmt_sig_out1	1'd1	no
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	-	-	-	gpio_sd0_out	1'd1	no
56	-	-	-	gpio_sd1_out	1'd1	no
57	-	-	-	gpio_sd2_out	1'd1	no
58	-	-	-	gpio_sd3_out	1'd1	no
59	-	-	-	I2SO_SD1_out	1'd1	no
60	-	-	-	-	1'd1	-
61	-	-	-	-	1'd1	-
62	-	-	-	-	1'd1	-
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	twai_rx	1	no	twai_tx	1'd1	no
75	-	-	-	twai_bus_off_on	1'd1	no

Signal No.	Input Signal	Default value	Direct Input through IO_MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output through IO_MUX
76	-	-	-	twai_clkout	1'd1	no
77	pcmfsync_in	0	no	bt_audio0_irq	1'd1	no
78	pcmclk_in	0	no	bt_audio1_irq	1'd1	no
79	pcmdin	0	no	bt_audio2_irq	1'd1	no
80	rw_wakeup_req	0	no	ble_audio0_irq	1'd1	no
81	-	-	-	ble_audio1_irq	1'd1	no
82	-	-	-	ble_audio2_irq	1'd1	no
83	-	-	-	pcmfsync_out	pcmfsync_en	no
84	-	-	-	pcmclk_out	pcmclk_en	no
85	-	-	-	pcmdout	pcmdout_en	no
86	-	-	-	ble_audio_sync0_p	1'd1	no
87	-	-	-	ble_audio_sync1_p	1'd1	no
88	-	-	-	ble_audio_sync2_p	1'd1	no
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	-	-	-	syncerr	!efuse_dis_btlc_gpio1	no

Signal No.	Input Signal	Default value	Direct Input through IO_MUX	Output Signal	Output enable signal when GPIO_FUNC*n*_OEN_SEL = 0	Direct Output through IO_MUX
102	-	-	-	syncfound_flag	!efuse_dis_btlc_gpio1	no
103	-	-	-	evt_cntl_immediate_abort	!(efuse_dis_btlc_gpio1&efuse_dis_btlc_gpio0)	no
104	-	-	-	linklbl	!efuse_dis_btlc_gpio1&!efuse_dis_btlc_gpio0	no
105	-	-	-	data_en	!efuse_dis_btlc_gpio1&!efuse_dis_btlc_gpio0	no
106	-	-	-	data	!efuse_dis_btlc_gpio1&!efuse_dis_btlc_gpio0	no
107	-	-	-	pkt_tx_on	!efuse_dis_btlc_gpio1	no
108	-	-	-	pkt_rx_on	!efuse_dis_btlc_gpio1	no
109	-	-	-	rw_tx_on	!efuse_dis_btlc_gpio1	no
110	-	-	-	rw_rx_on	!efuse_dis_btlc_gpio1	no
111	-	-	-	evt_req_p	!(efuse_dis_btlc_gpio1&efuse_dis_btlc_gpio0)	no
112	-	-	-	evt_stop_p	!(efuse_dis_btlc_gpio1&efuse_dis_btlc_gpio0)	no
113	-	-	-	bt_mode_on	!(efuse_dis_btlc_gpio1&efuse_dis_btlc_gpio0)	no
114	-	-	-	gpio_lc_diag0	!efuse_dis_btlc_gpio1	no
115	-	-	-	gpio_lc_diag1	!efuse_dis_btlc_gpio1	no
116	-	-	-	gpio_lc_diag2	!efuse_dis_btlc_gpio1	no
117	-	-	-	ch_idx	!efuse_dis_btlc_gpio1&!efuse_dis_btlc_gpio0	no
118	-	-	-	rx_window	!efuse_dis_btlc_gpio1	no
119	-	-	-	update_rx	!efuse_dis_btlc_gpio1	no
120	-	-	-	rx_status	!efuse_dis_btlc_gpio1	no
121	-	-	-	clk_gpio	!efuse_dis_btlc_gpio1	no
122	-	-	-	nbt_ble	!(efuse_dis_btlc_gpio1&efuse_dis_btlc_gpio0)	no
123	-	-	-	CLK_OUT_out1	1'd1	no
124	-	-	-	CLK_OUT_out2	1'd1	no
125	-	-	-	CLK_OUT_out3	1'd1	no
126	-	-	-	SPICS1_out	1'd1	no
127	-	-	-	usb_jtag_trst	1'd1	no

5.11 IO MUX Functions List

Table 5-2 shows the IO MUX functions of each pin.

Table 5-2. IO MUX Pin Functions

GPIO	Pin Name	Function 0	Function 1	Function 2	Function 3	DRV	Reset	Notes
0	XTAL_32K_P	GPIO0	GPIO0	-	-	2	0	R
1	XTAL_32K_N	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*	G
7	MTDO	MTDO	GPIO7	FSPID	-	2	1	G
8	GPIO8	GPIO8	GPIO8	-	-	2	1	-
9	GPIO9	GPIO9	GPIO9	-	-	2	3	-
10	GPIO10	GPIO10	GPIO10	FSPICS0	-	2	1	G
11	VDD_SPI	GPIO11	GPIO11	-	-	2	0	-
12	SPIHD	SPIHD	GPIO12	-	-	2	3	-
13	SPIWP	SPIWP	GPIO13	-	-	2	3	-
14	SPICS0	SPICS0	GPIO14	-	-	2	3	-
15	SPICLK	SPICLK	GPIO15	-	-	2	3	-
16	SPID	SPID	GPIO16	-	-	2	3	-
17	SPIQ	SPIQ	GPIO17	-	-	2	3	-
18	GPIO18	GPIO18	GPIO18	-	-	3	0	USB, G
19	GPIO19	GPIO19	GPIO19	-	-	3	0*	USB
20	U0RXD	U0RXD	GPIO20	-	-	2	1	G
21	U0TXD	U0TXD	GPIO21	-	-	2	1	-

Drive Strength

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

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

Reset Configurations

"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)

- **0*** - IE = 0, WPU = 0. The USB pull-up value of GPIO19 is 1 by default, therefore, the pin's pull-up resistor is enabled. For more information, see the note below.
- **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.
- **USB** - GPIO18 and GPIO19 are USB pins. The pull-up value of the two pins are controlled by the pins' pull-up value together with USB pull-up value. If any one of the pull-up value is 1, the pin's pull-up resistor will be enabled. The pull-up resistors of USB pins are controlled by USB_SERIAL_JTAG_DP_PULLUP.
- **G** - These pins have glitches during power-up. See details in Table 5-3.

Table 5-3. Power-Up Glitches on Pins

Pin	Glitch	Typical Time Period (ns)
MTCK	Low-level glitch	5
MTDO	Low-level glitch	5
GPIO10	Low-level glitch	5
U0RXD	Low-level glitch	5
GPIO18	Pull-up glitch	50000

5.12 Analog Functions List

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

Table 5-4. Analog Functions of IO MUX Pins

GPIO Num	Pin Name	Analog Function 0	Analog Function 1
0	XTAL_32K_P	XTAL_32K_P	ADC1_CH0
1	XTAL_32K_N	XTAL_32K_N	ADC1_CH1
2	GPIO2	-	ADC1_CH2
3	GPIO3	-	ADC1_CH3
4	MTMS	-	ADC1_CH4

Note:

1. The pin VDD_SPI can be configured as either power supply or normal GPIO.

2. The pins GPIO18 and GPIO19 can be configured as USB pins. For detailed configuration, please refer to 5 *USB Serial/JTAG Controller (USB_SERIAL_JTAG) [to be added later]*.

5.13 Register Summary

The addresses in this section are relative to GPIO Matrix, IO MUX and SDM base addresses provided in Table 3-4 in Chapter 3 *System and Memory*.

5.13.1 GPIO Matrix Register Summary

Name	Description	Address	Access
Configuration Registers			
GPIO_BT_SELECT_REG	GPIO bit select register	0x0000	R/W
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
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 PRO_CPU interrupt status register	0x005C	RO
GPIO_PCPU_NMI_INT_REG	GPIO PRO_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 pin0 configuration register	0x0074	R/W
GPIO_PIN1_REG	GPIO pin1 configuration register	0x0078	R/W
GPIO_PIN2_REG	GPIO pin2 configuration register	0x007C	R/W
GPIO_PIN3_REG	GPIO pin3 configuration register	0x0080	R/W
GPIO_PIN4_REG	GPIO pin4 configuration register	0x0084	R/W
GPIO_PIN5_REG	GPIO pin5 configuration register	0x0088	R/W
GPIO_PIN6_REG	GPIO pin6 configuration register	0x008C	R/W
GPIO_PIN7_REG	GPIO pin7 configuration register	0x0090	R/W
GPIO_PIN8_REG	GPIO pin8 configuration register	0x0094	R/W
GPIO_PIN9_REG	GPIO pin9 configuration register	0x0098	R/W
GPIO_PIN10_REG	GPIO pin10 configuration register	0x009C	R/W
GPIO_PIN11_REG	GPIO pin11 configuration register	0x00A0	R/W
GPIO_PIN12_REG	GPIO pin12 configuration register	0x00A4	R/W
GPIO_PIN13_REG	GPIO pin13 configuration register	0x00A8	R/W
GPIO_PIN14_REG	GPIO pin14 configuration register	0x00AC	R/W
GPIO_PIN15_REG	GPIO pin15 configuration register	0x00B0	R/W
GPIO_PIN16_REG	GPIO pin16 configuration register	0x00B4	R/W
GPIO_PIN17_REG	GPIO pin17 configuration register	0x00B8	R/W
GPIO_PIN18_REG	GPIO pin18 configuration register	0x00BC	R/W
GPIO_PIN19_REG	GPIO pin19 configuration register	0x00C0	R/W
GPIO_PIN20_REG	GPIO pin20 configuration register	0x00C4	R/W
GPIO_PIN21_REG	GPIO pin21 configuration register	0x00C8	R/W
Input Function Configuration Registers			
GPIO_FUNC0_IN_SEL_CFG_REG	Configuration register for input signal 0	0x0154	R/W

Name	Description	Address	Access
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
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
GPIO_FUNC21_OUT_SEL_CFG_REG	Configuration register for GPIO21 output	0x05A8	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

5.13.2 IO MUX Register Summary

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 XTAL_32K_P	0x0004	R/W
IO_MUX_GPIO1_REG	IO MUX configuration register for pin XTAL_32K_N	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

Name	Description	Address	Access
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
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 GPIO19	0x0050	R/W
IO_MUX_GPIO20_REG	IO MUX configuration register for pin U0RXD	0x0054	R/W
IO_MUX_GPIO21_REG	IO MUX configuration register for pin U0TXD	0x0058	R/W
Version Register			
IO_MUX_DATE_REG	IO MUX Version Control Register	0x00FC	R/W

5.13.3 SDM Register Summary

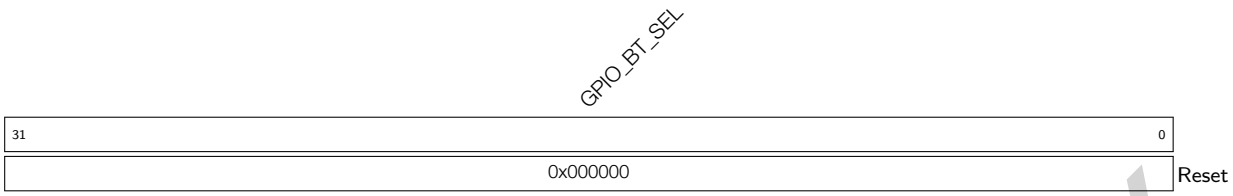
Name	Description	Address	Access
Configuration registers			
GPIOSD_SIGMADELTA0_REG	Duty Cycle Configuration Register of SDM0	0x0000	R/W
GPIOSD_SIGMADELTA1_REG	Duty Cycle Configuration Register of SDM1	0x0004	R/W
GPIOSD_SIGMADELTA2_REG	Duty Cycle Configuration Register of SDM2	0x0008	R/W
GPIOSD_SIGMADELTA3_REG	Duty Cycle Configuration Register of SDM3	0x000C	R/W
GPIOSD_SIGMADELTA_CG_REG	Clock Gating Configuration Register	0x0020	R/W
GPIOSD_SIGMADELTA_MISC_REG	MISC Register	0x0024	R/W
Version register			
GPIOSD_SIGMADELTA_VERSION_REG	Version Control Register	0x0028	R/W

5.14 Registers

The addresses in this section are relative to GPIO Matrix, IO MUX and SDM base addresses provided in Table 3-4 in Chapter 3 *System and Memory*.

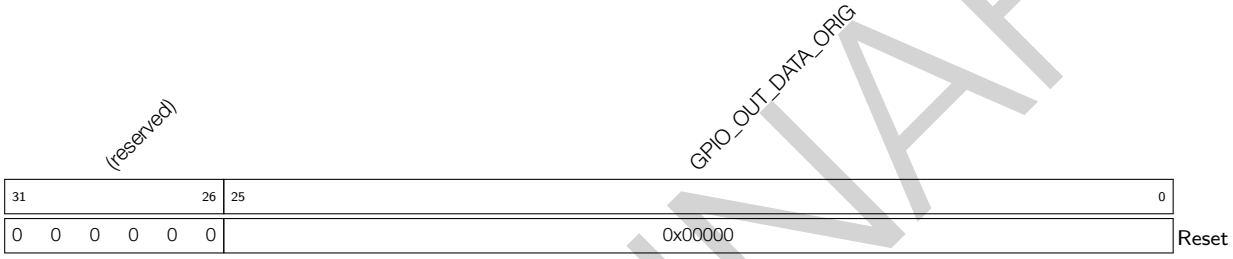
5.14.1 GPIO Matrix Registers

Register 5.1. GPIO_BT_SELECT_REG (0x0000)



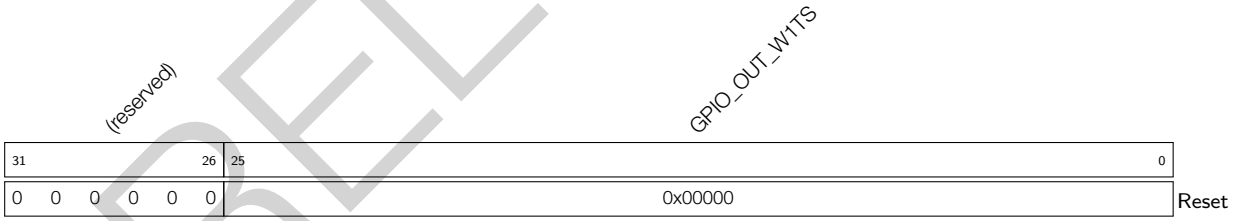
GPIO_BT_SEL Reserved (R/W)

Register 5.2. GPIO_OUT_REG (0x0004)



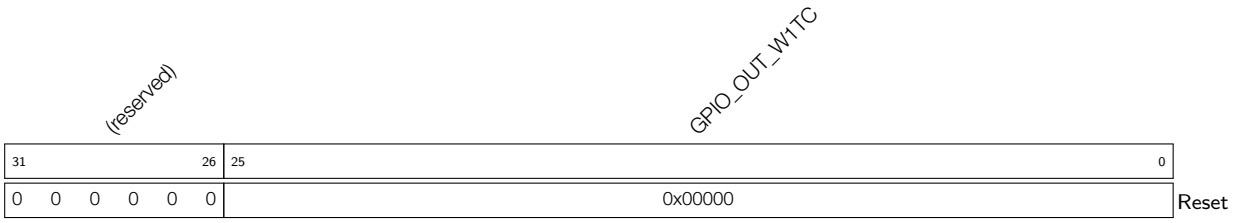
GPIO_OUT_DATA_ORIG GPIO0 ~ 21 output value in simple GPIO output mode. The values of bit0 ~ bit21 correspond to the output value of GPIO0 ~ GPIO21 respectively, and bit22 ~ bit25 are invalid. (R/W/SS)

Register 5.3. GPIO_OUT_W1TS_REG (0x0008)



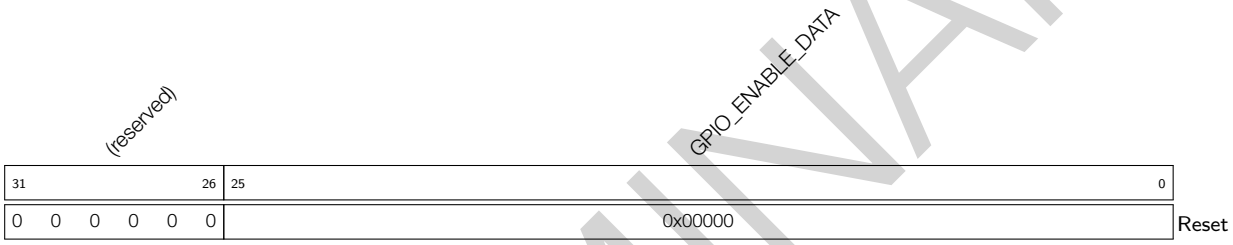
GPIO_OUT_W1TS GPIO0 ~ 21 output set register. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 are invalid. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT_REG will be set to 1. Recommended operation: use this register to set GPIO_OUT_REG. (WT)

Register 5.4. GPIO_OUT_W1TC_REG (0x000C)



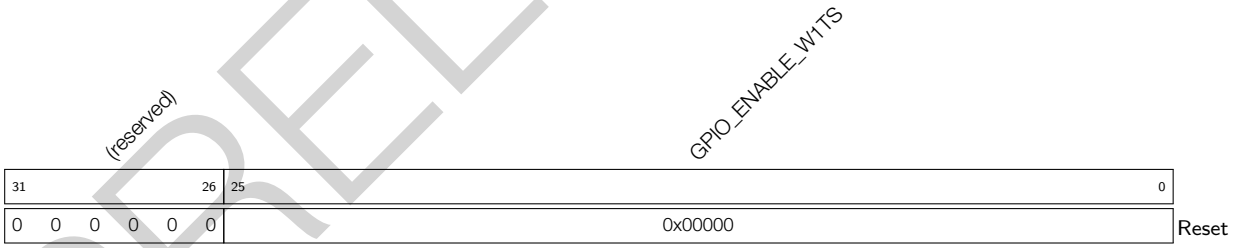
GPIO_OUT_W1TC GPIO0 ~ 21 output clear register. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 are invalid. If the value 1 is written to a bit here, the corresponding bit in GPIO_OUT_REG will be cleared. Recommended operation: use this register to clear GPIO_OUT_REG. (WT)

Register 5.5. GPIO_ENABLE_REG (0x0020)



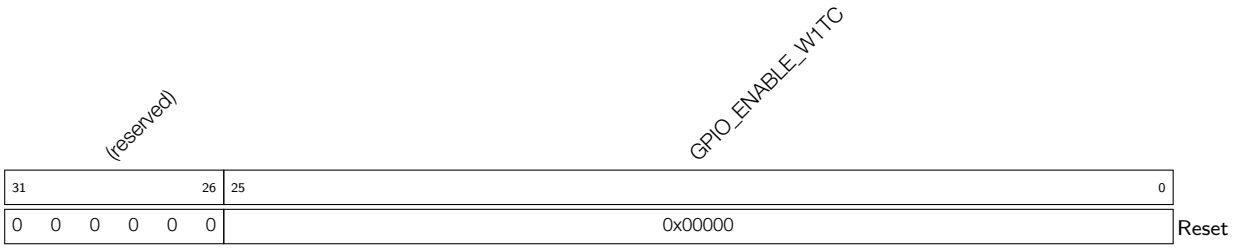
GPIO_ENABLE_DATA GPIO output enable register for GPIO0 ~ 21. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 are invalid. (R/W/SS)

Register 5.6. GPIO_ENABLE_W1TS_REG (0x0024)



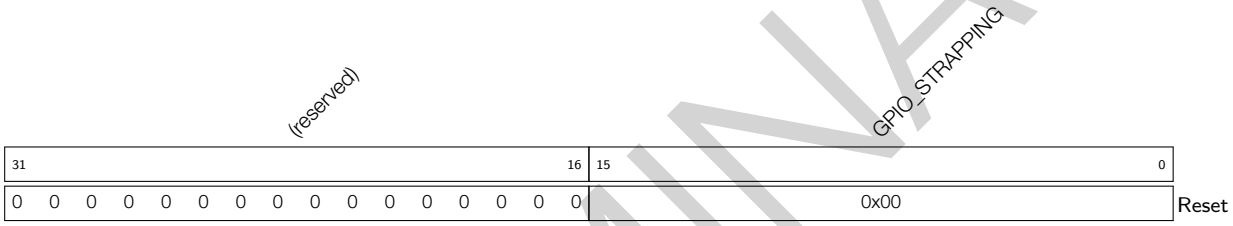
GPIO_ENABLE_W1TS GPIO0 ~ 21 output enable set register. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 are invalid. If the value 1 is written to a bit here, the corresponding bit in GPIO_ENABLE_REG will be set to 1. Recommended operation: use this register to set GPIO_ENABLE_REG. (WT)

Register 5.7. GPIO_ENABLE_W1TC_REG (0x0028)



GPIO_ENABLE_W1TC GPIO0 ~ 21 output enable clear register. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 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)

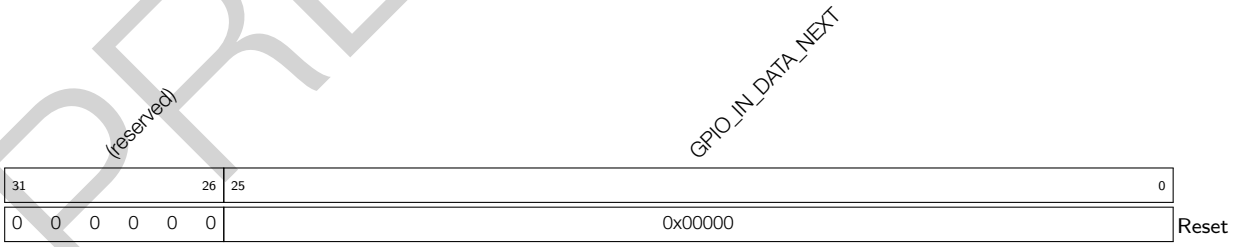
Register 5.8. GPIO_STRAP_REG (0x0038)



GPIO_STRAPPING GPIO strapping values. (RO)

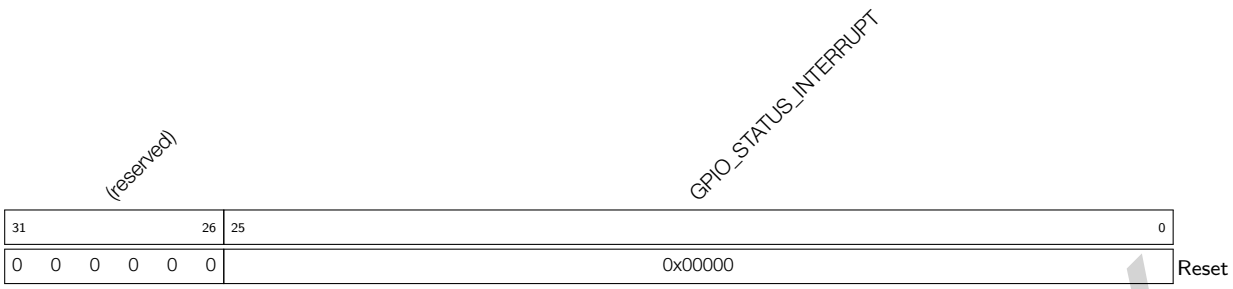
- bit 0: GPIO2
- bit 2: GPIO8
- bit 3: GPIO9

Register 5.9. GPIO_IN_REG (0x003C)



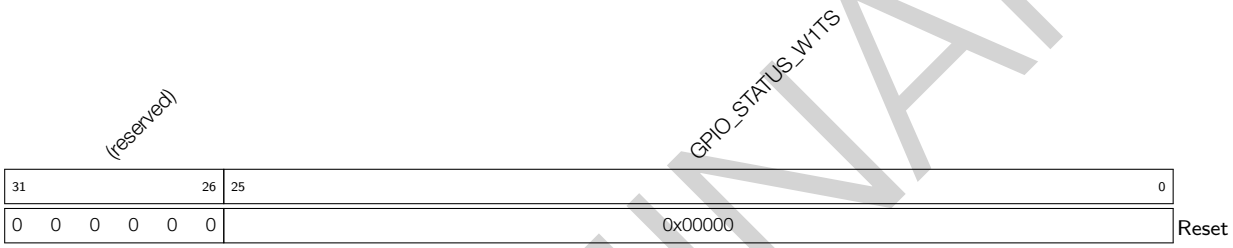
GPIO_IN_DATA_NEXT GPIO0 ~ 21 input value. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 are invalid. Each bit represents a pin input value, 1 for high level and 0 for low level. (RO)

Register 5.10. GPIO_STATUS_REG (0x0044)



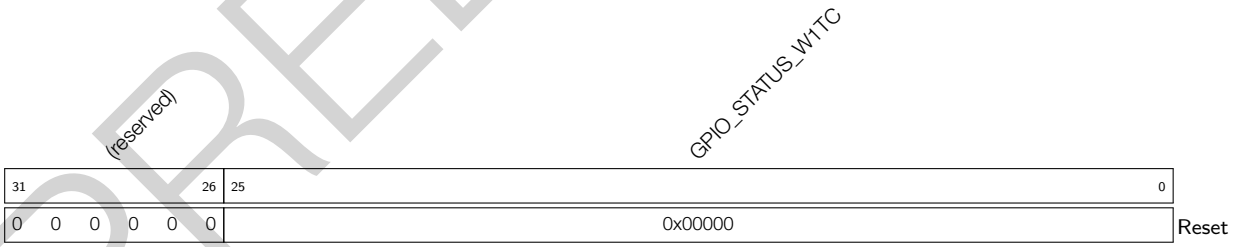
GPIO_STATUS_INTERRUPT GPIO0 ~ 21 interrupt status register. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 are invalid. (R/W/SS)

Register 5.11. GPIO_STATUS_W1TS_REG (0x0048)



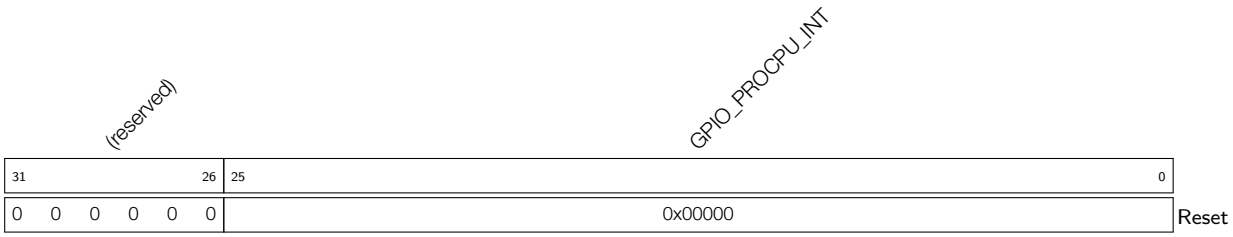
GPIO_STATUS_W1TS GPIO0 ~ 21 interrupt status set register. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 are invalid. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be set to 1. Recommended operation: use this register to set GPIO_STATUS_INTERRUPT. (WT)

Register 5.12. GPIO_STATUS_W1TC_REG (0x004C)



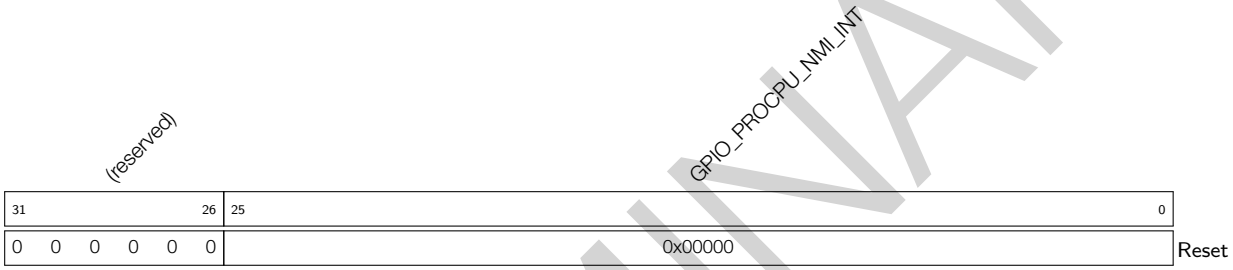
GPIO_STATUS_W1TC GPIO0 ~ 21 interrupt status clear register. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 are invalid. If the value 1 is written to a bit here, the corresponding bit in GPIO_STATUS_INTERRUPT will be cleared. Recommended operation: use this register to clear GPIO_STATUS_INTERRUPT. (WT)

Register 5.13. GPIO_PCPU_INT_REG (0x005C)



GPIO_PROCPU_INT GPIO0 ~ 21 PRO_CPU interrupt status. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 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 5.14. GPIO_PCPU_NMI_INT_REG (0x0060)



GPIO_PROCPU_NMI_INT GPIO0 ~ 21 PRO_CPU non-maskable interrupt status. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 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 5.15. GPIO_PIN*n*_REG (*n*: 0-21) (0x0074+4n*)**

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

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

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

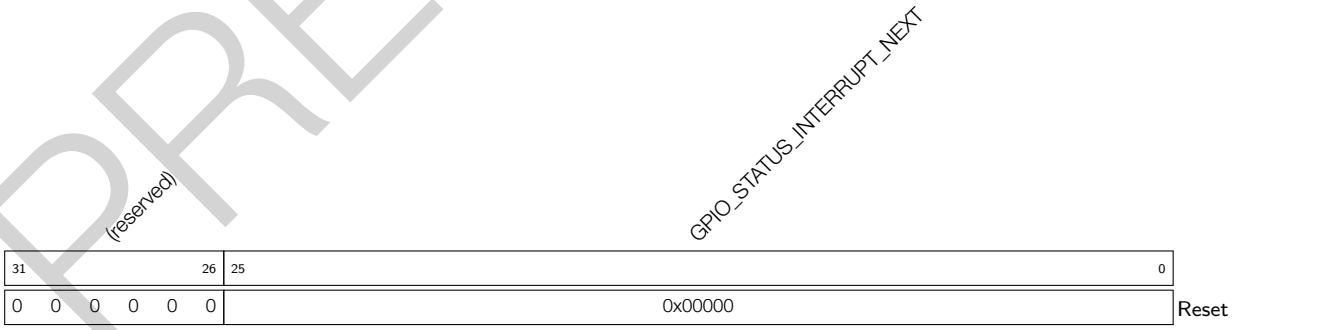
GPIO_PIN*n*_INT_TYPE Interrupt type selection. 0: GPIO interrupt disabled; 1: rising edge trigger; 2: falling edge trigger; 3: any edge trigger; 4: low level trigger; 5: high level trigger. (R/W)

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

GPIO_PIN*n*_CONFIG reserved (R/W)

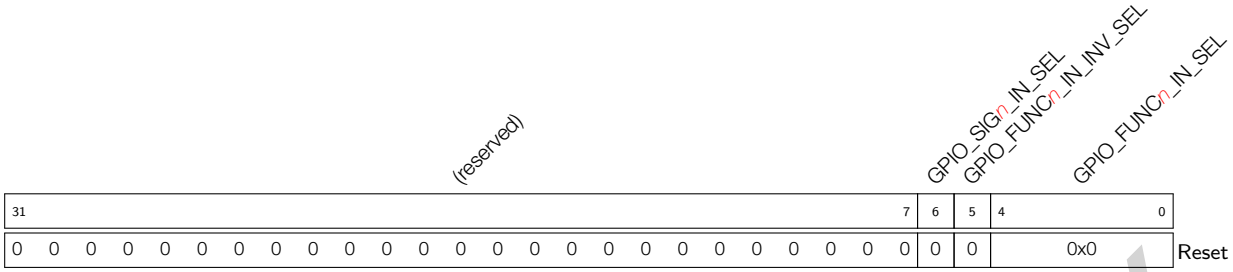
GPIO_PIN*n*_INT_ENA Interrupt enable bits. bit13: CPU interrupt enabled; bit14: CPU non-maskable interrupt enabled. (R/W)

Register 5.16. GPIO_STATUS_NEXT_REG (0x014C)



GPIO_STATUS_INTERRUPT_NEXT Interrupt source signal of GPIO0 ~ 21, could be rising edge interrupt, falling edge interrupt, level sensitive interrupt and any edge interrupt. Bit0 ~ bit21 are corresponding to GPIO0 ~ 21, and bit22 ~ bit25 are invalid. (RO)

Register 5.17. GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0-127) (0x0154+4n*)**

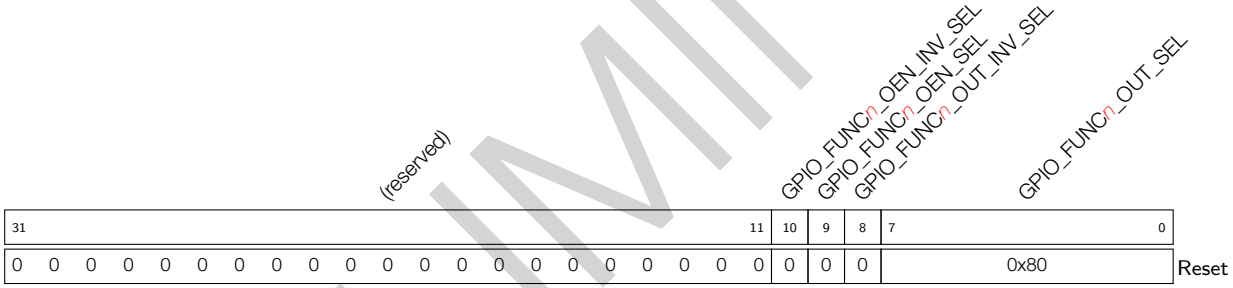


GPIO_FUNC*n*_IN_SEL Selection control for peripheral input signal *n*, selects a pin from the 22 GPIO matrix pins to connect this input signal. Or selects 0x1e for a constantly high input or 0x1f for a constantly low input. (R/W)

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 5.18. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-21) (0x0554+4n*)**



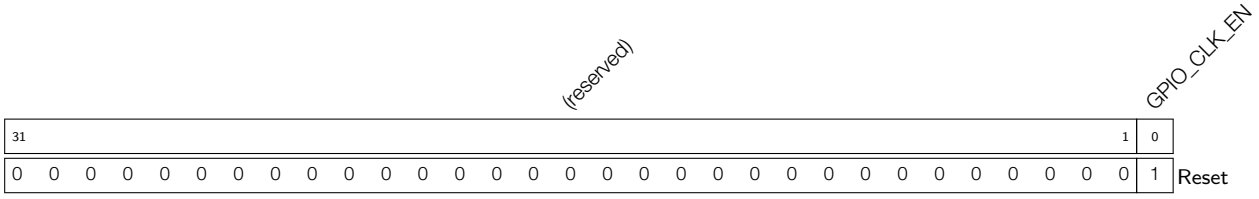
GPIO_FUNC*n*_OUT_SEL Selection control for GPIO output *n*. If a value *Y* (0<=Y<128) is written to this field, the peripheral output signal *Y* will be connected to GPIO output *X*. If a value 128 is written to this field, bit *n* of GPIO_OUT_REG and GPIO_ENABLE_REG will be selected as the output value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

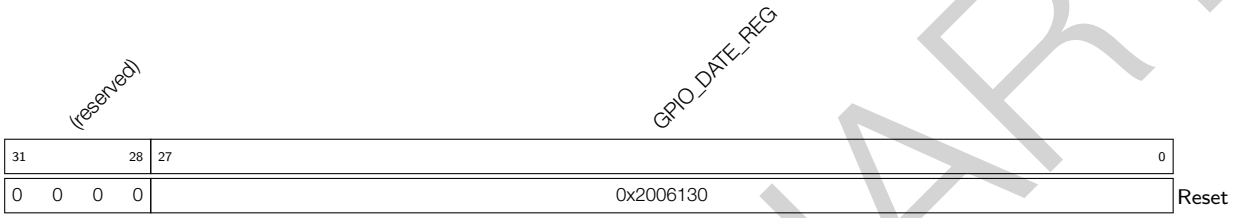
GPIO_FUNC*n*_OEN_INV_SEL 0: Do not invert the output enable signal; 1: Invert the output enable signal. (R/W)

Register 5.19. GPIO_CLOCK_GATE_REG (0x062C)



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

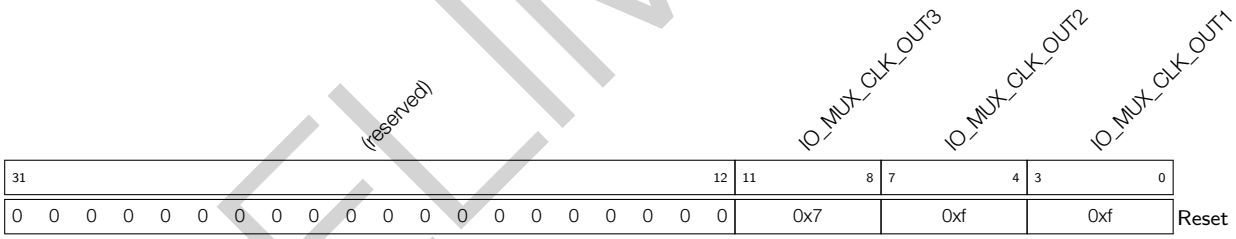
Register 5.20. GPIO_DATE_REG (0x06FC)



GPIO_DATE_REG Version control register (R/W)

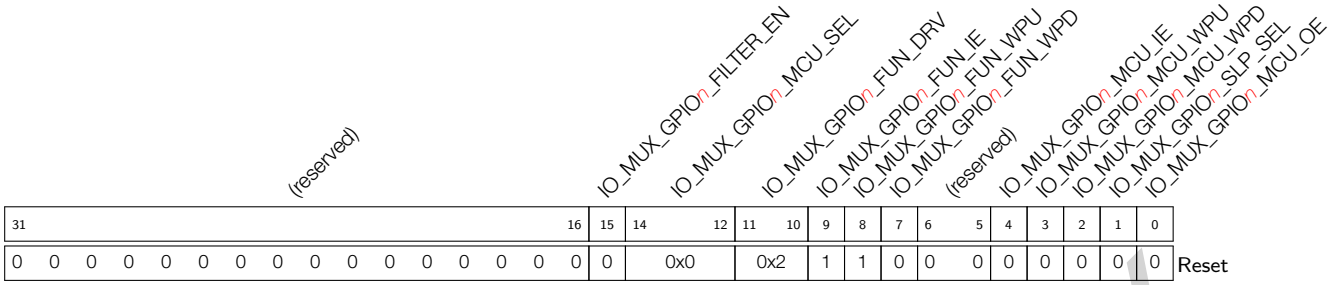
5.14.2 IO MUX Registers

Register 5.21. 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 5-1. (R/W)

Register 5.22. IO_MUX_GPIO*n*_REG (*n*: 0-21) (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*_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)

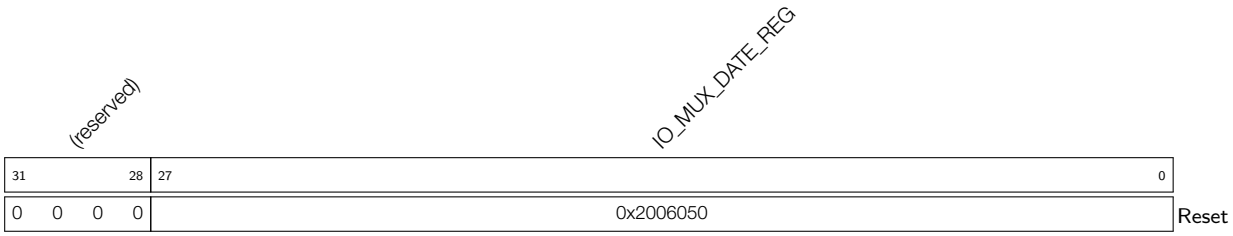
IO_MUX_GPIO*n*_FUN_IE Input enable of the pin. 1: input enabled; 0: input disabled. (R/W)

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

IO_MUX_GPIO*n*_MCU_SEL Select IO MUX function for this signal. 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; 2: Filter disabled. (R/W)

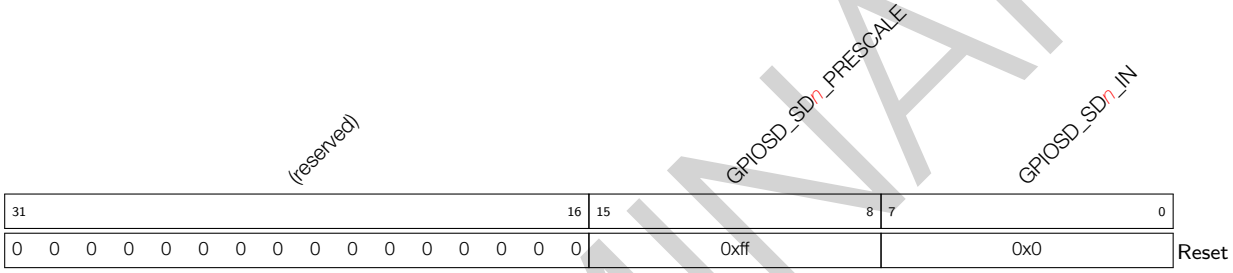
Register 5.23. IO_MUX_DATE_REG (0x00FC)



IO_MUX_DATE_REG Version control register (R/W)

5.14.3 SDM Output Registers

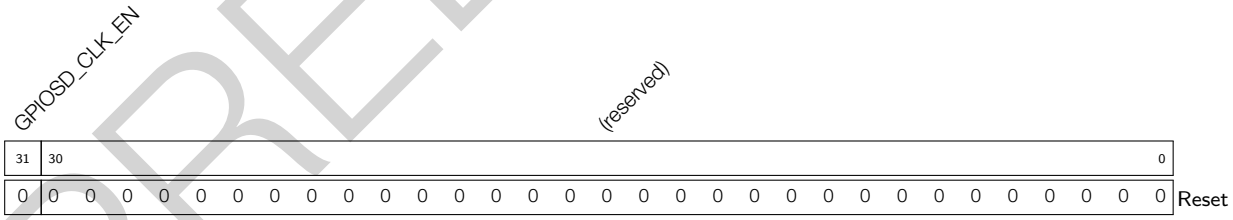
Register 5.24. GPIOSD_SIGMADELTA*n*_REG (*n*: 0-3) (0x0000+4n*)**



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

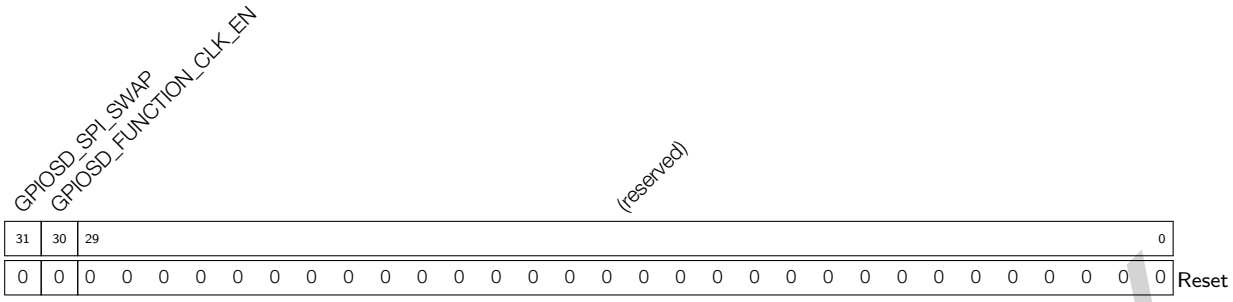
GPIOSD_SD*n*_PRESCALE This field is used to set a divider value to divide APB clock. (R/W)

Register 5.25. GPIOSD_SIGMADELTA_CG_REG (0x0020)



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

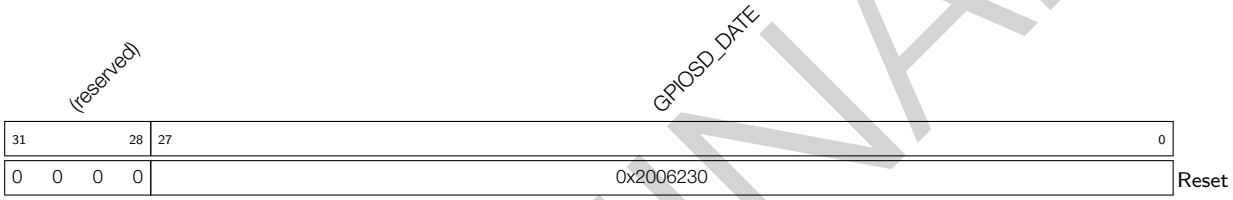
Register 5.26. GPIOSD_SIGMADELTA_MISC_REG (0x0024)



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

GPIOSD_SPI_SWAP Reserved. (R/W)

Register 5.27. GPIOSD_SIGMADELTA_VERSION_REG (0x0028)



GPIOSD_DATE Version Control Register. (R/W)

6 Reset and Clock

6.1 Reset

6.1.1 Overview

ESP32-C3 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

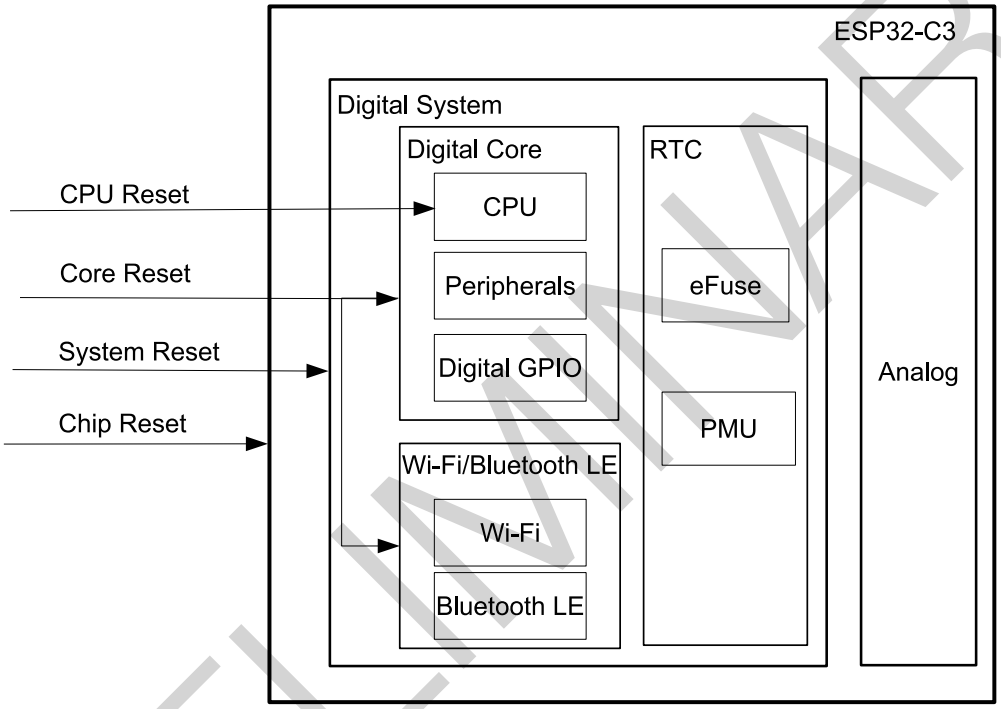


Figure 6-1. Reset Types

6.1.3 Features

- Support four reset levels:
  - CPU Reset: Only resets CPU core. Once such reset is triggered, 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: the CPU can trigger a software reset by configuring the corresponding registers.
  - Hardware Reset: Hardware reset is directly triggered by the circuit.

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 reset 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 7 *Watchdog Timers (WDT) [to be added later]*
0x13	CLK GLITCH reset	System Reset	See Chapter 16 *Clock Glitch Detection [to be added later]*
0x12	Super Watchdog reset	System Reset	See Chapter 7 *Watchdog Timers (WDT) [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 12 *Low-Power Management (RTC_CNTL) [to be added later]*
0x14	eFuse reset	Core Reset	Triggered by eFuse CRC error
0x17	Power glitch reset	Core Reset	Triggered by power glitch
0x07	MWDT0 core reset	Core Reset	See Chapter 7 *Watchdog Timers (WDT) [to be added later]*
0x08	MWDT1 core reset	Core Reset	See Chapter 7 *Watchdog Timers (WDT) [to be added later]*
0x09	RWDT core reset	Core Reset	See Chapter 7 *Watchdog Timers (WDT) [to be added later]*
0x0B	MWDT0 CPU reset	CPU Reset	See Chapter 7 *Watchdog Timers (WDT) [to be added later]*
0x0C	Software CPU reset	CPU Reset	Triggered by configuring RTC_CNTL_SW_PROCPU_RST
0x0D	RWDT CPU reset	CPU Reset	See Chapter 7 *Watchdog Timers (WDT) [to be added later]*
0x11	MWDT1 CPU reset	CPU Reset	See Chapter 7 *Watchdog Timers (WDT) [to be added later]*

[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 register configuration. See Chapter 12 *Low-Power Management (RTC_CNTL) [to be added later]*.

6.2 Clock

6.2.1 Overview

ESP32-C3 clocks are mainly sourced from 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

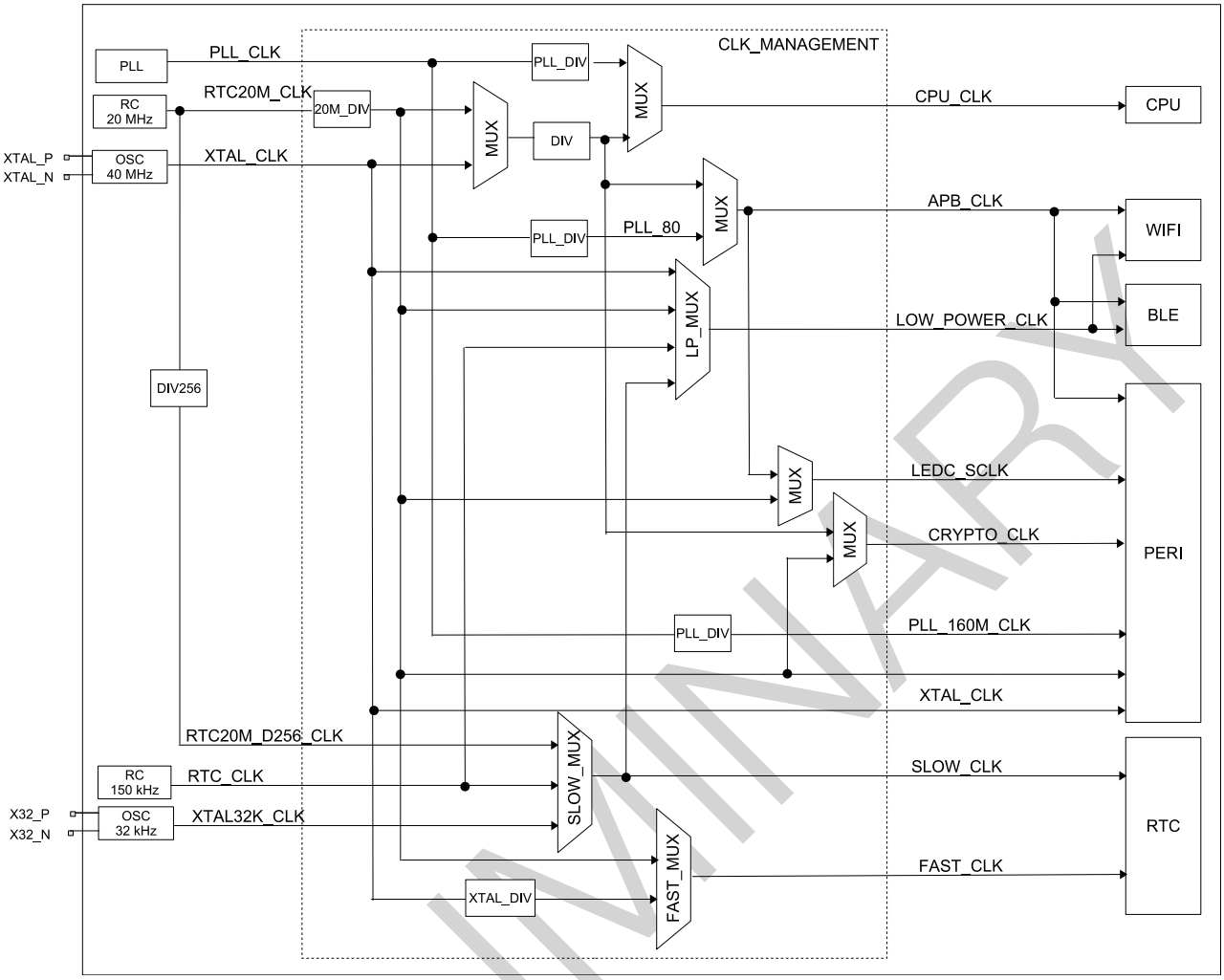


Figure 6-2. System Clock

6.2.3 Features

ESP32-C3 clocks can be classified in two types depending on their frequencies:

- High speed clocks for devices working at a higher frequency, such as CPU and digital peripherals
  - PLL_CLK (320 MHz or 480 MHz): internal PLL clock
  - XTAL_CLK (40 MHz): external crystal clock
- Slow speed clocks for low-power devices, such as RTC module and low-power peripherals
  - XTAL32K_CLK (32 kHz): external crystal clock
  - RTC20M_CLK (20 MHz by default): internal oscillator with adjustable frequency
  - RTC20M_D256_CLK (78.125 kHz by default): internal clock derived from RTC20M_CLK divided by 256
  - RTC_CLK (150 kHz by default): internal low power clock 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 160 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, RTC20M_CLK or XTAL_CLK as CPU_CLK clock source by configuring register 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_CLK Clock Source

SYSTEM_SOC_CLK_SEL Value	CPU Clock Source
0	XTAL_CLK
1	PLL_CLK
2	RTC20M_CLK

Table 6-3. CPU Clock Frequency

CPU Clock Source	SEL_0*	SEL_1*	SEL_2*	CPU Clock Frequency
XTAL_CLK	0	-	-	CPU_CLK = XTAL_CLK/(SYSTEM_PRE_DIV_CNT + 1) SYSTEM_PRE_DIV_CNT ranges from 0 ~ 1023. Default is 1
PLL_CLK (480 MHz)	1	1	0	CPU_CLK = PLL_CLK/6 CPU_CLK frequency is 80 MHz
PLL_CLK (480 MHz)	1	1	1	CPU_CLK = PLL_CLK/3 CPU_CLK frequency is 160 MHz
PLL_CLK (320 MHz)	1	0	0	CPU_CLK = PLL_CLK/4 CPU_CLK frequency is 80 MHz
PLL_CLK (320 MHz)	1	0	1	CPU_CLK = PLL_CLK/2 CPU_CLK frequency is 160 MHz
RTC20M_CLK	2	-	-	CPU_CLK = RTC20M_CLK/(SYSTEM_PRE_DIV_CNT + 1) SYSTEM_PRE_DIV_CNT ranges from 0 ~ 1023. Default is 1

* The value of register SYSTEM_SOC_CLK_SEL.

* The value of register SYSTEM_PLL_FREQ_SEL.

* The value of register SYSTEM_CPUPERIOD_SEL.

6.2.4.2 Peripheral Clock

Peripheral clocks include APB_CLK, CRYPTO_CLK, PLL_160M_CLK, LEDC_SCLK, XTAL_CLK, and RTC20M_CLK. Table 6-4 shows which clock can be used by each peripheral.

Table 6-4. Peripheral Clocks

Peripheral	XTAL_CLK	APB_CLK	PLL_160M_CLK	(RTC) FAST_CLK	RTC20M_CLK	CRYPTO_CLK	LEDC_SCLK
TIMG	Y	Y					
I2S	Y		Y				
UHCI		Y					
UART	Y	Y			Y		
RMT	Y	Y			Y		
I2C	Y				Y		
SPI	Y	Y					
eFuse Controller				Y			
SARADC		Y					
Temperature Sensor	Y				Y		
USB		Y					
CRYPTO						Y	
TWAI Controller		Y					
LEDC	Y	Y	Y		Y		Y
SYS_TIMER	Y	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	80 MHz
XTAL_CLK	CPU_CLK
RTC20M_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	160 MHz
XTAL_CLK	CPU_CLK
RTC20M_CLK	CPU_CLK

PLL_160M_CLK

PLL_160M_CLK is divided from PLL_CLK according to current PLL frequency.

LEDC_SCLK

LEDC module uses RTC20M_CLK as clock source when APB_CLK is disabled. In other words, when the system is in low-power mode, most peripherals will be halted (as APB_CLK is turned off), but LEDC can still work normally via RTC20M_CLK.

6.2.4.3 Wi-Fi and Bluetooth® LE Clock

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.

LOW_POWER_CLK uses XTAL32K_CLK, XTAL_CLK, RTC20M_CLK or SLOW_CLK (the low clock selected by RTC) as its clock source for Wi-Fi and Bluetooth LE in low-power mode.

6.2.4.4 RTC Clock

The clock sources for SLOW_CLK and FAST_CLK are low-frequency clocks. RTC module can operate when most other clocks are stopped. SLOW_CLK derived from RTC_CLK, XTAL32K_CLK or RTC20M_D256_CLK is used to clock Power Management module. FAST_CLK is used to clock On-chip Sensor module. It can be sourced from a divided XTAL_CLK or from a divided RTC20M_CLK.

7 Chip Boot Control

7.1 Overview

ESP32-C3 has three strapping pins:

- GPIO2
- GPIO8
- GPIO9

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

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

During system reset triggered by power-on, brown-out or by analog super watchdog (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 or shut down. Software can read the latch status (strapping value) from the register GPIO_STRAPPING.

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	Defualt Configuration
GPIO2	N/A
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 ESP32-C3. After the reset is released, the strapping pins work as normal-function pins.

7.2 Boot Mode Control

GPIO2, GPIO8, and GPIO9 control the boot mode after the reset is released.

Table 7-2. Boot Mode Control

Boot Mode	GPIO2	GPIO8	GPIO9
SPI Boot	1	x	1
Download Boot	1	1	0

Table 7-2 shows the strapping pin values of GPIO2, GPIO8 and GPIO9, and the associated boot modes. "x" means that this value is ignored.

In SPI Boot mode, the CPU boots the system by reading the program stored in SPI flash. SPI Boot mode can be

further classified as follows:

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

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

The following eFuses control boot mode behaviors:

- EFUSE_DIS_FORCE_DOWNLOAD

  If this eFuse is 0 (default), software can force switch the chip from SPI Boot mode to Download Boot mode by setting register RTC_CNTL_FORCE_DOWNLOAD_BOOT and triggering a CPU reset. If this eFuse is 1, RTC_CNTL_FORCE_DOWNLOAD_BOOT is disabled.

- EFUSE_DIS_DOWNLOAD_MODE

  If this eFuse is 1, Download Boot mode is disabled.

- EFUSE_ENABLE_SECURITY_DOWNLOAD

  If this eFuse is 1, Download Boot mode only allows reading, writing, and erasing plaintext flash and does not support any SRAM or register operations. Ignore this eFuse if Download Boot mode is disabled.

USB Serial/JTAG Controller can also force the chip into Download Boot mode from SPI Boot mode, as well as force the chip into SPI Boot mode from Download Boot mode. For detailed information, please refer to Chapter 5 *USB Serial/JTAG Controller (USB_SERIAL_JTAG) [to be added later]*.

7.3 ROM Code Printing Control

GPIO8 controls ROM code printing of information during the early 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
0	x	ROM code is always printed 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

ROM code will print to pin U0TXD (default) or to USB Serial/JTAG Controller during power-on, depending on the eFuse bit EFUSE_USB_PRINT_CHANNEL (0: USB; 1: UART). Note that if this eFuse bit is set to 0, i.e., USB is selected, but USB Serial/JTAG Controller is disabled, then ROM code will not print.

8 Timer Group (TIMG)

8.1 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 8-1, the ESP32-C3 chip contains two timer groups, namely timer group 0 and timer group 1. Each timer group consists of one general purpose timer referred to as T0 and one Main System Watchdog Timer. All general purpose timers are based on 16-bit prescalers and 54-bit auto-reload-capable up-down counters.

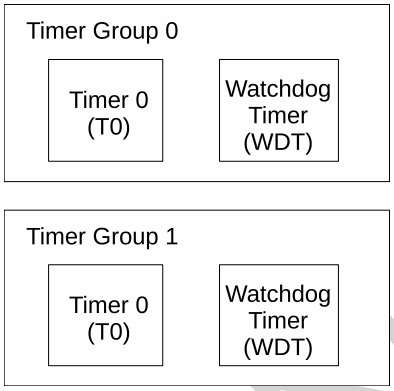


Figure 8-1. Timer Units within Groups

Note that while the Main System Watchdog Timer registers are described in this chapter, their functional description is included in the Chapter 7 *Watchdog Timers (WDT) [to be added later]*. Therefore, the term 'timers' within this chapter refers to the general purpose timers.

The timers' features are summarized as follows:

- A 16-bit clock prescaler, from 2 to 65536
- A 54-bit time-base counter programmable to incrementing or decrementing
- Able to read real-time value of the time-base counter
- Halting and resuming the time-base counter
- Programmable alarm generation
- Timer value reload (Auto-reload at alarm or software-controlled instant reload)
- Level interrupt generation

8.2 Functional Description

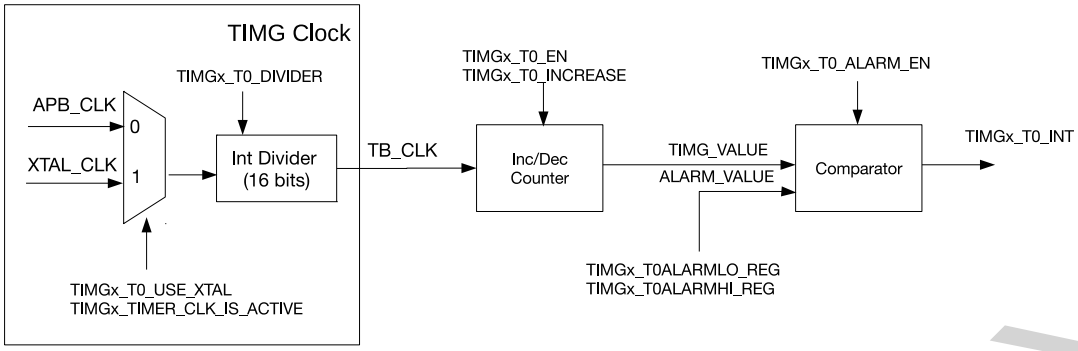


Figure 8-2. Timer Group Architecture

Figure8-2 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.

8.2.1 16-bit Prescaler and Clock Selection

The timer can select between the APB clock (APB_CLK) or external clock (XTAL_CLK) as its clock source by setting the TIMG_T0_USE_XTAL field of the TIMG_T0CONFIG_REG register. The selected clock is 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.

8.2.2 54-bit Time-base Counter

The 54-bit time-base counters are based on TB_CLK and can be configured to increment or decrement via the TIMG_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 while TIMG_T0_EN is set and this will cause the time-base counter to change direction instantly.

To read the 54-bit value of the time-base counter, the timer value must be latched to two registers before being read by the CPU (due to the CPU being 32-bit). By writing any value to the TIMG_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.

8.2.3 Alarm Generation

A timer can be configured to trigger an alarm when the timer's current value matches the alarm value. An alarm will cause an interrupt to occur and (optionally) an automatic reload of the timer's current value (see Section 8.2.4).

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, or lower than the alarm value (within a defined range) when the up-down counter decrements. Table 8-1 and Table 8-2 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 8-1. 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 8-2. 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.

8.2.4 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. 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.

8.2.5 SLOW_CLK Frequency Calculation

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

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

8.2.6 Interrupts

Each timer has its own interrupt line that can be routed to the CPU, and thus each timer group has a total of 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.

8.3 Configuration and Usage

8.3.1 Timer as a Simple Clock

1. Configure the time-base counter
   - Select clock source by setting or clearing TIMG_T0_USE_XTAL field.
   - 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.

8.3.2 Timer as One-shot Alarm

1. Configure the time-base counter following step 1 of Section 8.3.1.
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.

8.3.3 Timer as Periodic Alarm

1. Configure the time-base counter following step 1 in Section 8.3.1.
2. Configure the alarm following step 2 in Section 8.3.2.
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.

8.3.4 SLOW_CLK Frequency Calculation

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

8.4 Register Summary

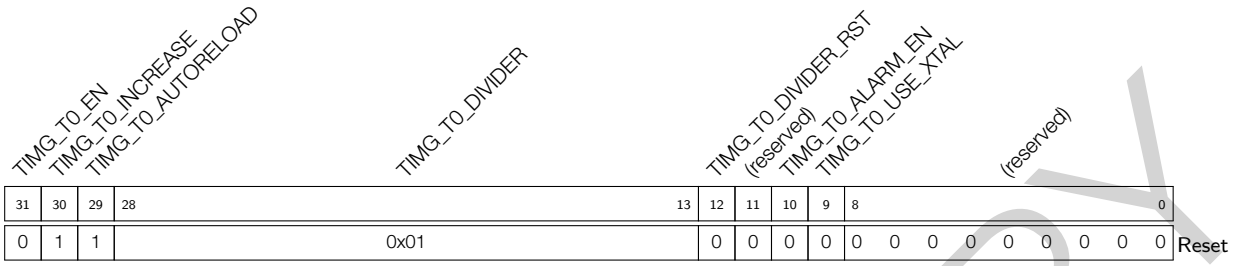
The addresses in this section are relative to Timer Group base addresses (one for Timer Group 0 and another one for Timer Group 1) provided in Table 3-4 in Chapter 3 *System and Memory*.

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 TIMGn_T0_(LO/HI)_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_T0_(LOADLOLOADHI)_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

8.5 Registers

The addresses in this section are relative to Timer Group base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Register 8.1. TIMG_T0CONFIG_REG (0x0000)



TIMG_T0_USE_XTAL 1: Use XTAL_CLK as the source clock of timer group. 0: Use APB_CLK as the source clock of timer group. (R/W)

TIMG_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)

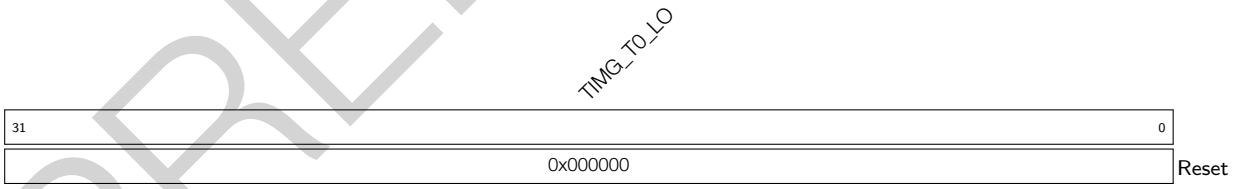
TIMG_T0_DIVIDER Timer 0 clock (T0_clk) prescaler value. (R/W)

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)

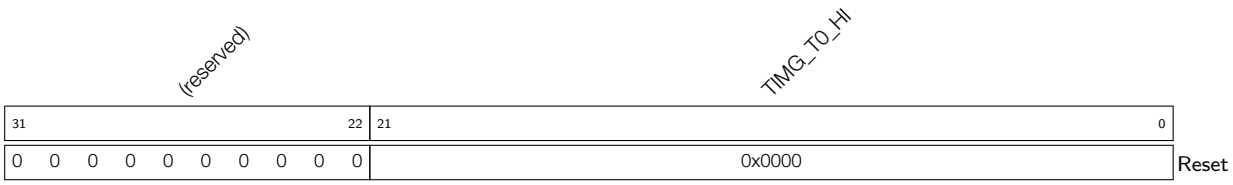
TIMG_T0_EN When set, the Timer 0 time-base counter is enabled. (R/W)

Register 8.2. TIMG_T0LO_REG (0x0004)



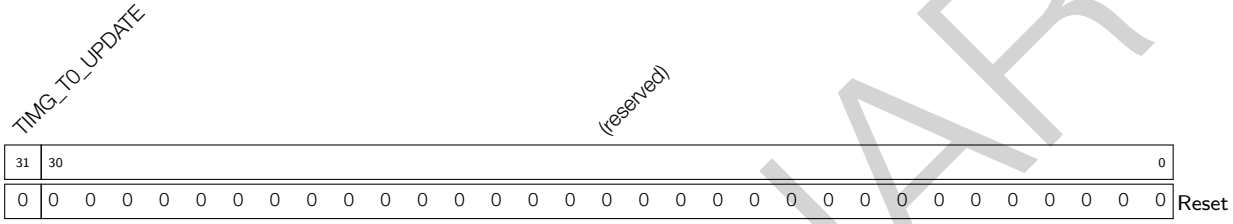
TIMG_T0_LO After writing to TIMG_T0UPDATE_REG, the low 32 bits of the time-base counter of Timer 0 can be read here. (RO)

Register 8.3. TIMG_T0HI_REG (0x0008)



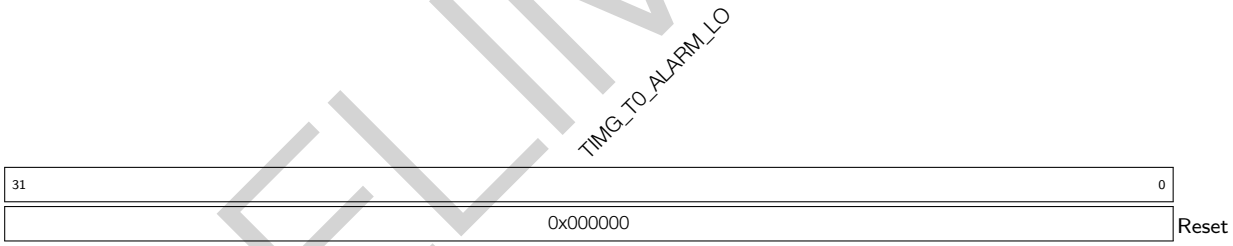
TIMG_T0_HI After writing to TIMG_T0UPDATE_REG, the high 22 bits of the time-base counter of Timer 0 can be read here. (RO)

Register 8.4. TIMG_T0UPDATE_REG (0x000C)



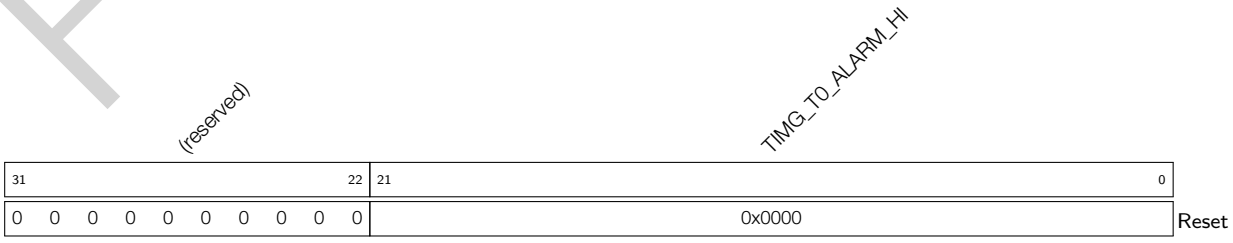
TIMG_T0_UPDATE After writing 0 or 1 to TIMG_T0UPDATE_REG, the counter value is latched. (R/W/SC)

Register 8.5. TIMG_T0ALARMLO_REG (0x0010)



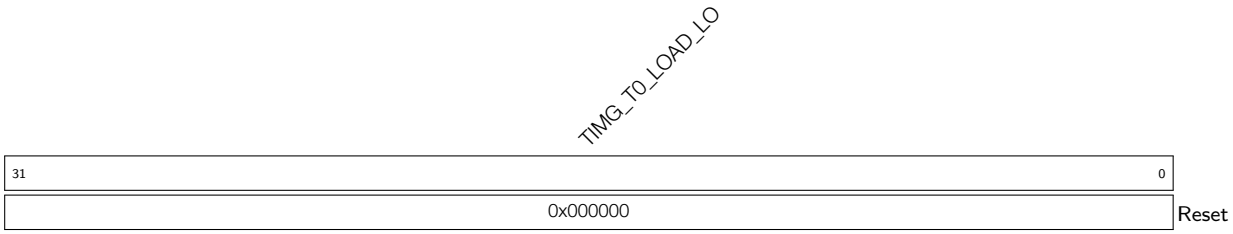
TIMG_T0_ALARM_LO Timer 0 alarm trigger time-base counter value, low 32 bits. (R/W)

Register 8.6. TIMG_T0ALARMHI_REG (0x0014)



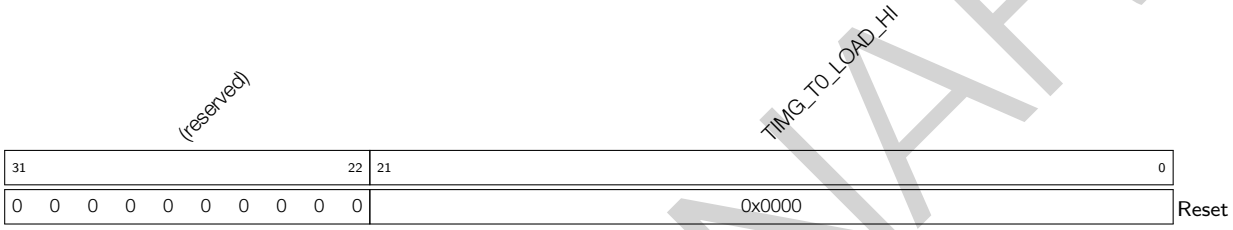
TIMG_T0_ALARM_HI Timer 0 alarm trigger time-base counter value, high 22 bits. (R/W)

Register 8.7. TIMG_T0LOADLO_REG (0x0018)



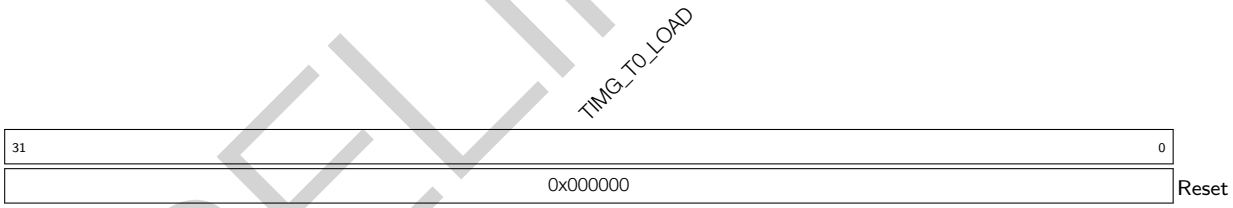
TIMG_T0_LOAD_LO Low 32 bits of the value that a reload will load onto Timer 0 time-base counter. (R/W)

Register 8.8. TIMG_T0LOADHI_REG (0x001C)



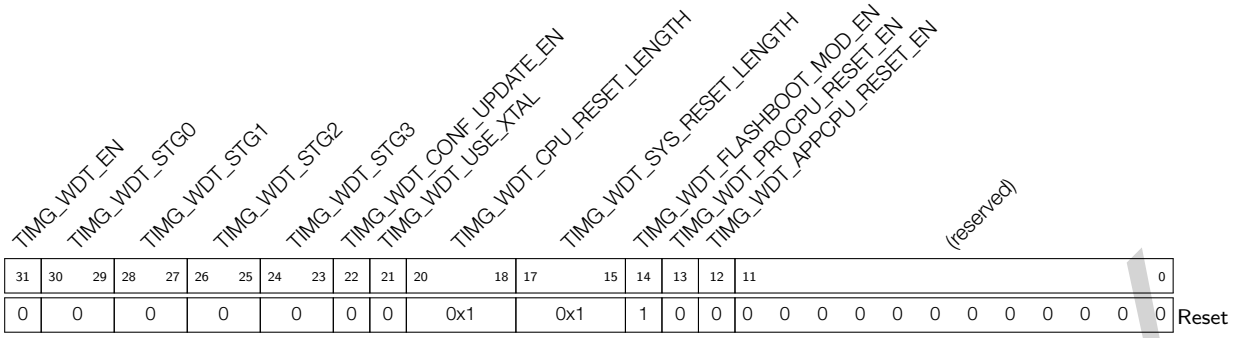
TIMG_T0_LOAD_HI High 22 bits of the value that a reload will load onto Timer 0 time-base counter. (R/W)

Register 8.9. TIMG_T0LOAD_REG (0x0020)



TIMG_T0_LOAD Write any value to trigger a Timer 0 time-base counter reload. (WT)

Register 8.10. TIMG_WDTCONFIG0_REG (0x0048)



TIMG_WDT_APPCPU_RESET_EN WDT reset CPU enable. (R/W)

TIMG_WDT_PROCPU_RESET_EN WDT reset CPU enable. (R/W)

TIMG_WDT_FLASHBOOT_MOD_EN When set, Flash boot protection is enabled. (R/W)

TIMG_WDT_SYS_RESET_LENGTH System reset signal length selection. 0: 100 ns, 1: 200 ns, 2: 300 ns, 3: 400 ns, 4: 500 ns, 5: 800 ns, 6: 1.6 μs, 7: 3.2 μs. (R/W)

TIMG_WDT_CPU_RESET_LENGTH CPU reset signal length selection. 0: 100 ns, 1: 200 ns, 2: 300 ns, 3: 400 ns, 4: 500 ns, 5: 800 ns, 6: 1.6 μs, 7: 3.2 μs. (R/W)

TIMG_WDT_USE_XTAL Chooses WDT clock. 0: APB_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)

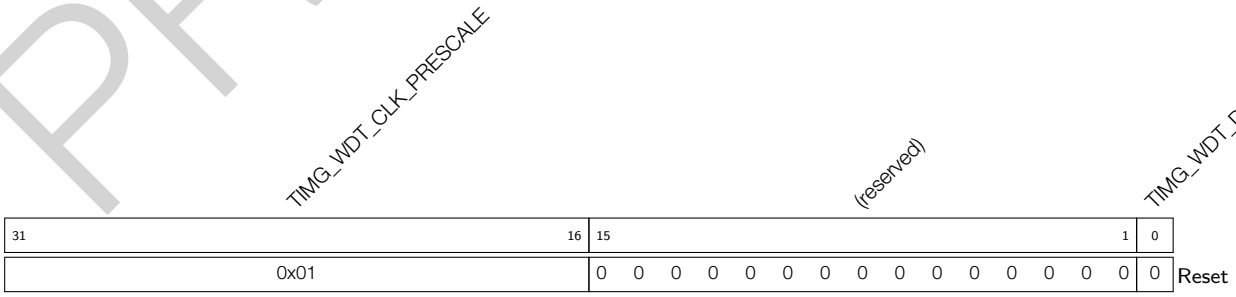
TIMG_WDT_STG2 Stage 2 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

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)

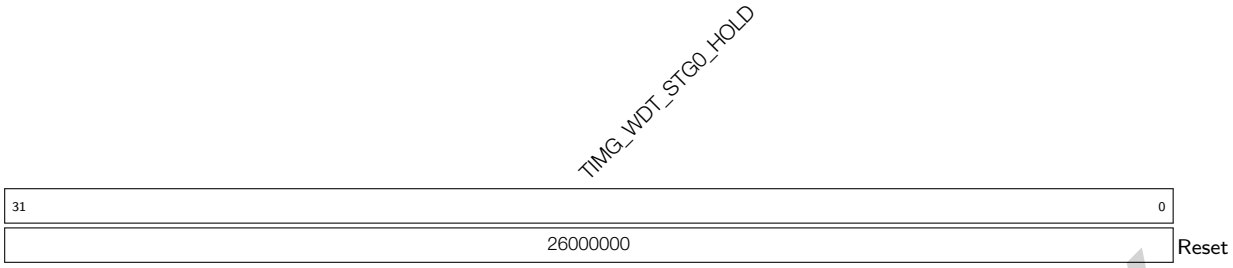
Register 8.11. TIMG_WDTCONFIG1_REG (0x004C)



TIMG_WDT_DIVCNT_RST When set, WDT 's clock divider counter will be reset. (WT)

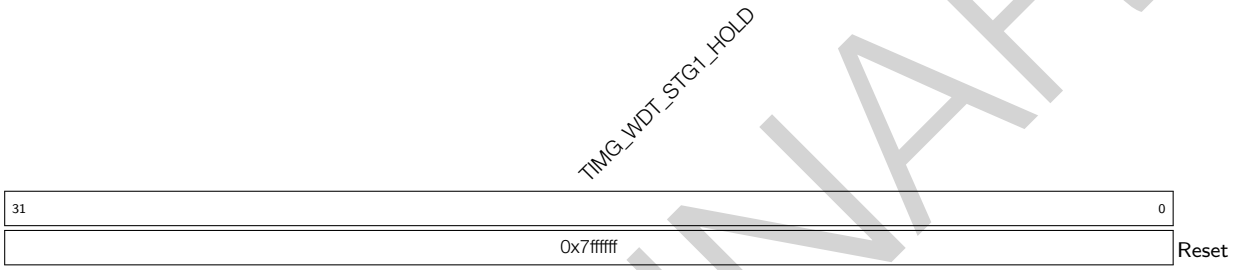
TIMG_WDT_CLK_PRESCALE MWDT clock prescaler value. MWDT clock period = 12.5 ns * TIMG_WDT_CLK_PRESCALE. (R/W)

Register 8.12. TIMG_WDTCONFIG2_REG (0x0050)



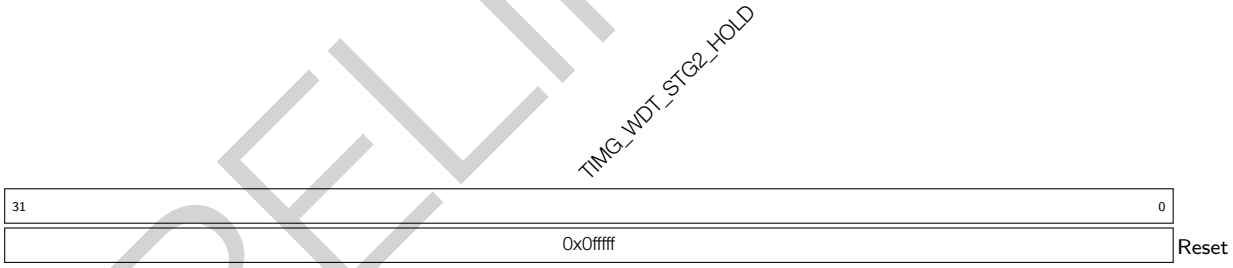
TIMG_WDT_STG0_HOLD Stage 0 timeout value, in MWDT clock cycles. (R/W)

Register 8.13. TIMG_WDTCONFIG3_REG (0x0054)



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

Register 8.14. TIMG_WDTCONFIG4_REG (0x0058)

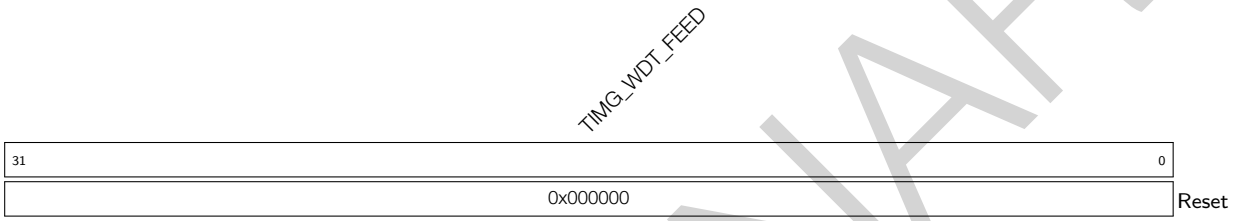


TIMG_WDT_STG2_HOLD Stage 2 timeout value, in MWDT clock cycles. (R/W)

Register 8.15. TIMG_WDTCONFIG5_REG (0x005C)

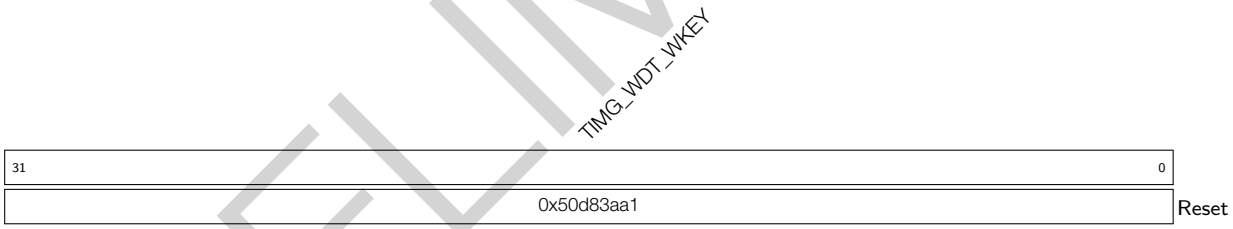
TIMG_WDT_STG3_HOLD Stage 3 timeout value, in MWDT clock cycles. (R/W)

Register 8.16. TIMG_WDTFEED_REG (0x0060)



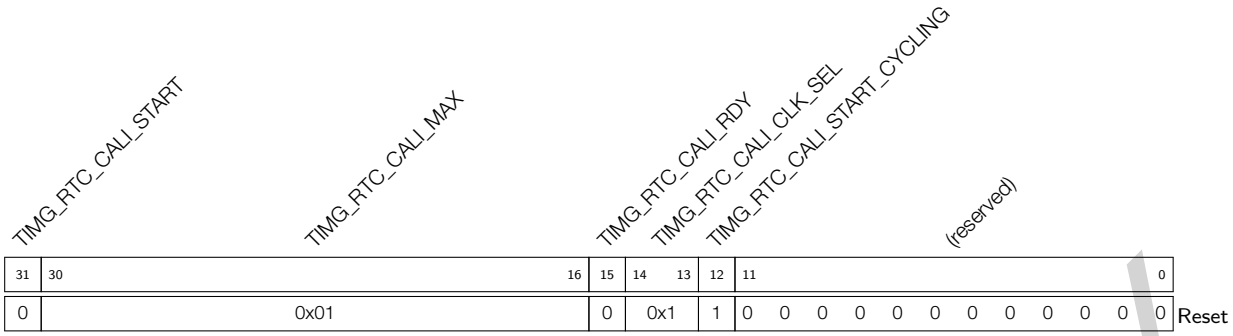
TIMG_WDT_FEED Write any value to feed the MWDT. (WO) (WT)

Register 8.17. 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 8.18. TIMG_RTCCALICFG_REG (0x0068)



TIMG_RTC_CALI_START_CYCLING Enables periodic frequency calculation. (R/W)

TIMG_RTC_CALI_CLK_SEL 0: RTC_CLK; 1: RTC20M_D256_CLK; 2: XTAL32K_CLK. (R/W)

TIMG_RTC_CALI_RDY Marks the completion of frequency calculation. (RO)

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

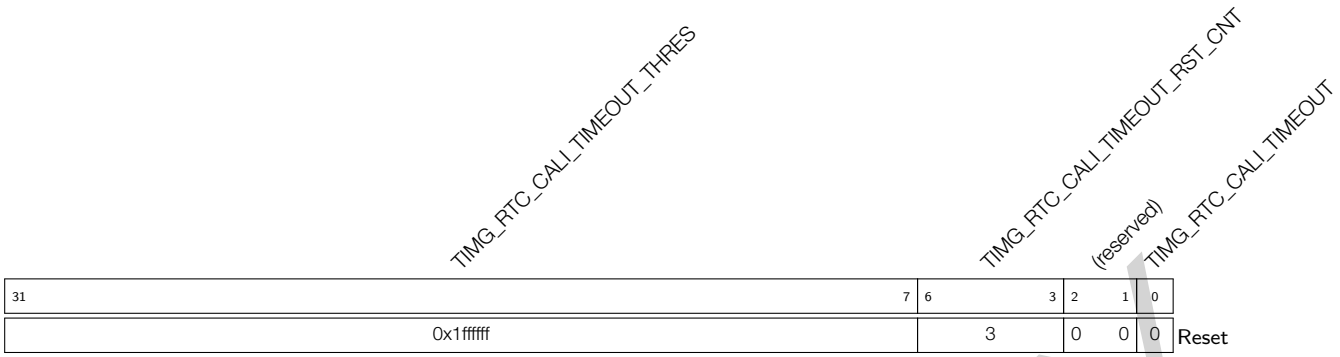
TIMG_RTC_CALI_START Enables one-shot frequency calculation. (R/W)

Register 8.19. TIMG_RTCCALICFG1_REG (0x006C)

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

TIMG_RTC_CALI_VALUE Frequency calculation result. (RO)

Register 8.20. TIMG_RTCCALICFG2_REG (0x0080)

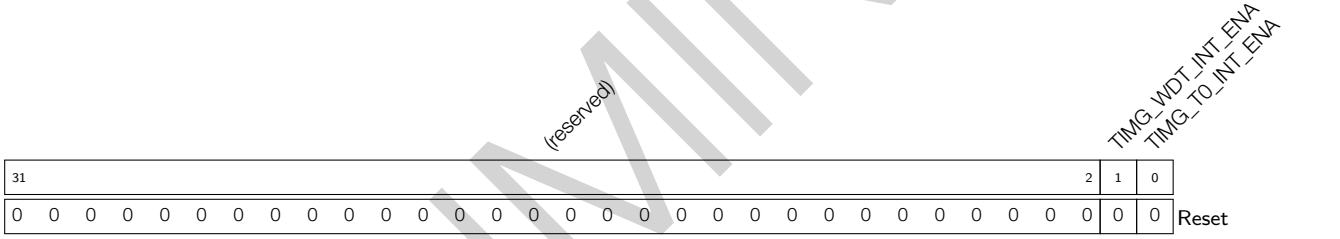


TIMG_RTC_CALI_TIMEOUT Indicates frequency calculation timeout. (RO)

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)

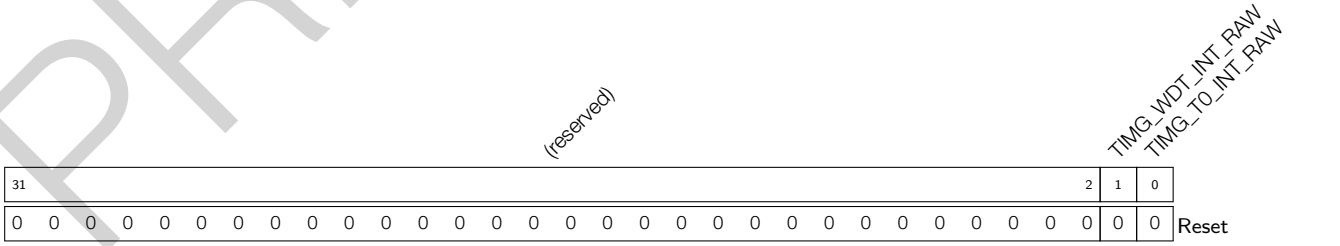
Register 8.21. TIMG_INT_ENA_TIMERS_REG (0x0070)



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)

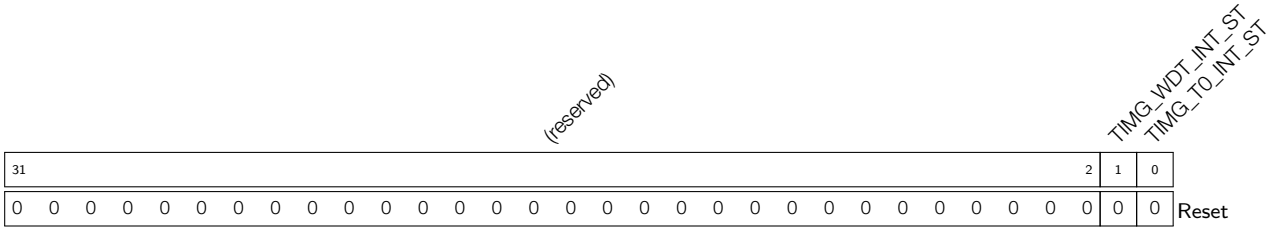
Register 8.22. TIMG_INT_RAW_TIMERS_REG (0x0074)



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)

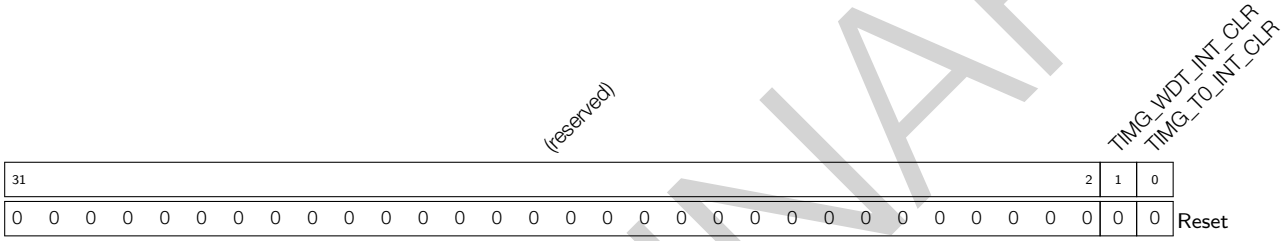
Register 8.23. TIMG_INT_ST_TIMERS_REG (0x0078)



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)

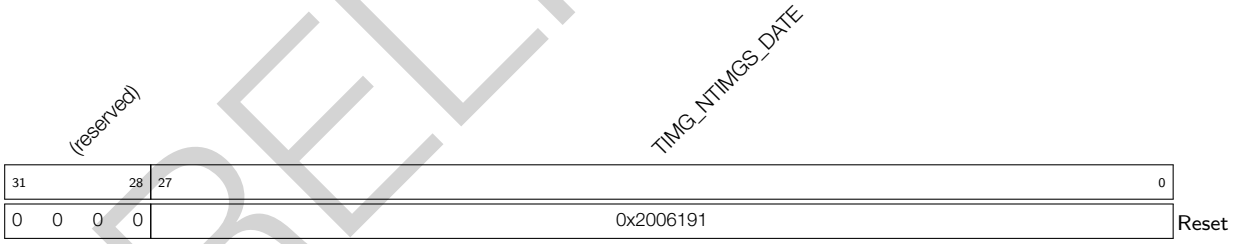
Register 8.24. TIMG_INT_CLR_TIMERS_REG (0x007C)



TIMG_T0_INT_CLR Set this bit to clear the TIMG_T0_INT interrupt. (WT)

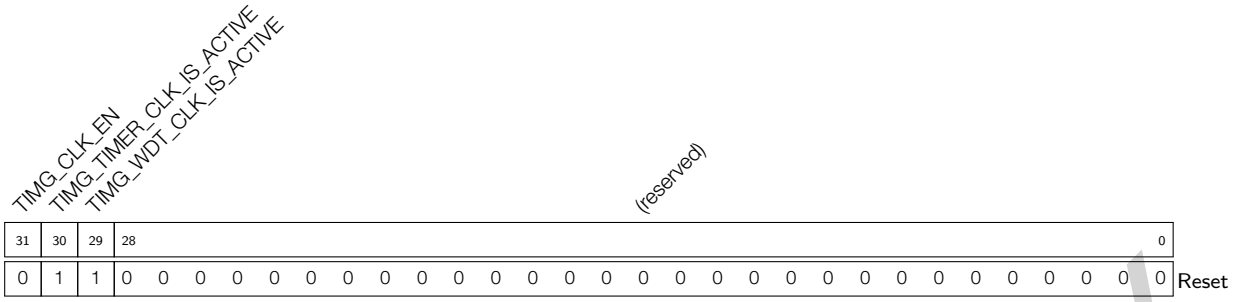
TIMG_WDT_INT_CLR Set this bit to clear the TIMG_WDT_INT interrupt. (WT)

Register 8.25. TIMG_NTIMERS_DATE_REG (0x00F8)



TIMG_NTIMGS_DATE Timer version control register (R/W)

Register 8.26. 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. 0: The clock used by software to read and write registers is on only when there is software operation. 1: The clock used by software to read and write registers is always on. (R/W)

9 SHA Accelerator (SHA)

9.1 Introduction

ESP32-C3 integrates an SHA accelerator, which is a hardware device that speeds up SHA algorithm significantly, compared to SHA algorithm implemented solely in software. The SHA accelerator integrated in ESP32-C3 has two working modes, which are Typical SHA and DMA-SHA.

9.2 Features

The following functionality is supported:

- The following hash algorithms introduced in FIPS PUB 180-4 Spec.
  - SHA-1
  - SHA-224
  - SHA-256
- Two working modes
  - Typical SHA
  - DMA-SHA
- Interleaved function when working in Typical SHA working mode
- Interrupt function when working in DMA-SHA working mode

9.3 Working Modes

The SHA accelerator integrated in ESP32-C3 has two working modes.

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

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

Table 9-1. SHA Accelerator Working Mode

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

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

Table 9-2. SHA Hash Algorithm Selection

Hash Algorithm	SHA_MODE_REG Configuration
SHA-1	0
SHA-224	1
SHA-256	2

Notice:
ESP32-C3's Digital Signature (DS) [to be added later] and HMAC Accelerator (HMAC) [to be added later] modules also call the SHA accelerator. Therefore, users cannot access the SHA accelerator when these modules are working.

9.4 Function Description

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

9.4.1 Preprocessing

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

9.4.1.1 Padding the Message

The SHA accelerator can only process message blocks of 512 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.

9.4.1.2 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.

9.4.1.3 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.

9.4.2 Hash Task Process

After the preprocessing, the ESP32-C3 SHA accelerator starts to hash a message M and generates message digest of different lengths, depending on different hash algorithms. As described above, the ESP32-C3 SHA accelerator supports two working modes, which are Typical SHA and DMA-SHA. The operation process for the SHA accelerator under two working modes is described in the following subsections.

9.4.2.1 Typical SHA Mode Process

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

However, ESP32-C3 SHA also supports optional "interleaved" message digest calculation. Users can insert new calculation (both Typical SHA and DMA-SHA) 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 9-2.
2. Process the current message block [1].
   - Write the message block in registers SHA_M_*n*_REG.
3. Start the SHA accelerator.
   - If this is the first time to execute this step, set the SHA_START_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the initial hash value stored in hardware for a given algorithm configured in Step 1 to start the calculation;

- If this is not the first time to execute this step[2], set the SHA_CONTINUE_REG register to 1 to start the SHA accelerator. In this case, the accelerator uses the hash value stored in the SHA_H_*n*_REG register to start calculation.

4. Check the progress of the current message block.
   - Poll register SHA_BUSY_REG until the content of this register becomes 0, indicating the accelerator has completed the calculation for the current message block and now is in the "idle" status [3].
5. Decide if you have more message blocks to process:
   - If yes, please go back to Step 2.
   - Otherwise, please continue.
6. Obtain the message digest.
   - Read the message digest from registers SHA_H_*n*_REG.

Note:

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

As mentioned above, ESP32-C3 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.

9.4.2.2 DMA-SHA Mode Process

ESP32-C3 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 9-2.
2. Configure the SHA_INT_ENA_REG register to enable or disable interrupt (Set 1 to enable).
3. Configure the number of message blocks.
   - Write the number of message blocks M to the SHA_DMA_BLOCK_NUM_REG register.
4. Start the DMA-SHA calculation.
   - If the current DMA-SHA calculation follows a previous calculation, firstly write the message digest from the previous calculation to registers SHA_H_*n*_REG, then write 1 to register SHA_DMA_CONTINUE_REG to start SHA accelerator;
   - Otherwise, write 1 to register SHA_DMA_START_REG to start the accelerator.
5. Wait till the completion of the DMA-SHA calculation, which happens when:
   - The content of SHA_BUSY_REG register becomes 0, or
   - An SHA interrupt occurs. In this case, please clear interrupt by writing 1 to the SHA_INT_CLEAR_REG register.
6. Obtain the message digest:
   - Read the message digest from registers SHA_H_*n*_REG.

9.4.3 Message Digest

After the hash task completes, the SHA accelerator writes the message digest from the task to registers SHA_H_*n*_REG(*n*: 0~7). The lengths of the generated message digest are different depending on different hash algorithms. For details, see Table 9-3 below:

Table 9-3. 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 9.4.1.2.

9.4.4 Interrupt

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

9.5 Register Summary

The addresses in this section are relative to the SHA accelerator base address provided in Table 3-4 in Chapter 3 *System and Memory*.

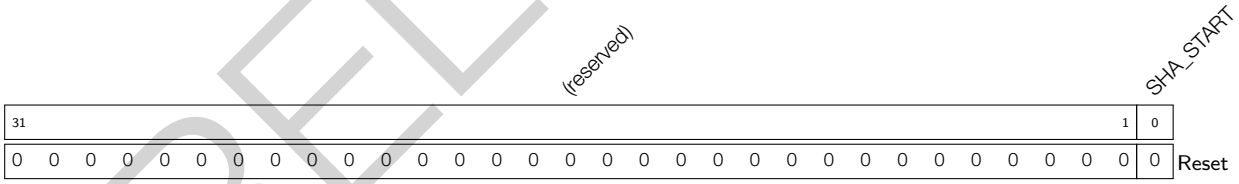
Name	Description	Address	Access
Control/Status registers			
SHA_CONTINUE_REG	Continues SHA operation (only effective in Typical SHA mode)	0x0014	WO
SHA_BUSY_REG	Indicates if SHA Accelerator is busy or not	0x0018	RO
SHA_DMA_START_REG	Starts the SHA accelerator for DMA-SHA operation	0x001C	WO
SHA_START_REG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

Name	Description	Address	Access
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
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

9.6 Registers

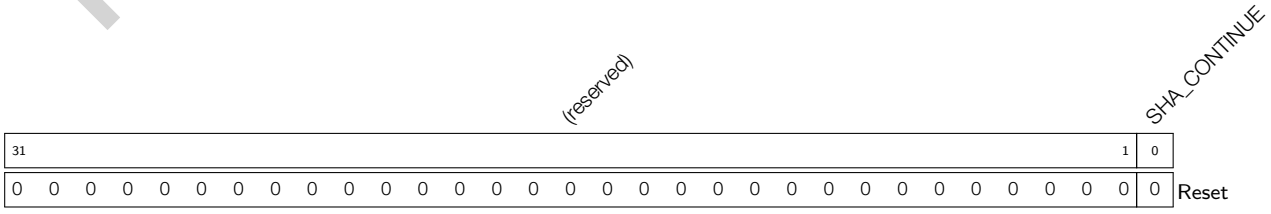
The addresses in this section are relative to the SHA accelerator base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Register 9.1. SHA_START_REG (0x0010)



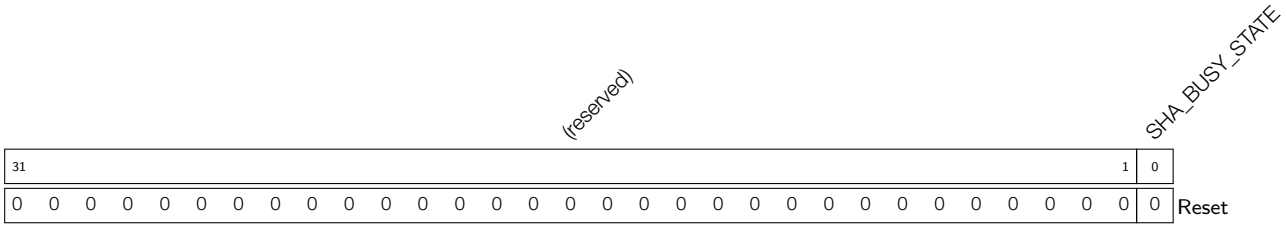
SHA_START Write 1 to start Typical SHA calculation. (WO)

Register 9.2. SHA_CONTINUE_REG (0x0014)



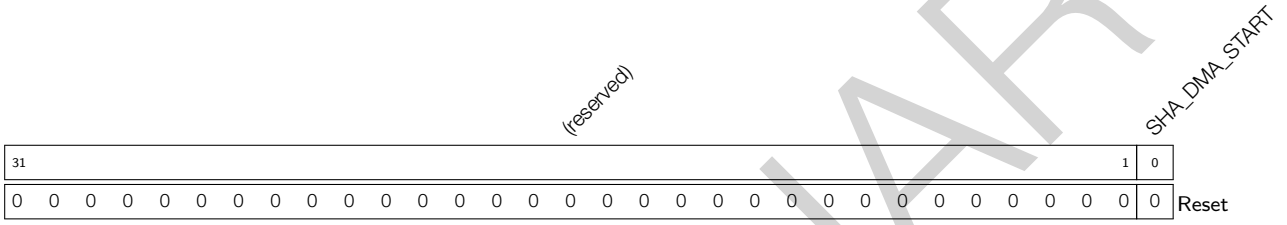
SHA_CONTINUE Write 1 to continue Typical SHA calculation. (WO)

Register 9.3. SHA_BUSY_REG (0x0018)



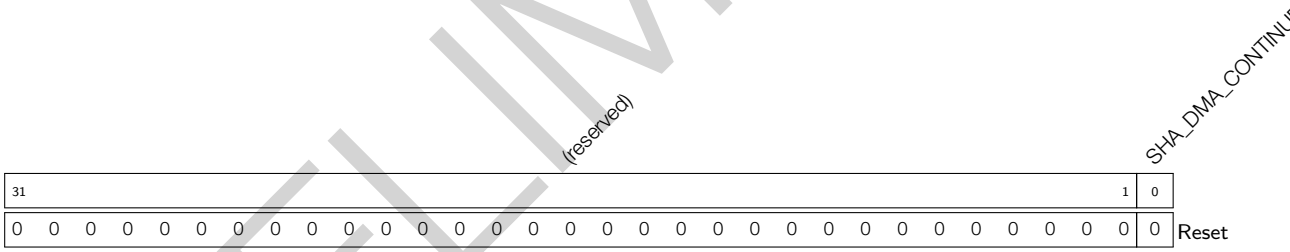
SHA_BUSY_STATE Indicates the states of SHA accelerator. (RO) 1'h0: idle 1'h1: busy

Register 9.4. SHA_DMA_START_REG (0x001C)



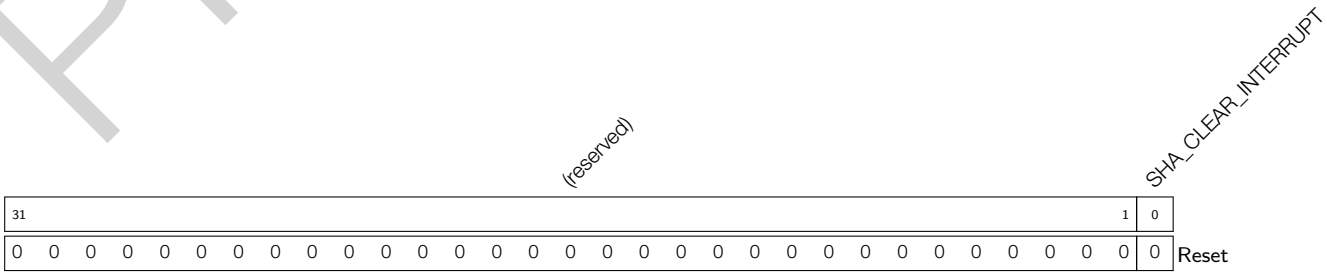
SHA_DMA_START Write 1 to start DMA-SHA calculation. (WO)

Register 9.5. SHA_DMA_CONTINUE_REG (0x0020)



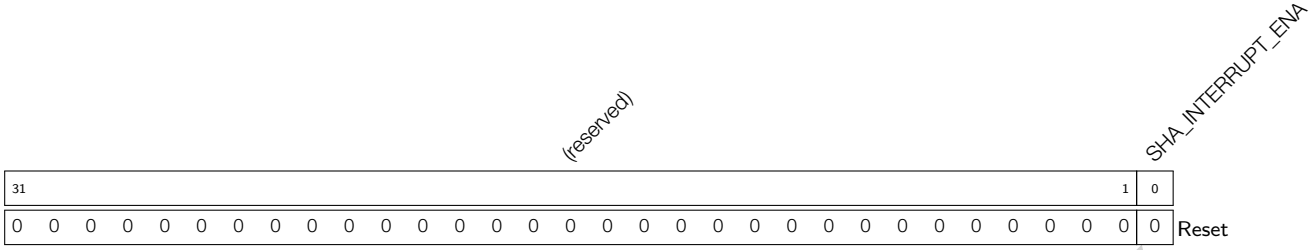
SHA_DMA_CONTINUE Write 1 to continue DMA-SHA calculation. (WO)

Register 9.6. SHA_INT_CLEAR_REG (0x0024)



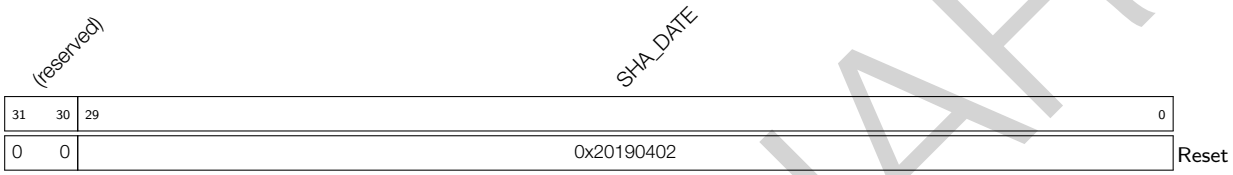
SHA_CLEAR_INTERRUPT Clears DMA-SHA interrupt. (WO)

Register 9.7. SHA_INT_ENA_REG (0x0028)



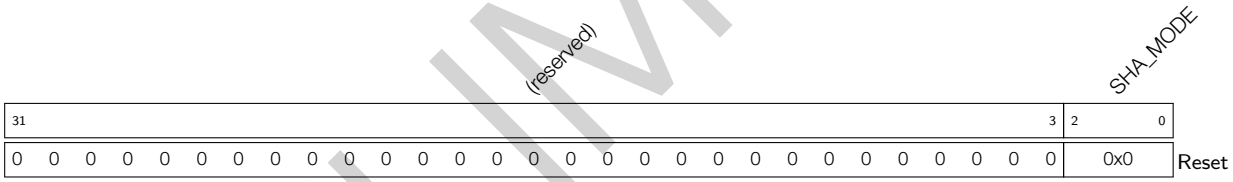
SHA_INTERRUPT_ENA Enables DMA-SHA interrupt. (R/W)

Register 9.8. SHA_DATE_REG (0x002C)



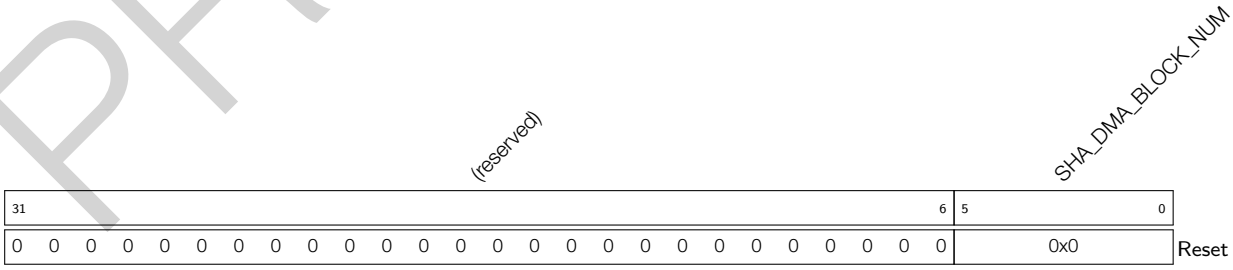
SHA_DATE Version control register. (R/W)

Register 9.9. SHA_MODE_REG (0x0000)



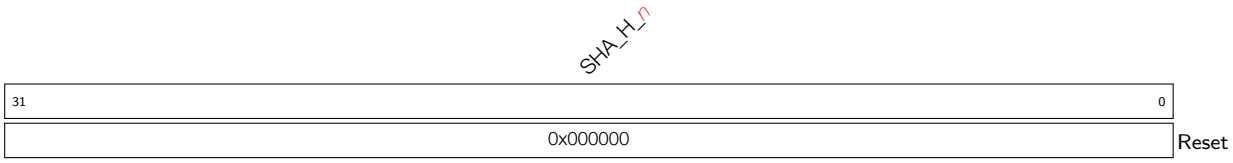
SHA_MODE Defines the SHA algorithm. For details, please see Table 9-2. (R/W)

Register 9.10. SHA_DMA_BLOCK_NUM_REG (0x000C)



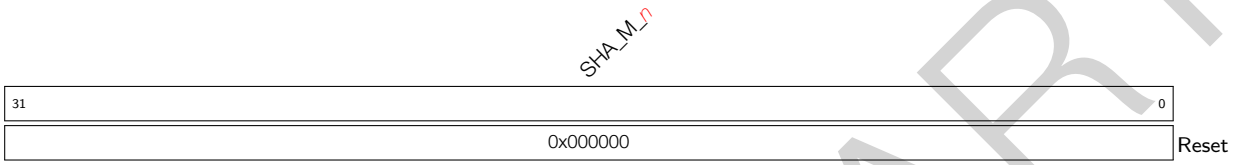
SHA_DMA_BLOCK_NUM Defines the DMA-SHA block number. (R/W)

Register 9.11. 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 9.12. SHA_M_*n*_REG (*n*: 0-15) (0x0080+4n*)**



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

10 AES Accelerator (AES)

10.1 Introduction

ESP32-C3 integrates an Advanced Encryption Standard (AES) Accelerator, which is a hardware device that speeds up AES Algorithm significantly, compared to AES algorithms implemented solely in software. The AES Accelerator integrated in ESP32-C3 has two working modes, which are Typical AES and DMA-AES.

10.2 Features

The following functionality is supported:

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

10.3 AES Working Modes

The AES Accelerator integrated in ESP32-C3 has two working modes, which are Typical AES and DMA-AES.

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

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

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

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

Table 10-1. AES Accelerator Working Mode

AES_DMA_ENABLE_REG	Working Mode
0	Typical AES
1	DMA-AES

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

Table 10-2. Key Length and Encryption/Decryption

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

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

Notice:

ESP32-C3's Digital Signature (DS) [to be added later] module will call the AES accelerator. Therefore, users cannot access the AES accelerator when Digital Signature (DS) [to be added later] module is working.

10.4 Typical AES Working Mode

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

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

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

10.4.1 Key, Plaintext, and Ciphertext

The encryption or decryption key is stored in AES_KEY_*n*_REG, which is a set of eight 32-bit registers.

- For AES-128 encryption/decryption, the 128-bit key is stored in AES_KEY_0_REG ~ AES_KEY_3_REG.
- For AES-256 encryption/decryption, the 256-bit key is stored in AES_KEY_0_REG ~ AES_KEY_7_REG.

The plaintext and ciphertext are stored in AES_TEXT_IN_*m*_REG and AES_TEXT_OUT_*m*_REG, which are two sets of four 32-bit registers.

- For AES-128/AES-256 encryption, the AES_TEXT_IN_*m*_REG registers are initialized with plaintext. Then, the AES Accelerator stores the ciphertext into AES_TEXT_OUT_*m*_REG after operation.
- For AES-128/AES-256 decryption, the AES_TEXT_IN_*m*_REG registers are initialized with ciphertext. Then, the AES Accelerator stores the plaintext into AES_TEXT_OUT_*m*_REG after operation.

10.4.2 Endianness

Text Endianness

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

Table 10-4. Text Endianness Type for Typical AES

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

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

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

Key Endianness

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

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

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

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

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

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

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

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

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

10.4.3 Operation Process

Single Operation

1. Write 0 to the AES_DMA_ENABLE_REG register.
2. Initialize registers AES_MODE_REG, AES_KEY_*n*_REG, AES_TEXT_IN_*m*_REG.
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait till the content of the AES_STATE_REG register becomes 0, which indicates the operation is completed.
5. Read results from the AES_TEXT_OUT_*m*_REG register.

Consecutive Operations

In consecutive operations, primarily the input AES_TEXT_IN_*m*_REG and output AES_TEXT_OUT_*m*_REG registers are being written and read, while the content of AES_DMA_ENABLE_REG, AES_MODE_REG, AES_KEY_*n*_REG is kept unchanged. Therefore, the initialization can be simplified during the consecutive operation.

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

10.5 DMA-AES Working Mode

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

Table 10-7. Block Cipher Mode

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

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

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

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

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

10.5.1 Key, Plaintext, and Ciphertext

Block Operation

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

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

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

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

Table 10-9. TEXT-PADDING

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

10.5.2 Endianness

Under the DMA-AES working mode, the transmission of source data and result data for AES Accelerator is solely controlled by DMA. Therefore, the AES Accelerator cannot control the Endianness of the source data and result

data, but does have requirement on how these data should be stored in memory and on the length of the data.

For example, let us assume DMA needs to write the following data into memory at address 0x0280.

- Data represented in hexadecimal:
  - 0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
- Data Length:
  - Equals to 2 blocks.

Then, this data will be stored in memory as shown in Table 10-10 below.

Table 10-10. Text Endianness for DMA-AES

Address	Byte	Address	Byte	Address	Byte	Address	Byte
0x0280	0x01	0x0281	0x02	0x0282	0x03	0x0283	0x04
0x0284	0x05	0x0285	0x06	0x0286	0x07	0x0287	0x08
0x0288	0x09	0x0289	0x0A	0x028A	0x0B	0x028B	0x0C
0x028C	0x0D	0x028D	0x0E	0x028E	0x0F	0x028F	0x10
0x0290	0x11	0x0291	0x12	0x0292	0x13	0x0293	0x14
0x0294	0x15	0x0295	0x16	0x0296	0x17	0x0297	0x18
0x0298	0x19	0x0299	0x1A	0x029A	0x1B	0x029B	0x1C
0x029C	0x1D	0x029D	0x1E	0x029E	0x1F	0x029F	0x20

10.5.3 Standard Incrementing Function

AES accelerator provides two Standard Incrementing Functions for the CTR block operation, which are INC_{32} and INC_{128} Standard Incrementing Functions. By setting the AES_INC_SEL_REG register to 0 or 1, users can choose the INC_{32} or INC_{128} functions respectively. For details on the Standard Incrementing Function, please see Chapter B.1 The Standard Incrementing Function in NIST SP 800-38A.

10.5.4 Block Number

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

10.5.5 Initialization Vector

AES_IV_MEM is a 16-byte memory, which is only available for AES Accelerator working in block operations. For CBC/OFB/CFB8/CFB128 operations, the AES_IV_MEM memory stores the Initialization Vector (IV). For the CTR operation, the AES_IV_MEM memory stores the Initial Counter Block (ICB).

Both IV and ICB are 128-bit strings, which can be divided into Byte0, Byte1, Byte2 ⋯ Byte15 (from left to right). AES_IV_MEM stores data following the Endianness pattern presented in Table 10-10, i.e. the most significant (i.e., left-most) byte Byte0 is stored at the lowest address while the least significant (i.e., right-most) byte Byte15 at the highest address.

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

10.5.6 Block Operation Process

1. Select one of DMA channels to connect with AES, configure the DMA chained list, and then start DMA. For details, please refer to Chapter 2 *GDMA Controller (GDMA)*.
2. Initialize the AES accelerator-related registers:
   - Write 1 to the AES_DMA_ENABLE_REG register.
   - Configure the AES_INT_ENA_REG register to enable or disable the interrupt function.
   - Initialize registers AES_MODE_REG and AES_KEY_*n*_REG.
   - Select block cipher mode by configuring the AES_BLOCK_MODE_REG register. For details, see Table 10-7.
   - Initialize the AES_BLOCK_NUM_REG register. For details, see Section 10.5.4.
   - Initialize the AES_INC_SEL_REG register (only needed when AES Accelerator is working under CTR block operation).
   - Initialize the AES_IV_MEM memory (This is always needed except for ECB block operation).
3. Start operation by writing 1 to the AES_TRIGGER_REG register.
4. Wait for the completion of computation, which happens when the content of AES_STATE_REG becomes 2 or the AES interrupt occurs.
5. Check if DMA completes data transmission from AES to memory. At this time, DMA had already written the result data in memory, which can be accessed directly. For details on DMA, please refer to Chapter 2 *GDMA Controller (GDMA)*.
6. Clear interrupt by writing 1 to the AES_INT_CLR_REG register, if any AES interrupt occurred during the computation.
7. Release the AES Accelerator by writing 0 to the AES_DMA_EXIT_REG register. After this, the content of the AES_STATE_REG register becomes 0. Note that, you can release DMA earlier, but only after Step 4 is completed.

10.6 Memory Summary

The addresses in this section are relative to the AES accelerator base address provided in Table 3-4 in Chapter 3 *System and Memory*.

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

10.7 Register Summary

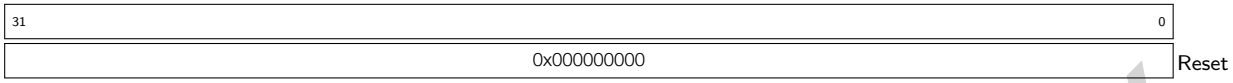
The addresses in this section are relative to the AES accelerator base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Name	Description	Address	Access
Key Registers			
AES_KEY_0_REG	AES key data register 0	0x0000	R/W
AES_KEY_1_REG	AES key data register 1	0x0004	R/W
AES_KEY_2_REG	AES key data register 2	0x0008	R/W
AES_KEY_3_REG	AES key data register 3	0x000C	R/W
AES_KEY_4_REG	AES key data register 4	0x0010	R/W
AES_KEY_5_REG	AES key data register 5	0x0014	R/W
AES_KEY_6_REG	AES key data register 6	0x0018	R/W
AES_KEY_7_REG	AES key data register 7	0x001C	R/W
TEXT_IN Registers			
AES_TEXT_IN_0_REG	Source text data register 0	0x0020	R/W
AES_TEXT_IN_1_REG	Source text data register 1	0x0024	R/W
AES_TEXT_IN_2_REG	Source text data register 2	0x0028	R/W
AES_TEXT_IN_3_REG	Source text data register 3	0x002C	R/W
TEXT_OUT Registers			
AES_TEXT_OUT_0_REG	Result text data register 0	0x0030	RO
AES_TEXT_OUT_1_REG	Result text data register 1	0x0034	RO
AES_TEXT_OUT_2_REG	Result text data register 2	0x0038	RO
AES_TEXT_OUT_3_REG	Result text data register 3	0x003C	RO
Configuration Registers			
AES_MODE_REG	Defines key length and encryption / decryption	0x0040	R/W
AES_DMA_ENABLE_REG	Selects the working mode of the AES accelerator	0x0090	R/W
AES_BLOCK_MODE_REG	Defines the block cipher mode	0x0094	R/W
AES_BLOCK_NUM_REG	Block number configuration register	0x0098	R/W
AES_INC_SEL_REG	Standard incrementing function register	0x009C	R/W
Controlling / Status Registers			
AES_TRIGGER_REG	Operation start controlling register	0x0048	WO
AES_STATE_REG	Operation status register	0x004C	RO
AES_DMA_EXIT_REG	Operation exit controlling register	0x00B8	WO
Interruption Registers			
AES_INT_CLR_REG	DMA-AES interrupt clear register	0x00AC	WO
AES_INT_ENA_REG	DMA-AES interrupt enable register	0x00B0	R/W

10.8 Registers

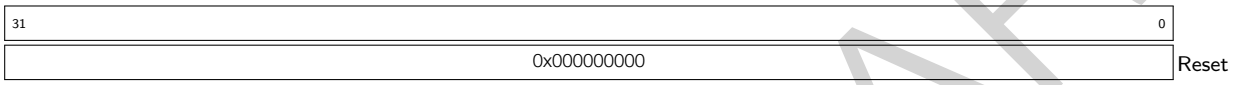
The addresses in this section are relative to the AES accelerator base address provided in Table 3-4 in Chapter 3 *System and Memory*.

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



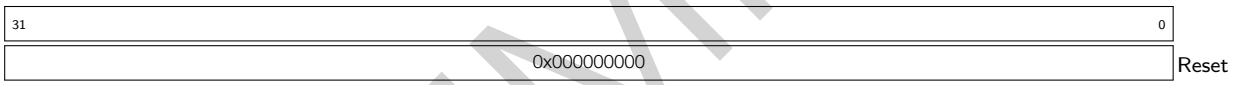
AES_KEY_*n*_REG (*n*: 0-7) Stores AES key data. (R/W)

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



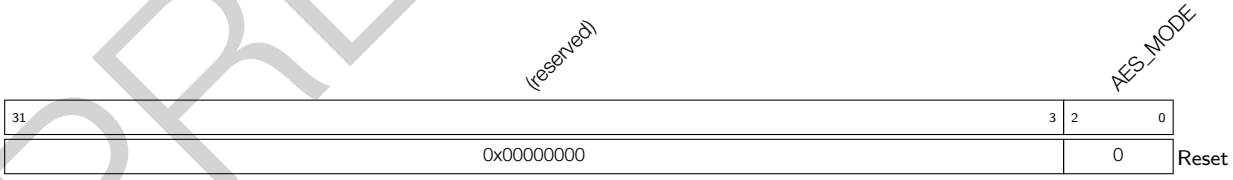
AES_TEXT_IN_*m*_REG (*m*: 0-3) Stores the source text data when the AES Accelerator operates in the Typical AES working mode. (R/W)

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



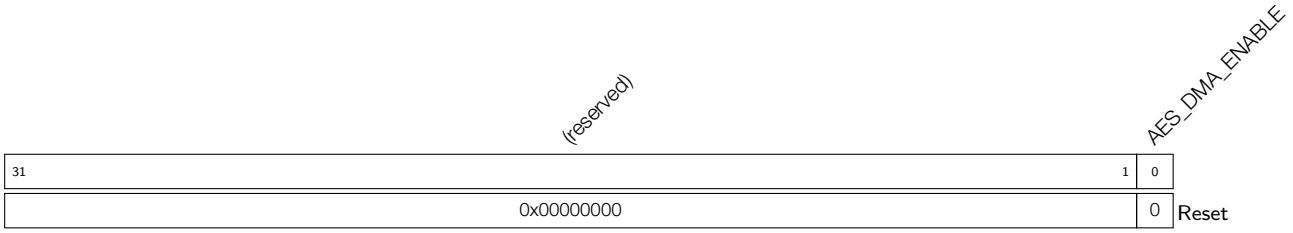
AES_TEXT_OUT_*m*_REG (*m*: 0-3) Stores the result text data when the AES Accelerator operates in the Typical AES working mode. (RO)

Register 10.4. AES_MODE_REG (0x0040)



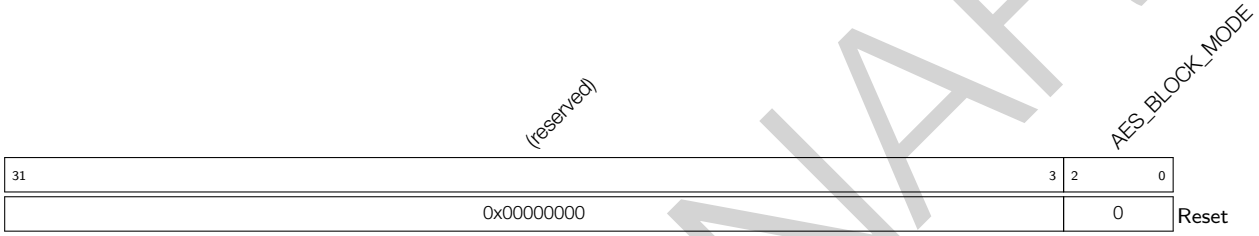
AES_MODE Defines the key length and encryption / decryption of the AES Accelerator. For details, see Table 10-2. (R/W)

Register 10.5. AES_DMA_ENABLE_REG (0x0090)



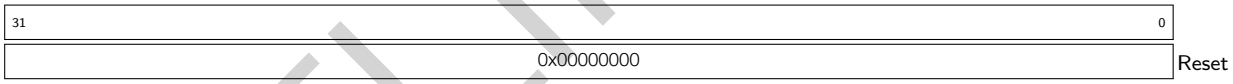
AES_DMA_ENABLE Defines the working mode of the AES Accelerator. 0: Typical AES, 1: DMA-AES. For details, see Table 10-1. (R/W)

Register 10.6. AES_BLOCK_MODE_REG (0x0094)



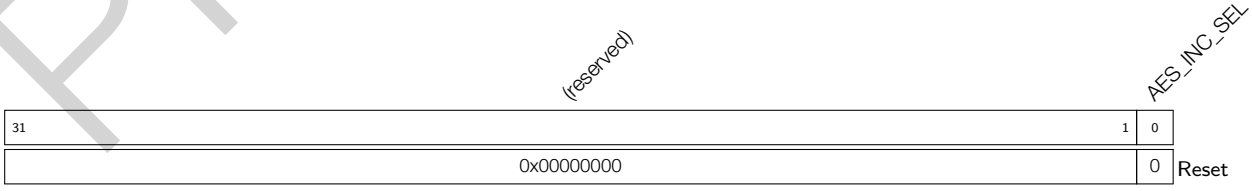
AES_BLOCK_MODE Defines the block cipher mode of the AES Accelerator operating under the DMA-AES working mode. For details, see Table 10-7. (R/W)

Register 10.7. AES_BLOCK_NUM_REG (0x0098)



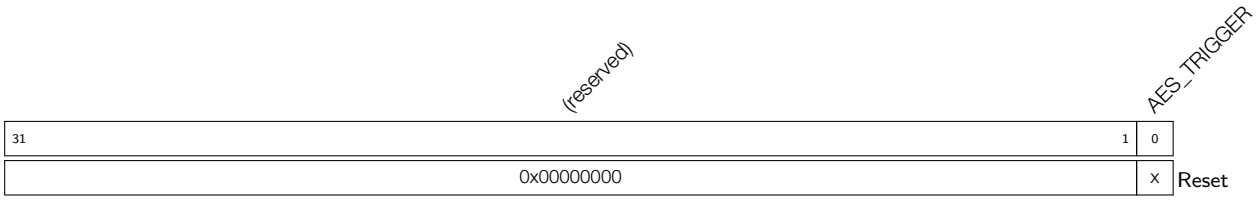
AES_BLOCK_NUM Stores the Block Number of plaintext or ciphertext when the AES Accelerator operates under the DMA-AES working mode. For details, see Section 10.5.4. (R/W)

Register 10.8. AES_INC_SEL_REG (0x009C)



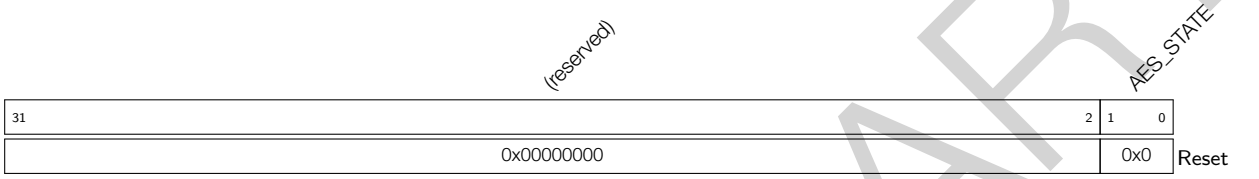
AES_INC_SEL Defines the Standard Incrementing Function for CTR block operation. Set this bit to 0 or 1 to choose INC_{32} or INC_{128}. (R/W)

Register 10.9. AES_TRIGGER_REG (0x0048)



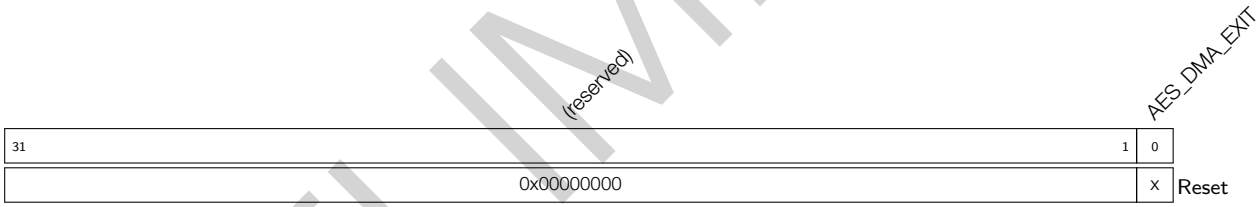
AES_TRIGGER Set this bit to 1 to start AES operation. (WO)

Register 10.10. AES_STATE_REG (0x004C)



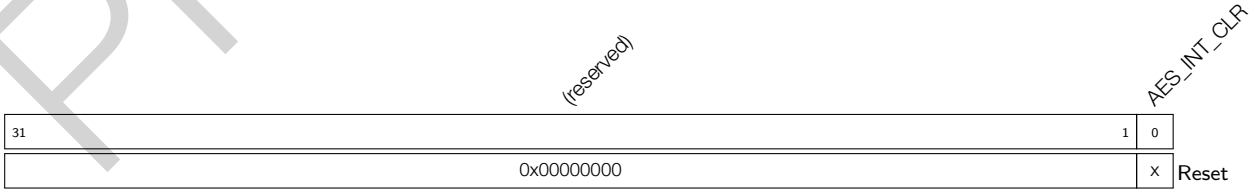
AES_STATE Stores the working status of the AES Accelerator. For details, see Table 10-3 for Typical AES working mode and Table 10-8 for DMA AES working mode. (RO)

Register 10.11. AES_DMA_EXIT_REG (0x00B8)



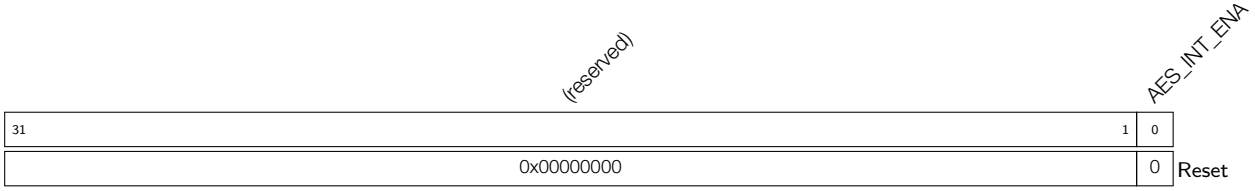
AES_DMA_EXIT Set this bit to 1 to exit AES operation. This register is only effective for DMA-AES operation. (WO)

Register 10.12. AES_INT_CLR_REG (0x00AC)



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

Register 10.13. AES_INT_ENA_REG (0x00B0)



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

11 RSA Accelerator (RSA)

11.1 Introduction

The RSA Accelerator provides hardware support for high precision computation used in various RSA asymmetric cipher algorithms by significantly reducing their software complexity. Compared with RSA algorithms implemented solely in software, this hardware accelerator can speed up RSA algorithms significantly. Besides, the RSA Accelerator also supports operands of different lengths, which provides more flexibility during the computation.

11.2 Features

The following functionality is supported:

- Large-number modular exponentiation with two optional acceleration options
- Large-number modular multiplication
- Large-number multiplication
- Operands of different lengths
- Interrupt on completion of computation

11.3 Functional Description

The RSA Accelerator is activated by setting the SYSTEM_CRYPTO_RSA_CLK_EN bit in the SYSTEM_PERIP_CLK
_EN1_REG register and clearing the SYSTEM_RSA_MEM_PD bit in the SYSTEM_RSA_PD_CTRL_REG register. This releases the RSA Accelerator from reset.

The RSA Accelerator is only available after the RSA-related memories are initialized. The content of the RSA_CLEAN
_REG register is 0 during initialization and will become 1 after the initialization is done. Therefore, it is advised to wait until RSA_CLEAN_REG becomes 1 before using the RSA Accelerator.

The RSA_INTERRUPT_ENA_REG register is used to control the interrupt triggered on completion of computation. Write 1 or 0 to this register to enable or disable interrupt. By default, the interrupt function of the RSA Accelerator is enabled.

Notice:

ESP32-C3's Digital Signature (DS) [to be added later] module also calls the RSA accelerator. Therefore, users cannot access the RSA accelerator when Digital Signature (DS) [to be added later] is working.

11.3.1 Large Number Modular Exponentiation

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

RSA Accelerator supports operands of length $N = 32 \times x$, where $x \in \{1, 2, 3, \ldots, 96\}$. The bit lengths of arguments Z, X, Y, M, and $\overline{r}$ can be arbitrary N, but all numbers in a calculation must be of the same length. The bit length of M' must be 32.

To represent the numbers used as operands, let us define a base-b positional notation, as follows:

$$b = 2^{32}$$

Using this notation, each number is represented by a sequence of base-b digits:

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1} Z_{n-2} \cdots Z_0)_b$$
$$X = (X_{n-1} X_{n-2} \cdots X_0)_b$$
$$Y = (Y_{n-1} Y_{n-2} \cdots Y_0)_b$$
$$M = (M_{n-1} M_{n-2} \cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1} \overline{r}_{n-2} \cdots \overline{r}_0)_b$$

Each of the n values in $Z_{n-1} \cdots Z_0$, $X_{n-1} \cdots X_0$, $Y_{n-1} \cdots Y_0$, $M_{n-1} \cdots M_0$, $\overline{r}_{n-1} \cdots \overline{r}_0$ represents one base-b digit (a 32-bit word).

Z_{n-1}, X_{n-1}, Y_{n-1}, M_{n-1} and $\overline{r}_{n-1}$ are the most significant bits of Z, X, Y, M, while Z_0, X_0, Y_0, M_0 and $\overline{r}_0$ are the least significant bits.

If we define $R = b^n$, the additional arguments can be calculated as $\overline{r} = R^2 \bmod M$.

The following equation in the form compatible with the extended binary GCD algorithm can be written as

$$M^{-1} \times M + 1 = R \times R^{-1}$$
$$M' = M^{-1} \bmod b$$

Large-number modular exponentiation can be implemented as follows:

1. Write 1 or 0 to the RSA_INTERRUPT_ENA_REG register to enable or disable the interrupt function.
2. Configure relevant registers:
   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.
   (b) Write M' to the RSA_M_PRIME_REG register.
   (c) Configure registers related to the acceleration options, which are described later in Section 11.3.4.
3. Write X_i, Y_i, M_i and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 96 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.
4. Write 1 to the RSA_MODEXP_START_REG register to start computation.

5. Wait for the completion of computation, which happens when the content of RSA_IDLE_REG becomes 1 or the RSA interrupt occurs.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n-1\}$ from RSA_Z_MEM.
7. Write 1 to RSA_CLEAR_INTERRUPT_REG to clear the interrupt, if you have enabled the interrupt function.

After the computation, the RSA_MODE_REG register, memory blocks RSA_Y_MEM and RSA_M_MEM, as well as the RSA_M_PRIME_REG remain unchanged. However, X_i in RSA_X_MEM and $\overline{r}_i$ in RSA_Z_MEM computation are overwritten, and only these overwritten memory blocks need to be re-initialized before starting another computation.

11.3.2 Large Number Modular Multiplication

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

The RSA Accelerator supports large-number modular multiplication with operands of 96 different lengths.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INTERRUPT_ENA_REG register to enable or disable the interrupt function.
2. Configure relevant registers:
   (a) Write $(\frac{N}{32} - 1)$ to the RSA_MODE_REG register.
   (b) Write M' to the RSA_M_PRIME_REG register.
3. Write X_i, Y_i, M_i, and $\overline{r}_i$ for $i \in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 96 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.
4. Write 1 to the RSA_MODMULT_START_REG register.
5. Wait for the completion of computation, which happens when the content of RSA_IDLE_REG becomes 1 or the RSA interrupt occurs.
6. Read the result Z_i for $i \in \{0, 1, \ldots, n-1\}$ from RSA_Z_MEM.
7. Write 1 to RSA_CLEAR_INTERRUPT_REG to clear the interrupt, if you have enabled the interrupt function.

After the computation, the length of operands in RSA_MODE_REG, the X_i in memory RSA_X_MEM, the Y_i in memory RSA_Y_MEM, the M_i in memory RSA_M_MEM, and the M' in memory RSA_M_PRIME_REG remain unchanged. However, the $\overline{r}_i$ in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

11.3.3 Large Number Multiplication

Large-number multiplication performs $Z = X \times Y$. The length of result Z is twice that of operand X and operand Y. Therefore, the RSA Accelerator only supports Large Number Multiplication with operand length $N = 32 \times x$, where $x \in \{1, 2, 3, \ldots, 48\}$. The length $\hat{N}$ of result Z is $2 \times N$.

The computation can be executed as follows:

1. Write 1 or 0 to the RSA_INTERRUPT_ENA_REG register to enable or disable the interrupt function.
2. Write $(\frac{\hat{N}}{32} - 1)$, i.e. $(\frac{N}{16} - 1)$ to the RSA_MODE_REG register.
3. Write X_i and Y_i for $\in \{0, 1, \ldots, n-1\}$ to memory blocks RSA_X_MEM and RSA_Z_MEM. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address. n is $\frac{N}{32}$.

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

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.
4. Write 1 to the RSA_MULT_START_REG register.
5. Wait for the completion of computation, which happens when the content of RSA_IDLE_REG becomes 1 or the RSA interrupt occurs.
6. Read the result Z_i for $i \in \{0, 1, \ldots, \hat{n}-1\}$ from the RSA_Z_MEM register. $\hat{n}$ is $2 \times n$.
7. Write 1 to RSA_CLEAR_INTERRUPT_REG to clear the interrupt, if you have enabled the interrupt function.

After the computation, the length of operands in RSA_MODE_REG and the X_i in memory RSA_X_MEM remain unchanged. However, the Y_i in memory RSA_Z_MEM has already been overwritten, and only this overwritten memory block needs to be re-initialized before starting another computation.

11.3.4 Options for Acceleration

The ESP32-C3 RSA accelerator also provides SEARCH and CONSTANT_TIME options that can be configured to accelerate the large-number modular exponentiation. By default, both options are configured for no acceleration. Users can choose to use one or two of these options to accelerate the computation.

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

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

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

where,

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

When either of these two options is configured for acceleration:

- SEARCH Option (Configuring RSA_SEARCH_ENABLE to 1 for acceleration)
  - The accelerator ignores the bit positions of $\widetilde{Y}_i$, where $i > \alpha$. Search position α is set by configuring the RSA_SEARCH_POS_REG register. The maximum value of α is N-1, which leads to the same result when this option is not used for acceleration. The best acceleration performance can be achieved by setting α to t, in which case, all the $\widetilde{Y}_{N-1}$, $\widetilde{Y}_{N-2}$, ..., $\widetilde{Y}_{t+1}$ of 0s are ignored during the calculation. Note that if you set α to be less than t, then the result of the modular exponentiation $Z = X^Y \bmod M$ will be incorrect.
- CONSTANT_TIME Option (Configuring RSA_CONSTANT_TIME_REG to 0 for acceleration)
  - The accelerator speeds up the calculation by simplifying the calculation concerning the 0 bits of Y. Therefore, the higher the proportion of bits 0 against bits 1, the better the acceleration performance is.

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

Table 11-1. Acceleration Performance

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

It's obvious that:

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

11.4 Memory Summary

The addresses in this section are relative to the RSA accelerator base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Table 11-2. RSA Accelerator Memory Blocks

Name	Description	Size (byte)	Starting Address	Ending Address	Access
RSA_M_MEM	Memory M	384	0x0000	0x017F	R/W
RSA_Z_MEM	Memory Z	384	0x0200	0x037F	R/W
RSA_Y_MEM	Memory Y	384	0x0400	0x057F	R/W
RSA_X_MEM	Memory X	384	0x0600	0x077F	R/W

11.5 Register Summary

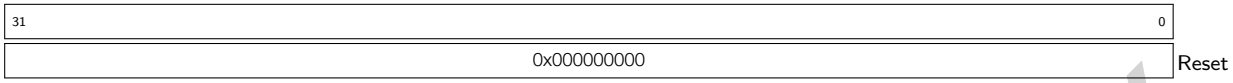
The addresses in this section are relative to the RSA accelerator base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Name	Description	Address	Access
Configuration Registers			
RSA_M_PRIME_REG	Register to store M'	0x0800	R/W
RSA_MODE_REG	RSA length mode	0x0804	R/W
RSA_CONSTANT_TIME_REG	The constant_time option	0x0820	R/W
RSA_SEARCH_ENABLE_REG	The search option	0x0824	R/W
RSA_SEARCH_POS_REG	The search position	0x0828	R/W
Status/Control Registers			
RSA_CLEAN_REG	RSA clean register	0x0808	RO
RSA_MODEXP_START_REG	Modular exponentiation starting bit	0x080C	WO
RSA_MODMULT_START_REG	Modular multiplication starting bit	0x0810	WO
RSA_MULT_START_REG	Normal multiplication starting bit	0x0814	WO
RSA_IDLE_REG	RSA idle register	0x0818	RO
Interrupt Registers			
RSA_CLEAR_INTERRUPT_REG	RSA clear interrupt register	0x081C	WO
RSA_INTERRUPT_ENA_REG	RSA interrupt enable register	0x082C	R/W
Version Register			
RSA_DATE_REG	Version control register	0x0830	R/W

11.6 Registers

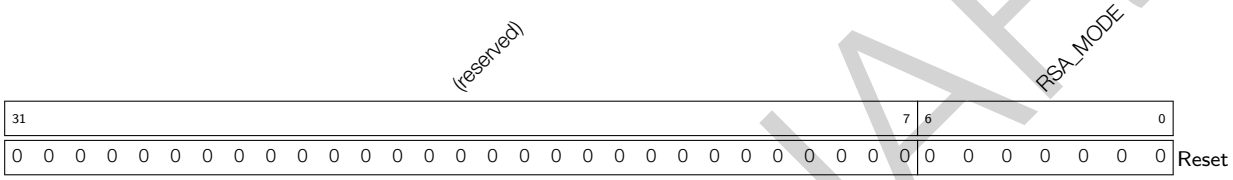
The addresses in this section are relative to the RSA accelerator base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Register 11.1. RSA_M_PRIME_REG (0x0800)



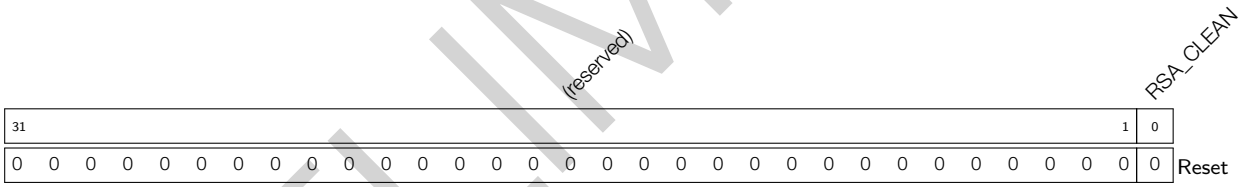
RSA_M_PRIME_REG Stores M'.(R/W)

Register 11.2. RSA_MODE_REG (0x0804)



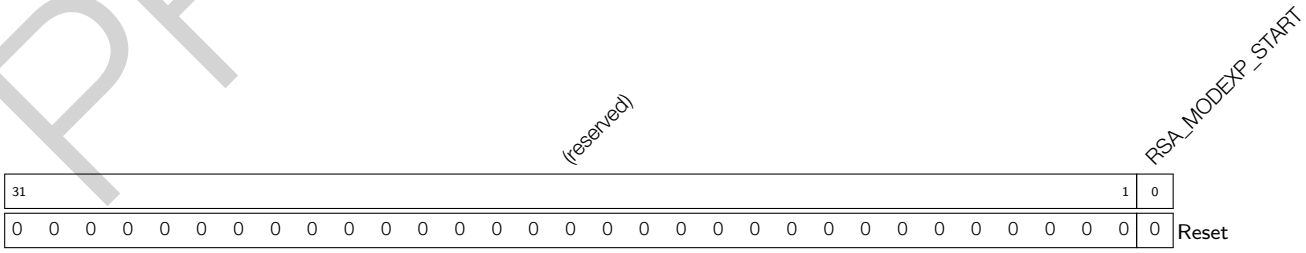
RSA_MODE Stores the mode of modular exponentiation. (R/W)

Register 11.3. RSA_CLEAN_REG (0x0808)



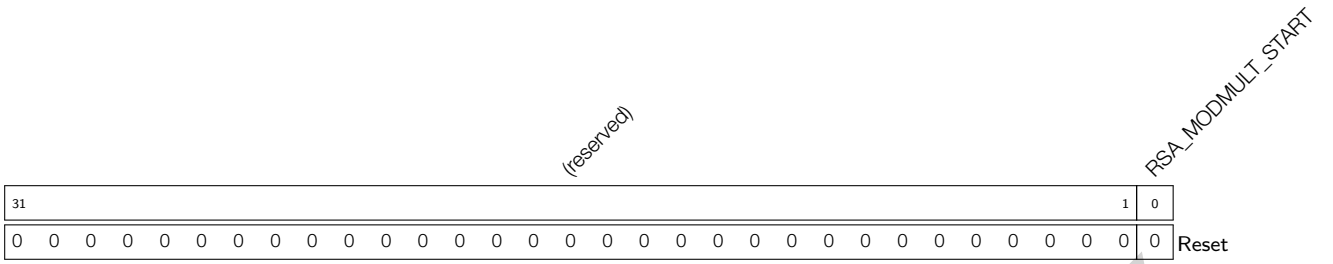
RSA_CLEAN The content of this bit is 1 when memories complete initialization. (RO)

Register 11.4. RSA_MODEXP_START_REG (0x080C)



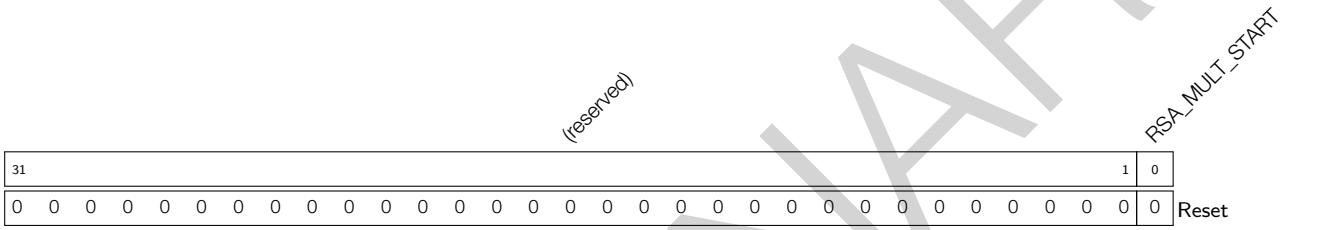
RSA_MODEXP_START Set this bit to 1 to start the modular exponentiation. (WO)

Register 11.5. RSA_MODMULT_START_REG (0x0810)



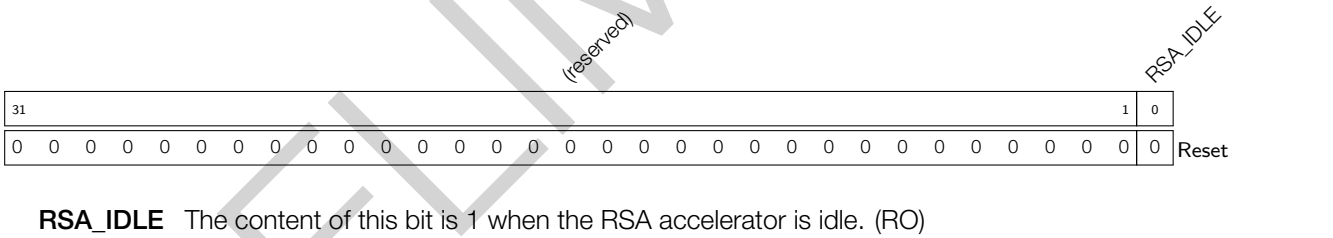
RSA_MODMULT_START Set this bit to 1 to start the modular multiplication. (WO)

Register 11.6. RSA_MULT_START_REG (0x0814)



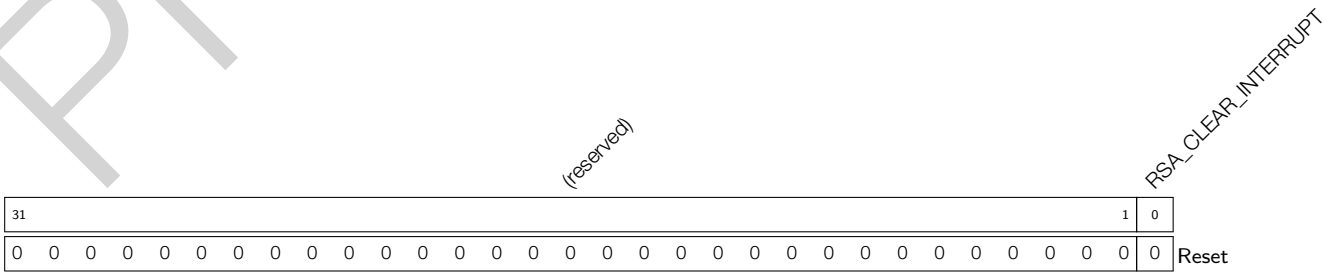
RSA_MULT_START Set this bit to 1 to start the multiplication. (WO)

Register 11.7. RSA_IDLE_REG (0x0818)



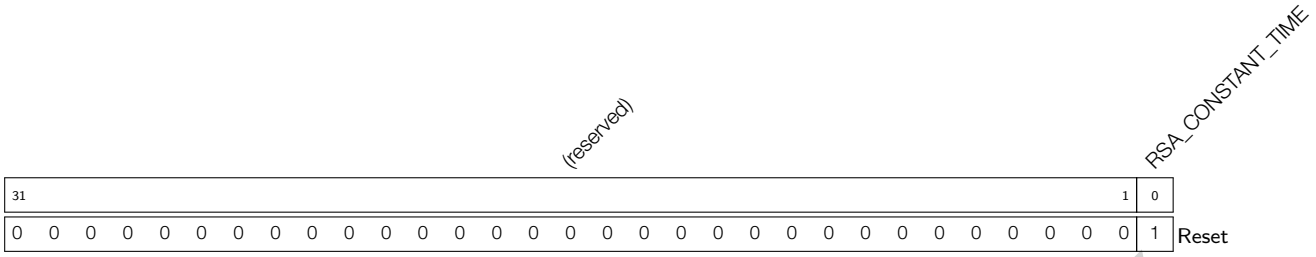
RSA_IDLE The content of this bit is 1 when the RSA accelerator is idle. (RO)

Register 11.8. RSA_CLEAR_INTERRUPT_REG (0x081C)



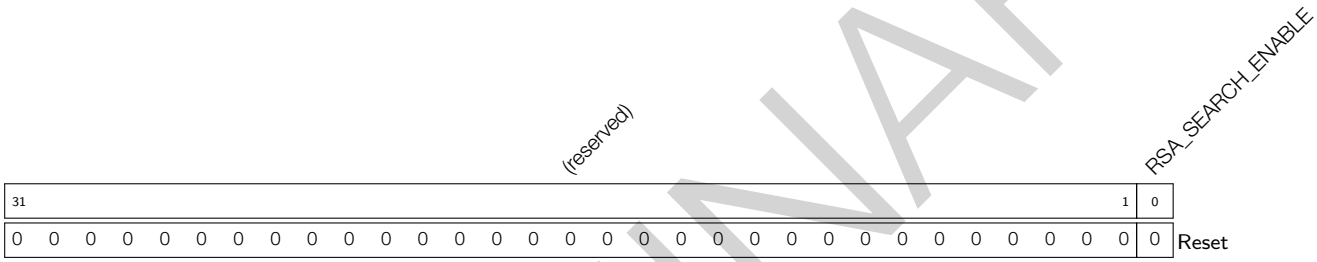
RSA_CLEAR_INTERRUPT Set this bit to 1 to clear the RSA interrupts. (WO)

Register 11.9. RSA_CONSTANT_TIME_REG (0x0820)



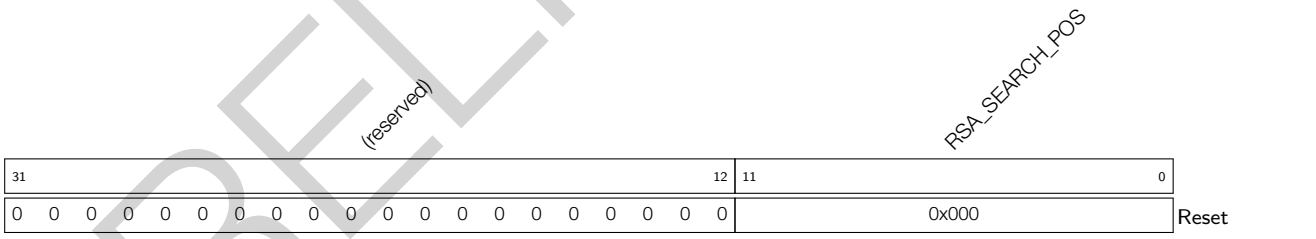
RSA_CONSTANT_TIME_REG Controls the constant_time option. 0: acceleration. 1: no acceleration (by default). (R/W)

Register 11.10. RSA_SEARCH_ENABLE_REG (0x0824)



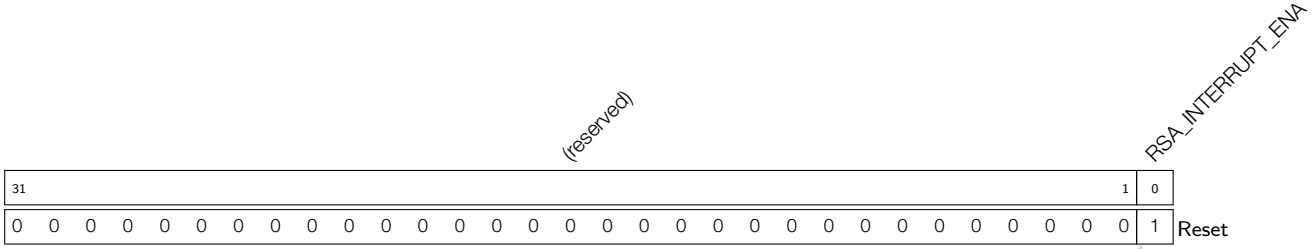
RSA_SEARCH_ENABLE Controls the search option. 0: no acceleration (by default). 1: acceleration. (R/W)

Register 11.11. RSA_SEARCH_POS_REG (0x0828)



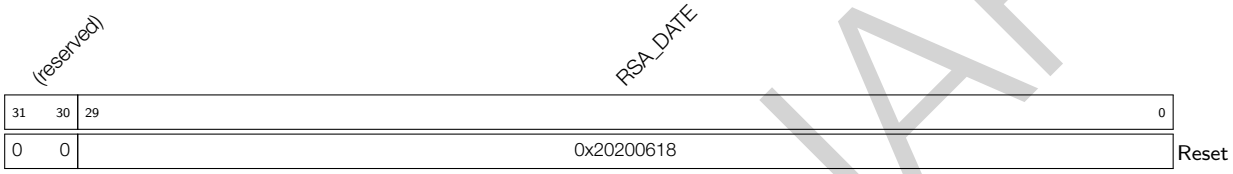
RSA_SEARCH_POS Is used to configure the starting address when the acceleration option of search is used. (R/W)

Register 11.12. RSA_INTERRUPT_ENA_REG (0x082C)



RSA_INTERRUPT_ENA Set this bit to 1 to enable the RSA interrupt. This option is enabled by default. (R/W)

Register 11.13. RSA_DATE_REG (0x0830)



RSA_DATE Version control register. (R/W)

12 Random Number Generator (RNG)

12.1 Introduction

The ESP32-C3 contains a true random number generator, which generates 32-bit random numbers that can be used for cryptographical operations, among other things.

12.2 Features

The random number generator in ESP32-C3 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.

12.3 Functional Description

Every 32-bit value that the system reads from the RNG_DATA_REG register of the random number generator is a true random number. These true random numbers are generated based on the **thermal noise** in the system and the **asynchronous clock mismatch**.

- **Thermal noise** comes from the high-speed ADC or SAR ADC or both. Whenever the high-speed ADC or SAR ADC is enabled, bit streams will be generated and fed into the random number generator through an XOR logic gate as random seeds.
- RTC20M_CLK is an **asynchronous clock** source and it increases the RNG entropy by introducing circuit metastability.

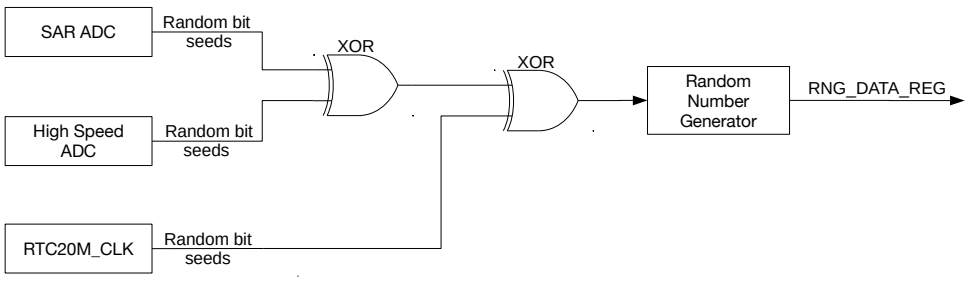


Figure 12-1. Noise Source

When there is noise coming from the SAR ADC, the random number generator is fed with a 2-bit entropy in one clock cycle of RTC20M_CLK (20 MHz), which is generated from an internal RC oscillator (see Chapter 6 *Reset and Clock* for details). Thus, it is advisable to read the RNG_DATA_REG register at a maximum rate of 1 MHz to obtain the maximum entropy.

When there is noise coming from the high-speed ADC, the random number generator is fed with a 2-bit entropy in one APB clock cycle, which is normally 80 MHz. Thus, it is advisable to read the RNG_DATA_REG register at a maximum rate of 5 MHz to obtain the maximum entropy.

A data sample of 2 GB, which is read from the random number generator at a rate of 5 MHz with only the high-speed ADC being enabled, has been tested using the Dieharder Random Number Testsuite (version 3.31.1). The sample passed all tests.

12.4 Programming Procedure

When using the random number generator, make sure at least either the SAR ADC, high-speed ADC[1], or RTC20M_CLK[2] 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 11 *On-Chip Sensors and Analog Signal Processing [to be added later]*.
- High-speed ADC is enabled automatically when the Wi-Fi or Bluetooth modules is enabled.
- RTC20M_CLK is enabled by setting the RTC_CNTL_DIG_CLK20M_EN bit in the RTC_CNTL_CLK_CONF_REG register.

Note:

1. Note that, when the Wi-Fi module is enabled, the value read from the high-speed ADC can be saturated in some extreme cases, which lowers the entropy. Thus, it is advisable to also enable the SAR ADC as the noise source for the random number generator for such cases.
2. Enabling RTC20M_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 12.3 above.

12.5 Register Summary

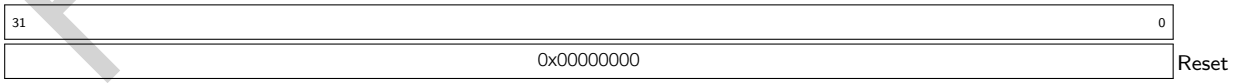
The address in the following table is relative to the random number generator base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Name	Description	Address	Access
RNG_DATA_REG	Random number data	0x00B0	RO

12.6 Register

The address in this section is relative to the random number generator base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Register 12.1. RNG_DATA_REG (0x00B0)



RNG_DATA Random number source. (RO)

13 UART Controller (UART)

13.1 Overview

In embedded system applications, data is required to be transferred in a simple way with minimal system resources. This can be achieved by a Universal Asynchronous Receiver/Transmitter (UART), which flexibly exchanges data with other peripheral devices in full-duplex mode. ESP32-C3 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 ESP32-C3 support various lengths of data bits and stop bits. These controllers also support software and hardware flow control as well as GDMA for seamless high-speed data transfer. This allows developers to use multiple UART ports at minimal software cost.

13.2 Features

Each UART controller has the following features:

- Three clock sources that can be divided
- Programmable baud rate
- 512 x 8-bit RAM shared by TX FIFOs and RX FIFOs of the two UART controllers
- Full-duplex asynchronous communication
- Automatic baud rate detection of input signals
- Data bits ranging from 5 to 8
- Stop bits whose length can be 1, 1.5, 2 or 3 bits
- Parity bit
- Special character AT_CMD detection
- RS485 protocol
- IrDA protocol
- High-speed data communication using GDMA
- UART as wake-up source
- Software and hardware flow control

13.3 UART Structure

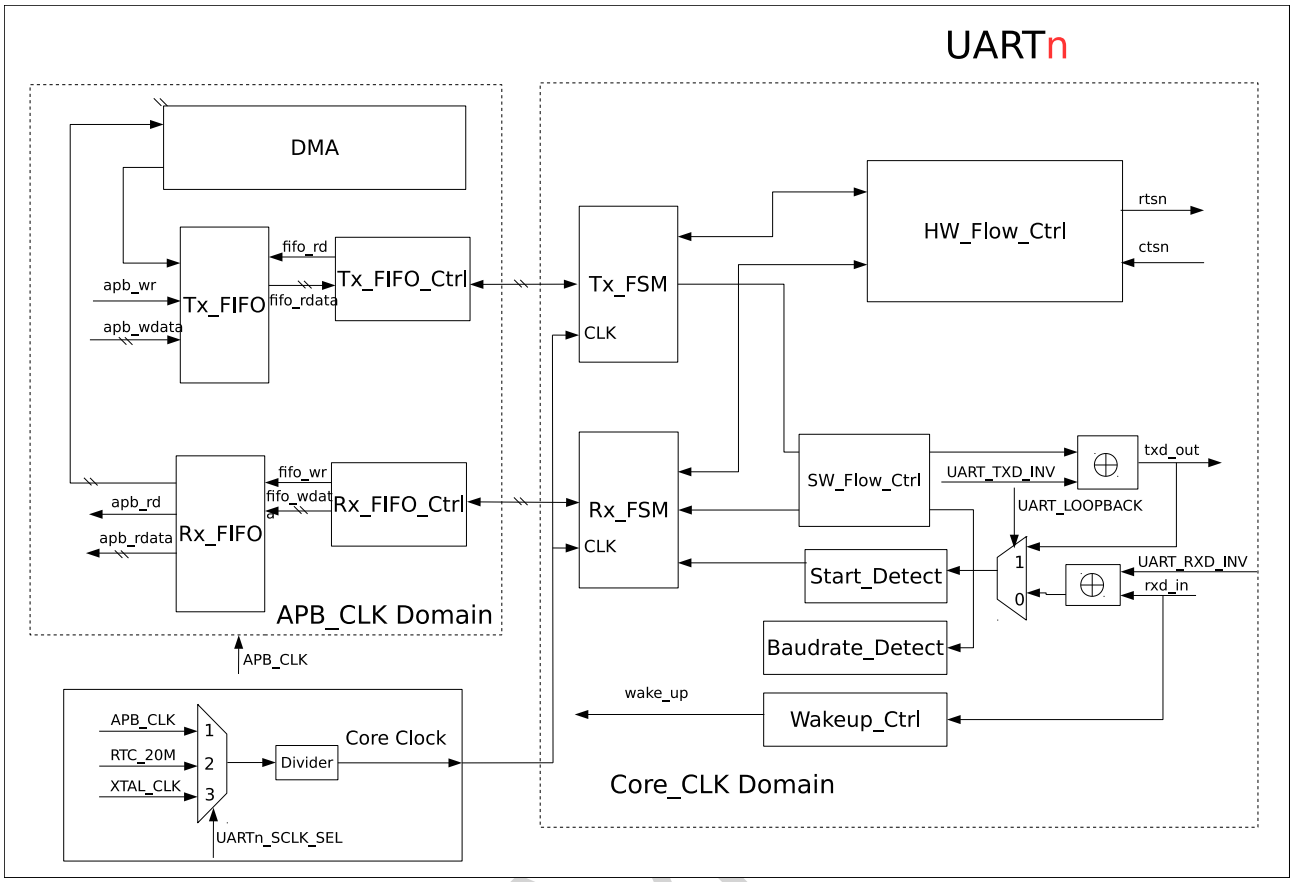


Figure 13-1. UART Structure

Figure 13-1 shows the basic structure 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). The UART Core has three clock sources: a 80 MHz APB_CLK, RTC20M_CLK and 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, or move data to Tx_FIFO using GDMA. 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, or receive data using GDMA.

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 Light-sleep mode (see Chapter 12 *Low-Power Management (RTC_CNTL) [to be added later]* 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 ESP32-C3 chip.

13.4 Functional Description

13.4.1 Clock and Reset

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 APB_CLK, RTC20M_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 ~ 256.

In cases when UART baud rate meet the needs, the UART Core can work at a lower clock frequency by division, to reduce power consumption. Usually the frequency of the UART Core's clock is lower than that of APB_CLK, and the UART Core's clock divisor can be configured to the maximum when baud rate can meet the needs. The frequency of the UART Core's clock can also be higher than that of APB_CLK, at most three times that of APB_CLK. The clock for the UART transmitter and the UART receiver can be controlled independently. To enable the clock for the UART transmitter, please set UART_TX_SCLK_EN; to enable the clock for the UART receiver, set UART_RX_SCLK_EN.

To ensure that the configured register values are synchronized from APB_CLK domain to Core Clock domain, please follow procedures in Section13.5.

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 that it is not recommended to reset the APB clock domain module or UART Core only.

13.4.2 UART RAM

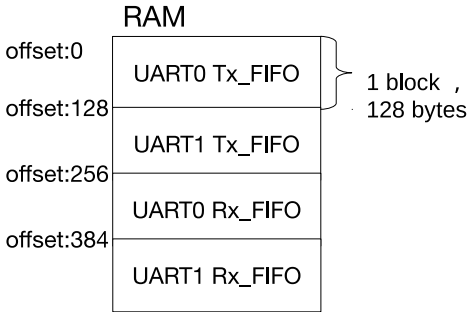


Figure 13-2. UART Controllers Sharing RAM

The two UART controllers on ESP32-C3 share 512 × 8 bits of FIFO RAM. As Figure 13-2 illustrates, RAM is divided into 4 blocks, each has 128 × 8 bits. Figure 13-2 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. The size of UART0 Tx_FIFO can be increased to 4 blocks (the whole RAM), the size of UART1 Tx_FIFO can be increased to 3 blocks (from offset 128 to the end address), the size of UART0 Rx_FIFO can be increased to 2 blocks (from offset 256 to the end address), but the size of UART1 Rx_FIFO cannot be increased. Please note that expanding one FIFO may take up the default space of other FIFOs. For example, by setting UART_TX_SIZE of UART0 to 2, the size of UART0 Tx_FIFO is increased by 128 bytes (from offset 0 to offset 255). In this case, UART0 Tx_FIFO takes up the default space for UART1 Tx_FIFO, and UART1's transmitting function cannot be used as a result.

When neither of the two UART controllers is active, RAM could enter low-power mode by setting UART_MEM_FORCE_PD.

UART0 Tx_FIFO and UART1 Tx_FIFO are reset by setting UART_TXFIFO_RST. UART0 Rx_FIFO and UART1 Rx_FIFO are reset by setting UART_RXFIFO_RST.

Data to be sent is written to TX FIFO via the APB bus or using GDMA, read automatically and converted from a frame into a bitstream by hardware Tx_FSM; data received is converted from a bitstream into a frame by hardware Rx_FSM, written into RX FIFO, and then stored into RAM via the APB bus or using GDMA. The two UART controllers share one GDMA channel.

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.

13.4.3 Baud Rate Generation and Detection

13.4.3.1 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 80 MHz input clock, the UART controller supports a maximum baud rate of 5 Mbaud.

The divisor of the baud rate divider is equal to UART_CLKDIV + (UART_CLKDIV_FRAG/16), meaning that the final baud rate is equal to INPUT_FREQ/(UART_CLKDIV + (UART_CLKDIV_FRAG/16)). For example, if UART_CLKDIV = 694 and UART_CLKDIV_FRAG = 7 then the divisor value is (694 + 7/16) = 694.4375. Note: INPUT_FREQ is the frequency of UART Cores' clock.

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 13-3, for every 16 output pulses, the generator divides either (UART_CLKDIV + 1) input pulses or UART_CLKDIV input pulses per output pulse. A total of UART_CLKDIV_FRAG output pulses are generated by dividing (UART_CLKDIV + 1) input pulses, and the remaining (16 - UART_CLKDIV_FRAG) output pulses are generated by dividing UART_CLKDIV input pulses.

The output pulses are interleaved as shown in Figure 13-3 below, to make the output timing more uniform:

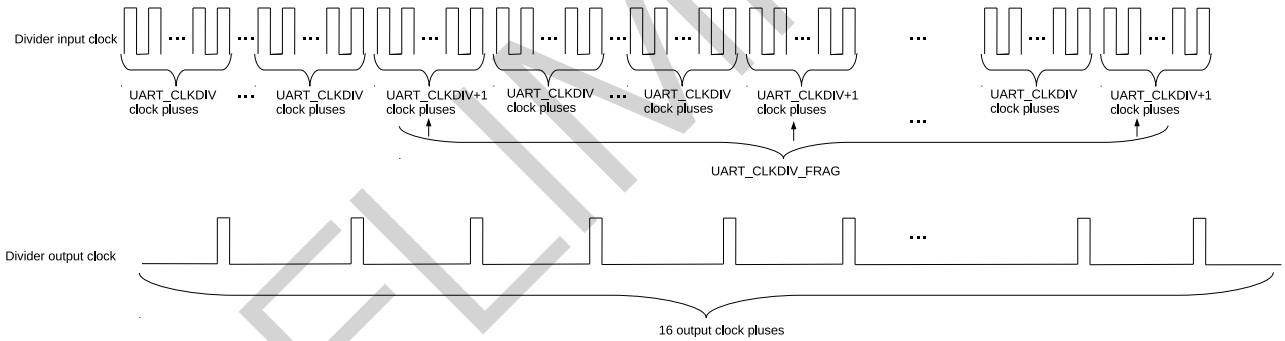


Figure 13-3. UART Controllers Division

To support IrDA (see Section 13.4.6 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.

13.4.3.2 Baud Rate Detection

Automatic baud rate detection (Autobaud) on UARTs is enabled by setting UART_AUTOBAUD_EN. The Baudrate_Detect module shown in Figure 13-1 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.

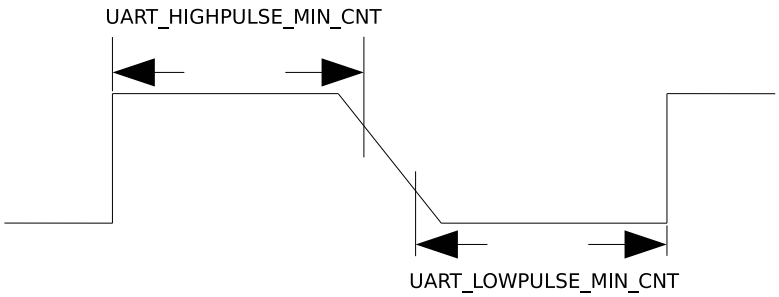


Figure 13-4. The Timing Diagram of Weak UART Signals Along Falling Edges

Baud rate can be determined in the following three ways:

1. Normally, to avoid sampling erroneous data along rising or falling edges in semi-stable state, which results in inaccuracy of UART_LOWPULSE_MIN_CNT or UART_HIGHPULSE_MIN_CNT, use a weighted average of these two values to eliminate errors. In this case, baud rate is calculated as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_LOWPULSE\_MIN\_CNT} + \text{UART\_HIGHPULSE\_MIN\_CNT} + 2)/2}$$

2. If UART signals are weak along falling edges as shown in Figure 13-4, which leads to inaccurate average of UART_LOWPULSE_MIN_CNT and UART_HIGHPULSE_MIN_CNT, use UART_POSEDGE_MIN_CNT to determine the transmitter's baud rate as follows:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_POSEDGE\_MIN\_CNT} + 1)/2}$$

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

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_NEGEDGE\_MIN\_CNT} + 1)/2}$$

13.4.4 UART Data Frame

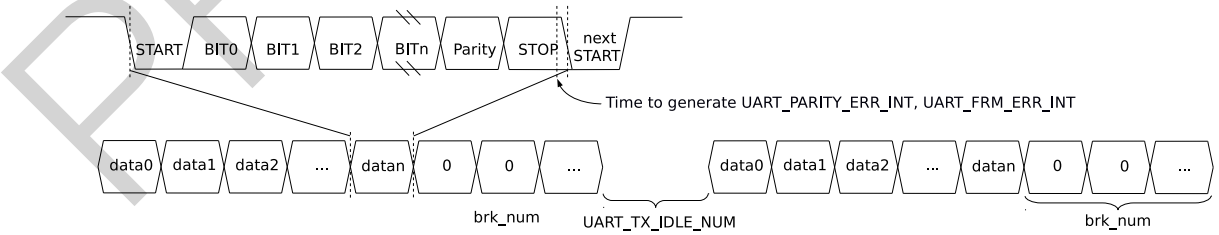


Figure 13-5. Structure of UART Data Frame

Figure 13-5 shows the basic structure of a data frame. A frame starts with one START bit, and ends with STOP bits which can be 1, 1.5, 2 or 3 bits long, configured by UART_STOP_BIT_NUM, UART_DL1_EN and UART_DL0_EN. The START bit is logical low, whereas STOP bits are logical high.

The actual data length can be anywhere between 5 ~ 8 bit, configured by UART_BIT_NUM. When UART_PARITY_EN is set, a parity bit is added after data bits. UART_PARITY is used to choose even parity or odd parity. When the receiver detects a parity bit error in data received, a UART_PARITY_ERR_INT interrupt is generated, and the data received is still stored into RX FIFO. When the receiver detects a data frame error, a UART_FRM_ERR_INT interrupt is generated, and 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.

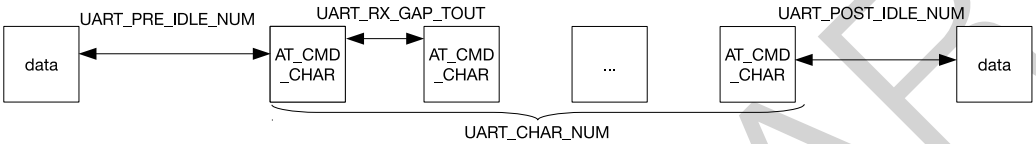


Figure 13-6. AT_CMD Character Structure

Figure 13-6 is the structure of a special character AT_CMD. If the receiver constantly receives AT_CMD_CHAR and the following conditions are met, a UART_AT_CMD_CHAR_DET_INT interrupt is generated.

- The interval between the first AT_CMD_CHAR and the last non-AT_CMD_CHAR character is at least UART_PRE_IDLE_NUM 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.

13.4.5 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.

13.4.5.1 Driver Control

As shown in Figure 13-7, in a two-wire multidrop network, an external RS485 transceiver is needed for differential to single-ended conversion. A RS485 transceiver contains a driver and a receiver. When a UART controller is not in transmitter mode, the connection to the differential line can be broken by disabling the driver. When DE is 1, the driver is enabled; when DE is 0, the driver is disabled.

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

DE can be controlled by either software or hardware. To reduce the cost of software, in our design DE is controlled by hardware. As shown in Figure 13-7, DE is connected to dtrn_out of UART (please refer to Section 13.4.8.1 for more details).

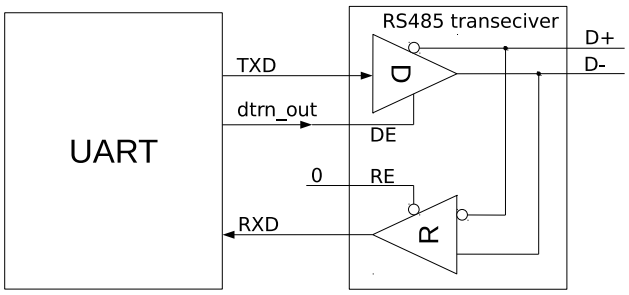


Figure 13-7. Driver Control Diagram in RS485 Mode

13.4.5.2 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. 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.

13.4.5.3 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 13-7, a UART controller may receive data in transmitter mode and snoop the bus. If UART_RS485RXBY_TX_EN is set, a UART controller may transmit data in receiver mode.

The two UART controllers can snoop data sent by themselves. In transmitter mode, when a UART controller monitors a collision between data sent and data received, a UART_RS485_CLASH_INT is generated; when a UART controller monitor a data frame error, a UART_RS485_FRM_ERR_INT interrupt is generated; when a UART controller monitors a polarity error, a UART_RS485_PARITY_ERR_INT is generated.

13.4.6 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 13-8, the IrDA encoder converts a NRZ (non-return to zero code) signal to a RZI (return to zero code) signal and sends it to the external driver and infrared LED. This encoder uses modulated signals whose pulse width is 3/16 bits to indicate logic "0", and low levels to indicate logic "1". The IrDA decoder receives signals from the infrared receiver and converts them to NRZ signals. In most cases, the receiver is high when it is idle, and the encoder output polarity is the opposite of the decoder input polarity. If a low pulse is detected, it indicates that a start bit has been received.

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

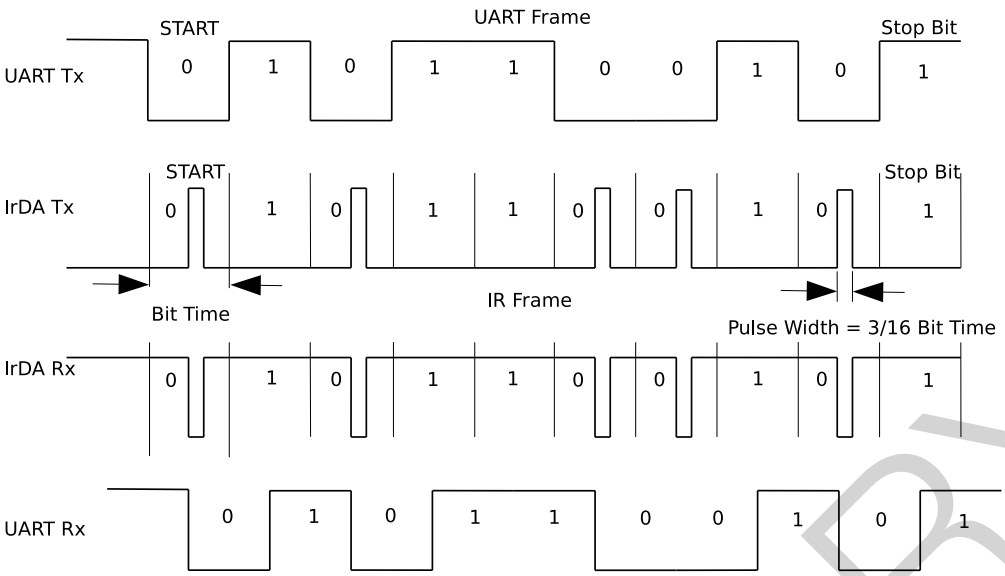


Figure 13-8. The Timing Diagram of Encoding and Decoding in SIR mode

The IrDA transceiver is half-duplex, meaning that it cannot send and receive data simultaneously. As shown in Figure 13-9, IrDA function is enabled by setting UART_IRDA_EN. When UART_IRDA_TX_EN is set (high), the IrDA transceiver is enabled to send data and not allowed to receive data; when UART_IRDA_TX_EN is reset (low), the IrDA transceiver is enabled to receive data and not allowed to send data.

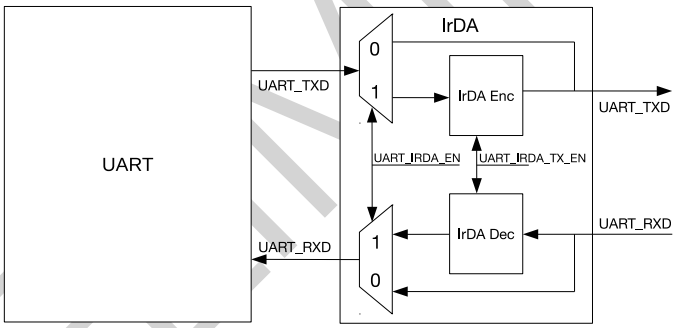


Figure 13-9. IrDA Encoding and Decoding Diagram

13.4.7 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 ESP32-C3.

13.4.8 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.

13.4.8.1 Hardware Flow Control

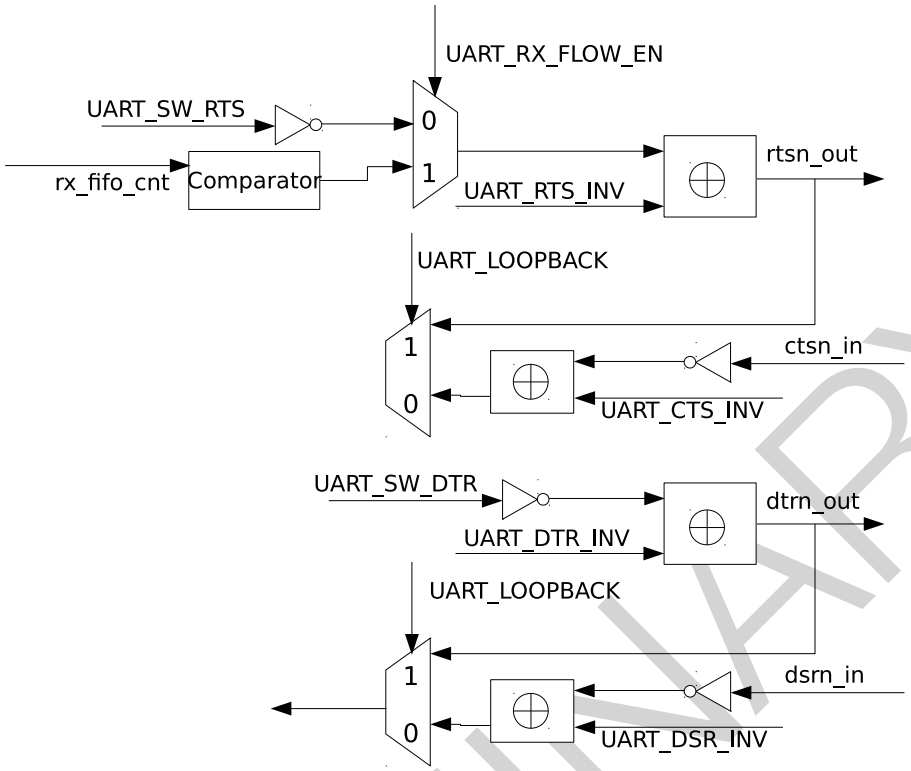


Figure 13-10. Hardware Flow Control Diagram

Figure 13-10 shows hardware flow control of a UART controller. Hardware flow control uses output signal rtsn_out and input signal dsrn_in. Figure 13-11 illustrates how these signals are connected between UART on ESP32-C3 (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.

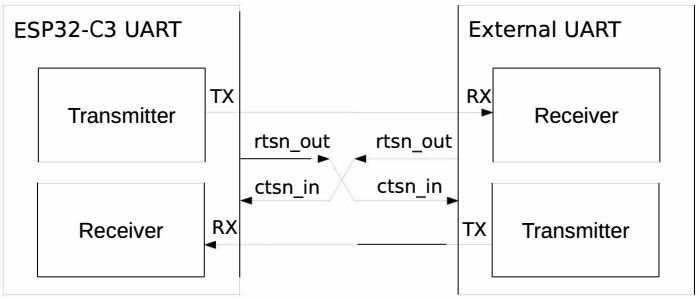


Figure 13-11. Connection between Hardware Flow Control Signals

When ctsn_in of IU0 is low, IU0 is allowed to send data; when ctsn_in is high, IU0 is not allowed to send data.

When IU0 detects an edge change of ctsn_in, a UART_CTS_CHG_INT interrupt is generated.

If dtrn_out of IU0 is high, it indicates that IU0 is ready to transmit data. dtrn_out is generated by configuring 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 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.

13.4.8.2 Software Flow Control

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

When using software flow control, hardware automatically detects if there are XON/XOFF characters in data flow received, and generate a UART_SW_XOFF_INT or a UART_SW_XON_INT interrupt accordingly. If an XOFF character is detected, the transmitter stops data transmission once the current byte has been transmitted; if an XON character is detected, the transmitter starts data transmission. In addition, software can force the transmitter to stop sending data by setting UART_FORCE_XOFF, or to start sending data by setting UART_FORCE_XON.

Software determines whether to insert flow control characters according to the remaining room in RX FIFO. When UART_SEND_XOFF is set, the transmitter sends an XOFF character configured by UART_XOFF_CHAR after the current byte in transmission; when UART_SEND_XON is set, the transmitter sends an XON character configured by UART_XON_CHAR after the current byte in transmission. If the RX FIFO of a UART controller stores more data than UART_XOFF_THRESHOLD, UART_SEND_XOFF is set by hardware. As a result, the transmitter sends an XOFF character 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.

13.4.9 GDMA Mode

The two UART controllers on ESP32-C3 share one TX/RX GDMA (general direct memory access) channel via UHCI. In GDMA mode, UART controllers support the decoding and encoding of HCI data packets. The UHCI_UART*n*_CE field determines which UART controller occupies the GDMA TX/RX channel.

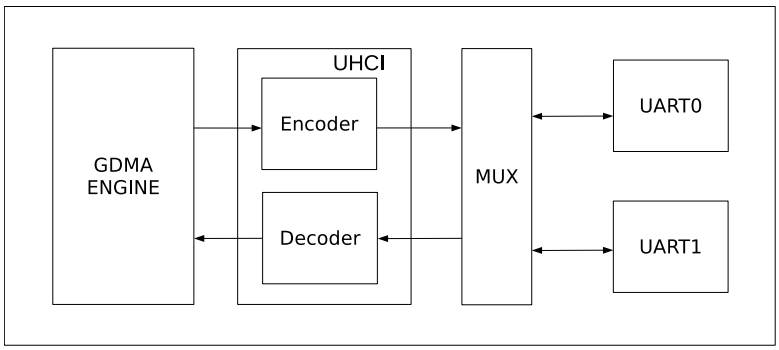


Figure 13-12. Data Transfer in GDMA Mode

Figure 13-12 shows how data is transferred using GDMA. Before GDMA receives data, software prepares an inlink. GDMA_INLINK_ADDR_CH*n* points to the first receive descriptor in the inlink. After GDMA_INLINK_START_CH*n* is set, UHCI sends data that UART has received to the decoder. The decoded data is then stored into the RAM pointed by the inlink under the control of GDMA.

Before GDMA sends data, software prepares an outlink and data to be sent. GDMA_OUTLINK_ADDR_CH*n* points to the first transmit descriptor in the outlink. After GDMA_OUTLINK_START_CH*n* is set, GDMA reads data from the RAM pointed by outlink. The data is then encoded by the encoder, and sent sequentially by the UART transmitter.

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

13.4.10 UART Interrupts

- UART_AT_CMD_CHAR_DET_INT: Triggered when the receiver detects an AT_CMD character.
- UART_RS485_CLASH_INT: Triggered when a collision is detected between the transmitter and the receiver in RS485 mode.
- UART_RS485_FRM_ERR_INT: Triggered when an error is detected in the data frame sent by the transmitter in RS485 mode.
- UART_RS485_PARITY_ERR_INT: Triggered when an error is detected in the parity bit sent by the transmitter in RS485 mode.
- UART_TX_DONE_INT: Triggered when all data in the transmitter's TX FIFO has been sent.
- UART_TX_BRK_IDLE_DONE_INT: Triggered when the transmitter stays idle for the minimum interval (threshold) after sending the last data bit.
- UART_TX_BRK_DONE_INT: Triggered when the transmitter has sent all NULL characters after all data in TX FIFO had been sent.
- UART_GLITCH_DET_INT: Triggered when the receiver detects a glitch in the middle of the start bit.

- UART_SW_XOFF_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XOFF character.
- UART_SW_XON_INT: Triggered when UART_SW_FLOW_CON_EN is set and the receiver receives a XON character.
- UART_RXFIFO_TOUT_INT: Triggered when the receiver takes more time than UART_RX_TOUT_THRHD to receive one byte.
- UART_BRK_DET_INT: Triggered when the receiver detects a NULL character after stop bits.
- UART_CTS_CHG_INT: Triggered when the receiver detects an edge change of CTSn signals.
- UART_DSR_CHG_INT: Triggered when the receiver detects an edge change of DSRn signals.
- UART_RXFIFO_OVF_INT: Triggered when the receiver receives more data than the capacity of RX FIFO.
- UART_FRM_ERR_INT: Triggered when the receiver detects a data frame error.
- UART_PARITY_ERR_INT: Triggered when the receiver detects a parity error.
- UART_TXFIFO_EMPTY_INT: Triggered when TX FIFO stores less data than what UART_TXFIFO_EMPTY_THRHD specifies.
- UART_RXFIFO_FULL_INT: Triggered when the receiver receives more data than what UART_RXFIFO_FULL_THRHD specifies.
- UART_WAKEUP_INT: Triggered when UART is woken up.

13.4.11 UHCI Interrupts

- UHCI_APP_CTRL1_INT: Triggered when software sets UHCI_APP_CTRL1_INT_RAW.
- UHCI_APP_CTRL0_INT: Triggered when software sets UHCI_APP_CTRL0_INT_RAW.
- UHCI_OUTLINK_EOF_ERR_INT: Triggered when an EOF error is detected in a transmit descriptor.
- UHCI_SEND_A_REG_Q_INT: Triggered when UHCI has sent a series of short packets using always_send.
- UHCI_SEND_S_REG_Q_INT: Triggered when UHCI has sent a series of short packets using single_send.
- UHCI_TX_HUNG_INT: Triggered when UHCI takes too long to read RAM using a GDMA transmit channel.
- UHCI_RX_HUNG_INT: Triggered when UHCI takes too long to receive data using a GDMA receive channel.
- UHCI_TX_START_INT: Triggered when GDMA detects a separator character.
- UHCI_RX_START_INT: Triggered when a separator character has been sent.

13.5 Programming Procedures

13.5.1 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.

13.5.1.1 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 13-1 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 13-1. 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

13.5.1.2 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 semi-stable state. When software turns on the clock, the configured values are stable to be correctly sampled. Static registers as listed in Table 13-2 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 13-2. 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]

13.5.1.3 Immediate Registers

Except those listed in Table 13-1 and Table 13-2, registers that can be configured by software are immediate registers read in APB_CLK domain, such as interrupt and FIFO configuration registers.

13.5.2 Detailed Steps

Figure 13-13 illustrates the process to program UART controllers, namely initialize UART, configure registers, enable the UART transmitter or receiver, and finish data transmission.

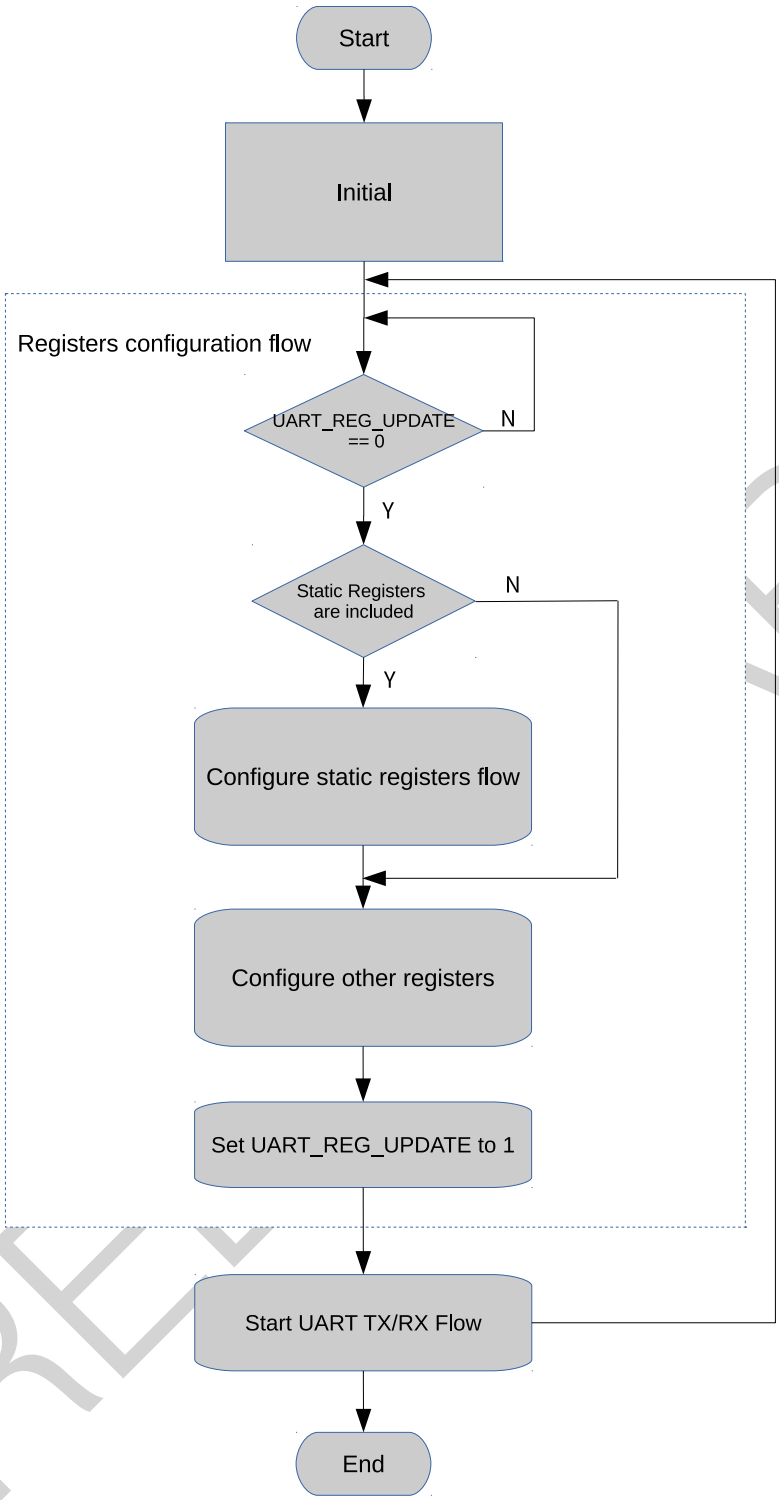


Figure 13-13. UART Programming Procedures

13.5.2.1 Initializing URAT*n*

To initialize URAT*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.

13.5.2.2 Configuring URAT*n* Communication

To configure URAT*n* communication:

- wait for UART_REG_UPDATE to become 0, which indicates the completion of last synchronization;
- configure static registers (if any) following Section 13.5.1.2;
- 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 application ...
- synchronize the configured values to Core Clock domain by writing 1 to UART_REG_UPDATE.

13.5.2.3 Enabling UART*n* Transmitter and Sending Data

To enable UART*n* transmitter:

- configure TXFIFO'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.

13.5.2.4 Enabling UART*n* Receiver and Retrieving Data

To enable UART*n* receiver:

- configure RXFIFO'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 RXFIFO is full;
- read data from RXFIFO via UART_RXFIFO_RD_BYTE, and obtain the number of bytes received in RXFIFO via UART_RXFIFO_CNT.

13.6 Register Summary

The addresses in this section are relative to UART Controller base address provided in Table 3-4 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
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	0x003C	R/W
UART_SWFC_CONF1_REG	Software flow-control character configuration	0x0040	R/W
UART_TXBRK_CONF_REG	TX break character configuration	0x0044	R/W
UART_IDLE_CONF_REG	Frame-end idle 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 transmit and receive 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
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

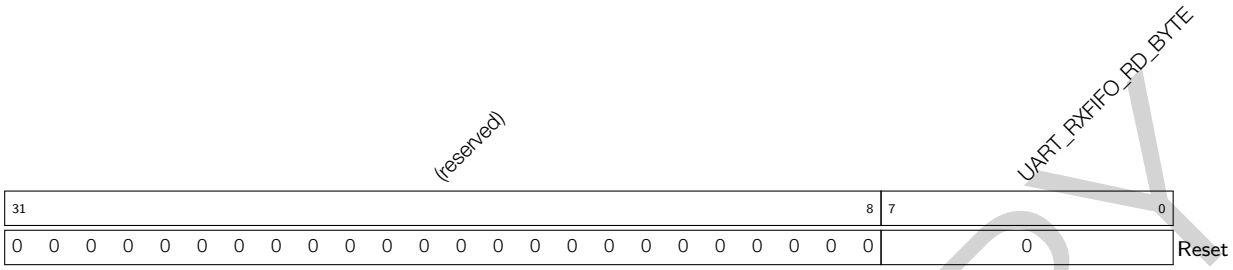
Name	Description	Address	Access
UART_ID_REG	UART ID register	0x0080	varies

Name	Description	Address	Access
Configuration Register			
UHCI_CONF0_REG	UHCI configuration register	0x0000	R/W
UHCI_CONF1_REG	UHCI configuration register	0x0014	varies
UHCI_ESCAPE_CONF_REG	Escape character configuration	0x0020	R/W
UHCI_HUNG_CONF_REG	Timeout configuration	0x0024	R/W
UHCI_ACK_NUM_REG	UHCI ACK number configuration	0x0028	varies
UHCI_QUICK_SENT_REG	UHCI quick send configuration register	0x0030	varies
UHCI_REG_Q0_WORD0_REG	Q0_WORD0 quick_sent register	0x0034	R/W
UHCI_REG_Q0_WORD1_REG	Q0_WORD1 quick_sent register	0x0038	R/W
UHCI_REG_Q1_WORD0_REG	Q1_WORD0 quick_sent register	0x003C	R/W
UHCI_REG_Q1_WORD1_REG	Q1_WORD1 quick_sent register	0x0040	R/W
UHCI_REG_Q2_WORD0_REG	Q2_WORD0 quick_sent register	0x0044	R/W
UHCI_REG_Q2_WORD1_REG	Q2_WORD1 quick_sent register	0x0048	R/W
UHCI_REG_Q3_WORD0_REG	Q3_WORD0 quick_sent register	0x004C	R/W
UHCI_REG_Q3_WORD1_REG	Q3_WORD1 quick_sent register	0x0050	R/W
UHCI_REG_Q4_WORD0_REG	Q4_WORD0 quick_sent register	0x0054	R/W
UHCI_REG_Q4_WORD1_REG	Q4_WORD1 quick_sent register	0x0058	R/W
UHCI_REG_Q5_WORD0_REG	Q5_WORD0 quick_sent register	0x005C	R/W
UHCI_REG_Q5_WORD1_REG	Q5_WORD1 quick_sent register	0x0060	R/W
UHCI_REG_Q6_WORD0_REG	Q6_WORD0 quick_sent register	0x0064	R/W
UHCI_REG_Q6_WORD1_REG	Q6_WORD1 quick_sent register	0x0068	R/W
UHCI_ESC_CONF0_REG	Escape sequence configuration register 0	0x006C	R/W
UHCI_ESC_CONF1_REG	Escape sequence configuration register 1	0x0070	R/W
UHCI_ESC_CONF2_REG	Escape sequence configuration register 2	0x0074	R/W
UHCI_ESC_CONF3_REG	Escape sequence configuration register 3	0x0078	R/W
UHCI_PKT_THRES_REG	Configure register for packet length	0x007C	R/W
Interrupt Register			
UHCI_INT_RAW_REG	Raw interrupt status	0x0004	varies
UHCI_INT_ST_REG	Masked interrupt status	0x0008	RO
UHCI_INT_ENA_REG	Interrupt enable bits	0x000C	R/W
UHCI_INT_CLR_REG	Interrupt clear bits	0x0010	WT
UHCI Status Register			
UHCI_STATE0_REG	UHCI receive status	0x0018	RO
UHCI_STATE1_REG	UHCI transmit status	0x001C	RO
UHCI_RX_HEAD_REG	UHCI packet header register	0x002C	RO
Version Register			
UHCI_DATE_REG	UHCI version control register	0x0080	R/W

13.7 Registers

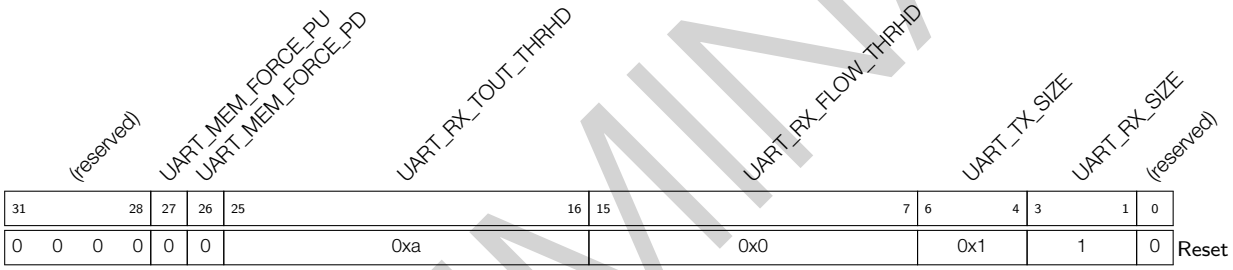
The addresses in this section are relative to UART Controller base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Register 13.1. UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE UART*n* accesses FIFO via this register. (RO)

Register 13.2. UART_MEM_CONF_REG (0x0060)



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

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

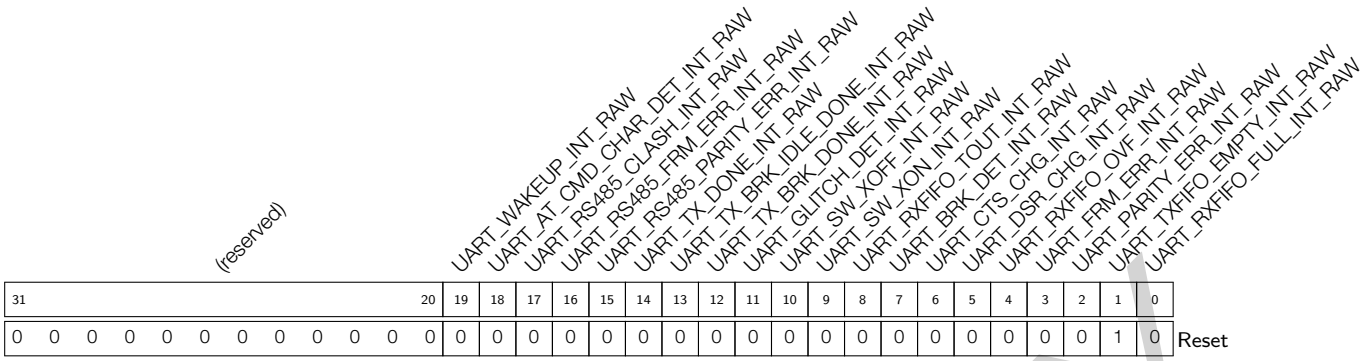
UART_RX_FLOW_THRHD This register is used to configure the maximum amount of data that can be received when hardware flow control works. (R/W)

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

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

Register 13.3. UART_INT_RAW_REG (0x0004)



UART_RXFIFO_FULL_INT_RAW This interrupt raw bit turns to high level when receiver receives more data than what UART_RXFIFO_FULL_THRHD specifies. (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 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 receiver detects a data frame error. (R/WTC/SS)

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

UART_DSR_CHG_INT_RAW This interrupt raw bit turns to high level when 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 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 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 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 receiver receives XON character when 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 receiver receives XOFF character when 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 receiver detects a glitch in the middle of a start bit. (R/WTC/SS)

Continued on the next page...

Register 13.3. 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 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 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 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 receiver detects a parity error from the echo of transmitter in RS485 mode. (R/WTC/SS)

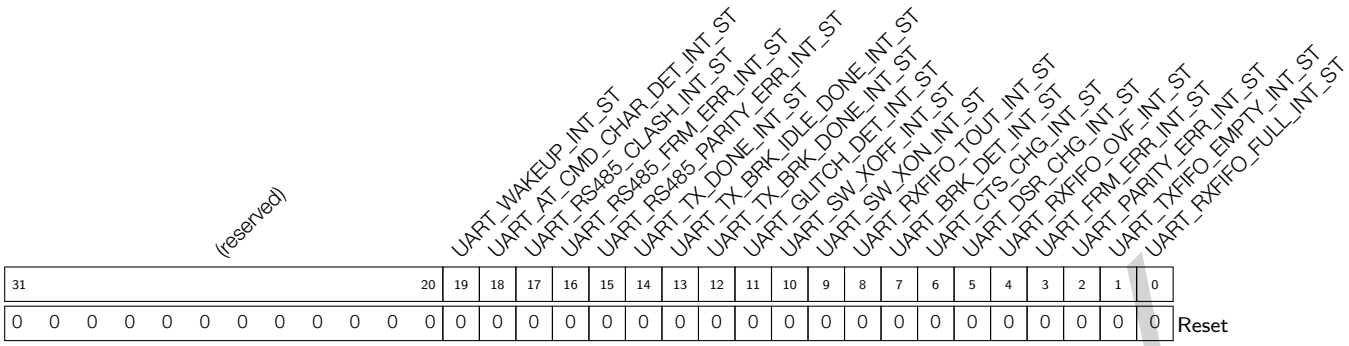
UART_RS485_FRM_ERR_INT_RAW This interrupt raw bit turns to high level when receiver detects a data frame error from the echo of transmitter in RS485 mode. (R/WTC/SS)

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

UART_AT_CMD_CHAR_DET_INT_RAW This interrupt raw bit turns to high level when receiver detects the configured UART_AT_CMD CHAR. (R/WTC/SS)

UART_WAKEUP_INT_RAW This interrupt raw bit turns to high level when input RXD edge changes more times than what UART_ACTIVE_THRESHOLD specifies in Light-sleep mode. (R/WTC/SS)

Register 13.4. UART_INT_ST_REG (0x0008)



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

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

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

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

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

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

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

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

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

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

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

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

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

Continued on the next page...

Register 13.4. UART_INT_ST_REG (0x0008)

Continued from the previous page...

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

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

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

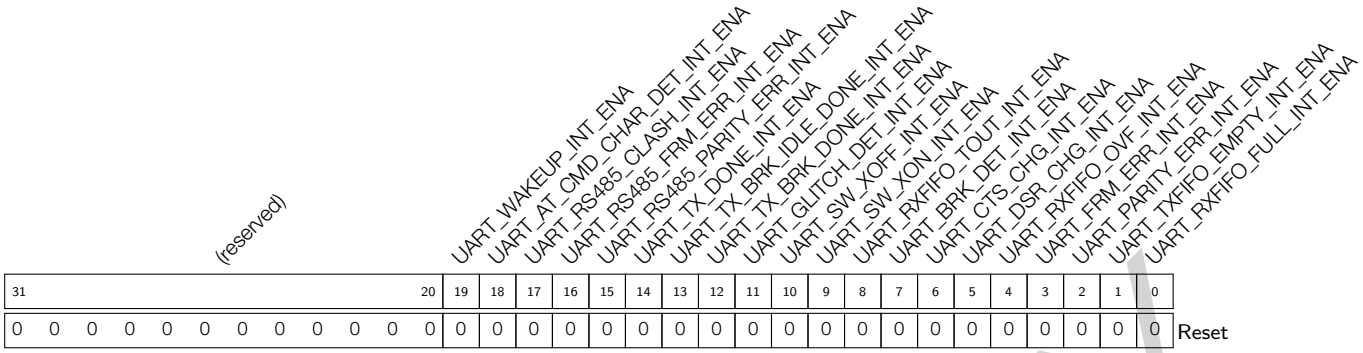
UART_RS485_FRM_ERR_INT_ST This is the status bit for UART_RS485_FRM_ERR_INT_RAW when UART_RS485_FM_ERR_INT_ENA is set to 1. (RO)

UART_RS485_CLASH_INT_ST This is the status bit for UART_RS485_CLASH_INT_RAW when UART_RS485_CLASH_INT_ENA is set to 1. (RO)

UART_AT_CMD_CHAR_DET_INT_ST This is the status bit for UART_AT_CMD_DET_INT_RAW when UART_AT_CMD_CHAR_DET_INT_ENA is set to 1. (RO)

UART_WAKEUP_INT_ST This is the status bit for UART_WAKEUP_INT_RAW when UART_WAKEUP_INT_ENA is set to 1. (RO)

Register 13.5. UART_INT_ENA_REG (0x000C)



UART_RXFIFO_FULL_INT_ENA This is the enable bit for UART_RXFIFO_FULL_INT_ST register. (R/W)

UART_TXFIFO_EMPTY_INT_ENA This is the enable bit for UART_TXFIFO_EMPTY_INT_ST register. (R/W)

UART_PARITY_ERR_INT_ENA This is the enable bit for UART_PARITY_ERR_INT_ST register. (R/W)

UART_FRM_ERR_INT_ENA This is the enable bit for UART_FRM_ERR_INT_ST register. (R/W)

UART_RXFIFO_OVF_INT_ENA This is the enable bit for UART_RXFIFO_OVF_INT_ST register. (R/W)

UART_DSR_CHG_INT_ENA This is the enable bit for UART_DSR_CHG_INT_ST register. (R/W)

UART_CTS_CHG_INT_ENA This is the enable bit for UART_CTS_CHG_INT_ST register. (R/W)

UART_BRK_DET_INT_ENA This is the enable bit for UART_BRK_DET_INT_ST register. (R/W)

UART_RXFIFO_TOUT_INT_ENA This is the enable bit for UART_RXFIFO_TOUT_INT_ST register. (R/W)

UART_SW_XON_INT_ENA This is the enable bit for UART_SW_XON_INT_ST register. (R/W)

UART_SW_XOFF_INT_ENA This is the enable bit for UART_SW_XOFF_INT_ST register. (R/W)

UART_GLITCH_DET_INT_ENA This is the enable bit for UART_GLITCH_DET_INT_ST register. (R/W)

UART_TX_BRK_DONE_INT_ENA This is the enable bit for UART_TX_BRK_DONE_INT_ST register. (R/W)

UART_TX_BRK_IDLE_DONE_INT_ENA This is the enable bit for UART_TX_BRK_IDLE_DONE_INT_ST register. (R/W)

UART_TX_DONE_INT_ENA This is the enable bit for UART_TX_DONE_INT_ST register. (R/W)

Continued on the next page...

Register 13.5. UART_INT_ENA_REG (0x000C)

Continued from the previous page...

UART_RS485_PARITY_ERR_INT_ENA This is the enable bit for UART_RS485_PARITY_ERR_INT_ST register. (R/W)

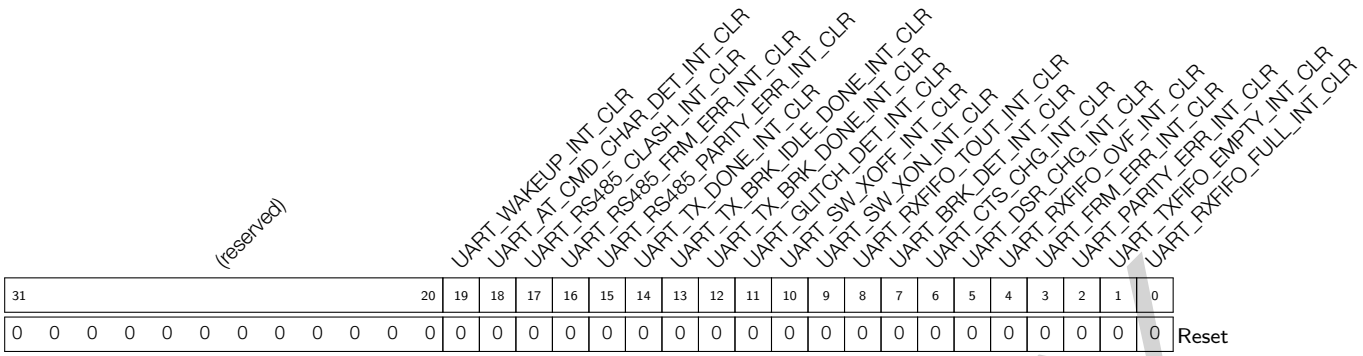
UART_RS485_FRM_ERR_INT_ENA This is the enable bit for UART_RS485_PARITY_ERR_INT_ST register. (R/W)

UART_RS485_CLASH_INT_ENA This is the enable bit for UART_RS485_CLASH_INT_ST register. (R/W)

UART_AT_CMD_CHAR_DET_INT_ENA This is the enable bit for UART_AT_CMD_CHAR_DET_INT_ST register. (R/W)

UART_WAKEUP_INT_ENA This is the enable bit for UART_WAKEUP_INT_ST register. (R/W)

Register 13.6. UART_INT_CLR_REG (0x0010)



UART_RXFIFO_FULL_INT_CLR Set this bit to clear UART_THE RXFIFO_FULL_INT_RAW interrupt. (WT)

UART_TXFIFO_EMPTY_INT_CLR Set this bit to clear UART_TXFIFO_EMPTY_INT_RAW interrupt. (WT)

UART_PARITY_ERR_INT_CLR Set this bit to clear UART_PARITY_ERR_INT_RAW interrupt. (WT)

UART_FRM_ERR_INT_CLR Set this bit to clear UART_FRM_ERR_INT_RAW interrupt. (WT)

UART_RXFIFO_OVF_INT_CLR Set this bit to clear UART_UART_RXFIFO_OVF_INT_RAW interrupt. (WT)

UART_DSR_CHG_INT_CLR Set this bit to clear UART_DSR_CHG_INT_RAW interrupt. (WT)

UART_CTS_CHG_INT_CLR Set this bit to clear UART_CTS_CHG_INT_RAW interrupt. (WT)

UART_BRK_DET_INT_CLR Set this bit to clear UART_BRK_DET_INT_RAW interrupt. (WT)

UART_RXFIFO_TOUT_INT_CLR Set this bit to clear UART_RXFIFO_TOUT_INT_RAW interrupt. (WT)

UART_SW_XON_INT_CLR Set this bit to clear UART_SW_XON_INT_RAW interrupt. (WT)

UART_SW_XOFF_INT_CLR Set this bit to clear UART_SW_XOFF_INT_RAW interrupt. (WT)

UART_GLITCH_DET_INT_CLR Set this bit to clear UART_GLITCH_DET_INT_RAW interrupt. (WT)

UART_TX_BRK_DONE_INT_CLR Set this bit to clear UART_TX_BRK_DONE_INT_RAW interrupt. (WT)

UART_TX_BRK_IDLE_DONE_INT_CLR Set this bit to clear UART_TX_BRK_IDLE_DONE_INT_RAW interrupt. (WT)

UART_TX_DONE_INT_CLR Set this bit to clear UART_TX_DONE_INT_RAW interrupt. (WT)

UART_RS485_PARITY_ERR_INT_CLR Set this bit to clear UART_RS485_PARITY_ERR_INT_RAW interrupt. (WT)

Continued on the next page...

Register 13.6. UART_INT_CLR_REG (0x0010)

Continued from the previous page...

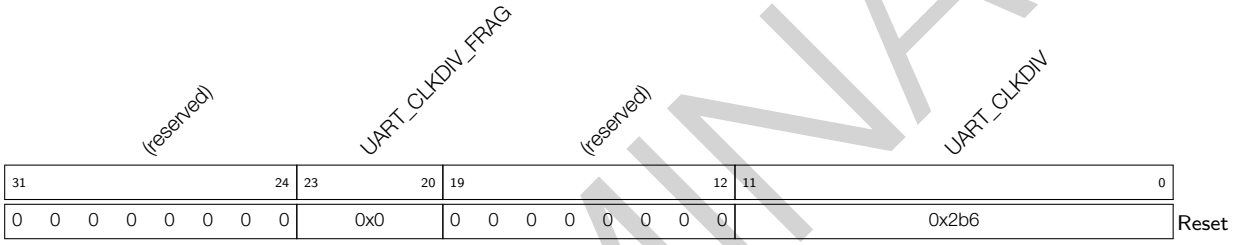
UART_RS485_FRM_ERR_INT_CLR Set this bit to clear UART_RS485_FRM_ERR_INT_RAW interrupt. (WT)

UART_RS485_CLASH_INT_CLR Set this bit to clear UART_RS485_CLASH_INT_RAW interrupt. (WT)

UART_AT_CMD_CHAR_DET_INT_CLR Set this bit to clear UART_AT_CMD_CHAR_DET_INT_RAW interrupt. (WT)

UART_WAKEUP_INT_CLR Set this bit to clear UART_WAKEUP_INT_RAW interrupt. (WT)

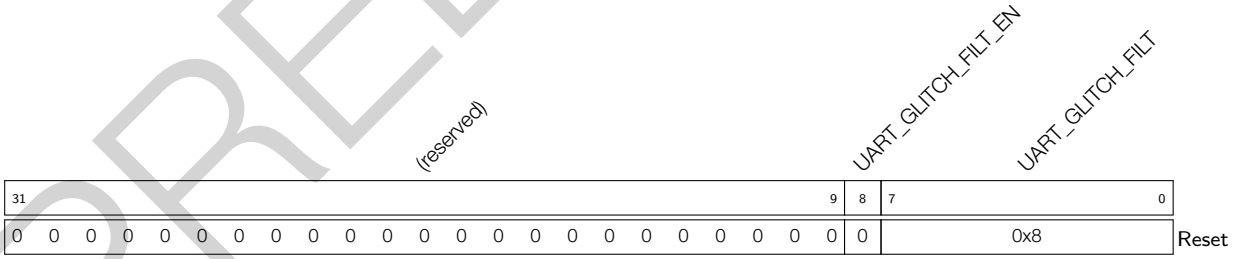
Register 13.7. UART_CLKDIV_REG (0x0014)



UART_CLKDIV The integral part of the frequency divisor. (R/W)

UART_CLKDIV_FRAG The decimal part of the frequency divisor. (R/W)

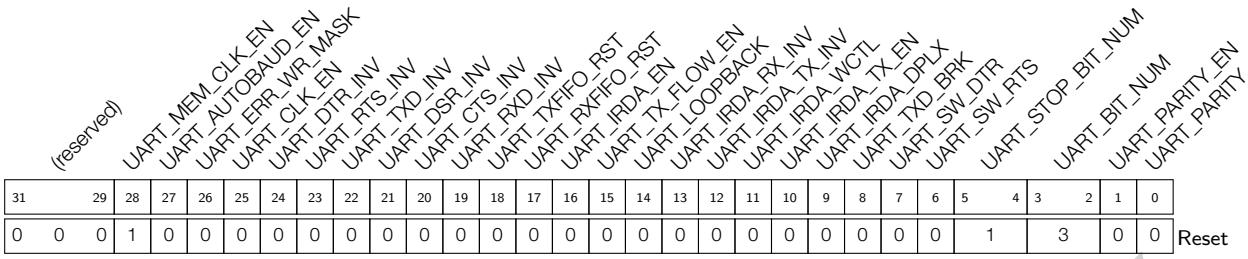
Register 13.8. UART_RX_FILT_REG (0x0018)



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 13.9. UART_CONF0_REG (0x0020)



UART_PARITY This register 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 register is used to set the length of data. (R/W)

UART_STOP_BIT_NUM This register is used to set the length of stop bit. (R/W)

UART_SW_RTS This register is used to configure the software RTS signal which is used in software flow control. (R/W)

UART_SW_DTR This register is used to configure the software DTR signal which is used in software flow control. (R/W)

UART_TXD_BRK Set this bit to enbale transmitter to send NULL when the process of sending data is done. (R/W)

UART_IRDA_DPLX Set this bit to enable IrDA loopback mode. (R/W)

UART_IRDA_TX_EN This is the start enable bit for IrDA transmitter. (R/W)

UART_IRDA_WCTL 1'h1: The IrDA transmitter's 11th bit is the same as 10th bit. 1'h0: Set IrDA transmitter's 11th bit to 0. (R/W)

UART_IRDA_TX_INV Set this bit to invert the level of IrDA transmitter. (R/W)

UART_IRDA_RX_INV Set this bit to invert the level of IrDA receiver. (R/W)

UART_LOOPBACK Set this bit to enable UART loopback test mode. (R/W)

UART_TX_FLOW_EN Set this bit to enable flow control function for transmitter. (R/W)

UART_IRDA_EN Set this bit to enable IrDA protocol. (R/W)

UART_RXFIFO_RST Set this bit to reset the UART RX FIFO. (R/W)

UART_TXFIFO_RST Set this bit to reset the UART TX FIFO. (R/W)

UART_RXD_INV Set this bit to inverse the level value of UART RXD signal. (R/W)

UART_CTS_INV Set this bit to inverse the level value of UART CTS signal. (R/W)

UART_DSR_INV Set this bit to inverse the level value of UART DSR signal. (R/W)

Continued on the next page...

Register 13.9. UART_CONF0_REG (0x0020)

Continued from the previous page...

UART_TXD_INV Set this bit to inverse the level value of UART TXD signal. (R/W)

UART_RTS_INV Set this bit to inverse the level value of UART RTS signal. (R/W)

UART_DTR_INV Set this bit to inverse the level value of UART DTR signal. (R/W)

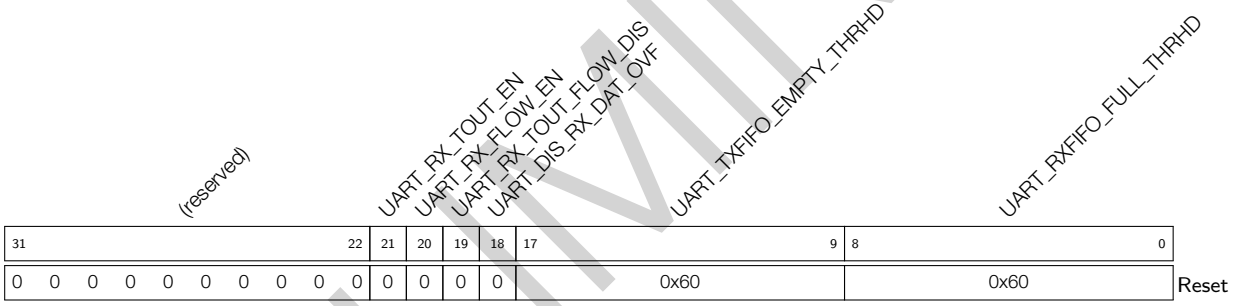
UART_CLK_EN 1'h1: Force clock on for register. 1'h0: Support clock only when application writes registers. (R/W)

UART_ERR_WR_MASK 1'h1: Receiver stops storing data into FIFO when data is wrong. 1'h0: Receiver stores the data even if the received data is wrong. (R/W)

UART_AUTOBAUD_EN This is the enable bit for detecting baud rate. (R/W)

UART_MEM_CLK_EN UART memory clock gate enable signal. (R/W)

Register 13.10. UART_CONF1_REG (0x0024)



UART_RXFIFO_FULL_THRHD It will produce UART_RXFIFO_FULL_INT interrupt when receiver receives more data than this register value. (R/W)

UART_TXFIFO_EMPTY_THRHD It will produce UART_TXFIFO_EMPTY_INT interrupt when the data amount in TX FIFO is less than this register value. (R/W)

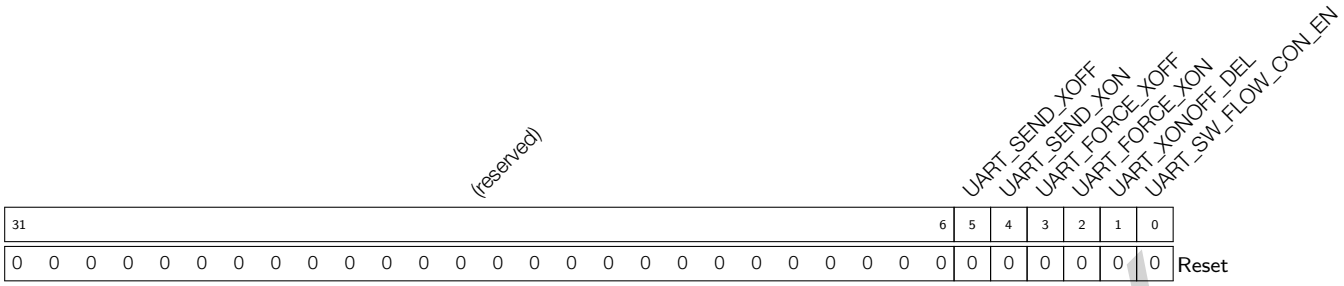
UART_DIS_RX_DAT_OVF Disable UART RX data overflow detection. (R/W)

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 13.11. UART_FLOW_CONF_REG (0x0034)



UART_SW_FLOW_CON_EN Set this bit to enable software flow control. It is used with register SW_XON or SW_XOFF. (R/W)

UART_XONOFF_DEL Set this bit to remove flow control char from the received data. (R/W)

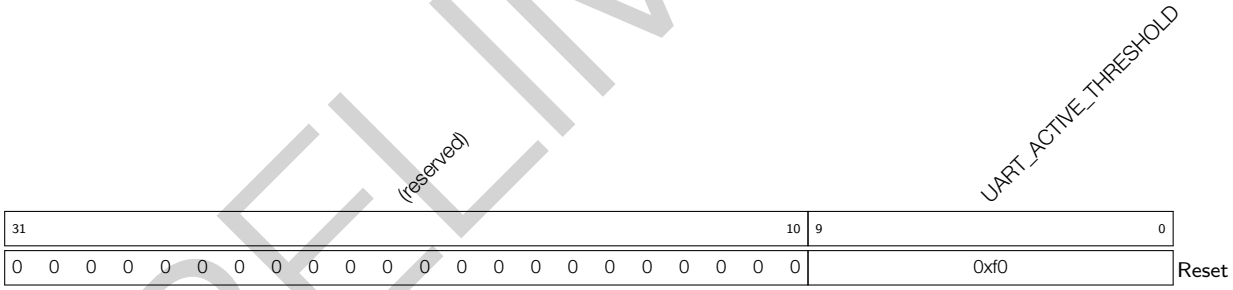
UART_FORCE_XON Set this bit to enable the transmitter to go on sending data. (R/W)

UART_FORCE_XOFF Set this bit to stop the transmitter from sending data. (R/W)

UART_SEND_XON Set this bit to send XON character. It is cleared by hardware automatically. (R/W/SS/SC)

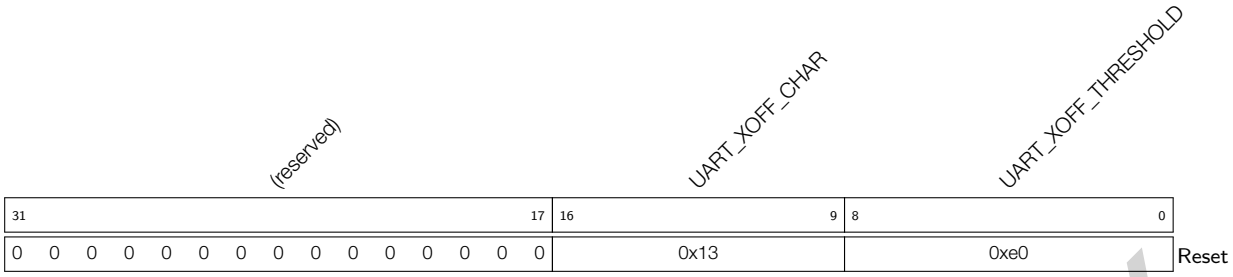
UART_SEND_XOFF Set this bit to send XOFF character. It is cleared by hardware automatically. (R/W/SS/SC)

Register 13.12. UART_SLEEP_CONF_REG (0x0038)



UART_ACTIVE_THRESHOLD The UART is activated from light-sleep mode when the input RXD edge changes more times than this register value. (R/W)

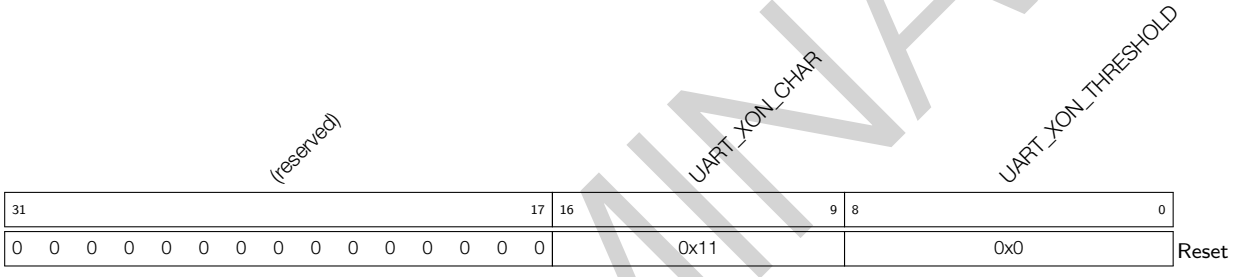
Register 13.13. UART_SWFC_CONF0_REG (0x003C)



UART_XOFF_THRESHOLD When the data amount in RX FIFO is more than this register value with UART_SW_FLOW_CON_EN set to 1, it will send a XOFF character. (R/W)

UART_XOFF_CHAR This register stores the XOFF flow control character. (R/W)

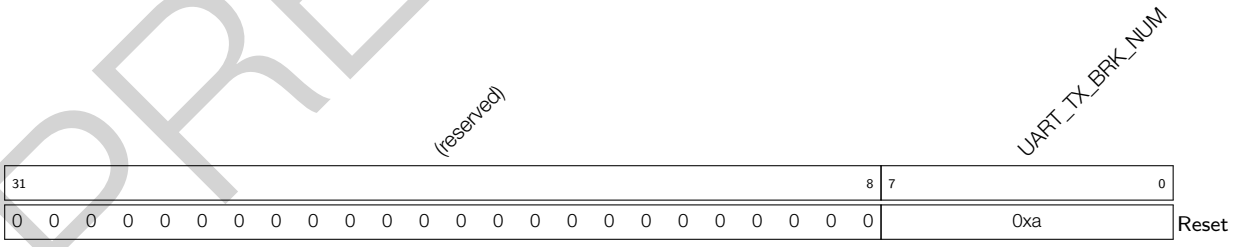
Register 13.14. UART_SWFC_CONF1_REG (0x0040)



UART_XON_THRESHOLD When the data amount in RX FIFO is less than this register value with UART_SW_FLOW_CON_EN set to 1, it will send a XON character. (R/W)

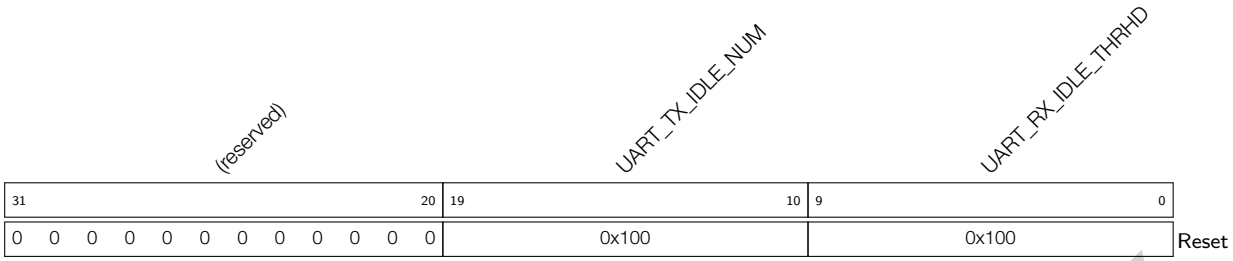
UART_XON_CHAR This register stores the XON flow control character. (R/W)

Register 13.15. UART_TXBRK_CONF_REG (0x0044)



UART_TX_BRK_NUM This register is used to configure the number of 0 to be sent after the process of sending data is done. It is active when txd_brk is set to 1. (R/W)

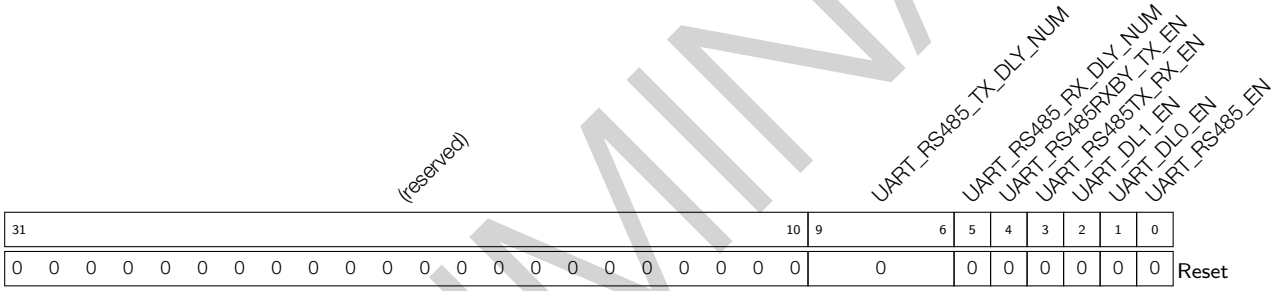
Register 13.16. UART_IDLE_CONF_REG (0x0048)



UART_RX_IDLE_THRHD It will produce frame end signal when receiver takes more time to receive one byte data than this register value, in the unit of bit time (the time it takes to transfer one bit). (R/W)

UART_TX_IDLE_NUM This register 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 13.17. UART_RS485_CONF_REG (0x004C)



UART_RS485_EN Set this bit to choose the RS485 mode. (R/W)

UART_DL0_EN Set this bit to delay the stop bit by 1 bit. (R/W)

UART_DL1_EN Set this bit to delay the stop bit by 1 bit. (R/W)

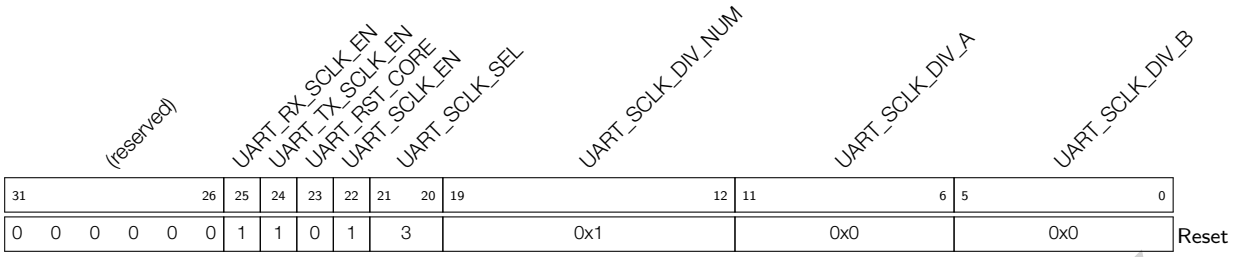
UART_RS485TX_RX_EN Set this bit to enable receiver could receive data when the transmitter is transmitting data in RS485 mode. (R/W)

UART_RS485RXBY_TX_EN 1'h1: enable RS485 transmitter to send data when RS485 receiver line is busy. (R/W)

UART_RS485_RX_DLY_NUM This register is used to delay the receiver's internal data signal. (R/W)

UART_RS485_TX_DLY_NUM This register is used to delay the transmitter's internal data signal. (R/W)

Register 13.18. 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 UART clock source select. 1: APB_CLK; 2: RTC20M_CLK; 3: XTAL_CLK. (R/W)

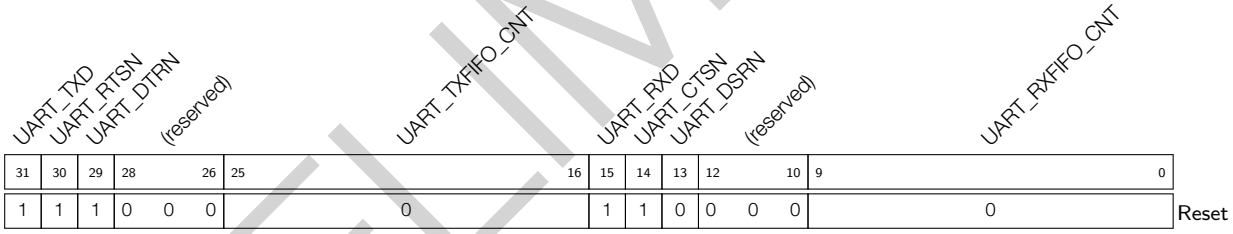
UART_SCLK_EN Set this bit to enable UART TX/RX clock. (R/W)

UART_RST_CORE Write 1 then write 0 to this bit, 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 13.19. UART_STATUS_REG (0x001C)



UART_RXFIFO_CNT Stores the byte number of valid data in RX FIFO. (RO)

UART_DSRN The register represent the level value of the internal UART DSR signal. (RO)

UART_CTSN This register represent the level value of the internal UART CTS signal. (RO)

UART_RXD This register represent the level value of the internal UART RXD signal. (RO)

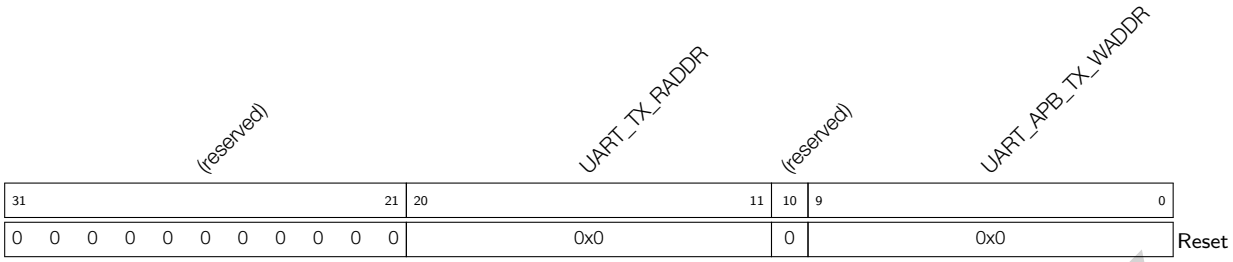
UART_TXFIFO_CNT Stores the byte number of data in TX FIFO. (RO)

UART_DTRN This bit represents the level of the internal UART DTR signal. (RO)

UART_RTSN This bit represents the level of the internal UART RTS signal. (RO)

UART_TXD This bit represents the level of the internal UART TXD signal. (RO)

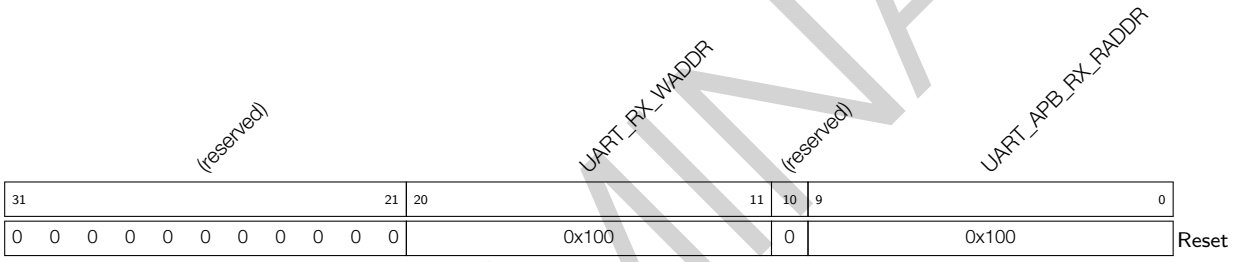
Register 13.20. UART_MEM_TX_STATUS_REG (0x0064)



UART_APB_TX_WADDR This register stores the offset address in TX FIFO when software writes TX FIFO via APB. (RO)

UART_TX_RADDR This register stores the offset address in TX FIFO when TX FSM reads data via Tx_FIFO_Ctrl. (RO)

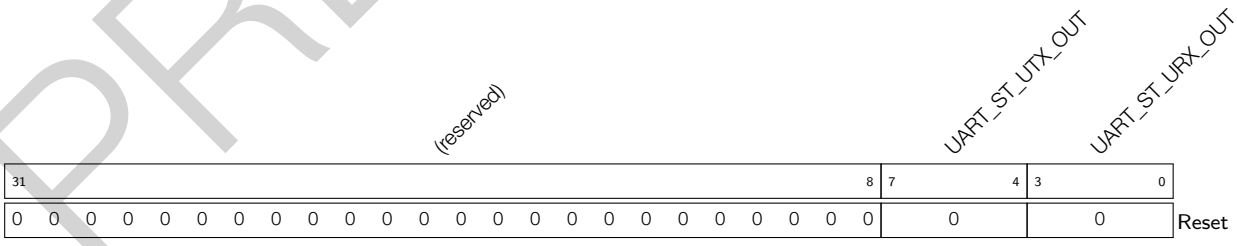
Register 13.21. UART_MEM_RX_STATUS_REG (0x0068)



UART_APB_RX_RADDR This register stores the offset address in RX FIFO when software reads data from RX FIFO via APB. UART0 is 10'h100. UART1 is 10'h180. (RO)

UART_RX_WADDR This register stores the offset address in RX FIFO when Rx_FIFO_Ctrl writes RX FIFO. (RO)

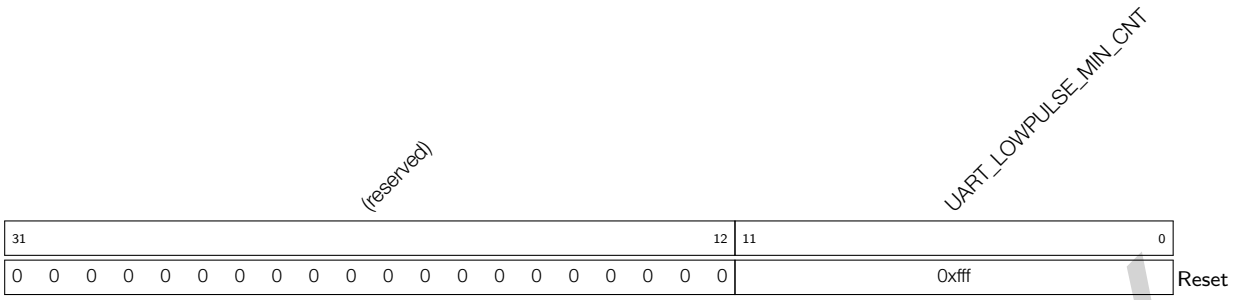
Register 13.22. UART_FSM_STATUS_REG (0x006C)



UART_ST_URX_OUT This is the status register of receiver. (RO)

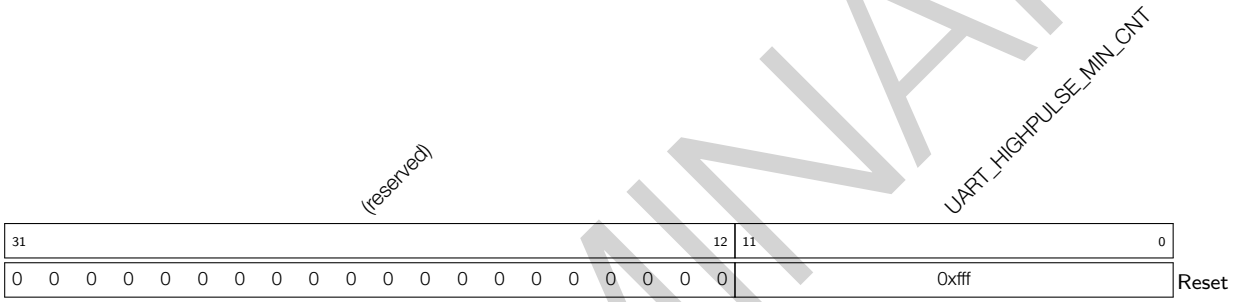
UART_ST_UTX_OUT This is the status register of transmitter. (RO)

Register 13.23. UART_LOWPULSE_REG (0x0028)



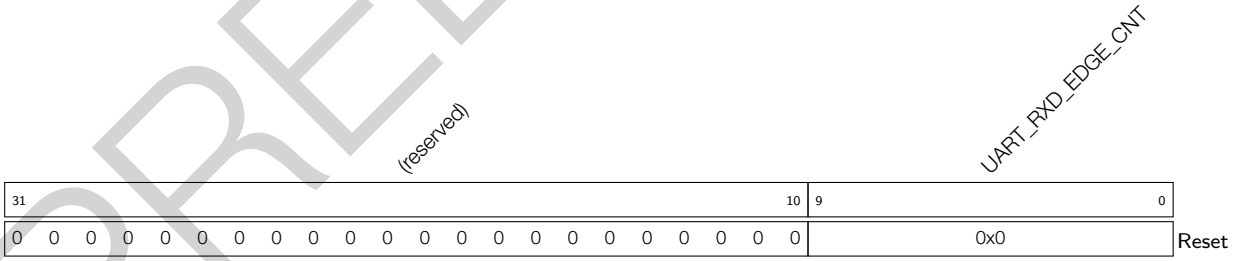
UART_LOWPULSE_MIN_CNT This register stores the value of the minimum duration time of the low level pulse, in the unit of APB_CLK cycles. It is used in baud rate detection. (RO)

Register 13.24. UART_HIGHPULSE_REG (0x002C)



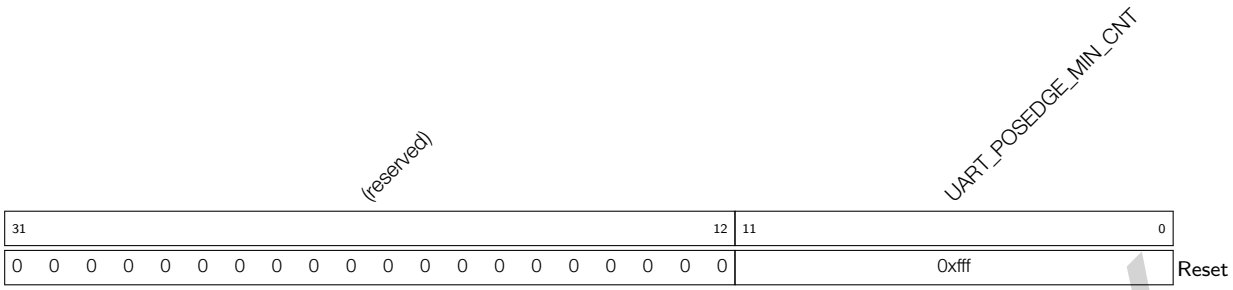
UART_HIGHPULSE_MIN_CNT This register stores the value of the maximum duration time for the high level pulse, in the unit of APB_CLK cycles. It is used in baud rate detection. (RO)

Register 13.25. UART_RXD_CNT_REG (0x0030)



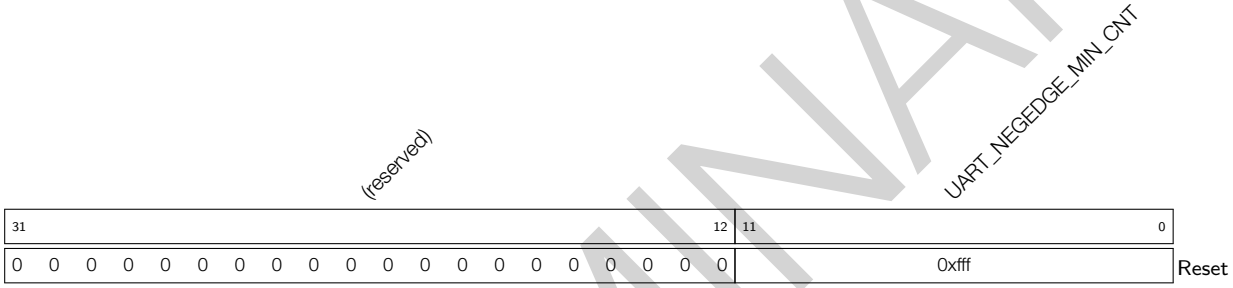
UART_RXD_EDGE_CNT This register stores the count of RXD edge change. It is used in baud rate detection. (RO)

Register 13.26. UART_POSPULSE_REG (0x0070)



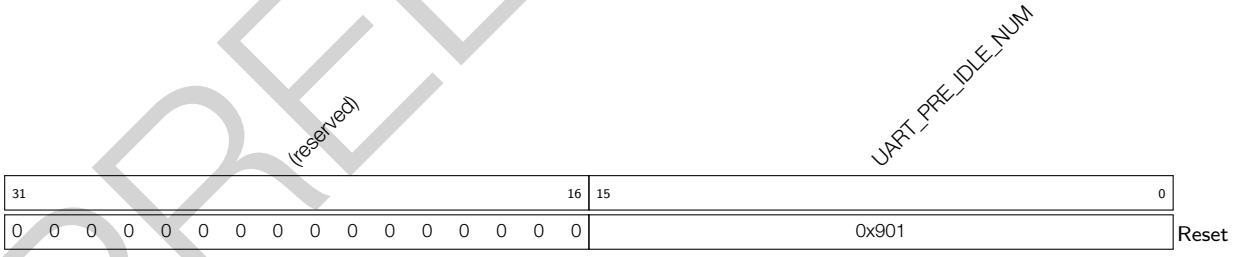
UART_POSEDGE_MIN_CNT This register stores the minimal input clock count between two positive edges. It is used in baud rate detection. (RO)

Register 13.27. UART_NEGPULSE_REG (0x0074)



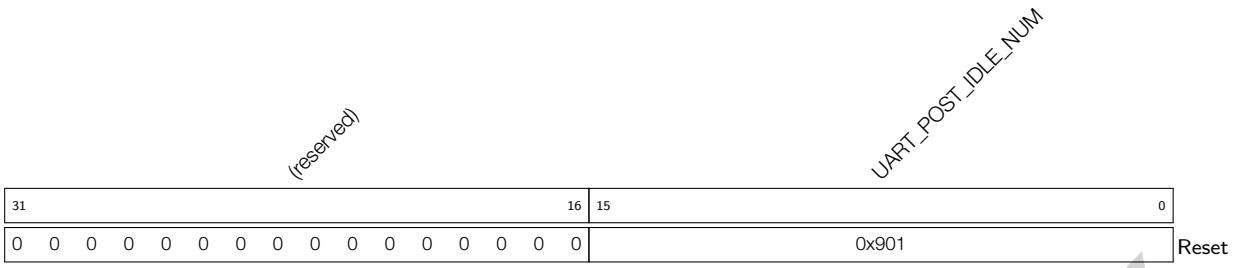
UART_NEGEDGE_MIN_CNT This register stores the minimal input clock count between two negative edges. It is used in baud rate detection. (RO)

Register 13.28. UART_AT_CMD_PRECNT_REG (0x0050)



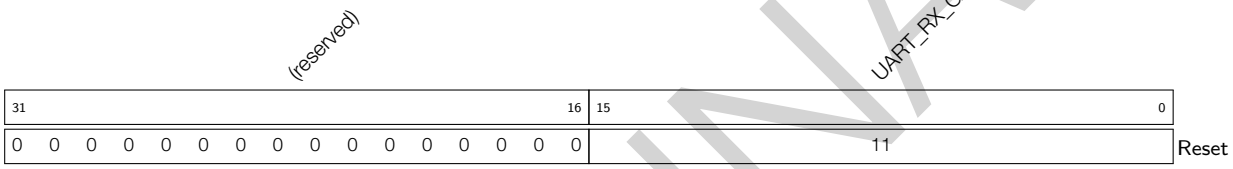
UART_PRE_IDLE_NUM This register is used to configure the idle duration time before the first AT_CMD is received by receiver, in the unit of bit time (the time it takes to transfer one bit). (R/W)

Register 13.29. UART_AT_CMD_POSTCNT_REG (0x0054)



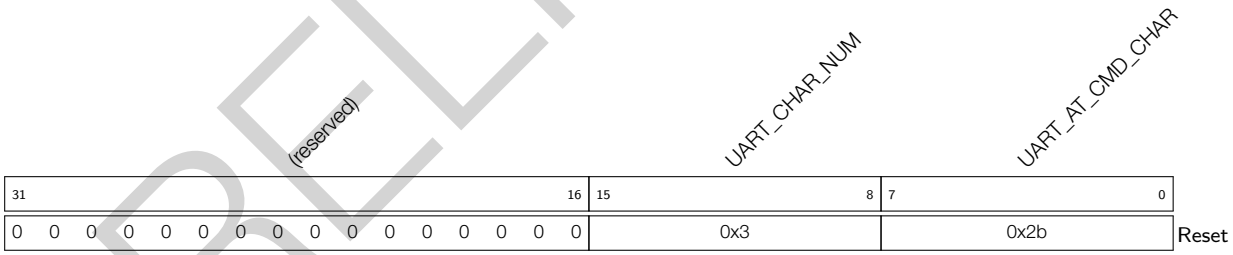
UART_POST_IDLE_NUM This register is used to configure the duration time between the last AT_CMD and the next data, in the unit of bit time (the time it takes to transfer one bit). (R/W)

Register 13.30. UART_AT_CMD_GAPTOUT_REG (0x0058)



UART_RX_GAP_TOUT This register is used to configure the duration time between the AT_CMD chars, in the unit of bit time (the time it takes to transfer one bit). (R/W)

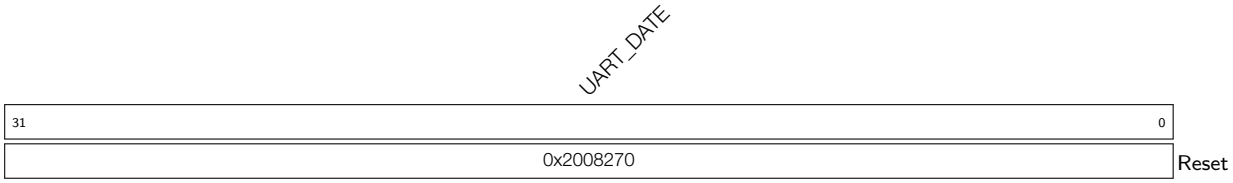
Register 13.31. UART_AT_CMD_CHAR_REG (0x005C)



UART_AT_CMD_CHAR This register is used to configure the content of AT_CMD character. (R/W)

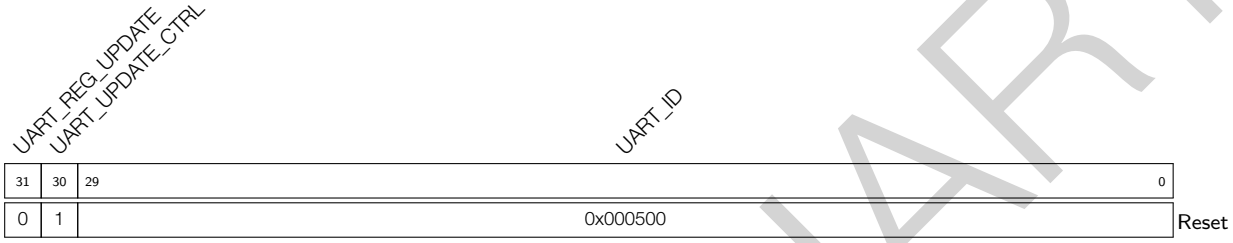
UART_CHAR_NUM This register is used to configure the number of continuous AT_CMD chars received by receiver. (R/W)

Register 13.32. UART_DATE_REG (0x007C)



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

Register 13.33. UART_ID_REG (0x0080)



UART_ID This register is used to configure the UART_ID. (R/W)

UART_UPDATE_CTRL This bit used to control register synchronize mode. This register must be cleared before write 1 to UART_REG_UPDATE to synchronize configure value to UART core clock. (R/W)

UART_REG_UPDATE Software write 1 would synchronize registers into UART Core clock domain and would be cleared by hardware after synchronization is done. (R/W/SC)

Register 13.34. UHCI_CONF0_REG (0x0000)

UHCI_TX_RST Write 1, then write 0 to this bit to reset decode state machine. (R/W)

UHCI_RX_RST Write 1, then write 0 to this bit to reset encode state machine. (R/W)

UHCI_UART0_CE Set this bit to link up HCI and UART0. (R/W)

UHCI_UART1_CE Set this bit to link up HCI and UART1. (R/W)

UHCI_SEPER_EN Set this bit to separate the data frame using a special char. (R/W)

UHCI_HEAD_EN Set this bit to encode the data packet with a formatting header. (R/W)

UHCI_CRC_REC_EN Set this bit to enable UHCI to receive the 16 bit CRC. (R/W)

UHCI_UART_IDLE_EOF_EN If this bit is set to 1, UHCI will end the payload receiving process when UART has been in idle state. (R/W)

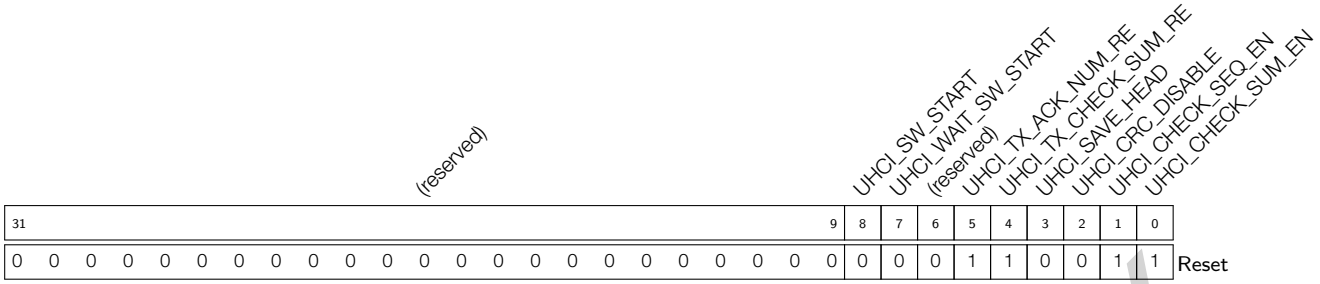
UHCI_LEN_EOF_EN If this bit is set to 1, UHCI decoder receiving payload data is end when the receiving byte count has reached the specified value. The value is payload length indicated by UHCI packet header when UHCI_HEAD_EN is 1 or the value is configuration value when UHCI_HEAD_EN is 0. If this bit is set to 0, UHCI decoder receiving payload data is end when 0xc0 is received. (R/W)

UHCI_ENCODE_CRC_EN Set this bit to enable data integrity checking by appending a 16 bit CCITT-CRC to end of the payload. (R/W)

UHCI_CLK_EN 1'b1: Force clock on for register. 1'b0: Support clock only when application writes registers. (R/W)

UHCI_UART_RX_BRK_EOF_EN If this bit is set to 1, UHCI will end payload receive process when NULL frame is received by UART. (R/W)

Register 13.35. UHCI_CONF1_REG (0x0014)



UHCI_CHECK_SUM_EN This is the enable bit to check header checksum when UHCI receives a data packet. (R/W)

UHCI_CHECK_SEQ_EN This is the enable bit to check sequence number when UHCI receives a data packet. (R/W)

UHCI_CRC_DISABLE Set this bit to support CRC calculation. Data Integrity Check Present bit in UHCI packet frame should be 1. (R/W)

UHCI_SAVE_HEAD Set this bit to save the packet header when HCI receives a data packet. (R/W)

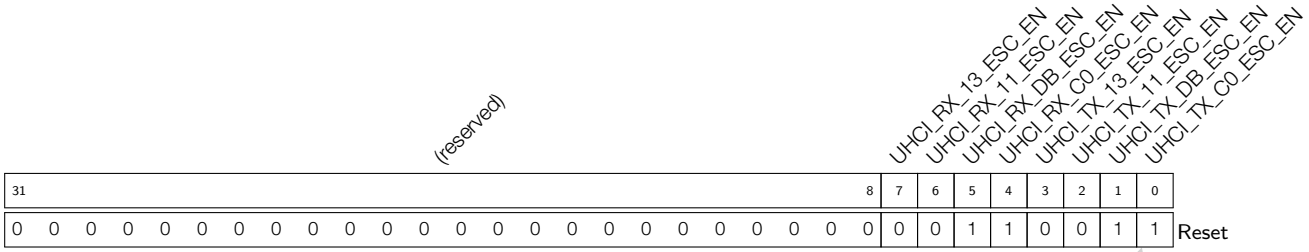
UHCI_TX_CHECK_SUM_RE Set this bit to encode the data packet with a checksum. (R/W)

UHCI_TX_ACK_NUM_RE Set this bit to encode the data packet with an acknowledgment when a reliable packet is to be transmit. (R/W)

UHCI_WAIT_SW_START The uhci-encoder will jump to ST_SW_WAIT status if this register is set to 1. (R/W)

UHCI_SW_START If current UHCI_ENCODE_STATE is ST_SW_WAIT, the UHCI will start to send data packet out when this bit is set to 1. (R/W/SC)

Register 13.36. UHCI_ESCAPE_CONF_REG (0x0020)



UHCI_TX_C0_ESC_EN Set this bit to enable decoding char 0xc0 when DMA receives data. (R/W)

UHCI_TX_DB_ESC_EN Set this bit to enable decoding char 0xdb when DMA receives data. (R/W)

UHCI_TX_11_ESC_EN Set this bit to enable decoding flow control char 0x11 when DMA receives data. (R/W)

UHCI_TX_13_ESC_EN Set this bit to enable decoding flow control char 0x13 when DMA receives data. (R/W)

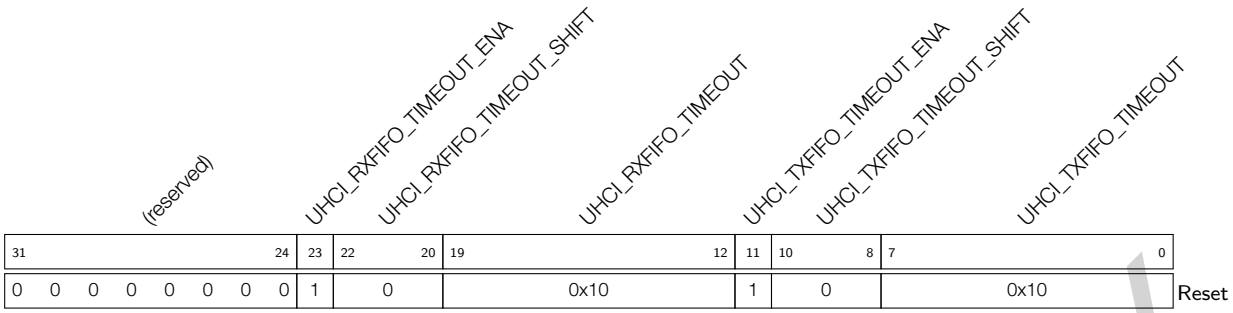
UHCI_RX_C0_ESC_EN Set this bit to enable replacing 0xc0 by special char when DMA sends data. (R/W)

UHCI_RX_DB_ESC_EN Set this bit to enable replacing 0xdb by special char when DMA sends data. (R/W)

UHCI_RX_11_ESC_EN Set this bit to enable replacing flow control char 0x11 by special char when DMA sends data. (R/W)

UHCI_RX_13_ESC_EN Set this bit to enable replacing flow control char 0x13 by special char when DMA sends data. (R/W)

Register 13.37. UHCI_HUNG_CONF_REG (0x0024)



UHCI_TXFIFO_TIMEOUT This register stores the timeout value. It will produce the UHCI_TX_HUNG_INT interrupt when DMA takes more time to receive data. (R/W)

UHCI_TXFIFO_TIMEOUT_SHIFT This register is used to configure the tick count maximum value. (R/W)

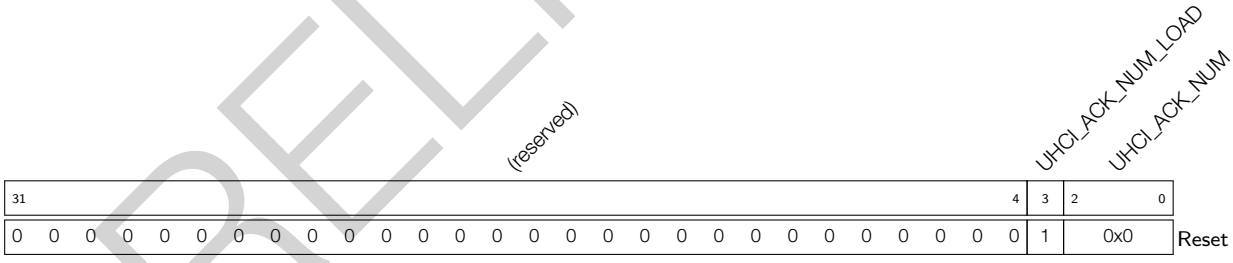
UHCI_TXFIFO_TIMEOUT_ENA This is the enable bit for Tx-FIFO receive-data timeout. (R/W)

UHCI_RXFIFO_TIMEOUT This register stores the timeout value. It will produce the UHCI_RX_HUNG_INT interrupt when DMA takes more time to read data from RAM. (R/W)

UHCI_RXFIFO_TIMEOUT_SHIFT This register is used to configure the tick count maximum value. (R/W)

UHCI_RXFIFO_TIMEOUT_ENA This is the enable bit for DMA send-data timeout. (R/W)

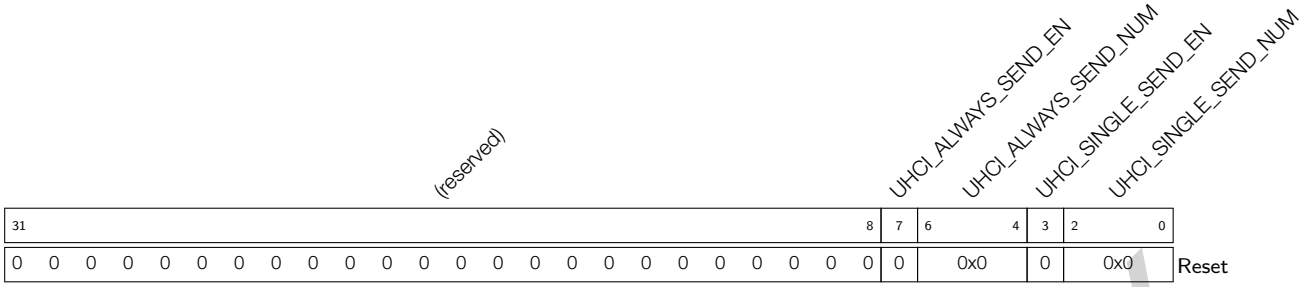
Register 13.38. UHCI_ACK_NUM_REG (0x0028)



UHCI_ACK_NUM This ACK number used in software flow control. (R/W)

UHCI_ACK_NUM_LOAD Set this bit to 1, the value configured by UHCI_ACK_NUM would be loaded. (WT)

Register 13.39. UHCI_QUICK_SENT_REG (0x0030)



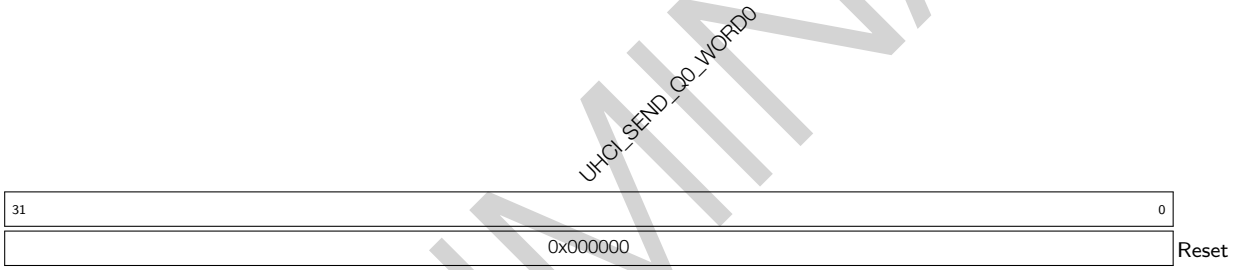
UHCI_SINGLE_SEND_NUM This register is used to specify the single_send register. (R/W)

UHCI_SINGLE_SEND_EN Set this bit to enable single_send mode to send short packet. (R/W/SC)

UHCI_ALWAYS_SEND_NUM This register is used to specify the always_send register. (R/W)

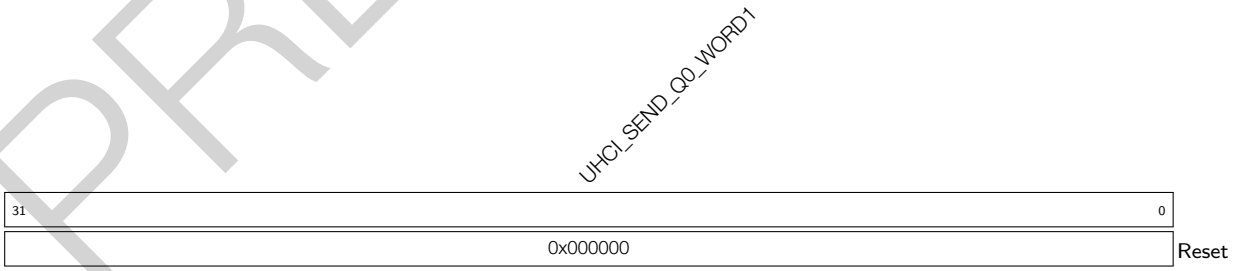
UHCI_ALWAYS_SEND_EN Set this bit to enable always_send mode to send short packet. (R/W)

Register 13.40. UHCI_REG_Q0_WORD0_REG (0x0034)



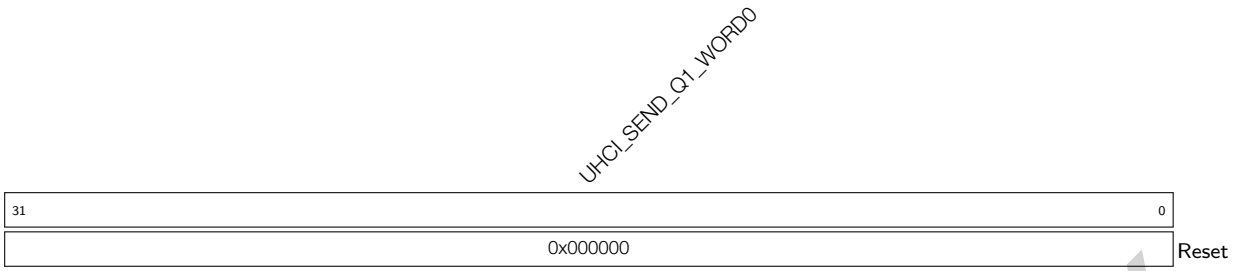
UHCI_SEND_Q0_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.41. UHCI_REG_Q0_WORD1_REG (0x0038)



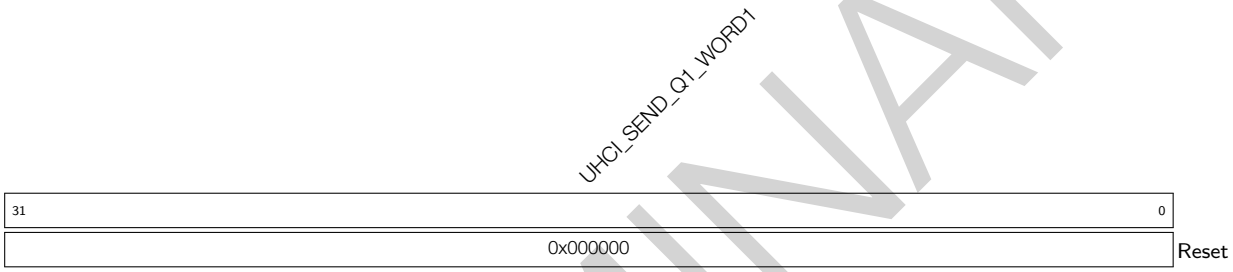
UHCI_SEND_Q0_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.42. UHCI_REG_Q1_WORD0_REG (0x003C)



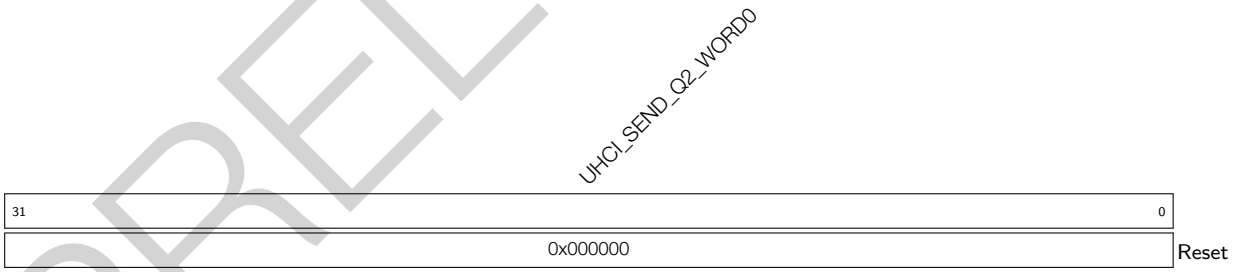
UHCI_SEND_Q1_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.43. UHCI_REG_Q1_WORD1_REG (0x0040)



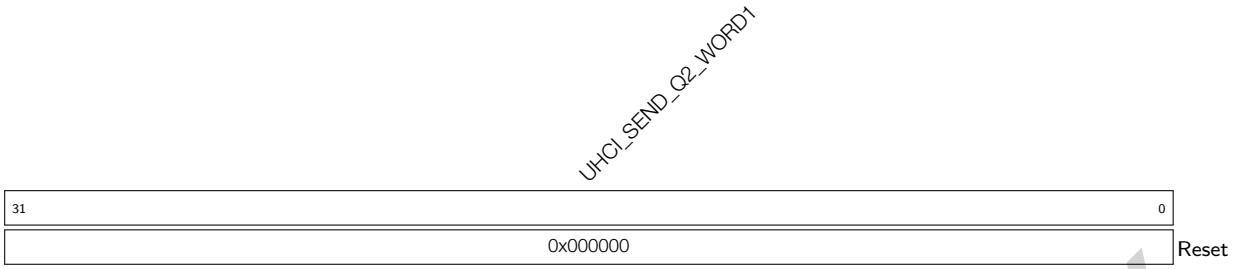
UHCI_SEND_Q1_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.44. UHCI_REG_Q2_WORD0_REG (0x0044)



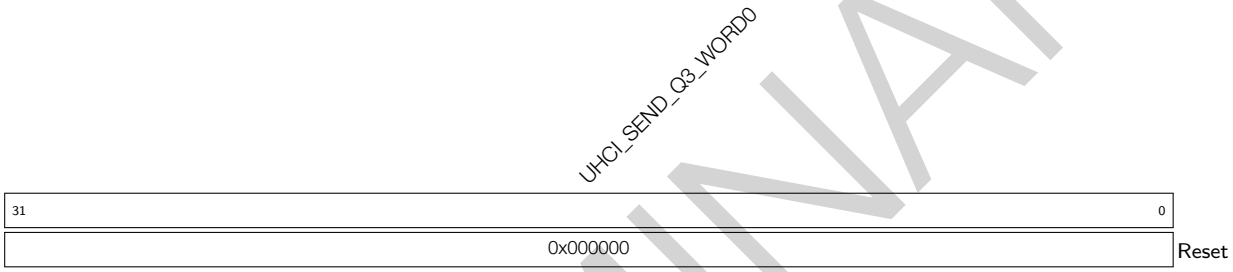
UHCI_SEND_Q2_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.45. UHCI_REG_Q2_WORD1_REG (0x0048)



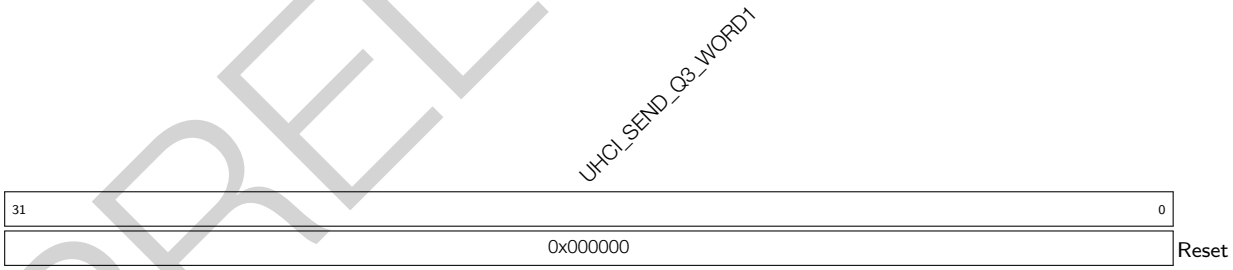
UHCI_SEND_Q2_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.46. UHCI_REG_Q3_WORD0_REG (0x004C)



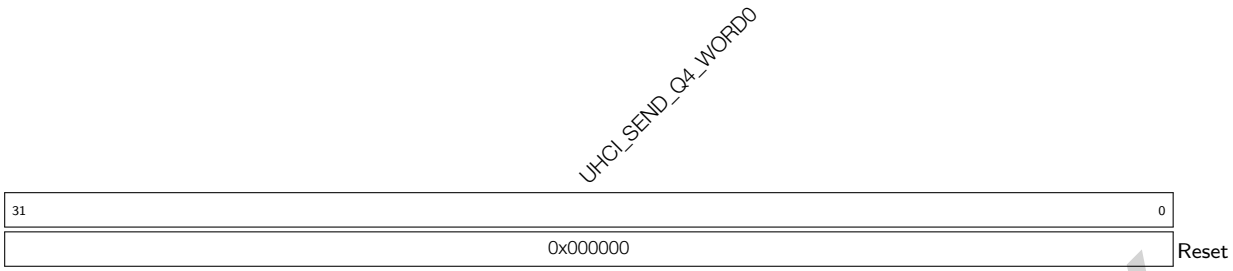
UHCI_SEND_Q3_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.47. UHCI_REG_Q3_WORD1_REG (0x0050)



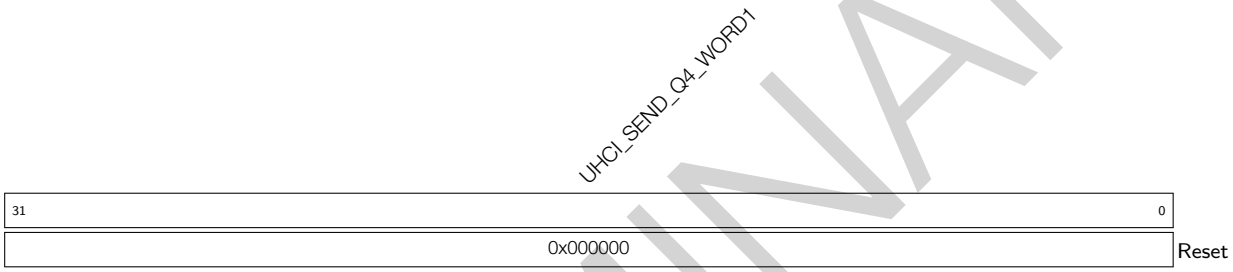
UHCI_SEND_Q3_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.48. UHCI_REG_Q4_WORD0_REG (0x0054)



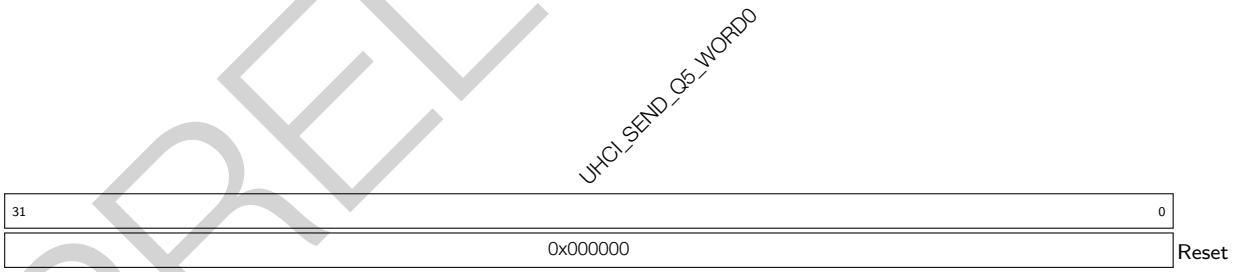
UHCI_SEND_Q4_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.49. UHCI_REG_Q4_WORD1_REG (0x0058)



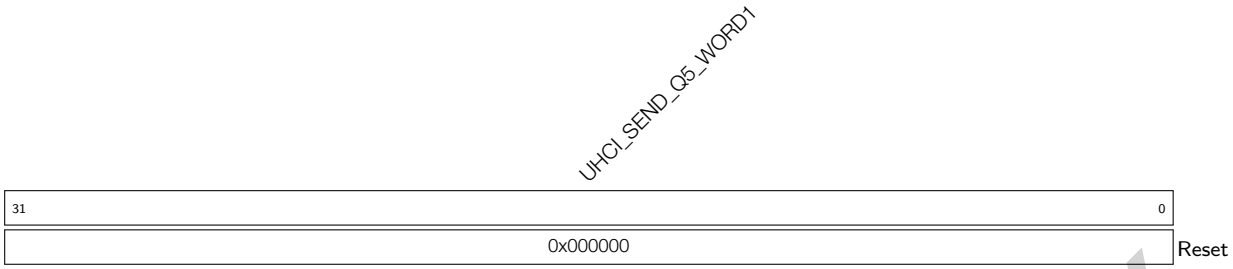
UHCI_SEND_Q4_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.50. UHCI_REG_Q5_WORD0_REG (0x005C)



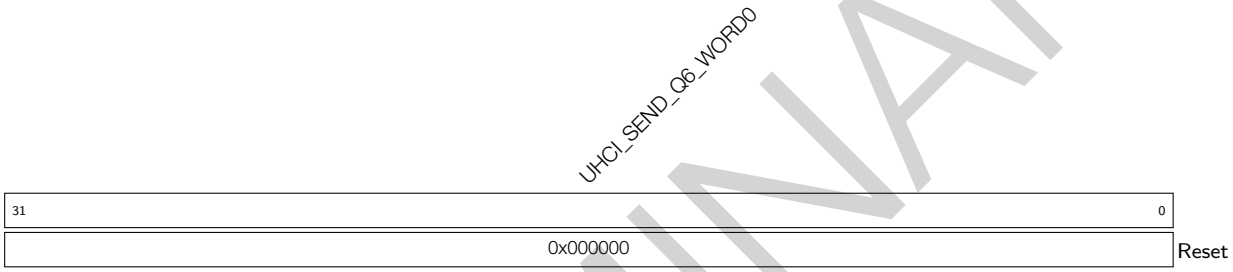
UHCI_SEND_Q5_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.51. UHCI_REG_Q5_WORD1_REG (0x0060)



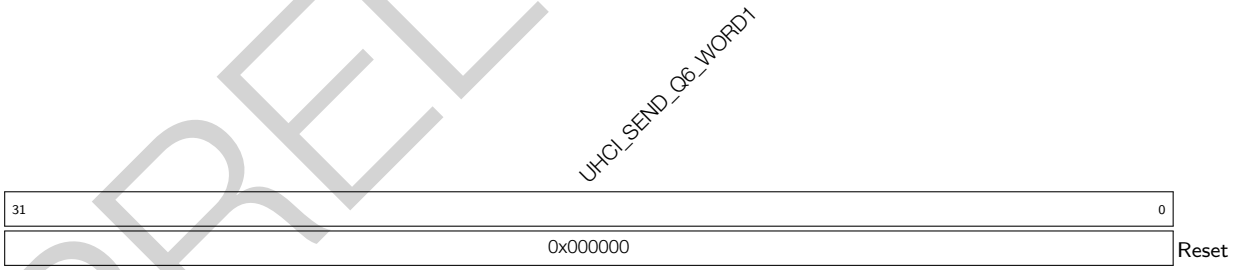
UHCI_SEND_Q5_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.52. UHCI_REG_Q6_WORD0_REG (0x0064)



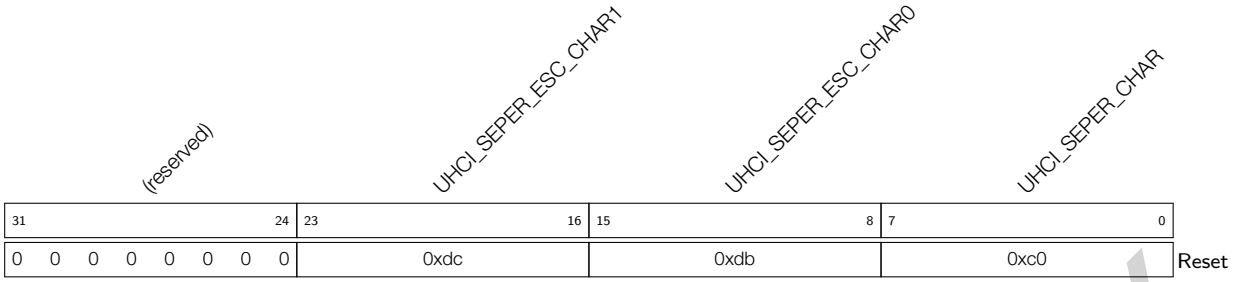
UHCI_SEND_Q6_WORD0 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.53. UHCI_REG_Q6_WORD1_REG (0x0068)



UHCI_SEND_Q6_WORD1 This register is used as a quick_sent register when specified by UHCI_ALWAYS_SEND_NUM or UHCI_SINGLE_SEND_NUM. (R/W)

Register 13.54. UHCI_ESC_CONF0_REG (0x006C)

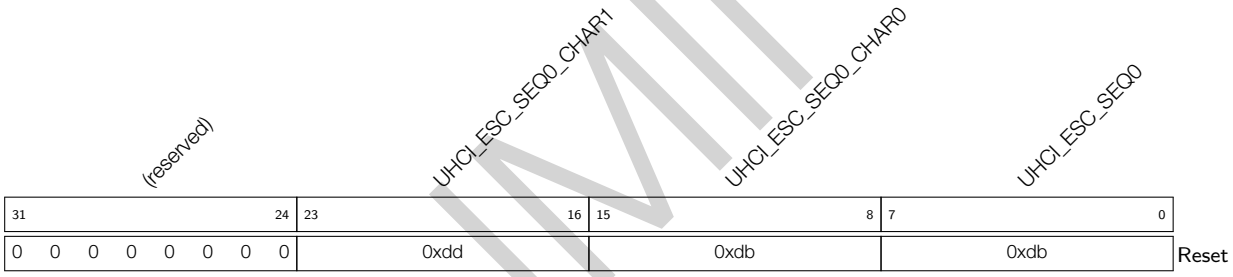


UHCI_SEPER_CHAR This register is used to define the separate char that need to be encoded, default is 0xc0. (R/W)

UHCI_SEPER_ESC_CHAR0 This register is used to define the first char of slip escape sequence when encoding the separate char, default is 0xdb. (R/W)

UHCI_SEPER_ESC_CHAR1 This register is used to define the second char of slip escape sequence when encoding the separate char, default is 0xdc. (R/W)

Register 13.55. UHCI_ESC_CONF1_REG (0x0070)

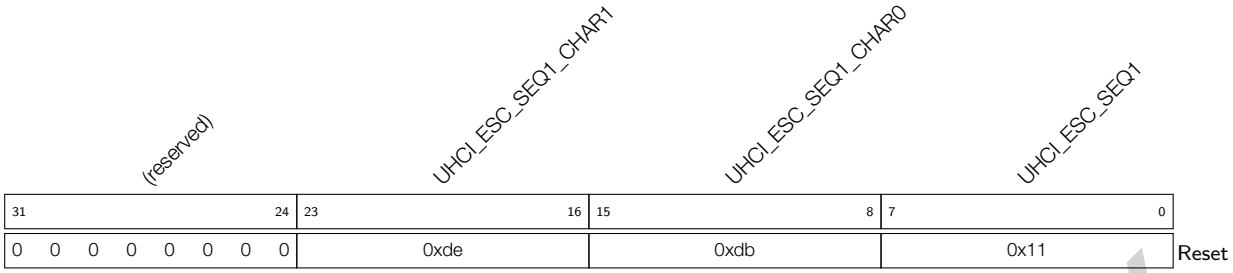


UHCI_ESC_SEQ0 This register is used to define a char that need to be encoded, default is 0xdb that used as the first char of slip escape sequence. (R/W)

UHCI_ESC_SEQ0_CHAR0 This register is used to define the first char of slip escape sequence when encoding the UHCI_ESC_SEQ0, default is 0xdb. (R/W)

UHCI_ESC_SEQ0_CHAR1 This register is used to define the second char of slip escape sequence when encoding the UHCI_ESC_SEQ0, default is 0xdd. (R/W)

Register 13.56. UHCI_ESC_CONF2_REG (0x0074)

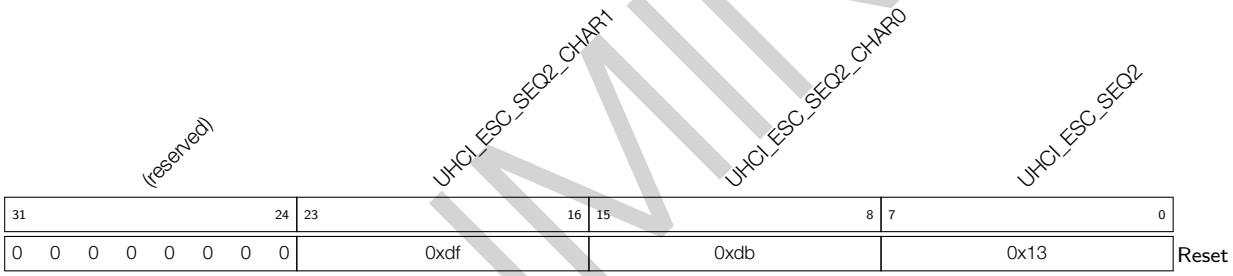


UHCI_ESC_SEQ1 This register is used to define a char that need to be encoded, default is 0x11 that used as flow control char. (R/W)

UHCI_ESC_SEQ1_CHAR0 This register is used to define the first char of slip escape sequence when encoding the UHCI_ESC_SEQ1, default is 0xdb. (R/W)

UHCI_ESC_SEQ1_CHAR1 This register is used to define the second char of slip escape sequence when encoding the UHCI_ESC_SEQ1, default is 0xde. (R/W)

Register 13.57. UHCI_ESC_CONF3_REG (0x0078)

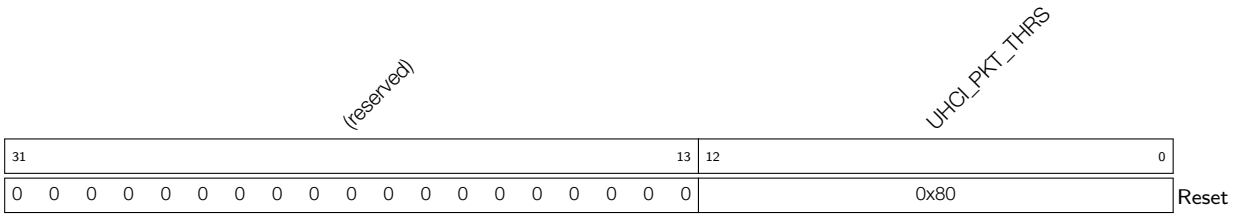


UHCI_ESC_SEQ2 This register is used to define a char that need to be decoded, default is 0x13 that used as flow control char. (R/W)

UHCI_ESC_SEQ2_CHAR0 This register is used to define the first char of slip escape sequence when encoding the UHCI_ESC_SEQ2, default is 0xdb. (R/W)

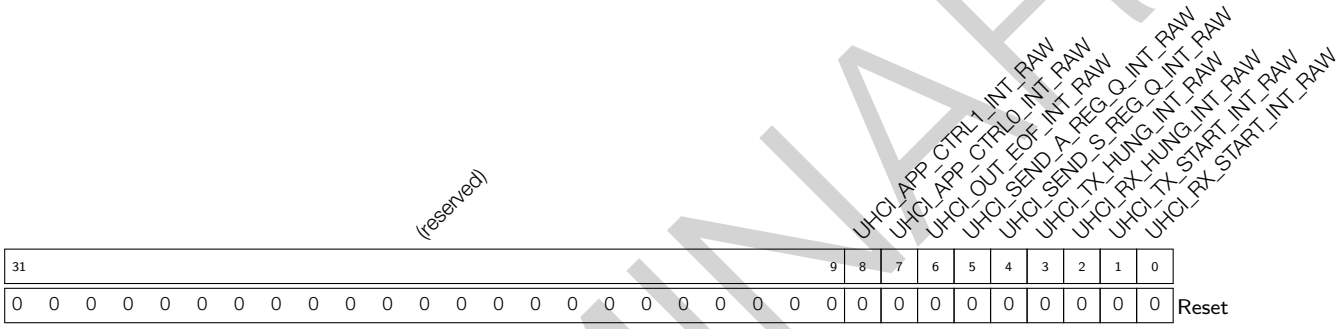
UHCI_ESC_SEQ2_CHAR1 This register is used to define the second char of slip escape sequence when encoding the UHCI_ESC_SEQ2, default is 0xdf. (R/W)

Register 13.58. UHCI_PKT_THRES_REG (0x007C)



UHCI_PKT_THRS This register is used to configure the maximum value of the packet length when UHCI_HEAD_EN is 0. (R/W)

Register 13.59. UHCI_INT_RAW_REG (0x0004)



UHCI_RX_START_INT_RAW This is the interrupt raw bit. Triggered when a separator char has been sent. (R/WTC/SS)

UHCI_TX_START_INT_RAW This is the interrupt raw bit. Triggered when UHCI detects a separator char. (R/WTC/SS)

UHCI_RX_HUNG_INT_RAW This is the interrupt raw bit. Triggered when UHCI takes more time to receive data than configure value. (R/WTC/SS)

UHCI_TX_HUNG_INT_RAW This is the interrupt raw bit. Triggered when UHCI takes more time to read data from RAM than the configured value. (R/WTC/SS)

UHCI_SEND_S_REG_Q_INT_RAW This is the interrupt raw bit. Triggered when UHCI has sent out a short packet using single_send registers. (R/WTC/SS)

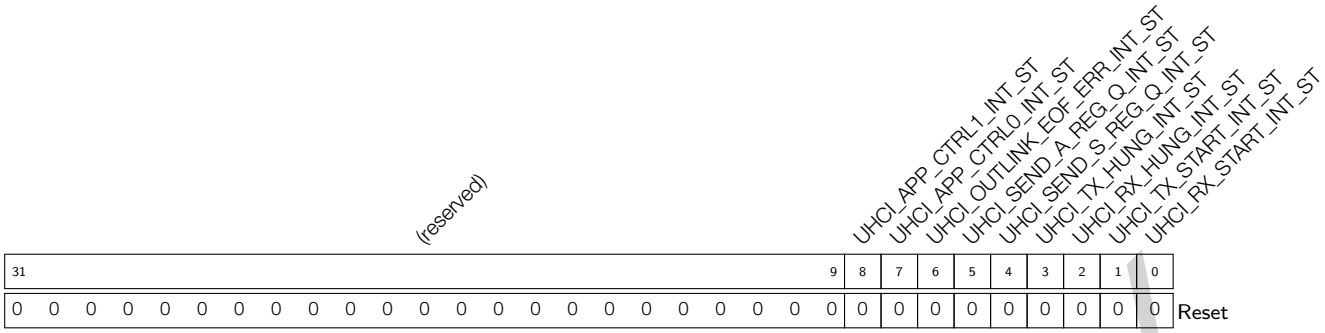
UHCI_SEND_A_REG_Q_INT_RAW This is the interrupt raw bit. Triggered when UHCI has sent out a short packet using always_send registers. (R/WTC/SS)

UHCI_OUT_EOF_INT_RAW This is the interrupt raw bit. Triggered when there are some errors in EOF in the transmit data. (R/WTC/SS)

UHCI_APP_CTRL0_INT_RAW This is the interrupt raw bit. Triggered when set this bit to 1. Clear it when write 0 to this bit. (R/W)

UHCI_APP_CTRL1_INT_RAW This is the interrupt raw bit. Triggered when set this bit to 1. Clear it when write 0 to this bit. (R/W)

Register 13.60. UHCI_INT_ST_REG (0x0008)



UHCI_RX_START_INT_ST This is the masked interrupt bit for UHCI_RX_START_INT interrupt when UHCI_RX_START_INT_ENA is set to 1. (RO)

UHCI_TX_START_INT_ST This is the masked interrupt bit for UHCI_TX_START_INT interrupt when UHCI_TX_START_INT_ENA is set to 1. (RO)

UHCI_RX_HUNG_INT_ST This is the masked interrupt bit for UHCI_RX_HUNG_INT interrupt when UHCI_RX_HUNG_INT_ENA is set to 1. (RO)

UHCI_TX_HUNG_INT_ST This is the masked interrupt bit for UHCI_TX_HUNG_INT interrupt when UHCI_TX_HUNG_INT_ENA is set to 1. (RO)

UHCI_SEND_S_REG_Q_INT_ST This is the masked interrupt bit for UHCI_SEND_S_REQ_Q_INT interrupt when UHCI_SEND_S_REQ_Q_INT_ENA is set to 1. (RO)

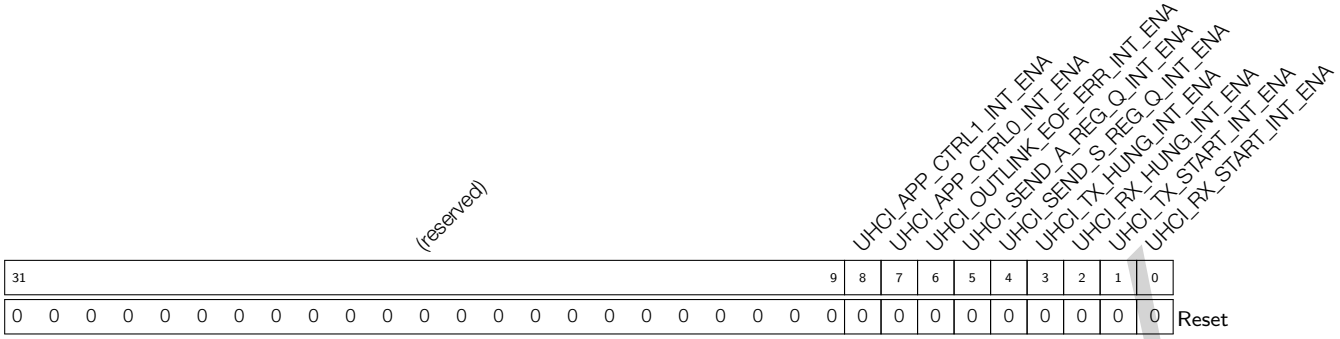
UHCI_SEND_A_REG_Q_INT_ST This is the masked interrupt bit for UHCI_SEND_A_REQ_Q_INT interrupt when UHCI_SEND_A_REQ_Q_INT_ENA is set to 1. (RO)

UHCI_OUTLINK_EOF_ERR_INT_ST This is the masked interrupt bit for UHCI_OUTLINK_EOF_ERR_INT interrupt when UHCI_OUTLINK_EOF_ERR_INT_ENA is set to 1. (RO)

UHCI_APP_CTRL0_INT_ST This is the masked interrupt bit for UHCI_APP_CTRL0_INT interrupt when UHCI_APP_CTRL0_INT_ENA is set to 1. (RO)

UHCI_APP_CTRL1_INT_ST This is the masked interrupt bit for UHCI_APP_CTRL1_INT interrupt when UHCI_APP_CTRL1_INT_ENA is set to 1. (RO)

Register 13.61. UHCI_INT_ENA_REG (0x000C)



UHCI_RX_START_INT_ENA This is the interrupt enable bit for UHCI_RX_START_INT interrupt. (R/W)

UHCI_TX_START_INT_ENA This is the interrupt enable bit for UHCI_TX_START_INT interrupt. (R/W)

UHCI_RX_HUNG_INT_ENA This is the interrupt enable bit for UHCI_RX_HUNG_INT interrupt. (R/W)

UHCI_TX_HUNG_INT_ENA This is the interrupt enable bit for UHCI_TX_HUNG_INT interrupt. (R/W)

UHCI_SEND_S_REG_Q_INT_ENA This is the interrupt enable bit for UHCI_SEND_S_REQ_Q_INT interrupt. (R/W)

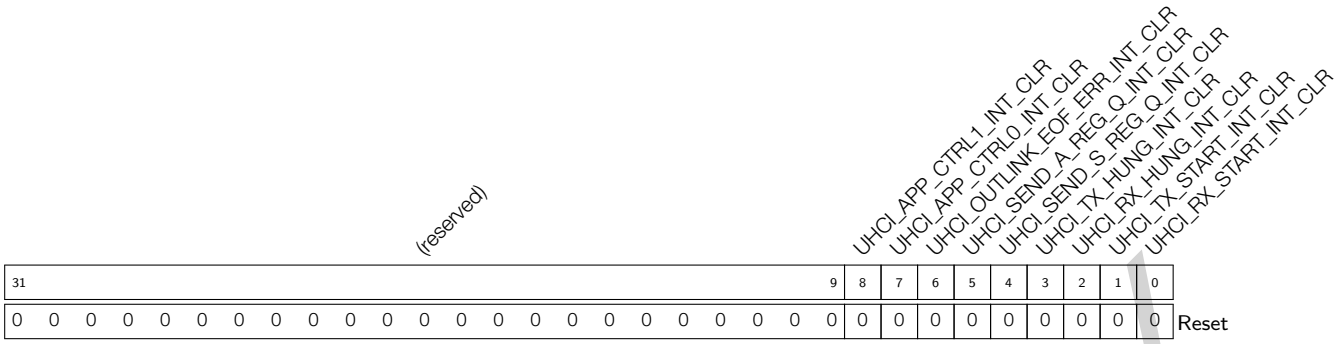
UHCI_SEND_A_REG_Q_INT_ENA This is the interrupt enable bit for UHCI_SEND_A_REQ_Q_INT interrupt. (R/W)

UHCI_OUTLINK_EOF_ERR_INT_ENA This is the interrupt enable bit for UHCI_OUTLINK_EOF_ERR_INT interrupt. (R/W)

UHCI_APP_CTRL0_INT_ENA This is the interrupt enable bit for UHCI_APP_CTRL0_INT interrupt. (R/W)

UHCI_APP_CTRL1_INT_ENA This is the interrupt enable bit for UHCI_APP_CTRL1_INT interrupt. (R/W)

Register 13.62. UHCI_INT_CLR_REG (0x0010)



UHCI_RX_START_INT_CLR Set this bit to clear UHCI_RX_START_INT interrupt. (WT)

UHCI_TX_START_INT_CLR Set this bit to clear UHCI_TX_START_INT interrupt. (WT)

UHCI_RX_HUNG_INT_CLR Set this bit to clear UHCI_RX_HUNG_INT interrupt. (WT)

UHCI_TX_HUNG_INT_CLR Set this bit to clear UHCI_TX_HUNG_INT interrupt. (WT)

UHCI_SEND_S_REG_Q_INT_CLR Set this bit to clear UHCI_SEND_S_REQ_Q_INT interrupt. (WT)

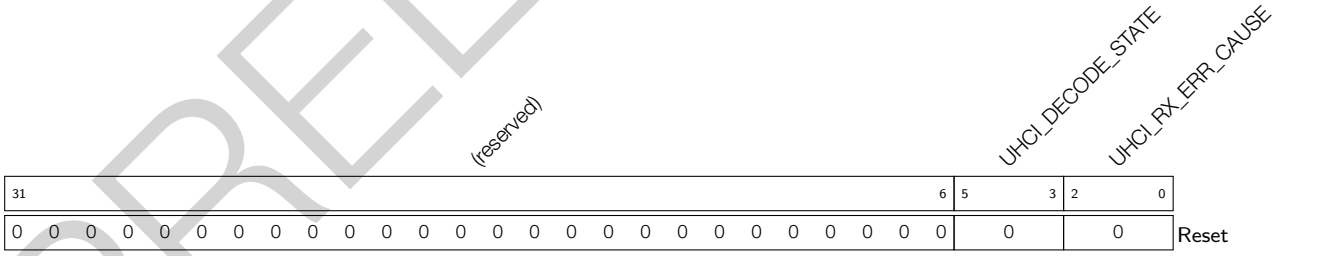
UHCI_SEND_A_REG_Q_INT_CLR Set this bit to clear UHCI_SEND_A_REQ_Q_INT interrupt. (WT)

UHCI_OUTLINK_EOF_ERR_INT_CLR Set this bit to clear UHCI_OUTLINK_EOF_ERR_INT interrupt. (WT)

UHCI_APP_CTRL0_INT_CLR Set this bit to clear UHCI_APP_CTRL0_INT interrupt. (WT)

UHCI_APP_CTRL1_INT_CLR Set this bit to clear UHCI_APP_CTRL1_INT interrupt. (WT)

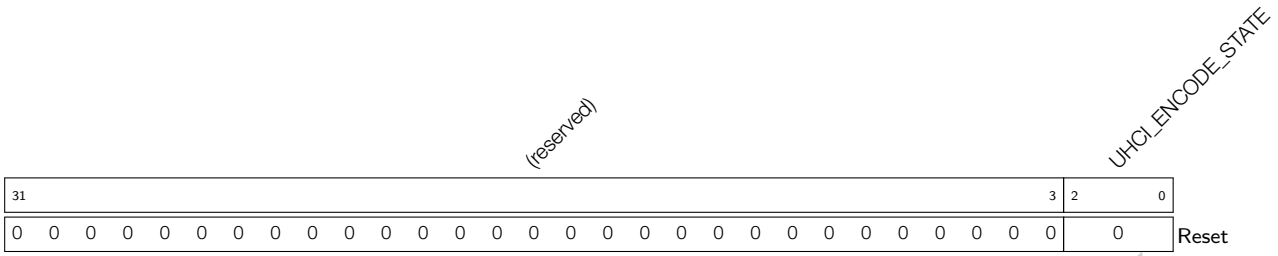
Register 13.63. UHCI_STATE0_REG (0x0018)



UHCI_RX_ERR_CAUSE This register indicates the error type when DMA has received a packet with error. 3'b001: Checksum error in HCI packet; 3'b010: Sequence number error in HCI packet; 3'b011: CRC bit error in HCI packet; 3'b100: 0xc0 is found but received HCI packet is not end; 3'b101: 0xc0 is not found when receiving HCI packet is end; 3'b110: CRC check error. (RO)

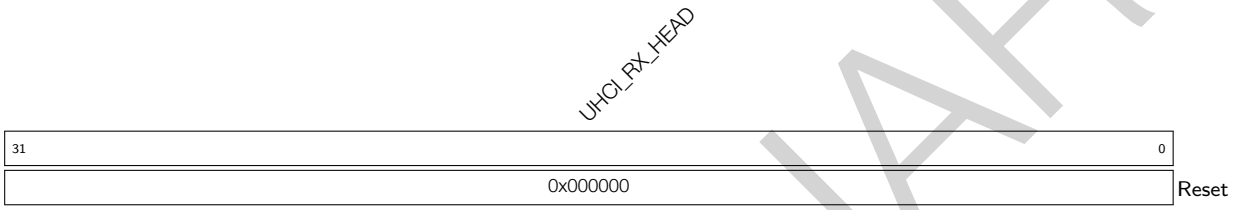
UHCI_DECODE_STATE UHCI decoder status. (RO)

Register 13.64. UHCI_STATE1_REG (0x001C)



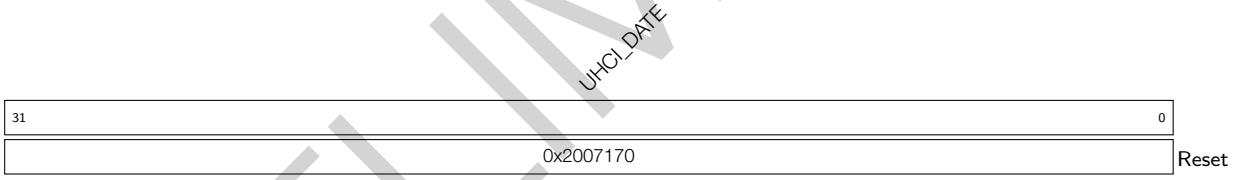
UHCI_ENCODE_STATE UHCI encoder status. (RO)

Register 13.65. UHCI_RX_HEAD_REG (0x002C)



UHCI_RX_HEAD This register stores the header of the current received packet. (RO)

Register 13.66. UHCI_DATE_REG (0x0080)



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

14 Two-wire Automotive Interface (TWAI)

The Two-wire Automotive Interface (TWAI®) is a multi-master, multi-cast communication protocol with functions such as error detection and signaling and inbuilt message priorities and arbitration. The TWAI protocol is suited for automotive and industrial applications (see Section 14.2 for more details).

ESP32-C3 contains a TWAI controller that can be connected to the TWAI bus via an external transceiver. The TWAI controller contains numerous advanced features, and can be utilized in a wide range of use cases such as automotive products, industrial automation controls, building automation, etc.

14.1 Features

The TWAI controller on ESP32-C3 supports the following features:

- Compatible with ISO 11898-1 protocol
- Supports Standard Frame Format (11-bit ID) and Extended Frame Format (29-bit ID)
- Bit rates from 1 Kbit/s to 1 Mbit/s
- Multiple modes of operation
  - Normal
  - Listen-only (no influence on bus)
  - Self-test (no acknowledgment required during data transmission)
- 64-byte Receive FIFO
- Special transmissions
  - Single-shot transmissions (does not automatically re-transmit upon error)
  - Self Reception (the TWAI controller transmits and receives messages simultaneously)
- Acceptance Filter (supports single and dual filter modes)
- Error detection and handling
  - Error Counters
  - Configurable Error Warning Limit
  - Error Code Capture
  - Arbitration Lost Capture

14.2 Functional Protocol

14.2.1 TWAI Properties

The TWAI protocol connects two or more nodes in a bus network, and allows nodes to exchange messages in a latency bounded manner. A TWAI bus has the following properties.

Single Channel and Non-Return-to-Zero: The bus consists of a single channel to carry bits, and thus communication is half-duplex. Synchronization is also implemented in this channel, so extra channels (e.g., clock

or enable) are not required. The bit stream of a TWAI message is encoded using the Non-Return-to-Zero (NRZ) method.

Bit Values: The single channel can either be in a dominant or recessive state, representing a logical 0 and a logical 1 respectively. A node transmitting data in a dominant state always overrides the other node transmitting data in a recessive state. The physical implementation on the bus is left to the application level to decide (e.g., differential pair or a single wire).

Bit Stuffing: Certain fields of TWAI messages are bit-stuffed. A transmitter that transmits five consecutive bits of the same value (e.g., dominant value or recessive value) should automatically insert a complementary bit. Likewise, a receiver that receives five consecutive bits should treat the next bit as a stuffed bit. Bit stuffing is applied to the following fields: SOF, arbitration field, control field, data field, and CRC sequence (see Section 14.2.2 for more details).

Multi-cast: All nodes receive the same bits as they are connected to the same bus. Data is consistent across all nodes unless there is a bus error (see Section 14.2.3 for more details).

Multi-master: Any node can initiate a transmission. If a transmission is already ongoing, a node will wait until the current transmission is over before initiating a new transmission.

Message Priority and Arbitration: If two or more nodes simultaneously initiate a transmission, the TWAI protocol ensures that one node will win arbitration of the bus. The arbitration field of the message transmitted by each node is used to determine which node will win arbitration.

Error Detection and Signaling: Each node actively monitors the bus for errors, and signals the detected errors by transmitting an error frame.

Fault Confinement: Each node maintains a set of error counters that are incremented/decremented according to a set of rules. When the error counters surpass a certain threshold, the node will automatically eliminate itself from the network by switching itself off.

Configurable Bit Rate: The bit rate for a single TWAI bus is configurable. However, all nodes on the same bus must operate at the same bit rate.

Transmitters and Receivers: At any point in time, a TWAI node can either be a transmitter or a receiver.

- A node generating a message is a transmitter. The node remains a transmitter until the bus is idle or until the node loses arbitration. Please note that nodes that have not lost arbitration can all be transmitters.
- All nodes that are not transmitters are receivers.

14.2.2 TWAI Messages

TWAI nodes use messages to transmit data, and signal errors to other nodes when detecting errors on the bus. Messages are split into various frame types, and some frame types will have different frame formats.

The TWAI protocol has of the following frame types:

- Data frame
- Remote frame
- Error frame
- Overload frame
- Interframe space

The TWAI protocol has the following frame formats:

- Standard Frame Format (SFF) that uses a 11-bit identifier
- Extended Frame Format (EFF) that uses a 29-bit identifier

14.2.2.1 Data Frames and Remote Frames

Data frames are used by nodes to send data to other nodes, and can have a payload of 0 to 8 data bytes. Remote frames are used for nodes to request a data frame with the same identifier from other nodes, and thus they do not contain any data bytes. However, data frames and remote frames share many fields. Figure 14-1 illustrates the fields and sub-fields of different frames and formats.

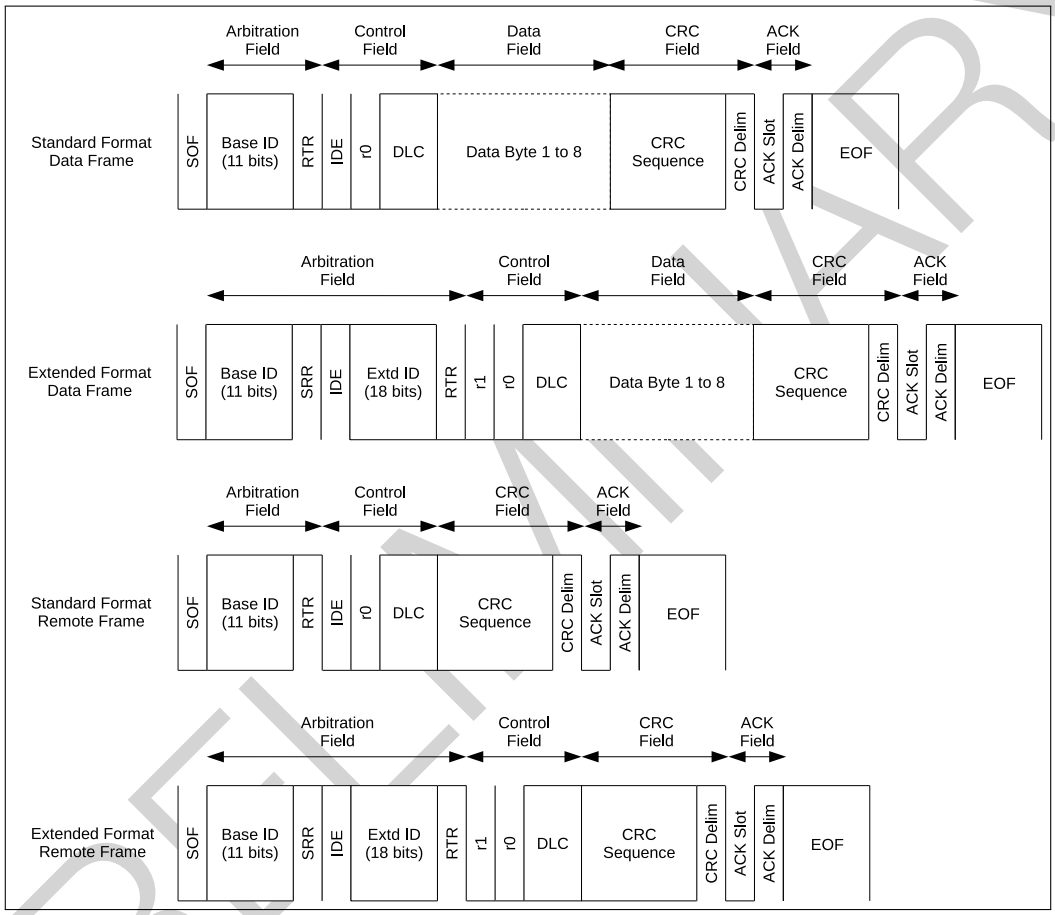


Figure 14-1. Bit Fields in Data Frames and Remote Frames

Arbitration Field

When two or more nodes transmits a data or remote frame simultaneously, the arbitration field is used to determine which node will win arbitration of the bus. In the arbitration field, if a node transmits a recessive bit while detects a dominant bit, this indicates that another node has overridden its recessive bit. Therefore, the node transmitting the recessive bit has lost arbitration of the bus and should immediately switch to be a receiver.

The arbitration field primarily consists of a frame identifier that is transmitted from the most significant bit first. Given that a dominant bit represents a logical 0, and a recessive bit represents a logical 1:

- A frame with the smallest ID value always wins arbitration.
- Given the same ID and format, data frames always prevail over remote frames due to their RTR bits being dominant.

- Given the same first 11 bits of ID, a Standard Format Data Frame always prevails over an Extended Format Data Frame due to its SRR bits being recessive.

Control Field

The control field primarily consists of the DLC (Data Length Code) which indicates the number of payload data bytes for a data frame, or the number of requested data bytes for a remote frame. The DLC is transmitted from the most significant bit first.

Data Field

The data field contains the actual payload data bytes of a data frame. Remote frames do not contain any data field.

CRC Field

The CRC field primarily consists of a CRC sequence. The CRC sequence is a 15-bit cyclic redundancy code calculated form the de-stuffed contents (everything from the SOF to the end of the data field) of a data or remote frame.

ACK Field

The ACK field primarily consists of an ACK Slot and an ACK Delim. The ACK field indicates that the receiver has received an effective message from the transmitter.

Table 14-1. Data Frames and Remote Frames in SFF and EFF

Data/Remote Frames	Description
SOF	The SOF (Start of Frame) is a single dominant bit used to synchronize nodes on the bus.
Base ID	The Base ID (ID.28 to ID.18) is the 11-bit identifier for SFF, or the first 11 bits of the 29-bit identifier for EFF.
RTR	The RTR (Remote Transmission Request) bit indicates whether the message is a data frame (dominant) or a remote frame (recessive). This means that a remote frame will always lose arbitration to a data frame if they have the same ID.
SRR	The SRR (Substitute Remote Request) bit is transmitted in EFF to substitute for the RTR bit at the same position in SFF.
IDE	The IDE (Identifier Extension) bit indicates whether the message is SFF (dominant) or EFF (recessive). This means that a SFF frame will always win arbitration over an EFF frame if they have the same Base ID.
Extd ID	The Extended ID (ID.17 to ID.0) is the remaining 18 bits of the 29-bit identifier for EFF.
r1	The r1 bit (reserved bit 1) is always dominant.
r0	The r0 bit (reserved bit 0) is always dominant.
DLC	The DLC (Data Length Code) is 4-bit long and should contain any value from 0 to 8. Data frames use the DLC to indicate the number of data bytes in the data frame. Remote frames used the DLC to indicate the number of data bytes to request from another node.
Data Bytes	The data payload of data frames. The number of bytes should match the value of DLC. Data byte 0 is transmitted first, and each data byte is transmitted from the most significant bit first.
CRC Sequence	The CRC sequence is a 15-bit cyclic redundancy code.

Data/Remote Frames	Description
CRC Delim	The CRC Delim (CRC Delimiter) is a single recessive bit that follows the CRC sequence.
ACK Slot	The ACK Slot (Acknowledgment Slot) is intended for receiver nodes to indicate that the data or remote frame was received without any issue. The transmitter node will send a recessive bit in the ACK Slot and receiver nodes should override the ACK Slot with a dominant bit if the frame was received without errors.
ACK Delim	The ACK Delim (Acknowledgment Delimiter) is a single recessive bit.
EOF	The EOF (End of Frame) marks the end of a data or remote frame, and consists of seven recessive bits.

14.2.2.2 Error and Overload Frames

Error Frames

Error frames are transmitted when a node detects a bus error. Error frames notably consist of an Error Flag which is made up of six consecutive bits of the same value, thus violating the bit-stuffing rule. Therefore, when a particular node detects a bus error and transmits an error frame, all other nodes will then detect a stuff error and transmit their own error frames in response. This has the effect of propagating the detection of a bus error across all nodes on the bus.

When a node detects a bus error, it will transmit an error frame starting from the next bit. However, if the type of bus error was a CRC error, then the error frame will start at the bit following the ACK Delim (see Section 14.2.3 for more details). The following Figure 14-2 shows different fields of an error frame:

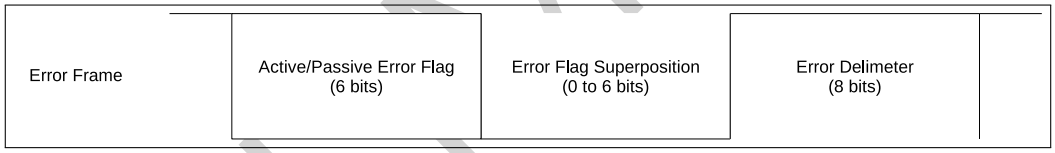


Figure 14-2. Fields of an Error Frame

Table 14-2. Error Frame

Error Frame	Description
Error Flag	The Error Flag has two forms, the Active Error Flag consisting of 6 dominant bits and the Passive Error Flag consisting of 6 recessive bits (unless overridden by dominant bits of other nodes). Active Error Flags are sent by error active nodes, whilst Passive Error Flags are sent by error passive nodes.
Error Flag Superposition	The Error Flag Superposition field meant to allow for other nodes on the bus to transmit their respective Active Error Flags. The superposition field can range from 0 to 6 bits, and ends when the first recessive bit is detected (i.e., the first it of the Delimiter).
Error Delimeter	The Delimiter field marks the end of the error/overload frame, and consists of 8 recessive bits.

Overload Frames

An overload frame has the same bit fields as an error frame containing an Active Error Flag. The key difference is in the cases that can trigger the transmission of an overload frame. Figure 14-3 below shows the bit fields of an overload frame.

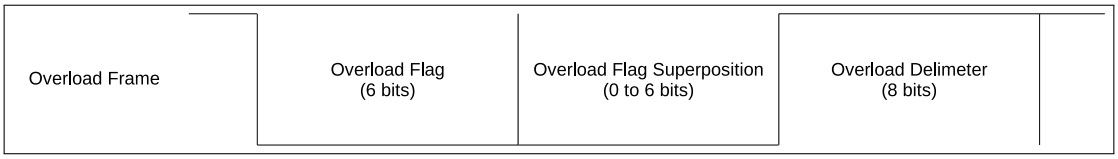


Figure 14-3. Fields of an Overload Frame

Table 14-3. Overload Frame

Overload Flag	Description
Overload Flag	Consists of 6 dominant bits. Same as an Active Error Flag.
Overload Flag Superposition	Allows for the superposition of Overload Flags from other nodes, similar to an Error Flag Superposition.
Overload Delimiter	Consists of 8 recessive bits. Same as an Error Delimiter.

Overload frames will be transmitted under the following cases:

1. A receiver requires a delay of the next data or remote frame.
2. A dominant bit is detected at the first and second bit of intermission.
3. A dominant bit is detected at the eighth (last) bit of an Error Delimiter. Note that in this case, TEC and REC will not be incremented (see Section 14.2.3 for more details).

Transmitting an overload frame due to one of the above cases must also satisfy the following rules:

- The start of an overload frame due to case 1 is only allowed to be started at the first bit time of an expected intermission.
- The start of an overload frame due to case 2 and 3 is only allowed to be started one bit after detecting the dominant bit.
- A maximum of two overload frames may be generated in order to delay the transmission of the next data or remote frame.

14.2.2.3 Interframe Space

The Interframe Space acts as a separator between frames. Data frames and remote frames must be separated from preceding frames by an Interframe Space, regardless of the preceding frame's type (data frame, remote frame, error frame, or overload frame). However, error frames and overload frames do not need to be separated from preceding frames.

Figure 14-4 shows the fields within an Interframe Space:

Table 14-4. Interframe Space

Interframe Space	Description
Intermission	The Intermission consists of 3 recessive bits.

Interframe Space	Description
Suspend Transmission	An Error Passive node that has just transmitted a message must include a Suspend Transmission field. This field consists of 8 recessive bits. Error Active nodes should not include this field.
Bus Idle	The Bus Idle field is of arbitrary length. Bus Idle ends when an SOF is transmitted. If a node has a pending transmission, the SOF should be transmitted at the first bit following Intermission.

14.2.3 TWAI Errors

14.2.3.1 Error Types

Bus Errors in TWAI are categorized into the following types:

Bit Error

A Bit Error occurs when a node transmits a bit value (i.e., dominant or recessive) but the opposite bit is detected (e.g., a dominant bit is transmitted but a recessive is detected). However, if the transmitted bit is recessive and is located in the Arbitration Field or ACK Slot or Passive Error Flag, then detecting a dominant bit will not be considered a Bit Error.

Stuff Error

A stuff error is detected when six consecutive bits of the same value are detected (which violats the bit-stuffing encoding rules).

CRC Error

A receiver of a data or remote frame will calculate CRC based on the bits it has received. A CRC error occurs when the CRC calculated by the receiver does not match the CRC sequence in the received data or remote Frame.

Format Error

A Format Error is detected when a format-fixed bit field of a message contains an illegal bit. For example, the r1 and r0 fields must be dominant.

ACK Error

An ACK Error occurs when a transmitter does not detect a dominant bit at the ACK Slot.

14.2.3.2 Error States

TWAI nodes implement fault confinement by each maintaining two error counters, where the counter values determine the error state. The two error counters are known as the Transmit Error Counter (TEC) and Receive Error Counter (REC). TWAI has the following error states.

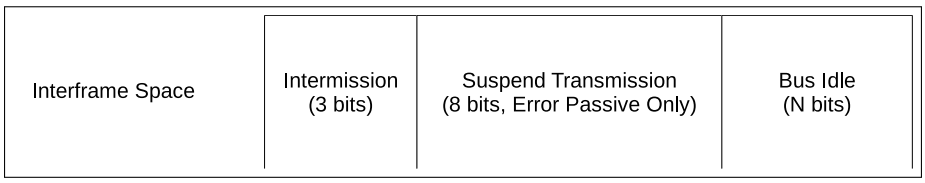


Figure 14-4. The Fields within an Interframe Space

Error Active

An Error Active node is able to participate in bus communication and transmit an Active Error Flag when it detects an error.

Error Passive

An Error Passive node is able to participate in bus communication, but can only transmit an Passive Error Flag when it detects an error. Error Passive nodes that have transmitted a data or remote frame must also include the Suspend Transmission field in the subsequent Interframe Space.

Bus Off

A Bus Off node is not permitted to influence the bus in any way (i.e., is not allowed to transmit data).

14.2.3.3 Error Counters

The TEC and REC are incremented/decremented according to the following rules. **Note that more than one rule can apply to a given message transfer.**

1. When a receiver detects an error, the REC is increased by 1, except when the detected error was a Bit Error during the transmission of an Active Error Flag or an Overload Flag.
2. When a receiver detects a dominant bit as the first bit after sending an Error Flag, the REC is increased by 8.
3. When a transmitter sends an Error Flag, the TEC is increased by 8. However, the following scenarios are exempt from this rule:
   - A transmitter is Error Passive since the transmitter generates an Acknowledgment Error because of not detecting a dominant bit in the ACK Slot, while detecting a dominant bit when sending a passive error flag. In this case, the TEC should not be increased.
   - A transmitter transmits an Error Flag due to a Stuff Error during Arbitration. If the stuffed bit should have been recessive but was monitored as dominant, then the TEC should not be increased.
4. If a transmitter detects a Bit Error whilst sending an Active Error Flag or Overload Flag, the TEC is increased by 8.
5. If a receiver detects a Bit Error while sending an Active Error Flag or Overload Flag, the REC is increased by 8.
6. A node can tolerate up to 7 consecutive dominant bits after sending an Active/Passive Error Flag, or Overload Flag. After detecting the 14th consecutive dominant bit (when sending an Active Error Flag or Overload Flag), or the 8th consecutive dominant bit following a Passive Error Flag, a transmitter will increase its TEC by 8 and a receiver will increase its REC by 8. Every additional 8 consecutive dominant bits will also increase the TEC (for transmitters) or REC (for receivers) by 8 as well.
7. When a transmitter has transmitted a message (getting ACK and no errors until the EOF is complete), the TEC is decremented by 1, unless the TEC is already at 0.

8. When a receiver successfully receives a message (no errors before ACK Slot, and successful sending of ACK), the REC is decremented.
    - If the REC is between 1 and 127, the REC will be decremented by 1.
    - If the REC is greater than 127, the REC will be set to 127.
    - If the REC is 0, the REC will remain 0.
9. A node becomes Error Passive when its TEC and/or REC is greater than or equal to 128. Though the node becomes Error Passive, it still sends an Active Error Flag. Note that once the REC has reached to 128, any further increases to its value are invalid until the REC returns to a value less than 128.
10. A node becomes Bus Off when its TEC is greater than or equal to 256.
11. An Error Passive node becomes Error Active when both the TEC and REC are less than or equal to 127.
12. A Bus Off node can become Error Active (with both its TEC and REC reset to 0) after it monitors 128 occurrences of 11 consecutive recessive bits on the bus.

14.2.4 TWAI Bit Timing

14.2.4.1 Nominal Bit

The TWAI protocol allows a TWAI bus to operate at a particular bit rate. However, all nodes within a TWAI bus must operate at the same bit rate.

- **The Nominal Bit Rate** is defined as the number of bits transmitted per second.
- **The Nominal Bit Time** is defined as **1/Nominal Bit Rate**.

A single Nominal Bit Time is divided into multiple segments, and each segment is made up of multiple Time Quanta. A **Time Quantum** is a minimum unit of time, and is implemented as some form of prescaled clock signal in each node. Figure 14-5 illustrates the segments within a single Nominal Bit Time.

TWAI controllers will operate in time steps of one Time Quanta where the state of the TWAI bus is analyzed. If the bus states in two consecutive Time Quantas are different (i.e., recessive to dominant or vice versa), it means an edge is generated. The intersection of PBS1 and PBS2 is considered the Sample Point and the sampled bus value is considered the value of that bit.

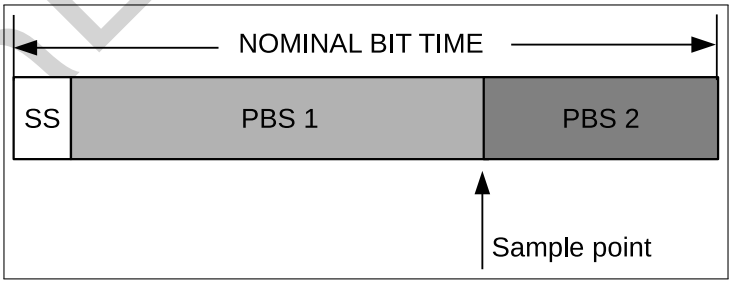


Figure 14-5. Layout of a Bit

Table 14-5. Segments of a Nominal Bit Time

Segment	Description
SS	The SS (Synchronization Segment) is 1 Time Quantum long. If all nodes are perfectly synchronized, the edge of a bit will lie in the SS.
PBS1	PBS1 (Phase Buffer Segment 1) can be 1 to 16 Time Quanta long. PBS1 is meant to compensate for the physical delay times within the network. PBS1 can also be lengthened for synchronization purposes.
PBS2	PBS2 (Phase Buffer Segment 2) can be 1 to 8 Time Quanta long. PBS2 is meant to compensate for the information processing time of nodes. PBS2 can also be shortened for synchronization purposes.

14.2.4.2 Hard Synchronization and Resynchronization

Due to clock skew and jitter, the bit timing of nodes on the same bus may become out of phase. Therefore, a bit edge may come before or after the SS. To ensure that the internal bit timing clocks of each node are kept in phase, TWAI has various methods of synchronization. The **Phase Error "e"** is measured in the number of Time Quanta and relative to the SS.

- A positive Phase Error (e > 0) is when the edge lies after the SS and before the Sample Point (i.e., the edge is late).
- A negative Phase Error (e < 0) is when the edge lies after the Sample Point of the previous bit and before SS (i.e., the edge is early).

To correct for Phase Errors, there are two forms of synchronization, known as **Hard Synchronization** and **Resynchronization**. **Hard Synchronization** and **Resynchronization** obey the following rules:

- Only one synchronization may occur in a single bit time.
- Synchronizations only occurs on recessive to dominant edges.

Hard Synchronization

Hard Synchronization occurs on the recessive to dominant (i.e., the first SOF bit after Bus Idle) edges when the bus is idle. All nodes will restart their internal bit timings so that the recessive to dominant edge lies within the SS of the restarted bit timing.

Resynchronization

Resynchronization occurs on recessive to dominant edges when the bus is not idel. If the edge has a positive Phase Error (e > 0), PBS1 is lengthened by a certain number of Time Quanta. If the edge has a negative Phase Error (e < 0), PBS2 will be shortened by a certain number of Time Quanta.

The number of Time Quanta to lengthen or shorten depends on the magnitude of the Phase Error, and is also limited by the Synchronization Jump Width (SJW) value which is programmable.

- When the magnitude of the Phase Error (**e**) is less than or equal to the SJW, PBS1/PBS2 are lengthened/shortened by the **e** number of Time Quanta. This has a same effect as Hard Synchronization.
- When the magnitude of the Phase Error is greater to the SJW, PBS1/PBS2 are lengthened/shortened by the SJW number of Time Quanta. This means it may take multiple bits of synchronization before the Phase Error is entirely corrected.

14.3 Architectural Overview

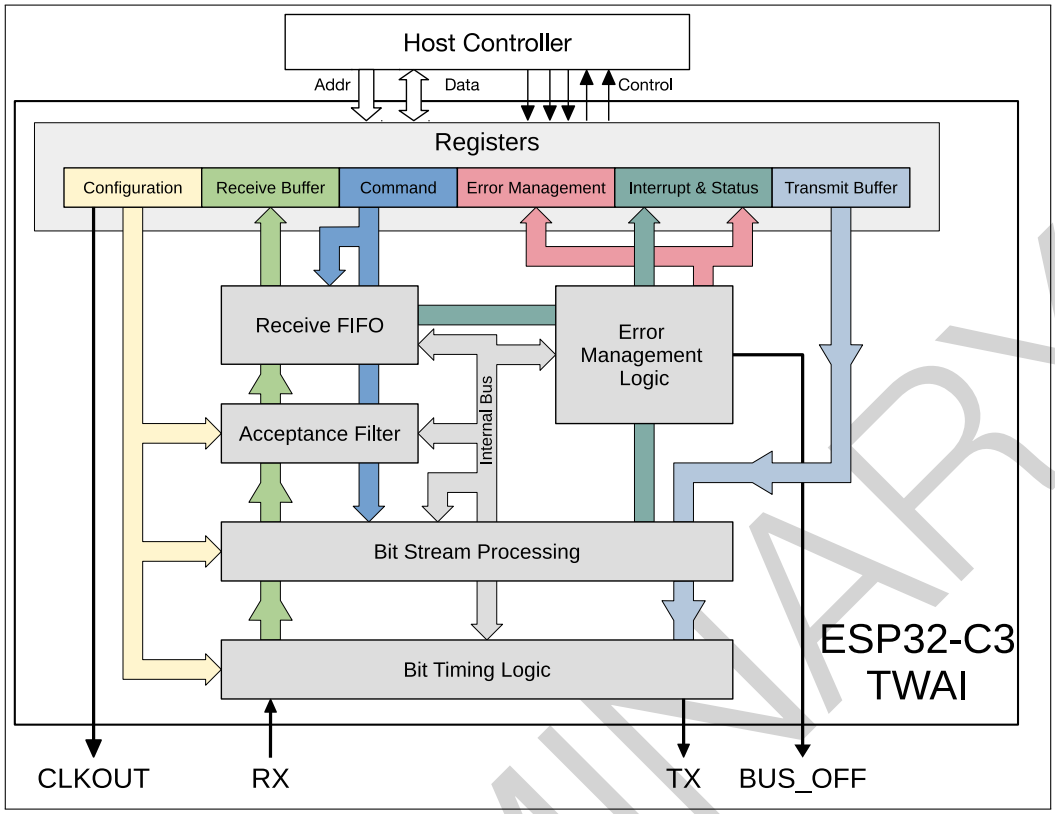


Figure 14-6. TWAI Overview Diagram

The major functional blocks of the TWAI controller are shown in Figure 14-6.

14.3.1 Registers Block

The ESP32-C3 CPU accesses peripherals using 32-bit aligned words. However, the majority of registers in the TWAI controller only contain useful data at the least significant byte (bits [7:0]). Therefore, in these registers, bits [31:8] are ignored on writes, and return 0 on reads.

Configuration Registers

The configuration registers store various configuration items for the TWAI controller such as bit rates, operation mode, Acceptance Filter, etc. Configuration registers can only be modified whilst the TWAI controller is in Reset Mode (See Section 14.4.1).

Command Registers

The command register is used by the CPU to drive the TWAI controller to initiate certain actions such as transmitting a message or clearing the Receive Buffer. The command register can only be modified when the TWAI controller is in Operation Mode (see section 14.4.1).

Interrupt & Status Registers

The interrupt register indicates what events have occurred in the TWAI controller (each event is represented by a separate bit). The status register indicates the current status of the TWAI controller.

Error Management Registers

The error management registers include error counters and capture registers. The error counter registers represent TEC and REC values. The capture registers will record information about instances where TWAI

controller detects a bus error, or when it loses arbitration.

Transmit Buffer Registers

The transmit buffer is a 13-byte buffer used to store a TWAI message to be transmitted.

Receive Buffer Registers

The Receive Buffer is a 13-byte buffer which stores a single message. The Receive Buffer acts as a window of Receive FIFO, whose first message will be mapped into the Receive Buffer.

Note that the Transmit Buffer registers, Receive Buffer registers, and the Acceptance Filter registers share the same address range (offset 0x0040 to 0x0070). Their access is governed by the following rules:

- When the TWAI controller is in Reset Mode, all reads and writes to the address range maps to the Acceptance Filter registers.
- When the TWAI controller is in Operation Mode:
  - All reads to the address range maps to the Receive Buffer registers.
  - All writes to the address range maps to the Transmit Buffer registers.

14.3.2 Bit Stream Processor

The Bit Stream Processing (BSP) module frames data from the Transmit Buffer (e.g. bit stuffing and additional CRC fields) and generates a bit stream for the Bit Timing Logic (BTL) module. At the same time, the BSP module is also responsible for processing the received bit stream (e.g., de-stuffing and verifying CRC) from the BTL module and placing the message into the Receive FIFO. The BSP will also detect errors on the TWAI bus and report them to the Error Management Logic (EML).

14.3.3 Error Management Logic

The Error Management Logic (EML) module updates the TEC and REC, records error information like error types and positions, and updates the error state of the TWAI controller such that the BSP module generates the correct Error Flags. Furthermore, this module also records the bit position when the TWAI controller loses arbitration.

14.3.4 Bit Timing Logic

The Bit Timing Logic (BTL) module transmits and receives messages at the configured bit rate. The BTL module also handles bit timing synchronization so that communication remains stable. A single bit time consists of multiple programmable segments that allows users to set the length of each segment to account for factors such as propagation delay and controller processing time, etc.

14.3.5 Acceptance Filter

The Acceptance Filter is a programmable message filtering unit that allows the TWAI controller to accept or reject a received message based on the message's ID field. Only accepted messages will be stored in the Receive FIFO. The Acceptance Filter's registers can be programmed to specify a single filter, or two separate filters (dual filter mode).

14.3.6 Receive FIFO

The Receive FIFO is a 64-byte buffer (inside the TWAI controller) that stores received messages accepted by the Acceptance Filter. Messages in the Receive FIFO can vary in size (between 3 to 13-bytes). When the Receive FIFO is full (or does not have enough space to store the next received message in its entirety), the Overrun Interrupt will be triggered, and any subsequent received messages will be lost until adequate space is cleared in the Receive FIFO. The first message in the Receive FIFO will be mapped to the 13-byte Receive Buffer until that message is cleared (using the Release Receive Buffer command bit). After being cleared, the Receive Buffer will map to the next message in the Receive FIFO, and the space occupied by the previous message in the Receive FIFO can be used to receive new messages.

14.4 Functional Description

14.4.1 Modes

The ESP32-C3 TWAI controller has two working modes: Reset Mode and Operation Mode. Reset Mode and Operation Mode are entered by setting or clearing the TWAI_RESET_MODE bit.

14.4.1.1 Reset Mode

Entering Reset Mode is required in order to modify the various configuration registers of the TWAI controller. When entering Reset Mode, the TWAI controller is essentially disconnected from the TWAI bus. When in Reset Mode, the TWAI controller will not be able to transmit any messages (including error signals). Any transmission in progress is immediately terminated. Likewise, the TWAI controller will not be able to receive any messages either.

14.4.1.2 Operation Mode

In operation mode, the TWAI controller connects to the bus and write-protect all configuration registers to ensure consistency during operation. When in Operation Mode, the TWAI controller can transmit and receive messages (including error signaling) depending on which operation sub-mode the TWAI controller was configured with. The TWAI controller supports the following operation sub-modes:

- **Normal Mode:** The TWAI controller can transmit and receive messages including error signals (such as error and overload Frames).
- **Self-test Mode:** Self-test mode is similar to normal Mode, but the TWAI controller will consider the transmission of a data or RTR frame successful and do not generate an ACK error even if it was not acknowledged. This is commonly used when the TWAI controller does self-test.
- **Listen-only Mode:** The TWAI controller will be able to receive messages, but will remain completely passive on the TWAI bus. Thus, the TWAI controller will not be able to transmit any messages, acknowledgments, or error signals. The error counters will remain frozen. This mode is useful for TWAI bus monitoring.

Note that when exiting Reset Mode (i.e., entering Operation Mode), the TWAI controller must wait for 11 consecutive recessive bits to occur before being able to fully connect the TWAI bus (i.e., be able to transmit or receive).

14.4.2 Bit Timing

The operating bit rate of the TWAI controller must be configured whilst the TWAI controller is in Reset Mode. The bit rate is configured using TWAI_BUS_TIMING_0_REG and TWAI_BUS_TIMING_1_REG, and the two registers contain the following fields:

The following Table 14-6 illustrates the bit fields of TWAI_BUS_TIMING_0_REG.

Table 14-6. Bit Information of TWAI_BUS_TIMING_0_REG (0x18)

Bit 31-16	Bit 15	Bit 14	Bit 13	Bit 12	.....	Bit 1	Bit 0
Reserved	SJW.1	SJW.0	Reserved	BRP.12	.....	BRP.1	BRP.0

Notes:

- BRP: The TWAI Time Quanta clock is derived from the APB clock that is usually 80 MHz. The Baud Rate Prescaler (BRP) field is used to define the prescaler according to the equation below, where t_{Tq} is the Time Quanta clock cycle and t_{CLK} is APB clock cycle:
  $t_{Tq} = 2 \times t_{CLK} \times (2^{12} \times BRP.12 + 2^{11} \times BRP.11 + ... + 2^{1} \times BRP.1 + 2^{0} \times BRP.0 + 1)$
- SJW: Synchronization Jump Width (SJW) is configured in SJW.0 and SJW.1 where SJW = (2 x SJW.1 + SJW.0 + 1)

The following Table 14-7 illustrates the bit fields of TWAI_BUS_TIMING_1_REG.

Table 14-7. Bit Information of TWAI_BUS_TIMING_1_REG (0x1c)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	SAM	PBS2.2	PBS2.1	PBS2.0	PBS1.3	PBS1.2	PBS1.1	PBS1.0

Notes:

- PBS1: The number of Time Quanta in Phase Buffer Segment 1 is defined according to the following equation: (8 x PBS1.3 + 4 x PBS1.2 + 2 x PBS1.1 + PBS1.0 + 1)
- PBS2: The number of Time Quanta in Phase Buffer Segment 2 is defined according to the following equation: (4 x PBS2.2 + 2 x PBS2.1 + PBS2.0 + 1)
- SAM: Enables triple sampling if set to 1. This is useful for low/medium speed buses to filter spikes on the bus line.

14.4.3 Interrupt Management

The ESP32-C3 TWAI controller provides eight interrupts, each represented by a single bit in the TWAI_INT_RAW_REG. For a particular interrupt to be triggered, the corresponding enable bit in TWAI_INT ENA_REG must be set.

The TWAI controller provides the following interrupts:

- Receive Interrupt
- Transmit Interrupt
- Error Warning Interrupt
- Data Overrun Interrupt
- Error Passive Interrupt

- Arbitration Lost Interrupt
- Bus Error Interrupt
- Bus Status Interrupt

The TWAI controller's interrupt signal to the interrupt matrix will be asserted whenever one or more interrupt bits are set in the TWAI_INT_RAW_REG, and deasserted when all bits in TWAI_INT_RAW_REG are cleared. The majority of interrupt bits in TWAI_INT_RAW_REG are automatically cleared when the register is read, except for the Receive Interrupt which can only be cleared when all the messages are released by setting the TWAI_RELEASE_BUF bit.

14.4.3.1 Receive Interrupt (RXI)

The Receive Interrupt (RXI) is asserted whenever the TWAI controller has received messages that are pending to be read from the Receive Buffer (i.e., when TWAI_RX_MESSAGE_CNT_REG > 0). Pending received messages includes valid messages in the Receive FIFO and also overrun messages. The RXI will not be deasserted until all pending received messages are cleared using the TWAI_RELEASE_BUF command bit.

14.4.3.2 Transmit Interrupt (TXI)

The Transmit Interrupt (TXI) is triggered whenever Transmit Buffer becomes free, indicating another message can be loaded into the Transmit Buffer to be transmitted. The Transmit Buffer becomes free under the following scenarios:

- A message transmission has completed successfully, i.e., acknowledged without any errors. (Any failed messages will automatically be resent.)
- A single shot transmission has completed (successfully or unsuccessfully, indicated by the TWAI_TX_COMPLETE bit).
- A message transmission was aborted using the TWAI_ABORT_TX command bit.

14.4.3.3 Error Warning Interrupt (EWI)

The Error Warning Interrupt (EWI) is triggered whenever there is a change to the TWAI_ERR_ST and TWAI_BUS_OFF_ST bits of the TWAI_STATUS_REG (i.e., transition from 0 to 1 or vice versa). Thus, an EWI could indicate one of the following events, depending on the values TWAI_ERR_ST and TWAI_BUS_OFF_ST at the moment when the EWI is triggered.

- If TWAI_ERR_ST = 0 and TWAI_BUS_OFF_ST = 0:
  - If the TWAI controller was in the Error Active state, it indicates both the TEC and REC have returned below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
    - If the TWAI controller was previously in the Bus Off Recovery state, it indicates that Bus Recovery has completed successfully.
- If TWAI_ERR_ST = 1 and TWAI_BUS_OFF_ST = 0: The TEC or REC error counters have exceeded the threshold value set by TWAI_ERR_WARNING_LIMIT_REG.
- If TWAI_ERR_ST = 1 and TWAI_BUS_OFF_ST = 1: The TWAI controller has entered the BUS_OFF state (due to the TEC >= 256).

- If TWAI_ERR_ST = 0 and TWAI_BUS_OFF_ST = 1: The TWAI controller's TEC has dropped below the threshold value set by TWAI_ERR_WARNING_LIMIT_REG during BUS_OFF recovery.

14.4.3.4 Data Overrun Interrupt (DOI)

The Data Overrun Interrupt (DOI) is triggered whenever the Receive FIFO has overrun. The DOI indicates that the Receive FIFO is full and should be cleared immediately to prevent any further overrun messages.

The DOI is only triggered by the first message that causes the Receive FIFO to overrun (i.e., the transition from the Receive FIFO not being full to the Receive FIFO overrunning). Any subsequent overrun messages will not trigger the DOI again. The DOI could be triggered again when all received messages (valid or overrun) have been cleared.

14.4.3.5 Error Passive Interrupt (TXI)

The Error Passive Interrupt (EPI) is triggered whenever the TWAI controller switches from Error Active to Error Passive, or vice versa.

14.4.3.6 Arbitration Lost Interrupt (ALI)

The Arbitration Lost Interrupt (ALI) is triggered whenever the TWAI controller is attempting to transmit a message and loses arbitration. The bit position where the TWAI controller lost arbitration is automatically recorded in Arbitration Lost Capture register (TWAI_ARB LOST CAP_REG). When the ALI occurs again, the Arbitration Lost Capture register will no longer record new bit location until it is cleared (via CPU reading this register).

14.4.3.7 Bus Error Interrupt (BEI)

The Bus Error Interrupt (BEI) is triggered whenever TWAI controller detects an error on the TWAI bus. When a bus error occurs, the Bus Error type and its bit position are automatically recorded in the Error Code Capture register (TWAI_ERR_CODE_CAP_REG). When the BEI occurs again, the Error Code Capture register will no longer record new error information until it is cleared (via a read from the CPU).

14.4.3.8 Bus Status Interrupt (BSI)

The Bus Status Interrupt (BSI) is triggered whenever TWAI controller is switching between receive/transmit status and idle status. When a BSI occurs, the current status of TWAI controller can be measured by reading TWAI_RX_ST and TWAI_TX_ST in TWAI_STATUS_REG register.

14.4.4 Transmit and Receive Buffers

14.4.4.1 Overview of Buffers

Table 14-8. Buffer Layout for Standard Frame Format and Extended Frame Format

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
TWAI Address	Content	TWAI Address	Content
0x40	TX/RX frame information	0x40	TX/RX frame information
0x44	TX/RX identifier 1	0x44	TX/RX identifier 1

Standard Frame Format (SFF)		Extended Frame Format (EFF)	
TWAI Address	Content	TWAI Address	Content
0x48	TX/RX identifier 2	0x48	TX/RX identifier 2
0x4c	TX/RX data byte 1	0x4c	TX/RX identifier 3
0x50	TX/RX data byte 2	0x50	TX/RX identifier 4
0x54	TX/RX data byte 3	0x54	TX/RX data byte 1
0x58	TX/RX data byte 4	0x58	TX/RX data byte 2
0x5c	TX/RX data byte 5	0x5c	TX/RX data byte 3
0x60	TX/RX data byte 6	0x60	TX/RX data byte 4
0x64	TX/RX data byte 7	0x64	TX/RX data byte 5
0x68	TX/RX data byte 8	0x68	TX/RX data byte 6
0x6c	reserved	0x6c	TX/RX data byte 7
0x70	reserved	0x70	TX/RX data byte 8

Table 14-8 illustrates the layout of the Transmit Buffer and Receive Buffer registers. Both the Transmit and Receive Buffer registers share the same address space and are only accessible when the TWAI controller is in Operation Mode. The CPU accesses Transmit Buffer registers for write operations, and Receive Buffer registers for read operations . Both buffers share the exact same register layout and fields to represent a message (received or to be transmitted). The Transmit Buffer registers are used to configure a TWAI message to be transmitted. The CPU would write to the Transmit Buffer registers specifying the message's frame type, frame format, frame ID, and frame data (payload). Once the Transmit Buffer is configured, the CPU would then initiate the transmission by setting the TWAI_TX_REQ bit in TWAI_CMD_REG.

- For a self-reception request, set the TWAI_SELF_RX_REQ bit instead.
- For a single-shot transmission, set both the TWAI_TX_REQ and the TWAI_ABORT_TX simultaneously.

The Receive Buffer registers map the first message in the Receive FIFO. The CPU would read the Receive Buffer registers to obtain the first message's frame type, frame format, frame ID, and frame data (payload). Once the message has been read from the Receive Buffer registers, the CPU can set the TWAI_RELEASE_BUF bit in TWAI_CMD_REG to clear the Receive Buffer registers. If there are still messages in the Receive FIFO, the Receive Buffer registers will map the first message again.

14.4.4.2 Frame Information

The frame information is one byte long and specifies a message's frame type, frame format, and length of data. The frame information fields are shown in Table 14-9.

Table 14-9. TX/RX Frame Information (SFF/EFF) TWAI Address 0x40

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	FF[1]	RTR[2]	X[3]	X[3]	DLC.3[4]	DLC.2[4]	DLC.1[4]	DLC.0[4]

Notes:

1. FF: The Frame Format (FF) bit specifies whether the message is Extended Frame Format (EFF) or Standard Frame Format (SFF). The message is EFF when FF bit is 1, and SFF when FF bit is 0.
2. RTR: The Remote Transmission Request (RTR) bit specifies whether the message is a data frame or a remote frame. The message is a remote frame when the RTR bit is 1, and a data frame when the RTR bit is

0.

3. X: Don't care, can be any value.
4. DLC: The Data Length Code (DLC) field specifies the number of data bytes for a data frame, or the number of data bytes to request in a remote frame. TWAI data frames are limited to a maximum payload of 8 data bytes, and thus the DLC should range anywhere from 0 to 8.

14.4.4.3 Frame Identifier

The Frame Identifier fields is two-byte (11-bit) long if the message is SFF, and four-byte (29-bit) long if the message is EFF.

The Frame Identifier fields for an SFF (11-bit) message is shown in Table 14-10 ~ 14-11.

Table 14-10. TX/RX Identifier 1 (SFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3

Table 14-11. TX/RX Identifier 2 (SFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]	X[2]	X[2]

Notes:

1. Don't care. Recommended to be compatible with receive buffer (i.e., set to RTR) in case of using the self reception functionality (or together with self-test functionality).
2. Don't care. Recommended to be compatible with receive buffer (i.e., set to 0) in case of using the self reception functionality (or together with self-test functionality).

The Frame Identifier fields for an EFF (29-bits) message is shown in Table 14-12 ~ 14-15.

Table 14-12. TX/RX Identifier 1 (EFF); TWAI Address 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Table 14-13. TX/RX Identifier 2 (EFF); TWAI Address 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Table 14-14. TX/RX Identifier 3 (EFF); TWAI Address 0x4c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Table 14-15. TX/RX Identifier 4 (EFF); TWAI Address 0x50

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ID.4	ID.3	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]

Notes:

1. Don't care. Recommended to be compatible with receive buffer (i.e., set to RTR) in case of using the self reception functionality (or together with self-test functionality).
2. Don't care. Recommended to be compatible with receive buffer (i.e., set to 0) in case of using the self reception functionality (or together with self-test functionality).

14.4.4.4 Frame Data

The Frame Data field contains the payloads of transmitted or received data frame, and can range from 0 to eight bytes. The number of valid bytes should be equal to the DLC. However, if the DLC is larger than eight bytes, the number of valid bytes would still be limited to eight. Remote frames do not have data payloads, so their Frame Data fields will be unused.

For example, when transmitting a data frame with five bytes, the CPU should write five to the DLC field, and then write data to the corresponding register of the first to the fifth data field. Likewise, when the CPU receives a data frame with a DLC of five data bytes, only the first to the fifth data byte will contain valid payload data for the CPU to read.

14.4.5 Receive FIFO and Data Overruns

The Receive FIFO is a 64-byte internal buffer used to store received messages in First In First Out order. A single received message can occupy between 3 to 13 bytes of space in the Receive FIFO, and their endianness is identical to the register layout of the Receive Buffer registers. The Receive Buffer registers are mapped to the bytes of the first message in the Receive FIFO.

When the TWAI controller receives a message, it will increment the value of TWAI_RX_MESSAGE_COUNTER up to a maximum of 64. If there is adequate space in the Receive FIFO, the message contents will be written into the Receive FIFO. Once a message has been read from the Receive Buffer, the TWAI_RELEASE_BUF bit should be set. This will decrement TWAI_RX_MESSAGE_COUNTER and free the space occupied by the first message in the Receive FIFO. The Receive Buffer will then map to the next message in the Receive FIFO.

A data overrun occurs when the TWAI controller receives a message, but the Receive FIFO lacks the adequate free space to store the received message in its entirety (either due to the message contents being larger than the free space in the Receive FIFO, or the Receive FIFO being completely full).

When a data overrun occurs:

- The free space left in the Receive FIFO is filled with the partial contents of the overrun message. If the Receive FIFO is already full, then none of the overrun message's contents will be stored.
- When data in the Receive FIFO overruns for the first time, a Data Overrun Interrupt will be triggered.
- Each overrun message will still increment the TWAI_RX_MESSAGE_COUNTER up to a maximum of 64.
- The Receive FIFO will internally mark overrun messages as invalid. The TWAI_MISS_ST bit can be used to determine whether the message currently mapped to by the Receive Buffer is valid or overrun.

To clear an overrun Receive FIFO, the TWAI_RELEASE_BUF must be called repeatedly until TWAI_RX_MESSAGE_COUNTER is 0. This has the effect of reading all valid messages in the Receive FIFO and clearing all overrun messages.

14.4.6 Acceptance Filter

The Acceptance Filter allows the TWAI controller to filter out received messages based on their ID (and optionally their first data byte and frame type). Only accepted messages are passed on to the Receive FIFO. The use of Acceptance Filters allows a more lightweight operation of the TWAI controller (e.g., less use of Receive FIFO, fewer Receive Interrupts) since the TWAI Controller only need to handle a subset of messages.

The Acceptance Filter configuration registers can only be accessed whilst the TWAI controller is in Reset Mode, since they share the same address spaces with the Transmit Buffer and Receive Buffer registers.

The configuration registers consist of a 32-bit Acceptance Code Value and a 32-bit Acceptance Mask Value. The Acceptance Code value specifies a bit pattern which each filtered bit of the message must match in order for the message to be accepted. The Acceptance Mask Value is able to mask out certain bits of the Code value (i.e., set as “Don’t Care” bits). Each filtered bit of the message must either match the acceptance code or be masked in order for the message to be accepted, as demonstrated in Figure 14-7.

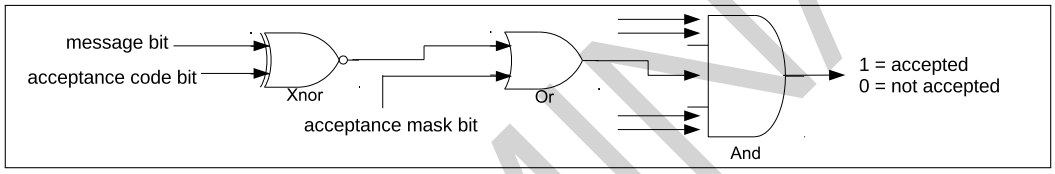


Figure 14-7. Acceptance Filter

The TWAI controller Acceptance Filter allows the 32-bit Acceptance Code and Mask Values to either define a single filter (i.e., Single Filter Mode), or two filters (i.e., Dual Filter Mode). How the Acceptance Filter interprets the 32-bit code and mask values is dependent on filter mode and the format of received messages (i.e., SFF or EFF).

14.4.6.1 Single Filter Mode

Single Filter Mode is enabled by setting the TWAI_RX_FILTER_MODE bit to 1. This will cause the 32-bit code and mask values to define a single filter. The single filter can filter the following bits of a data or remote frame:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 and Data byte 2
- EFF
  - The entire 29-bit ID
  - RTR bit

The following Figure 14-8 illustrates how the 32-bit code and mask values will be interpreted under Single Filter Mode.

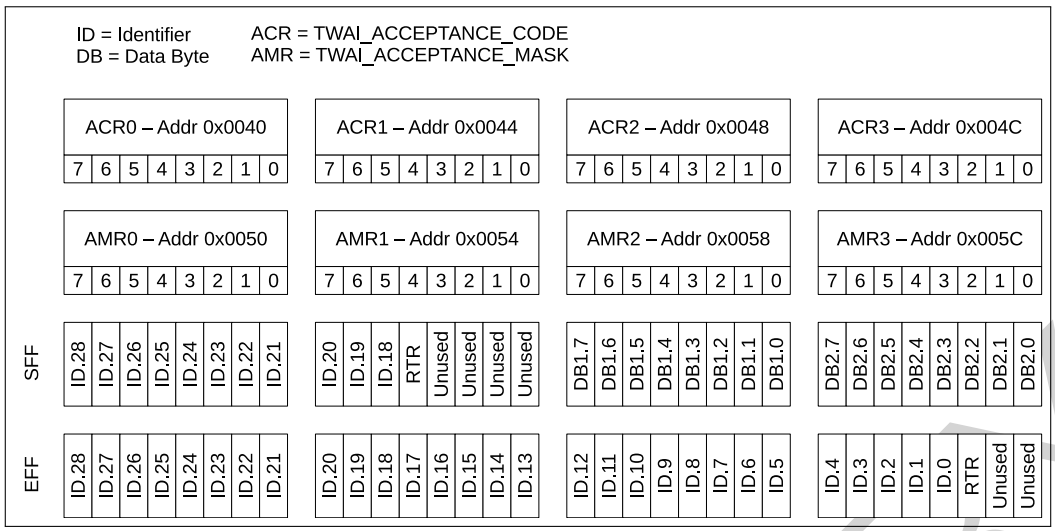


Figure 14-8. Single Filter Mode

14.4.6.2 Dual Filter Mode

Dual Filter Mode is enabled by clearing the TWAI_RX_FILTER_MODE bit to 0. This will cause the 32-bit code and mask values to define a two separate filters referred to as filter 1 or filter 2. Under Dual Filter Mode, a message will be accepted if it is accepted by one of the two filters.

The two filters can filter the following bits of a data or remote frame:

- SFF
  - The entire 11-bit ID
  - RTR bit
  - Data byte 1 (for filter 1 only)
- EFF
  - The first 16 bits of the 29-bit ID

The following Figure 14-9 illustrates how the 32-bit code and mask values will be interpreted in Dual Filter Mode.

14.4.7 Error Management

The TWAI protocol requires that each TWAI node maintains the Transmit Error Counter (TEC) and Receive Error Counter (REC). The value of both error counters determines the current error state of the TWAI controller (i.e., Error Active, Error Passive, Bus-Off). The TWAI controller stores the TEC and REC values in TWAI_TX_ERR_CNT_REG and TWAI_RX_ERR_CNT_REG respectively, and they can be read by the CPU anytime. In addition to the error states, the TWAI controller also offers an Error Warning Limit (EWL) feature that can warn users of the occurrence of severe bus errors before the TWAI controller enters the Error Passive state.

The current error state of the TWAI controller is indicated via a combination of the following values and status bits: TEC, REC, TWAI_ERR_ST, and TWAI_BUS_OFF_ST. Certain changes to these values and bits will also trigger interrupts, thus allowing the users to be notified of error state transitions (see section 14.4.3). The following figure 14-10 shows the relation between the error states, values and bits, and error state related interrupts.

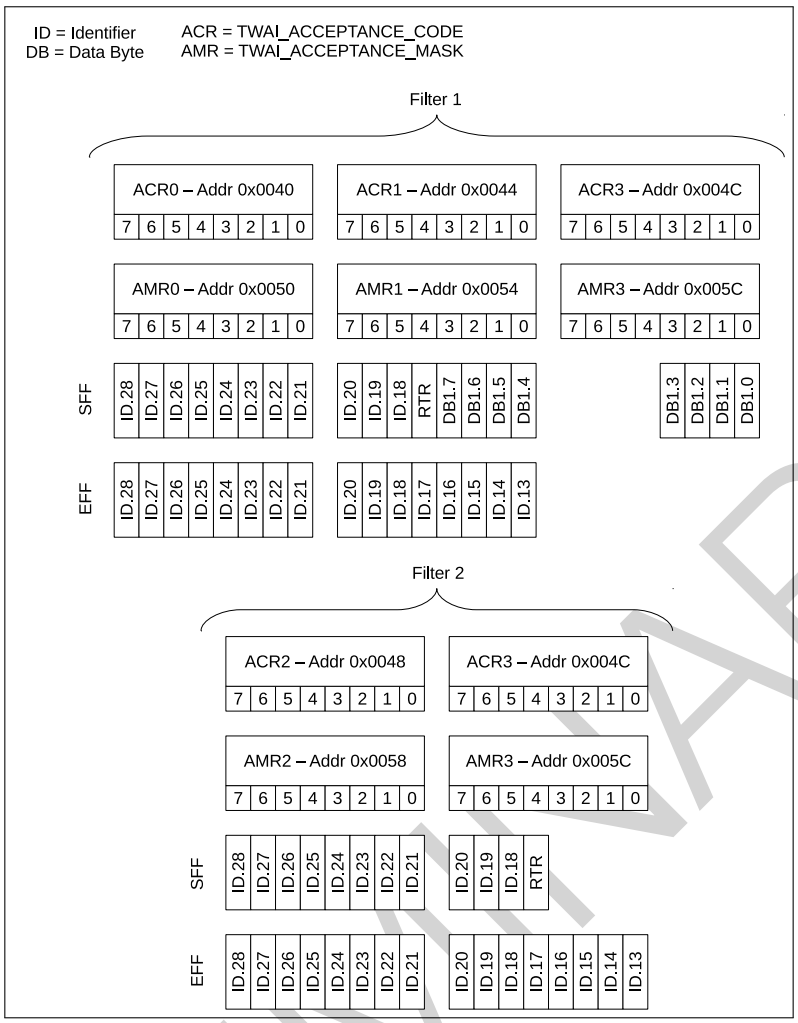


Figure 14-9. Dual Filter Mode

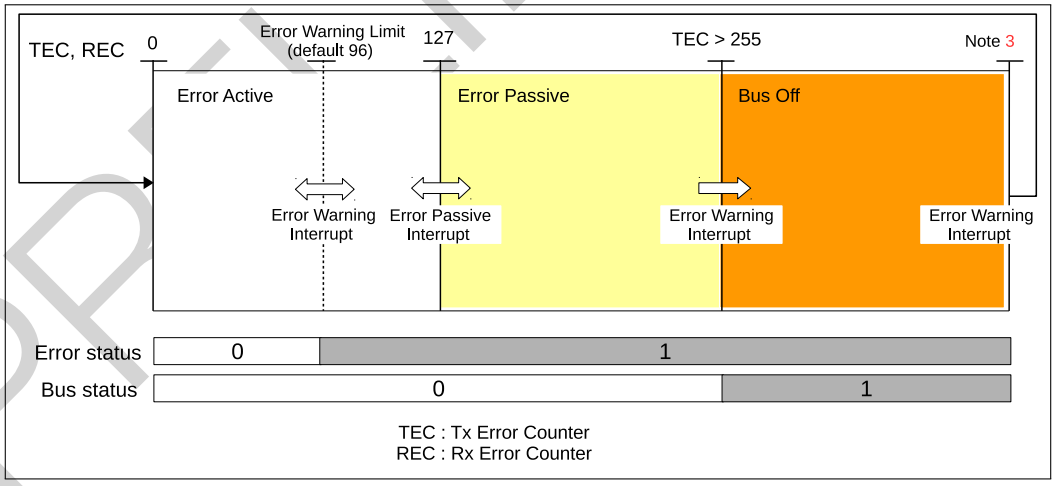


Figure 14-10. Error State Transition

14.4.7.1 Error Warning Limit

The Error Warning Limit (EWL) is a configurable threshold value for the TEC and REC, which will trigger an interrupt when exceeded. The EWL is intended to serve as a warning about severe TWAI bus errors, and is triggered before the TWAI controller enters the Error Passive state. The EWL is configured in

TWAI_ERR_WARNING_LIMIT_REG and can only be configured whilst the TWAI controller is in Reset Mode. The TWAI_ERR_WARNING_LIMIT_REG has a default value of 96. When the values of TEC and/or REC are larger than or equal to the EWL value, the TWAI_ERR_ST bit is immediately set to 1. Likewise, when the values of both the TEC and REC are smaller than the EWL value, the TWAI_ERR_ST bit is immediately reset to 0. The Error Warning Interrupt is triggered whenever the value of the TWAI_ERR_ST bit (or the TWAI_BUS_OFF_ST) changes.

14.4.7.2 Error Passive

The TWAI controller is in the Error Passive state when the TEC or REC value exceeds 127. Likewise, when both the TEC and REC are less than or equal to 127, the TWAI controller enters the Error Active state. The Error Passive Interrupt is triggered whenever the TWAI controller transitions from the Error Active state to the Error Passive state or vice versa.

14.4.7.3 Bus-Off and Bus-Off Recovery

The TWAI controller enters the Bus-Off state when the TEC value exceeds 255. On entering the Bus-Off state, the TWAI controller will automatically do the following:

- Set REC to 0
- Set TEC to 127
- Set the TWAI_BUS_OFF_ST bit to 1
- Enter Reset Mode

The Error Warning Interrupt is triggered whenever the value of the TWAI_BUS_OFF_ST bit (or the TWAI_ERR_ST bit) changes.

To return to the Error Active state, the TWAI controller must undergo Bus-Off Recovery. Bus-Off Recovery requires the TWAI controller to observe 128 occurrences of 11 consecutive recessive bits on the bus. To initiate Bus-Off Recovery (after entering the Bus-Off state), the TWAI controller should enter Operation Mode by setting the TWAI_RESET_MODE bit to 0. The TEC tracks the progress of Bus-Off Recovery by decrementing the TEC each time when the TWAI controller observes 11 consecutive recessive bits. When Bus-Off Recovery has completed (i.e., TEC has decremented from 127 to 0), the TWAI_BUS_OFF_ST bit will automatically be reset to 0, thus triggering the Error Warning Interrupt.

14.4.8 Error Code Capture

The Error Code Capture (ECC) feature allows the TWAI controller to record the error type and bit position of a TWAI bus error in the form of an error code. Upon detecting a TWAI bus error, the Bus Error Interrupt is triggered and the error code is recorded in TWAI_ERR_CODE_CAP_REG. Subsequent bus errors will trigger the Bus Error Interrupt, but their error codes will not be recorded until the current error code is read from the TWAI_ERR_CODE_CAP_REG.

The following Table 14-16 shows the fields of the TWAI_ERR_CODE_CAP_REG:

Table 14-16. Bit Information of TWAI_ERR_CODE_CAP_REG (0x30)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	ERRC.1[1]	ERRC.0[1]	DIR[2]	SEG.4[3]	SEG.3[3]	SEG.2[3]	SEG.1[3]	SEG.0[3]

Notes:

- ERRC: The Error Code (ERRC) indicates the type of bus error: 00 for bit error, 01 for format error, 10 for stuff error, and 11 for other types of error.
- DIR: The Direction (DIR) indicates whether the TWAI controller was transmitting or receiving when the bus error occurred: 0 for transmitter, 1 for receiver.
- SEG: The Error Segment (SEG) indicates which segment of the TWAI message (i.e., bit position) the bus error occurred at.

The following Table 14-17 shows how to interpret the SEG.0 to SEG.4 bits.

Table 14-17. Bit Information of Bits SEG.4 - SEG.0

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	Description
0	0	0	1	1	start of frame
0	0	0	1	0	ID.28 ~ ID.21
0	0	1	1	0	ID.20 ~ ID.18
0	0	1	0	0	bit SRTR
0	0	1	0	1	bit IDE
0	0	1	1	1	ID.17 ~ ID.13
0	1	1	1	1	ID.12 ~ ID.5
0	1	1	1	0	ID.4 ~ ID.0
0	1	1	0	0	bit RTR
0	1	1	0	1	reserved bit 1
0	1	0	0	1	reserved bit 0
0	1	0	1	1	data length code
0	1	0	1	0	data field
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimiter
1	1	0	0	1	ACK slot
1	1	0	1	1	ACK delimiter
1	1	0	1	0	end of frame
1	0	0	1	0	intermission
1	0	0	0	1	active error flag
1	0	1	1	0	passive error flag
1	0	0	1	1	tolerate dominant bits
1	0	1	1	1	error delimiter
1	1	1	0	0	overload flag

Notes:

- Bit SRTR: under Standard Frame Format.
- Bit IDE: Identifier Extension Bit, 0 for Standard Frame Format.

14.4.9 Arbitration Lost Capture

The Arbitration Lost Capture (ALC) feature allows the TWAI controller to record the bit position where it loses arbitration. When the TWAI controller loses arbitration, the bit position is recorded in TWAI_ARB LOST CAP_REG and the Arbitration Lost Interrupt is triggered.

Subsequent losses in arbitration will trigger the Arbitration Lost Interrupt, but will not be recorded in TWAI_ARB_LOST_CAP_REG until the current Arbitration Lost Capture is read from the TWAI_ERR_CODE_CAP_REG.

Table 14-18 illustrates bits and fields of TWAI_ERR_CODE_CAP_REG whilst Figure 14-11 illustrates the bit positions of a TWAI message.

Table 14-18. Bit Information of TWAI_ARB LOST CAP_REG (0x2c)

Bit 31-5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	BITNO.4[1]	BITNO.3[1]	BITNO.2[1]	BITNO.1[1]	BITNO.0[1]

Notes:

- BITNO: Bit Number (BITNO) indicates the nth bit of a TWAI message where arbitration was lost.

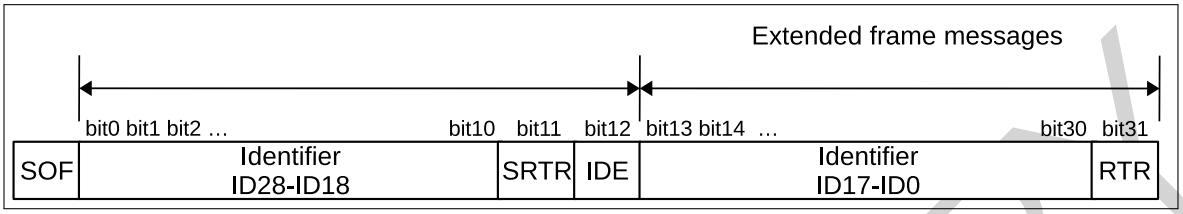


Figure 14-11. Positions of Arbitration Lost Bits

14.5 Register Summary

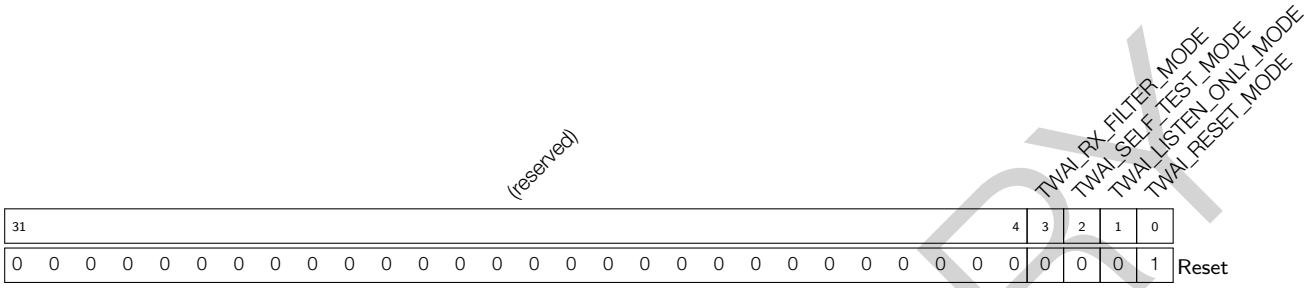
'|' here means separate line to distinguish between TWAI working modes discussed in Section 14.4.1 *Modes*. The left describes the access in Operation Mode. The right belongs to Reset Mode and is marked in red. The addresses in this section are relative to Two-wire Automotive Interface base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Name	Description	Address	Access
Configuration Registers			
TWAI_MODE_REG	Mode Register	0x0000	R/W
TWAI_BUS_TIMING_0_REG	Bus Timing Register 0	0x0018	RO | R/W
TWAI_BUS_TIMING_1_REG	Bus Timing Register 1	0x001C	RO | R/W
TWAI_ERR_WARNING_LIMIT_REG	Error Warning Limit Register	0x0034	RO | R/W
TWAI_DATA_0_REG	Data Register 0	0x0040	WO | R/W
TWAI_DATA_1_REG	Data Register 1	0x0044	WO | R/W
TWAI_DATA_2_REG	Data Register 2	0x0048	WO | R/W
TWAI_DATA_3_REG	Data Register 3	0x004C	WO | R/W
TWAI_DATA_4_REG	Data Register 4	0x0050	WO | R/W
TWAI_DATA_5_REG	Data Register 5	0x0054	WO | R/W
TWAI_DATA_6_REG	Data Register 6	0x0058	WO | R/W
TWAI_DATA_7_REG	Data Register 7	0x005C	WO | R/W
TWAI_DATA_8_REG	Data Register 8	0x0060	WO | RO
TWAI_DATA_9_REG	Data Register 9	0x0064	WO | RO
TWAI_DATA_10_REG	Data Register 10	0x0068	WO | RO
TWAI_DATA_11_REG	Data Register 11	0x006C	WO | RO
TWAI_DATA_12_REG	Data Register 12	0x0070	WO | RO
TWAI_CLOCK_DIVIDER_REG	Clock Divider Register	0x007C	varies
Contro Registers			
TWAI_CMD_REG	Command Register	0x0004	WO
Status Register			
TWAI_STATUS_REG	Status Register	0x0008	RO
TWAI_ARB LOST CAP_REG	Arbitration Lost Capture Register	0x002C	RO
TWAI_ERR_CODE_CAP_REG	Error Code Capture Register	0x0030	RO
TWAI_RX_ERR_CNT_REG	Receive Error Counter Register	0x0038	RO | R/W
TWAI_TX_ERR_CNT_REG	Transmit Error Counter Register	0x003C	RO | R/W
TWAI_RX_MESSAGE_CNT_REG	Receive Message Counter Register	0x0074	RO
Interrupt Registers			
TWAI_INT_RAW_REG	Interrupt Register	0x000C	RO
TWAI_INT ENA_REG	Interrupt Enable Register	0x0010	R/W

14.6 Registers

'I' here means separate line. The left describes the access in Operation Mode. The right belongs to Reset Mode with red color. The addresses in this section are relative to Two-wire Automotive Interface base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Register 14.1. TWAI_MODE_REG (0x0000)



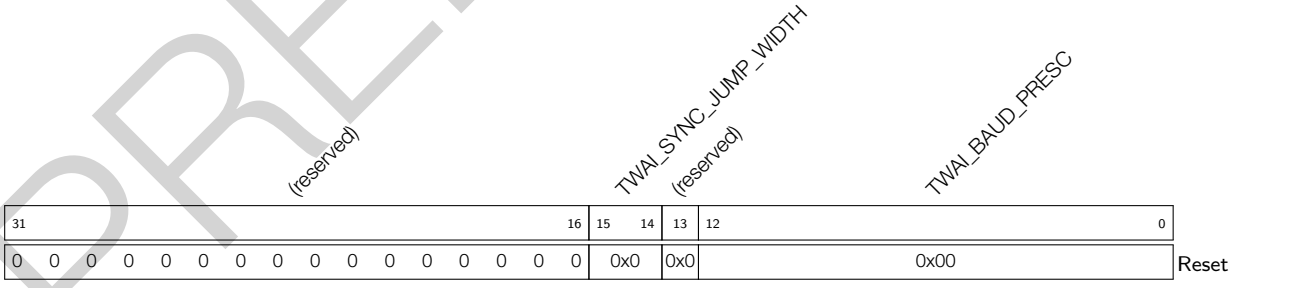
TWAI_RESET_MODE This bit is used to configure the operation mode of the TWAI Controller. 1: Reset mode; 0: Operation mode (R/W)

TWAI_LISTEN_ONLY_MODE 1: Listen only mode. In this mode the nodes will only receive messages from the bus, without generating the acknowledge signal nor updating the RX error counter. (R/W)

TWAI_SELF_TEST_MODE 1: Self test mode. In this mode the TX nodes can perform a successful transmission without receiving the acknowledge signal. This mode is often used to test a single node with the self reception request command. (R/W)

TWAI_RX_FILTER_MODE This bit is used to configure the filter mode. 0: Dual filter mode; 1: Single filter mode (R/W)

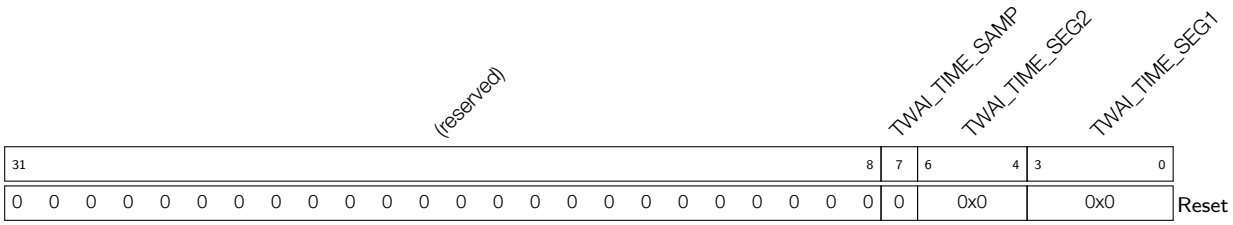
Register 14.2. TWAI_BUS_TIMING_0_REG (0x0018)



TWAI_BAUD_PRESC Baud Rate Prescaler value, determines the frequency dividing ratio. (RO | R/W)

TWAI_SYNC_JUMP_WIDTH Synchronization Jump Width (SJW), 1 ~ 14 Tq wide. (RO | R/W)

Register 14.3. TWAI_BUS_TIMING_1_REG (0x001C)

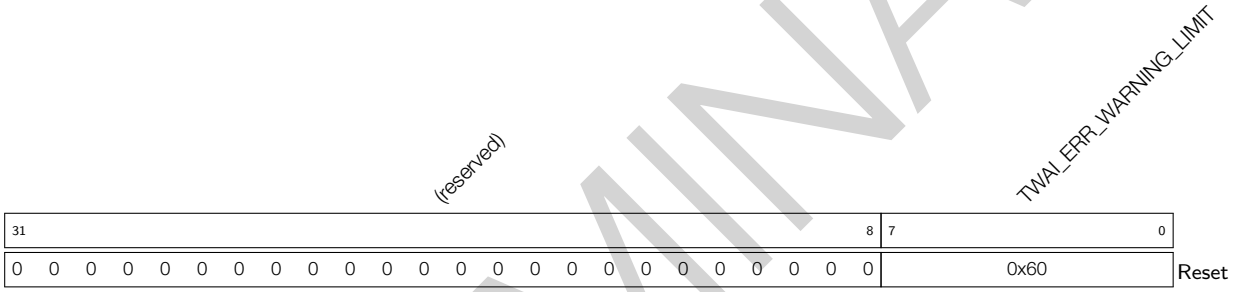


TWAI_TIME_SEG1 The width of PBS1. (RO | R/W)

TWAI_TIME_SEG2 The width of PBS2. (RO | R/W)

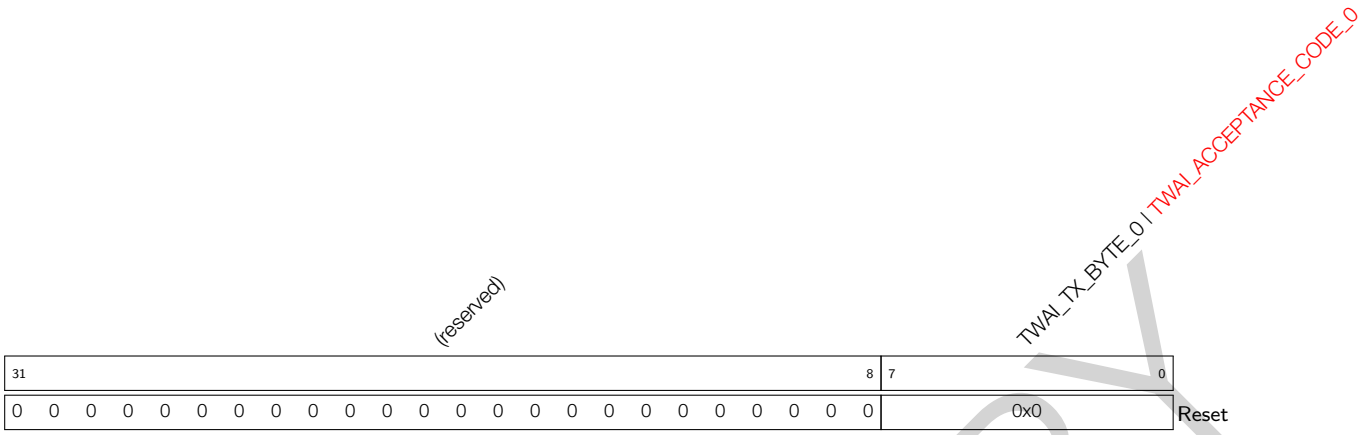
TWAI_TIME_SAMP The number of sample points. 0: the bus is sampled once; 1: the bus is sampled three times (RO | R/W)

Register 14.4. TWAI_ERR_WARNING_LIMIT_REG (0x0034)



TWAI_ERR_WARNING_LIMIT Error warning threshold. In the case when any of an error counter value exceeds the threshold, or all the error counter values are below the threshold, an error warning interrupt will be triggered (given the enable signal is valid). (RO | R/W)

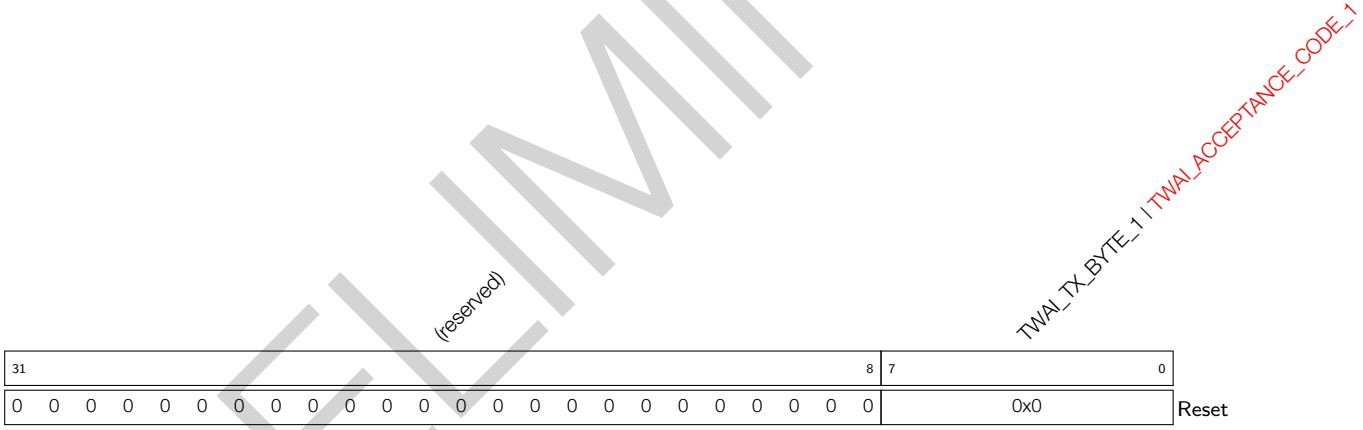
Register 14.5. TWAI_DATA_0_REG (0x0040)



TWAI_TX_BYTE_0 Stored the 0th byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_CODE_0 Stored the 0th byte of the filter code in reset mode. (R/W)

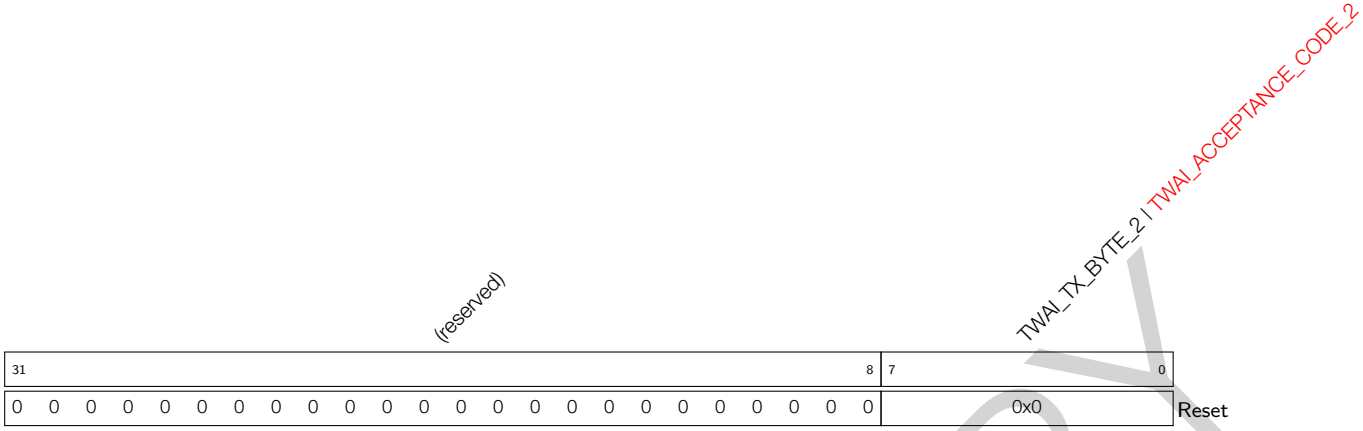
Register 14.6. TWAI_DATA_1_REG (0x0044)



TWAI_TX_BYTE_1 Stored the 1st byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_CODE_1 Stored the 1st byte of the filter code in reset mode. (R/W)

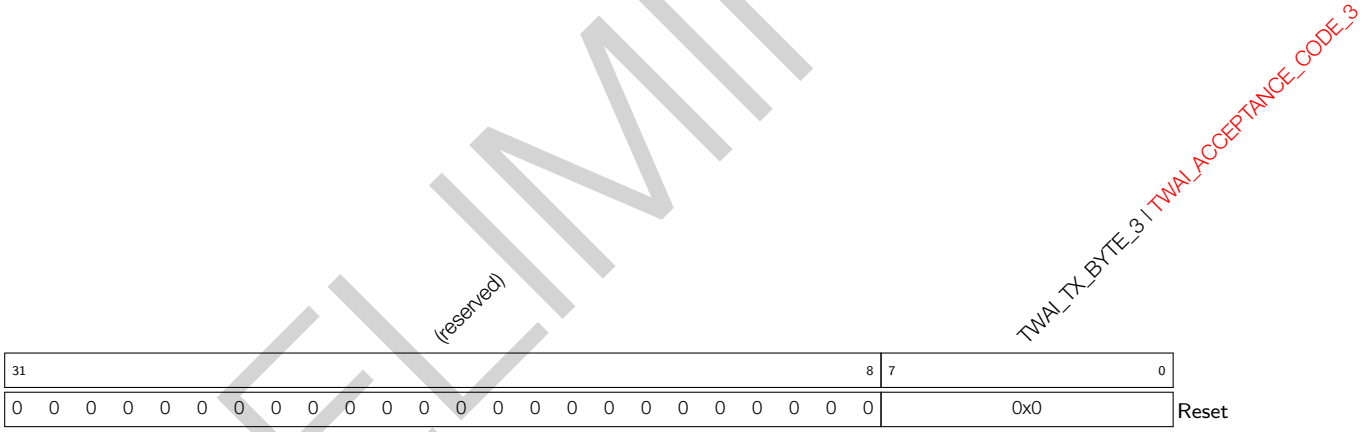
Register 14.7. TWAI_DATA_2_REG (0x0048)



TWAI_TX_BYTE_2 Stored the 2nd byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_CODE_2 Stored the 2nd byte of the filter code in reset mode. (R/W)

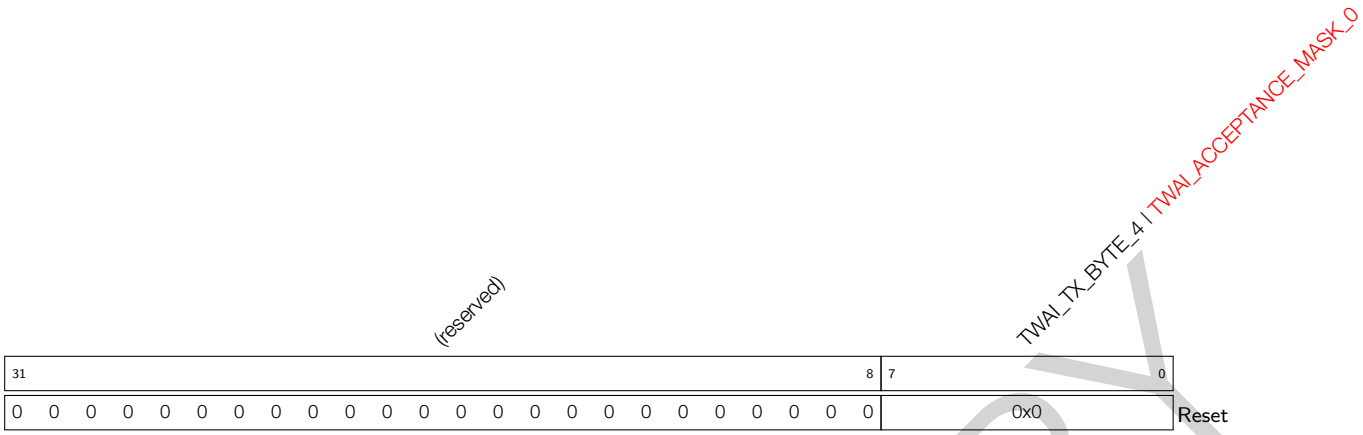
Register 14.8. TWAI_DATA_3_REG (0x004C)



TWAI_TX_BYTE_3 Stored the 3rd byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_CODE_3 Stored the 3rd byte of the filter code in reset mode. (R/W)

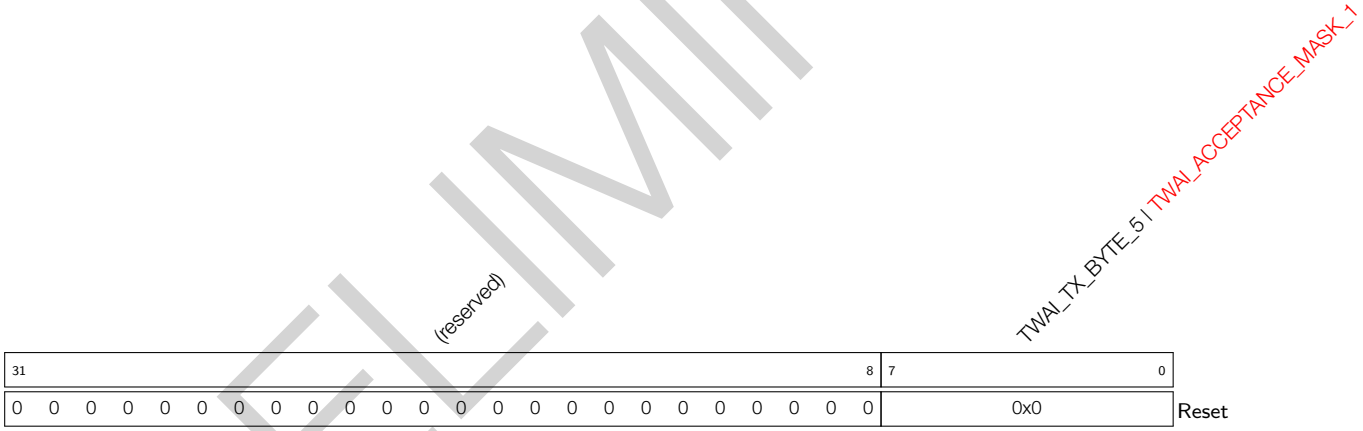
Register 14.9. TWAI_DATA_4_REG (0x0050)



TWAI_TX_BYTE_4 Stored the 4th byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_MASK_0 Stored the 0th byte of the filter code in reset mode. (R/W)

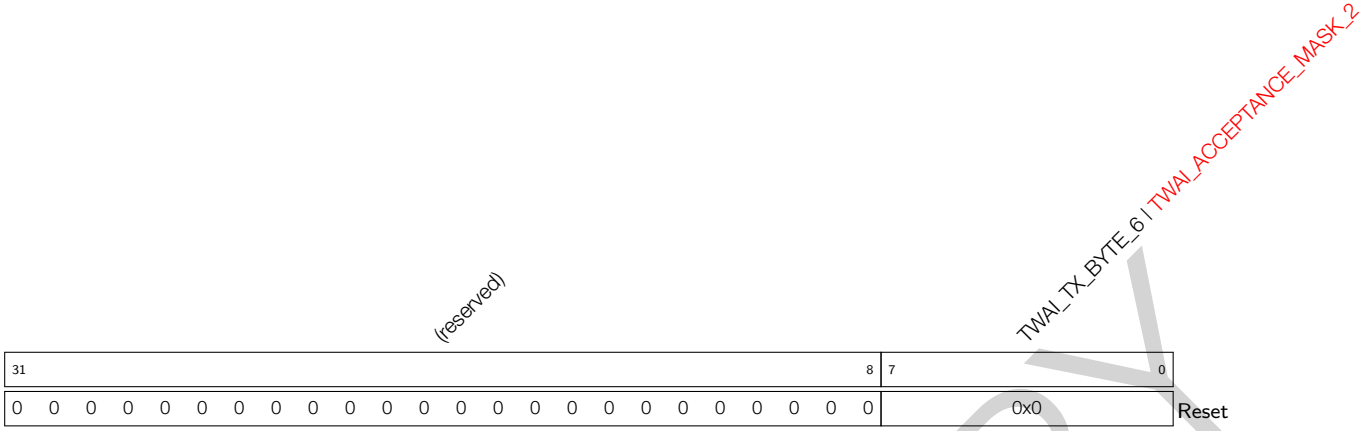
Register 14.10. TWAI_DATA_5_REG (0x0054)



TWAI_TX_BYTE_5 Stored the 5th byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_MASK_1 Stored the 1st byte of the filter code in reset mode. (R/W)

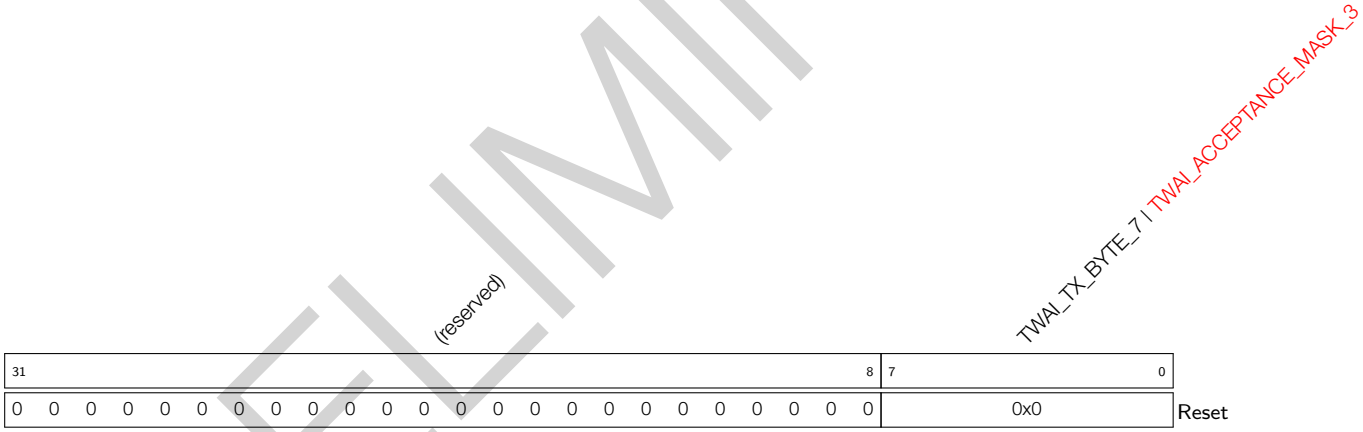
Register 14.11. TWAI_DATA_6_REG (0x0058)



TWAI_TX_BYTE_6 Stored the 6th byte information of the data to be transmitted in operation mode. (WO)

TWAI_ACCEPTANCE_MASK_2 Stored the 2nd byte of the filter code in reset mode. (R/W)

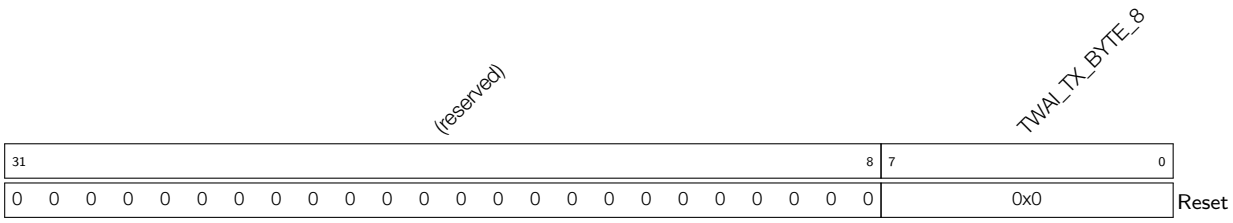
Register 14.12. TWAI_DATA_7_REG (0x005C)



TWAI_TX_BYTE_7 Stored the 7th byte information of the data to be transmitted in operation mode. (WO)

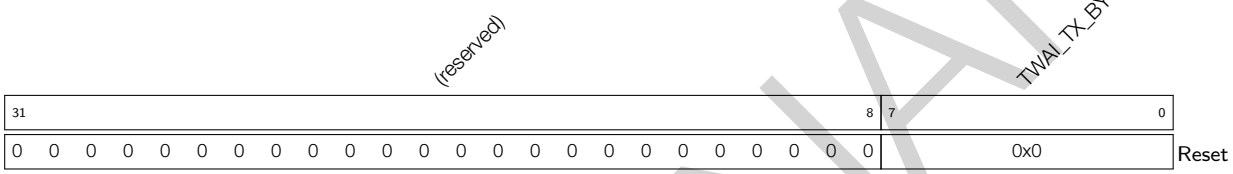
TWAI_ACCEPTANCE_MASK_3 Stored the 3rd byte of the filter code in reset mode. (R/W)

Register 14.13. TWAI_DATA_8_REG (0x0060)



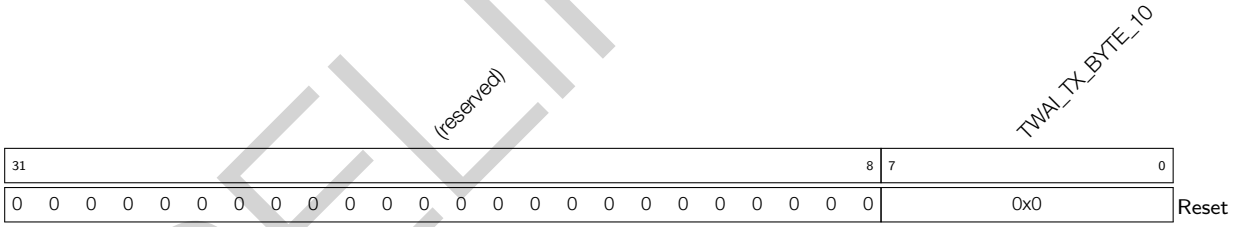
TWAI_TX_BYTE_8 Stored the 8th byte information of the data to be transmitted in operation mode. (WO)

Register 14.14. TWAI_DATA_9_REG (0x0064)



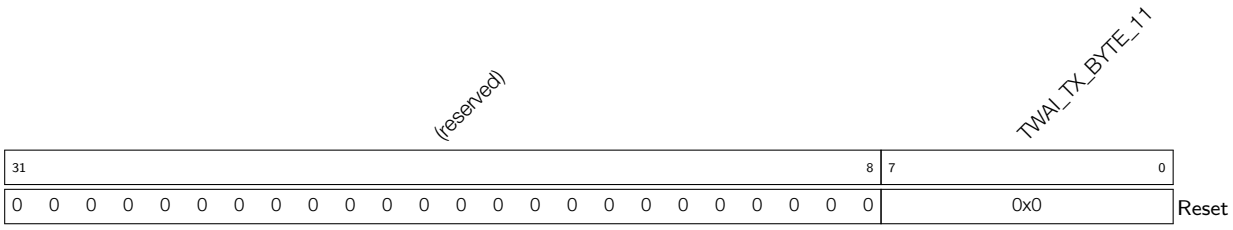
TWAI_TX_BYTE_9 Stored the 9th byte information of the data to be transmitted in operation mode. (WO)

Register 14.15. TWAI_DATA_10_REG (0x0068)



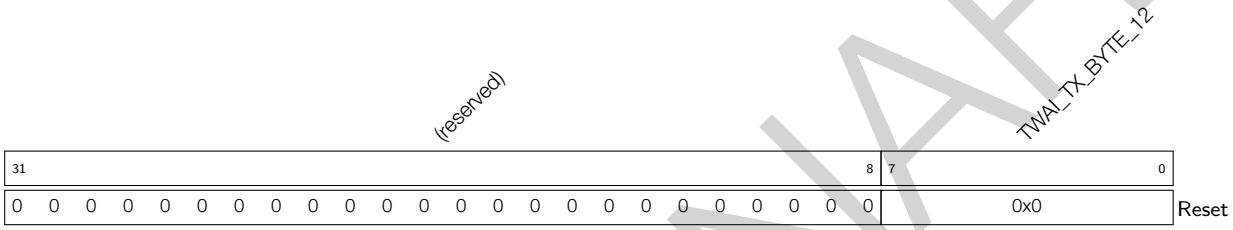
TWAI_TX_BYTE_10 Stored the 10th byte information of the data to be transmitted in operation mode. (WO)

Register 14.16. TWAI_DATA_11_REG (0x006C)



TWAI_TX_BYTE_11 Stored the 11th byte information of the data to be transmitted in operation mode. (WO)

Register 14.17. TWAI_DATA_12_REG (0x0070)



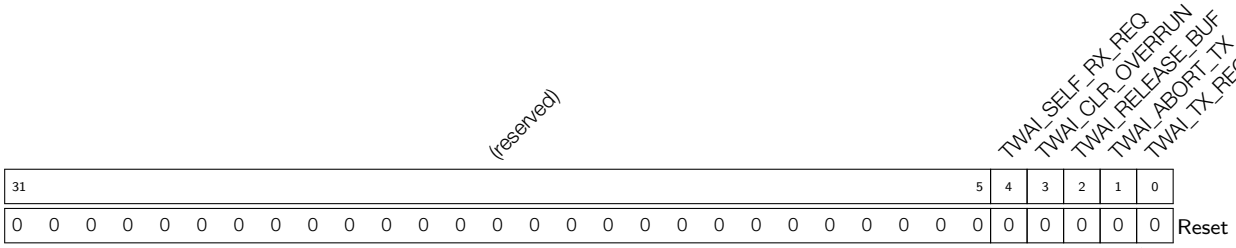
TWAI_TX_BYTE_12 Stored the 12th byte information of the data to be transmitted in operation mode. (WO)

Register 14.18. TWAI_CLOCK_DIVIDER_REG (0x007C)

TWAI_CD These bits are used to configure the divisor of the external CLKOUT pin. (R/W)

TWAI_CLOCK_OFF This bit can be configured in reset mode. 1: Disable the external CLKOUT pin; 0: Enable the external CLKOUT pin (RO | R/W)

Register 14.19. TWAI_CMD_REG (0x0004)



TWAI_TX_REQ Set the bit to 1 to drive nodes to start transmission. (WO)

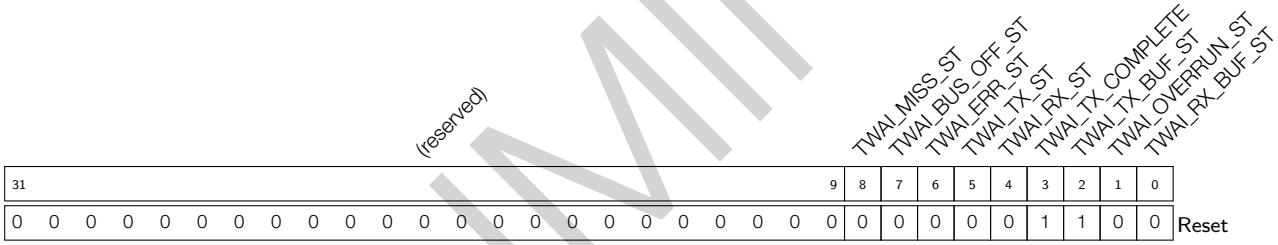
TWAI_ABORT_TX Set the bit to 1 to cancel a pending transmission request. (WO)

TWAI_RELEASE_BUF Set the bit to 1 to release the RX buffer. (WO)

TWAI_CLR_OVERRUN Set the bit to 1 to clear the data overrun status bit. (WO)

TWAI_SELF_RX_REQ Self reception request command. Set the bit to 1 to allow a message be transmitted and received simultaneously. (WO)

Register 14.20. TWAI_STATUS_REG (0x0008)



TWAI_RX_BUF_ST 1: The data in the RX buffer is not empty, with at least one received data packet. (RO)

TWAI_OVERRUN_ST 1: The RX FIFO is full and data overrun has occurred. (RO)

TWAI_TX_BUF_ST 1: The TX buffer is empty, the CPU may write a message into it. (RO)

TWAI_TX_COMPLETE 1: The TWAI controller has successfully received a packet from the bus. (RO)

TWAI_RX_ST 1: The TWAI Controller is receiving a message from the bus. (RO)

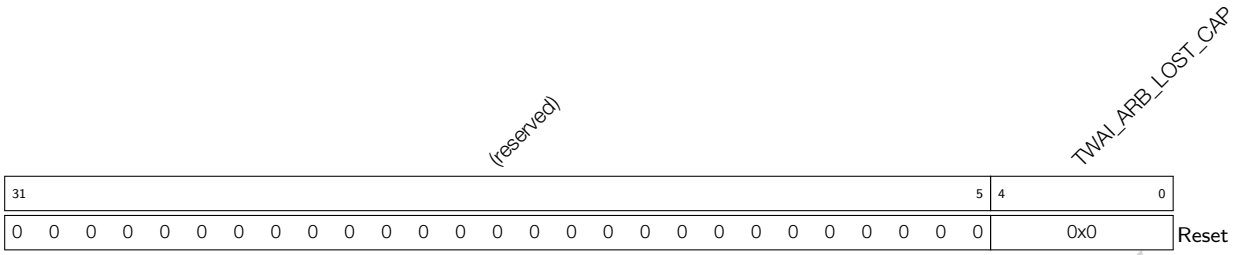
TWAI_TX_ST 1: The TWAI Controller is transmitting a message to the bus. (RO)

TWAI_ERR_ST 1: At least one of the RX/TX error counter has reached or exceeded the value set in register TWAI_ERR_WARNING_LIMIT_REG. (RO)

TWAI_BUS_OFF_ST 1: In bus-off status, the TWAI Controller is no longer involved in bus activities. (RO)

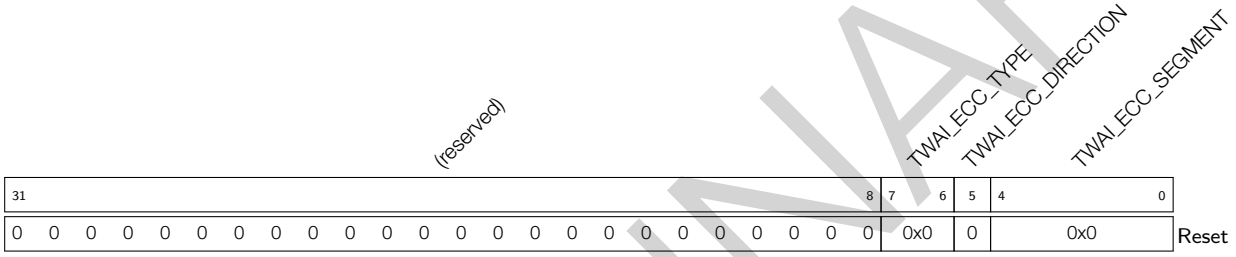
TWAI_MISS_ST This bit reflects whether the data packet in the RX FIFO is complete. 1: The current packet is missing; 0: The current packet is complete (RO)

Register 14.21. TWAI_ARB LOST CAP_REG (0x002C)



TWAI_ARB_LOST_CAP This register contains information about the bit position of lost arbitration. (RO)

Register 14.22. TWAI_ERR_CODE_CAP_REG (0x0030)

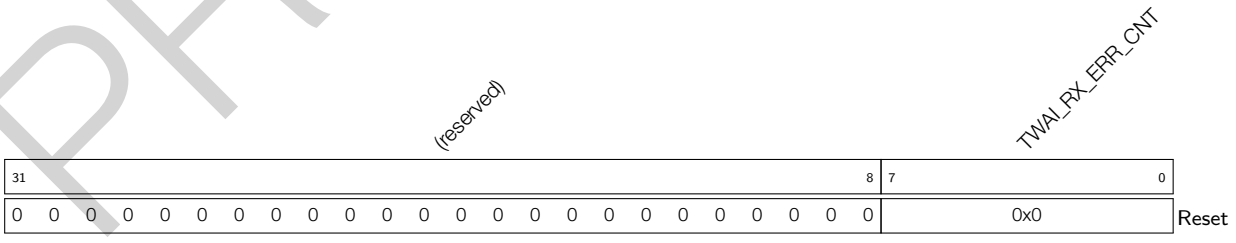


TWAI_ECC_SEGMENT This register contains information about the location of errors, see Table 14-16 for details. (RO)

TWAI_ECC_DIRECTION This register contains information about transmission direction of the node when error occurs. 1: Error occurs when receiving a message; 0: Error occurs when transmitting a message (RO)

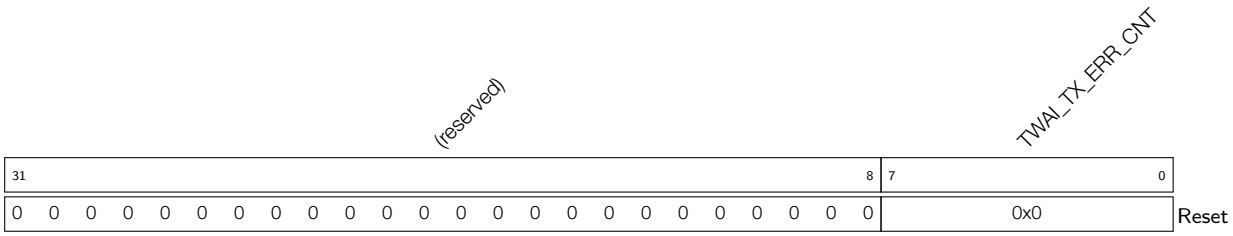
TWAI_ECC_TYPE This register contains information about error types: 00: bit error; 01: form error; 10: stuff error; 11: other type of error (RO)

Register 14.23. TWAI_RX_ERR_CNT_REG (0x0038)



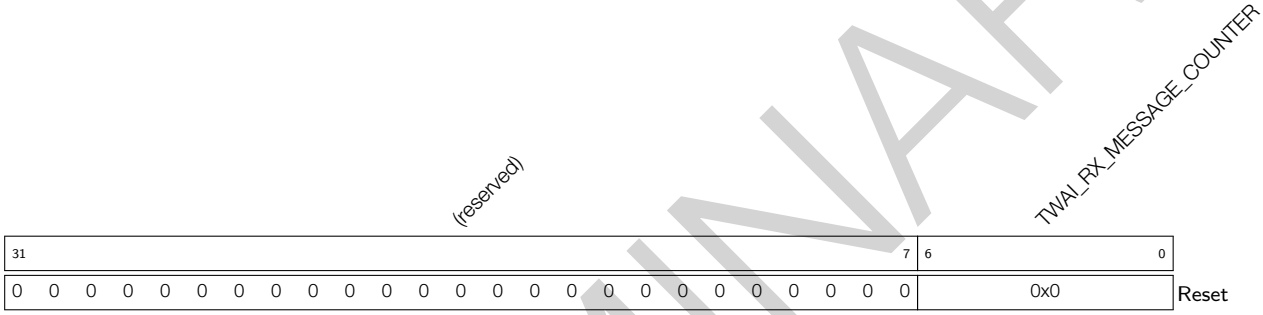
TWAI_RX_ERR_CNT The RX error counter register, reflects value changes in reception status. (RO | R/W)

Register 14.24. TWAI_TX_ERR_CNT_REG (0x003C)



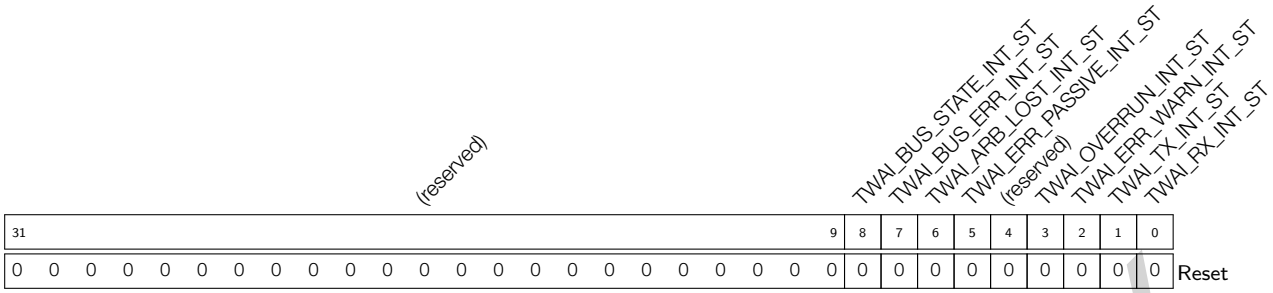
TWAI_TX_ERR_CNT The TX error counter register, reflects value changes in transmission status. (RO | R/W)

Register 14.25. TWAI_RX_MESSAGE_CNT_REG (0x0074)



TWAI_RX_MESSAGE_COUNTER This register reflects the number of messages available within the RX FIFO. (RO)

Register 14.26. TWAI_INT_RAW_REG (0x000C)



TWAI_RX_INT_ST Receive interrupt. If this bit is set to 1, it indicates there are messages to be handled in the RX FIFO. (RO)

TWAI_TX_INT_ST Transmit interrupt. If this bit is set to 1, it indicates the message transmission is finished and a new transmission is able to start. (RO)

TWAI_ERR_WARN_INT_ST Error warning interrupt. If this bit is set to 1, it indicates the error status signal and the bus-off status signal of Status register have changed (e.g., switched from 0 to 1 or from 1 to 0). (RO)

TWAI_OVERRUN_INT_ST Data overrun interrupt. If this bit is set to 1, it indicates a data overrun interrupt is generated in the RX FIFO. (RO)

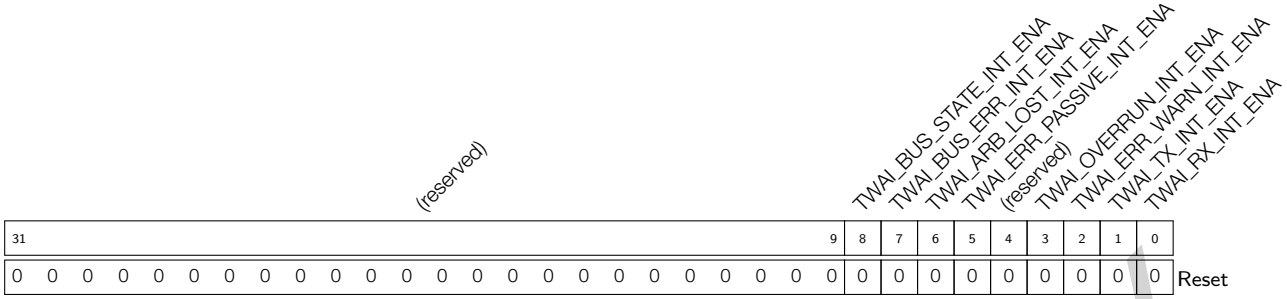
TWAI_ERR_PASSIVE_INT_ST Error passive interrupt. If this bit is set to 1, it indicates the TWAI Controller is switched between error active status and error passive status due to the change of error counters. (RO)

TWAI_ARB_LOST_INT_ST Arbitration lost interrupt. If this bit is set to 1, it indicates an arbitration lost interrupt is generated. (RO)

TWAI_BUS_ERR_INT_ST Error interrupt. If this bit is set to 1, it indicates an error is detected on the bus. (RO)

TWAI_BUS_STATE_INT_ST Bus state interrupt. If this bit is set to 1, it indicates the status of TWAI controller has changed. (RO)

Register 14.27. TWAI_INT ENA_REG (0x0010)



TWAI_RX_INT_ENA Set this bit to 1 to enable receive interrupt. (R/W)

TWAI_TX_INT_ENA Set this bit to 1 to enable transmit interrupt. (R/W)

TWAI_ERR_WARN_INT_ENA Set this bit to 1 to enable error warning interrupt. (R/W)

TWAI_OVERRUN_INT_ENA Set this bit to 1 to enable data overrun interrupt. (R/W)

TWAI_ERR_PASSIVE_INT_ENA Set this bit to 1 to enable error passive interrupt. (R/W)

TWAI_ARB_LOST_INT_ENA Set this bit to 1 to enable arbitration lost interrupt. (R/W)

TWAI_BUS_ERR_INT_ENA Set this bit to 1 to enable bus error interrupt. (R/W)

TWAI_BUS_STATE_INT_ENA Set this bit to 1 to enable bus state interrupt. (R/W)

15 LED PWM Controller (LEDC)

15.1 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.

15.2 Features

The LED PWM Controller has the following features:

- Six independent PWM generators (i.e. six channels)
- Four independent timers that support division by fractions
- Automatic duty cycle fading (i.e. gradual increase/decrease of a PWM's duty cycle without interference from the processor) with interrupt generation on fade completion
- Adjustable phase of PWM signal output
- PWM signal output in low-power mode (Light-sleep mode)
- Maximum PWM resolution: 14 bits

Note that the four timers are identical regarding their features and operation. The following sections refer to the timers collectively as Timer*x* (where *x* ranges from 0 to 3). Likewise, the 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).

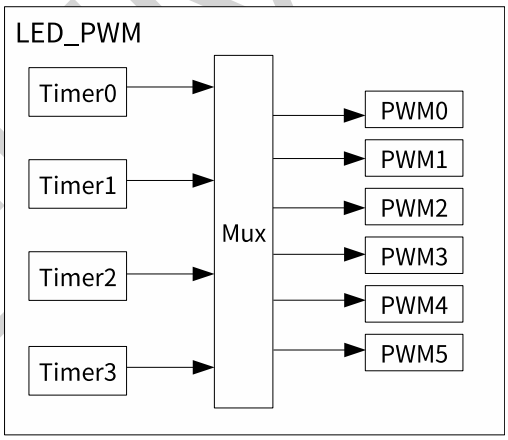


Figure 15-1. LED PWM Architecture

15.3 Functional Description

15.3.1 Architecture

Figure 15-1 shows the architecture of the LED PWM Controller.

The four timers can be independently configured (i.e. 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 15-2 illustrates the main functional blocks of the timer and the PWM generator.

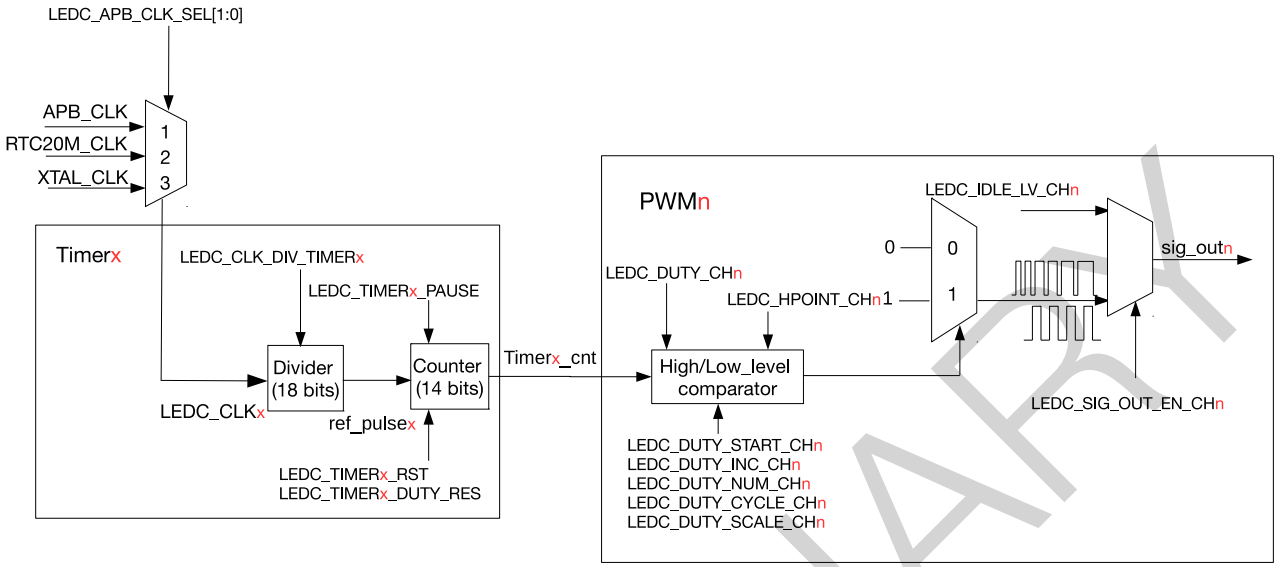


Figure 15-2. LED PWM Generator Diagram

15.3.2 Timers

Each timer in LED PWM Controller internally maintains a timebase counter. Referring to Figure 15-2, 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.

15.3.2.1 Clock Source

Software configuring registers for LED PWM is 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 18 in Chapter 9 *System Registers (SYSREG) [to be added later]*.

Timers in the LED PWM Controller choose their common clock source from one of the following clock signals: APB_CLK, RTC20M_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:

- APB_CLK: Set LEDC_APB_CLK_SEL[1:0] to 1
- RTC20M_CLK: Set LEDC_APB_CLK_SEL[1:0] to 2
- XTAL_CLK: Set LEDC_APB_CLK_SEL[1:0] to 3

The LEDC_CLK*x* signal will then be passed through the clock divider.

15.3.2.2 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 15-2).

The LEDC_CLK_DIV_TIMER*x* divider value is a fractional clock divider. Thus, it supports non-integer divider values. LEDC_CLK_DIV_TIMER*x* is configured via the LEDC_CLK_DIV_TIMER*x* field according to the following equation.

$$LEDC\_CLK\_DIV\_TIMERx = A + \frac{B}{256}$$

- *A* corresponds to the most significant 10 bits of LEDC_CLK_DIV_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 consist of (*A*+1) LEDC_CLK*x* clock pulses
- A number of (256-*B*) ref_pulse*x* clock pulses will consist of *A* LEDC_CLK*x* clock pulses
- The ref_pulse*x* clock pulses consisting of (*A*+1) pulses are evenly distributed amongst those consisting of *A* pulses

Figure 15-3 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*.

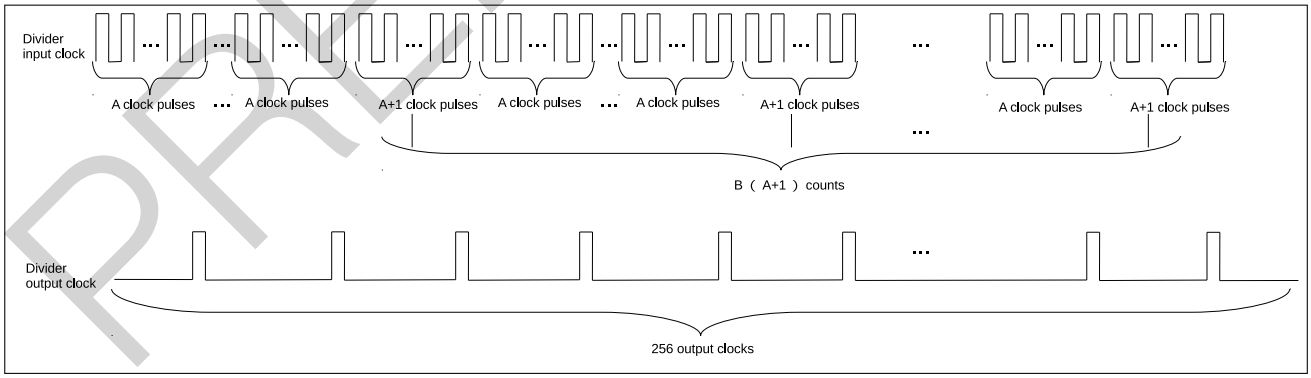


Figure 15-3. Frequency Division When LEDC_CLK_DIV_TIMER*x* is a Non-Integer Value

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.

15.3.2.3 14-bit Counter

Each timer contains a 14-bit timebase counter that uses ref_pulse*x* as its reference clock (see Figure 15-2). The LEDC_TIMER*x*_DUTY_RES field configures the overflow value of this 14-bit counter. Hence, the maximum resolution of the PWM signal is 14 bits. The counter counts up to $2^{LEDC\_TIMERx\_DUTY\_RES} - 1$, overflows and begins counting from 0 again. The counter's value can be read, reset, and suspended by software.

The counter can trigger LEDC_TIMER*x*_OVF_INT interrupt (generated automatically by hardware without configuration) every time the counter overflows. It can also be configured to trigger LEDC_OVF_CNT_CH*n*_INT interrupt after the counter overflows $LEDC\_OVF\_NUM\_CHn + 1$ times. To configure LEDC_OVF_CNT_CH*n*_INT interrupt, please:

1. Configure LEDC_TIMER_SEL_CH*n* as the counter for the PWM generator
2. Enable the counter by setting LEDC_OVF_CNT_EN_CH*n*
3. Set LEDC_OVF_NUM_CH*n* to the number of counter overflows to generate an interrupt, minus 1
4. Enable the overflow interrupt by setting LEDC_OVF_CNT_CH*n*_INT_ENA
5. Set LEDC_TIMER*x*_DUTY_RES to enable the timer and wait for a LEDC_OVF_CNT_CH*n*_INT interrupt

Referring to Figure 15-2, the frequency of a PWM generator output signal (sig_out*n*) is dependent on the frequency of the timer's clock source (LEDC_CLK*x*), the clock divider value (LEDC_CLK_DIV_TIMER*x*), and the range of the counter (LEDC_TIMER*x*_DUTY_RES):

$$f_{\text{PWM}} = \frac{f_{\text{LEDC\_CLK}x}}{\text{LEDC\_CLK\_DIV}x \cdot 2^{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}}}$$

To change the overflow value at runtime, first set the LEDC_TIMER*x*_DUTY_RES field, and then set the LEDC_TIMER*x*_PARA_UP field. This will cause the newly configured values to take effect upon the next overflow of the counter. If LEDC_OVF_CNT_EN_CH*n* field is reconfigured, LEDC_TIMER*x*_PARA_UP should also be set to apply the new configuration. In summary, these configuration values need to be updated by setting LEDC_TIMER*x*_PARA_UP. LEDC_TIMER*x*_PARA_UP field will be automatically cleared by hardware.

15.3.3 PWM Generators

To generate a PWM signal, a PWM generator (PWM*n*) selects a timer (Timer*x*). Each PWM generator can be configured separately by setting LEDC_TIMER_SEL_CH*n* to use one of four timers to generate the PWM output.

As shown in Figure 15-2, 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 15-4 illustrates how Hpoint*n* or Lpoint*n* are used to generate a fixed duty cycle PWM output signal.

For a particular PWM generator (PWM*n*), its Hpoint*n* is sampled from the LEDC_HPOINT_CH*n* field each time the selected timer's counter overflows. Likewise, Lpoint*n* is also sampled on every counter overflow and is calculated from the sum of the LEDC_DUTY_CH*n*[18:4] and LEDC_HPOINT_CH*n* fields. By setting Hpoint*n* and Lpoint*n* via

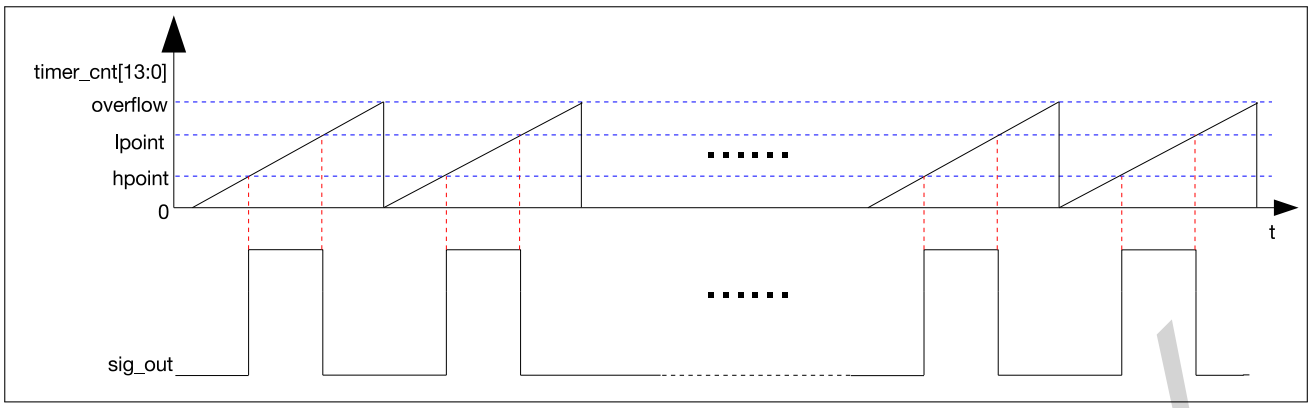


Figure 15-4. LED_PWM Output Signal Diagram

the LEDC_HPOINT_CH*n* and LEDC_DUTY_CH*n*[18:4] fields, the relative phase and duty cycle of the PWM output can be set.

The PWM output signal (sig_out*n*) is enabled by setting LEDC_SIG_OUT_EN_CH*n*. When LEDC_SIG_OUT_EN_CH*n* is cleared, PWM signal output is disabled, and the output signal (sig_out*n*) will output a constant level as specified by LEDC_IDLE_LV_CH*n*.

The bits LEDC_DUTY_CH*n*[3:0] are used to dither the duty cycles of the PWM output signal (sig_out*n*) by periodically altering the duty cycle of sig_out*n*. When LEDC_DUTY_CH*n*[3:0] is set to a non-zero value, then for every 16 cycles of sig_out*n*, LEDC_DUTY_CH*n*[3:0] of those cycles will have PWM pulses that are one timer tick longer than the other (16- LEDC_DUTY_CH*n*[3:0]) cycles. For instance, if LEDC_DUTY_CH*n*[18:4] is set to 10 and LEDC_DUTY_CH*n*[3:0] is set to 5, then 5 of 16 cycles will have a PWM pulse with a duty value of 11 and the rest of the 16 cycles will have a PWM pulse with a duty value of 10. The average duty cycle after 16 cycles is 10.3125.

If fields LEDC_TIMER_SEL_CH*n*, LEDC_HPOINT_CH*n*, LEDC_DUTY_CH*n*[18:4] and LEDC_SIG_OUT_EN_CH*n* are reconfigured, LEDC_PARA_UP_CH*n* must be set to apply the new configuration. This will cause the newly configured values to take effect upon the next overflow of the counter. LEDC_PARA_UP_CH*n* field will be automatically cleared by hardware.

15.3.4 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 15-5 illustrates Duty Cycle Fading.

Duty Cycle Fading is configured using the following register fields:

- LEDC_DUTY_CH*n* is used to set the initial value of Lpoint*n*
- LEDC_DUTY_START_CH*n* will enable/disable duty cycle fading when set/cleared
- LEDC_DUTY_CYCLE_CH*n* sets the number of counter overflow cycles for every Lpoint*n* increment/decrement. In other words, Lpoint*n* will be incremented/decremented after LEDC_DUTY_CYCLE_CH*n* counter overflows.
- LEDC_DUTY_INC_CH*n* configures whether Lpoint*n* is incremented/decremented if set/cleared
- LEDC_DUTY_SCALE_CH*n* sets the amount that Lpoint*n* is incremented/decremented

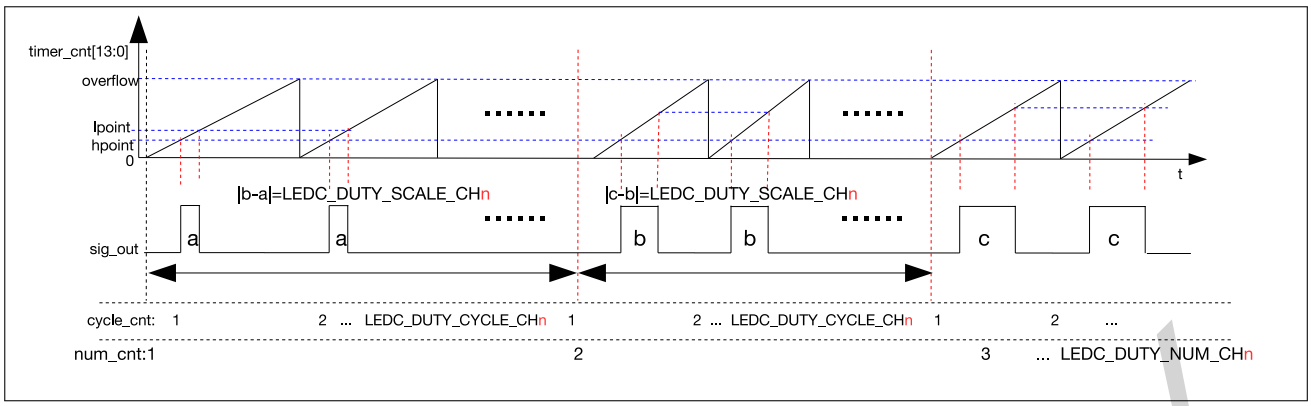


Figure 15-5. Output Signal Diagram of Fading Duty Cycle

- 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.

15.3.5 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.

15.4 Register Summary

The addresses in this section are relative to the LED PWM Controller base address provided in Table 3-4 in Chapter 3 *System and Memory*.

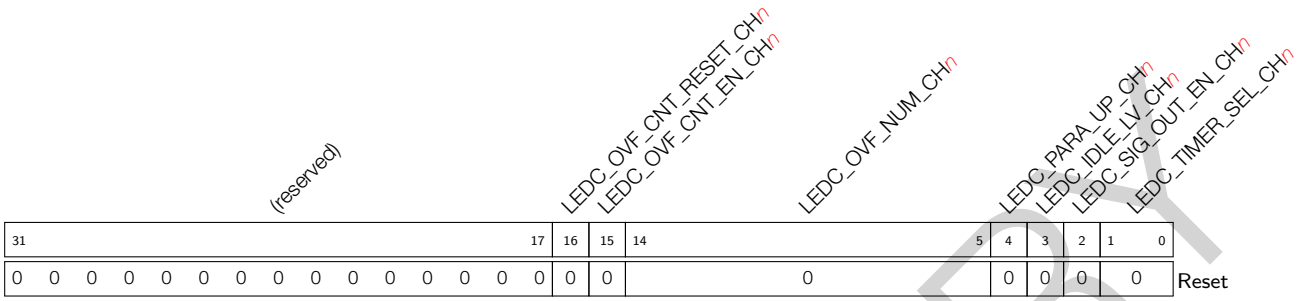
Name	Description	Address	Access
Configuration Register			
LEDC_CH0_CONF0_REG	Configuration register 0 for channel 0	0x0000	varies
LEDC_CH0_CONF1_REG	Configuration register 1 for channel 0	0x000C	varies
LEDC_CH1_CONF0_REG	Configuration register 0 for channel 1	0x0014	varies
LEDC_CH1_CONF1_REG	Configuration 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

15.5 Registers

The addresses in this section are relative to LED PWM Controller base address provided in Table 3-4 in Chapter 3 *System and Memory*.

Register 15.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+20n*)**



LEDC_TIMER_SEL_CH*n* This field is used to select one of the timers for channel *n*.

0: select Timer0; 1: select Timer1; 2: select Timer2; 3: select Timer3 (R/W)

LEDC_SIG_OUT_EN_CH*n* Set this bit to enable signal output on channel *n*. (R/W)

LEDC_IDLE_LV_CH*n* This bit is used to control the output value when channel *n* is inactive (when LEDC_SIG_OUT_EN_CH*n* is 0). (R/W)

LEDC_PARA_UP_CH*n* This bit is used to update the listed fields below for channel *n*, and will be automatically cleared by hardware. (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*

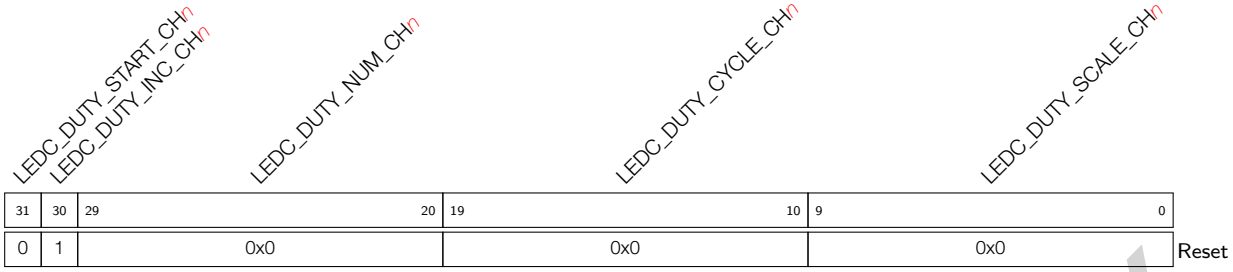
LEDC_OVF_NUM_CH*n* This field is used to configure the maximum times of overflow minus 1.

The LEDC_OVF_CNT_CH*n*_INT interrupt will be triggered when channel *n* overflows for (LEDC_OVF_NUM_CH*n* + 1) times. (R/W)

LEDC_OVF_CNT_EN_CH*n* This bit is used to count the number of times when the timer selected by channel *n* overflows. (R/W)

LEDC_OVF_CNT_RESET_CH*n* Set this bit to reset the timer-overflow counter of channel *n*. (WT)

Register 15.2. LEDC_CH*n*_CONF1_REG (*n*: 0-5) (0x000C+20n*)**



LEDC_DUTY_SCALE_CH*n* This field configures the step size of the duty cycle change during fading. (R/W)

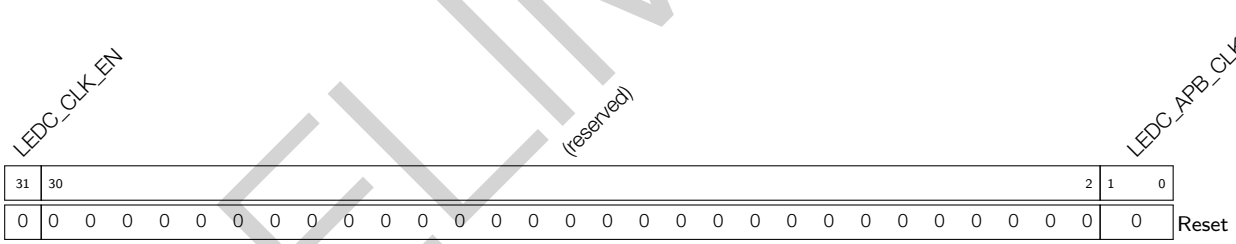
LEDC_DUTY_CYCLE_CH*n* The duty will change every LEDC_DUTY_CYCLE_CH*n* cycle on channel *n*. (R/W)

LEDC_DUTY_NUM_CH*n* This field controls the number of times the duty cycle will be changed. (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 15.3. LEDC_CONF_REG (0x00D0)



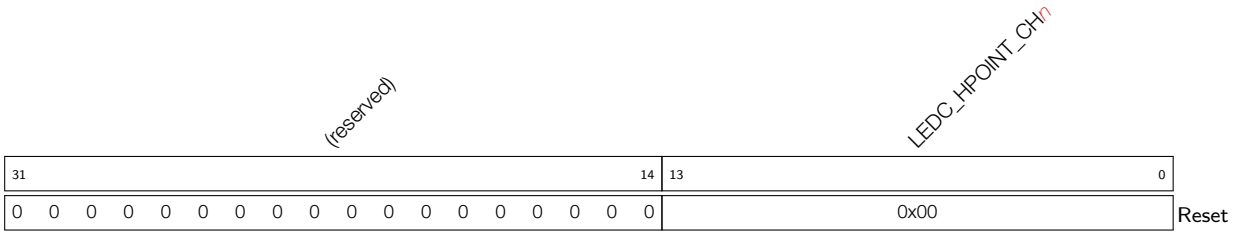
LEDC_APB_CLK_SEL This field is used to select the common clock source for all the 4 timers.

1: APB_CLK; 2: RTC20M_CLK; 3: XTAL_CLK. (R/W)

LEDC_CLK_EN This bit is used to control the clock.

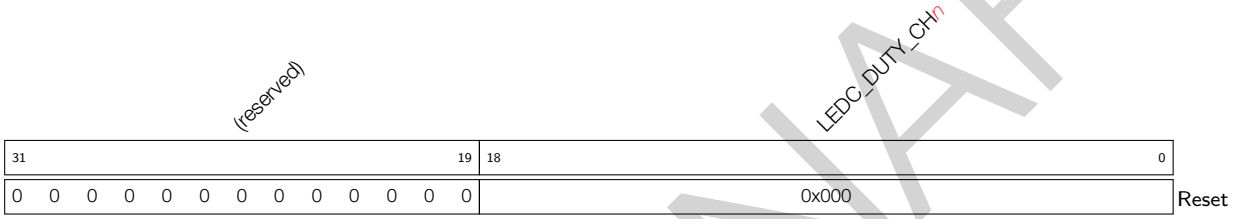
1: Force clock on for register. 0: Support clock only when application writes registers. (R/W)

Register 15.4. LEDC_CH*n*_HPOINT_REG (*n*: 0-5) (0x0004+20n*)**



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)

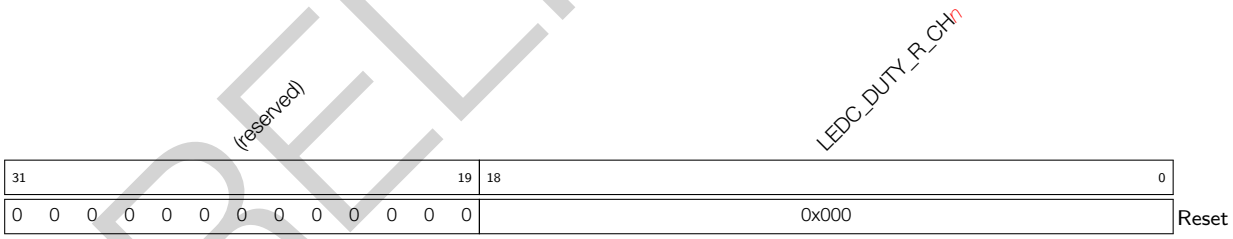
Register 15.5. LEDC_CH*n*_DUTY_REG (*n*: 0-5) (0x0008+20n*)**



LEDC_DUTY_CH*n* This field is used to change the output duty by controlling the Lpoint.

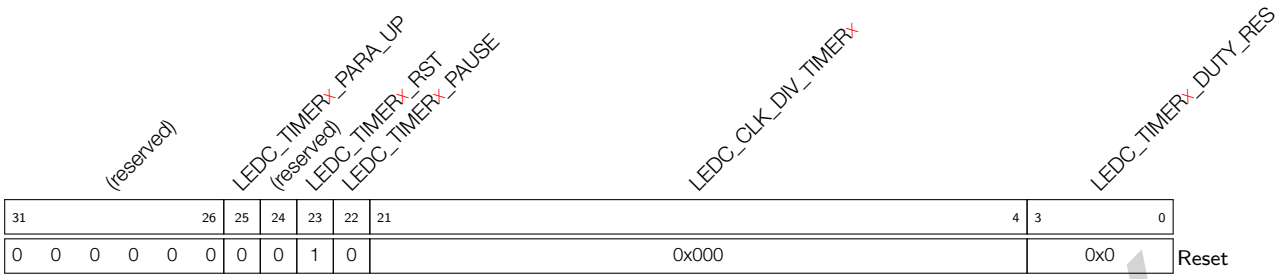
The output value turns to low when the selected timer for this channel has reached the Lpoint. (R/W)

Register 15.6. LEDC_CH*n*_DUTY_R_REG (*n*: 0-5) (0x0010+20n*)**



LEDC_DUTY_R_CH*n* This field stores the current duty cycle of the output signal on channel *n*. (RO)

Register 15.7. LEDC_TIMERx_CONF_REG (x: 0-3) (0x00A0+8*x)



LEDC_TIMERx_DUTY_RES This field is used to control the range of the counter in timer *x*. (R/W)

LEDC_CLK_DIV_TIMERx This field is used to configure the divisor for the divider in timer *x*.

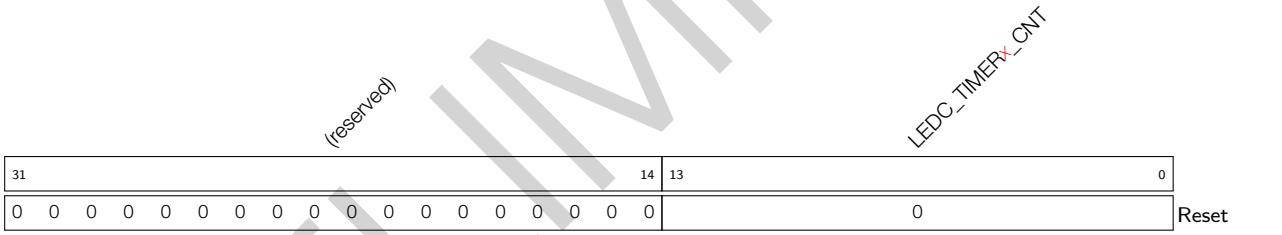
The least significant eight bits represent the fractional part. (R/W)

LEDC_TIMERx_PAUSE This bit is used to suspend the counter in timer *x*. (R/W)

LEDC_TIMERx_RST This bit is used to reset timer *x*. The counter will show 0 after reset. (R/W)

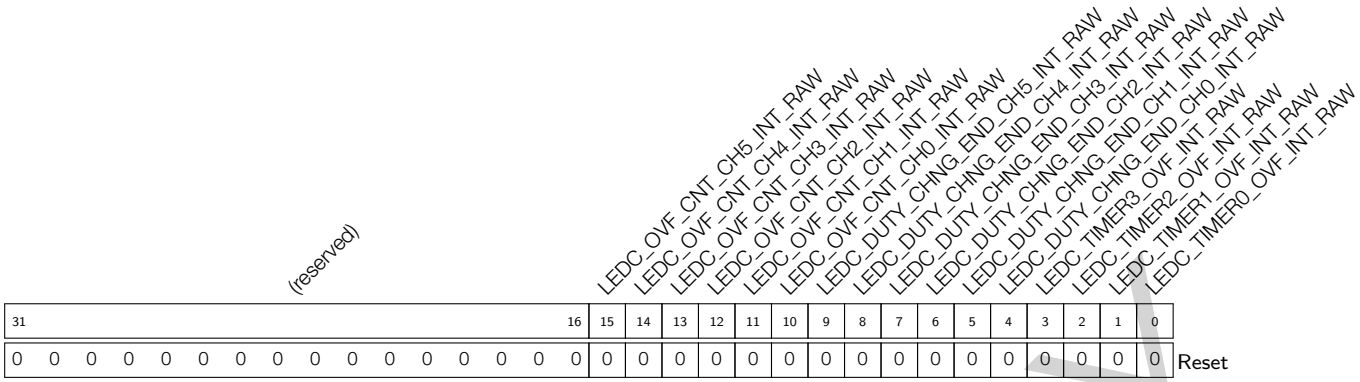
LEDC_TIMERx_PARA_UP Set this bit to update LEDC_CLK_DIV_TIMERx and LEDC_TIMERx_DUTY_RES. (WT)

Register 15.8. LEDC_TIMERx_VALUE_REG (x: 0-3) (0x00A4+8*x)



LEDC_TIMERx_CNT This field stores the current counter value of timer *x*. (RO)

Register 15.9. LEDC_INT_RAW_REG (0x00C0)

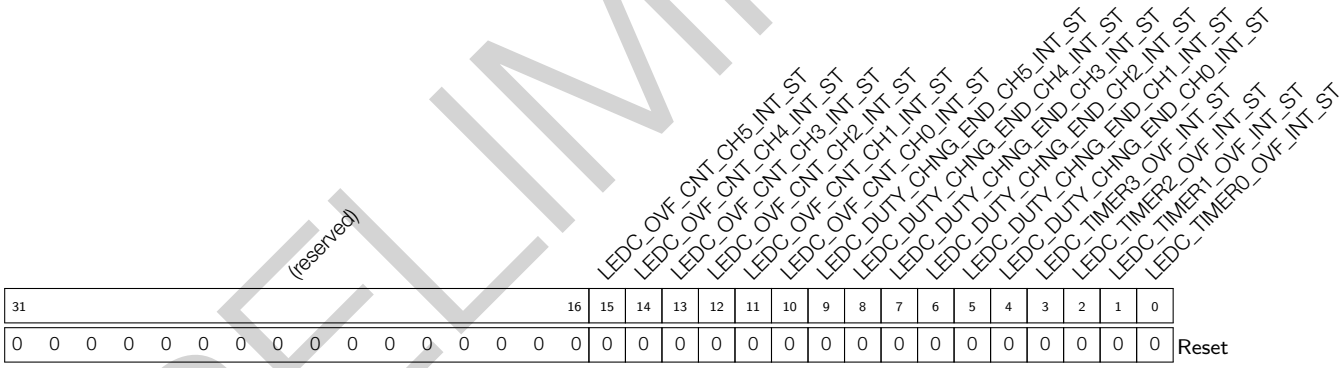


LEDC_TIMER*x*_OVF_INT_RAW Triggered when the timer*x* has reached its maximum counter value. (R/WTC/SS)

LEDC_DUTY_CHNG_END_CH*n*_INT_RAW Interrupt raw bit for channel *n*. Triggered when the gradual change of duty has finished. (R/WTC/SS)

LEDC_OVF_CNT_CH*n*_INT_RAW Interrupt raw bit for channel *n*. Triggered when the ovf_cnt has reached the value specified by LEDC_OVF_NUM_CH*n*. (R/WTC/SS)

Register 15.10. LEDC_INT_ST_REG (0x00C4)

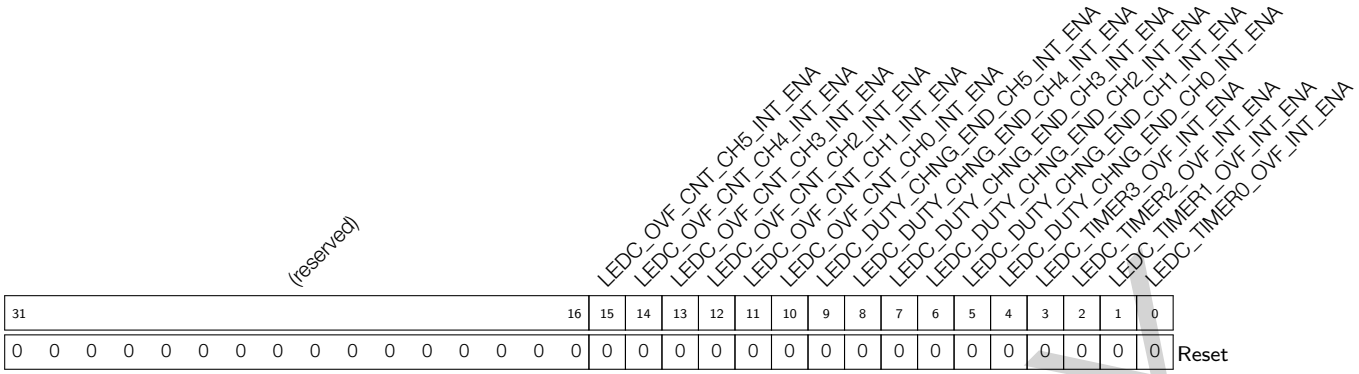


LEDC_TIMER*x*_OVF_INT_ST This is the masked interrupt status bit for the LEDC_TIMER*x*_OVF_INT interrupt when LEDC_TIMER*x*_OVF_INT_ENA is set to 1. (RO)

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 15.11. LEDC_INT_ENA_REG (0x00C8)

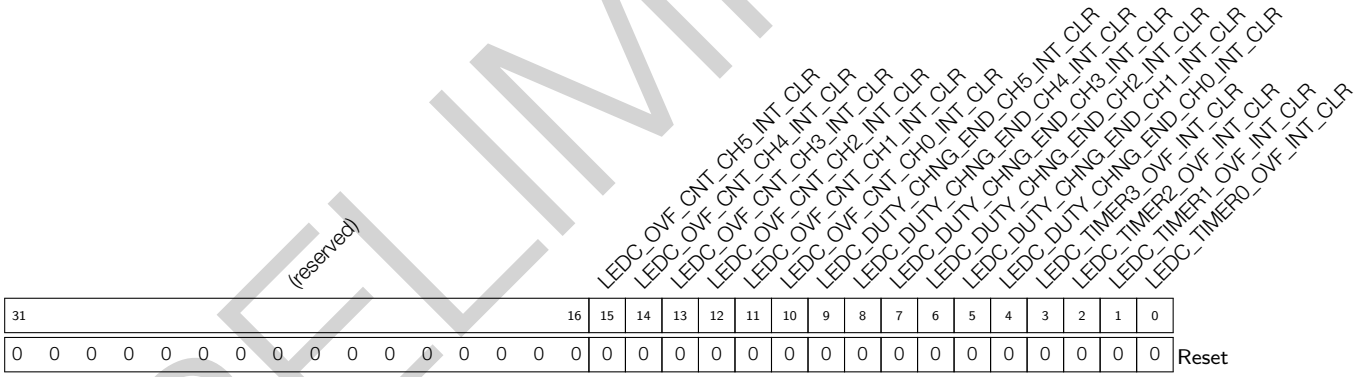


LEDC_TIMER*x*_OVF_INT_ENA The interrupt enable bit for the LEDC_TIMER*x*_OVF_INT interrupt. (R/W)

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 15.12. LEDC_INT_CLR_REG (0x00CC)

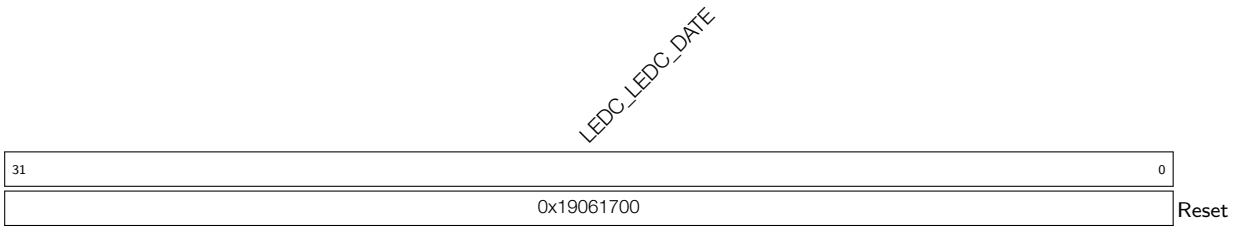


LEDC_TIMER*x*_OVF_INT_CLR Set this bit to clear the LEDC_TIMER*x*_OVF_INT interrupt. (WT)

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 15.13. LEDC_DATE_REG (0x00FC)



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

Glossary

Abbreviations for Peripherals

AES	AES (Advanced Encryption Standard) Accelerator
BOOTCTRL	Chip Boot Control
DS	Digital Signature
DMA	DMA (Direct Memory Access) Controller
eFuse	eFuse Controller
HMAC	HMAC (Hash-based Message Authentication Code) Accelerator
I2C	I2C (Inter-Integrated Circuit) Controller
I2S	I2S (Inter-IC Sound) Controller
LEDC	LED Control PWM (Pulse Width Modulation)
MCPWM	Motor Control PWM (Pulse Width Modulation)
PCNT	Pulse Count Controller
RMT	Remote Control Peripheral
RNG	Random Number Generator
RSA	RSA (Rivest Shamir Adleman) Accelerator
SDHOST	SD/MMC Host Controller
SHA	SHA (Secure Hash Algorithm) Accelerator
SPI	SPI (Serial Peripheral Interface) Controller
SYSTIMER	System Timer
TIMG	Timer Group
TWAI	Two-wire Automotive Interface
UART	UART (Universal Asynchronous Receiver-Transmitter) Controller
ULP Coprocessor	Ultra-low-power Coprocessor
USB OTG	USB On-The-Go
WDT	Watchdog Timers

Abbreviations for Registers

ISO	Isolation. When a module is power down, its output pins will be stuck in unknown state (some middle voltage). "ISO" registers will control to isolate its output pins to be a determined value, so it will not affect the status of other working modules which are not power down.
NMI	Non-maskable interrupt.
REG	Register.
R/W	Read/write. Software can read and write to these bits.
RO	Read-only. Software can only read these bits.
SYSREG	System Registers
WO	Write-only. Software can only write to these bits.

Revision History

Date	Version	Release notes
2021-04-08	V0.1	Preliminary release
2021-05-27	V0.2	Added the following chapters: • Chapter 4 *eFuse Controller (EFUSE)* • Chapter 13 *UART Controller (UART)* • Chapter 8 *Timer Group (TIMG)* • Chapter 2 *GDMA Controller (GDMA)* • Chapter 15 *LED PWM Controller (LEDC)* Updated the Chapter 5 *IO MUX and GPIO Matrix (GPIO, IO MUX)* Adjusted the order of chapters.

www.espressif.com

Disclaimer and Copyright Notice