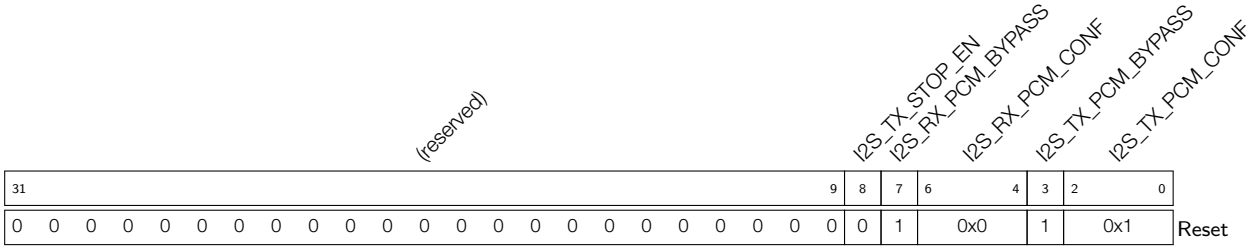
Register 12.28: I2S_CONF1_REG (0x00a0)



I2S_TX_STOP_EN Set this bit and the transmitter will stop transmitting BCK signal and WS signal when tx FIFO is empty. (R/W)

I2S_RX_PCM_BYPASS Set this bit to bypass the Compress/Decompress module for the received data. (R/W)

I2S_RX_PCM_CONF Compress/Decompress module configuration bit. (R/W)

0: Decompress received data;

1: Compress received data.

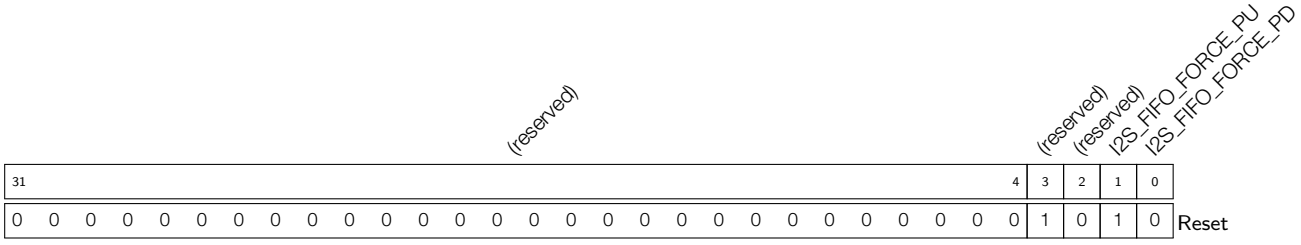
I2S_TX_PCM_BYPASS Set this bit to bypass the Compress/Decompress module for the transmitted data. (R/W)

I2S_TX_PCM_CONF Compress/Decompress module configuration bit. (R/W)

0: Decompress transmitted data;

1: Compress transmitted data.

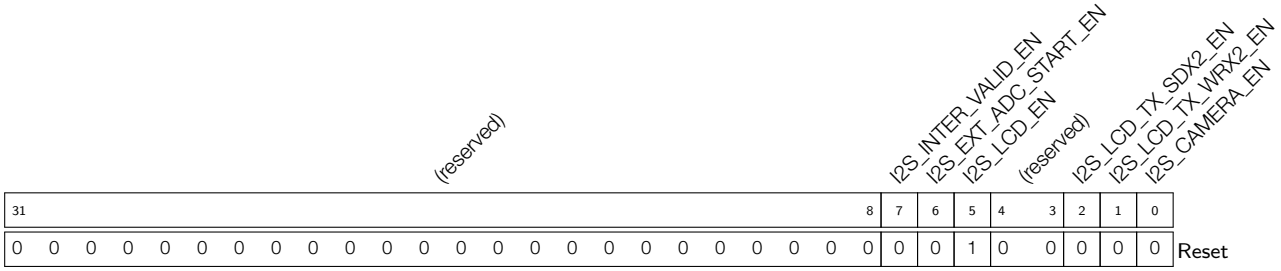
Register 12.29: I2S_PD_CONF_REG (0x00a4)



I2S_FIFO_FORCE_PU Force FIFO power-up. (R/W)

I2S_FIFO_FORCE_PD Force FIFO power-down. (R/W)

Register 12.30: I2S_CONF2_REG (0x00a8)



I2S_INTER_VALID_EN Set this bit to enable camera's internal validation. (R/W)

I2S_EXT_ADC_START_EN Set this bit to enable the start of external ADC . (R/W)

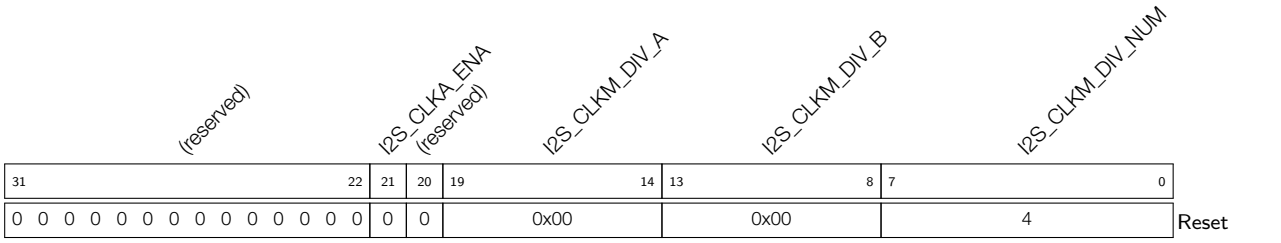
I2S_LCD_EN Set this bit to enable LCD mode. (R/W)

I2S_LCD_TX_SDX2_EN Set this bit to duplicate data pairs (Data Frame, Form 2) in LCD mode. (R/W)

I2S_LCD_TX_WRX2_EN One datum will be written twice in LCD mode. (R/W)

I2S_CAMERA_EN Set this bit to enable camera mode. (R/W)

Register 12.31: I2S_CLKM_CONF_REG (0x00ac)



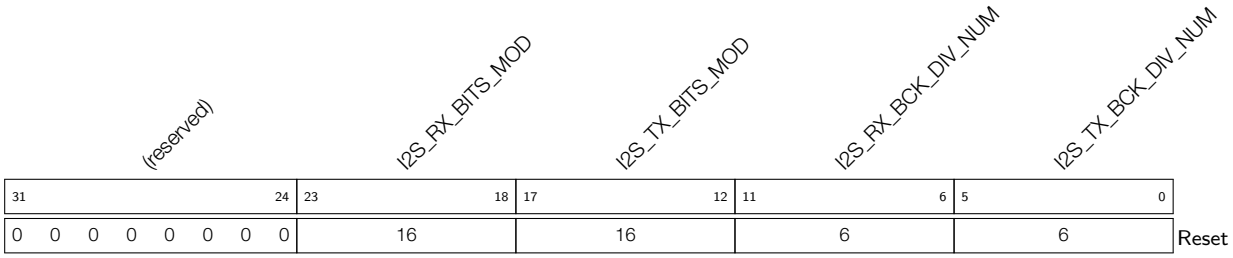
I2S_CLKA_ENA Set this bit to enable clk_apll. (R/W)

I2S_CLKM_DIV_A Fractional clock divider's denominator value. (R/W)

I2S_CLKM_DIV_B Fractional clock divider's numerator value. (R/W)

I2S_CLKM_DIV_NUM I2S clock divider's integral value. (R/W)

Register 12.32: I2S_SAMPLE_RATE_CONF_REG (0x00b0)



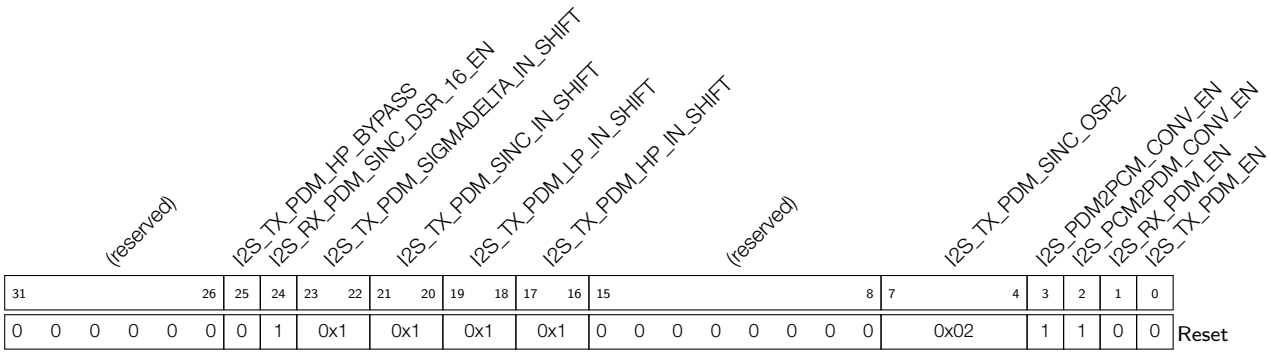
I2S_RX_BITS_MOD Set the bits to configure the bit length of I2S receiver channel. (R/W)

I2S_TX_BITS_MOD Set the bits to configure the bit length of I2S transmitter channel. (R/W)

I2S_RX_BCK_DIV_NUM Bit clock configuration bit in receiver mode. (R/W)

I2S_TX_BCK_DIV_NUM Bit clock configuration bit in transmitter mode. (R/W)

Register 12.33: I2S_PDM_CONF_REG (0x00b4)



I2S_TX_PDM_HP_BYPASS Set this bit to bypass the transmitter's PDM HP filter. (R/W)

I2S_RX_PDM_SINC_DSR_16_EN PDM downsampling rate for filter group 1 in receiver mode. (R/W)

1: downsampling rate = 128;

0: downsampling rate = 64.

I2S_TX_PDM_SIGMADELTA_IN_SHIFT Adjust the size of the input signal into filter module. (R/W)

0: divided by 2; 1: multiplied by 1; 2: multiplied by 2; 3: multiplied by 4.

I2S_TX_PDM_SINC_IN_SHIFT Adjust the size of the input signal into filter module. (R/W)

0: divided by 2; 1: multiplied by 1; 2: multiplied by 2; 3: multiplied by 4.

I2S_TX_PDM_LP_IN_SHIFT Adjust the size of the input signal into filter module. (R/W)

0: divided by 2; 1: multiplied by 1; 2: multiplied by 2; 3: multiplied by 4.

I2S_TX_PDM_HP_IN_SHIFT Adjust the size of the input signal into filter module. (R/W)

0: divided by 2; 1: multiplied by 1; 2: multiplied by 2; 3: multiplied by 4.

I2S_TX_PDM_SINC_OSR2 Upsampling rate = 64×i2s_tx_pdm_sinc_osr2 (R/W)

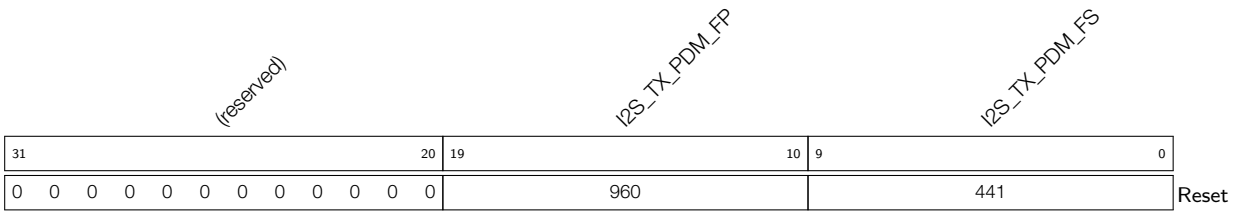
I2S_PDM2PCM_CONV_EN Set this bit to enable PDM-to-PCM converter. (R/W)

I2S_PCM2PDM_CONV_EN Set this bit to enable PCM-to-PDM converter. (R/W)

I2S_RX_PDM_EN Set this bit to enable receiver's PDM mode. (R/W)

I2S_TX_PDM_EN Set this bit to enable transmitter's PDM mode. (R/W)

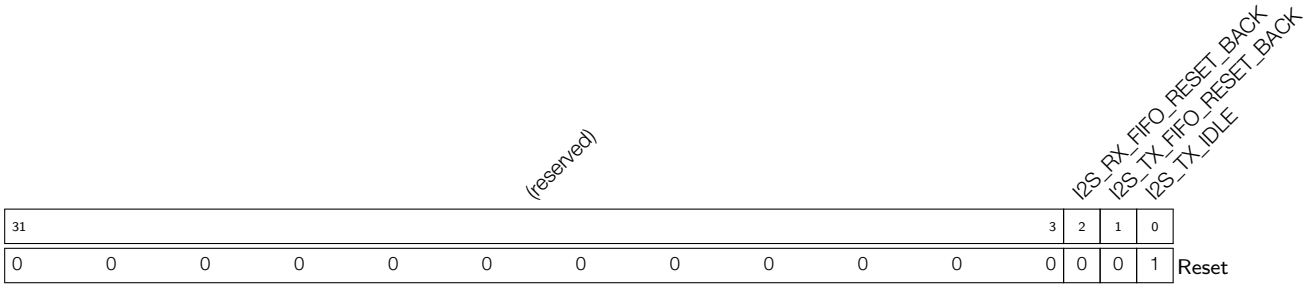
Register 12.34: I2S_PDM_FREQ_CONF_REG (0x00b8)



I2S_TX_PDM_FP PCM-to-PDM converter's PDM frequency parameter. (R/W)

I2S_TX_PDM_FS PCM-to-PDM converter's PCM frequency parameter. (R/W)

Register 12.35: I2S_STATE_REG (0x00bc)



I2S_RX_FIFO_RESET_BACK This bit is used to confirm if the Rx FIFO reset is done. 1: reset is not ready; 0: reset is ready. (RO)

I2S_TX_FIFO_RESET_BACK This bit is used to confirm if the Tx FIFO reset is done. 1: reset is not ready; 0: reset is ready. (RO)

I2S_TX_IDLE The status bit of the transmitter. 1: the transmitter is idle; 0: the transmitter is busy. (RO)

13. UART Controllers

13.1 Overview

Embedded applications often require a simple method of exchanging data between devices that need minimal system resources. The Universal Asynchronous Receiver/Transmitter (UART) is one such standard that can realize a flexible full-duplex data exchange among different devices. The three UART controllers available on a chip are compatible with UART-enabled devices from various manufacturers. The UART can also carry out an IrDA (Infrared Data Exchange), or function as an RS-485 modem.

All UART controllers integrated in the ESP32 feature an identical set of registers for ease of programming and flexibility. In this documentation, these controllers are referred to as UART*n*, where *n* = 0, 1, and 2, referring to UART0, UART1, and UART2, respectively.

13.2 UART Features

The UART modules have the following main features:

- Programmable baud rate
- 1024 x 8-bit RAM shared by three UART transmit-FIFOs and receive-FIFOs
- Supports input baud rate self-check
- Supports 5/6/7/8 bits of data length
- Supports 1/1.5/2/3/4 STOP bits
- Supports parity bit
- Supports RS485 Protocol
- Supports IrDA Protocol
- Supports DMA to communicate data in high speed
- Supports UART wake-up
- Supports both software and hardware flow control

13.3 Functional Description

13.3.1 Introduction

UART is a character-oriented data link that can be used to achieve communication between two devices. The asynchronous mode of transmission means that it is not necessary to add clocking information to the data being sent. This, in turn, requires that the data rate, STOP bits, parity, etc., be identical at the transmitting and receiving end for the devices to communicate successfully.

A typical UART frame begins with a START bit, followed by a “character” and an optional parity bit for error detection, and it ends with a STOP condition. The UART controllers available on the ESP32 provide hardware support for multiple lengths of data and STOP bits. In addition, the controllers support both software and hardware flow control, as well as DMA, for seamless high-speed data transfer. This allows the developer to employ multiple UART ports in the system with minimal software overhead.

13.3.2 UART Architecture

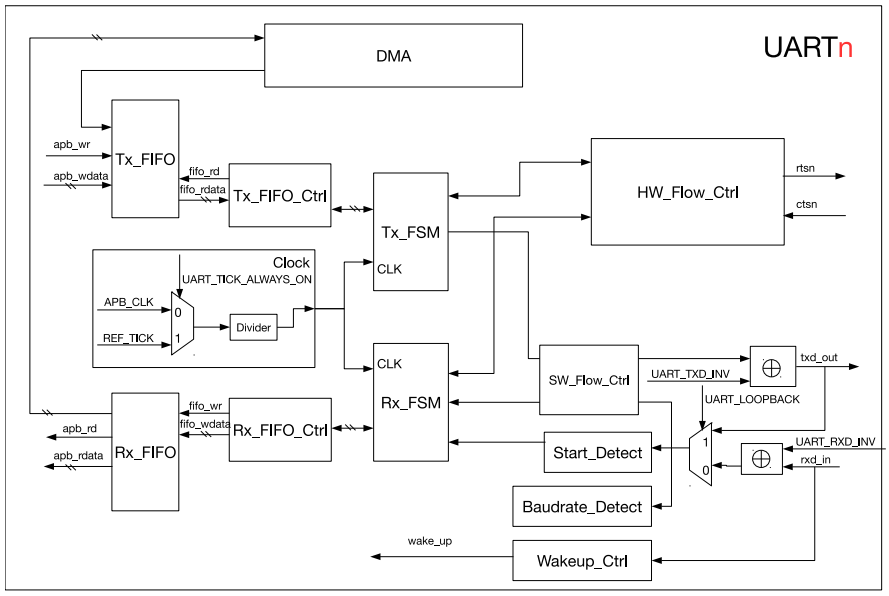


Figure 79: UART Basic Structure

Figure 79 shows the basic block diagram of the UART controller. The UART block can derive its clock from two sources: the 80-MHz APB_CLK, or the reference clock REF_TICK (please refer to Chapter Reset and Clock for more details). These two clock sources can be selected by configuring UART_TICK_REF_ALWAYS_ON.

Then, a divider in the clock path divides the selected clock source to generate clock signals that drive the UART module. UART_CLKDIV_REG contains the clock divider value in two parts — UART_CLKDIV (integral part) and UART_CLKDIV_FRAG (decimal part).

The UART controller can be further broken down into two functional blocks — the transmit block and the receive block.

The transmit block contains a transmit-FIFO buffer, which buffers data awaiting to be transmitted. Software can write Tx_FIFO via APB, and transmit data into Tx_FIFO via DMA. Tx_FIFO_Ctrl is used to control read- and write-access to the Tx_FIFO. When Tx_FIFO is not null, Tx_FSM reads data via Tx_FIFO_Ctrl, and transmits data out according to the set frame format. The outgoing bit stream can be inverted by appropriately configuring the register UART_TXD_INV.

The receive-block contains a receive-FIFO buffer, which buffers incoming data awaiting to be processed. The input bit stream, rxd_in, is fed to the UART controller. Negation of the input stream can be controlled by configuring the UART_RXD_INV register. Baudrate_Detect measures the baud rate of the input signal by measuring the minimum pulse width of the input bit stream. Start_Detect is used to detect a START bit in a frame of incoming data. After detecting the START bit, RX_FSM stores data retrieved from the received frame into Rx_FIFO through Rx_FIFO_Ctrl.

Software can read data in the Rx_FIFO through the APB. In order to free the CPU from engaging in data transfer operations, the DMA can be configured for sending or receiving data.

HW_Flow_Ctrl is able to control the data flow of rxd_in and txd_out through standard UART RTS and CTS flow control signals (rtsn_out and ctsn_in). SW_Flow_Ctrl controls the data flow by inserting special characters in the incoming and outgoing data flow. When UART is in Light-sleep mode (refer to Chapter Low-Power Management), Wakeup_Ctrl will start counting pulses in rxd_in. If the number of pulses is greater than UART_ACTIVE_THRESHOLD, a wake_up signal will be generated and sent to RTC. RTC will then wake up the

UART controller.

13.3.3 UART RAM

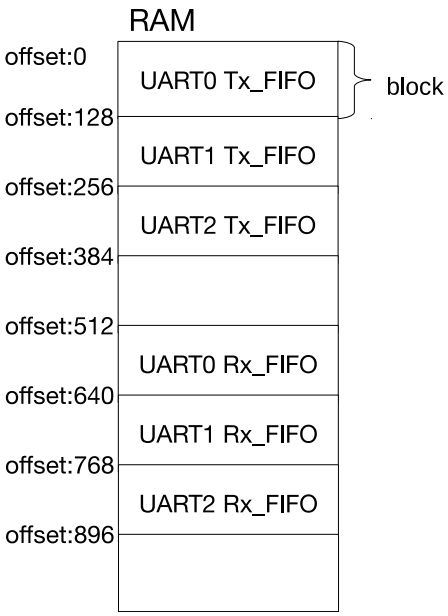


Figure 80: UART Shared RAM

Three UART controllers share a 1024 x 8-bit RAM space. As illustrated in Figure 80, RAM is allocated in different blocks. One block holds 128 x 8-bit data. Figure 80 illustrates the default RAM allocated to Tx_FIFO and Rx_FIFO of the three UART controllers. Tx_FIFO of UART*n* can be extended by setting UART*n*_TX_SIZE, while Rx_FIFO of UART*n* can be extended by setting UART*n*_RX_SIZE.

NOTICE: Extending the FIFO space of a UART controller may take up the FIFO space of another UART controller.

If none of the UART controllers is active, setting UART_MEM_PD, UART1_MEM_PD, and UART2_MEM_PD can prompt the RAM to enter low-power mode.

In UART0, bit UART_TXFIFO_RST and bit UART_RXFIFO_RST can be set to reset Tx_FIFO or Rx_FIFO, respectively. In UART1, bit UART1_TXFIFO_RST and bit UART1_RXFIFO_RST can be set to reset Tx_FIFO or Rx_FIFO, respectively.

Note:
UART2 doesn't have any register to reset Tx_FIFO or Rx_FIFO, and the UART1_TXFIFO_RST and UART1_RXFIFO_RST in UART1 may impact the functioning of UART2. Therefore, these 2 registers in UART1 should only be used when the Tx_FIFO and Rx_FIFO in UART2 do not have any data.

13.3.4 Baud Rate Detection

Setting UART_AUTOBAUD_EN for a UART controller will enable the baud rate detection function. The Baudrate_Detect block shown in Figure 79 can filter glitches with a pulse width lower than UART_GLITCH_FILT.

In order to use the baud rate detection feature, some random data should be sent to the receiver before starting

the UART communication stream. This is required so that the baud rate can be determined based on the pulse width. UART_LOWPULSE_MIN_CNT stores minimum low-pulse width, UART_HIGHPULSE_MIN_CNT stores minimum high-pulse width. By reading these two registers, software can calculate the baud rate of the transmitter.

13.3.5 UART Data Frame

Figure 81 shows the basic data frame structure. A data frame starts with a START condition and ends with a STOP condition. The START condition requires 1 bit and the STOP condition can be realized using 1/1.5/2/3/4-bit widths (as set by UART_BIT_NUM, UART_DL1_EN, and UAR_DL0_EN). The START is low level, while the STOP is high level.

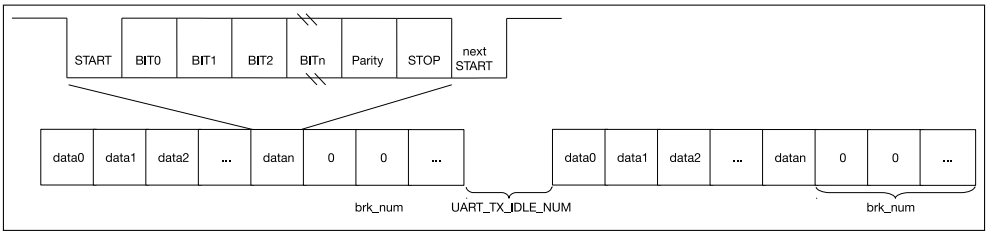


Figure 81: UART Data Frame Structure

The length of a character (BIT0 to BITn) can comprise 5 to 8 bits and can be configured by UART_BIT_NUM. When UART_PARITY_EN is set, the UART controller hardware will add the appropriate parity bit after the data. UART_PARITY is used to select odd parity or even parity. If the receiver detects an error in the input character, interrupt UART_PARITY_ERR_INT will be generated. If the receiver detects an error in the frame format, interrupt UART_FRM_ERR_INT will be generated.

Interrupt UART_TX_DONE_INT will be generated when all data in Tx_FIFO have been transmitted. When UART_TXD_BRK is set, the transmitter sends several NULL characters after the process of sending data is completed. The number of NULL characters can be configured by UART_TX_BRK_NUM. After the transmitter finishes sending all NULL characters, interrupt UART_TX_BRK_DONE_INT will be generated. The minimum interval between data frames can be configured with UART_TX_IDLE_NUM. If the idle time of a data frame is equal to, or larger than, the configured value of register UART_TX_IDLE_NUM, interrupt UART_TX_BRK_IDLE_DONE_INT will be generated.

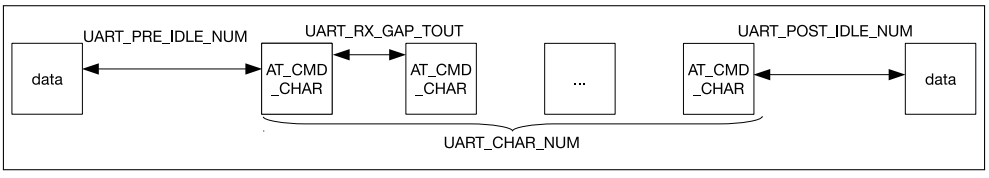


Figure 82: AT_CMD Character Format

Figure 82 shows a special AT_CMD character format. If the receiver constantly receives UART_AT_CMD_CHAR characters and these characters satisfy the following conditions, interrupt UART_AT_CMD_CHAR_DET_INT will be generated.

- Between the first UART_AT_CMD_CHAR and the last non-UART_AT_CMD_CHAR, there are at least UART_PER_IDLE_NUM APB clock cycles.
- Between every UART_AT_CMD_CHAR character there are at least UART_RX_GAP_TOUT APB clock cycles.

- The number of received UART_AT_CMD_CHAR characters must be equal to, or greater than, UART_CHAR_NUM.
- Between the last UART_AT_CMD_CHAR character received and the next non-UART_AT_CMD_CHAR, there are at least UART_POST_IDLE_NUM APB clock cycles.

13.3.6 Flow Control

UART controller supports both hardware and software flow control. Hardware flow control regulates data flow through input signal dsrn_in and output signal rtsn_out. Software flow control regulates data flow by inserting special characters in the flow of sent data and by detecting special characters in the flow of received data.

13.3.6.1 Hardware Flow Control

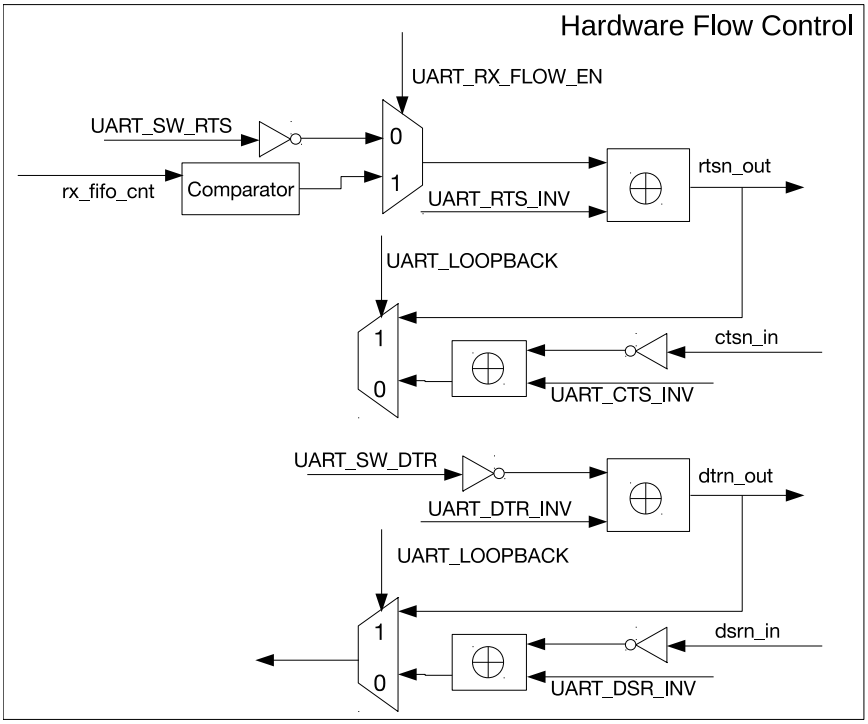


Figure 83: Hardware Flow Control

Figure 83 illustrates how the UART hardware flow control works. In hardware flow control, a high state of the output signal rtsn_out signifies that a data transmission is requested, while a low state of the same signal notifies the counterpart to stop data transmission until rtsn_out is pulled high again. There are two ways for a transmitter to realize hardware flow control:

- UART_RX_FLOW_EN is 0: The level of rtsn_out can be changed by configuring UART_SW_RTS.
- UART_RX_FLOW_EN is 1: If data in Rx_FIFO is greater than UART_RXFIFO_FULL_THRHD, the level of rtsn_out will be lowered.

If the UART controller detects an edge on ctsn_in, it will generate interrupt UART_CTS_CHG_INT and will stop transmitting data, once the current data transmission is completed.

The high level of the output signal dtrn_out signifies that the transmitter has finished data preparation. UART controller will generate interrupt UART_DSR_CHG_INT, after it detects an edge on the input signal dsrn_in. After

the software detects the above-mentioned interrupt, the input signal level of dsrn_in can be figured out by reading UART_DSRN. The software then decides whether it is able to receive data at that time or not.

Setting UART_LOOPBACK will enable the UART loopback detection function. In this mode, the output signal txd_out of UART is connected to its input signal rxd_in, rtsn_out is connected to ctsn_in, and dtrn_out is connected to dsrn_out. If the data transmitted corresponds to the data received, UART is able to transmit and receive data normally.

13.3.6.2 Software Flow Control

Software can force the transmitter to stop transmitting data by setting UART_FORCE_XOFF, as well as force the transmitter to continue sending data by setting UART_FORCE_XON.

UART can also control the software flow by transmitting special characters. Setting UART_SW_FLOW_CON_EN will enable the software flow control function. If the number of data bytes that UART has received exceeds that of the UART_XOFF threshold, the UART controller can send UART_XOFF_CHAR to instruct its counterpart to stop data transmission.

When UART_SW_FLOW_CON_EN is 1, software can send flow control characters at any time. When UART_SEND_XOFF is set, the transmitter will insert a UART_XOFF_CHAR and send it after the current data transmission is completed. When UART_SEND_XON is set, the transmitter will insert a UART_XON_CHAR and send it after the current data transmission is completed.

13.3.7 UART DMA

For information on the UART DMA, please refer to Chapter DMA Controller.

13.3.8 UART Interrupts

- UART_AT_CMD_CHAR_DET_INT: Triggered when the receiver detects the configured at_cmd char.
- UART_RS485_CLASH_INT: Triggered when a collision is detected between transmitter and receiver in RS-485 mode.
- UART_RS485_FRM_ERR_INT: Triggered when a data frame error is detected in RS-485.
- UART_RS485_PARITY_ERR_INT: Triggered when a parity error is detected in RS-485 mode.
- UART_TX_DONE_INT: Triggered when the transmitter has sent out all FIFO data.
- UART_TX_BRK_IDLE_DONE_INT: Triggered when the transmitter's idle state has been kept to a minimum after sending the last data.
- UART_TX_BRK_DONE_INT: Triggered when the transmitter completes sending NULL characters, after all data in transmit-FIFO are sent.
- UART_GLITCH_DET_INT: Triggered when the receiver detects a START bit.
- UART_SW_XOFF_INT: Triggered, if the receiver gets an Xon char when uart_sw_flow_con_en is set to 1.
- UART_SW_XON_INT: Triggered, if the receiver gets an Xoff char when uart_sw_flow_con_en is set to 1.
- UART_RXFIFO_TOUT_INT: Triggered when the receiver takes more time than rx_tout_thrhd to receive a byte.

- UART_BRK_DET_INT: Triggered when the receiver detects a 0 level after the STOP bit.
- UART_CTS_CHG_INT: Triggered when the receiver detects an edge change of the CTSn signal.
- UART_DSR_CHG_INT: Triggered when the receiver detects an edge change of the DSRn signal.
- UART_RXFIFO_OVF_INT: Triggered when the receiver gets more data than the FIFO can store.
- 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 in the data.
- UART_TXFIFO_EMPTY_INT: Triggered when the amount of data in the transmit-FIFO is less than what tx_mem_cnttxfifo_cnt specifies.
- UART_RXFIFO_FULL_INT: Triggered when the receiver gets more data than what (rx_flow_thrhd_h3, rx_flow_thrhd) specifies.

13.3.9 UCHI Interrupts

- UHCI_SEND_A_REG_Q_INT: When using the always_send registers to send a series of short packets, this is triggered when DMA has sent a short packet.
- UHCI_SEND_S_REG_Q_INT: When using the single_send registers to send a series of short packets, this is triggered when DMA has sent a short packet.
- UHCI_OUT_TOTAL_EOF_INT: Triggered when all data have been sent.
- UHCI_OUTLINK_EOF_ERR_INT: Triggered when there are some errors in EOF in the outlink descriptor.
- UHCI_IN_DSCR_EMPTY_INT: Triggered when there are not enough inlinks for DMA.
- UHCI_OUT_DSCR_ERR_INT: Triggered when there are some errors in the inlink descriptor.
- UHCI_IN_DSCR_ERR_INT: Triggered when there are some errors in the outlink descriptor.
- UHCI_OUT_EOF_INT: Triggered when the current descriptor's EOF bit is 1.
- UHCI_OUT_DONE_INT: Triggered when an outlink descriptor is completed.
- UHCI_IN_ERR_EOF_INT: Triggered when there are some errors in EOF in the inlink descriptor.
- UHCI_IN_SUC_EOF_INT: Triggered when a data packet has been received.
- UHCI_IN_DONE_INT: Triggered when an inlink descriptor has been completed.
- UHCI_TX_HUNG_INT: Triggered when DMA takes much time to read data from RAM.
- UHCI_RX_HUNG_INT: Triggered when DMA takes much time to receive data .
- UHCI_TX_START_INT: Triggered when DMA detects a separator char.
- UHCI_RX_START_INT: Triggered when a separator char has been sent.

13.4 Register Summary

Name	Description	UART0	UART1	UART2	Acc
Configuration registers					
UART_CONF0_REG	Configuration register 0	0x3FF40020	0x3FF50020	0x3FF6E020	R/W

UART_CONF1_REG	Configuration register 1	0x3FF40024	0x3FF50024	0x3FF6E024	R/W
UART_CLKDIV_REG	Clock divider configuration	0x3FF40014	0x3FF50014	0x3FF6E014	R/W
UART_FLOW_CONF_REG	Software flow-control configuration	0x3FF40034	0x3FF50034	0x3FF6E034	R/W
UART_SWFC_CONF_REG	Software flow-control character configuration	0x3FF4003C	0x3FF5003C	0x3FF6E03C	R/W
UART_SLEEP_CONF_REG	Sleep-mode configuration	0x3FF40038	0x3FF50038	0x3FF6E038	R/W
UART_IDLE_CONF_REG	Frame-end idle configuration	0x3FF40040	0x3FF50040	0x3FF6E040	R/W
UART_RS485_CONF_REG	RS485 mode configuration	0x3FF40044	0x3FF50044	0x3FF6E044	R/W
Status registers					
UART_STATUS_REG	UART status register	0x3FF4001C	0x3FF5001C	0x3FF6E01C	RO
Autobaud registers					
UART_AUTOBAUD_REG	Autobaud configuration register	0x3FF40018	0x3FF50018	0x3FF6E018	R/W
UART_LOWPULSE_REG	Autobaud minimum low pulse duration register	0x3FF40028	0x3FF50028	0x3FF6E028	RO
UART_HIGHPULSE_REG	Autobaud minimum high pulse duration register	0x3FF4002C	0x3FF5002C	0x3FF6E02C	RO
UART_POSPULSE_REG	Autobaud high pulse register	0x3FF40068	0x3FF50068	0x3FF6E068	RO
UART_NEGPULSE_REG	Autobaud low pulse register	0x3FF4006C	0x3FF5006C	0x3FF6E06C	RO
UART_RXD_CNT_REG	Autobaud edge change count register	0x3FF40030	0x3FF50030	0x3FF6E030	RO
AT escape seqence detection configuration					
UART_AT_CMD_PRECNT_REG	Pre-sequence timing configuration	0x3FF40048	0x3FF50048	0x3FF6E048	R/W
UART_AT_CMD_POSTCNT_REG	Post-sequence timing configuration	0x3FF4004C	0x3FF5004C	0x3FF6E04C	R/W
UART_AT_CMD_GAPTOUT_REG	Timeout configuration	0x3FF40050	0x3FF50050	0x3FF6E050	R/W
UART_AT_CMD_CHAR_REG	AT escape sequence detection configuration	0x3FF40054	0x3FF50054	0x3FF6E054	R/W
FIFO configuration					
UART_FIFO_REG	FIFO data register	0x3FF40000	0x3FF50000	0x3FF6E000	RO
UART_MEM_CONF_REG	UART threshold and allocation configuration	0x3FF40058	0x3FF50058	0x3FF6E058	R/W
UART_MEM_CNT_STATUS_REG	Receive and transmit memory configuration	0x3FF40064	0x3FF50064	0x3FF6E064	RO
Interrupt registers					

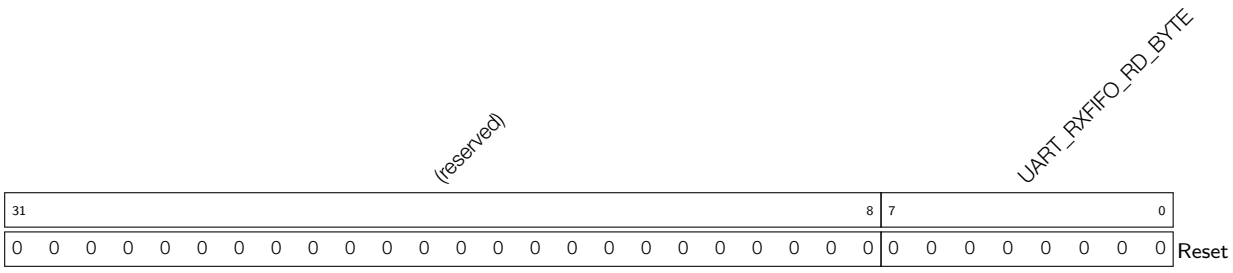
UART_INT_RAW_REG	Raw interrupt status	0x3FF40004	0x3FF50004	0x3FF6E004	RO
UART_INT_ST_REG	Masked interrupt status	0x3FF40008	0x3FF50008	0x3FF6E008	RO
UART_INT_ENA_REG	Interrupt enable bits	0x3FF4000C	0x3FF5000C	0x3FF6E00C	R/W
UART_INT_CLR_REG	Interrupt clear bits	0x3FF40010	0x3FF50010	0x3FF6E010	WO

Name	Description	UDMA0	UDMA1	Acc
Configuration registers				
UHCI_CONF0_REG	UART and frame separation config	0x3FF54000	0x3FF4C000	R/W
UHCI_CONF1_REG	UHCI config register	0x3FF5402C	0x3FF4C02C	R/W
UHCI_ESCAPE_CONF_REG	Escape characters configuration	0x3FF54064	0x3FF4C064	R/W
UHCI_HUNG_CONF_REG	Timeout configuration	0x3FF54068	0x3FF4C068	R/W
UHCI_ESC_CONF0_REG	Escape sequence configuration register 0	0x3FF540B0	0x3FF4C0B0	R/W
UHCI_ESC_CONF1_REG	Escape sequence configuration register 1	0x3FF540B4	0x3FF4C0B4	R/W
UHCI_ESC_CONF2_REG	Escape sequence configuration register 2	0x3FF540B8	0x3FF4C0B8	R/W
UHCI_ESC_CONF3_REG	Escape sequence configuration register 3	0x3FF540BC	0x3FF4C0BC	R/W
DMA configuration				
UHCI_DMA_OUT_LINK_REG	Link descriptor address and control	0x3FF54024	0x3FF4C024	R/W
UHCI_DMA_IN_LINK_REG	Link descriptor address and control	0x3FF54028	0x3FF4C028	R/W
UHCI_DMA_OUT_PUSH_REG	FIFO data push register	0x3FF54018	0x3FF4C018	R/W
UHCI_DMA_IN_POP_REG	FIFO data pop register	0x3FF54020	0x3FF4C020	RO
DMA status				
UHCI_DMA_OUT_STATUS_REG	DMA FIFO status	0x3FF54014	0x3FF4C014	RO
UHCI_DMA_OUT_EOF_DES_ADDR_REG	Out EOF link descriptor address on success	0x3FF54038	0x3FF4C038	RO
UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG	Out EOF link descriptor address on error	0x3FF54044	0x3FF4C044	RO
UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG	In EOF link descriptor address on success	0x3FF5403C	0x3FF4C03C	RO
UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG	In EOF link descriptor address on error	0x3FF54040	0x3FF4C040	RO
UHCI_DMA_IN_DSCR_REG	Current inlink descriptor, first word	0x3FF5404C	0x3FF4C04C	RO
UHCI_DMA_IN_DSCR_BF0_REG	Current inlink descriptor, second word	0x3FF54050	0x3FF4C050	RO
UHCI_DMA_IN_DSCR_BF1_REG	Current inlink descriptor, third word	0x3FF54054	0x3FF4C054	RO

UHCI_DMA_OUT_DSCR_REG	Current outlink descriptor, first word	0x3FF54058	0x3FF4C058	RO
UHCI_DMA_OUT_DSCR_BF0_REG	Current outlink descriptor, second word	0x3FF5405C	0x3FF4C05C	RO
UHCI_DMA_OUT_DSCR_BF1_REG	Current outlink descriptor, third word	0x3FF54060	0x3FF4C060	RO
Interrupt registers				
UHCI_INT_RAW_REG	Raw interrupt status	0x3FF54004	0x3FF4C004	RO
UHCI_INT_ST_REG	Masked interrupt status	0x3FF54008	0x3FF4C008	RO
UHCI_INT_ENA_REG	Interrupt enable bits	0x3FF5400C	0x3FF4C00C	R/W
UHCI_INT_CLR_REG	Interrupt clear bits	0x3FF54010	0x3FF4C010	WO

13.5 Registers

Register 13.1: UART_FIFO_REG (0x0)



UART_RXFIFO_RD_BYTE This register stores one byte of data, as read from the Rx_FIFO. (RO)

Register 13.2: UART_INT_RAW_REG (0x4)

Bit(s)	Field	Reset
31:19	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0
18	UART_AT_CMD_CHAR_DET_INT_RAW	0
17	UART_RS485_CLASH_INT_RAW	0
16	UART_RS485_FRM_ERR_INT_RAW	0
15	UART_RS485_PARITY_ERR_INT_RAW	0
14	UART_TX_DONE_INT_RAW	0
13	UART_TX_BRK_IDLE_DONE_INT_RAW	0
12	UART_TX_BRK_DONE_INT_RAW	0
11	UART_GLITCH_DET_INT_RAW	0
10	UART_SW_XOFF_INT_RAW	0
9	UART_SW_XON_INT_RAW	0
8	UART_RXFIFO_TOUT_INT_RAW	0
7	UART_BRK_DET_INT_RAW	0
6	UART_CTS_CHG_INT_RAW	0
5	UART_DSR_CHG_INT_RAW	0
4	UART_RXFIFO_OVF_INT_RAW	0
3	UART_FRM_ERR_INT_RAW	0
2	UART_PARITY_ERR_INT_RAW	0
1	UART_TXFIFO_EMPTY_INT_RAW	0
0	UART_RXFIFO_FULL_INT_RAW	0

UART_AT_CMD_CHAR_DET_INT_RAW The raw interrupt status bit for the UART_AT_CMD_CHAR_DET_INT interrupt. (RO)

UART_RS485_CLASH_INT_RAW The raw interrupt status bit for the UART_RS485_CLASH_INT interrupt. (RO)

UART_RS485_FRM_ERR_INT_RAW The raw interrupt status bit for the UART_RS485_FRM_ERR_INT interrupt. (RO)

UART_RS485_PARITY_ERR_INT_RAW The raw interrupt status bit for the UART_RS485_PARITY_ERR_INT interrupt. (RO)

UART_TX_DONE_INT_RAW The raw interrupt status bit for the UART_TX_DONE_INT interrupt. (RO)

UART_TX_BRK_IDLE_DONE_INT_RAW The raw interrupt status bit for the UART_TX_BRK_IDLE_DONE_INT interrupt. (RO)

UART_TX_BRK_DONE_INT_RAW The raw interrupt status bit for the UART_TX_BRK_DONE_INT interrupt. (RO)

UART_GLITCH_DET_INT_RAW The raw interrupt status bit for the UART_GLITCH_DET_INT interrupt. (RO)

UART_SW_XOFF_INT_RAW The raw interrupt status bit for the UART_SW_XOFF_INT interrupt. (RO)

UART_SW_XON_INT_RAW The raw interrupt status bit for the UART_SW_XON_INT interrupt. (RO)

UART_RXFIFO_TOUT_INT_RAW The raw interrupt status bit for the UART_RXFIFO_TOUT_INT interrupt. (RO)

UART_BRK_DET_INT_RAW The raw interrupt status bit for the UART_BRK_DET_INT interrupt. (RO)

UART_CTS_CHG_INT_RAW The raw interrupt status bit for the UART_CTS_CHG_INT interrupt. (RO)

UART_DSR_CHG_INT_RAW The raw interrupt status bit for the UART_DSR_CHG_INT interrupt. (RO)

UART_RXFIFO_OVF_INT_RAW The raw interrupt status bit for the UART_RXFIFO_OVF_INT interrupt. (RO)

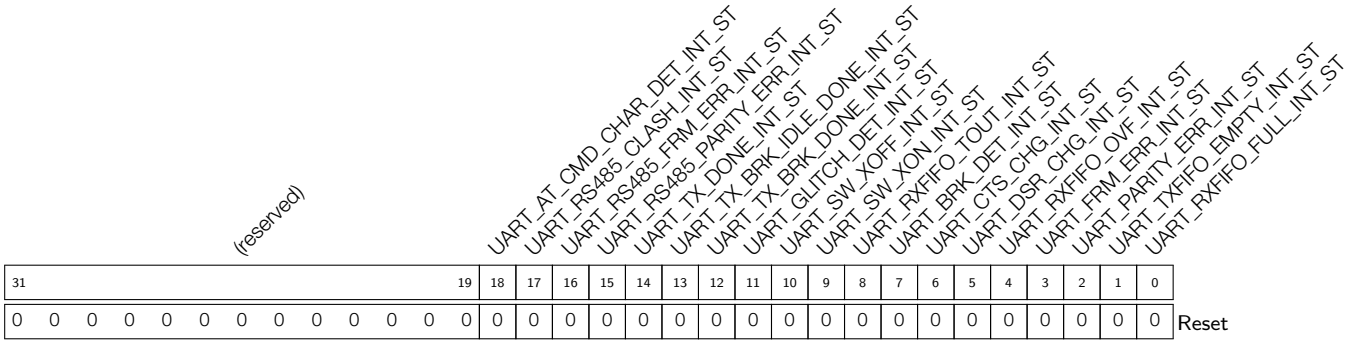
UART_FRM_ERR_INT_RAW The raw interrupt status bit for the UART_FRM_ERR_INT interrupt. (RO)

UART_PARITY_ERR_INT_RAW The raw interrupt status bit for the UART_PARITY_ERR_INT interrupt. (RO)

UART_TXFIFO_EMPTY_INT_RAW The raw interrupt status bit for the UART_TXFIFO_EMPTY_INT interrupt. (RO)

UART_RXFIFO_FULL_INT_RAW The raw interrupt status bit for the UART_RXFIFO_FULL_INT interrupt. (RO)

Register 13.3: UART_INT_ST_REG (0x8)



UART_AT_CMD_CHAR_DET_INT_ST The masked interrupt status bit for the UART_AT_CMD_CHAR_DET_INT interrupt. (RO)

UART_RS485_CLASH_INT_ST The masked interrupt status bit for the UART_RS485_CLASH_INT interrupt. (RO)

UART_RS485_FRM_ERR_INT_ST The masked interrupt status bit for the UART_RS485_FRM_ERR_INT interrupt. (RO)

UART_RS485_PARITY_ERR_INT_ST The masked interrupt status bit for the UART_RS485_PARITY_ERR_INT interrupt. (RO)

UART_TX_DONE_INT_ST The masked interrupt status bit for the UART_TX_DONE_INT interrupt. (RO)

UART_TX_BRK_IDLE_DONE_INT_ST The masked interrupt status bit for the UART_TX_BRK_IDLE_DONE_INT interrupt. (RO)

UART_TX_BRK_DONE_INT_ST The masked interrupt status bit for the UART_TX_BRK_DONE_INT interrupt. (RO)

UART_GLITCH_DET_INT_ST The masked interrupt status bit for the UART_GLITCH_DET_INT interrupt. (RO)

UART_SW_XOFF_INT_ST The masked interrupt status bit for the UART_SW_XOFF_INT interrupt. (RO)

UART_SW_XON_INT_ST The masked interrupt status bit for the UART_SW_XON_INT interrupt. (RO)

UART_RXFIFO_TOUT_INT_ST The masked interrupt status bit for the UART_RXFIFO_TOUT_INT interrupt. (RO)

UART_BRK_DET_INT_ST The masked interrupt status bit for the UART_BRK_DET_INT interrupt. (RO)

UART_CTS_CHG_INT_ST The masked interrupt status bit for the UART_CTS_CHG_INT interrupt. (RO)

UART_DSR_CHG_INT_ST The masked interrupt status bit for the UART_DSR_CHG_INT interrupt. (RO)

UART_RXFIFO_OVF_INT_ST The masked interrupt status bit for the UART_RXFIFO_OVF_INT interrupt. (RO)

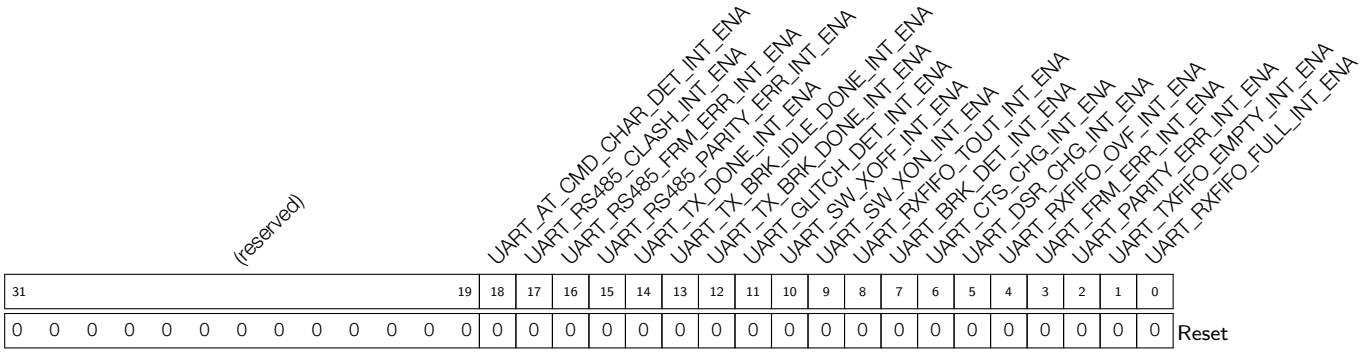
UART_FRM_ERR_INT_ST The masked interrupt status bit for the UART_FRM_ERR_INT interrupt. (RO)

UART_PARITY_ERR_INT_ST The masked interrupt status bit for the UART_PARITY_ERR_INT interrupt. (RO)

UART_TXFIFO_EMPTY_INT_ST The masked interrupt status bit for the UART_TXFIFO_EMPTY_INT interrupt. (RO)

UART_RXFIFO_FULL_INT_ST The masked interrupt status bit for UART_RXFIFO_FULL_INT. (RO)

Register 13.4: UART_INT_ENA_REG (0xC)



UART_AT_CMD_CHAR_DET_INT_ENA The interrupt enable bit for the UART_AT_CMD_CHAR_DET_INT interrupt. (R/W)

UART_RS485_CLASH_INT_ENA The interrupt enable bit for the UART_RS485_CLASH_INT interrupt. (R/W)

UART_RS485_FRM_ERR_INT_ENA The interrupt enable bit for the UART_RS485_FRM_ERR_INT interrupt. (R/W)

UART_RS485_PARITY_ERR_INT_ENA The interrupt enable bit for the UART_RS485_PARITY_ERR_INT interrupt. (R/W)

UART_TX_DONE_INT_ENA The interrupt enable bit for the UART_TX_DONE_INT interrupt. (R/W)

UART_TX_BRK_IDLE_DONE_INT_ENA The interrupt enable bit for the UART_TX_BRK_IDLE_DONE_INT interrupt. (R/W)

UART_TX_BRK_DONE_INT_ENA The interrupt enable bit for the UART_TX_BRK_DONE_INT interrupt. (R/W)

UART_GLITCH_DET_INT_ENA The interrupt enable bit for the UART_GLITCH_DET_INT interrupt. (R/W)

UART_SW_XOFF_INT_ENA The interrupt enable bit for the UART_SW_XOFF_INT interrupt. (R/W)

UART_SW_XON_INT_ENA The interrupt enable bit for the UART_SW_XON_INT interrupt. (R/W)

UART_RXFIFO_TOUT_INT_ENA The interrupt enable bit for the UART_RXFIFO_TOUT_INT interrupt. (R/W)

UART_BRK_DET_INT_ENA The interrupt enable bit for the UART_BRK_DET_INT interrupt. (R/W)

UART_CTS_CHG_INT_ENA The interrupt enable bit for the UART_CTS_CHG_INT interrupt. (R/W)

UART_DSR_CHG_INT_ENA The interrupt enable bit for the UART_DSR_CHG_INT interrupt. (R/W)

UART_RXFIFO_OVF_INT_ENA The interrupt enable bit for the UART_RXFIFO_OVF_INT interrupt. (R/W)

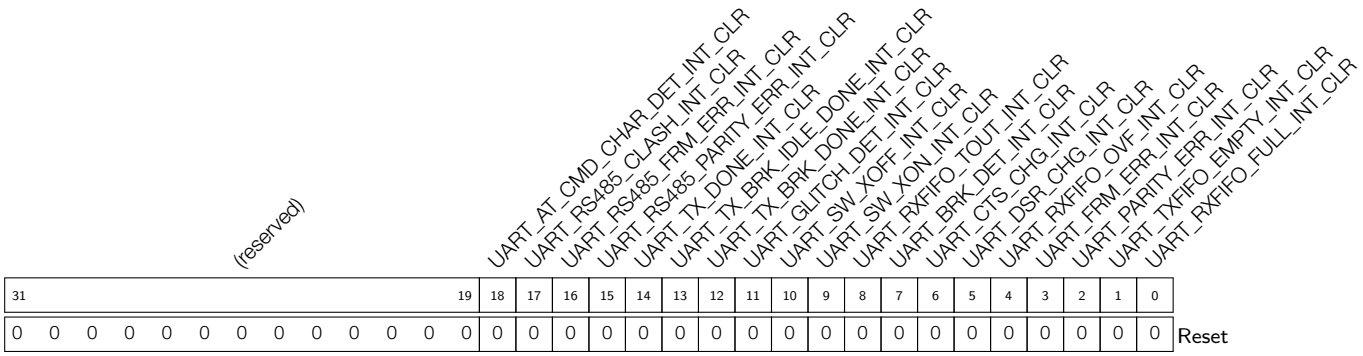
UART_FRM_ERR_INT_ENA The interrupt enable bit for the UART_FRM_ERR_INT interrupt. (R/W)

UART_PARITY_ERR_INT_ENA The interrupt enable bit for the UART_PARITY_ERR_INT interrupt. (R/W)

UART_TXFIFO_EMPTY_INT_ENA The interrupt enable bit for the UART_TXFIFO_EMPTY_INT interrupt. (R/W)

UART_RXFIFO_FULL_INT_ENA The interrupt enable bit for the UART_RXFIFO_FULL_INT interrupt. (R/W)

Register 13.5: UART_INT_CLR_REG (0x10)



UART_AT_CMD_CHAR_DET_INT_CLR Set this bit to clear the UART_AT_CMD_CHAR_DET_INT interrupt. (WO)

UART_RS485_CLASH_INT_CLR Set this bit to clear the UART_RS485_CLASH_INT interrupt. (WO)

UART_RS485_FRM_ERR_INT_CLR Set this bit to clear the UART_RS485_FRM_ERR_INT interrupt. (WO)

UART_RS485_PARITY_ERR_INT_CLR Set this bit to clear the UART_RS485_PARITY_ERR_INT interrupt. (WO)

UART_TX_DONE_INT_CLR Set this bit to clear the UART_TX_DONE_INT interrupt. (WO)

UART_TX_BRK_IDLE_DONE_INT_CLR Set this bit to clear the UART_TX_BRK_IDLE_DONE_INT interrupt. (WO)

UART_TX_BRK_DONE_INT_CLR Set this bit to clear the UART_TX_BRK_DONE_INT interrupt. (WO)

UART_GLITCH_DET_INT_CLR Set this bit to clear the UART_GLITCH_DET_INT interrupt. (WO)

UART_SW_XOFF_INT_CLR Set this bit to clear the UART_SW_XOFF_INT interrupt. (WO)

UART_SW_XON_INT_CLR Set this bit to clear the UART_SW_XON_INT interrupt. (WO)

UART_RXFIFO_TOUT_INT_CLR Set this bit to clear the UART_RXFIFO_TOUT_INT interrupt. This bit can be set only when both rxfifo_cnt and rx_mem_cnt are 0. (WO)

UART_BRK_DET_INT_CLR Set this bit to clear the UART_BRK_DET_INT interrupt. (WO)

UART_CTS_CHG_INT_CLR Set this bit to clear the UART_CTS_CHG_INT interrupt. (WO)

UART_DSR_CHG_INT_CLR Set this bit to clear the UART_DSR_CHG_INT interrupt. (WO)

UART_RXFIFO_OVF_INT_CLR Set this bit to clear the UART_RXFIFO_OVF_INT interrupt. (WO)

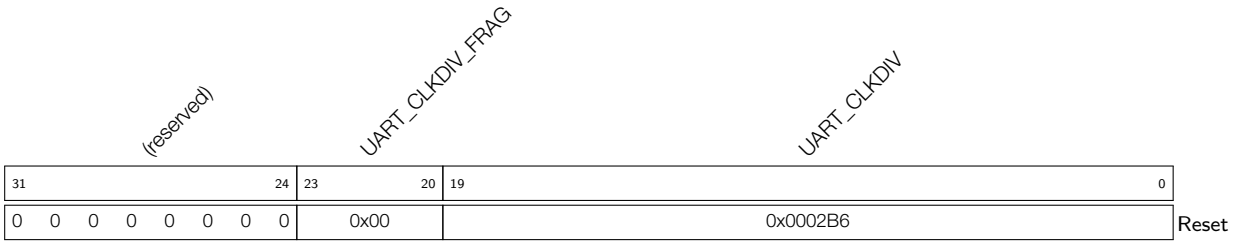
UART_FRM_ERR_INT_CLR Set this bit to clear the UART_FRM_ERR_INT interrupt. (WO)

UART_PARITY_ERR_INT_CLR Set this bit to clear the UART_PARITY_ERR_INT interrupt. (WO)

UART_TXFIFO_EMPTY_INT_CLR Set this bit to clear the UART_TXFIFO_EMPTY_INT interrupt. (WO)

UART_RXFIFO_FULL_INT_CLR Set this bit to clear the UART_RXFIFO_FULL_INT interrupt. This bit can be set only when data in Rx_FIFO is less than UART_RXFIFO_FULL_THRHD. (WO)

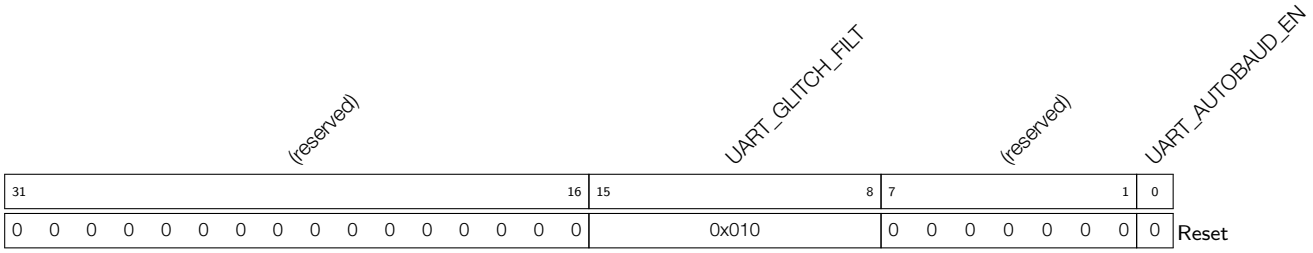
Register 13.6: UART_CLKDIV_REG (0x14)



UART_CLKDIV_FRAG The decimal part of the frequency divider factor. (R/W)

UART_CLKDIV The integral part of the frequency divider factor. (R/W)

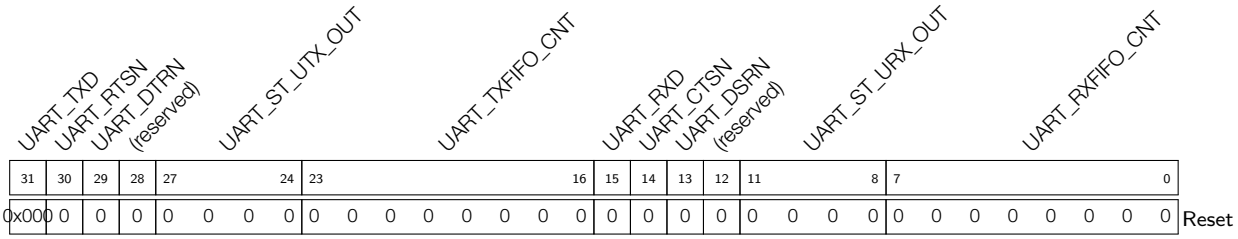
Register 13.7: UART_AUTOBAUD_REG (0x18)



UART_GLITCH_FILT When the input pulse width is lower than this value, the pulse is ignored. This register is used in the autobauding process. (R/W)

UART_AUTOBAUD_EN This is the enable bit for autobaud. (R/W)

Register 13.8: UART_STATUS_REG (0x1C)



UART_TXD This bit represents the level of the internal UART RxD signal. (RO)

UART_RTSN This bit corresponds to the level of the internal UART CTS signal. (RO)

UART_DTRN This bit corresponds to the level of the internal UAR DSR signal. (RO)

UART_ST_UTX_OUT This register stores the state of the transmitter's finite state machine. 0: TX_IDLE; 1: TX_STRT; 2: TX_DAT0; 3: TX_DAT1; 4: TX_DAT2; 5: TX_DAT3; 6: TX_DAT4; 7: TX_DAT5; 8: TX_DAT6; 9: TX_DAT7; 10: TX_PRTY; 11: TX_STP1; 12: TX_STP2; 13: TX_DL0; 14: TX_DL1. (RO)

UART_TXFIFO_CNT (tx_mem_cnt, txfifo_cnt) stores the number of bytes of valid data in transmit-FIFO. tx_mem_cnt stores the three most significant bits, txfifo_cnt stores the eight least significant bits. (RO)

UART_RXD This bit corresponds to the level of the internal UART RxD signal. (RO)

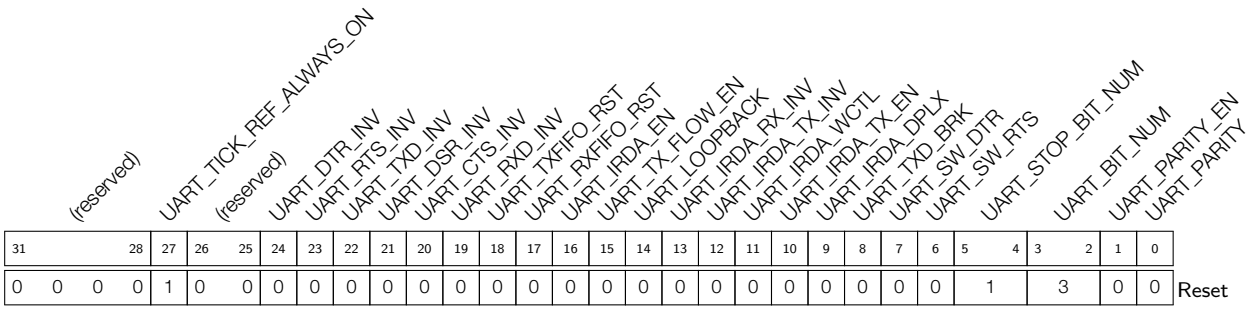
UART_CTSN This bit corresponds to the level of the internal UART CTS signal. (RO)

UART_DSRN This bit corresponds to the level of the internal UAR DSR signal. (RO)

UART_ST_URX_OUT This register stores the value of the receiver's finite state machine. 0: RX_IDLE; 1: RX_STRT; 2: RX_DAT0; 3: RX_DAT1; 4: RX_DAT2; 5: RX_DAT3; 6: RX_DAT4; 7: RX_DAT5; 8: RX_DAT6; 9: RX_DAT7; 10: RX_PRTY; 11: RX_STP1; 12:RX_STP2; 13: RX_DL1. (RO)

UART_RXFIFO_CNT (rx_mem_cnt, rxfifo_cnt) stores the number of bytes of valid data in the receive-FIFO. rx_mem_cnt register stores the three most significant bits, rxfifo_cnt stores the eight least significant bits. (RO)

Register 13.9: UART_CONF0_REG (0x20)



UART_TICK_REF_ALWAYS_ON This register is used to select the clock; 1: APB clock; 0: REF_TICK. (R/W)

UART_DTR_INV Set this bit to invert the level of the UART DTR signal. (R/W)

UART_RTS_INV Set this bit to invert the level of the UART RTS signal. (R/W)

UART_TXD_INV Set this bit to invert the level of the UART TxD signal. (R/W)

UART_DSR_INV Set this bit to invert the level of the UART DSR signal. (R/W)

UART_CTS_INV Set this bit to invert the level of the UART CTS signal. (R/W)

UART_RXD_INV Set this bit to invert the level of the UART Rxd signal. (R/W)

UART_TXFIFO_RST Set this bit to reset the UART transmit-FIFO. **NOTICE**: UART2 doesn't have any register to reset Tx_FIFO or Rx_FIFO, and the UART1_TXFIFO_RST and UART1_RXFIFO_RST in UART1 may impact the functioning of UART2. Therefore, these two registers in UART1 should only be used when the Tx_FIFO and Rx_FIFO in UART2 do not have any data. (R/W)

UART_RXFIFO_RST Set this bit to reset the UART receive-FIFO. **NOTICE**: UART2 doesn't have any register to reset Tx_FIFO or Rx_FIFO, and the UART1_TXFIFO_RST and UART1_RXFIFO_RST in UART1 may impact the functioning of UART2. Therefore, these two registers in UART1 should only be used when the Tx_FIFO and Rx_FIFO in UART2 do not have any data. (R/W)

UART_IRDA_EN Set this bit to enable the IrDA protocol. (R/W)

UART_TX_FLOW_EN Set this bit to enable the flow control function for the transmitter. (R/W)

UART_LOOPBACK Set this bit to enable the UART loopback test mode. (R/W)

UART_IRDA_RX_INV Set this bit to invert the level of the IrDA receiver. (R/W)

UART_IRDA_TX_INV Set this bit to invert the level of the IrDA transmitter. (R/W)

UART_IRDA_WCTL 1: The IrDA transmitter's 11th bit is the same as its 10th bit; 0: set IrDA transmitter's 11th bit to 0. (R/W)

UART_IRDA_TX_EN This is the start enable bit of the IrDA transmitter. (R/W)

UART_IRDA_DPLX Set this bit to enable the IrDA loopback mode. (R/W)

UART_TXD_BRK Set this bit to enable the transmitter to send NULL, when the process of sending data is completed. (R/W)

UART_SW_DTR This register is used to configure the software DTR signal used in software flow control. (R/W)

UART_SW_RTS This register is used to configure the software RTS signal used in software flow control. (R/W)

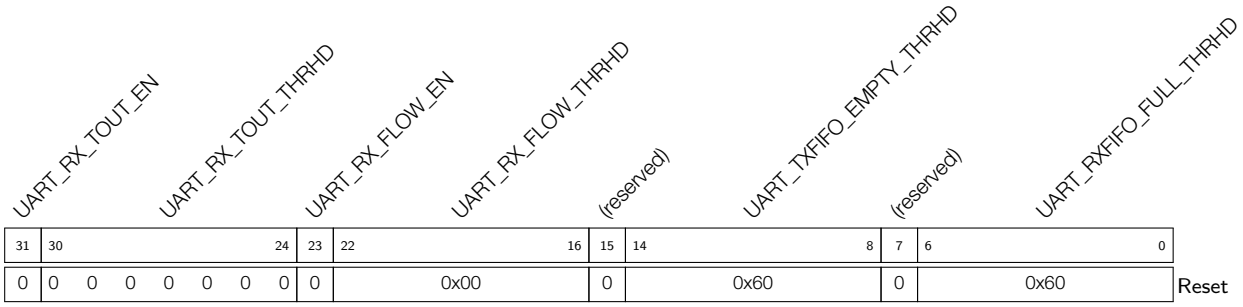
UART_STOP_BIT_NUM This register is used to set the length of the stop bit; 1: 1 bit, 2: 1.5 bits. (R/W)

UART_BIT_NUM This register is used to set the length of data; 0: 5 bits, 1: 6 bits, 2: 7 bits, 3: 8 bits. (R/W)

UART_PARITY_EN Set this bit to enable the UART parity check. (R/W)

UART_PARITY This register is used to configure the parity check mode; 0: even, 1: odd. (R/W)

Register 13.10: UART_CONF1_REG (0x24)



UART_RX_TOUT_EN This is the enable bit for the UART receive-timeout function. (R/W)

UART_RX_TOUT_THRHD This register is used to configure the UART receiver's timeout value when receiving a byte. When using APB_CLK as the clock source, the register counts by UART baud cycle. When using REF_TICK as the clock source, the register counts by $\frac{\text{APB\_CLK frequency}}{\text{(REF\_TICK frequency)*8}}$. (R/W)

UART_RX_FLOW_EN This is the flow enable bit of the UART receiver; 1: choose software flow control by configuring the sw_rts signal; 0: disable software flow control. (R/W)

UART_RX_FLOW_THRHD When the receiver gets more data than its threshold value, the receiver produces a signal that tells the transmitter to stop transferring data. The threshold value is (rx_flow_thrhd_h3, rx_flow_thrhd). (R/W)

UART_TXFIFO_EMPTY_THRHD When the data amount in transmit-FIFO is less than its threshold value, it will produce a TXFIFO_EMPTY_INT_RAW interrupt. The threshold value is (tx_mem_empty_thrhd, txfifo_empty_thrhd). (R/W)

UART_RXFIFO_FULL_THRHD When the receiver gets more data than its threshold value, the receiver will produce an RXFIFO_FULL_INT_RAW interrupt. The threshold value is (rx_flow_thrhd_h3, rxfifo_full_thrhd). (R/W)

Register 13.11: UART_LOWPULSE_REG (0x28)



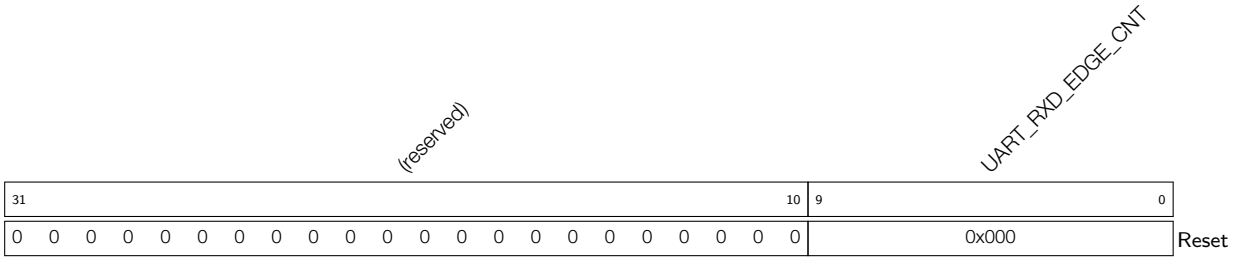
UART_LOWPULSE_MIN_CNT This register stores the value of the minimum duration of the low-level pulse. It is used in the baud rate detection process. (RO)

Register 13.12: UART_HIGHPULSE_REG (0x2C)



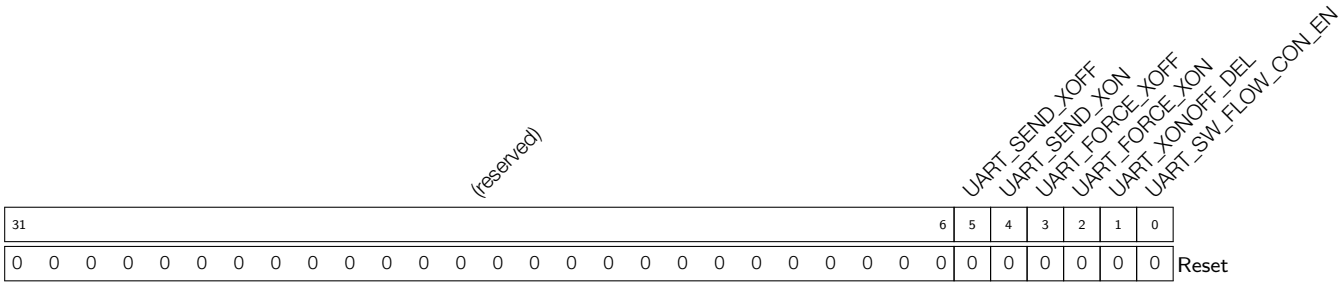
UART_HIGHPULSE_MIN_CNT This register stores the value of the minimum duration of the high level pulse. It is used in baud rate detection process. (RO)

Register 13.13: UART_RXD_CNT_REG (0x30)



UART_RXD_EDGE_CNT This register stores the count of the RxD edge change. It is used in the baud rate detection process. (RO)

Register 13.14: UART_FLOW_CONF_REG (0x34)



UART_SEND_XOFF Hardware auto-clear; set to 1 to send Xoff char. (R/W)

UART_SEND_XON Hardware auto-clear; set to 1 to send Xon char. (R/W)

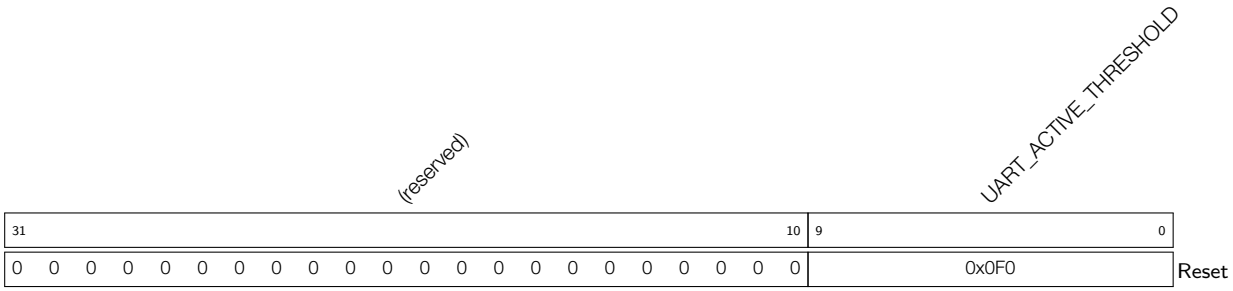
UART_FORCE_XOFF Set this bit to set the CTSn and enable the transmitter to continue sending data. (R/W)

UART_FORCE_XON Set this bit to clear the CTSn and stop the transmitter from sending data. (R/W)

UART_XONOFF_DEL Set this bit to remove the flow-control char from the received data. (R/W)

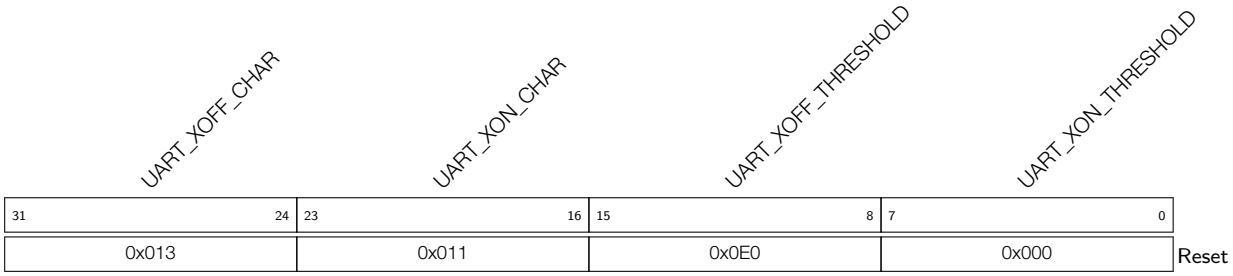
UART_SW_FLOW_CON_EN Set this bit to enable software flow control. It is used with register sw_xon or sw_xoff. (R/W)

Register 13.15: UART_SLEEP_CONF_REG (0x38)



UART_ACTIVE_THRESHOLD When the input RxD edge changes more times than what this register indicates, the system emerges from Light-sleep mode and becomes active. (R/W)

Register 13.16: UART_SWFC_CONF_REG (0x3C)



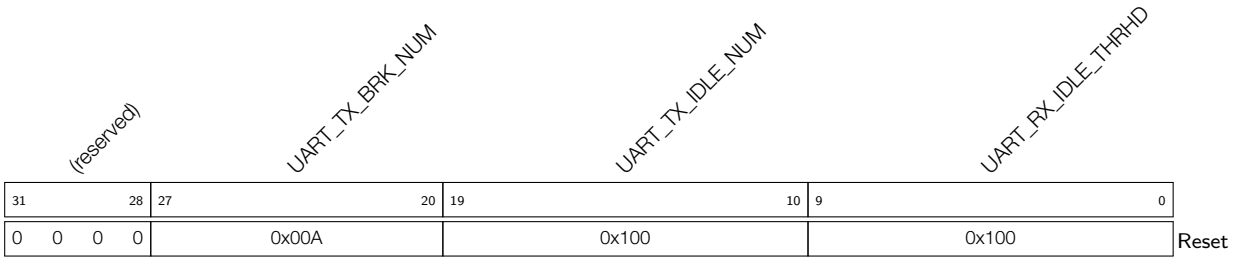
UART_XOFF_CHAR This register stores the Xoff flow control char. (R/W)

UART_XON_CHAR This register stores the Xon flow control char. (R/W)

UART_XOFF_THRESHOLD When the data amount in receive-FIFO is less than what this register indicates, it will send an Xon char, with uart_sw_flow_con_en set to 1. (R/W)

UART_XON_THRESHOLD When the data amount in receive-FIFO is more than what this register indicates, it will send an Xoff char, with uart_sw_flow_con_en set to 1. (R/W)

Register 13.17: UART_IDLE_CONF_REG (0x40)

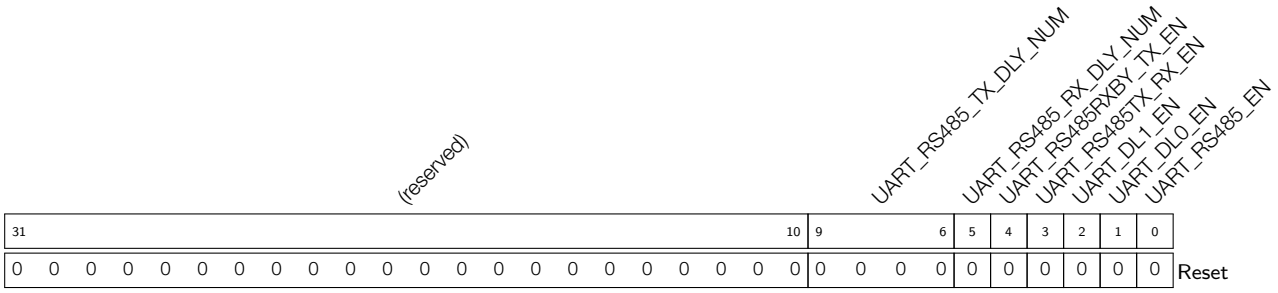


UART_TX_BRK_NUM This register is used to configure the number of zeros (0) sent, after the process of sending data is completed. It is active when txd_brk is set to 1. (R/W)

UART_TX_IDLE_NUM This register is used to configure the duration between transfers. (R/W)

UART_RX_IDLE_THRHD When the receiver takes more time to receive Byte data than what this register indicates, it will produce a frame-end signal. (R/W)

Register 13.18: UART_RS485_CONF_REG (0x44)



UART_RS485_TX_DLY_NUM This register is used to delay the transmitter's internal data signal. (R/W)

UART_RS485_RX_DLY_NUM This register is used to delay the receiver's internal data signal. (R/W)

UART_RS485RXBY_TX_EN 1: enable the RS-485 transmitter to send data, when the RS-485 receiver line is busy; 0: the RS-485 transmitter should not send data, when its receiver is busy. (R/W)

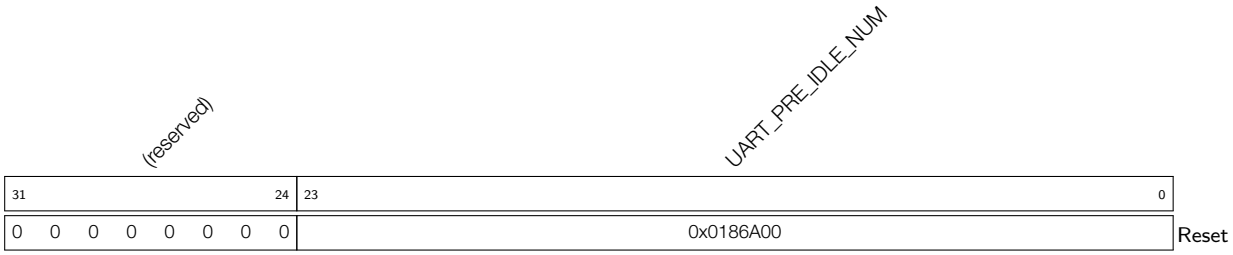
UART_RS485TX_RX_EN Set this bit to enable the transmitter's output signal loop back to the receiver's input signal. (R/W)

UART_DL1_EN Set this bit to delay the STOP bit by 1 bit. (R/W)

UART_DL0_EN Set this bit to delay the STOP bit by 1 bit. (R/W)

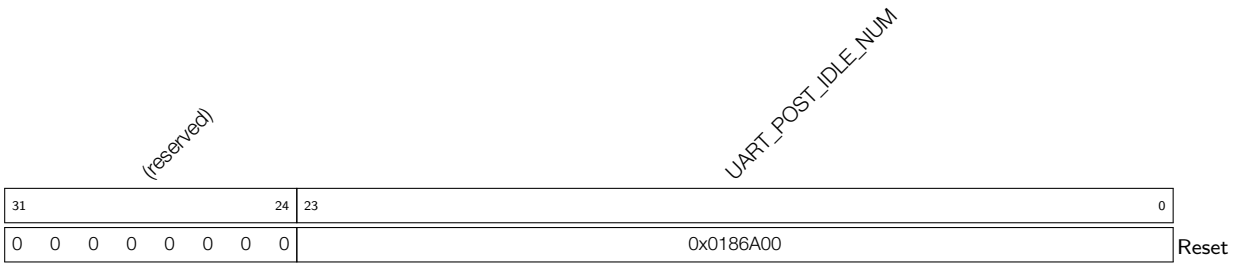
UART_RS485_EN Set this bit to choose the RS-485 mode. (R/W)

Register 13.19: UART_AT_CMD_PRECNT_REG (0x48)



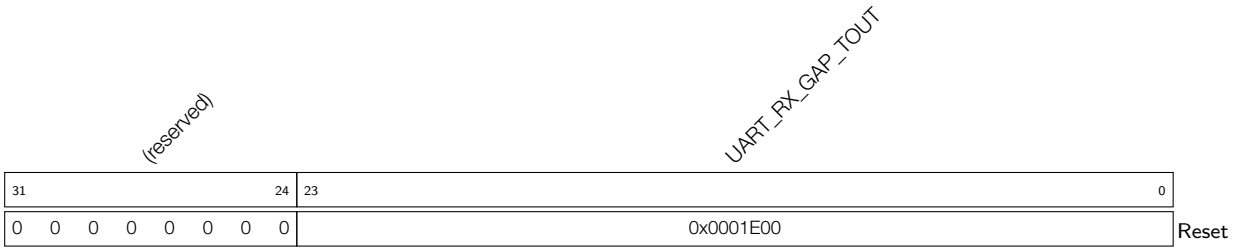
UART_PRE_IDLE_NUM This register is used to configure the idle-time duration before the first at_cmd is received by the receiver. When the duration is less than what this register indicates, it will not take the next data received as an at_cmd char. (R/W)

Register 13.20: UART_AT_CMD_POSTCNT_REG (0x4c)



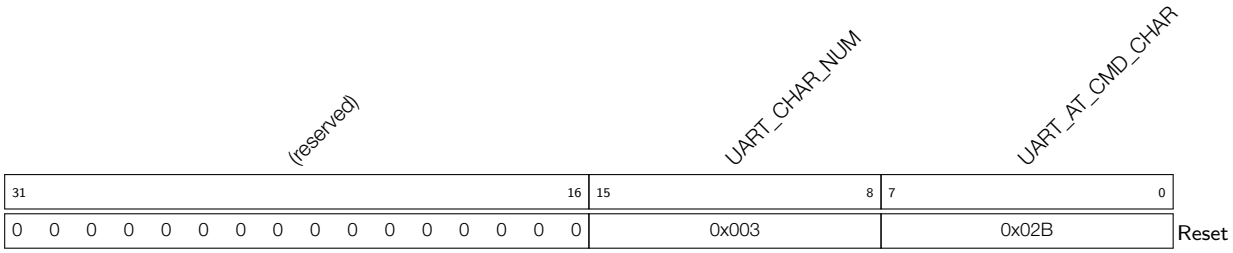
UART_POST_IDLE_NUM This register is used to configure the duration between the last at_cmd and the next data. When the duration is less than what this register indicates, it will not take the previous data as an at_cmd char. (R/W)

Register 13.21: UART_AT_CMD_GAPTOUT_REG (0x50)



UART_RX_GAP_TOUT This register is used to configure the duration between the at_cmd chars. When the duration is less than what this register indicates, it will not take the data as continuous at_cmd chars. (R/W)

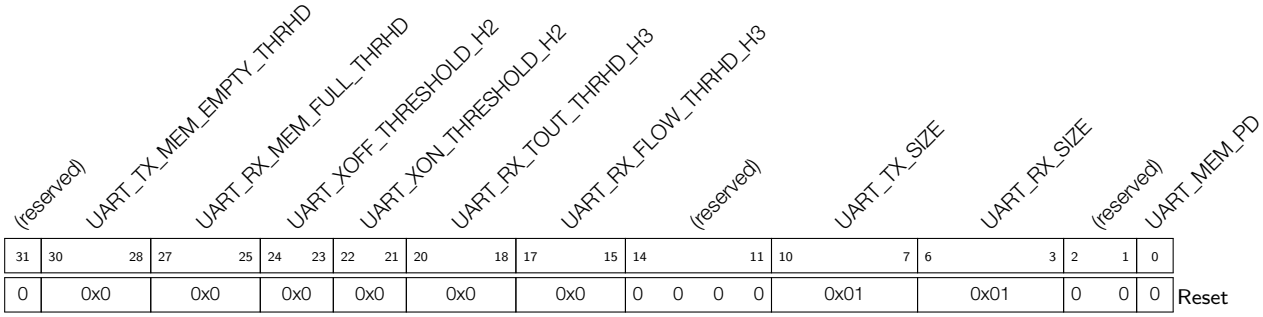
Register 13.22: UART_AT_CMD_CHAR_REG (0x54)



UART_CHAR_NUM This register is used to configure the number of continuous at_cmd chars received by the receiver. (R/W)

UART_AT_CMD_CHAR This register is used to configure the content of an at_cmd char. (R/W)

Register 13.23: UART_MEM_CONF_REG (0x58)



UART_TX_MEM_EMPTY_THRHD Refer to the description of txfifo_empty_thrhd. (R/W)

UART_RX_MEM_FULL_THRHD Refer to the description of rxfifo_full_thrhd. (R/W)

UART_XOFF_THRESHOLD_H2 Refer to the description of uart_xoff_threshold. (R/W)

UART_XON_THRESHOLD_H2 Refer to the description of uart_xon_threshold. (R/W)

UART_RX_TOUT_THRHD_H3 Refer to the description of rx_tout_thrhd. (R/W)

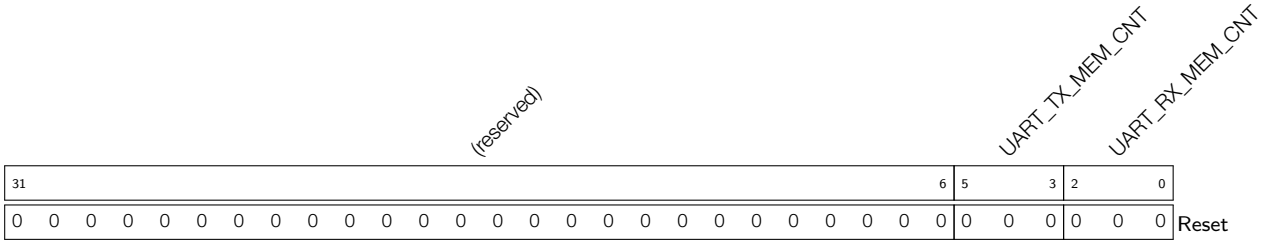
UART_RX_FLOW_THRHD_H3 Refer to the description of rx_flow_thrhd. (R/W)

UART_TX_SIZE This register is used to configure the amount of memory allocated to the transmit-FIFO. The default number is 128 bytes. (R/W)

UART_RX_SIZE This register is used to configure the amount of memory allocated to the receive-FIFO. The default number is 128 bytes. (R/W)

UART_MEM_PD Set this bit to power down the memory. When the reg_mem_pd register is set to 1 for all UART controllers, Memory will enter the low-power mode. (R/W)

Register 13.24: UART_MEM_CNT_STATUS_REG (0x64)



UART_TX_MEM_CNT Refer to the description of txfifo_cnt. (RO)

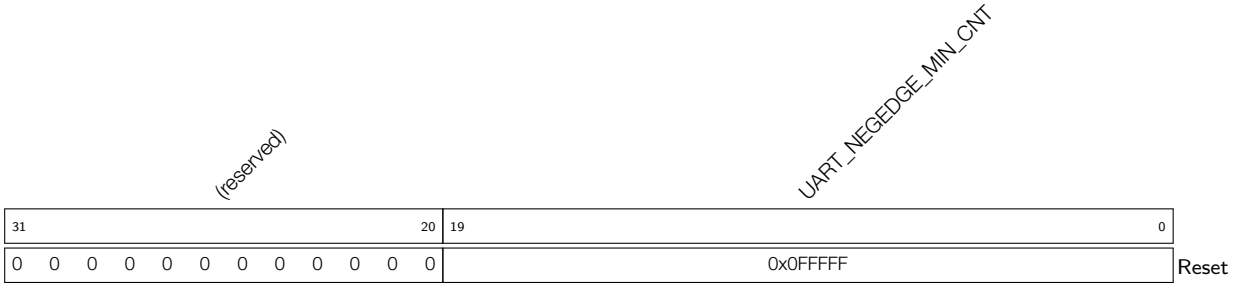
UART_RX_MEM_CNT Refer to the description of rxfifo_cnt. (RO)

Register 13.25: UART_POSPULSE_REG (0x68)



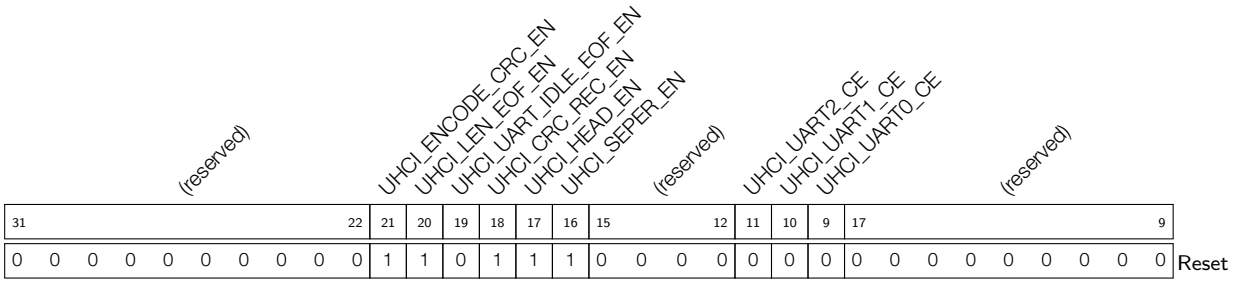
UART_POSEDGE_MIN_CNT This register stores the count of RxD positive edges. It is used in the autobaud detection process. (RO)

Register 13.26: UART_NEGPULSE_REG (0x6c)



UART_NEGEDGE_MIN_CNT This register stores the count of RxD negative edges. It is used in the autobaud detection process. (RO)

Register 13.27: UHCI_CONF0_REG (0x0)



UHCI_ENCODE_CRC_EN Reserved. Please initialize it to 0. (R/W)

UHCI_LEN_EOF_EN Reserved. Please initialize it to 0. (R/W)

UHCI_UART_IDLE_EOF_EN Reserved. Please initialize it to 0. (R/W)

UHCI_CRC_REC_EN Reserved. Please initialize it to 0. (R/W)

UHCI_HEAD_EN Reserved. Please initialize it to 0. (R/W)

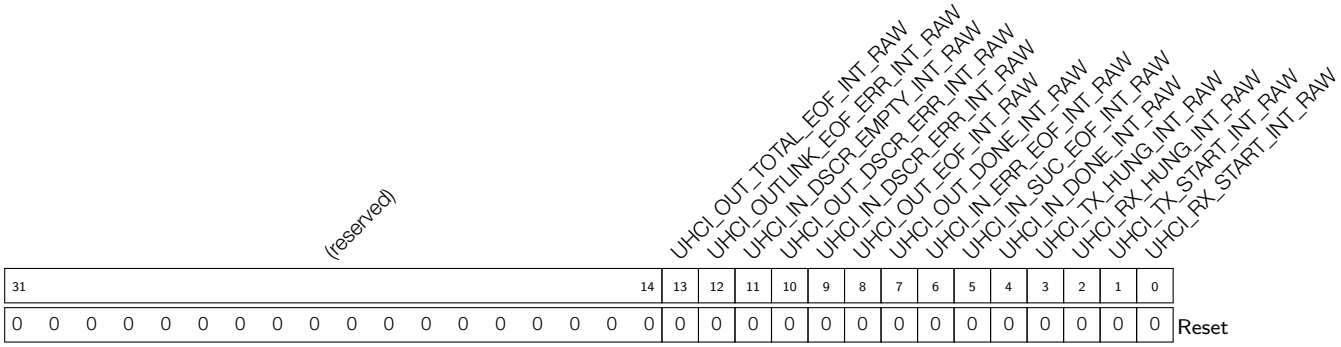
UHCI_SEPER_EN Set this bit to use a special char and separate the data frame. (R/W)

UHCI_UART2_CE Set this bit to use UART2 and transmit or receive data. (R/W)

UHCI_UART1_CE Set this bit to use UART1 and transmit or receive data. (R/W)

UHCI_UART0_CE Set this bit to use UART and transmit or receive data. (R/W)

Register 13.28: UHCI_INT_RAW_REG (0x4)



UHCI_OUT_TOTAL_EOF_INT_RAW The raw interrupt status bit for the UHCI_OUT_TOTAL_EOF_INT interrupt. (RO)

UHCI_OUTLINK_EOF_ERR_INT_RAW The raw interrupt status bit for the UHCI_OUTLINK_EOF_ERR_INT interrupt. (RO)

UHCI_IN_DSCR_EMPTY_INT_RAW The raw interrupt status bit for the UHCI_IN_DSCR_EMPTY_INT interrupt. (RO)

UHCI_OUT_DSCR_ERR_INT_RAW The raw interrupt status bit for the UHCI_OUT_DSCR_ERR_INT interrupt. (RO)

UHCI_IN_DSCR_ERR_INT_RAW The raw interrupt status bit for the UHCI_IN_DSCR_ERR_INT interrupt. (RO)

UHCI_OUT_EOF_INT_RAW The raw interrupt status bit for the UHCI_OUT_EOF_INT interrupt. (RO)

UHCI_OUT_DONE_INT_RAW The raw interrupt status bit for the UHCI_OUT_DONE_INT interrupt. (RO)

UHCI_IN_ERR_EOF_INT_RAW The raw interrupt status bit for the UHCI_IN_ERR_EOF_INT interrupt. (RO)

UHCI_IN_SUC_EOF_INT_RAW The raw interrupt status bit for the UHCI_IN_SUC_EOF_INT interrupt. (RO)

UHCI_IN_DONE_INT_RAW The raw interrupt status bit for the UHCI_IN_DONE_INT interrupt. (RO)

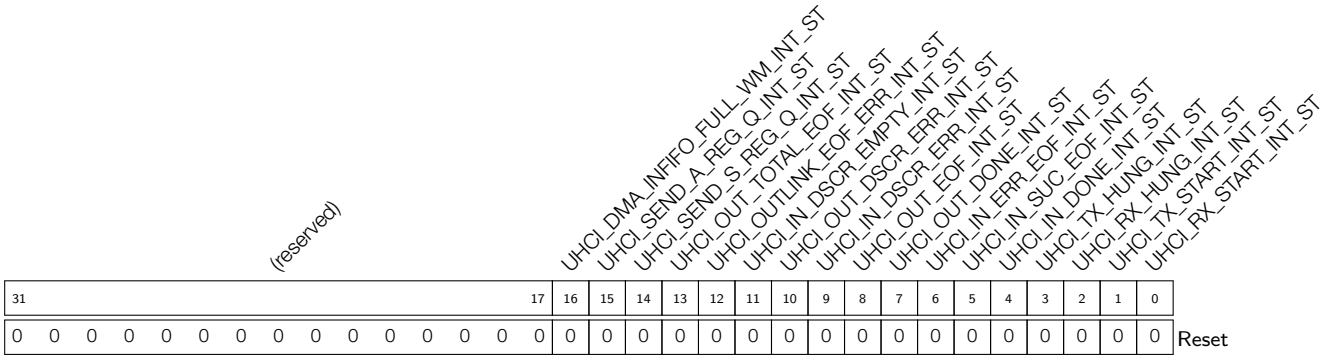
UHCI_TX_HUNG_INT_RAW The raw interrupt status bit for the UHCI_TX_HUNG_INT interrupt. (RO)

UHCI_RX_HUNG_INT_RAW The raw interrupt status bit for the UHCI_RX_HUNG_INT interrupt. (RO)

UHCI_TX_START_INT_RAW The raw interrupt status bit for the UHCI_TX_START_INT interrupt. (RO)

UHCI_RX_START_INT_RAW The raw interrupt status bit for the UHCI_RX_START_INT interrupt. (RO)

Register 13.29: UHCI_INT_ST_REG (0x8)



UHCI_SEND_A_REG_Q_INT_ST The masked interrupt status bit for the UHCI_SEND_A_REG_Q_INT interrupt. (RO)

UHCI_SEND_S_REG_Q_INT_ST The masked interrupt status bit for the UHCI_SEND_S_REG_Q_INT interrupt. (RO)

UHCI_OUT_TOTAL_EOF_INT_ST The masked interrupt status bit for the UHCI_OUT_TOTAL_EOF_INT interrupt. (RO)

UHCI_OUTLINK_EOF_ERR_INT_ST The masked interrupt status bit for the UHCI_OUTLINK_EOF_ERR_INT interrupt. (RO)

UHCI_IN_DSCR_EMPTY_INT_ST The masked interrupt status bit for the UHCI_IN_DSCR_EMPTY_INT interrupt. (RO)

UHCI_OUT_DSCR_ERR_INT_ST The masked interrupt status bit for the UHCI_OUT_DSCR_ERR_INT interrupt. (RO)

UHCI_IN_DSCR_ERR_INT_ST The masked interrupt status bit for the UHCI_IN_DSCR_ERR_INT interrupt. (RO)

UHCI_OUT_EOF_INT_ST The masked interrupt status bit for the UHCI_OUT_EOF_INT interrupt. (RO)

UHCI_OUT_DONE_INT_ST The masked interrupt status bit for the UHCI_OUT_DONE_INT interrupt. (RO)

UHCI_IN_ERR_EOF_INT_ST The masked interrupt status bit for the UHCI_IN_ERR_EOF_INT interrupt. (RO)

UHCI_IN_SUC_EOF_INT_ST The masked interrupt status bit for the UHCI_IN_SUC_EOF_INT interrupt. (RO)

UHCI_IN_DONE_INT_ST The masked interrupt status bit for the UHCI_IN_DONE_INT interrupt. (RO)

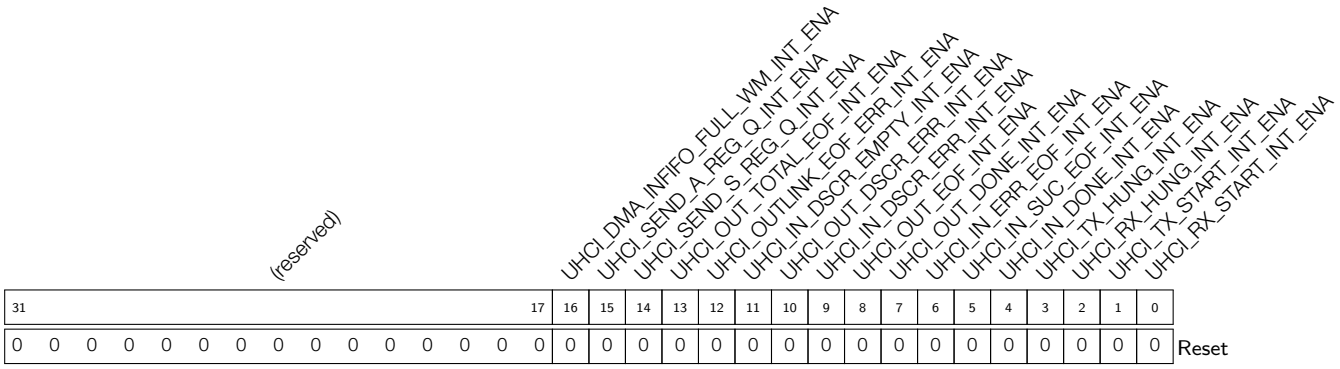
UHCI_TX_HUNG_INT_ST The masked interrupt status bit for the UHCI_TX_HUNG_INT interrupt. (RO)

UHCI_RX_HUNG_INT_ST The masked interrupt status bit for the UHCI_RX_HUNG_INT interrupt. (RO)

UHCI_TX_START_INT_ST The masked interrupt status bit for the UHCI_TX_START_INT interrupt. (RO)

UHCI_RX_START_INT_ST The masked interrupt status bit for the UHCI_RX_START_INT interrupt. (RO)

Register 13.30: UHCI_INT_ENA_REG (0xC)



UHCI_SEND_A_REG_Q_INT_ENA The interrupt enable bit for the UHCI_SEND_A_REG_Q_INT interrupt. (R/W)

UHCI_SEND_S_REG_Q_INT_ENA The interrupt enable bit for the UHCI_SEND_S_REG_Q_INT interrupt. (R/W)

UHCI_OUT_TOTAL_EOF_INT_ENA The interrupt enable bit for the UHCI_OUT_TOTAL_EOF_INT interrupt. (R/W)

UHCI_OUTLINK_EOF_ERR_INT_ENA The interrupt enable bit for the UHCI_OUTLINK_EOF_ERR_INT interrupt. (R/W)

UHCI_IN_DSCR_EMPTY_INT_ENA The interrupt enable bit for the UHCI_IN_DSCR_EMPTY_INT interrupt. (R/W)

UHCI_OUT_DSCR_ERR_INT_ENA The interrupt enable bit for the UHCI_OUT_DSCR_ERR_INT interrupt. (R/W)

UHCI_IN_DSCR_ERR_INT_ENA The interrupt enable bit for the UHCI_IN_DSCR_ERR_INT interrupt. (R/W)

UHCI_OUT_EOF_INT_ENA The interrupt enable bit for the UHCI_OUT_EOF_INT interrupt. (R/W)

UHCI_OUT_DONE_INT_ENA The interrupt enable bit for the UHCI_OUT_DONE_INT interrupt. (R/W)

UHCI_IN_ERR_EOF_INT_ENA The interrupt enable bit for the UHCI_IN_ERR_EOF_INT interrupt. (R/W)

UHCI_IN_SUC_EOF_INT_ENA The interrupt enable bit for the UHCI_IN_SUC_EOF_INT interrupt. (R/W)

UHCI_IN_DONE_INT_ENA The interrupt enable bit for the UHCI_IN_DONE_INT interrupt. (R/W)

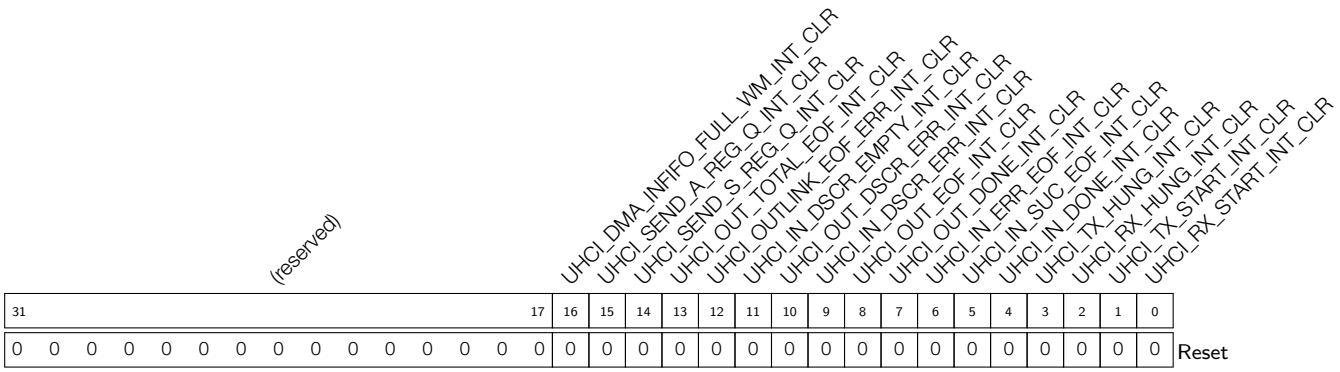
UHCI_TX_HUNG_INT_ENA The interrupt enable bit for the UHCI_TX_HUNG_INT interrupt. (R/W)

UHCI_RX_HUNG_INT_ENA The interrupt enable bit for the UHCI_RX_HUNG_INT interrupt. (R/W)

UHCI_TX_START_INT_ENA The interrupt enable bit for the UHCI_TX_START_INT interrupt. (R/W)

UHCI_RX_START_INT_ENA The interrupt enable bit for the UHCI_RX_START_INT interrupt. (R/W)

Register 13.31: UHCI_INT_CLR_REG (0x10)



UHCI_SEND_A_REG_Q_INT_CLR Set this bit to clear the UHCI_SEND_A_REG_Q_INT interrupt. (WO)

UHCI_SEND_S_REG_Q_INT_CLR Set this bit to clear the UHCI_SEND_S_REG_Q_INT interrupt. (WO)

UHCI_OUT_TOTAL_EOF_INT_CLR Set this bit to clear the UHCI_OUT_TOTAL_EOF_INT interrupt. (WO)

UHCI_OUTLINK_EOF_ERR_INT_CLR Set this bit to clear the UHCI_OUTLINK_EOF_ERR_INT interrupt. (WO)

UHCI_IN_DSCR_EMPTY_INT_CLR Set this bit to clear the UHCI_IN_DSCR_EMPTY_INT interrupt. (WO)

UHCI_OUT_DSCR_ERR_INT_CLR Set this bit to clear the UHCI_OUT_DSCR_ERR_INT interrupt. (WO)

UHCI_IN_DSCR_ERR_INT_CLR Set this bit to clear the UHCI_IN_DSCR_ERR_INT interrupt. (WO)

UHCI_OUT_EOF_INT_CLR Set this bit to clear the UHCI_OUT_EOF_INT interrupt. (WO)

UHCI_OUT_DONE_INT_CLR Set this bit to clear the UHCI_OUT_DONE_INT interrupt. (WO)

UHCI_IN_ERR_EOF_INT_CLR Set this bit to clear the UHCI_IN_ERR_EOF_INT interrupt. (WO)

UHCI_IN_SUC_EOF_INT_CLR Set this bit to clear the UHCI_IN_SUC_EOF_INT interrupt. (WO)

UHCI_IN_DONE_INT_CLR Set this bit to clear the UHCI_IN_DONE_INT interrupt. (WO)

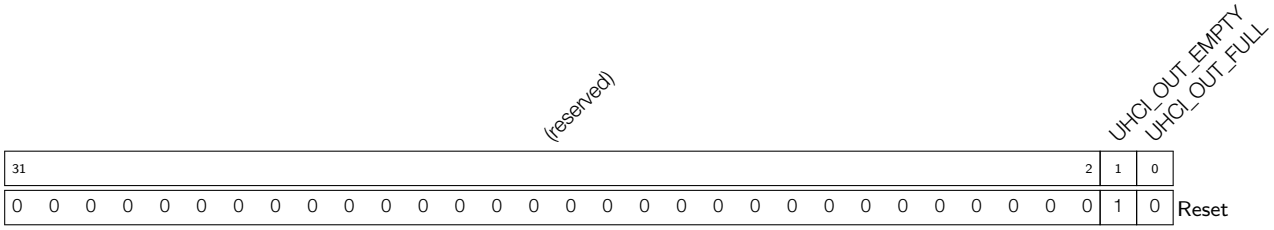
UHCI_TX_HUNG_INT_CLR Set this bit to clear the UHCI_TX_HUNG_INT interrupt. (WO)

UHCI_RX_HUNG_INT_CLR Set this bit to clear the UHCI_RX_HUNG_INT interrupt. (WO)

UHCI_TX_START_INT_CLR Set this bit to clear the UHCI_TX_START_INT interrupt. (WO)

UHCI_RX_START_INT_CLR Set this bit to clear the UHCI_RX_START_INT interrupt. (WO)

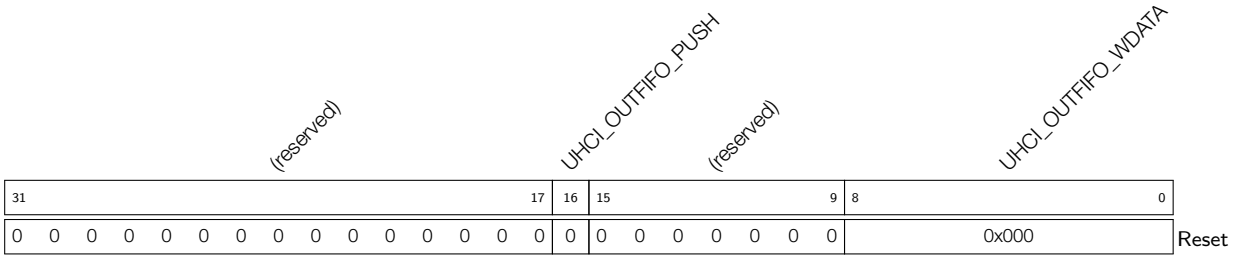
Register 13.32: UHCI_DMA_OUT_STATUS_REG (0x14)



UHCI_OUT_EMPTY 1: DMA inlink descriptor's FIFO is empty. (RO)

UHCI_OUT_FULL 1: DMA outlink descriptor's FIFO is full. (RO)

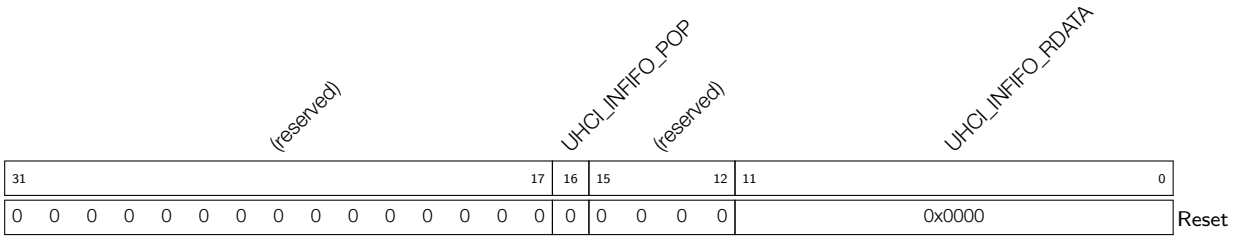
Register 13.33: UHCI_DMA_OUT_PUSH_REG (0x18)



UHCI_OUTFIFO_PUSH Set this bit to push data into DMA FIFO. (R/W)

UHCI_OUTFIFO_WDATA This is the data that need to be pushed into DMA FIFO. (R/W)

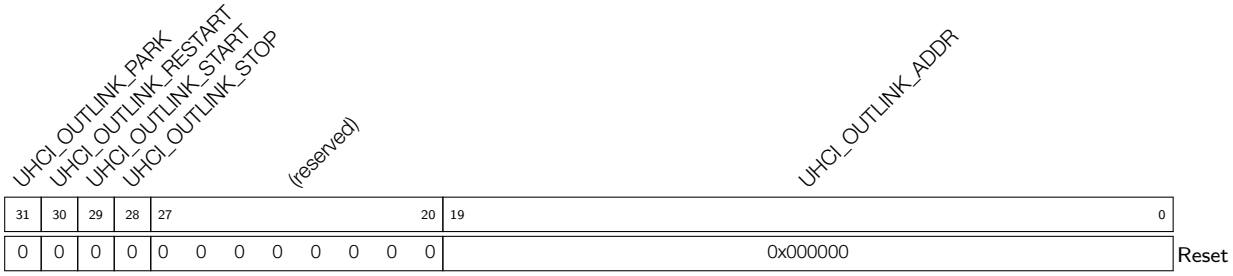
Register 13.34: UHCI_DMA_IN_POP_REG (0x20)



UHCI_INFIFO_POP Set this bit to pop data from DMA FIFO. (R/W)

UHCI_INFIFO_RDATA This register stores the data popping from DMA FIFO. (RO)

Register 13.35: UHCI_DMA_OUT_LINK_REG (0x24)



UHCI_OUTLINK_PARK 1: the outlink descriptor's FSM is in idle state; 0: the outlink descriptor's FSM is working. (RO)

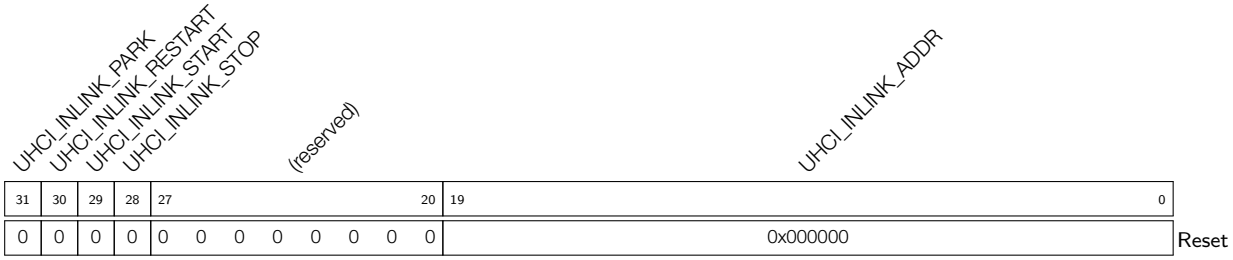
UHCI_OUTLINK_RESTART Set this bit to restart the outlink descriptor from the last address. (R/W)

UHCI_OUTLINK_START Set this bit to start a new outlink descriptor. (R/W)

UHCI_OUTLINK_STOP Set this bit to stop dealing with the outlink descriptor. (R/W)

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

Register 13.36: UHCI_DMA_IN_LINK_REG (0x28)



UHCI_INLINK_PARK 1: the inlink descriptor's FSM is in idle state; 0: the inlink descriptor's FSM is working. (RO)

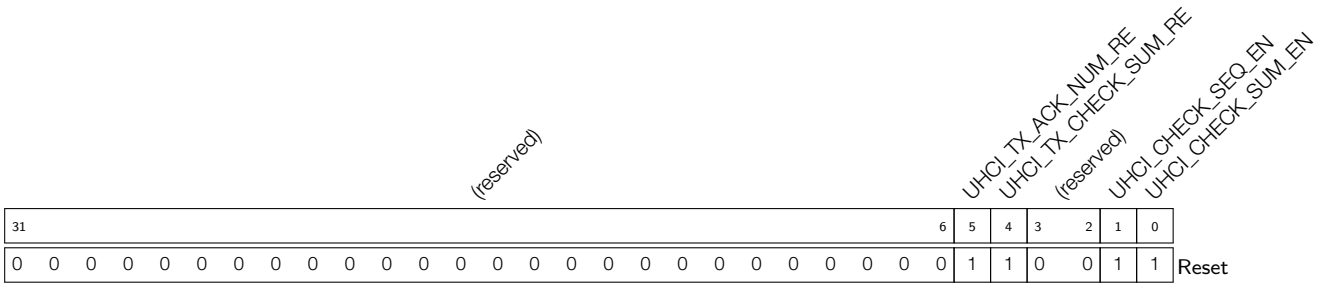
UHCI_INLINK_RESTART Set this bit to mount new inlink descriptors. (R/W)

UHCI_INLINK_START Set this bit to start dealing with the inlink descriptors. (R/W)

UHCI_INLINK_STOP Set this bit to stop dealing with the inlink descriptors. (R/W)

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

Register 13.37: UHCI_CONF1_REG (0x2C)



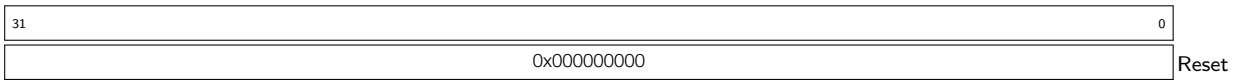
UHCI_TX_ACK_NUM_RE Reserved. Please initialize to 0. (R/W)

UHCI_TX_CHECK_SUM_RE Reserved. Please initialize to 0. (R/W)

UHCI_CHECK_SEQ_EN Reserved. Please initialize to 0. (R/W)

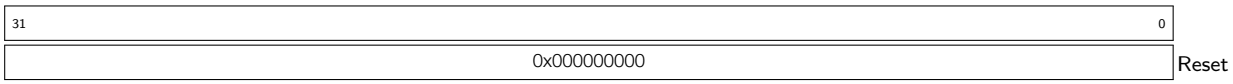
UHCI_CHECK_SUM_EN Reserved. Please initialize to 0. (R/W)

Register 13.38: UHCI_DMA_OUT_EOF_DES_ADDR_REG (0x38)



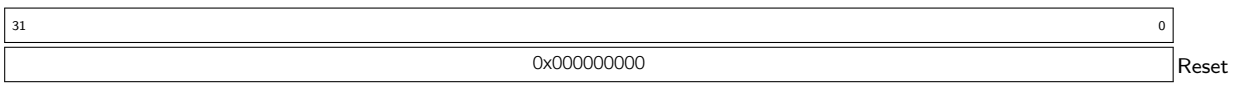
UHCI_DMA_OUT_EOF_DES_ADDR_REG This register stores the address of the outlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 13.39: UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG (0x3C)



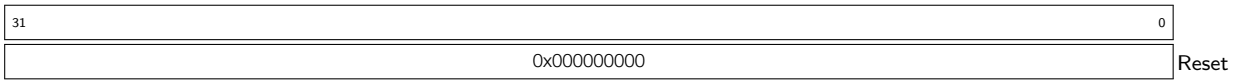
UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG This register stores the address of the inlink descriptor when the EOF bit in this descriptor is 1. (RO)

Register 13.40: UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG (0x40)



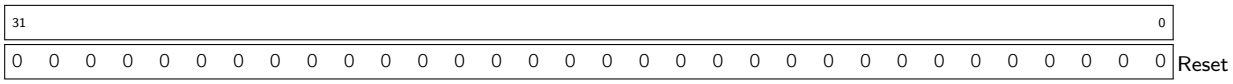
UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG This register stores the address of the inlink descriptor when there are some errors in this descriptor. (RO)

Register 13.41: UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG (0x44)



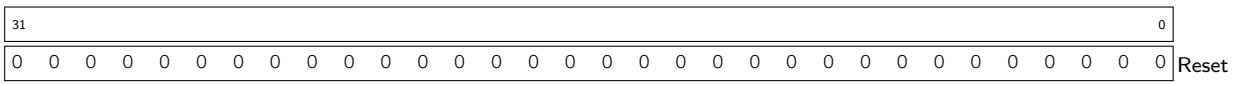
UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG This register stores the address of the outlink descriptor when there are some errors in this descriptor. (RO)

Register 13.42: UHCI_DMA_IN_DSCR_REG (0x4C)



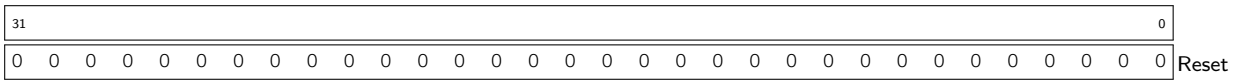
UHCI_DMA_IN_DSCR_REG The address of the current inlink descriptor *x*. (RO)

Register 13.43: UHCI_DMA_IN_DSCR_BF0_REG (0x50)



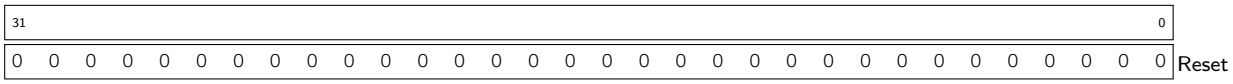
UHCI_DMA_IN_DSCR_BF0_REG The address of the last inlink descriptor *x-1*. (RO)

Register 13.44: UHCI_DMA_IN_DSCR_BF1_REG (0x54)



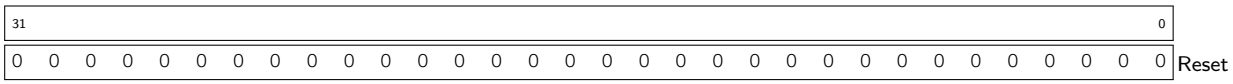
UHCI_DMA_IN_DSCR_BF1_REG The address of the second-to-last inlink descriptor *x-2*. (RO)

Register 13.45: UHCI_DMA_OUT_DSCR_REG (0x58)



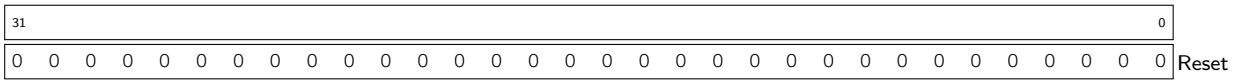
UHCI_DMA_OUT_DSCR_REG The address of the current outlink descriptor *y*. (RO)

Register 13.46: UHCI_DMA_OUT_DSCR_BF0_REG (0x5C)



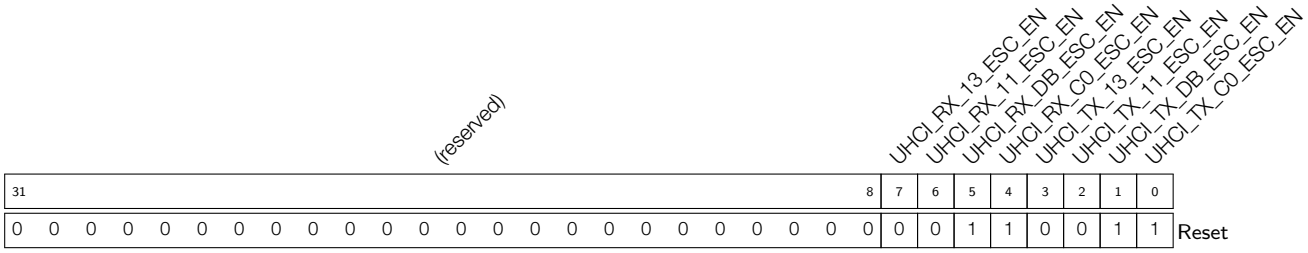
UHCI_DMA_OUT_DSCR_BF0_REG The address of the last outlink descriptor *y-1*. (RO)

Register 13.47: UHCI_DMA_OUT_DSCR_BF1_REG (0x60)



UHCI_DMA_OUT_DSCR_BF1_REG The address of the second-to-last outlink descriptor *y-2*. (RO)

Register 13.48: UHCI_ESCAPE_CONF_REG (0x64)



UHCI_RX_13_ESC_EN Set this bit to enable replacing flow control char 0x13, when DMA sends data. (R/W)

UHCI_RX_11_ESC_EN Set this bit to enable replacing flow control char 0x11, when DMA sends data. (R/W)

UHCI_RX_DB_ESC_EN Set this bit to enable replacing 0xdb char, when DMA sends data. (R/W)

UHCI_RX_C0_ESC_EN Set this bit to enable replacing 0xc0 char, when DMA sends data. (R/W)

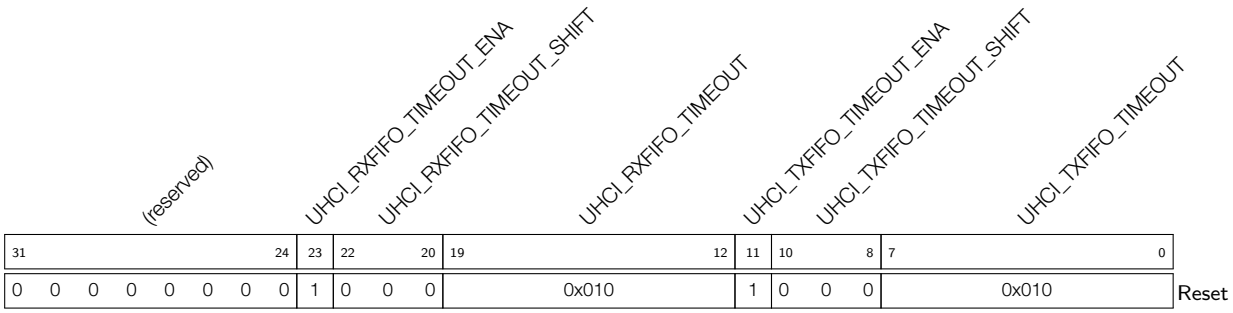
UHCI_TX_13_ESC_EN Set this bit to enable decoding flow control char 0x13, when DMA receives data. (R/W)

UHCI_TX_11_ESC_EN Set this bit to enable decoding flow control char 0x11, when DMA receives data. (R/W)

UHCI_TX_DB_ESC_EN Set this bit to enable decoding 0xdb char, when DMA receives data. (R/W)

UHCI_TX_C0_ESC_EN Set this bit to enable decoding 0xc0 char, when DMA receives data. (R/W)

Register 13.49: UHCI_HUNG_CONF_REG (0x68)



UHCI_RXFIFO_TIMEOUT_ENA This is the enable bit for DMA send-data timeout. (R/W)

UHCI_RXFIFO_TIMEOUT_SHIFT The tick count is cleared when its value is equal to or greater than (17'd8000»reg_rxfifo_timeout_shift). (R/W)

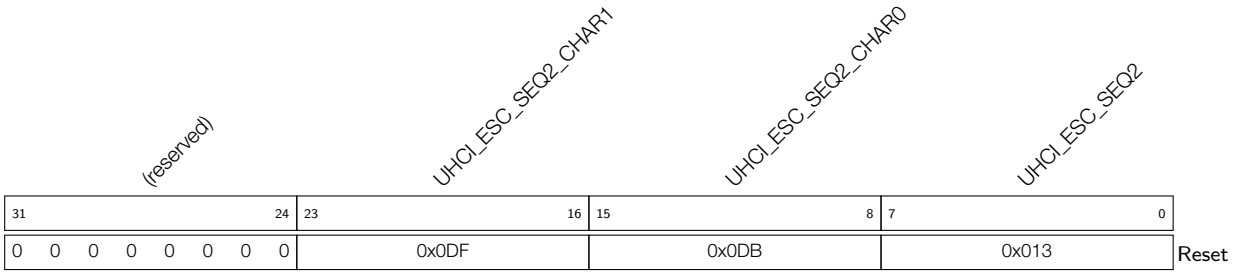
UHCI_RXFIFO_TIMEOUT This register stores the timeout value. When DMA takes more time to read data from RAM than what this register indicates, it will produce the UHCI_RX_HUNG_INT interrupt. (R/W)

UHCI_TXFIFO_TIMEOUT_ENA The enable bit for Tx FIFO receive-data timeout (R/W)

UHCI_TXFIFO_TIMEOUT_SHIFT The tick count is cleared when its value is equal to or greater than (17'd8000»reg_txfifo_timeout_shift). (R/W)

UHCI_TXFIFO_TIMEOUT This register stores the timeout value. When DMA takes more time to receive data than what this register indicates, it will produce the UHCI_TX_HUNG_INT interrupt. (R/W)

Register 13.50: UHCI_ESC_CONF*n*_REG (*n*: 0-3) (0xB0+4n*)**



UHCI_ESC_SEQ2_CHAR1 This register stores the second char used to replace the reg_esc_seq2 in data. (R/W)

UHCI_ESC_SEQ2_CHAR0 This register stores the first char used to replace the reg_esc_seq2 in data. (R/W)

UHCI_ESC_SEQ2 This register stores the flow_control char to turn off the flow_control. (R/W)

14. LED_PWM

14.1 Introduction

The LED_PWM controller is primarily designed to control the intensity of LEDs, although it can be used to generate PWM signals for other purposes as well. It has 16 channels which can generate independent waveforms that can be used to drive RGB LED devices. For maximum flexibility, the high-speed as well as the low-speed channels can be driven from one of four high-speed/low-speed timers. The PWM controller also has the ability to automatically increase or decrease the duty cycle gradually, allowing for fades without any processor interference. To increase resolution, the LED_PWM controller is also able to dither between two values, when a fractional PWM value is configured.

The LED_PWM controller has eight high-speed and eight low-speed PWM generators. In this document, they will be referred to as hsch*n* and lsch*n*, respectively. These channels can be driven from four timers which will be indicated by h_timer*x* and l_timer*x*.

14.2 Functional Description

14.2.1 Architecture

Figure 84 shows the architecture of the LED_PWM controller. As can be seen in the figure, the LED_PWM controller contains eight high-speed and eight low-speed channels. There are four high-speed clock modules for the high-speed channels, from which one h_timer*x* can be selected. There are also four low-speed clock modules for the low-speed channels, from which one l_timer*x* can be selected.

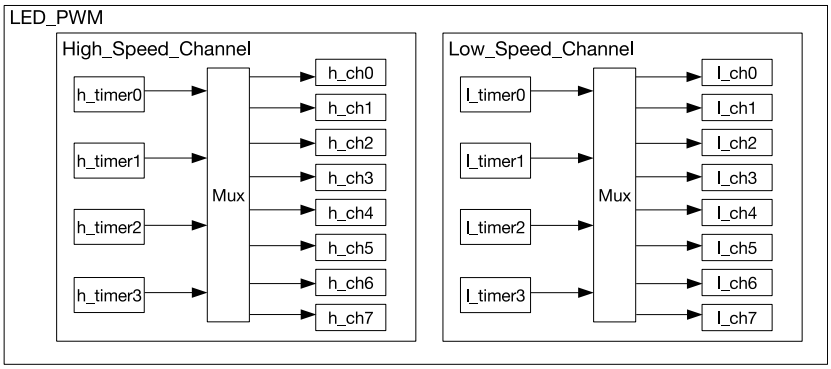


Figure 84: LED_PWM Architecture

Figure 85 illustrates a PWM channel with its selected timer; in this instance a high-speed channel and associated high-speed timer.

14.2.2 Timers

A high-speed timer consists of a multiplexer to select one of two clock sources: either REF_TICK or APB_CLK. For more information on the clock sources, please see Chapter Reset And Clock. The input clock is divided down by a divider first. The division factor is specified by LEDC_CLK_DIV_NUM_HSTIMER*x* which contains a fixed point number: the highest 10 bits represent the integer portion A, while the lowest eight bits contain the fractional portion B. The effective division factor is as follows:

$$LEDC\_CLK\_DIV\_NUM\_HSTIMERx = A\frac{B}{256}$$

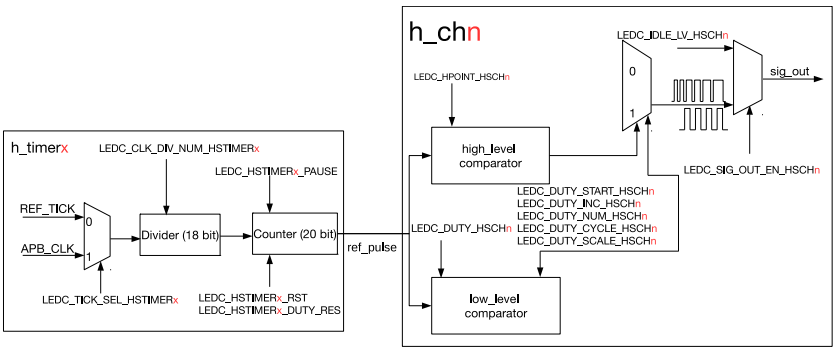


Figure 85: LED_PWM High-speed Channel Diagram

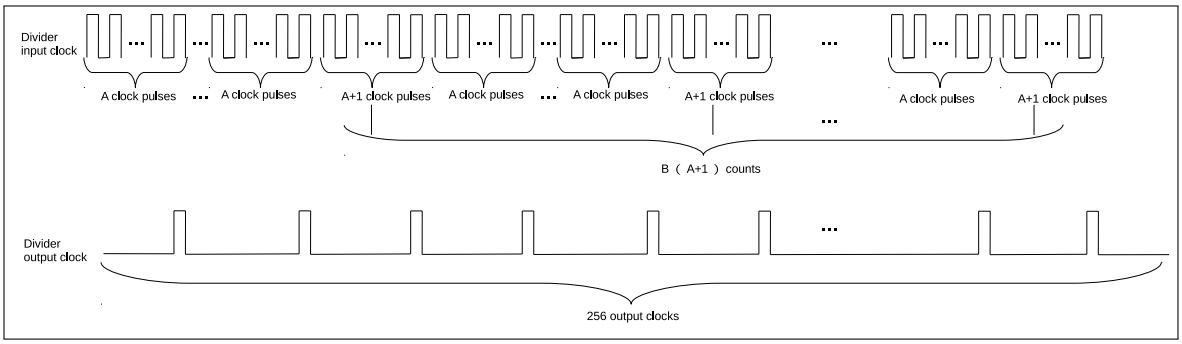


Figure 86: LED_PWM Divider

Figure 86 shows the input/output clock when the fractional portion B is not 0. As shown in the firgure, the 256 output clocks consist of B output clocks as result of division by (A+1) divider and (256-B) output clocks as result of division by A divider. The B output clocks are evenly distributed in the 256 output clocks.

The output clock of the divider is the base clock for the counter which will count up to the value specified in LEDC_HSTIMERx_DUTY_RES. An overflow interrupt will be generated once the counting value reaches $2^{LEDC\_HSTIMERx\_DUTY\_RES} - 1$, at which point the counter restarts counting from 0. It is also possible to reset, suspend, and read the values of the counter by software.

The output signal of the timer is the 20-bit value generated by the counter. The cycle period of this signal defines the frequency of the signals of any PWM channels connected to this timer. This frequency depends on both the division factor of the divider, as well as the range of the counter:

$$f_{\text{sig\_out}} = \frac{f_{\text{REF\_TICK}} \cdot (!\text{LEDC\_TICK\_SEL\_HSTIMERx}) + f_{\text{APB\_CLK}} \cdot \text{LEDC\_TICK\_SEL\_HSTIMERx}}{\text{LEDC\_CLK\_DIV\_NUM\_HSTIMERx} \cdot 2^{\text{LEDC\_HSTIMERx\_DUTY\_RES}}}$$

The low-speed timers l_timerx on the low-speed channel differ from the high-speed timers h_timerx in two aspects:

1. Where the high-speed timer clock source can be clocked from REF_TICK or APB_CLK, the low-speed timers are sourced from either REF_TICK or SLOW_CLOCK. The SLOW_CLOCK source can be either APB_CLK (80 MHz) or 8 MHz, and can be selected using LEDC_APB_CLK_SEL.
2. The high-speed counter and divider are glitch-free, which means that if the software modifies the maximum counter or divisor value, the update will come into effect after the next overflow interrupt. In contrast, the low-speed counter and divider will update these values only when LEDC_LSTIMERx_PARA_UP is set.

14.2.3 Channels

A channel takes the 20-bit value from the counter of the selected high-speed timer and compares it to a set of two values in order to set the channel output. The first value it is compared to is the content of LEDC_HPOINT_HSCH*n*; if these two match, the output will be latched high. The second value is the sum of LEDC_HPOINT_HSCH*n* and LEDC_DUTY_HSCH*n*[24..4]. When this value is reached, the output is latched low. By using these two values, the relative phase and the duty cycle of the PWM output can be set. Figure 87 illustrates this.

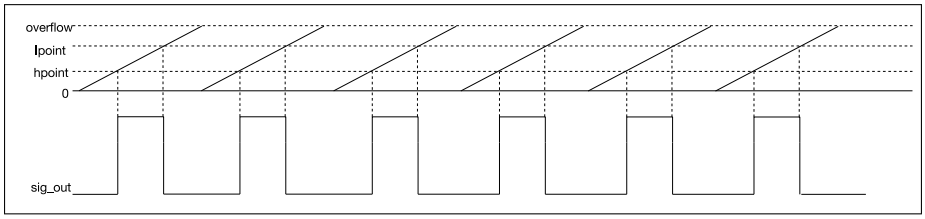


Figure 87: LED PWM Output Signal Diagram

LEDC_DUTY_HSCH*n* is a fixed-point register with four fractional bits. As mentioned before, when LEDC_DUTY_HSCH*n*[24..4] is used in the PWM calculation directly, LEDC_DUTY_HSCH*n*[3..0] can be used to dither the output. If this value is non-zero, with a statistical chance of LEDC_DUTY_HSCH*n*[3..0]/16, the actual PWM pulse will be one cycle longer. This effectively increases the resolution of the PWM generator to 25 bits, but at the cost of a slight jitter in the duty cycle.

The channels also have the ability to automatically fade from one duty cycle value to another. This feature is enabled by setting LEDC_DUTY_START_HSCH*n*. When this bit is set, the PWM controller will automatically increment or decrement the value in LEDC_DUTY_HSCH*n*, depending on whether the bit LEDC_DUTY_INC_HSCH*n* is set or cleared, respectively. The speed the duty cycle changes is defined as such: every time the LEDC_DUTY_CYCLE_HSCH*n* cycles, the content of LEDC_DUTY_SCALE_HSCH*n* is added to or subtracted from LEDC_DUTY_HSCH*n*[24..4]. The length of the fade can be limited by setting LEDC_DUTY_NUM_HSCH*n*: the fade will only last that number of cycles before finishing. A finished fade also generates an interrupt.

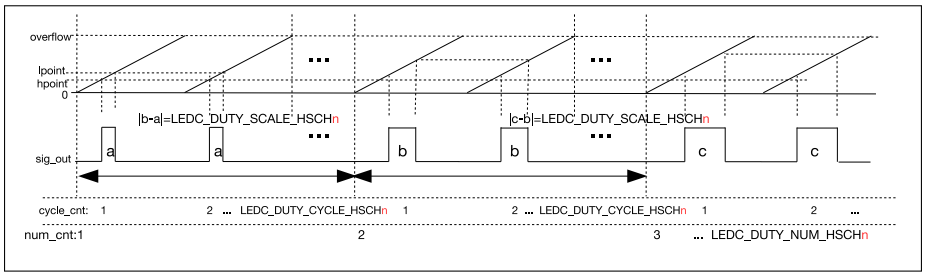


Figure 88: Output Signal Diagram of Fading Duty Cycle

Figure 88 is an illustration of this. In this configuration, LEDC_DUTY_NUM_HSCH*n*_R increases by LEDC_DUTY_SCALE_HSCH*n* for every LEDC_DUTY_CYCLE_HSCH*n* clock cycles, which is reflected in the duty cycle of the output signal.

Notes

- When LEDC is in fade mode, configure the second fade only after LEDC_DUTY_CHNG_END_HSCH*n* or LEDC_DUTY_CHNG_END_LSCH*n* interrupt is generated.
- When LEDC is in decremental fade mode and LEDC_DUTY_HSCH*n* is $2^{LEDC\_HSTIMER_x\_DUTY\_RES}$,

LEDC_DUTY_SCALE_HSCH*n* cannot be set to 1. When LEDC is in decremental fade mode and LEDC_DUTY_LSCH*n* is $2^{LEDC\_LSTIMERx\_DUTY\_RES}$, LEDC_DUTY_SCALE_LSCH*n* cannot be set to 1.

14.2.4 Interrupts

- LEDC_DUTY_CHNG_END_LSCH*n*_INT: Triggered when a fade on a low-speed channel has finished.
- LEDC_DUTY_CHNG_END_HSCH*n*_INT: Triggered when a fade on a high-speed channel has finished.
- LEDC_HS_TIMER*x*_OVF_INT: Triggered when a high-speed timer has reached its maximum counter value.
- LEDC_LS_TIMER*x*_OVF_INT: Triggered when a low-speed timer has reached its maximum counter value.

14.3 Register Summary

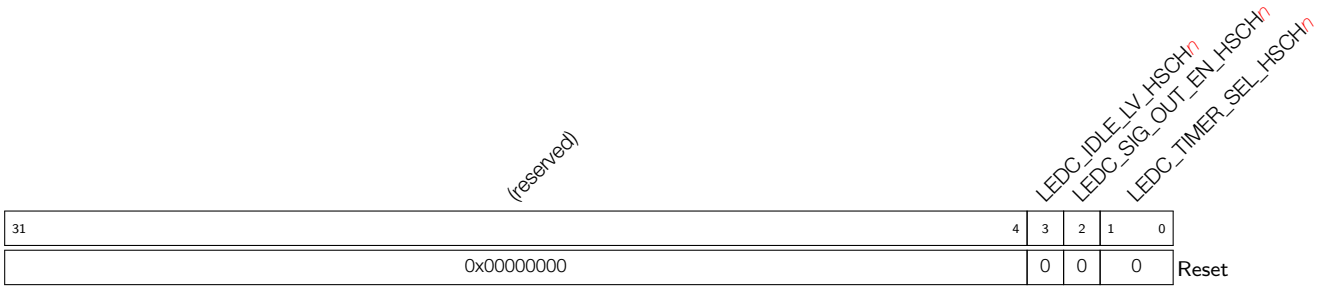
Name	Description	Address	Access
Configuration registers			
LEDC_CONF_REG	Global ledc configuration register	0x3FF59190	R/W
LEDC_HSCH0_CONF0_REG	Configuration register 0 for high-speed channel 0	0x3FF59000	R/W
LEDC_HSCH1_CONF0_REG	Configuration register 0 for high-speed channel 1	0x3FF59014	R/W
LEDC_HSCH2_CONF0_REG	Configuration register 0 for high-speed channel 2	0x3FF59028	R/W
LEDC_HSCH3_CONF0_REG	Configuration register 0 for high-speed channel 3	0x3FF5903C	R/W
LEDC_HSCH4_CONF0_REG	Configuration register 0 for high-speed channel 4	0x3FF59050	R/W
LEDC_HSCH5_CONF0_REG	Configuration register 0 for high-speed channel 5	0x3FF59064	R/W
LEDC_HSCH6_CONF0_REG	Configuration register 0 for high-speed channel 6	0x3FF59078	R/W
LEDC_HSCH7_CONF0_REG	Configuration register 0 for high-speed channel 7	0x3FF5908C	R/W
LEDC_HSCH0_CONF1_REG	Configuration register 1 for high-speed channel 0	0x3FF5900C	R/W
LEDC_HSCH1_CONF1_REG	Configuration register 1 for high-speed channel 1	0x3FF59020	R/W
LEDC_HSCH2_CONF1_REG	Configuration register 1 for high-speed channel 2	0x3FF59034	R/W
LEDC_HSCH3_CONF1_REG	Configuration register 1 for high-speed channel 3	0x3FF59048	R/W
LEDC_HSCH4_CONF1_REG	Configuration register 1 for high-speed channel 4	0x3FF5905C	R/W
LEDC_HSCH5_CONF1_REG	Configuration register 1 for high-speed channel 5	0x3FF59070	R/W
LEDC_HSCH6_CONF1_REG	Configuration register 1 for high-speed channel 6	0x3FF59084	R/W
LEDC_HSCH7_CONF1_REG	Configuration register 1 for high-speed channel 7	0x3FF59098	R/W
LEDC_LSCH0_CONF0_REG	Configuration register 0 for low-speed channel 0	0x3FF590A0	R/W
LEDC_LSCH1_CONF0_REG	Configuration register 0 for low-speed channel 1	0x3FF590B4	R/W
LEDC_LSCH2_CONF0_REG	Configuration register 0 for low-speed channel 2	0x3FF590C8	R/W
LEDC_LSCH3_CONF0_REG	Configuration register 0 for low-speed channel 3	0x3FF590DC	R/W
LEDC_LSCH4_CONF0_REG	Configuration register 0 for low-speed channel 4	0x3FF590F0	R/W
LEDC_LSCH5_CONF0_REG	Configuration register 0 for low-speed channel 5	0x3FF59104	R/W
LEDC_LSCH6_CONF0_REG	Configuration register 0 for low-speed channel 6	0x3FF59118	R/W
LEDC_LSCH7_CONF0_REG	Configuration register 0 for low-speed channel 7	0x3FF5912C	R/W
LEDC_LSCH0_CONF1_REG	Configuration register 1 for low-speed channel 0	0x3FF590AC	R/W
LEDC_LSCH1_CONF1_REG	Configuration register 1 for low-speed channel 1	0x3FF590C0	R/W
LEDC_LSCH2_CONF1_REG	Configuration register 1 for low-speed channel 2	0x3FF590D4	R/W
LEDC_LSCH3_CONF1_REG	Configuration register 1 for low-speed channel 3	0x3FF590E8	R/W
LEDC_LSCH4_CONF1_REG	Configuration register 1 for low-speed channel 4	0x3FF590FC	R/W

Name	Description	Address	Access
LEDC_LSCH5_CONF1_REG	Configuration register 1 for low-speed channel 5	0x3FF59110	R/W
LEDC_LSCH6_CONF1_REG	Configuration register 1 for low-speed channel 6	0x3FF59124	R/W
LEDC_LSCH7_CONF1_REG	Configuration register 1 for low-speed channel 7	0x3FF59138	R/W
Duty-cycle registers			
LEDC_HSCH0_DUTY_REG	Initial duty cycle for high-speed channel 0	0x3FF59008	R/W
LEDC_HSCH1_DUTY_REG	Initial duty cycle for high-speed channel 1	0x3FF5901C	R/W
LEDC_HSCH2_DUTY_REG	Initial duty cycle for high-speed channel 2	0x3FF59030	R/W
LEDC_HSCH3_DUTY_REG	Initial duty cycle for high-speed channel 3	0x3FF59044	R/W
LEDC_HSCH4_DUTY_REG	Initial duty cycle for high-speed channel 4	0x3FF59058	R/W
LEDC_HSCH5_DUTY_REG	Initial duty cycle for high-speed channel 5	0x3FF5906C	R/W
LEDC_HSCH6_DUTY_REG	Initial duty cycle for high-speed channel 6	0x3FF59080	R/W
LEDC_HSCH7_DUTY_REG	Initial duty cycle for high-speed channel 7	0x3FF59094	R/W
LEDC_HSCH0_DUTY_R_REG	Current duty cycle for high-speed channel 0	0x3FF59010	RO
LEDC_HSCH1_DUTY_R_REG	Current duty cycle for high-speed channel 1	0x3FF59024	RO
LEDC_HSCH2_DUTY_R_REG	Current duty cycle for high-speed channel 2	0x3FF59038	RO
LEDC_HSCH3_DUTY_R_REG	Current duty cycle for high-speed channel 3	0x3FF5904C	RO
LEDC_HSCH4_DUTY_R_REG	Current duty cycle for high-speed channel 4	0x3FF59060	RO
LEDC_HSCH5_DUTY_R_REG	Current duty cycle for high-speed channel 5	0x3FF59074	RO
LEDC_HSCH6_DUTY_R_REG	Current duty cycle for high-speed channel 6	0x3FF59088	RO
LEDC_HSCH7_DUTY_R_REG	Current duty cycle for high-speed channel 7	0x3FF5909C	RO
LEDC_LSCH0_DUTY_REG	Initial duty cycle for low-speed channel 0	0x3FF590A8	R/W
LEDC_LSCH1_DUTY_REG	Initial duty cycle for low-speed channel 1	0x3FF590BC	R/W
LEDC_LSCH2_DUTY_REG	Initial duty cycle for low-speed channel 2	0x3FF590D0	R/W
LEDC_LSCH3_DUTY_REG	Initial duty cycle for low-speed channel 3	0x3FF590E4	R/W
LEDC_LSCH4_DUTY_REG	Initial duty cycle for low-speed channel 4	0x3FF590F8	R/W
LEDC_LSCH5_DUTY_REG	Initial duty cycle for low-speed channel 5	0x3FF5910C	R/W
LEDC_LSCH6_DUTY_REG	Initial duty cycle for low-speed channel 6	0x3FF59120	R/W
LEDC_LSCH7_DUTY_REG	Initial duty cycle for low-speed channel 7	0x3FF59134	R/W
LEDC_LSCH0_DUTY_R_REG	Current duty cycle for low-speed channel 0	0x3FF590B0	RO
LEDC_LSCH1_DUTY_R_REG	Current duty cycle for low-speed channel 1	0x3FF590C4	RO
LEDC_LSCH2_DUTY_R_REG	Current duty cycle for low-speed channel 2	0x3FF590D8	RO
LEDC_LSCH3_DUTY_R_REG	Current duty cycle for low-speed channel 3	0x3FF590EC	RO
LEDC_LSCH4_DUTY_R_REG	Current duty cycle for low-speed channel 4	0x3FF59100	RO
LEDC_LSCH5_DUTY_R_REG	Current duty cycle for low-speed channel 5	0x3FF59114	RO
LEDC_LSCH6_DUTY_R_REG	Current duty cycle for low-speed channel 6	0x3FF59128	RO
LEDC_LSCH7_DUTY_R_REG	Current duty cycle for low-speed channel 7	0x3FF5913C	RO
Timer registers			
LEDC_HSTIMER0_CONF_REG	High-speed timer 0 configuration	0x3FF59140	R/W
LEDC_HSTIMER1_CONF_REG	High-speed timer 1 configuration	0x3FF59148	R/W
LEDC_HSTIMER2_CONF_REG	High-speed timer 2 configuration	0x3FF59150	R/W
LEDC_HSTIMER3_CONF_REG	High-speed timer 3 configuration	0x3FF59158	R/W
LEDC_HSTIMER0_VALUE_REG	High-speed timer 0 current counter value	0x3FF59144	RO
LEDC_HSTIMER1_VALUE_REG	High-speed timer 1 current counter value	0x3FF5914C	RO
LEDC_HSTIMER2_VALUE_REG	High-speed timer 2 current counter value	0x3FF59154	RO

Name	Description	Address	Access
LEDC_HSTIMER3_VALUE_REG	High-speed timer 3 current counter value	0x3FF5915C	RO
LEDC_LSTIMER0_CONF_REG	Low-speed timer 0 configuration	0x3FF59160	R/W
LEDC_LSTIMER1_CONF_REG	Low-speed timer 1 configuration	0x3FF59168	R/W
LEDC_LSTIMER2_CONF_REG	Low-speed timer 2 configuration	0x3FF59170	R/W
LEDC_LSTIMER3_CONF_REG	Low-speed timer 3 configuration	0x3FF59178	R/W
LEDC_LSTIMER0_VALUE_REG	Low-speed timer 0 current counter value	0x3FF59164	RO
LEDC_LSTIMER1_VALUE_REG	Low-speed timer 1 current counter value	0x3FF5916C	RO
LEDC_LSTIMER2_VALUE_REG	Low-speed timer 2 current counter value	0x3FF59174	RO
LEDC_LSTIMER3_VALUE_REG	Low-speed timer 3 current counter value	0x3FF5917C	RO
Interrupt registers			
LEDC_INT_RAW_REG	Raw interrupt status	0x3FF59180	RO
LEDC_INT_ST_REG	Masked interrupt status	0x3FF59184	RO
LEDC_INT_ENA_REG	Interrupt enable bits	0x3FF59188	R/W
LEDC_INT_CLR_REG	Interrupt clear bits	0x3FF5918C	WO

14.4 Registers

Register 14.1: LEDC_HSCH*n*_CONF0_REG (*n*: 0-7) (0x1C+0x10n*)**



LEDC_IDLE_LV_HSCH*n* This bit is used to control the output value when high-speed channel *n* is inactive. (R/W)

LEDC_SIG_OUT_EN_HSCH*n* This is the output enable control bit for high-speed channel *n*. (R/W)

LEDC_TIMER_SEL_HSCH*n* There are four high-speed timers. These two bits are used to select one of them for high-speed channel *n*: (R/W)

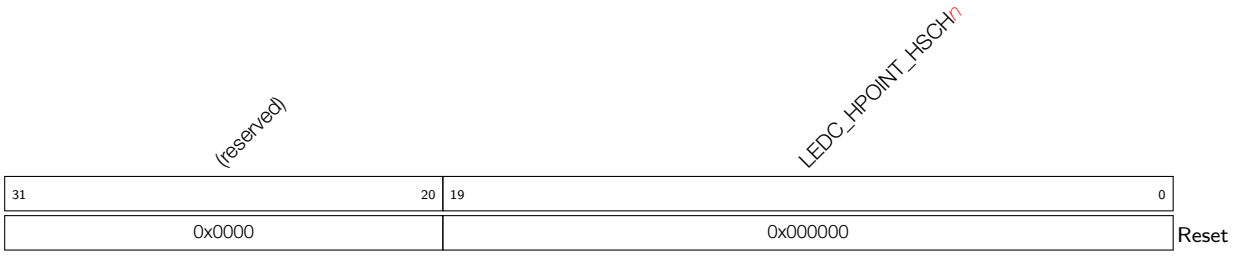
0: select hstimer0;

1: select hstimer1;

2: select hstimer2;

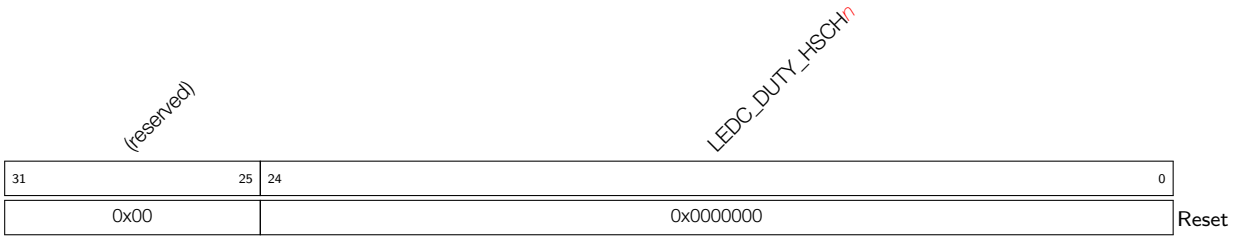
3: select hstimer3.

Register 14.2: LEDC_HSCH*n*_HPOINT_REG (*n*: 0-7) (0x20+0x10n*)**



LEDC_HPOINT_HSCH*n* The output value changes to high when htimerx(x=[0,3]), selected by high-speed channel *n*, has reached LEDC_HPOINT_HSCH*n*[19:0]. (R/W)

Register 14.3: LEDC_HSCH*n*_DUTY_REG (*n*: 0-7) (0x24+0x10n*)**



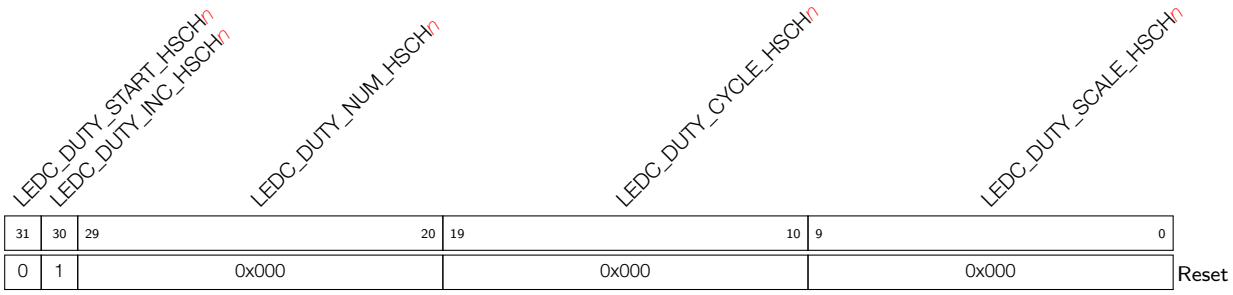
LEDC_DUTY_HSCH*n* The register is used to control output duty. When hstimerx(x=[0,3]), selected by high-speed channel *n*, has reached LEDC_LPOINT_HSCH*n*, the output signal changes to low. (R/W)

LEDC_LPOINT_HSCH*n*=LEDC_LPOINT_HSCH*n*[19:0]+LEDC_DUTY_HSCH*n*[24:4] (1)

LEDC_LPOINT_HSCH*n*=LEDC_LPOINT_HSCH*n*[19:0]+LEDC_DUTY_HSCH*n*[24:4] +1) (2)

See the Functional Description for more information on when (1) or (2) is chosen.

Register 14.4: LEDC_HSCH*n*_CONF1_REG (*n*: 0-7) (0x28+0x10n*)**



LEDC_DUTY_START_HSCH*n* When LEDC_DUTY_NUM_HSCH*n*, LEDC_DUTY_CYCLE_HSCH*n* and LEDC_DUTY_SCALE_HSCH*n* has been configured, these register will not take effect until LEDC_DUTY_START_HSCH*n* is set. This bit is automatically cleared by hardware. (R/W)

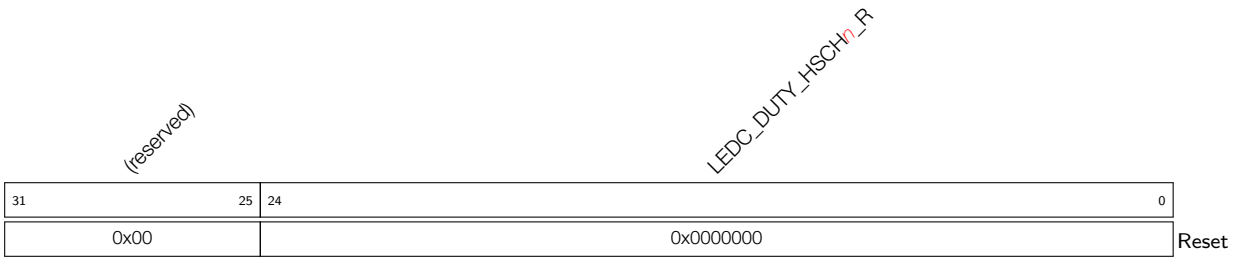
LEDC_DUTY_INC_HSCH*n* This register is used to increase or decrease the duty of output signal for high-speed channel *n*. (R/W)

LEDC_DUTY_NUM_HSCH*n* This register is used to control the number of times the duty cycle is increased or decreased for high-speed channel *n*. (R/W)

LEDC_DUTY_CYCLE_HSCH*n* This register is used to increase or decrease the duty cycle every time LEDC_DUTY_CYCLE_HSCH*n* cycles for high-speed channel *n*. (R/W)

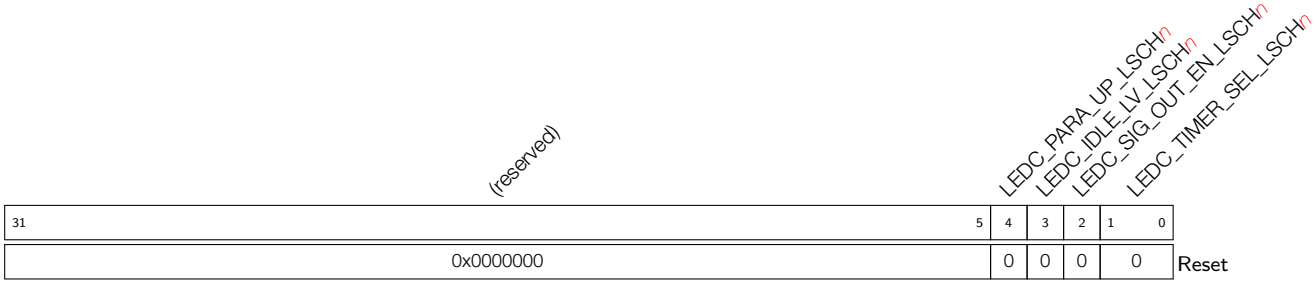
LEDC_DUTY_SCALE_HSCH*n* This register is used to increase or decrease the step scale for high-speed channel *n*. (R/W)

Register 14.5: LEDC_HSCH*n*_DUTY_R_REG (*n*: 0-7) (0x2C+0x10n*)**



LEDC_DUTY_HSCH*n*_R This register represents the current duty cycle of the output signal for high-speed channel *n*. (RO)

Register 14.6: LEDC_LSCH*n*_CONF0_REG (*n*: 0-7) (0xBC+0x10n*)**



LEDC_PARA_UP_LSCH*n* This bit is used to update register LEDC_LSCH*n*_HPOINT and LEDC_LSCH*n*_DUTY for low-speed channel *n*. (R/W)

LEDC_IDLE_LV_LSCH*n* This bit is used to control the output value, when low-speed channel *n* is inactive. (R/W)

LEDC_SIG_OUT_EN_LSCH*n* This is the output enable control bit for low-speed channel *n*. (R/W)

LEDC_TIMER_SEL_LSCH*n* There are four low-speed timers, the two bits are used to select one of them for low-speed channel *n*. (R/W)

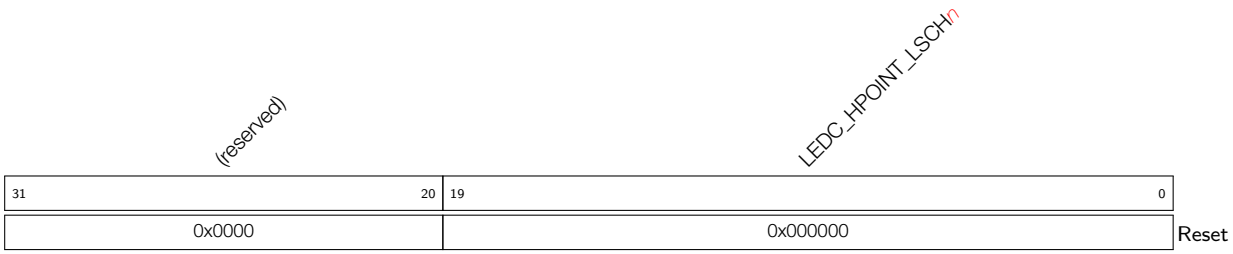
0: select lstimer0;

1: select lstimer1;

2: select lstimer2;

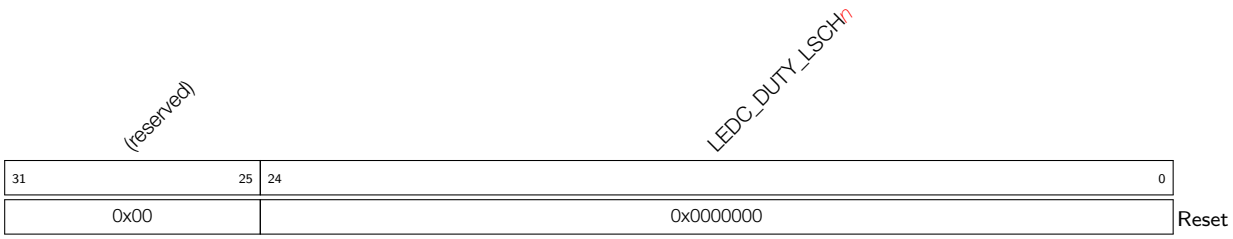
3: select lstimer3.

Register 14.7: LEDC_LSCH*n*_HPOINT_REG (*n*: 0-7) (0xC0+0x10n*)**



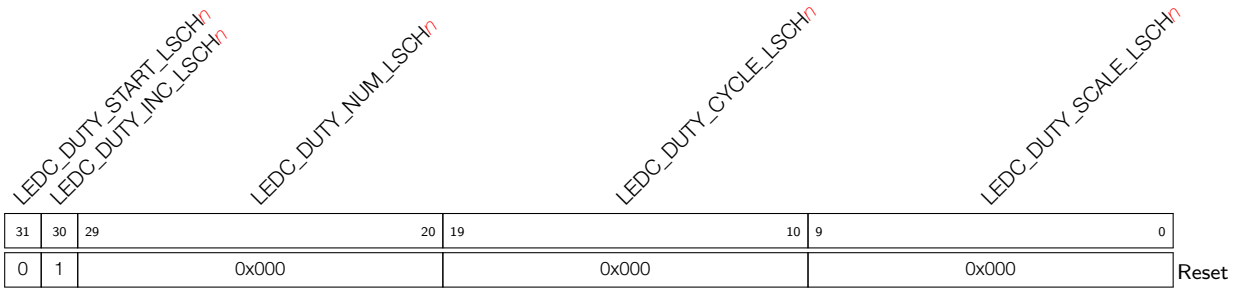
LEDC_HPOINT_LSCH*n* The output value changes to high when lstimerx(x=[0,3]), selected by low-speed channel *n*, has reached LEDC_HPOINT_LSCH*n*[19:0]. (R/W)

Register 14.8: LEDC_LSCH*n*_DUTY_REG (*n*: 0-7) (0xC4+0x10n*)**



LEDC_DUTY_LSCH*n* The register is used to control output duty. When lstimerx(x=[0,3]), chosen by low-speed channel *n*, has reached LEDC_LPOINT_LSCH*n*,the output signal changes to low. (R/W)
LEDC_LPOINT_LSCH*n*=(LEDC_HPOINT_LSCH*n*[19:0]+LEDC_DUTY_LSCH*n*[24:4]) (1)
LEDC_LPOINT_LSCH*n*=(LEDC_HPOINT_LSCH*n*[19:0]+LEDC_DUTY_LSCH*n*[24:4] +1) (2)
See the Functional Description for more information on when (1) or (2) is chosen.

Register 14.9: LEDC_LSCH*n*_CONF1_REG (*n*: 0-7) (0xC8+0x10n*)**



LEDC_DUTY_START_LSCH*n* When LEDC_DUTY_NUM_HSCH*n*, LEDC_DUTY_CYCLE_HSCH*n* and LEDC_DUTY_SCALE_HSCH*n* have been configured, these settings will not take effect until set LEDC_DUTY_START_HSCH*n*. This bit is automatically cleared by hardware. (R/W)

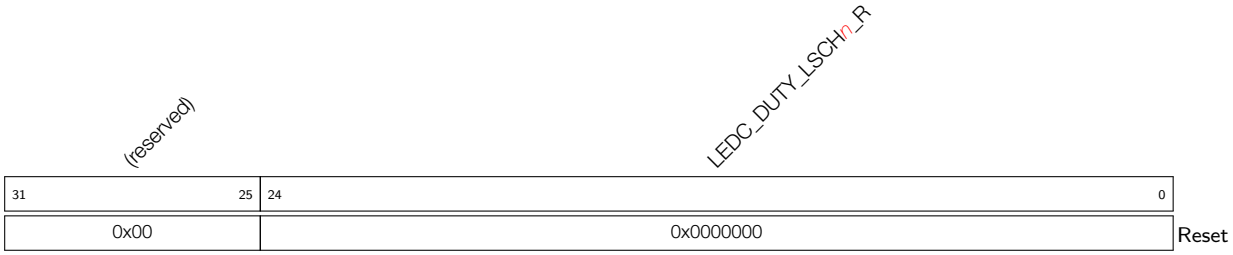
LEDC_DUTY_INC_LSCH*n* This register is used to increase or decrease the duty of output signal for low-speed channel *n*. (R/W)

LEDC_DUTY_NUM_LSCH*n* This register is used to control the number of times the duty cycle is increased or decreased for low-speed channel *n*. (R/W)

LEDC_DUTY_CYCLE_LSCH*n* This register is used to increase or decrease the duty every LEDC_DUTY_CYCLE_LSCH*n* cycles for low-speed channel *n*. (R/W)

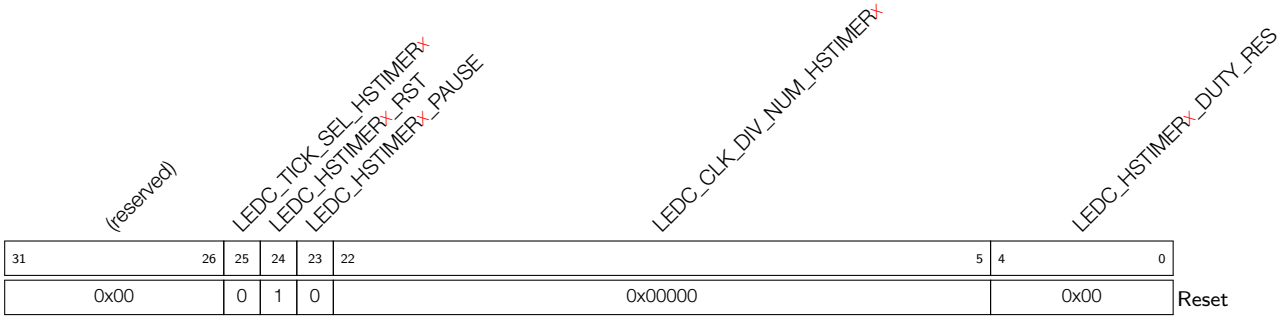
LEDC_DUTY_SCALE_LSCH*n* This register is used to increase or decrease the step scale for low-speed channel *n*. (R/W)

Register 14.10: LEDC_LSCH*n*_DUTY_R_REG (*n*: 0-7) (0xCC+0x10n*)**



LEDC_DUTY_LSCH*n*_R This register represents the current duty of the output signal for low-speed channel *n*. (RO)

Register 14.11: LEDC_HSTIMERx_CONF_REG (x: 0-3) (0x140+8*x)



LEDC_TICK_SEL_HSTIMERx This bit is used to select APB_CLK or REF_TICK for high-speed timer *x*. (R/W)

1: APB_CLK;

0: REF_TICK.

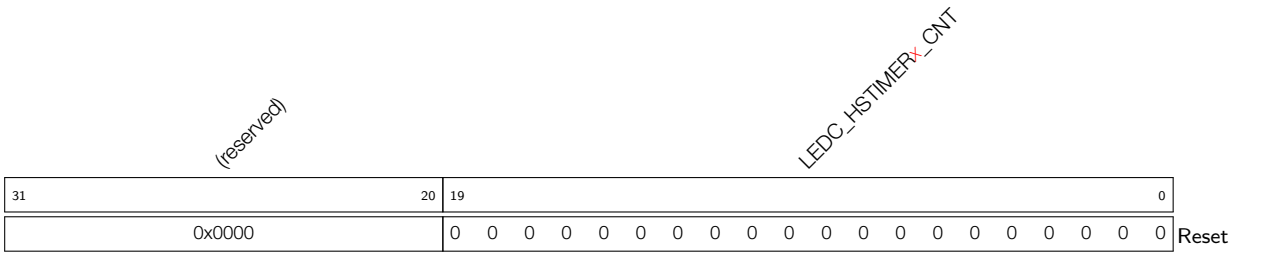
LEDC_HSTIMERx_RST This bit is used to reset high-speed timer *x*. The counter value will be 'zero' after reset. (R/W)

LEDC_HSTIMERx_PAUSE This bit is used to suspend the counter in high-speed timer *x*. (R/W)

LEDC_CLK_DIV_NUM_HSTIMERx This register is used to configure the division factor for the divider in high-speed timer *x*. The least significant eight bits represent the fractional part. (R/W)

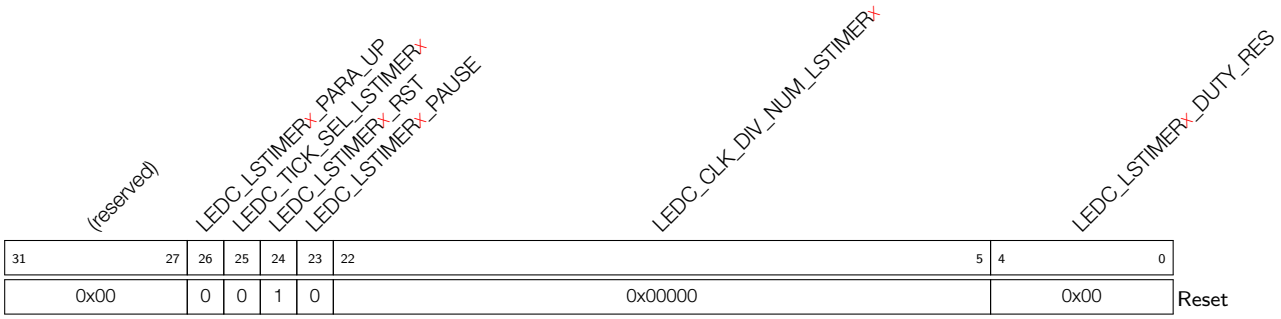
LEDC_HSTIMERx_DUTY_RES This register is used to control the range of the counter in high-speed timer *x*. The counter range is [0, $2^{LEDC\_HSTIMERx\_DUTY\_RES}$], the maximum bit width for counter is 20. (R/W)

Register 14.12: LEDC_HSTIMERx_VALUE_REG (x: 0-3) (0x144+8*x)



LEDC_HSTIMERx_CNT Software can read this register to get the current counter value of high-speed timer *x*. (RO)

Register 14.13: LEDC_LSTIMERx_CONF_REG (x: 0-3) (0x160+8*x)



LEDC_LSTIMERx_PARA_UP Set this bit to update LEDC_CLK_DIV_NUM_LSTIMEx and LEDC_LSTIMERx_DUTY_RES. (R/W)

LEDC_TICK_SEL_LSTIMERx This bit is used to select SLOW_CLK or REF_TICK for low-speed timer x. (R/W)

1: SLOW_CLK;

0: REF_TICK.

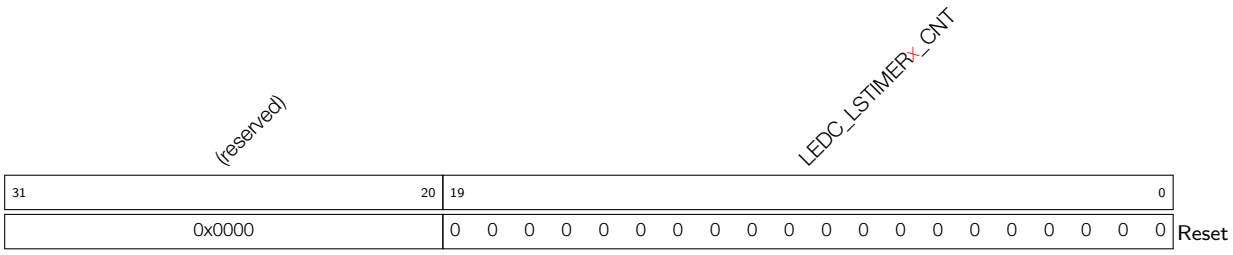
LEDC_LSTIMERx_RST This bit is used to reset low-speed timer x. The counter will show 0 after reset. (R/W)

LEDC_LSTIMERx_PAUSE This bit is used to suspend the counter in low-speed timer x. (R/W)

LEDC_CLK_DIV_NUM_LSTIMERx This register is used to configure the division factor for the divider in low-speed timer x. The least significant eight bits represent the fractional part. (R/W)

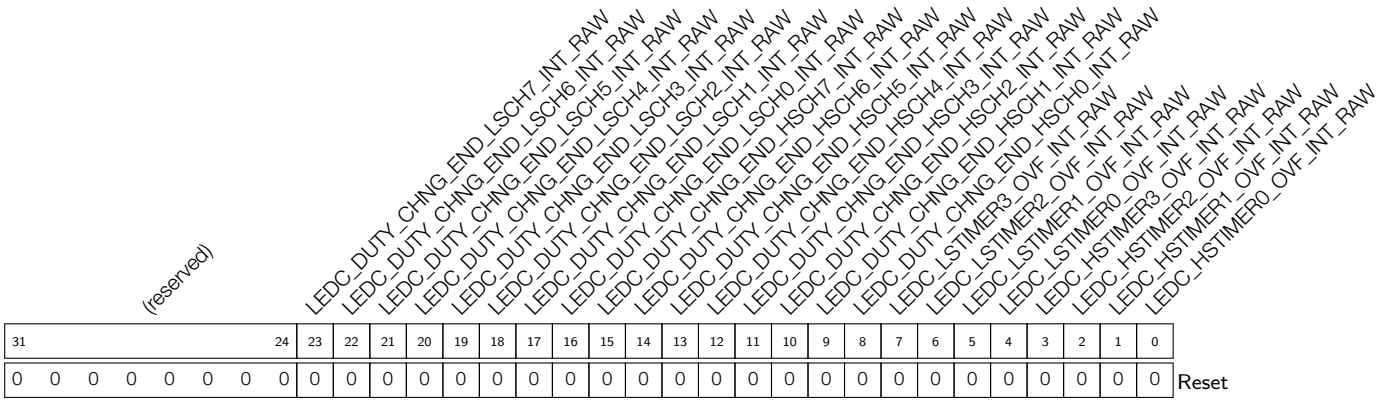
LEDC_LSTIMERx_DUTY_RES This register is used to control the range of the counter in low-speed timer x. The counter range is [0, $2^{LEDC\_LSTIMERx\_DUTY\_RES}$], the max bit width for counter is 20. (R/W)

Register 14.14: LEDC_LSTIMERx_VALUE_REG (x: 0-3) (0x164+8*x)



LEDC_LSTIMERx_CNT Software can read this register to get the current counter value of low-speed timer x. (RO)

Register 14.15: LEDC_INT_RAW_REG (0x0180)



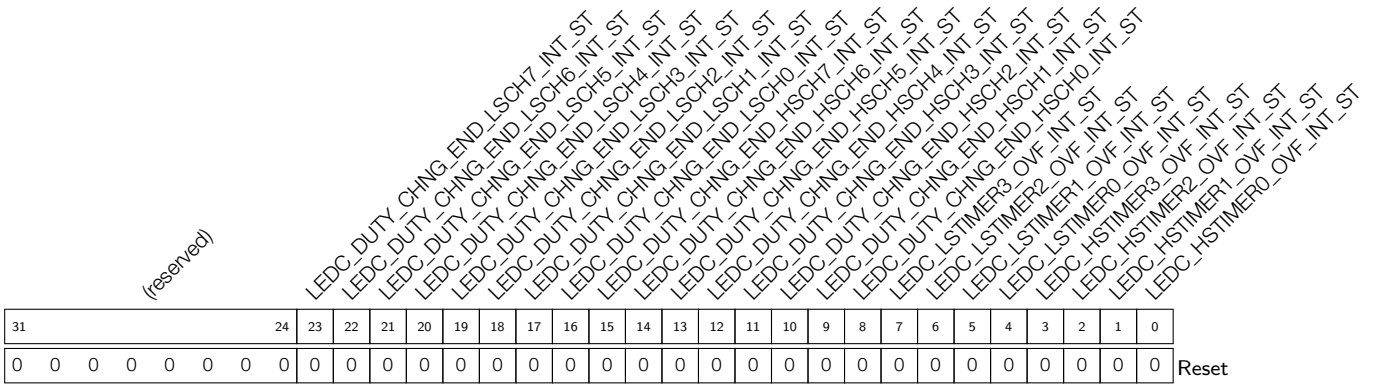
LEDC_DUTY_CHNG_END_LSCH*n*_INT_RAW The raw interrupt status bit for the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (RO)

LEDC_DUTY_CHNG_END_HSCH*n*_INT_RAW The raw interrupt status bit for the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (RO)

LEDC_LSTIMER*x*_OVF_INT_RAW The raw interrupt status bit for the LEDC_LSTIMER*x*_OVF_INT interrupt. (RO)

LEDC_HSTIMER*x*_OVF_INT_RAW The raw interrupt status bit for the LEDC_HSTIMER*x*_OVF_INT interrupt. (RO)

Register 14.16: LEDC_INT_ST_REG (0x0184)



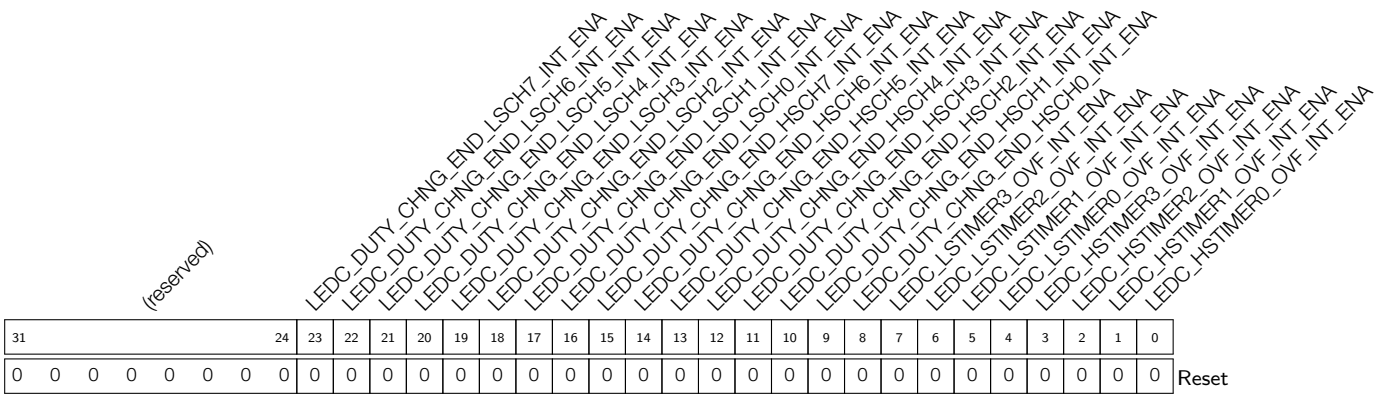
LEDC_DUTY_CHNG_END_LSCH*n*_INT_ST The masked interrupt status bit for the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (RO)

LEDC_DUTY_CHNG_END_HSCH*n*_INT_ST The masked interrupt status bit for the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (RO)

LEDC_LSTIMER*x*_OVF_INT_ST The masked interrupt status bit for the LEDC_LSTIMER*x*_OVF_INT interrupt. (RO)

LEDC_HSTIMER*x*_OVF_INT_ST The masked interrupt status bit for the LEDC_HSTIMER*x*_OVF_INT interrupt. (RO)

Register 14.17: LEDC_INT_ENA_REG (0x0188)



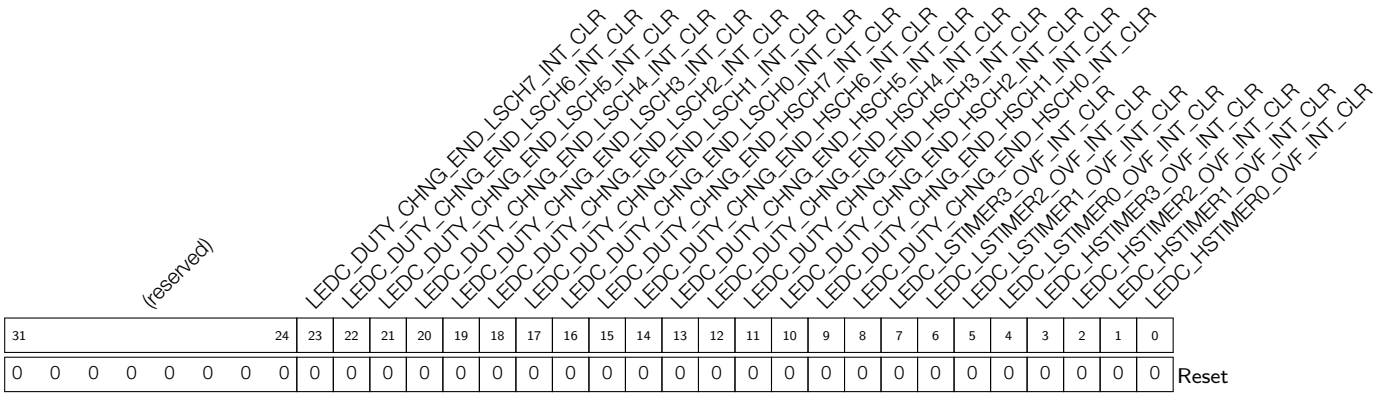
LEDC_DUTY_CHNG_END_LSCH*n*_INT_ENA The interrupt enable bit for the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (R/W)

LEDC_DUTY_CHNG_END_HSCH*n*_INT_ENA The interrupt enable bit for the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (R/W)

LEDC_LSTIMER*x*_OVF_INT_ENA The interrupt enable bit for the LEDC_LSTIMER*x*_OVF_INT interrupt. (R/W)

LEDC_HSTIMER*x*_OVF_INT_ENA The interrupt enable bit for the LEDC_HSTIMER*x*_OVF_INT interrupt. (R/W)

Register 14.18: LEDC_INT_CLR_REG (0x018C)



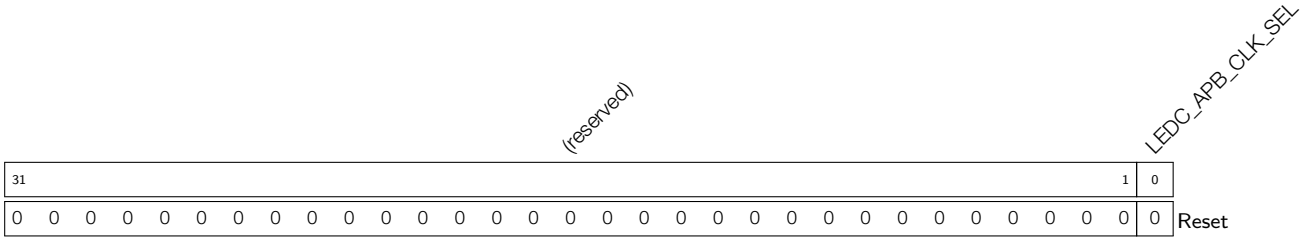
LEDC_DUTY_CHNG_END_LSCH*n*_INT_CLR Set this bit to clear the LEDC_DUTY_CHNG_END_LSCH*n*_INT interrupt. (WO)

LEDC_DUTY_CHNG_END_HSCH*n*_INT_CLR Set this bit to clear the LEDC_DUTY_CHNG_END_HSCH*n*_INT interrupt. (WO)

LEDC_LSTIMER*x*_OVF_INT_CLR Set this bit to clear the LEDC_LSTIMER*x*_OVF_INT interrupt. (WO)

LEDC_HSTIMER*x*_OVF_INT_CLR Set this bit to clear the LEDC_HSTIMER*x*_OVF_INT interrupt. (WO)

Register 14.19: LEDC_CONF_REG (0x0190)



LEDC_APB_CLK_SEL This bit is used to set the frequency of SLOW_CLK. (R/W)

0: 8 MHz;

1: 80 MHz.

15. Remote Control Peripheral

15.1 Introduction

The RMT (Remote Control) module is primarily designed to send and receive infrared remote control signals that implement on-off keying in a carrier frequency, but due to its design it can be used to generate various types of signals. An RMT transmitter does this by reading consecutive duration values of an active and inactive output from the built-in RAM block, optionally modulating it with a carrier wave. A receiver will inspect its input signal, optionally filtering it, and will place the lengths of time the signal is active and inactive in the RAM block.

The RMT module has eight channels, numbered zero to seven; registers, signals and blocks that are duplicated in each channel are indicated by an *n* which is used as a placeholder for the channel number.

15.2 Functional Description

15.2.1 RMT Architecture

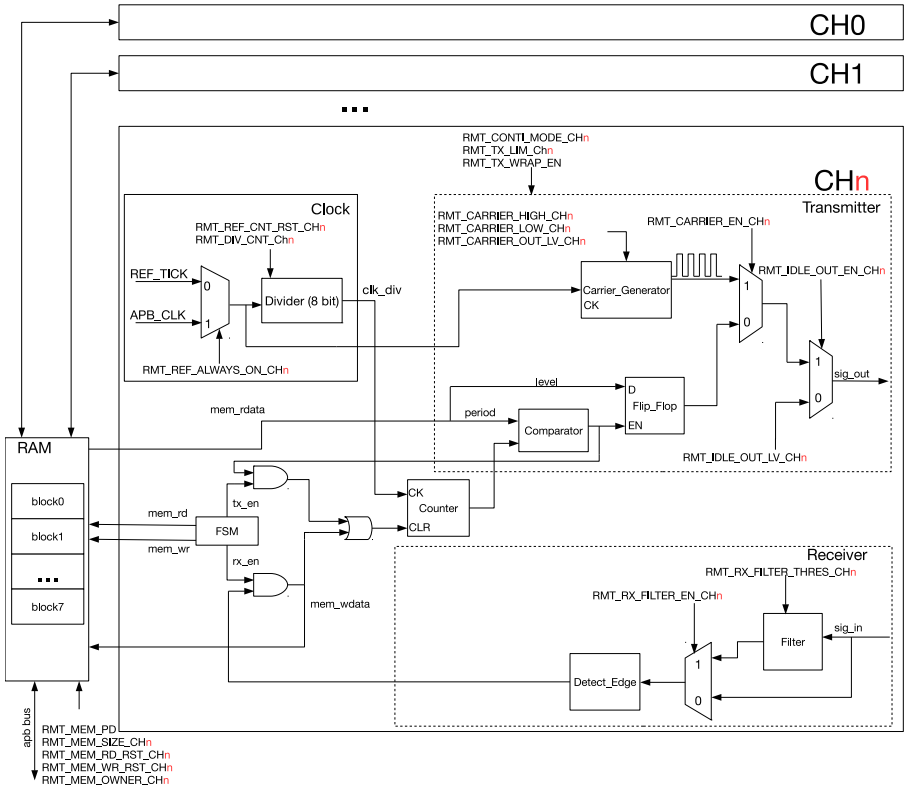


Figure 89: RMT Architecture

The RMT module contains eight channels. Each channel has both a transmitter and a receiver, but only one of them can be active in every channel. The eight channels share a 512x32-bit RAM block which can be read and written by the processor cores over the APB bus, read by the transmitters, and written by the receivers. The transmitted signal can optionally be modulated by a carrier wave. Each channel is clocked by a divided-down signal derived from either the APB bus clock or REF_TICK.

15.2.2 RMT RAM

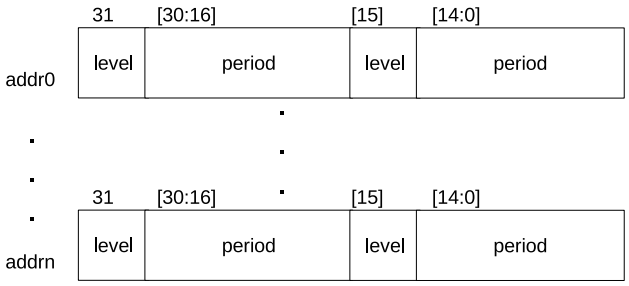


Figure 90: Data Structure

The data structure in RAM is shown in Figure 90. Each 32-bit value contains two 16-bit entries, with two fields in every entry, ”level” and ”period”. ”Level” indicates whether a high-/low-level value was received or is going to be sent, while ”period” points out the divider-clock cycles for which the level lasts. A zero period is interpreted as an end-marker: the transmitter will stop transmitting once it has read this, and the receiver will write this, once it has detected that the signal it received has gone idle.

Normally, only one block of 64x32-bit worth of data can be sent or received. If the data size is larger than this block size, blocks can be extended or the channel can be configured for the wraparound mode.

The RMT RAM can be accessed via the APB bus. The initial address is 0x3FF56800. The RAM block is divided into eight 64x32-bit blocks. By default, each channel uses one block (block zero for channel zero, block one for channel one, and so on). Users can extend the memory to a specific channel by configuring the RMT_MEM_SIZE_CH*n* register; setting this to >1 will prompt the channel to use the memory of subsequent channels as well. The RAM address range of channel *n* is start_addr_CH*n* to end_addr_CH*n*, which is defined by:

start_addr_ch$n$ = 0x3FF56800 + $64 * 4 * n$, and
end_addr_ch$n$ = 0x3FF56800 + $(64 * 4 * n + 64 * 4 *$ RMT_MEM_SIZE_CH$n$$)\mathrm{mod}(512 * 4) - 4$

To protect a receiver from overwriting the blocks a transmitter is about to transmit, RMT_MEM_OWNER_CH*n* can be configured to designate the owner, be it a transmitter or receiver, of channel *n*'s RAM block. This way, if this ownership is violated, the RMT_CH*n*_ERR interrupt will be generated.

Note: When enabling the continuous transmission mode by setting RMT_REG_TX_CONTI_MODE, the transmitter will transmit the data on the channel continuously, that is, from the first byte to the last one, then from the first to the last again, and so on. In this mode, there will be an idle level lasting one clk_div cycle between N and N+1 transmissions.

15.2.3 Clock

The main clock of a channel is generated by taking either the 80 MHz APB clock or REF_TICK (usually 1MHz), according to the state of RMT_REF_ALWAYS_ON_CH*n*. (For more information on clock sources, please see Chapter Reset And Clock.) Then, the aforementioned state gets scaled down using a configurable 8-bit divider to create the channel clock which is used by both the carrier wave generator and the counter. The divider value can be set by configuring RMT_DIV_CNT_CH*n*.

15.2.4 Transmitter

When the RMT_TX_START_CH*n* register is 1, the transmitter of channel *n* will start reading and sending data from RAM. The transmitter will receive a 32-bit value each time it reads from RAM. Of these 32 bits, the low 16-bit entry is sent first and the high entry second.

To transmit more data than can be fitted in the channel's RAM, the wraparound mode can be enabled. In this mode, when the transmitter has reached the last entry in the channel's memory, it will loop back to the first byte. To use this mechanism for sending more data than can be fitted in the channel's RAM, fill the RAM with the initial events and set RMT_CH*n*_TX_LIM_REG to cause an RMT_CH*n*_TX_THR_EVENT_INT interrupt before the wraparound happens. Then, when the interrupt happens, the already sent data should be replaced by subsequent events, so that when the wraparound happens the transmitter will seamlessly continue sending the new events.

With or without the wraparound mode enabled, transmission ends when an entry with zero length is encountered. When this happens, the transmitter will generate an RMT_CH*n*_TX_END_INT interrupt and return to the idle state. When a transmitter is in the idle state, the output level defaults to end-mark 0. Users can also configure RMT_IDLE_OUT_EN_CH*n* and RMT_IDLE_OUT_LV_CH*n* to control the output level manually.

The output of the transmitter can be modulated using a carrier wave by setting RMT_CARRIER_EN_CH*n*. The carrier frequency and duty cycle can be configured by adjusting the carrier's high and low durations in channel-clock cycles, in RMT_CARRIER_HIGH_CH*n* and RMT_CARRIER_HIGH_CH*n*.

15.2.5 Receiver

When RMT_RX_EN_CH*n* is set to 1, the receiver in channel *n* becomes active, measuring the duration between input signal edges. These will be written as period/level value pairs to the channel RAM in the same fashion as the transmitter sends them. Receiving ends, when the receiver detects no change in signal level for more than RMT_IDLE_THRES_CH*n* channel clock ticks. The receiver will write a final entry with 0 period, generate an RMT_CH*n*_RX_END_INT_RAW interrupt and return to the idle state.

The receiver has an input signal filter which can be configured using RMT_RX_FILTER_EN_CH*n*: The filter will remove pulses with a length of less than RMT_RX_FILTER_THRES_CH*n* in APB clock periods.

When the RMT module is inactive, the RAM can be put into low-power mode by setting the RMT_MEM_PD register to 1.

15.2.6 Interrupts

- RMT_CH*n*_TX_THR_EVENT_INT: Triggered when the amount of data the transmitter has sent matches the value of RMT_CH*n*_TX_LIM_REG.
- RMT_CH*n*_TX_END_INT: Triggered when the transmitter has finished transmitting the signal.
- RMT_CH*n*_RX_END_INT: Triggered when the receiver has finished receiving a signal.

15.3 Register Summary

Name	Description	Address	Access
Configuration registers			
RMT_CH0CONF0_REG	Channel 0 config register 0	0x3FF56020	R/W

RMT_CH0CONF1_REG	Channel 0 config register 1	0x3FF56024	R/W
RMT_CH1CONF0_REG	Channel 1 config register 0	0x3FF56028	R/W
RMT_CH1CONF1_REG	Channel 1 config register 1	0x3FF5602C	R/W
RMT_CH2CONF0_REG	Channel 2 config register 0	0x3FF56030	R/W
RMT_CH2CONF1_REG	Channel 2 config register 1	0x3FF56034	R/W
RMT_CH3CONF0_REG	Channel 3 config register 0	0x3FF56038	R/W
RMT_CH3CONF1_REG	Channel 3 config register 1	0x3FF5603C	R/W
RMT_CH4CONF0_REG	Channel 4 config register 0	0x3FF56040	R/W
RMT_CH4CONF1_REG	Channel 4 config register 1	0x3FF56044	R/W
RMT_CH5CONF0_REG	Channel 5 config register 0	0x3FF56048	R/W
RMT_CH5CONF1_REG	Channel 5 config register 1	0x3FF5604C	R/W
RMT_CH6CONF0_REG	Channel 6 config register 0	0x3FF56050	R/W
RMT_CH6CONF1_REG	Channel 6 config register 1	0x3FF56054	R/W
RMT_CH7CONF0_REG	Channel 7 config register 0	0x3FF56058	R/W
RMT_CH7CONF1_REG	Channel 7 config register 1	0x3FF5605C	R/W
Interrupt registers			
RMT_INT_RAW_REG	Raw interrupt status	0x3FF560A0	RO
RMT_INT_ST_REG	Masked interrupt status	0x3FF560A4	RO
RMT_INT_ENA_REG	Interrupt enable bits	0x3FF560A8	R/W
RMT_INT_CLR_REG	Interrupt clear bits	0x3FF560AC	WO
Carrier wave duty cycle registers			
RMT_CH0CARRIER_DUTY_REG	Channel 0 duty cycle configuration register	0x3FF560B0	R/W
RMT_CH1CARRIER_DUTY_REG	Channel 1 duty cycle configuration register	0x3FF560B4	R/W
RMT_CH2CARRIER_DUTY_REG	Channel 2 duty cycle configuration register	0x3FF560B8	R/W
RMT_CH3CARRIER_DUTY_REG	Channel 3 duty cycle configuration register	0x3FF560BC	R/W
RMT_CH4CARRIER_DUTY_REG	Channel 4 duty cycle configuration register	0x3FF560C0	R/W
RMT_CH5CARRIER_DUTY_REG	Channel 5 duty cycle configuration register	0x3FF560C4	R/W
RMT_CH6CARRIER_DUTY_REG	Channel 6 duty cycle configuration register	0x3FF560C8	R/W
RMT_CH7CARRIER_DUTY_REG	Channel 7 duty cycle configuration register	0x3FF560CC	R/W
Tx event configuration registers			
RMT_CH0_TX_LIM_REG	Channel 0 Tx event configuration register	0x3FF560D0	R/W
RMT_CH1_TX_LIM_REG	Channel 1 Tx event configuration register	0x3FF560D4	R/W
RMT_CH2_TX_LIM_REG	Channel 2 Tx event configuration register	0x3FF560D8	R/W
RMT_CH3_TX_LIM_REG	Channel 3 Tx event configuration register	0x3FF560DC	R/W
RMT_CH4_TX_LIM_REG	Channel 4 Tx event configuration register	0x3FF560E0	R/W
RMT_CH5_TX_LIM_REG	Channel 5 Tx event configuration register	0x3FF560E4	R/W
RMT_CH6_TX_LIM_REG	Channel 6 Tx event configuration register	0x3FF560E8	R/W
RMT_CH7_TX_LIM_REG	Channel 7 Tx event configuration register	0x3FF560EC	R/W
Other registers			
RMT_APB_CONF_REG	RMT-wide configuration register	0x3FF560F0	R/W

15.4 Registers

Register 15.1: RMT_CH*n*CONF0_REG (*n*: 0-7) (0x0020+8n*)**



RMT_MEM_PD This bit is used to power down the entire RMT RAM block. (It only exists in RMT_CH0CONF0). 1: power down memory; 0: power up memory. (R/W)

RMT_CARRIER_OUT_LV_CH*n* This bit is used for configuration when the carrier wave is being transmitted. Transmit on low output level with 1, and transmit on high output level with 0. (R/W)

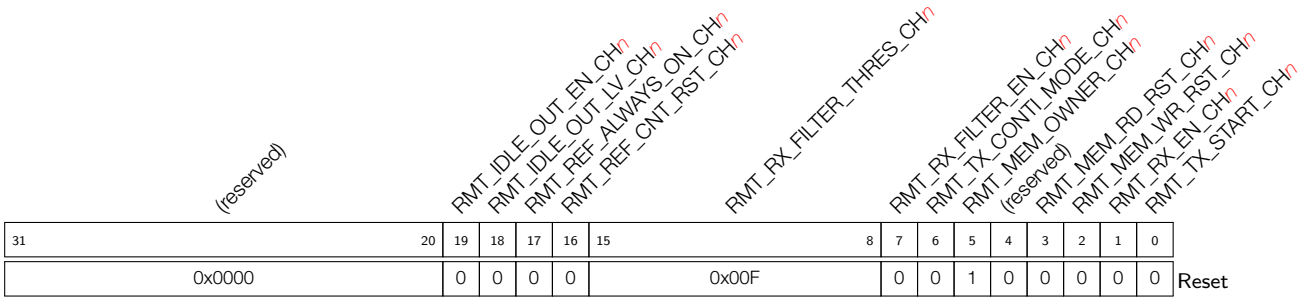
RMT_CARRIER_EN_CH*n* This is the carrier modulation enable-control bit for channel *n*. Carrier modulation is enabled with 1, while carrier modulation is disabled with 0. (R/W)

RMT_MEM_SIZE_CH*n* This register is used to configure the amount of memory blocks allocated to channel *n*. (R/W)

RMT_IDLE_THRES_CH*n* In receive mode, when no edge is detected on the input signal for longer than REG_IDLE_THRES_CH*n* channel clock cycles, the receive process is finished. (R/W)

RMT_DIV_CNT_CH*n* This register is used to set the divider for the channel clock of channel *n*. (R/W)

Register 15.2: RMT_CH*n*CONF1_REG (*n*: 0-7) (0x0024+8n*)**



RMT_IDLE_OUT_EN_CH*n* This is the output enable-control bit for channel *n* in IDLE state. (R/W)

RMT_IDLE_OUT_LV_CH*n* This bit configures the level of output signals in channel *n* when the latter is in IDLE state. (R/W)

RMT_REF_ALWAYS_ON_CH*n* This bit is used to select the channel's base clock. 1:clk_apb; 0:clk_ref. (R/W)

RMT_REF_CNT_RST_CH*n* Setting this bit resets the clock divider of channel *n*. (R/W)

RMT_RX_FILTER_THRES_CH*n* In receive mode, channel *n* ignores input pulse when the pulse width is smaller than this value in APB clock periods. (R/W)

RMT_RX_FILTER_EN_CH*n* This is the receive filter's enable-bit for channel *n*. (R/W)

RMT_TX_CONTI_MODE_CH*n* If this bit is set, instead of going to an idle state when transmission ends, the transmitter will restart transmission. This results in a repeating output signal. (R/W)

RMT_MEM_OWNER_CH*n* This bit marks channel *n*'s RAM block ownership. Number 1 indicates that the receiver is using the RAM, while 0 indicates that the transmitter is using the RAM. (R/W)

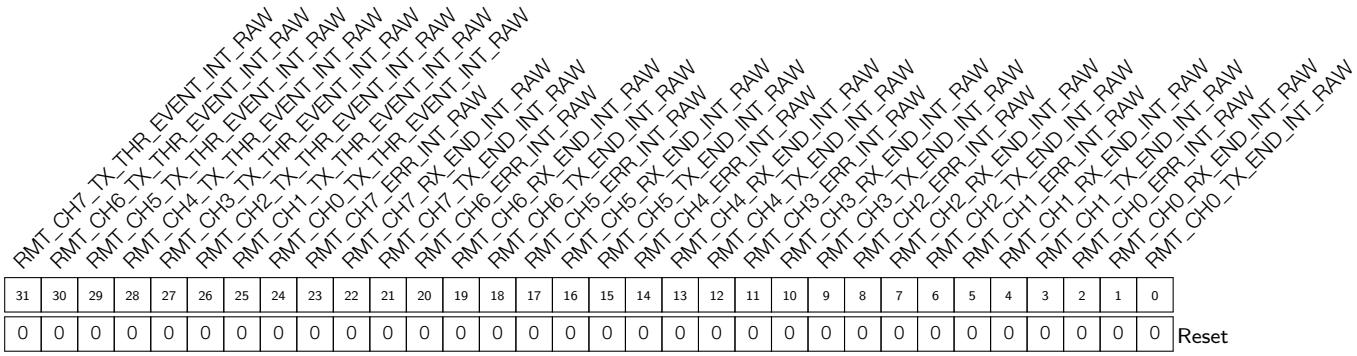
RMT_MEM_RD_RST_CH*n* Set this bit to reset the read-RAM address for channel *n* by accessing the transmitter. (R/W)

RMT_MEM_WR_RST_CH*n* Set this bit to reset the write-RAM address for channel *n* by accessing the receiver. (R/W)

RMT_RX_EN_CH*n* Set this bit to enable receiving data on channel *n*. (R/W)

RMT_TX_START_CH*n* Set this bit to start sending data on channel *n*. (R/W)

Register 15.3: RMT_INT_RAW_REG (0x00A0)



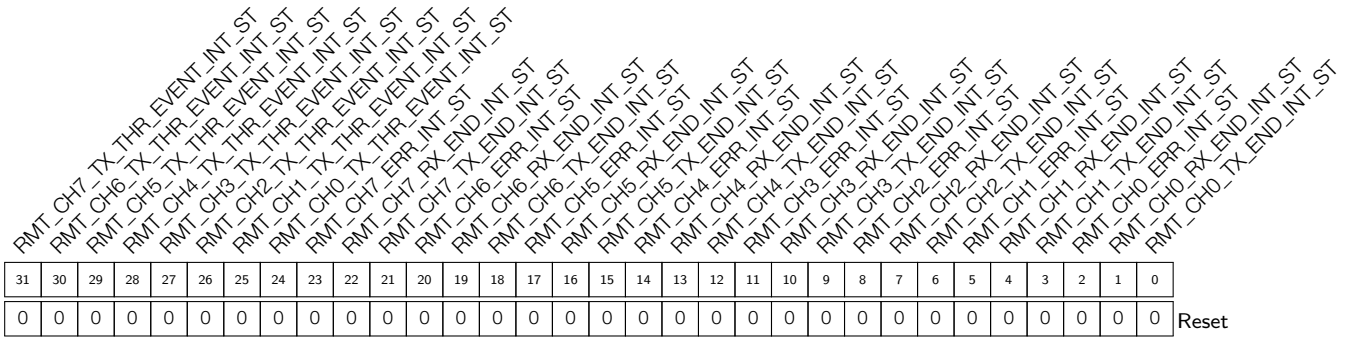
RMT_CH*n*_TX_THR_EVENT_INT_RAW The raw interrupt status bit for the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (RO)

RMT_CH*n*_ERR_INT_RAW The raw interrupt status bit for the RMT_CH*n*_ERR_INT interrupt. (RO)

RMT_CH*n*_RX_END_INT_RAW The raw interrupt status bit for the RMT_CH*n*_RX_END_INT interrupt. (RO)

RMT_CH*n*_TX_END_INT_RAW The raw interrupt status bit for the RMT_CH*n*_TX_END_INT interrupt. (RO)

Register 15.4: RMT_INT_ST_REG (0x00A4)



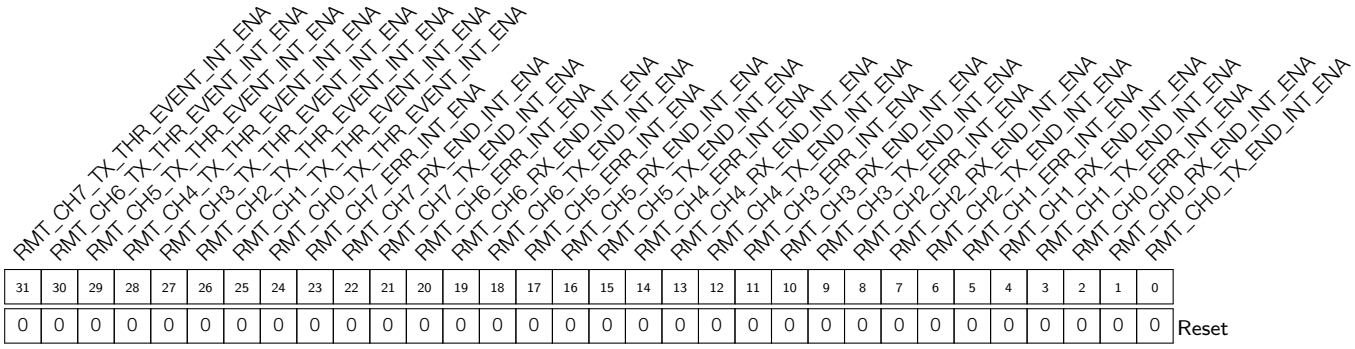
RMT_CH*n*_TX_THR_EVENT_INT_ST The masked interrupt status bit for the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (RO)

RMT_CH*n*_ERR_INT_ST The masked interrupt status bit for the RMT_CH*n*_ERR_INT interrupt. (RO)

RMT_CH*n*_RX_END_INT_ST The masked interrupt status bit for the RMT_CH*n*_RX_END_INT interrupt. (RO)

RMT_CH*n*_TX_END_INT_ST The masked interrupt status bit for the RMT_CH*n*_TX_END_INT interrupt. (RO)

Register 15.5: RMT_INT_ENA_REG (0x00A8)



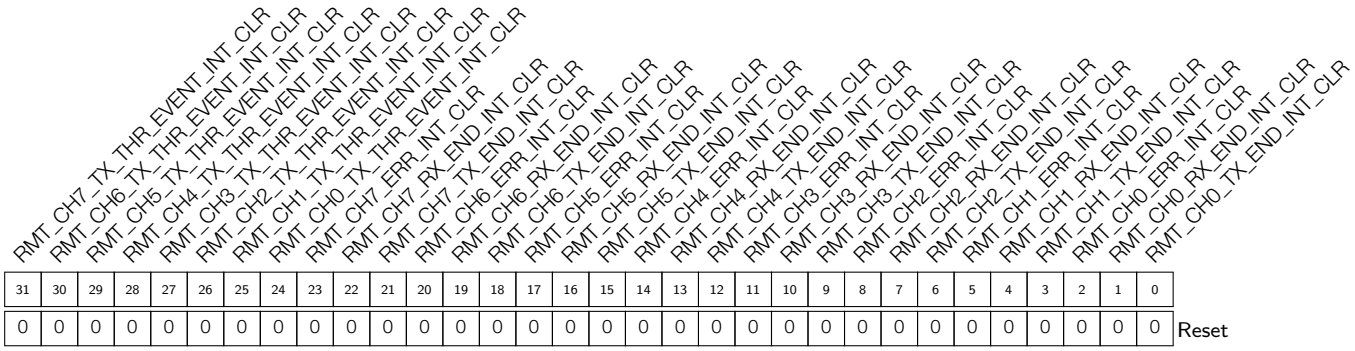
RMT_CH*n*_TX_THR_EVENT_INT_ENA The interrupt enable bit for the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (R/W)

RMT_CH*n*_ERR_INT_ENA The interrupt enable bit for the RMT_CH*n*_ERROR_INT interrupt. (R/W)

RMT_CH*n*_RX_END_INT_ENA The interrupt enable bit for the RMT_CH*n*_RX_END_INT interrupt. (R/W)

RMT_CH*n*_TX_END_INT_ENA The interrupt enable bit for the RMT_CH*n*_TX_END_INT interrupt. (R/W)

Register 15.6: RMT_INT_CLR_REG (0x00AC)



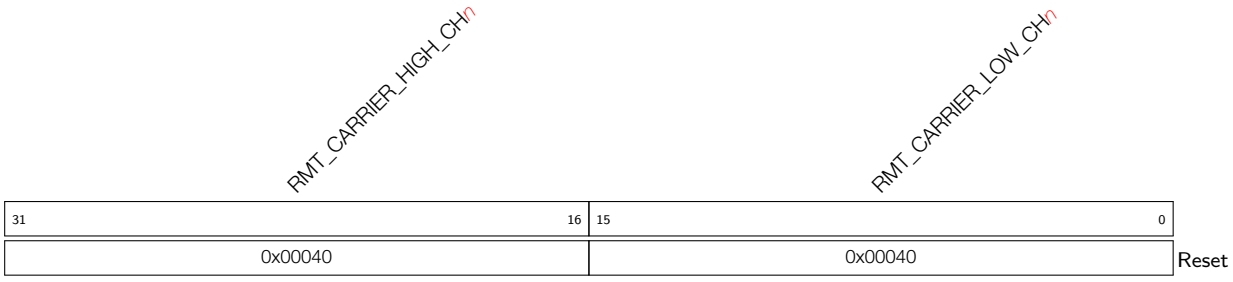
RMT_CH*n*_TX_THR_EVENT_INT_CLR Set this bit to clear the RMT_CH*n*_TX_THR_EVENT_INT interrupt. (WO)

RMT_CH*n*_ERR_INT_CLR Set this bit to clear the RMT_CH*n*_ERR_INT interrupt. (WO)

RMT_CH*n*_RX_END_INT_CLR Set this bit to clear the RMT_CH*n*_RX_END_INT interrupt. (WO)

RMT_CH*n*_TX_END_INT_CLR Set this bit to clear the RMT_CH*n*_TX_END_INT interrupt. (WO)

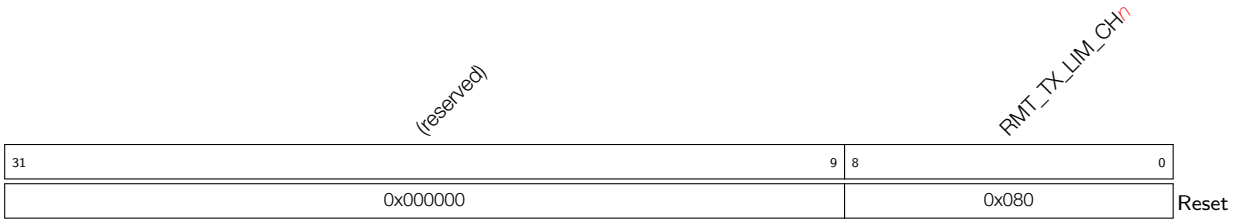
Register 15.7: RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-7) (0x00B0+4*n)



RMT_CARRIER_HIGH_CH*n* This field is used to configure the carrier wave's high-level clock period for channel *n*. The clock source can be either REF_TICK or APB_CLK. (R/W)

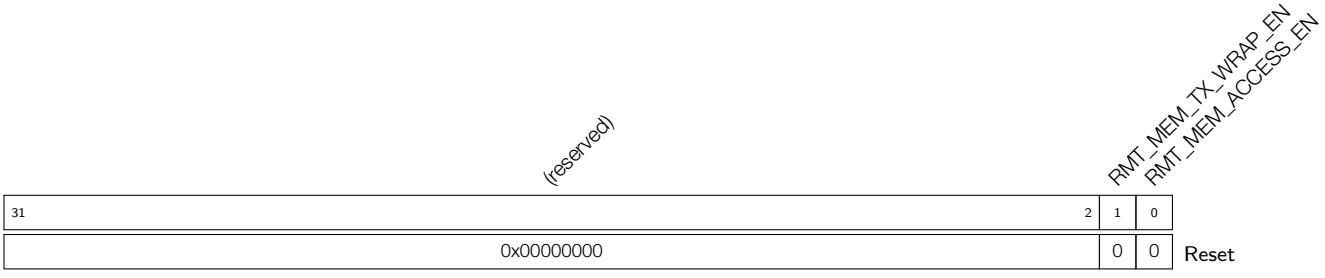
RMT_CARRIER_LOW_CH*n* This field is used to configure the carrier wave's low-level clock period for channel *n*. The clock source can be either REF_TICK or APB_CLK. (R/W)

Register 15.8: RMT_CH*n*_TX_LIM_REG (*n*: 0-7) (0x00D0+4*n)



RMT_TX_LIM_CH*n* When channel *n* sends more entries than specified here, it produces a TX_THR_EVENT interrupt. (R/W)

Register 15.9: RMT_APB_CONF_REG (0x00F0)



RMT_MEM_TX_WRAP_EN This bit enables wraparound mode: when the transmitter of a channel has reached the end of its memory block, it will resume sending at the start of its memory region. (R/W)

RMT_MEM_ACCESS_EN This bit must be 1 in order to access the RMT memory.

16. MCPWM

16.1 Introduction

The **M**otor **C**ontrol **P**ulse **W**idth **M**odulator (MCPWM) peripheral is intended for motor and power control. It provides six PWM outputs that can be set up to operate in several topologies. One common topology uses a pair of PWM outputs driving an H-bridge to control motor rotation speed and rotation direction.

The timing and control resources inside are allocated into two major types of submodules: PWM timers and PWM operators. Each PWM timer provides timing references that can either run freely or be synced to other timers or external sources. Each PWM operator has all necessary control resources to generate waveform pairs for one PWM channel. The MCPWM peripheral also contains a dedicated capture submodule that is used in systems where accurate timing of external events is important.

ESP32 contains two MCPWM peripherals: MCPWM0 and MCPWM1. Their control registers are located in 4-KB memory blocks starting at memory locations 0x3FF5E000 and 0x3FF6C000 respectively.

16.2 Features

Each MCPWM peripheral has one clock divider (prescaler), three PWM timers, three PWM operators, and a capture module. Figure 91 shows the submodules inside and the signals on the interface. PWM timers are used for generating timing references. The PWM operators generate desired waveform based on the timing references. Any PWM operator can be configured to use the timing references of any PWM timers. Different PWM operators can use the same PWM timer's timing references to produce related PWM signals. PWM operators can also use different PWM timers' values to produce the PWM signals that work alone. Different PWM timers can also be synced together.

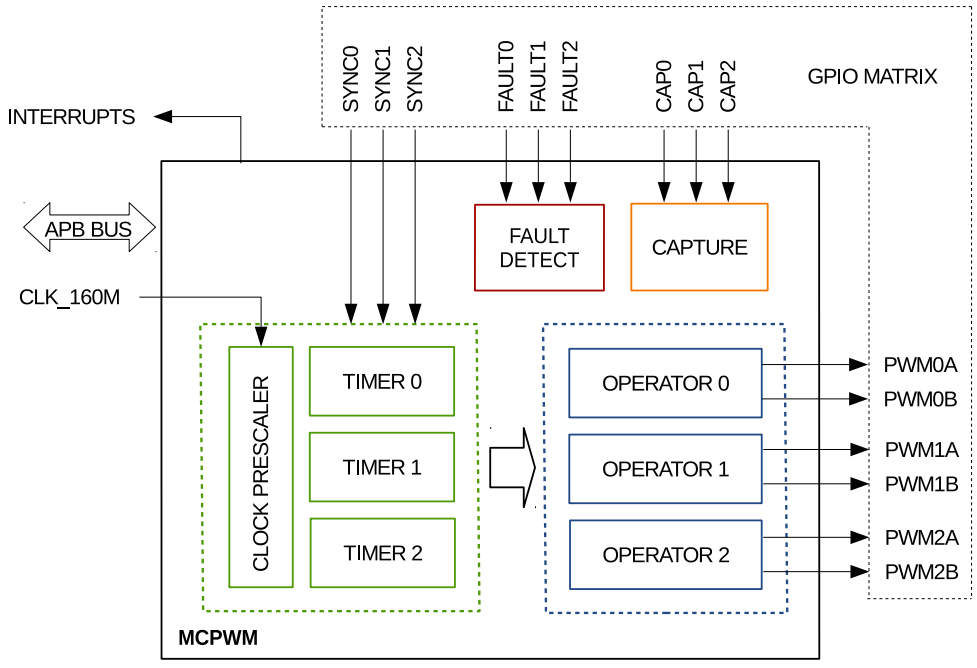


Figure 91: MCPWM Module Overview

An overview of the submodules' function in Figure 91 is shown below:

- PWM Timers 0, 1 and 2
  - Every PWM timer has a dedicated 8-bit clock prescaler.
  - The 16-bit counter in the PWM timer can work in count-up mode, count-down mode or count-up-down mode.
  - A hardware sync can trigger a reload on the PWM timer with a phase register. It will also trigger the prescaler' restart, so that the timer's clock can also be synced. The source of the sync can come from any GPIO or any other PWM timer's sync_out.
- PWM Operators 0, 1 and 2
  - Every PWM operator has two PWM outputs: PWM*x*A and PWM*x*B. They can work independently, in symmetric and asymmetric configuration.
  - Software, asynchronous override control of PWM signals.
  - Configurable dead-time on rising and falling edges; each set up independently.
  - All events can trigger CPU interrupts.
  - Modulating of PWM output by high-frequency carrier signals, useful when gate drives are insulated with a transformer.
  - Period, time stamps and important control registers have shadow registers with flexible updating methods.
- Fault Detection Module
  - Programmable fault handling allocated on fault condition in both cycle-by-cycle mode and one-shot mode.
  - A fault condition can force the PWM output to either high or low logic levels.
- Capture Module
  - Speed measurement of rotating machinery (for example, toothed sprockets sensed with Hall sensors)
  - Measurement of elapsed time between position sensor pulses
  - Period and duty-cycle measurement of pulse train signals
  - Decoding current or voltage amplitude derived from duty-cycle-encoded signals from current/voltage sensors
  - Three individual capture channels, each of which has a time-stamp register (32 bits)
  - Selection of edge polarity and prescaling of input capture signal
  - The capture timer can sync with a PWM timer or external signals.
  - Interrupt on each of the three capture channels

16.3 Submodules

16.3.1 Overview

This section lists the configuration parameters of key submodules. For information on adjusting a specific parameter, e.g. synchronization source of PWM timer, please refer to Section 16.3.2 for details.

16.3.1.1 Prescaler Submodule



Figure 92: Prescaler Submodule

Configuration parameter:

- Scale the PWM clock according to CLK_160M.

16.3.1.2 Timer Submodule

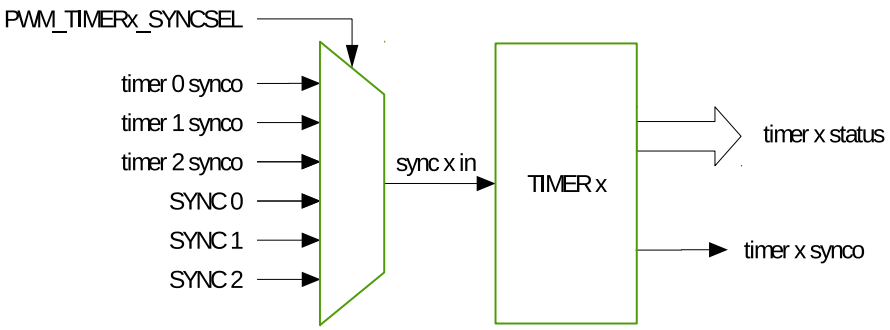


Figure 93: Timer Submodule

Configuration parameters:

- Set the PWM timer frequency or period.
- Configure the working mode for the timer:
  - Count-Up Mode: for asymmetric PWM outputs
  - Count-Down Mode: for asymmetric PWM outputs
  - Count-Up-Down Mode: for symmetric PWM outputs
- Configure the the reloading phase (including the value and the phase) used during software and hardware synchronization.

- Synchronize the PWM timers with each other. Either hardware or software synchronization may be used.
- Configure the source of the PWM timer's the synchronization input to one of the seven sources below:
  - The three PWM timer's synchronization outputs.
  - Three synchronization signals from the GPIO matrix: SYNC0, SYNC1, SYNC2.
  - No synchronization input signal selected
- Configure the source of the PWM timer's synchronization output to one of the four sources below:
  - Synchronization input signal
  - Event generated when value of the PWM timer is equal to zero
  - Event generated when value of the PWM timer is equal to period
  - No synchronization output generated
- Configure the method of period updating.

16.3.1.3 Operator Submodule

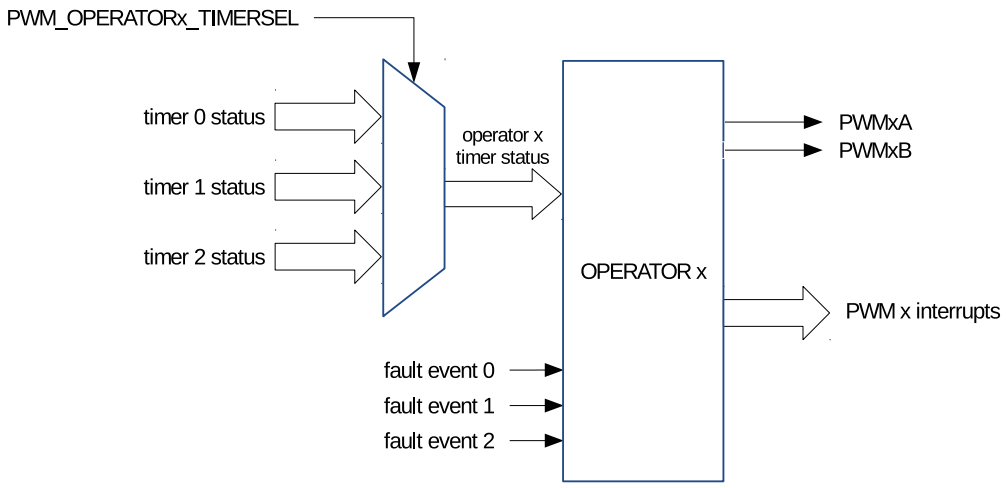


Figure 94: Operator Submodule

The configuration parameters of the operator submodule are shown in Table 68.

Table 68: Configuration Parameters of the Operator Submodule

Submodule	Configuration Parameter or Option
PWM Generator	• Set up the PWM duty cycle for PWM*x*A and/or PWM*x*B output. • Set up at which time the timing events occur. • Define what action should be taken on timing events: – Switch high or low PWM*x*A and/or PWM*x*B outputs – Toggle PWM*x*A and/or PWM*x*B outputs – Take no action on outputs • Use direct s/w control to force the state of PWM outputs • Add a dead time to raising and / or failing edge on PWM outputs. • Configure update method for this submodule.
Dead Time Generator	• Control of complementary dead time relationship between upper and lower switches. • Specify the dead time on rising edge. • Specify the dead time on falling edge. • Bypass the dead time generator module. The PWM waveform will pass through without inserting dead time. • Allow PWM*x*B phase shifting with respect to the PWM*x*A output. • Configure updating method for this submodule.
PWM Carrier	• Enable carrier and set up carrier frequency. • Configure duration of the first pulse in the carrier waveform. • Set up the duty cycle of the following pulses. • Bypass the PWM carrier module. The PWM waveform will be passed through without modification.
Fault Handler	• Configure if and how the PWM module should react the fault event signals. • Specify the action taken when a fault event occurs: – Force PWM*x*A and/or PWM*x*B high. – Force PWM*x*A and/or PWM*x*B low. – Configure PWM*x*A and/or PWM*x*B to ignore any fault event. • Configure how often the PWM should react to fault events: – One-shot – Cycle-by-cycle • Generate interrupts. • Bypass the fault handler submodule entirely. • Set up an option for cycle-by-cycle actions clearing. • If desired, independently-configured actions can be taken when time-base counter is counting down or up.

16.3.1.4 Fault Detection Submodule

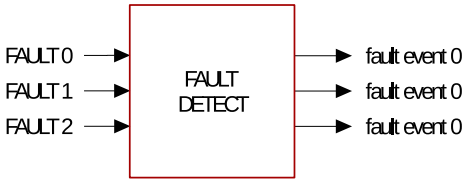


Figure 95: Fault Detection Submodule

Configuration parameters:

- Enable fault event generation and configure the polarity of fault event generation for every fault signal
- Generate fault event interrupts

16.3.1.5 Capture Submodule

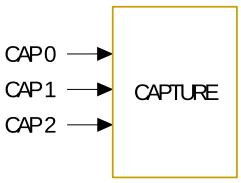


Figure 96: Capture Submodule

Configuration parameters:

- Select the edge polarity and prescaling of the capture input.
- Set up a software-triggered capture.
- Configure the capture timer's sync trigger and sync phase.
- Software syncs the capture timer.

16.3.2 PWM Timer Submodule

Each MCPWM module has three PWM timer submodules. Any of them can determine the necessary event timing for any of the three PWM operator submodules. Built-in synchronization logic allows multiple PWM timer submodules, in one or more MCPWM modules, to work together as a system, when using synchronization signals from the GPIO matrix.

16.3.2.1 Configurations of the PWM Timer Submodule

Users can configure the following functions of the PWM timer submodule:

- Control how often events occur by specifying the PWM timer frequency or period.

- Configure a particular PWM timer to synchronize with other PWM timers or modules.
- Get a PWM timer in phase with other PWM timers or modules.
- Set one of the following timer counting modes: count-up, count-down, count-up-down.
- Change the rate of the PWM timer clock (PT_clk) with a prescaler. Each timer has its own prescaler configured with PWM_TIMER*x*_PRESCALE of register PWM_TIMER0_CFG0_REG. The PWM timer increments or decrements at a slower pace, depending on the setting of this register.

16.3.2.2 PWM Timer's Working Modes and Timing Event Generation

The PWM timer has three working modes, selected by the PWM*x* timer mode register:

- Count-Up Mode:
  In this mode, the PWM timer increments from zero until reaching the value configured in the period register. Once done, the PWM timer returns to zero and starts increasing again. PWM period is equal to the period value configured in register.
- Count-Down Mode:
  The PWM timer decrements to zero, starting from the value configured in the period register. After reaching zero, it is set back to the period value. Then it starts to decrement again. In this case, the PWM period is also equal to the value configured in the period register.
- Count-Up-Down Mode:
  This is a combination of the two modes mentioned above. The PWM timer starts increasing from zero until the period value is reached. Then, the timer decreases back to zero. This pattern is then repeated. The PWM period is the result of the value in the period register multiplied by 2.

Figures 97 to 100 show PWM timer waveforms in different modes, including timer behavior during synchronization events.

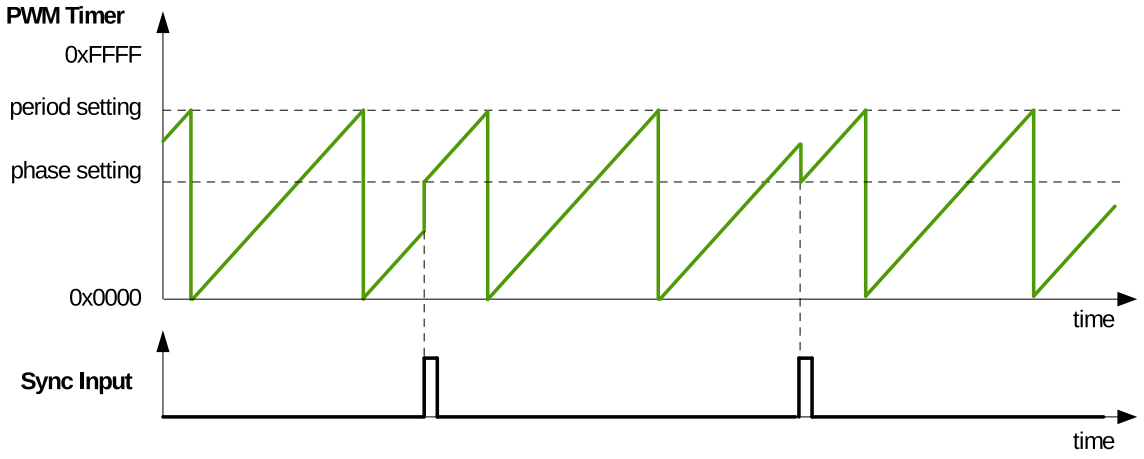


Figure 97: Count-Up Mode Waveform

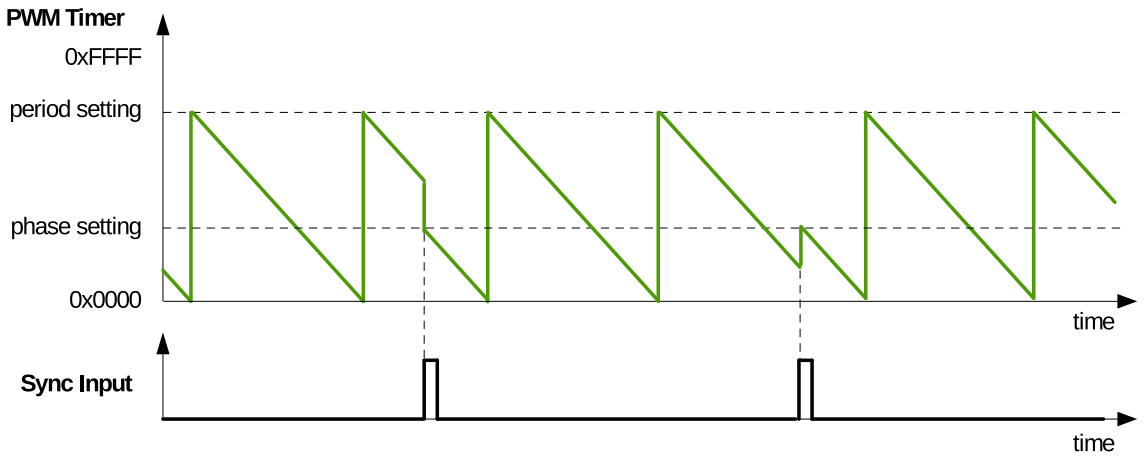


Figure 98: Count-Down Mode Waveforms

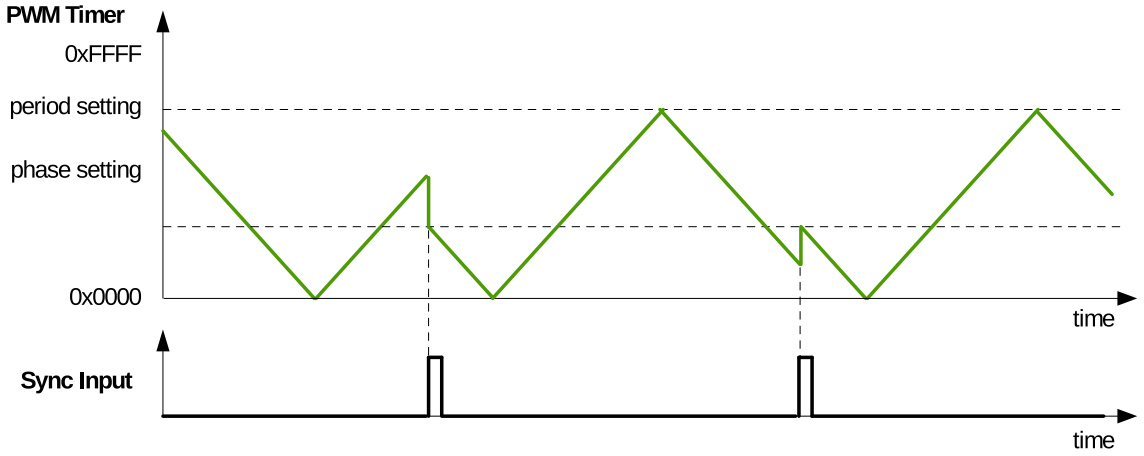


Figure 99: Count-Up-Down Mode Waveforms, Count-Down at Synchronization Event

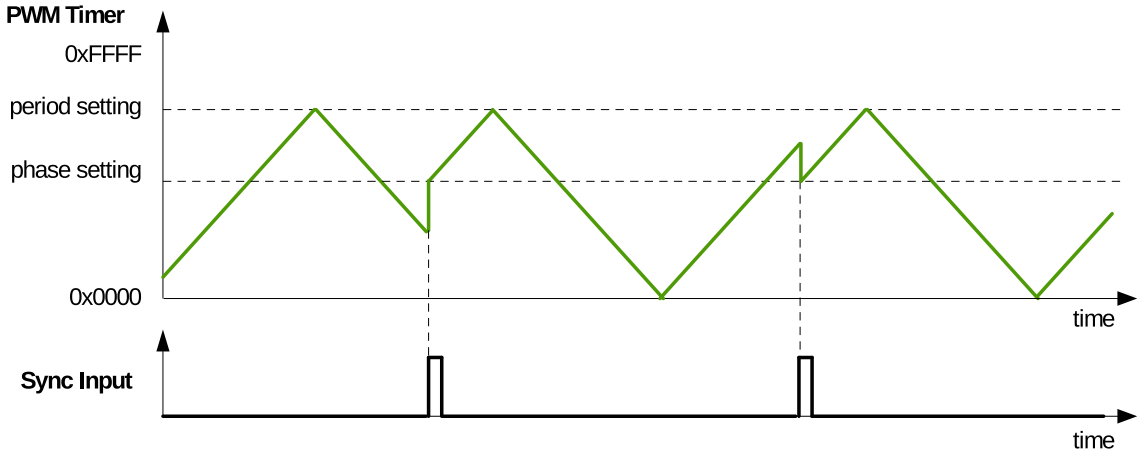


Figure 100: Count-Up-Down Mode Waveforms, Count-Up at Synchronization Event

When the PWM timer is running, it generates the following timing events periodically and automatically:

- UTEP
  The timing event generated when the PWM timer's value equals to the value of the period register (PWM_TIMER*x*_PERIOD) and when the PWM timer is increasing.
- UTEZ
  The timing event generated when the PWM timer's value equals to zero and when the PWM timer is increasing.
- DTEP
  The timing event generated when the PWM timer's value equals to the value of the period register (PWM_TIMER*x*_PERIOD) and when the PWM timer is decreasing.
- DTEZ
  The timing event generated when the PWM timer's value equals to zero and when the PWM timer is decreasing.

Figures 101 to 103 show the timing waveforms of U/DTEP and U/DTEZ.

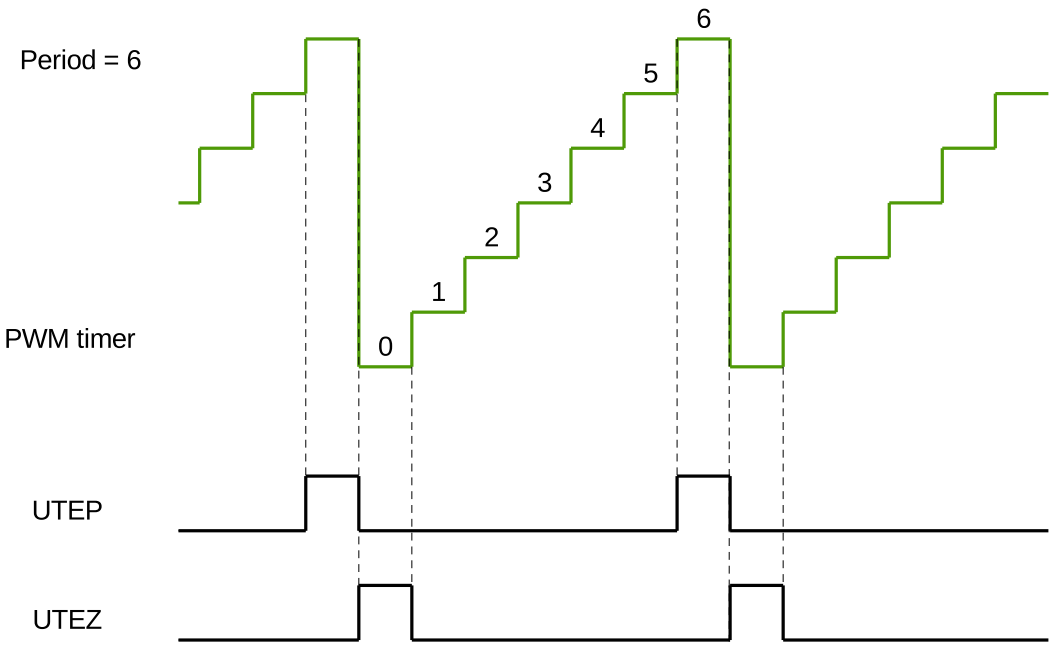


Figure 101: UTEP and UTEZ Generation in Count-Up Mode

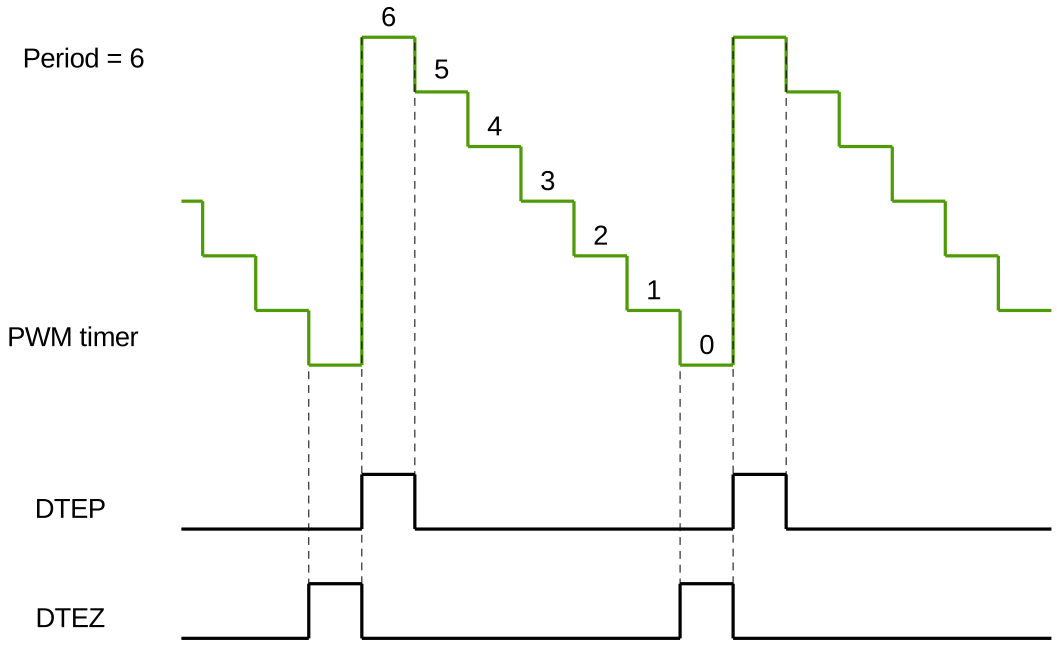


Figure 102: DTEP and DTEZ Generation in Count-Down Mode

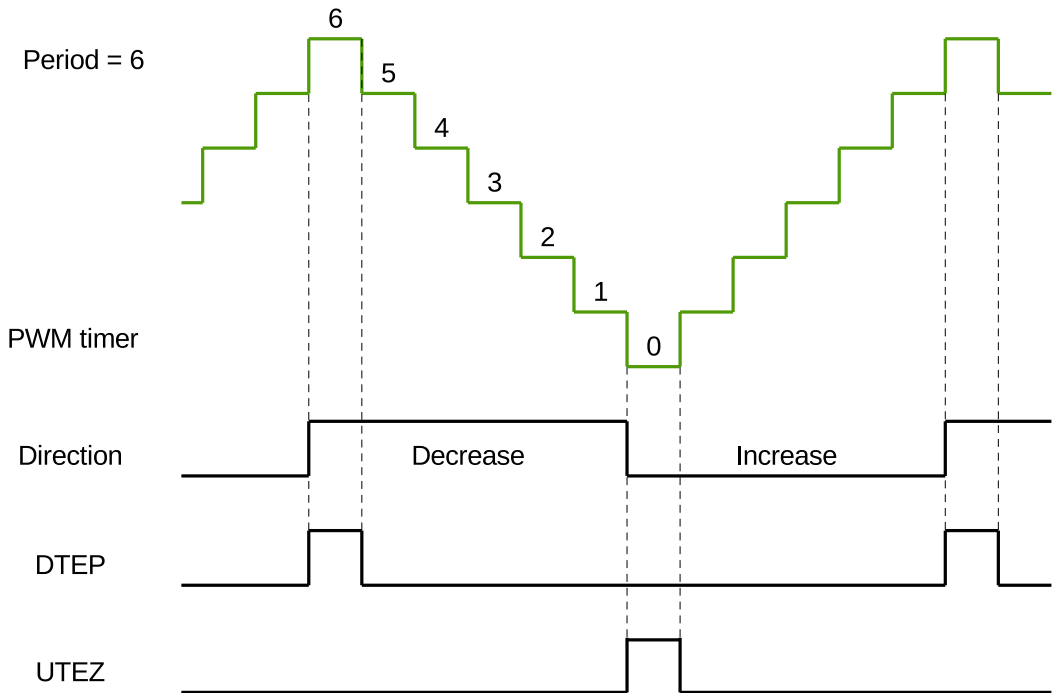


Figure 103: DTEP and UTEZ Generation in Count-Up-Down Mode

16.3.2.3 PWM Timer Shadow Register

The PWM timer's period register and the PWM timer's clock prescaler register have shadow registers. The purpose of a shadow register is to save a copy of the value to be written into the active register at a specific moment synchronized with the hardware. Both register types are defined as follows:

- Active Register
  This register is directly responsible for controlling all actions performed by hardware.
- Shadow Register
  It acts as a temporary buffer for a value to be written on the active register. Before this happens, the content of the shadow register has no direct effect on the controlled hardware. At a specific, user-configured point in time, the value saved in the shadow register is copied to the active register. This helps to prevent spurious operation of the hardware, which may happen when a register is asynchronously modified by software. Both the shadow register and the active register have the same memory address. The software always writes into, or reads from the shadow register. The moment of updating the active register is determined by its specific update method register. The update can start when the PWM timer is equal to zero, when the PWM timer is equal to period,at a synchronization moment, or immediately. Software can trigger a globally forced update which will prompt all registers in the module to be updated according to shadow registers.

16.3.2.4 PWM Timer Synchronization and Phase Locking

The PWM modules adopt a flexible synchronization method. Each PWM timer has a synchronization input and a synchronization output. The synchronization input can be selected from three synchronization outputs and three synchronization signals from the GPIO matrix. The synchronization output can be generated from the synchronization input signal, or when the PWM timer's value is equal to period or zero. Thus, the PWM timers can be chained together with their phase locked. During synchronization, the PWM timer clock prescaler will reset its counter in order to synchronize the PWM timer clock.

16.3.3 PWM Operator Submodule

The PWM Operator submodule has the following functions:

- Generates a PWM signal pair, based on timing references obtained from the corresponding PWM timer.
- Each signal out of the PWM signal pair includes a specific pattern of dead time.
- Superimposes a carrier on the PWM signal, if configured to do so.
- Handles response under fault conditions.

Figure 104 shows the block diagram of a PWM operator.

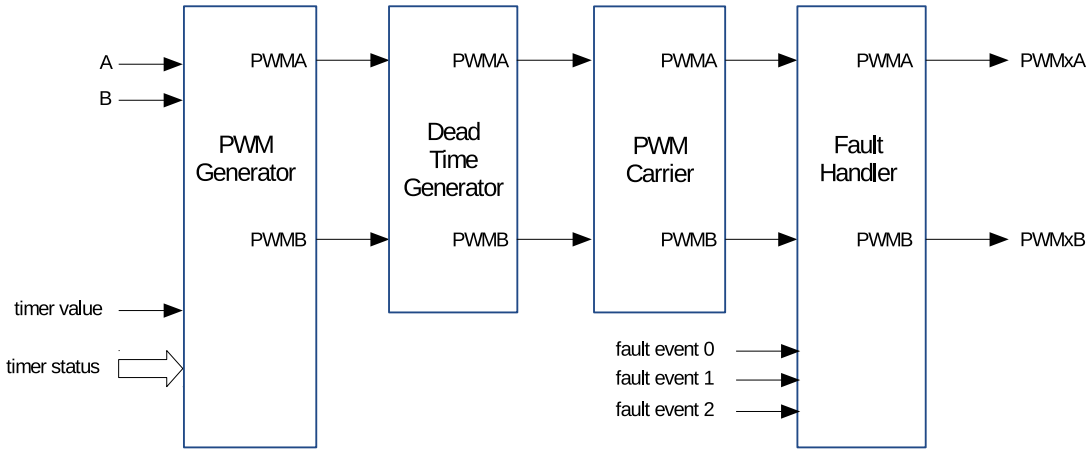


Figure 104: Submodules Inside the PWM Operator

16.3.3.1 PWM Generator Submodule

Purpose of the PWM Generator Submodule

In this submodule, important timing events are generated or imported. The events are then converted into specific actions to generate the desired waveforms at the PWM*x*A and PWM*x*B outputs.

The PWM generator submodule performs the following actions:

- Generation of timing events based on time stamps configured using the A and B registers. Events happen when the following conditions are satisfied:
  - UTEA: the PWM timer is counting up and its value is equal to register A.
  - UTEB: the PWM timer is counting up and its value is equal to register B.
  - DTEA: the PWM timer is counting down and its value is equal to register A.
  - DTEB: the PWM timer is counting down and its value is equal to register B.
- Generation of U/DT1, U/DT2 timing events based on fault or synchronization events.
- Management of priority when these timing events occur concurrently.
- Qualification and generation of set, clear and toggle actions, based on the timing events.
- Controlling of the PWM duty cycle, depending on configuration of the PWM generator submodule.
- Handling of new time stamp values, using shadow, registers to prevent glitches in the PWM cycle.

PWM Operator Shadow Registers

The time stamp registers A and B, as well as action configuration registers PWM_GEN*x*_A_REG and PWM_GEN*x*_B_REG are shadowed. Shadowing provides a way of updating registers in sync with the hardware. For a description of the shadow registers, please see 16.3.2.3.

Timing Events

For convenience, all timing signals and events are summarized in Table 69.

Table 69: Timing Events Used in PWM Generator

Signal	Event Description	PWM Timer Operation
DTEP	PWM timer value is equal to the period register value	PWM timer counts down.
DTEZ	PWM timer value is equal to zero	
DTEA	PWM timer value is equal to A register	
DTEB	PWM timer value is equal to B register	
DT0 event	Based on fault or synchronization events	
DT1 event	Based on fault or synchronization events	
UTEP	PWM timer value is equal to the period register value	PWM timer counts up.
UTEZ	PWM timer value is equal to zero	
UTEA	PWM timer value is equal to A register	
UTEB	PWM timer value is equal to B register	
UT0 event	Based on fault or synchronization events	
UT1 event	Based on fault or synchronization events	
Software-force event	Software-initiated asynchronous event	N/A

The purpose of a software-force event is to impose non-continuous or continuous changes on the PWM*x*A and PWM*x*B outputs. The change is done asynchronously. Software-force control is handled by the PWM_PWM_GEN*x*_FORCE_REG registers.

The selection and configuration of T0/T1 in the PWM generator submodule is independent of the configuration of fault events in the fault handler submodule. A particular trip event may or may not be configured to cause trip action in the fault handler submodule, but the same event can be used by the PWM generator to trigger T0/T1 for controlling PWM waveforms.

It is important to know that when the PWM timer is in count-up-down mode, it will always decrement after a TEP event, and will always increment after a TEZ event. So when the PWM timer is in count-up-down mode, DTEP and UTEZ events will occur, while the events UTEP and DTEZ will never occur.

The PWM generator can handle multiple events at the same time. Events are prioritized by the hardware and relevant details are provided in Table 70 and Table 71. Priority levels range from 1 (the highest) to 7 (the lowest). Please note that the priority of TEP and TEZ events depends on the PWM timer's direction.

If the value of A or B is set to be greater than the period, then U/DTEA and U/DTEB will never occur.

Table 70: Timing Events Priority When PWM Timer Increments

Priority Level	Event
1 (highest)	Software-force event
2	UTEP
3	UT0
4	UT1
5	UTEB
6	UTEA
7 (lowest)	UTEZ

Table 71: Timing Events Priority when PWM Timer Decrements

Priority level	Event
1 (highest)	Software-force event
2	DTEZ
3	DT0
4	DT1
5	DTEB
6	DTEA
7 (lowest)	DTEP

Notes:

1. UTEP and UTEZ do not happen simultaneously. When the PWM timer is in count-up mode, UTEP will always happen one cycle earlier than UTEZ, as demonstrated in Figure 101, so their action on PWM signals will not interrupt each other. When the PWM timer is in count-up-down mode, UTEP will not occur.
2. DTEP and DTEZ do not happen simultaneously. When the PWM timer is in count-down mode, DTEZ will always happen one cycle earlier than DTEP, as demonstrated in Figure 102, so their action on PWM signals will not interrupt each other. When the PWM timer is in count-up-down mode, DTEZ will not occur.

PWM Signal Generation

The PWM generator submodule controls the behavior of outputs PWM*x*A and PWM*x*B when a particular timing event occurs. The timing events are further qualified by the PWM timer's counting direction (up or down). Knowing the counting direction, the submodule may then perform an independent action at each stage of the PWM timer counting up or down.

The following actions may be configured on outputs PWM*x*A and PWM*x*B:

- Set High:
  Set the output of PWM*x*A or PWM*x*B to a high level.
- Clear Low:
  Clear the output of PWM*x*A or PWM*x*B by setting it to a low level.
- Toggle:
  Change the current output level of PWM*x*A or PWM*x*B to the opposite value. If it is currently pulled high, pull it low, or vice versa.
- Do Nothing:
  Keep both outputs PWM*x*A and PWM*x*B unchanged. In this state, interrupts can still be triggered.

The configuration of actions on outputs is done by using registers PWN_GEN*x*_A_REG and PWN_GEN*x*_B_REG. So, the action to be taken on each output is set independently. Also there is great flexibility in selecting actions to be taken on a given output based on events. More specifically, any event listed in Table 69 can operate on either output PWM*x*A or PWM*x*B. To check out registers for particular generator 0, 1 or 2, please refer to register description in Section 16.4.

Waveforms for Common Configurations

Figure 105 presents the symmetric PWM waveform generated when the PWM timer is counting up and down. DC 0%–100% modulation can be calculated via the formula below:

$$Duty = (Period - A) \div Period$$

If A matches the PWM timer value and the PWM timer is incrementing, then the PWM output is pulled up. If A matches the PWM timer value while the PWM timer is decrementing, then the PWM output is pulled low.

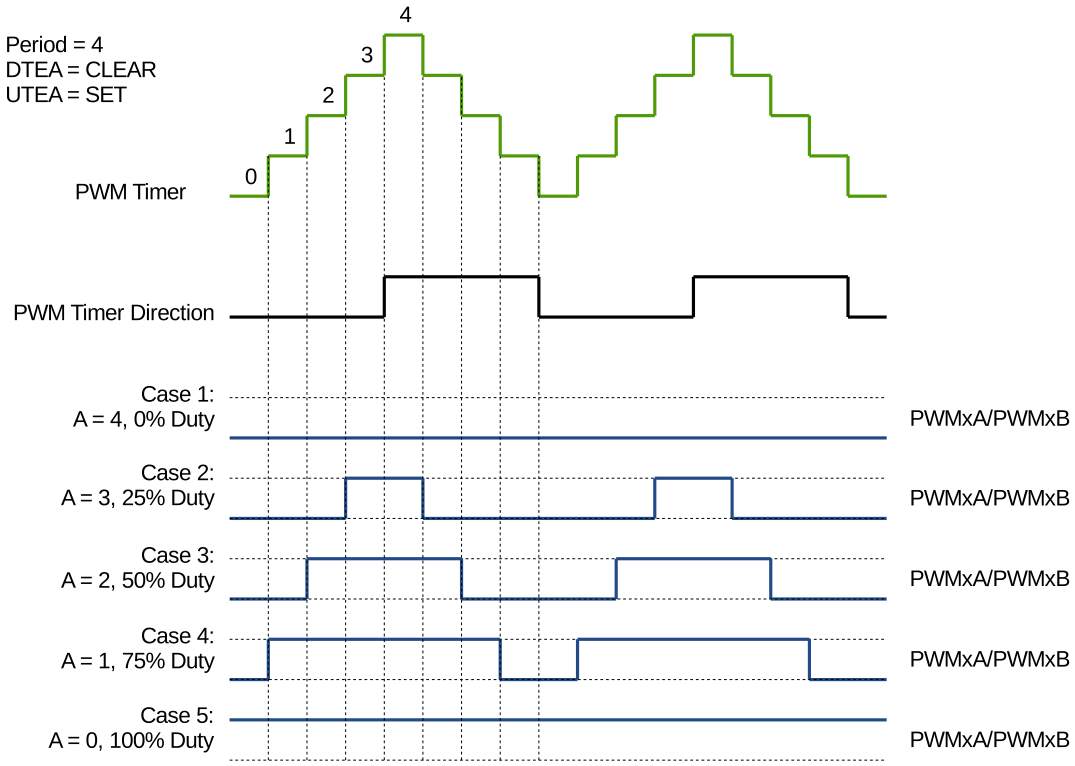


Figure 105: Symmetrical Waveform in Count-Up-Down Mode

The PWM waveforms in Figures 106 to 109 show some common PWM operator configurations. The following conventions are used in the figures:

- Period A and B refer to the values written in the corresponding registers.
- PWM*x*A and PWM*x*B are the output signals of PWM Operator *x*.

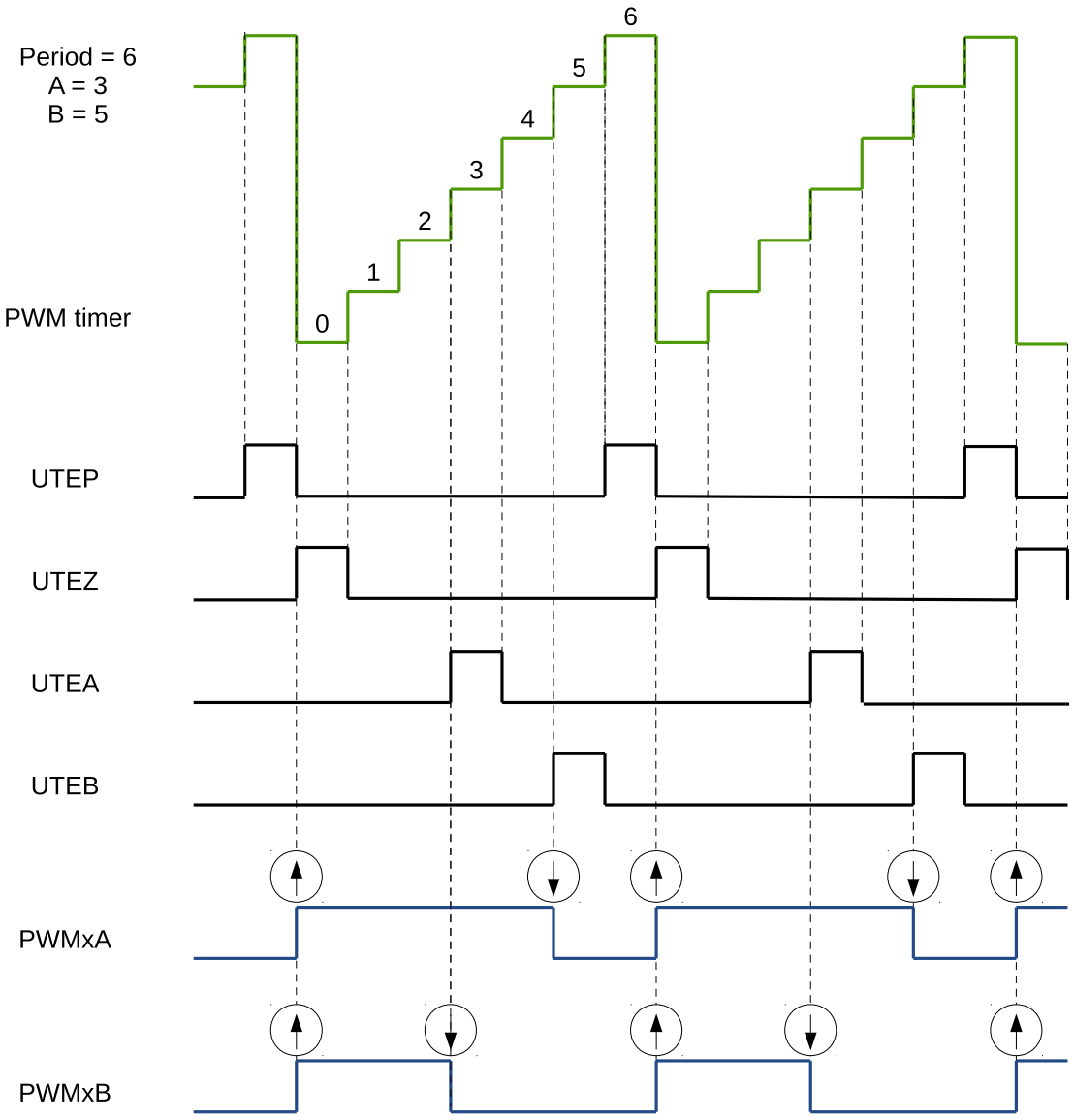


Figure 106: Count-Up, Single Edge Asymmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High

The duty modulation for PWM*x*A is set by B, active high and proportional to B.
The duty modulation for PWM*x*B is set by A, active high and proportional to A.

$$Period = (PWM\_TIMERx\_PERIOD + 1) \times T_{PT\_clk}$$

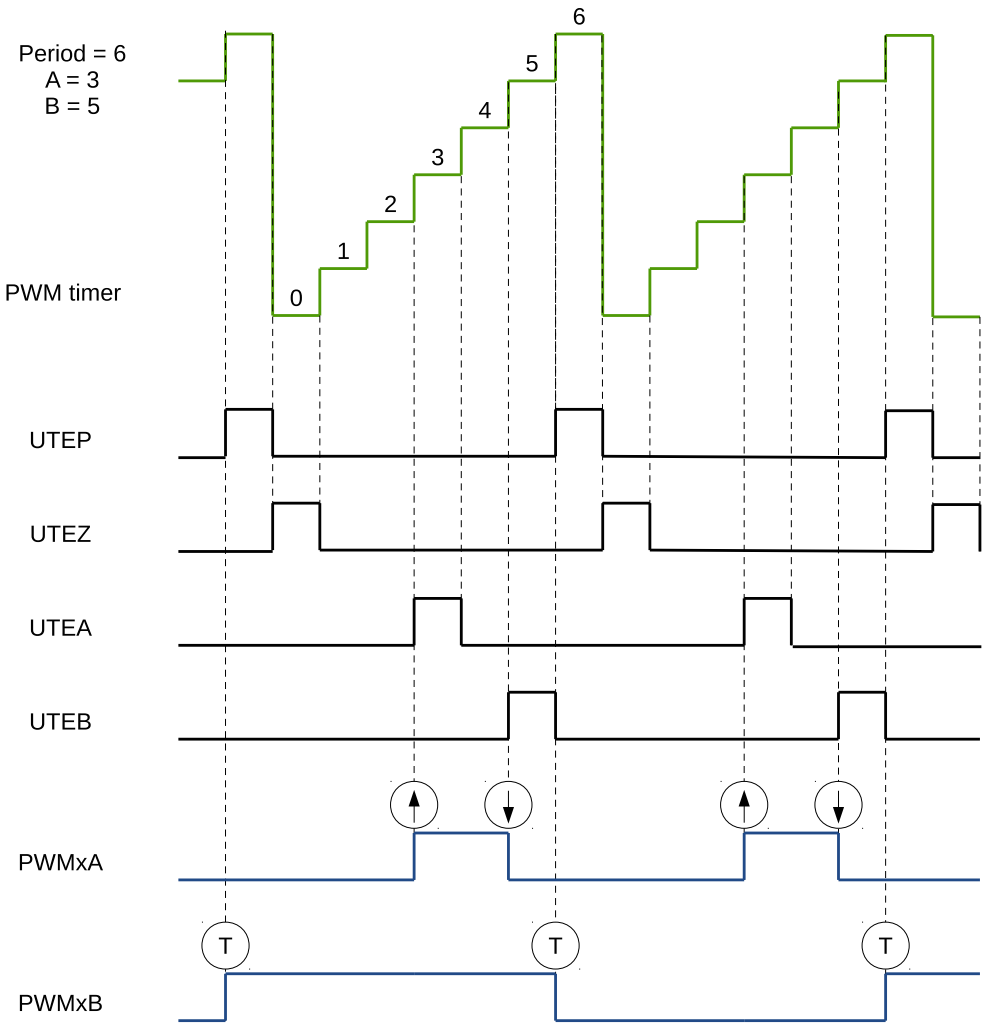


Figure 107: Count-Up, Pulse Placement Asymmetric Waveform with Independent Modulation on PWM*x*A

Pulses may be generated anywhere within the PWM cycle (zero – period).

PWM*x*A's high time duty is proportional to (B – A).

$$Period = (PWM\_TIMERx\_PERIOD + 1) \times T_{PT\_clk}$$

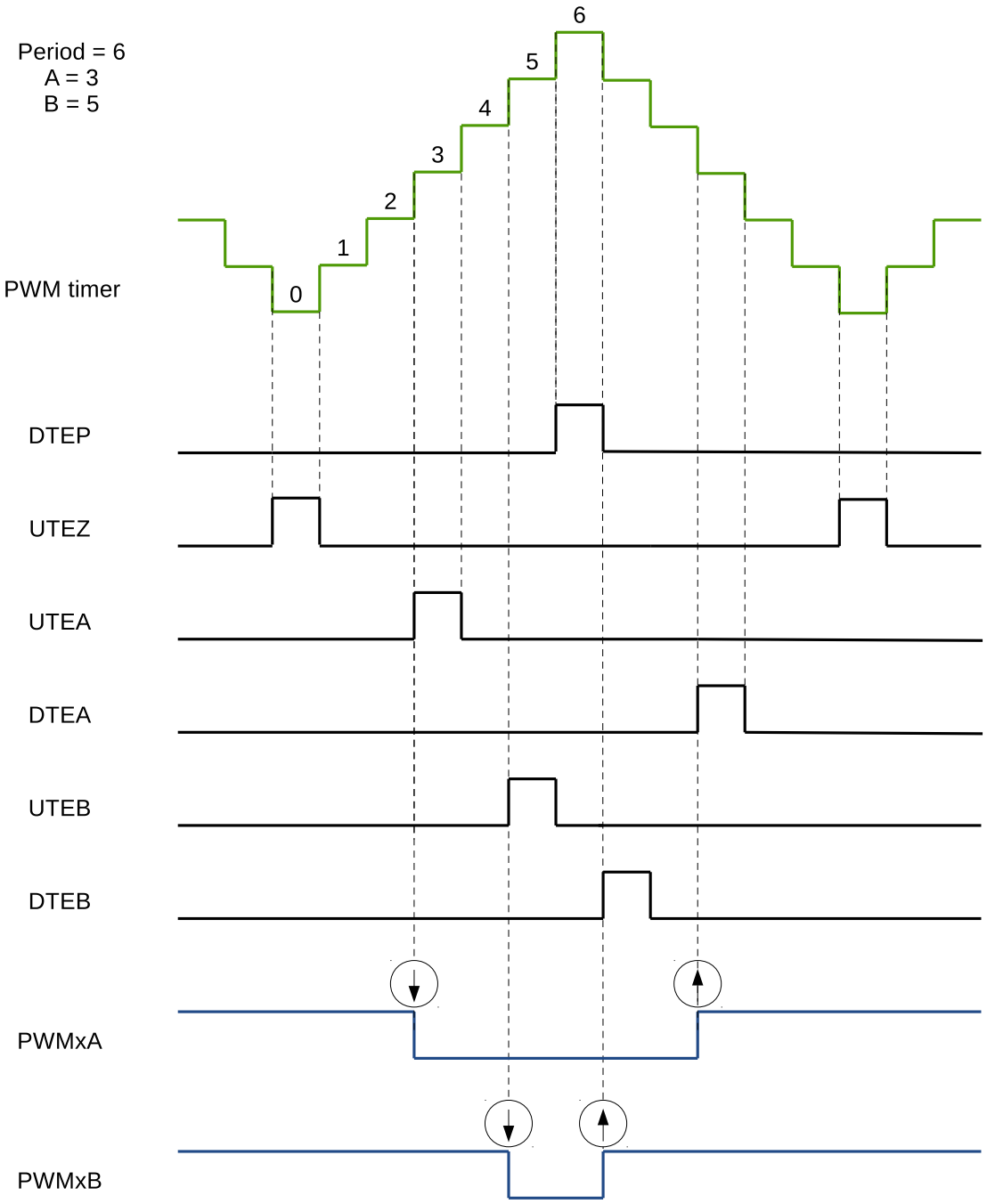


Figure 108: Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Active High

The duty modulation for PWM*x*A is set by A, active high and proportional to A.
The duty modulation for PWM*x*B is set by B, active high and proportional to B.
Outputs PWM*x*A and PWM*x*B can drive independent switches.

$$Period = 2 \times PWM\_TIMERx\_PERIOD \times T_{PT\_clk}$$

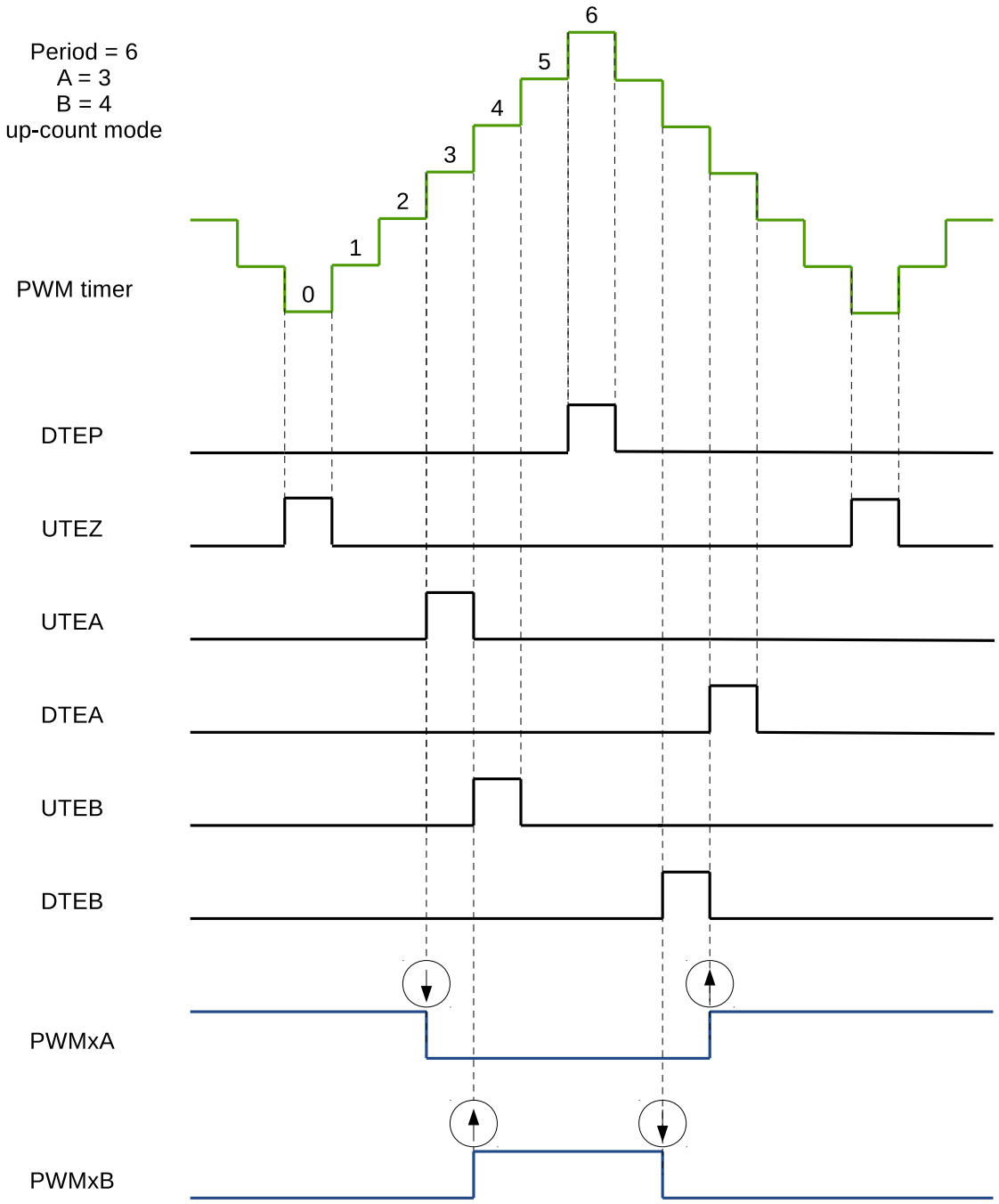


Figure 109: Count-Up-Down, Dual Edge Symmetric Waveform, with Independent Modulation on PWM*x*A and PWM*x*B — Complementary

The duty modulation of PWM*x*A is set by A, is active high and proportional to A.
The duty modulation of PWM*x*B is set by B, is active low and proportional to B.
Outputs PWM*x* can drive upper/lower (complementary) switches.
Dead-time = B – A; Edge placement is fully programmable by software. Use the dead-time generator module if another edge delay method is required.

$$Period = 2 \times PWM\_TIMERx\_PERIOD \times T_{PT\_clk}$$

Software-Force Events

There are two types of software-force events inside the PWM generator:

- Non-continuous-immediate (NCI) software-force events
  Such types of events are immediately effective on PWM outputs when triggered by software. The forcing is non-continuous, meaning the next active timing events will be able to alter the PWM outputs.
- Continuous (CNTU) software-force events
  Such types of events are continuous. The forced PWM outputs will continue until they are released by software. The events' triggers are configurable. They can be timing events or immediate events.

Figure 110 shows a waveform of NCI software-force events. NCI events are used to force PWM*x*A output low. Forcing on PWM*x*B is disabled in this case.

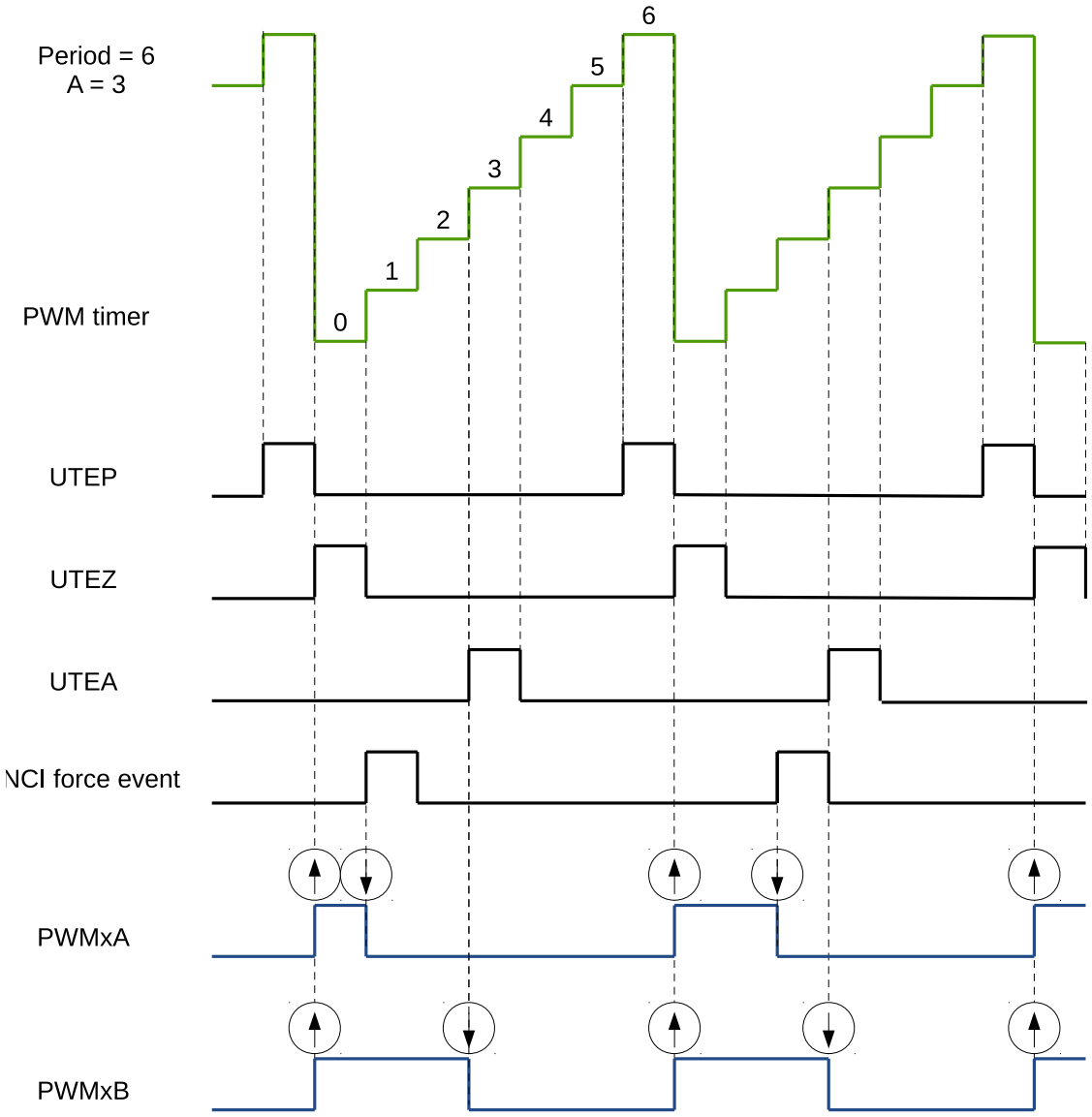


Figure 110: Example of an NCI Software-Force Event on PWM*x*A

Figure 111 shows a waveform of CNTU software-force events. UTEZ events are selected as triggers for CNTU software-force events. CNTU is used to force the PWM*x*B output low. Forcing on PWM*x*A is disabled.

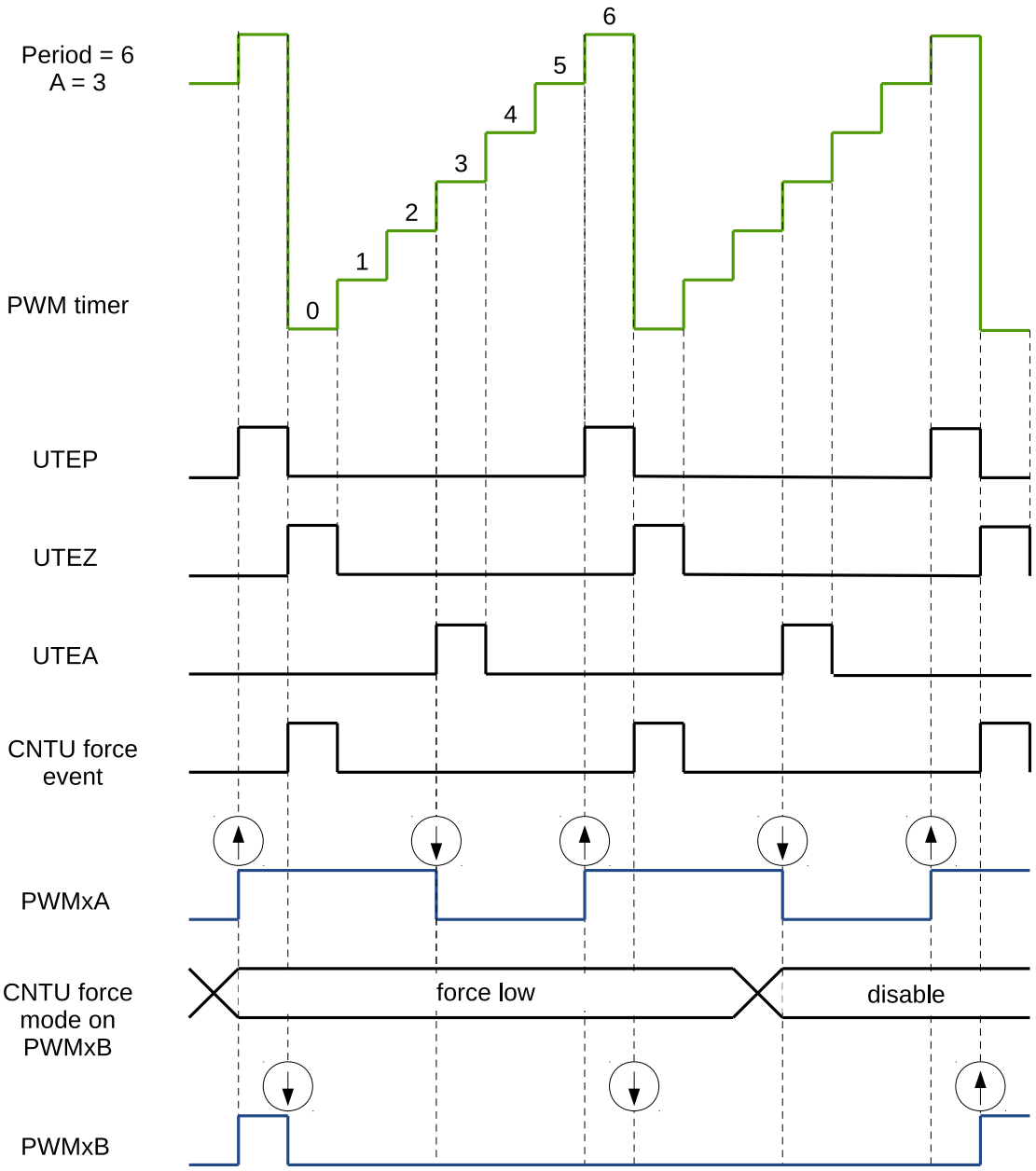


Figure 111: Example of a CNTU Software-Force Event on PWM*x*B

16.3.3.2 Dead Time Generator Submodule

Purpose of the Dead Time Generator Submodule

Several options to generate signals on PWM*x*A and PWM*x*B outputs, with a specific placement of signal edges, have been discussed in section 16.3.3.1. The required dead time is obtained by altering the edge placement between signals and by setting the signal's duty cycle. Another option is to control the dead time using a specialized submodule – the Dead Time Generator.

The key functions of the dead time generator submodule are as follows:

- Generating signal pairs (PWM*x*A and PWM*x*B) with a dead time from a single PWM*x*A input
- Creating a dead time by adding delay to signal edges:
  - Rising edge delay (RED)
  - Falling edge delay (FED)
- Configuring the signal pairs to be:
  - Active high complementary (AHC)
  - Active low complementary (ALC)
  - Active high (AH)
  - Active low (AL)
- This submodule may also be bypassed, if the dead time is configured directly in the generator submodule.

Dead Time Generator's Shadow Registers

Delay registers RED and FED are shadowed with registers PWM_DT*x*_RED_CFG_REG and PWM_DT*x*_FED_CFG_REG. For the description of shadow registers, please see section 16.3.2.3.

Highlights for Operation of the Dead Time Generator

Options for setting up the dead-time submodule are shown in Figure 112.

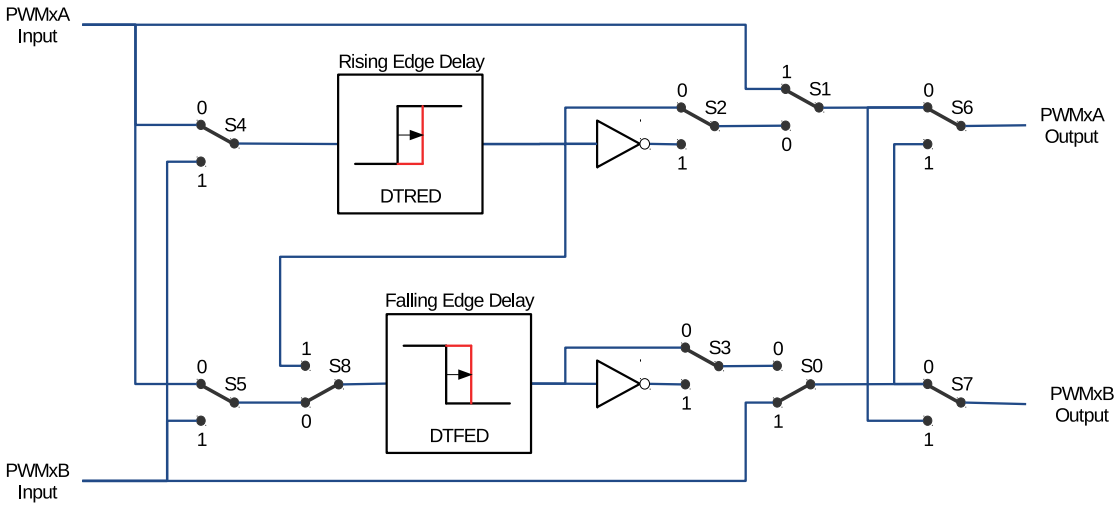


Figure 112: Options for Setting up the Dead Time Generator Submodule

S0-8 in the figure above are switches controlled by registers PWM_DT*x*_CFG_REG shown in Table 72.

Table 72: Dead Time Generator Switches Control Registers

Switch	Register
S0	PWM_DT*x*_B_OUTBYPASS
S1	PWM_DT*x*_A_OUTBYPASS
S2	PWM_DT*x*_RED_OUTINVERT
S3	PWM_DT*x*_FED_OUTINVERT
S4	PWM_DT*x*_RED_INSEL
S5	PWM_DT*x*_FED_INSEL
S6	PWM_DT*x*_A_OUTSWAP
S7	PWM_DT*x*_B_OUTSWAP
S8	PWM_DT*x*_DEB_MODE

All switch combinations are supported, but not all of them represent the typical modes of use. Table 73 documents some typical dead time configurations. In these configurations the position of S4 and S5 sets PWM*x*A as the common source of both falling-edge and rising-edge delay. The modes presented in table 73 may be categorized as follows:

- **Mode 1: Bypass delays on both falling (FED) as well as raising edge (RED)**
  In this mode the dead time submodule is disabled. Signals PWM*x*A and PWM*x*B pass through without any modifications.
- **Mode 2-5: Classical Dead Time Polarity Settings**
  These modes represent typical configurations of polarity and should cover the active-high/low modes in available industry power switch gate drivers. The typical waveforms are shown in Figures 113 to 116.
- **Modes 6 and 7: Bypass delay on falling edge (FED) or rising edge (RED)**

In these modes, either RED (Rising Edge Delay) or FED (Falling Edge Delay) is bypassed. As a result, the corresponding delay is not applied.

Table 73: Typical Dead Time Generator Operating Modes

Mode	Mode Description	S0	S1	S2	S3
1	PWM*x*A and PWM*x*B Pass Through/No Delay	1	1	X	X
2	Active High Complementary (AHC), see Figure 113	0	0	0	1
3	Active Low Complementary (ALC), see Figure 114	0	0	1	0
4	Active High (AH), see Figure 115	0	0	0	0
5	Active Low (AL), see Figure 116	0	0	1	1
6	PWM*x*A Output = PWM*x*A In (No Delay) PWM*x*B Output = PWM*x*A Input with Falling Edge Delay	0	1	0 or 1	0 or 1
7	PWM*x*A Output = PWM*x*A Input with Rising Edge Delay PWM*x*B Output = PWM*x*B Input with No Delay	1	0	0 or 1	0 or 1
Note: For all the modes above, the position of the binary switches S4 to S8 is set to 0.					

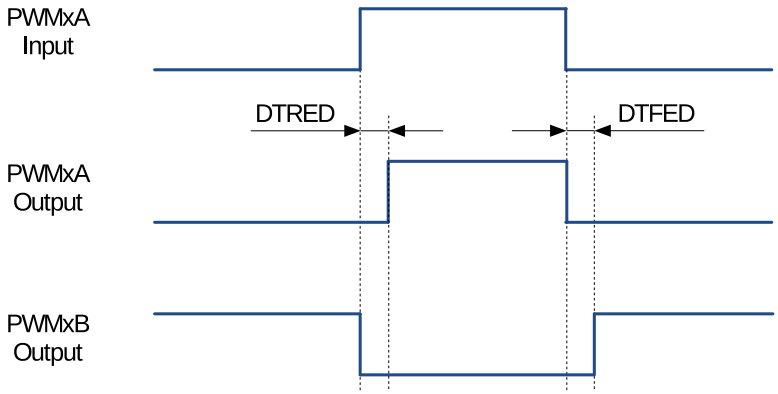


Figure 113: Active High Complementary (AHC) Dead Time Waveforms

Rising edge (RED) and falling edge (FED) delays may be set up independently. The delay value is programmed using the 16-bit registers PWM_DT*x*_RED and PWM_DT*x*_FED. The register value represents the number of clock (DT_clk) periods by which a signal edge is delayed. DT_CLK can be selected from PWM_clk or PT_clk through register PWM_DT*x*_CLK_SEL.

To calculate the delay on falling edge (FED) and rising edge (RED), use the following formulas:

$$FED = PWM\_DTx\_FED \times T_{DT\_clk}$$

$$RED = PWM\_DTx\_RED \times T_{DT\_clk}$$

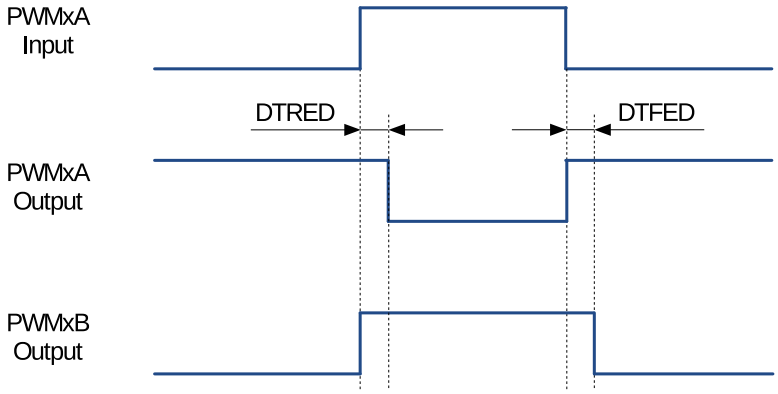


Figure 114: Active Low Complementary (ALC) Dead Time Waveforms

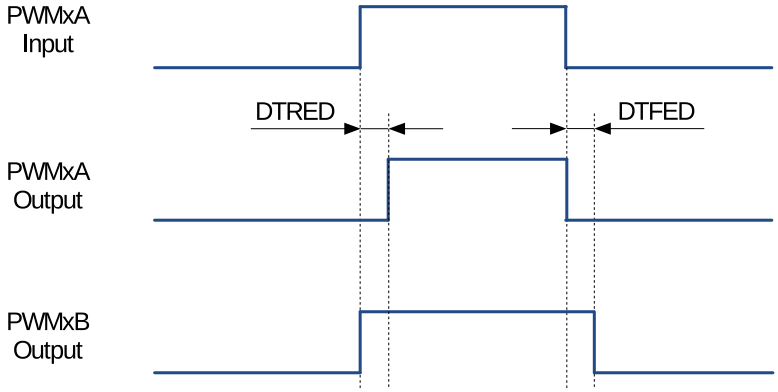


Figure 115: Active High (AH) Dead Time Waveforms

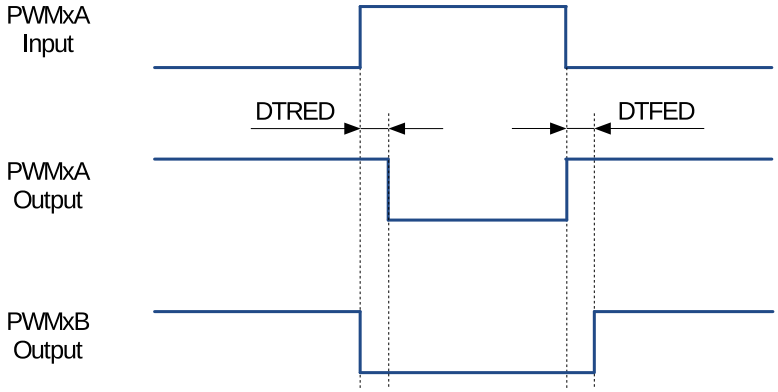


Figure 116: Active Low (AL) Dead Time Waveforms

16.3.3.3 PWM Carrier Submodule

The coupling of PWM output to a motor driver may need isolation with a transformer. Transformers deliver only AC signals, while the duty cycle of a PWM signal may range anywhere from 0% to 100%. The PWM carrier submodule passes such a PWM signal through a transformer by using a high frequency carrier to modulate the signal.

Function Overview

The following key characteristics of this submodule are configurable:

- Carrier frequency
- Pulse width of the first pulse
- Duty cycle of the second and the subsequent pulses
- Enabling/disabling the carrier function

Operational Highlights

The PWM carrier clock (PC_clk) is derived from PWM_clk. The frequency and duty cycle are configured by the PWM_CARRIER*x*_PRESCALE and PWM_CARRIER*x*_DUTY bits in the PWM_CARRIER*x*_CFG_REG register. The purpose of one-shot pulses is to provide high-energy impulse to reliably turn on the power switch. Subsequent pulses sustain the power-on status. The width of a one-shot pulse is configurable with the PWM_CARRIER*x*_OSHTWTH bits. Enabling/disabling of the carrier submodule is done with the PWM_CARRIER*x*_EN bit.

Waveform Examples

Figure 117 shows an example of waveforms, where a carrier is superimposed on original PWM pulses. This figure do not show the first one-shot pulse and the duty-cycle control. Related details are covered in the following two sections.

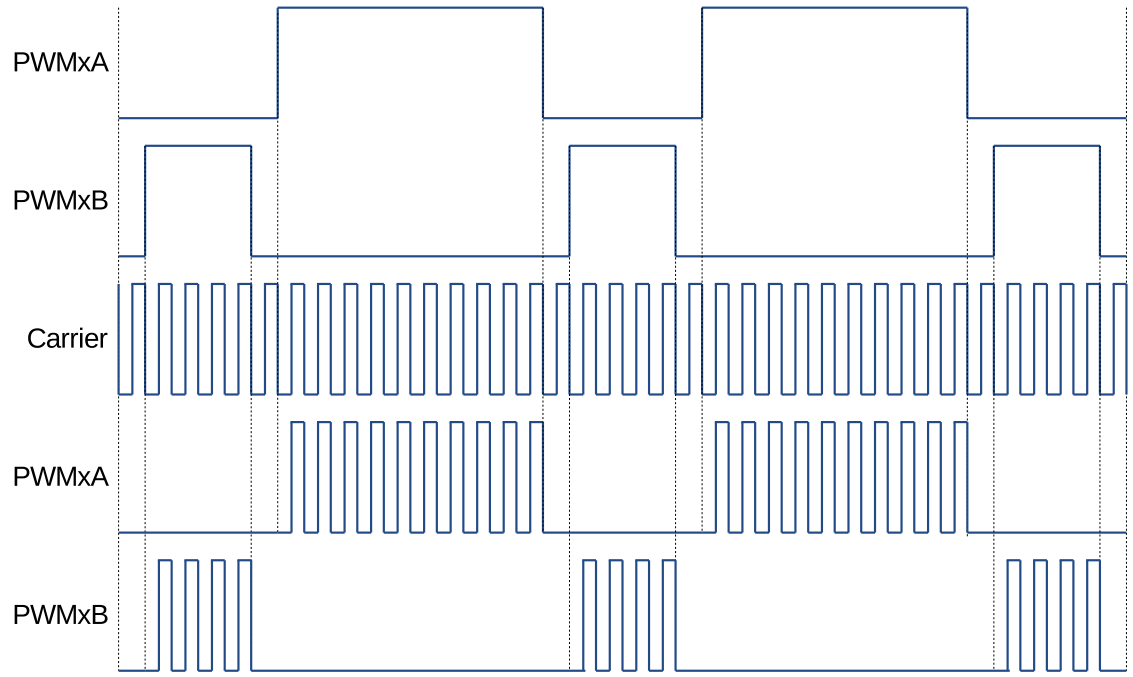


Figure 117: Example of Waveforms Showing PWM Carrier Action

One-Shot Pulse

The width of the first pulse is configurable. It may assume one of 16 possible values and is described by the formula below:

$$T_{1stpulse} = T_{PWM\_clk} \times 8 \times (PWM\_CARRIERx\_PRESCALE + 1) \times (PWM\_CARRIERx\_OSHTWTH + 1)$$

Where:

- T_{PMW_clk} is the period of the PWM clock (PWM_clk).
- $(PWM\_CARRIERx\_OSHTWTH + 1)$ is the width of the first pulse (whose value ranges from 1 to 16).
- $(PWM\_CARRIERx\_PRESCALE + 1)$ is the PWM carrier clock's (PC_clk) prescaler value.

The first one-shot pulse and subsequent sustaining pulses are shown in Figure 118.

Duty Cycle Control

After issuing the first one-shot pulse, the remaining PWM signal is modulated according to the carrier frequency. Users can configure the duty cycle of this signal. Tuning of duty may be required, so that the signal passes through the isolating transformer and can still operate (turn on/off) the motor drive, changing rotation speed and direction.

The duty cycle may be set to one of seven values, using PWM_CARRIERx_DUTY, or bits [7:5] of register PWM_CARRIERx_CFG_REG.

Below is the formula for calculating the duty cycle:

$$Duty = PWM\_CARRIERx\_DUTY \div 8$$

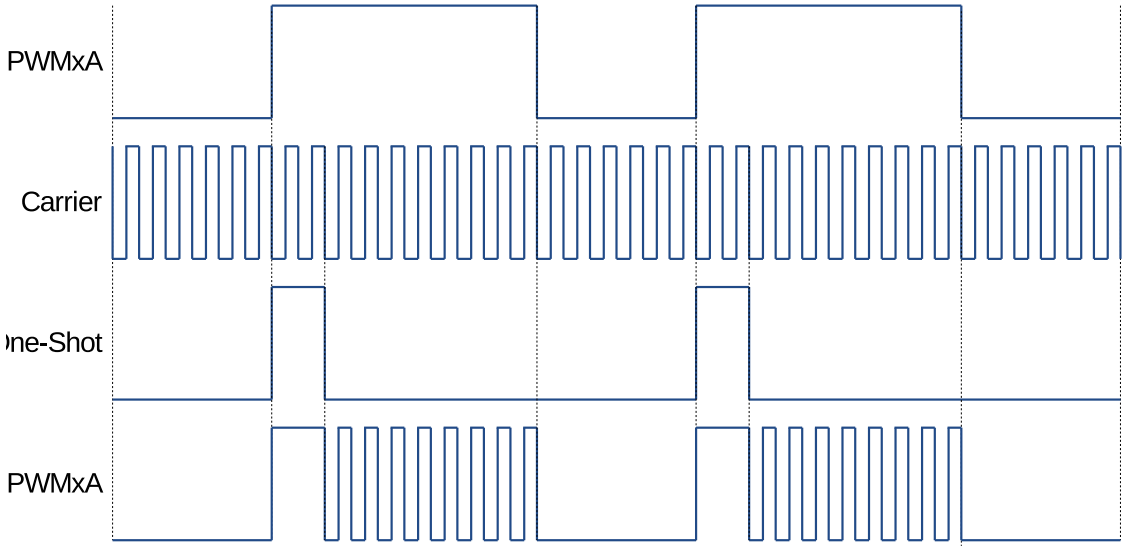


Figure 118: Example of the First Pulse and the Subsequent Sustaining Pulses of the PWM Carrier Submodule

All seven settings of the duty cycle are shown in Figure 119.

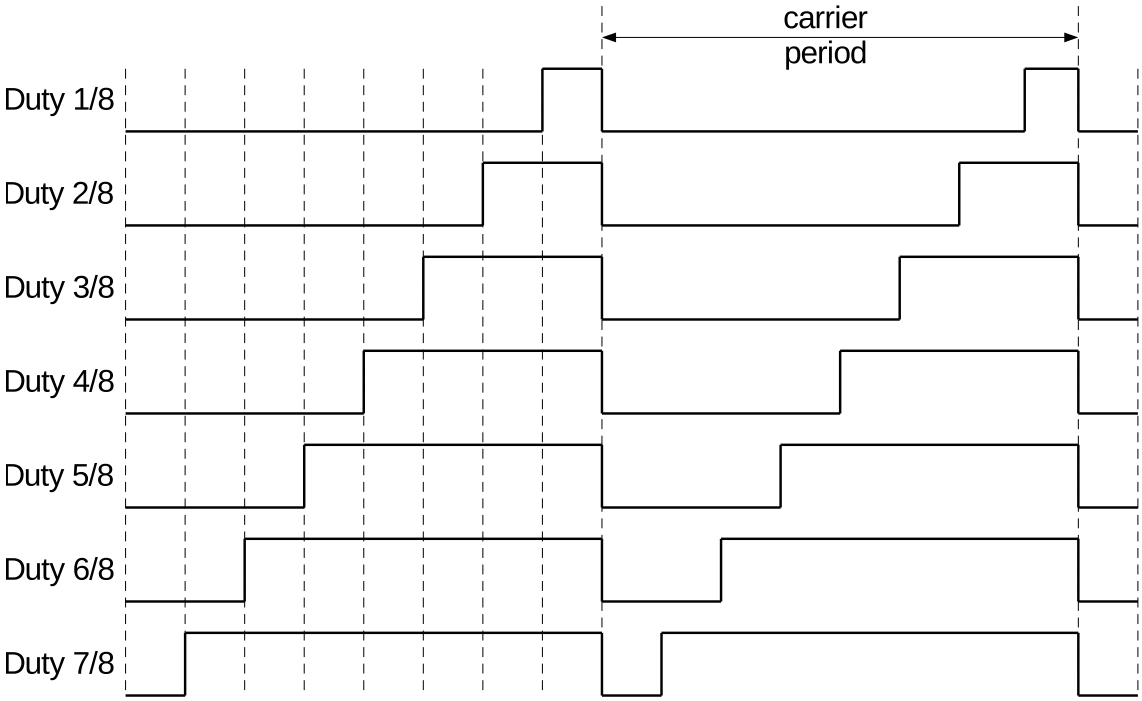


Figure 119: Possible Duty Cycle Settings for Sustaining Pulses in the PWM Carrier Submodule

16.3.3.4 Fault Handler Submodule

Each MCPWM peripheral is connected to three fault signals (FAULT0, FAULT1 and FAULT2) which are sourced from the GPIO matrix. These signals are intended to indicate external fault conditions, and may be preprocessed by the fault detection submodule to generate fault events. Fault events can then execute the user code to control

MCPWM outputs in response to specific faults.

Function of Fault Handler Submodule

The key actions performed by the fault handler submodule are:

- Forcing outputs PWM*x*A and PWM*x*B, upon detected fault, to one of the following states:
  - High
  - Low
  - Toggle
  - No action taken
- Execution of one-shot trip (OST) upon detection of over-current conditions/short circuits.
- Cycle-by-cycle tripping (CBC) to provide current-limiting operation.
- Allocation of either one-shot or cycle-by-cycle operation for each fault signal.
- Generation of interrupts for each fault input.
- Support for software-force tripping.
- Enabling or disabling of submodule function as required.

Operation and Configuration Tips

This section provides the operational tips and set-up options for the fault handler submodule.

Fault signals coming from pads are sampled and synced in the GPIO matrix. In order to guarantee the successful sampling of fault pulses, each pulse duration must be at least two APB clock cycles. The fault detection submodule will then sample fault signals by using PWM_clk. So, the duration of fault pulses coming from GPIO matrix must be at least one PWM_clk cycle. Differently put, regardless of the period relation between APB clock and PWM_clk, the width of fault signal pulses on pads must be at least equal to the sum of two APB clock cycles and one PWM_clk cycle.

Each level of fault signals, FAULT0 to FAULT2, can be used by the fault handler submodule to generate fault events (fault_event0 to fault_event2). Every fault event can be configured individually to provide CBC action, OST action, or none.

- **Cycle-by-Cycle (CBC) action:**
  When CBC action is triggered, the state of PWM*x*A and PWM*x*B will be changed immediately according to the configuration of registers PWM_FH*x*_A_CBC_U/D and PWM_FH*x*_B_CBC_U/D. Different actions can be indicted when the PWM timer is incrementing or decrementing. Different CBC action interrupts can be triggered for different fault events. Status register PWM_FH*x*_CBC_ON indicates whether a CBC action is on or off. When the fault event is no longer present, CBC actions on PWM*x*A/B will be cleared at a specified point, which is either a D/UTEP or D/UTEZ event. Register PWM_FH*x*_CBCPULSE determines at which event PWM*x*A and PWM*x*B will be able to resume normal actions. Therefore, in this mode, the CBC action is cleared or refreshed upon every PWM cycle.
- **One-Shot (OST) action:**

When OST action is triggered, the state of PWM*x*A and PWM*x*B will be changed immediately, depending on the setting of registers PWM_FH*x*_A_OST_U/D and PWM_FH*x*_B_OST_U/D. Different actions can be configured when PWM timer is incrementing or decrementing. Different OST action interrupts can be triggered form different fault events. Status register PWM_FH*x*_OST_ON indicates whether an OST action is on or off. The OST actions on PWM*x*A/B are not automatically cleared when the fault event is no longer present. One-shot actions must be cleared manually by negating the value stored in register PWM_FH*x*_CLR_OST.

16.3.4 Capture Submodule

16.3.4.1 Introduction

The capture submodule contains three complete capture channels. Channel inputs CAP0, CAP1 and CAP2 are sourced from the GPIO matrix. Thanks to the flexibility of the GPIO matrix, CAP0, CAP1 and CAP2 can be configured from any PAD input. Multiple capture channels can be sourced from the same PAD input, while prescaling for each channel can be set differently. Also, capture channels are sourced from different PADs. This provides several options for handling capture signals by hardware in the background, instead of having them processed directly by the CPU. A capture submodule has the following independent key resources:

- One 32-bit timer (counter) which can be synchronized with the PWM timer, another submodule or software.
- Three capture channels, each equipped with a 32-bit time-stamp and a capture prescaler.
- Independent edge polarity (rising/falling edge) selection for any capture channel.
- Input capture signal prescaling (from 1 to 256).
- Interrupt capabilities on any of the three capture events.

16.3.4.2 Capture Timer

The capture timer is a 32-bit counter incrementing continuously, once enabled. On the input it has an APB clock running typically at 80 MHz. At a sync event the counter is loaded with phase stored in register PWM_CAP_TIMER_PHASE_REG. Sync events can come from PWM timers sync-out, PWM module sync-in or software. The capture timer provides timing references for all three capture channels.

16.3.4.3 Capture Channel

The capture signal coming to a capture channel will be inverted first, if needed, and then prescaled. Finally, specified edges of preprocessed capture signal will trigger capture events. When a capture event occurs, the capture timer's value is stored in time-stamp register PWM_CAP_CH*x*_REG. Different interrupts can be generated for different capture channels at capture events. The edge that triggers a capture event is recorded in register PWM_CAP*x*_EDGE. The capture event can be also forced by software.

16.4 Register Summary

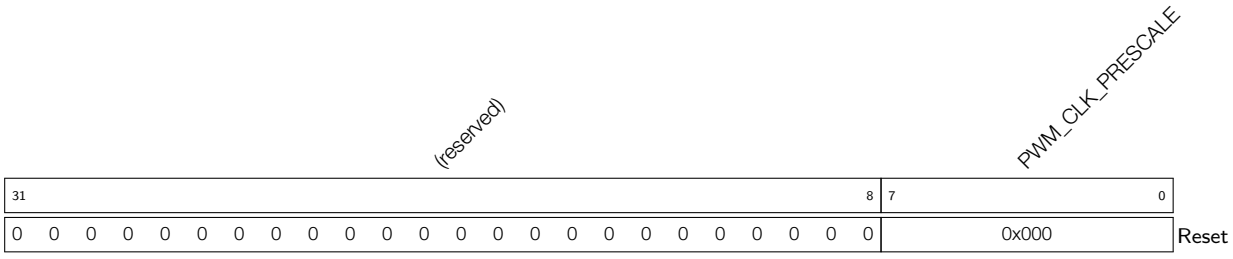
Name	Description	PWM0	PWM1	Acc
Prescaler configuration				
PWM_CLK_CFG_REG	Configuration of the prescaler	0x3FF5E000	0x3FF6C000	R/W
PWM Timer 0 Configuration and status				
PWM_TIMER0_CFG0_REG	Timer period and update method	0x3FF5E004	0x3FF6C004	R/W
PWM_TIMER0_CFG1_REG	Working mode and start/stop control	0x3FF5E008	0x3FF6C008	R/W
PWM_TIMER0_SYNC_REG	Synchronization settings	0x3FF5E00C	0x3FF6C00C	R/W
PWM_TIMER0_STATUS_REG	Timer status	0x3FF5E010	0x3FF6C010	RO
PWM Timer 1 Configuration and Status				
PWM_TIMER1_CFG0_REG	Timer update method and period	0x3FF5E014	0x3FF6C014	R/W
PWM_TIMER1_CFG1_REG	Working mode and start/stop control	0x3FF5E018	0x3FF6C018	R/W
PWM_TIMER1_SYNC_REG	Synchronization settings	0x3FF5E01C	0x3FF6C01C	R/W
PWM_TIMER1_STATUS_REG	Timer status	0x3FF5E020	0x3FF6C020	RO
PWM Timer 2 Configuration and status				
PWM_TIMER2_CFG0_REG	Timer update method and period	0x3FF5E024	0x3FF6C024	R/W
PWM_TIMER2_CFG1_REG	Working mode and start/stop control	0x3FF5E028	0x3FF6C028	R/W
PWM_TIMER2_SYNC_REG	Synchronization settings	0x3FF5E02C	0x3FF6C02C	R/W
PWM_TIMER2_STATUS_REG	Timer status	0x3FF5E030	0x3FF6C030	RO
Common configuration for PWM timers				
PWM_TIMER_SYNCI_CFG_REG	Synchronization input selection for timers	0x3FF5E034	0x3FF6C034	R/W
PWM_OPERATOR_TIMERSEL_REG	Select specific timer for PWM operators	0x3FF5E038	0x3FF6C038	R/W
PWM Operator 0 Configuration and Status				
PWM_GEN0_STMP_CFG_REG	Transfer status and update method for time stamp registers A and B	0x3FF5E03C	0x3FF6C03C	R/W
PWM_GEN0_TSTMP_A_REG	Shadow register for register A	0x3FF5E040	0x3FF6C040	R/W
PWM_GEN0_TSTMP_B_REG	Shadow register for register B	0x3FF5E044	0x3FF6C044	R/W
PWM_GEN0_CFG0_REG	Fault event T0 and T1 handling	0x3FF5E048	0x3FF6C048	R/W
PWM_GEN0_FORCE_REG	Permissives to force PWM0A and PWM0B outputs by software	0x3FF5E04C	0x3FF6C04C	R/W
PWM_GEN0_A_REG	Actions triggered by events on PWM0A	0x3FF5E050	0x3FF6C050	R/W
PWM_GEN0_B_REG	Actions triggered by events on PWM0B	0x3FF5E054	0x3FF6C054	R/W
PWM_DT0_CFG_REG	Dead time type selection and configuration	0x3FF5E058	0x3FF6C058	R/W
PWM_DT0_FED_CFG_REG	Shadow register for falling edge delay (FED)	0x3FF5E05C	0x3FF6C05C	R/W
PWM_DT0_RED_CFG_REG	Shadow register for rising edge delay (RED)	0x3FF5E060	0x3FF6C060	R/W
PWM_CARRIER0_CFG_REG	Carrier enable and configuration	0x3FF5E064	0x3FF6C064	R/W

Name	Description	PWM0	PWM1	Acc
PWM_FH0_CFG0_REG	Actions on PWM0A and PWM0B on trip events	0x3FF5E068	0x3FF6C068	R/W
PWM_FH0_CFG1_REG	Software triggers for fault handler actions	0x3FF5E06C	0x3FF6C06C	R/W
PWM_FH0_STATUS_REG	Status of fault events	0x3FF5E070	0x3FF6C070	RO
PWM Operator 1 Configuration and Status				
PWM_GEN1_STMP_CFG_REG	Transfer status and update method for time stamp registers A and B	0x3FF5E074	0x3FF6C074	R/W
PWM_GEN1_TSTMP_A_REG	Shadow register for register A	0x3FF5E078	0x3FF6C078	R/W
PWM_GEN1_TSTMP_B_REG	Shadow register for register B	0x3FF5E07C	0x3FF6C07C	R/W
PWM_GEN1_CFG0_REG	Fault event T0 and T1 handling	0x3FF5E080	0x3FF6C080	R/W
PWM_GEN1_FORCE_REG	Permissives to force PWM1A and PWM1B outputs by software	0x3FF5E084	0x3FF6C084	R/W
PWM_GEN1_A_REG	Actions triggered by events on PWM1A	0x3FF5E088	0x3FF6C088	R/W
PWM_GEN1_B_REG	Actions triggered by events on PWM1B	0x3FF5E08C	0x3FF6C08C	R/W
PWM_DT1_CFG_REG	Dead time type selection and configuration	0x3FF5E090	0x3FF6C090	R/W
PWM_DT1_FED_CFG_REG	Shadow register for FED	0x3FF5E094	0x3FF6C094	R/W
PWM_DT1_RED_CFG_REG	Shadow register for RED	0x3FF5E098	0x3FF6C098	R/W
PWM_CARRIER1_CFG_REG	Carrier enable and configuration	0x3FF5E09C	0x3FF6C09C	R/W
PWM_FH1_CFG0_REG	Actions on PWM1A and PWM1B on fault events	0x3FF5E0A0	0x3FF6C0A0	R/W
PWM_FH1_CFG1_REG	Software triggers for fault handler actions	0x3FF5E0A4	0x3FF6C0A4	R/W
PWM_FH1_STATUS_REG	Status of fault events	0x3FF5E0A8	0x3FF6C0A8	RO
PWM Operator 2 Configuration and Status				
PWM_GEN2_STMP_CFG_REG	Transfer status and updating method for time stamp registers A and B	0x3FF5E0AC	0x3FF6C0AC	R/W
PWM_GEN2_TSTMP_A_REG	Shadow register for register A	0x3FF5E0B0	0x3FF6C0B0	R/W
PWM_GEN2_TSTMP_B_REG	Shadow register for register B	0x3FF5E0B4	0x3FF6C0B4	R/W
PWM_GEN2_CFG0_REG	Fault event T0 and T1 handling	0x3FF5E080	0x3FF6C080	R/W
PWM_GEN2_FORCE_REG	Permissives to force PWM2A and PWM2B outputs by software	0x3FF5E0BC	0x3FF6C0BC	R/W
PWM_GEN2_A_REG	Actions triggered by events on PWM2A	0x3FF5E0C0	0x3FF6C0C0	R/W
PWM_GEN2_B_REG	Actions triggered by events on PWM2B	0x3FF5E0C4	0x3FF6C0C4	R/W
PWM_DT2_CFG_REG	Dead time type selection and configuration	0x3FF5E0C8	0x3FF6C0C8	R/W
PWM_DT2_FED_CFG_REG	Shadow register for FED	0x3FF5E0CC	0x3FF6C0CC	R/W
PWM_DT2_RED_CFG_REG	Shadow register for RED	0x3FF5E0D0	0x3FF6C0D0	R/W
PWM_CARRIER2_CFG_REG	Carrier enable and configuration	0x3FF5E0D4	0x3FF6C0D4	R/W

Name	Description	PWM0	PWM1	Acc
PWM_FH2_CFG0_REG	Actions at PWM2A and PWM2B on trip events	0x3FF5E0D8	0x3FF6C0D8	R/W
PWM_FH2_CFG1_REG	Software triggers for fault handler actions	0x3FF5E0DC	0x3FF6C0DC	R/W
PWM_FH2_STATUS_REG	Status of fault events	0x3FF5E0E0	0x3FF6C0E0	RO
Fault Detection Configuration and Status				
PWM_FAULT_DETECT_REG	Fault detection configuration and status	0x3FF5E0E4	0x3FF6C0E4	R/W
Capture Configuration and Status				
PWM_CAP_TIMER_CFG_REG	Configure capture timer	0x3FF5E0E8	0x3FF6C0E8	R/W
PWM_CAP_TIMER_PHASE_REG	Phase for capture timer sync	0x3FF5E0EC	0x3FF6C0EC	R/W
PWM_CAP_CH0_CFG_REG	Capture channel 0 configuration and enable	0x3FF5E0F0	0x3FF6C0F0	R/W
PWM_CAP_CH1_CFG_REG	Capture channel 1 configuration and enable	0x3FF5E0F4	0x3FF6C0F4	R/W
PWM_CAP_CH2_CFG_REG	Capture channel 2 configuration and enable	0x3FF5E0F8	0x3FF6C0F8	R/W
PWM_CAP_CH0_REG	Value of last capture on channel 0	0x3FF5E0FC	0x3FF6C0FC	RO
PWM_CAP_CH1_REG	Value of last capture on channel 1	0x3FF5E100	0x3FF6C100	RO
PWM_CAP_CH2_REG	Value of last capture on channel 2	0x3FF5E104	0x3FF6C104	RO
PWM_CAP_STATUS_REG	Edge of last capture trigger	0x3FF5E108	0x3FF6C108	RO
Enable update of active registers				
PWM_UPDATE_CFG_REG	Enable update	0x3FF5E10C	0x3FF6C10C	R/W
Manage Interrupts				
INT_ENA_PWM_REG	Interrupt enable bits	0x3FF5E110	0x3FF6C110	R/W
INT_RAW_PWM_REG	Raw interrupt status	0x3FF5E114	0x3FF6C114	RO
INT_ST_PWM_REG	Masked interrupt status	0x3FF5E118	0x3FF6C118	RO
INT_CLR_PWM_REG	Interrupt clear bits	0x3FF5E11C	0x3FF6C11C	WO

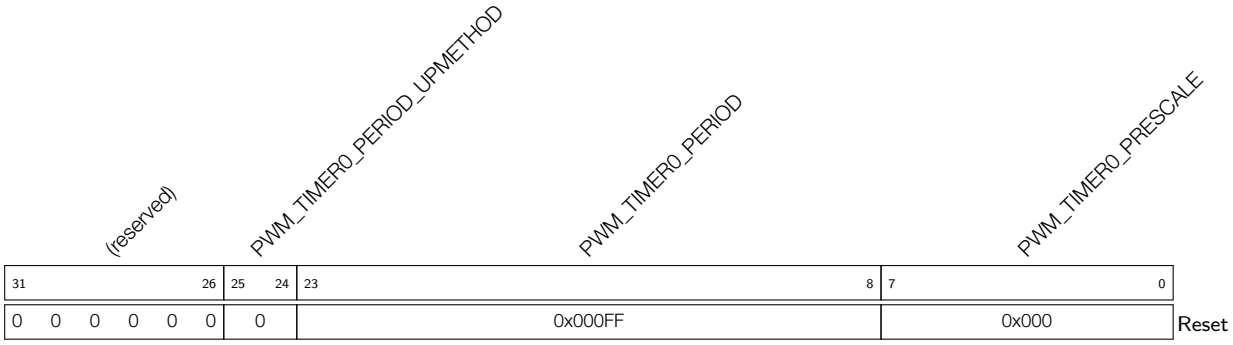
16.5 Registers

Register 16.1: PWM_CLK_CFG_REG (0x0000)



PWM_CLK_PRESCALE Period of PWM_clk = 6.25ns * (PWM_CLK_PRESCALE + 1). (R/W)

Register 16.2: PWM_TIMER0_CFG0_REG (0x0004)

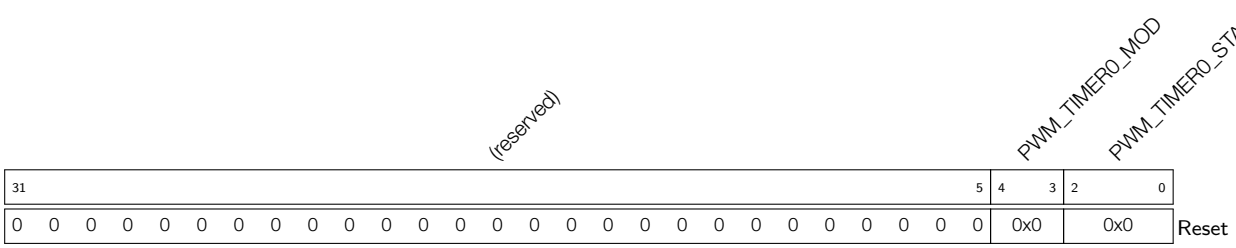


PWM_TIMER0_PERIOD_UPMETHOD Updating method for active register of PWM timer0 period. 0: immediately, 1: update at TEZ, 2: update at sync, 3: update at TEZ or sync. TEZ here and below means that the event that happens when the timer equals to zero. (R/W)

PWM_TIMER0_PERIOD Period shadow register of PWM timer0. (R/W)

PWM_TIMER0_PRESCALE Period of PT0_clk = Period of PWM_clk * (PWM_TIMER0_PRESCALE + 1). (R/W)

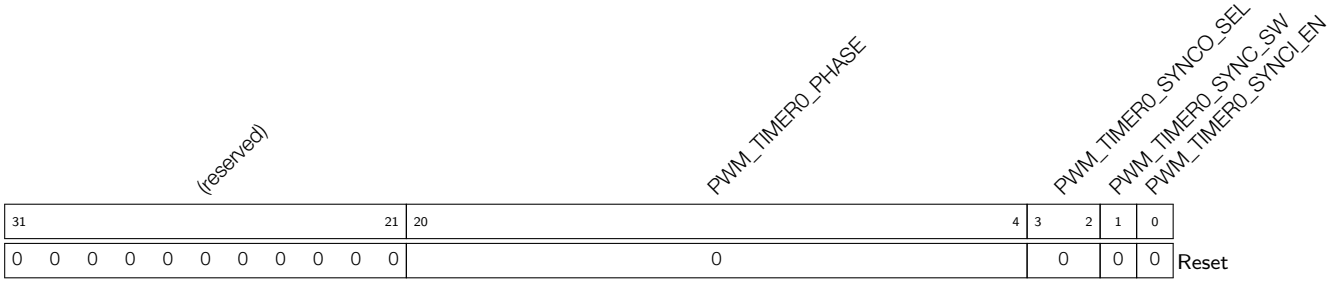
Register 16.3: PWM_TIMER0_CFG1_REG (0x0008)



PWM_TIMER0_MOD PWM timer0 working mode. 0: freeze, 1: increase mode, 2: decrease mode, 3: up-down mode. (R/W)

PWM_TIMER0_START PWM timer0 start and stop control. 0: if PWM timer0 starts, then stops at TEZ; 1: if timer0 starts, then stops at TEP; 2: PWM timer0 starts and runs on; 3: timer0 starts and stops at the next TEZ; 4: timer0 starts and stops at the next TEP. TEP here and below means the event that happens when the timer equals to period. (R/W)

Register 16.4: PWM_TIMER0_SYNC_REG (0x000c)



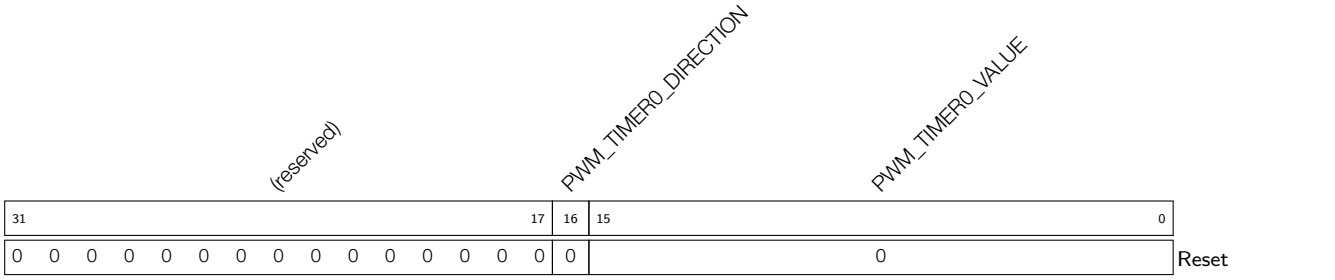
PWM_TIMER0_PHASE Phase for timer reload at sync event. (R/W)

PWM_TIMER1_SYNCO_SEL PWM timer0 sync_out selection. 0: sync_in; 1: TEZ; 2: TEP; otherwise: sync_out is always 0. (R/W)

PWM_TIMER1_SYNC_SW Toggling this bit will trigger a software sync. (R/W)

PWM_TIMER1_SYNCI_EN When set, timer reloading with phase on sync input event is enabled. (R/W)

Register 16.5: PWM_TIMER0_STATUS_REG (0x0010)



PWM_TIMER0_DIRECTION Current direction of the PWM timer0 counter. 0: increment, 1: decrement. (RO)

PWM_TIMER0_VALUE Current value of the PWM timer0 counter. (RO)

Register 16.6: PWM_TIMER1_CFG0_REG (0x0014)

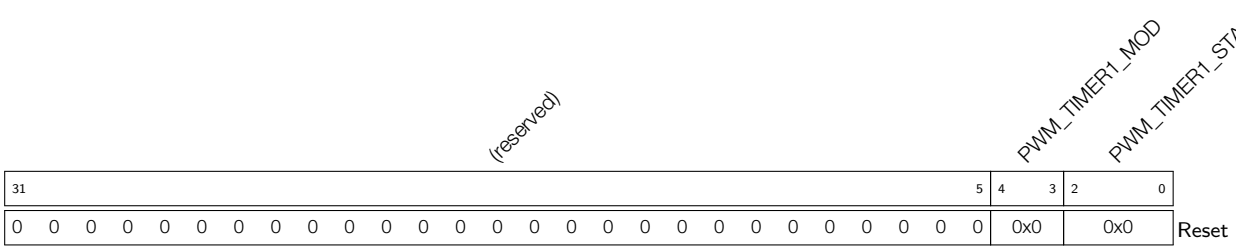


PWM_TIMER1_PERIOD_UPMETHOD Updating method for the active register of PWM timer1 period. 0: immediately, 1: update at TEZ, 2: update at sync, 3: update at TEZ or sync. (R/W)

PWM_TIMER1_PERIOD Period shadow register of the PWM timer1. (R/W)

PWM_TIMER1_PRESCALE Period of PT1_clk = Period of PWM_clk * (PWM_TIMER1_PRESCALE + 1) (R/W)

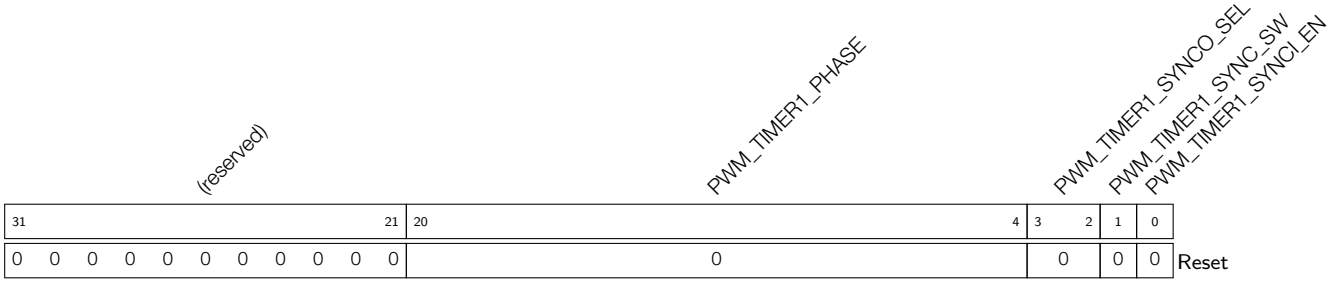
Register 16.7: PWM_TIMER1_CFG1_REG (0x0018)



PWM_TIMER1_MOD PWM timer1 working mode. 0: freeze, 1: increase mode, 2: decrease mode, 3: up-down mode. (R/W)

PWM_TIMER1_START PWM timer1 start and stop control. 0: if PWM timer1 starts, then stops at TEZ; 1: if PWM timer1 starts, then stops at TEP; 2: PWM timer1 starts and runs on; 3: PWM timer1 starts and stops at the next TEZ; 4: PWM timer1 starts and stops at the next TEP. (R/W)

Register 16.8: PWM_TIMER1_SYNC_REG (0x001c)



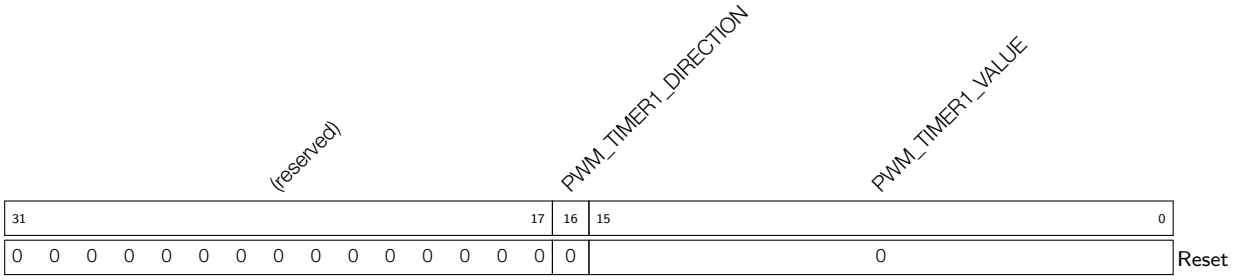
PWM_TIMER1_PHASE Phase for timer reload at sync event. (R/W)

PWM_TIMER1_SYNCO_SEL PWM timer1 sync_out selection. 0: sync_in; 1: TEZ; 2: TEP; otherwise: sync_out is always 0. (R/W)

PWM_TIMER1_SYNC_SW Toggling this bit will trigger a software sync. (R/W)

PWM_TIMER1_SYNCI_EN When set, timer reloading with phase at a sync input event is enabled. (R/W)

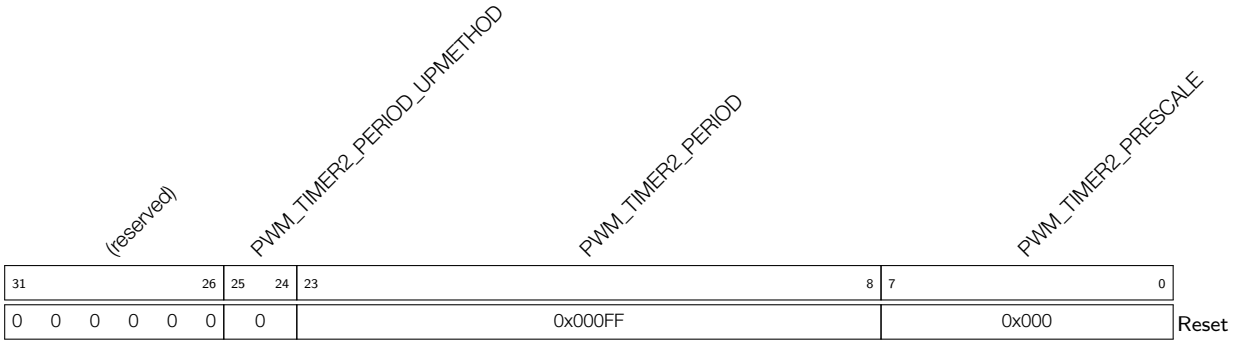
Register 16.9: PWM_TIMER1_STATUS_REG (0x0020)



PWM_TIMER1_DIRECTION Current direction of the PWM timer1 counter. 0: increment 1: decrement. (RO)

PWM_TIMER1_VALUE Current value of the PWM timer1 counter. (RO)

Register 16.10: PWM_TIMER2_CFG0_REG (0x0024)

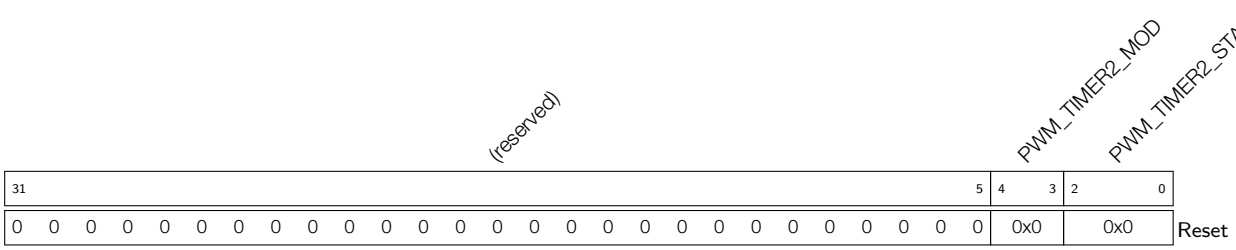


PWM_TIMER2_PERIOD_UPMETHOD Updating method for active register of PWM timer2 period. 0: immediately, 1: update at TEZ, 2: update at sync, 3: update at TEZ or sync. (R/W)

PWM_TIMER2_PERIOD Period shadow register of PWM timer2. (R/W)

PWM_TIMER2_PRESCALE Period of PT2_clk = Period of PWM_clk * (PWM_TIMER2_PRESCALE + 1). (R/W)

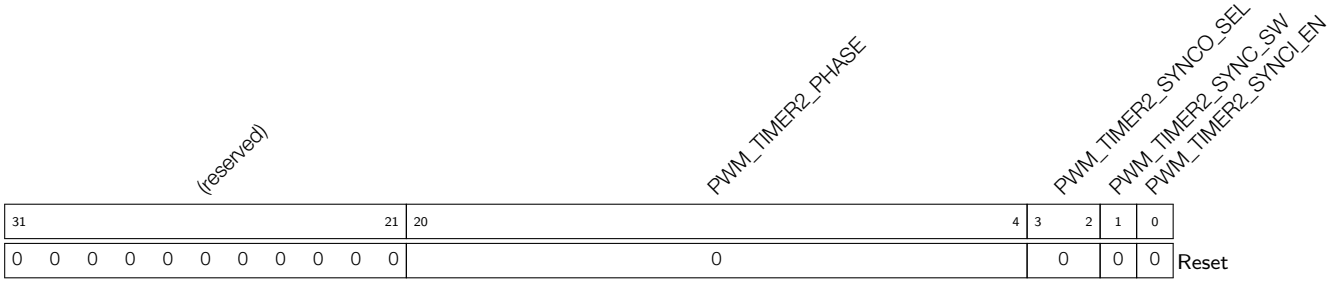
Register 16.11: PWM_TIMER2_CFG1_REG (0x0028)



PWM_TIMER2_MOD PWM timer2 working mode. 0: freeze, 1: increase mode, 2: decrease mode, 3: up-down mode. (R/W)

PWM_TIMER2_START PWM timer2 start and stop control. 0: if PWM timer2 starts, then stops at TEZ; 1: if PWM timer2 starts, then stops at TEP; 2: PWM timer2 starts and runs on; 3: PWM timer2 starts and stops at the next TEZ; 4: PWM timer2 starts and stops at the next TEP. (R/W)

Register 16.12: PWM_TIMER2_SYNC_REG (0x002c)



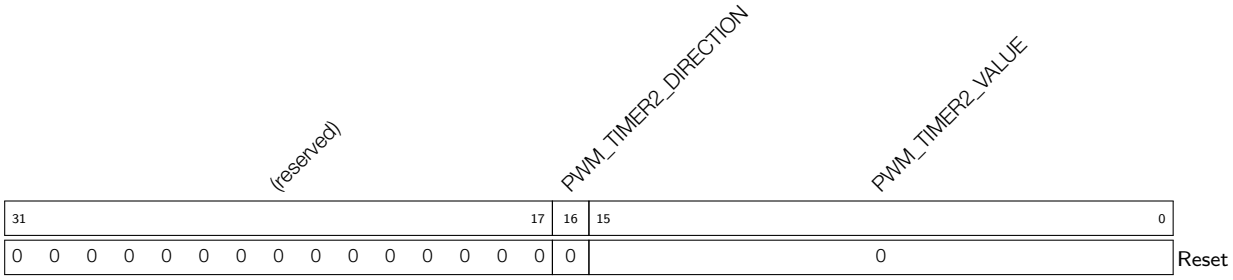
PWM_TIMER2_PHASE Phase for timer reload at sync event. (R/W)

PWM_TIMER2_SYNCO_SEL PWM timer2 sync_out selection. 0: sync_in; 1: TEZ; 2: TEP; otherwaise: sync_out is always 0. (R/W)

PWM_TIMER2_SYNC_SW Toggling this bit will trigger a software sync. (R/W)

PWM_TIMER2_SYNCI_EN When set, timer reloading with phase on sync input event is enabled. (R/W)

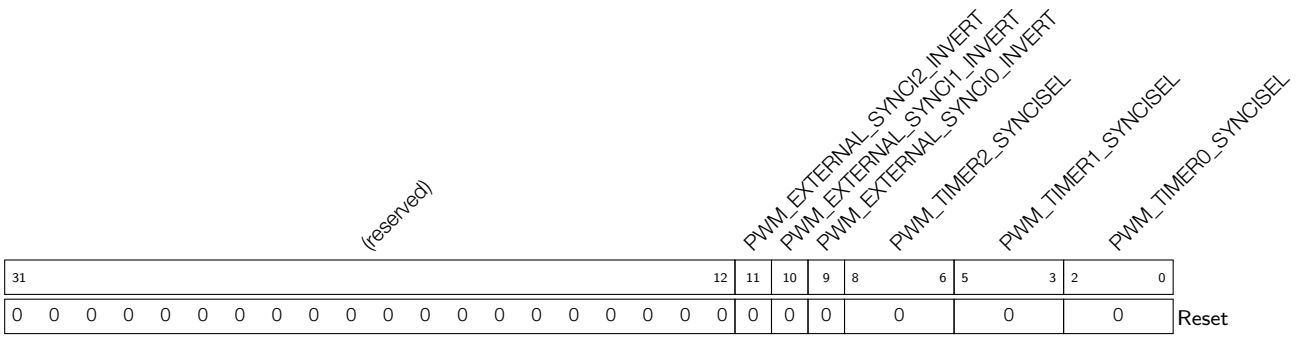
Register 16.13: PWM_TIMER2_STATUS_REG (0x0030)



PWM_TIMER2_DIRECTION Current direction of the PWM timer2 counter. 0: increment, 1: decrement. (RO)

PWM_TIMER2_VALUE Current value of the PWM timer2 counter. (RO)

Register 16.14: PWM_TIMER_SYNCI_CFG_REG (0x0034)



PWM_EXTERNAL_SYNCI2_INVERT Invert SYNC2 from GPIO matrix. (R/W)

PWM_EXTERNAL_SYNCI1_INVERT Invert SYNC1 from GPIO matrix. (R/W)

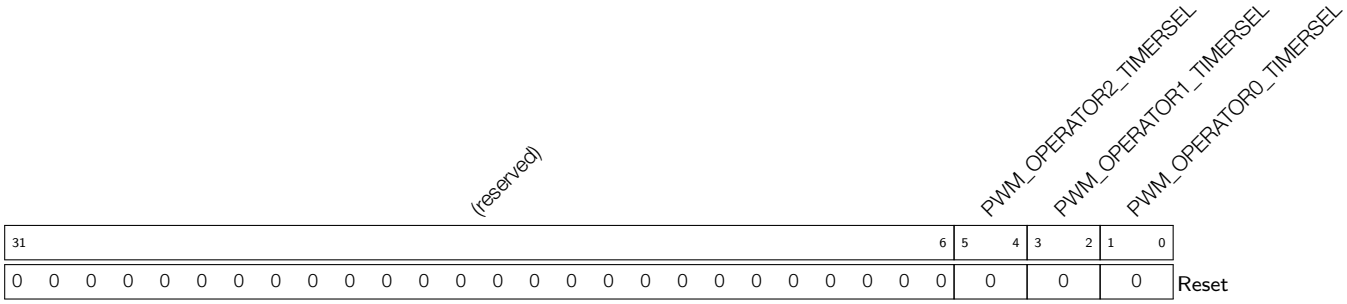
PWM_EXTERNAL_SYNCI0_INVERT Invert SYNC0 from GPIO matrix. (R/W)

PWM_TIMER2_SYNCISEL Select sync input for PWM timer2. 1: PWM timer0 sync_out, 2: PWM timer1 sync_out, 3: PWM timer2 sync_out, 4: SYNC0 from GPIO matrix, 5: SYNC1 from GPIO matrix, 6: SYNC2 from GPIO matrix, other values: no sync input selected. (R/W)

PWM_TIMER1_SYNCISEL Select sync input for PWM timer1. 1: PWM timer0 sync_out, 2: PWM timer1 sync_out, 3: PWM timer2 sync_out, 4: SYNC0 from GPIO matrix, 5: SYNC1 from GPIO matrix, 6: SYNC2 from GPIO matrix, other values: no sync input selected. (R/W)

PWM_TIMER0_SYNCISEL Select sync input for PWM timer0. 1: PWM timer0 sync_out, 2: PWM timer1 sync_out, 3: PWM timer2 sync_out, 4: SYNC0 from GPIO matrix, 5: SYNC1 from GPIO matrix, 6: SYNC2 from GPIO matrix, other values: no sync input selected. (R/W)

Register 16.15: PWM_OPERATOR_TIMERSEL_REG (0x0038)

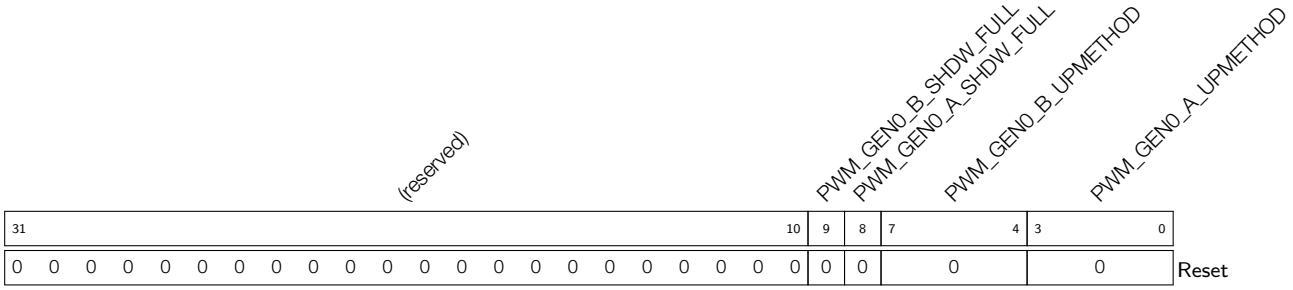


PWM_OPERATOR2_TIMERSEL Select the PWM timer for PWM operator2's timing reference. 0: timer0, 1: timer1, 2: timer2. (R/W)

PWM_OPERATOR1_TIMERSEL Select the PWM timer for PWM operator1's timing reference. 0: timer0, 1: timer1, 2: timer2. (R/W)

PWM_OPERATOR0_TIMERSEL Select the PWM timer for PWM operator0's timing reference. 0: timer0, 1: timer1, 2: timer2. (R/W)

Register 16.16: PWM_GEN0_STMP_CFG_REG (0x003c)



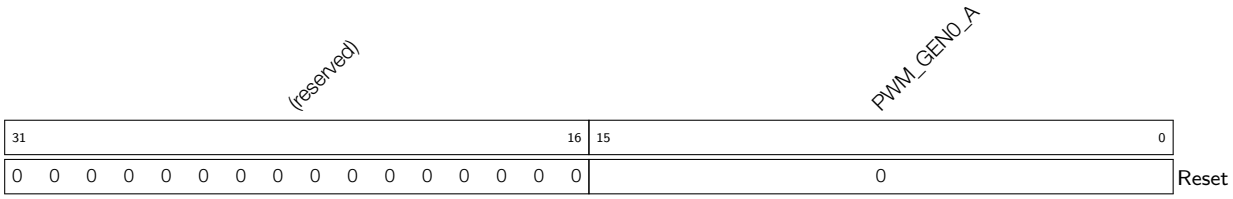
PWM_GEN0_B_SHDW_FULL Set and reset by hardware. If set, PWM generator 0 time stamp B's shadow register.ister is filled and to be transferred to time stamp B's active register. If cleared, time stamp B's active register has been updated with Shadow register latest value. (RO)

PWM_GEN0_A_SHDW_FULL Set and reset by hardware. If set, PWM generator 0 time stamp A's shadow register.ister is filled and to be transferred to time stamp A's active register. If cleared, time stamp A's active register has been updated with Shadow register latest value. (RO)

PWM_GEN0_B_UPMETHOD Updating method for PWM generator 0 time stamp B's active register. When all bits are set to 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

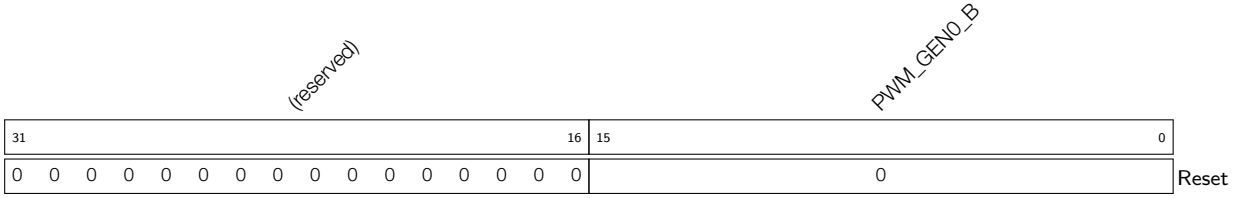
PWM_GEN0_A_UPMETHOD Updating method for PWM generator 0 time stamp A's active register. When all bits are set to 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

Register 16.17: PWM_GEN0_TSTMP_A_REG (0x0040)



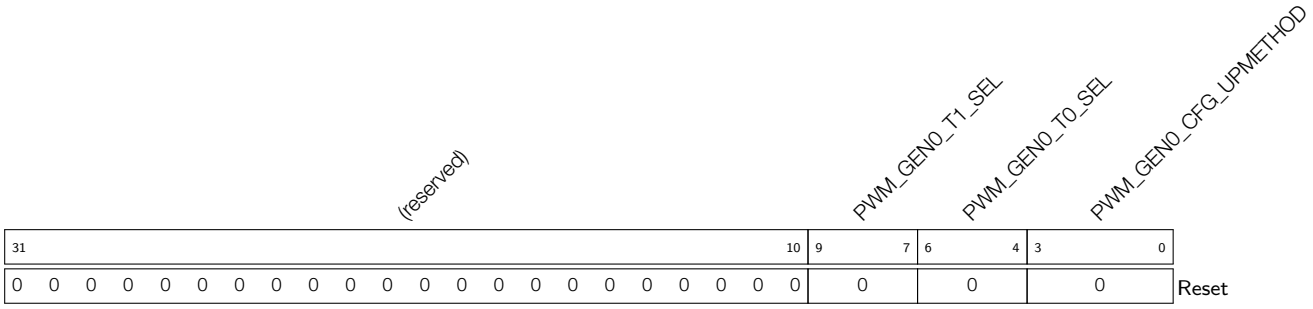
PWM_GEN0_A PWM generator 0 time stamp A's shadow register. (R/W)

Register 16.18: PWM_GEN0_TSTMP_B_REG (0x0044)



PWM_GEN0_B PWM generator 0 time stamp B's shadow register. (R/W)

Register 16.19: PWM_GEN0_CFG0_REG (0x0048)

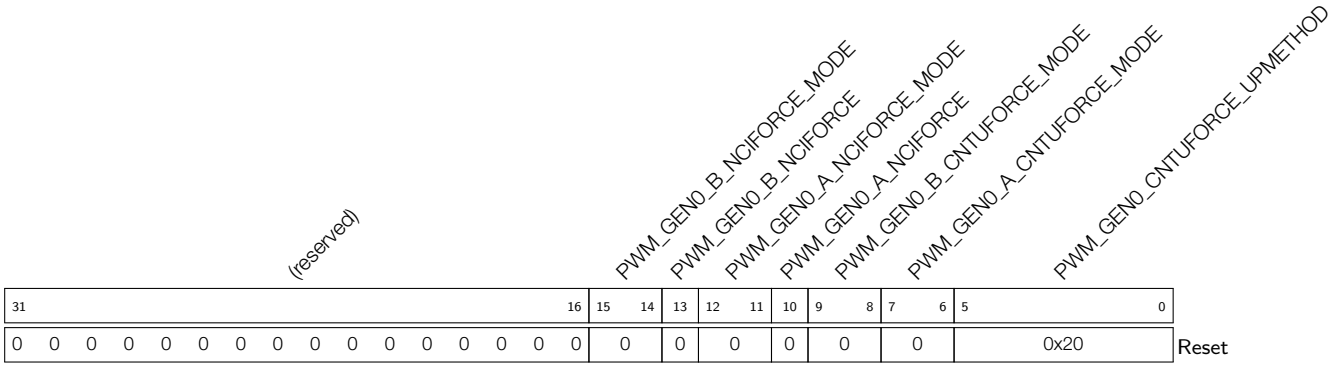


PWM_GEN0_T1_SEL Source selection for PWM generator 0 event_t1, taking effect immediately. 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

PWM_GEN0_T0_SEL Source selection for PWM generator 0 event_t0, taking effect immediately, 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

PWM_GEN0_CFG_UPMETHOD Updating method for PWM generator 0's active register of configuration. When all bits are set to 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

Register 16.20: PWM_GEN0_FORCE_REG (0x004c)



PWM_GEN0_B_NCIFORCE_MODE Non-continuous immediate software-force mode for PWM0B. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

PWM_GEN0_B_NCIFORCE Trigger of non-continuous immediate software-force event for PWM0B; a toggle will trigger a force event. (R/W)

PWM_GEN0_A_NCIFORCE_MODE Non-continuous immediate software-force mode for PWM0A, 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

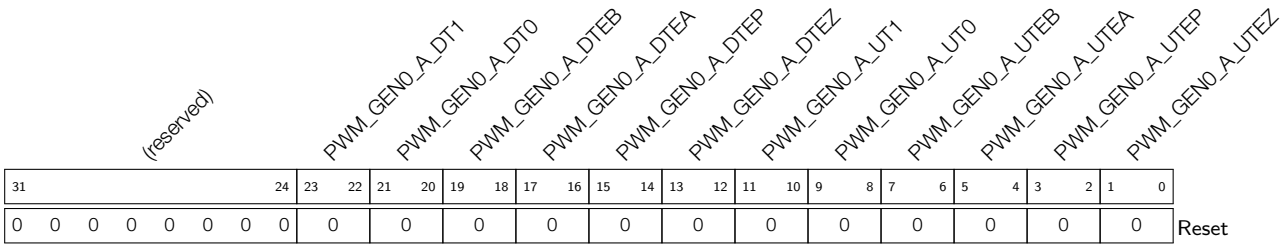
PWM_GEN0_A_NCIFORCE Trigger of non-continuous immediate software-force event for PWM0A; a toggle will trigger a force event. (R/W)

PWM_GEN0_B_CNTUFORCE_MODE Continuous software-force mode for PWM0B. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

PWM_GEN0_A_CNTUFORCE_MODE Continuous software-force mode for PWM0A. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

PWM_GEN0_CNTUFORCE_UPMETHOD Updating method for continuous software force of PWM generator0. When all bits are set to 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: TEA; when bit3 is set to 1: TEB; when bit4 is set to 1: sync; when bit5 is set to 1: disable update. (TEA/B here and below means an event generated when the timer's value equals to that of register A/B.) (R/W)

Register 16.21: PWM_GEN0_A_REG (0x0050)



PWM_GEN0_A_DT1 Action on PWM0A triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

PWM_GEN0_A_DT0 Action on PWM0A triggered by event_t0 when the timer decreases. (R/W)

PWM_GEN0_A_DTEB Action on PWM0A triggered by event TEB when the timer decreases. (R/W)

PWM_GEN0_A_DTEA Action on PWM0A triggered by event TEA when the timer decreases. (R/W)

PWM_GEN0_A_DTEP Action on PWM0A triggered by event TEP when the timer decreases. (R/W)

PWM_GEN0_A_DTEZ Action on PWM0A triggered by event TEZ when the timer decreases. (R/W)

PWM_GEN0_A_UT1 Action on PWM0A triggered by event_t1 when the timer increases. (R/W)

PWM_GEN0_A_UT0 Action on PWM0A triggered by event_t0 when the timer increases. (R/W)

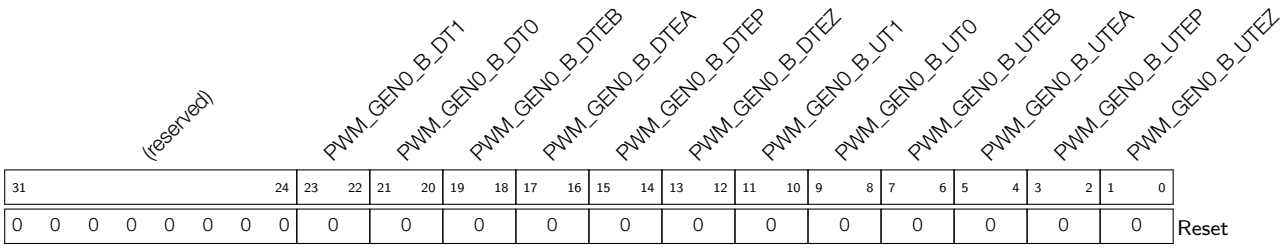
PWM_GEN0_A_UTEB Action on PWM0A triggered by event TEB when the timer increases. (R/W)

PWM_GEN0_A_UTEA Action on PWM0A triggered by event TEA when the timer increases. (R/W)

PWM_GEN0_A_UTEP Action on PWM0A triggered by event TEP when the timer increases. (R/W)

PWM_GEN0_A_UTEZ Action on PWM0A triggered by event TEZ when the timer increases. (R/W)

Register 16.22: PWM_GEN0_B_REG (0x0054)



PWM_GEN0_B_DT1 Action on PWM0B triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

PWM_GEN0_B_DT0 Action on PWM0B triggered by event_t0 when the timer decreases. (R/W)

PWM_GEN0_B_DTEB Action on PWM0B triggered by event TEB when the timer decreases. (R/W)

PWM_GEN0_B_DTEA Action on PWM0B triggered by event TEA when the timer decreases. (R/W)

PWM_GEN0_B_DTEP Action on PWM0B triggered by event TEP when the timer decreases. (R/W)

PWM_GEN0_B_DTEZ Action on PWM0B triggered by event TEZ when the timer decreases. (R/W)

PWM_GEN0_B_UT1 Action on PWM0B triggered by event_t1 when the timer increases. (R/W)

PWM_GEN0_B_UT0 Action on PWM0B triggered by event_t0 when the timer increases. (R/W)

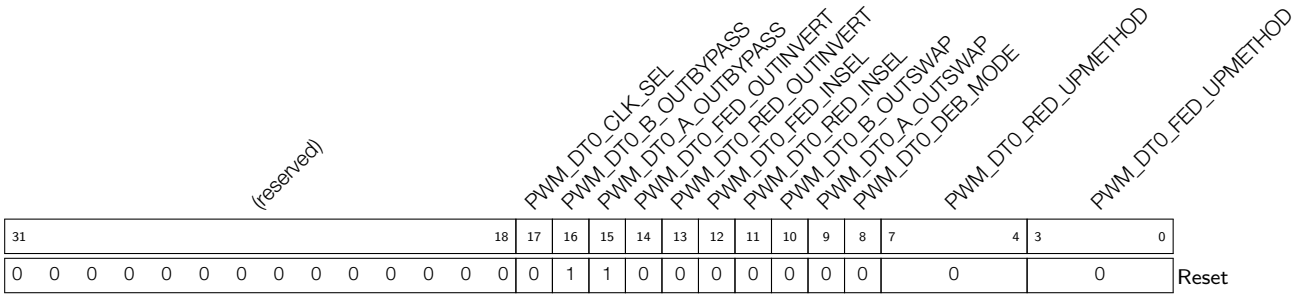
PWM_GEN0_B_UTEB Action on PWM0B triggered by event TEB when the timer increases. (R/W)

PWM_GEN0_B_UTEA Action on PWM0B triggered by event TEA when the timer increases. (R/W)

PWM_GEN0_B_UTEP Action on PWM0B triggered by event TEP when the timer increases. (R/W)

PWM_GEN0_B_UTEZ Action on PWM0B triggered by event TEZ when the timer increases. (R/W)

Register 16.23: PWM_DT0_CFG_REG (0x0058)



PWM_DT0_CLK_SEL Dead time generator 0 clock selection. 0: PWM_clk, 1: PT_clk. (R/W)

PWM_DT0_B_OUTBYPASS S0 in Table 72. (R/W)

PWM_DT0_A_OUTBYPASS S1 in Table 72. (R/W)

PWM_DT0_FED_OUTINVERT S3 in Table 72. (R/W)

PWM_DT0_RED_OUTINVERT S2 in Table 72. (R/W)

PWM_DT0_FED_INSEL S5 in Table 72. (R/W)

PWM_DT0_RED_INSEL S4 in Table 72. (R/W)

PWM_DT0_B_OUTSWAP S7 in Table 72. (R/W)

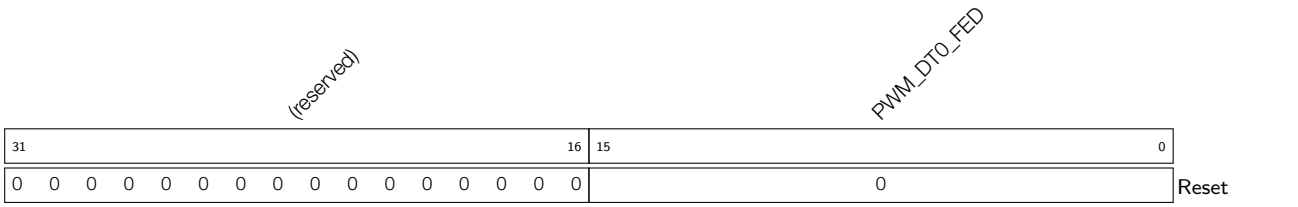
PWM_DT0_A_OUTSWAP S6 in Table 72. (R/W)

PWM_DT0_DEB_MODE S8 in Table 72, dual-edge B mode. 0: FED/RED take effect on different paths separately, 1: FED/RED take effect on B path. (R/W)

PWM_DT0_RED_UPMETHOD Updating method for RED (rising edge delay) active register. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

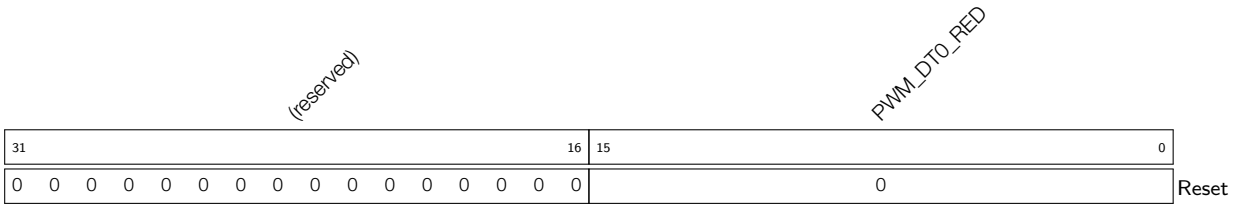
PWM_DT0_FED_UPMETHOD Updating method for FED (falling edge delay) active register. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

Register 16.24: PWM_DT0_FED_CFG_REG (0x005c)



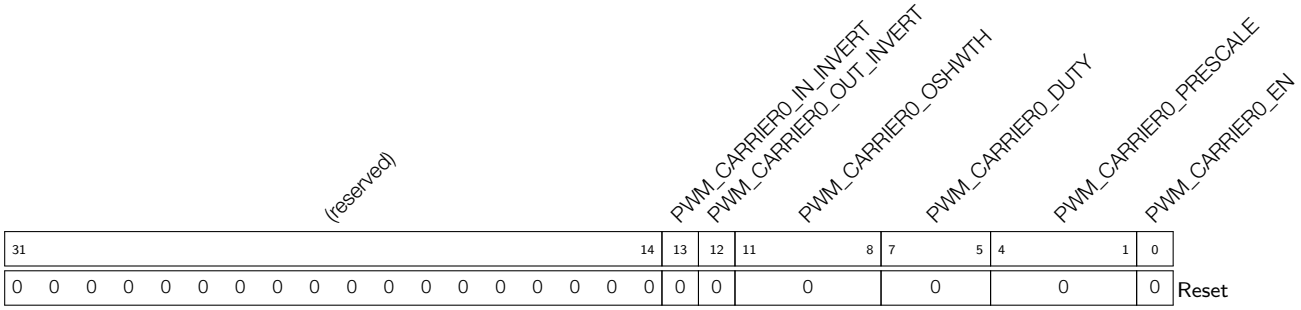
PWM_DT0_FED Shadow register for FED. (R/W)

Register 16.25: PWM_DT0_RED_CFG_REG (0x0060)



PWM_DT0_RED Shadow register for RED. (R/W)

Register 16.26: PWM_CARRIER0_CFG_REG (0x0064)



PWM_CARRIER0_IN_INVERT When set, invert the input of PWM0A and PWM0B for this submodule. (R/W)

PWM_CARRIER0_OUT_INVERT When set, invert the output of PWM0A and PWM0B for this submodule. (R/W)

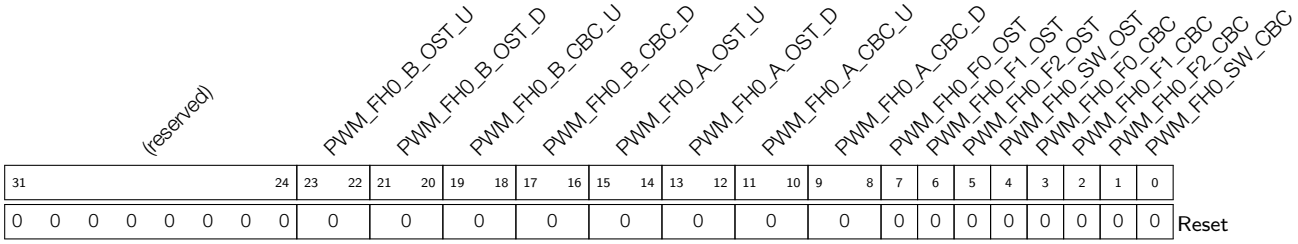
PWM_CARRIER0_OSHWTH Width of the first pulse in number of periods of the carrier. (R/W)

PWM_CARRIER0_DUTY Carrier duty selection. Duty = PWM_CARRIER0_DUTY/8. (R/W)

PWM_CARRIER0_PRESCALE PWM carrier0 clock (PC_clk) prescale value. Period of PC_clk = period of PWM_clk * (PWM_CARRIER0_PRESCALE + 1). (R/W)

PWM_CARRIER0_EN When set, carrier0 function is enabled. When cleared, carrier0 is bypassed. (R/W)

Register 16.27: PWM_FH0_CFG0_REG (0x0068)



PWM_FH0_B_OST_U One-shot mode action on PWM0B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH0_B_OST_D One-shot mode action on PWM0B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH0_B_CBC_U Cycle-by-cycle mode action on PWM0B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH0_B_CBC_D Cycle-by-cycle mode action on PWM0B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH0_A_OST_U One-shot mode action on PWM0A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH0_A_OST_D One-shot mode action on PWM0A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH0_A_CBC_U Cycle-by-cycle mode action on PWM0A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH0_A_CBC_D Cycle-by-cycle mode action on PWM0A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH0_F0_OST event_f0 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

PWM_FH0_F1_OST event_f1 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

PWM_FH0_F2_OST event_f2 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

PWM_FH0_SW_OST Enable register for software-forced one-shot mode action. 0: disable, 1: enable. (R/W)

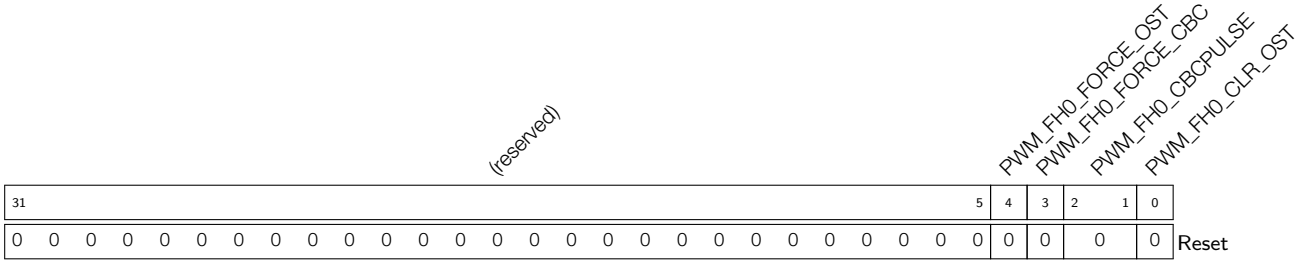
PWM_FH0_F0_CBC event_f0 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

PWM_FH0_F1_CBC event_f1 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

PWM_FH0_F2_CBC event_f2 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

PWM_FH0_SW_CBC Enable register for software-forced cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

Register 16.28: PWM_FH0_CFG1_REG (0x006c)



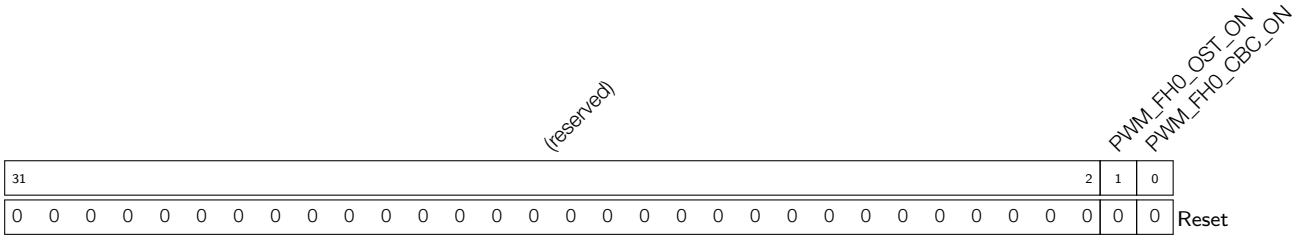
PWM_FH0_FORCE_OST A toggle (software negation of this bit's value) triggers a one-shot mode action. (R/W)

PWM_FH0_FORCE_CBC A toggle triggers a cycle-by-cycle mode action. (R/W)

PWM_FH0_CBCPULSE The cycle-by-cycle mode action refresh moment selection. When bit0 is set to 1: TEZ; when bit1 is set to 1: TEP. (R/W)

PWM_FH0_CLR_OST A toggle will clear on-going one-shot mode action. (R/W)

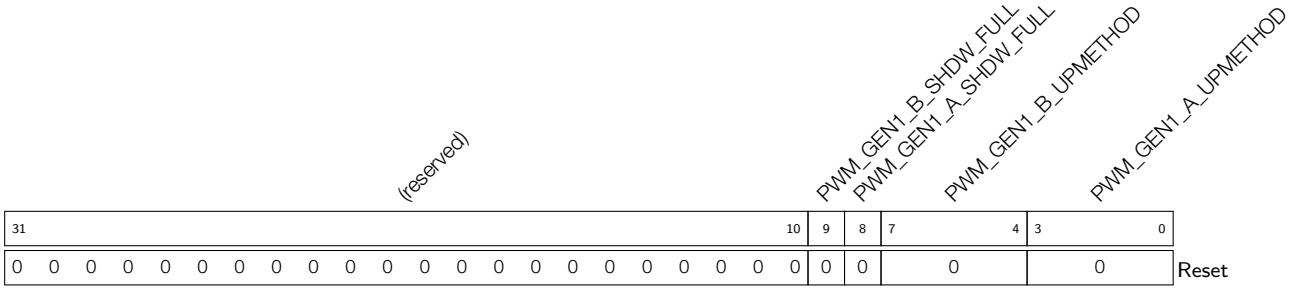
Register 16.29: PWM_FH0_STATUS_REG (0x0070)



PWM_FH0_OST_ON Set and reset by hardware. If set, a one-shot mode action is on-going. (RO)

PWM_FH0_CBC_ON Set and reset by hardware. If set, a cycle-by-cycle mode action is on-going. (RO)

Register 16.30: PWM_GEN1_STMP_CFG_REG (0x0074)



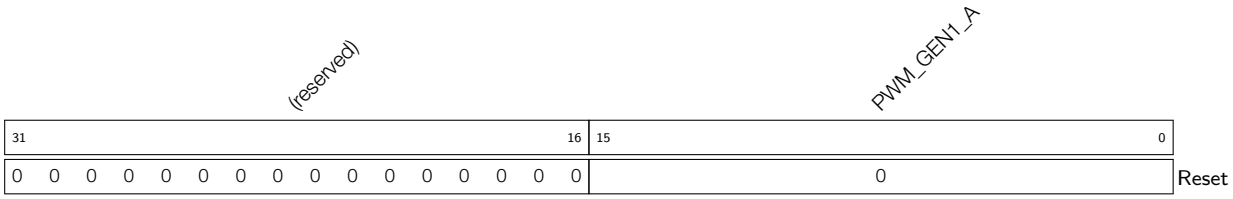
PWM_GEN1_B_SHDW_FULL Set and reset by hardware. If set, PWM generator 1 time stamp B's shadow register is filled and to be transferred to time stamp B's active register. If cleared, time stamp B's active register has been updated with shadow register's latest value. (RO)

PWM_GEN1_A_SHDW_FULL Set and reset by hardware. If set, PWM generator 1 time stamp A's shadow register is filled and to be transferred to time stamp A's active register. If cleared, time stamp A's active register has been updated with shadow register latest value. (RO)

PWM_GEN1_B_UPMETHOD Updating method for PWM generator 1 time stamp B's active register. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

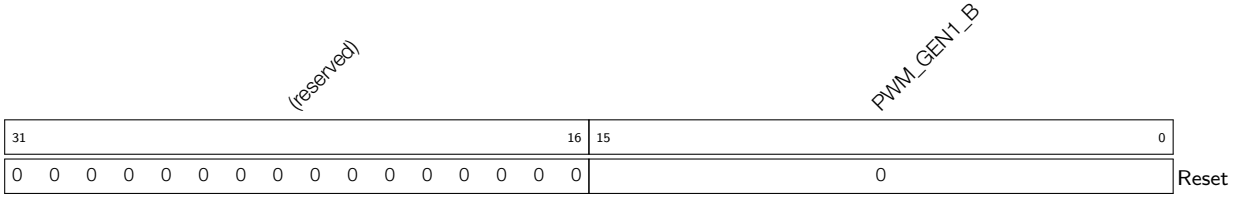
PWM_GEN1_A_UPMETHOD Updating method for PWM generator 1 time stamp A's active register. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

Register 16.31: PWM_GEN1_TSTMP_A_REG (0x0078)



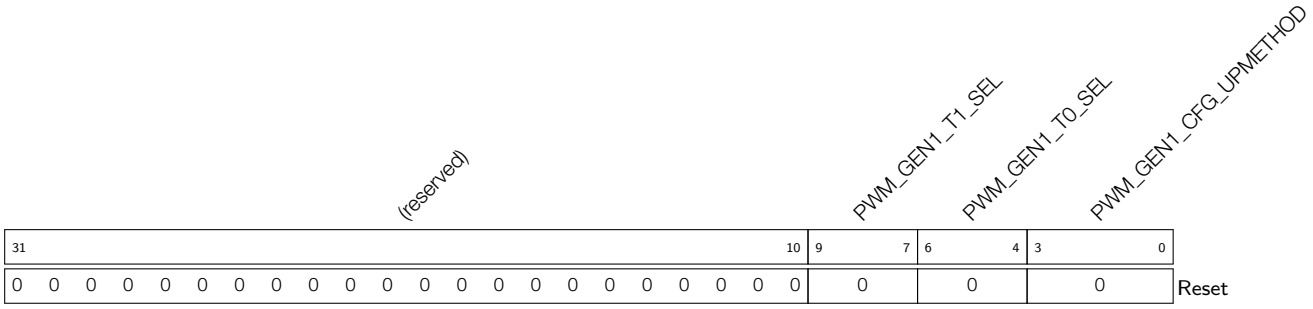
PWM_GEN1_A PWM generator 1 time stamp A's shadow register. (R/W)

Register 16.32: PWM_GEN1_TSTMP_B_REG (0x007c)



PWM_GEN1_B PWM generator 1 time stamp B's shadow register. (R/W)

Register 16.33: PWM_GEN1_CFG0_REG (0x0080)

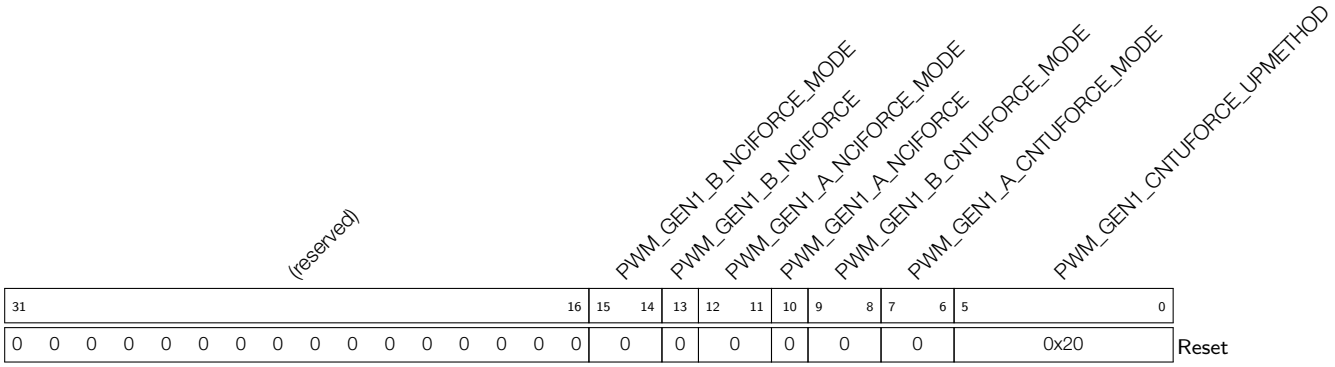


PWM_GEN1_T1_SEL Source selection for PWM generator1 event_t1, taking effect immediately, 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

PWM_GEN1_T0_SEL Source selection for PWM generator1 event_t0, taking effect immediately, 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

PWM_GEN1_CFG_UPMETHOD Updating method for PWM generator1's active register of configuration. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync. bit3: disable the update. (R/W)

Register 16.34: PWM_GEN1_FORCE_REG (0x0084)



PWM_GEN1_B_NCIFORCE_MODE Non-continuous immediate software-force mode for PWM1B. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

PWM_GEN1_B_NCIFORCE Trigger of non-continuous immediate software-force event for PWM1B; a toggle will trigger a force event. (R/W)

PWM_GEN1_A_NCIFORCE_MODE Non-continuous immediate software-force mode for PWM1A. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

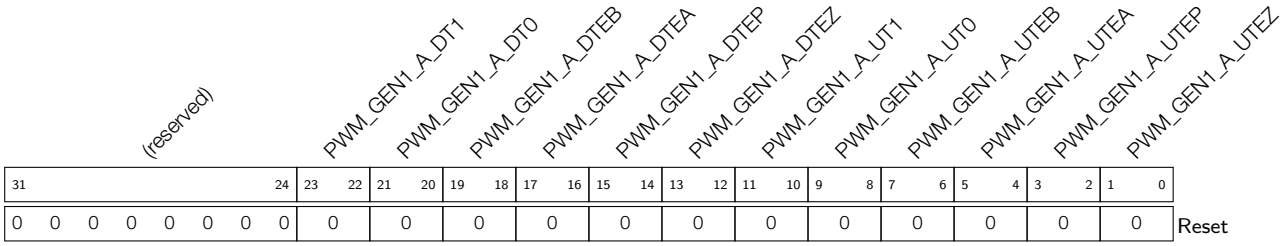
PWM_GEN1_A_NCIFORCE Trigger of non-continuous immediate software-force event for PWM1A; a toggle will trigger a force event. (R/W)

PWM_GEN1_B_CNTUFORCE_MODE Continuous software-force mode for PWM1B. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

PWM_GEN1_A_CNTUFORCE_MODE Continuous software-force mode for PWM1A. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

PWM_GEN1_CNTUFORCE_UPMETHOD Updating method for continuous software force of PWM generator1. When all bits are set to 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: TEA; when bit3 is set to 1: TEB; when bit4 is set to 1: sync; when bit5 is set to 1: disable update. (TEA/B here and below means an event generated when the timer's value equals to that of register A/B). (R/W)

Register 16.35: PWM_GEN1_A_REG (0x0088)



PWM_GEN1_A_DT1 Action on PWM1A triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

PWM_GEN1_A_DT0 Action on PWM1A triggered by event_t0 when the timer decreases. (R/W)

PWM_GEN1_A_DTEB Action on PWM1A triggered by event TEB when the timer decreases. (R/W)

PWM_GEN1_A_DTEA Action on PWM1A triggered by event TEA when the timer decreases. (R/W)

PWM_GEN1_A_DTEP Action on PWM1A triggered by event TEP when the timer decreases. (R/W)

PWM_GEN1_A_DTEZ Action on PWM1A triggered by event TEZ when the timer decreases. (R/W)

PWM_GEN1_A_UT1 Action on PWM1A triggered by event_t1 when the timer increases. (R/W)

PWM_GEN1_A_UT0 Action on PWM1A triggered by event_t0 when the timer increases. (R/W)

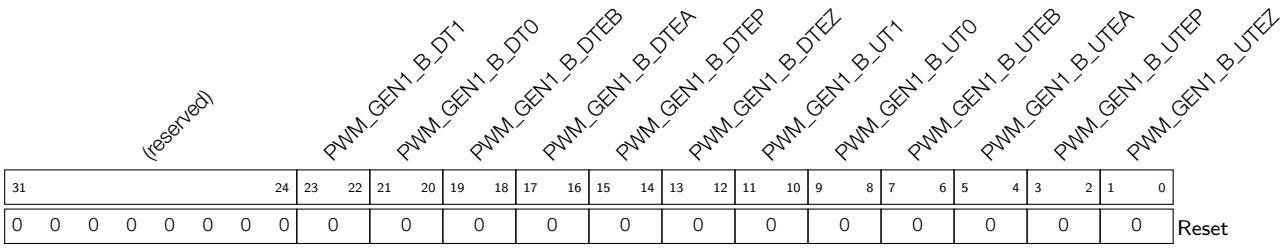
PWM_GEN1_A_UTEB Action on PWM1A triggered by event TEB when the timer increases. (R/W)

PWM_GEN1_A_UTEA Action on PWM1A triggered by event TEA when the timer increases. (R/W)

PWM_GEN1_A_UTEP Action on PWM1A triggered by event TEP when the timer increases. (R/W)

PWM_GEN1_A_UTEZ Action on PWM1A triggered by event TEZ when the timer increases. (R/W)

Register 16.36: PWM_GEN1_B_REG (0x008c)



PWM_GEN1_B_DT1 Action on PWM1B triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

PWM_GEN1_B_DT0 Action on PWM1B triggered by event_t0 when the timer decreases. (R/W)

PWM_GEN1_B_DTEB Action on PWM1B triggered by event TEB when the timer decreases. (R/W)

PWM_GEN1_B_DTEA Action on PWM1B triggered by event TEA when the timer decreases. (R/W)

PWM_GEN1_B_DTEP Action on PWM1B triggered by event TEP when the timer decreases. (R/W)

PWM_GEN1_B_DTEZ Action on PWM1B triggered by event TEZ when the timer decreases. (R/W)

PWM_GEN1_B_UT1 Action on PWM1B triggered by event_t1 when the timer increases. (R/W)

PWM_GEN1_B_UT0 Action on PWM1B triggered by event_t0 when the timer increases. (R/W)

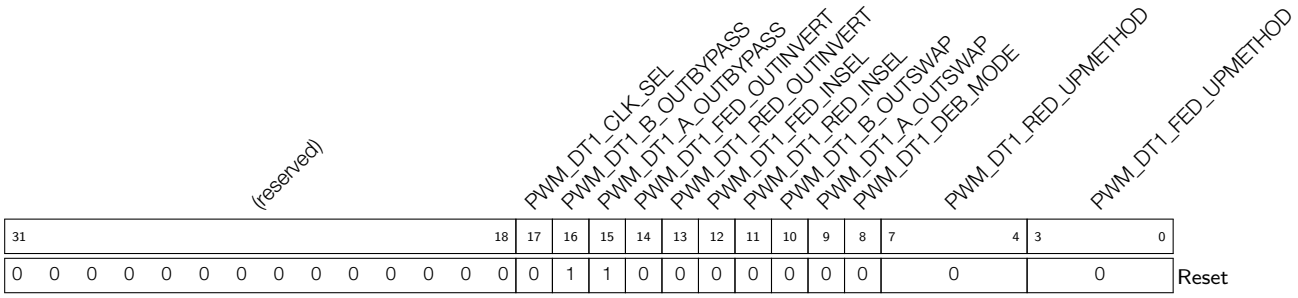
PWM_GEN1_B_UTEB Action on PWM1B triggered by event TEB when the timer increases. (R/W)

PWM_GEN1_B_UTEA Action on PWM1B triggered by event TEA when the timer increases. (R/W)

PWM_GEN1_B_UTEP Action on PWM1B triggered by event TEP when the timer increases. (R/W)

PWM_GEN1_B_UTEZ Action on PWM1B triggered by event TEZ when the timer increases. (R/W)

Register 16.37: PWM_DT1_CFG_REG (0x0090)



PWM_DT1_CLK_SEL Dead time generator 1 clock selection. 0: PWM_clk, 1: PT_clk. (R/W)

PWM_DT1_B_OUTBYPASS S0 in Table 72. (R/W)

PWM_DT1_A_OUTBYPASS S1 in Table 72. (R/W)

PWM_DT1_FED_OUTINVERT S3 in Table 72. (R/W)

PWM_DT1_RED_OUTINVERT S2 in Table 72. (R/W)

PWM_DT1_FED_INSEL S5 in Table 72. (R/W)

PWM_DT1_RED_INSEL S4 in Table 72. (R/W)

PWM_DT1_B_OUTSWAP S7 in Table 72. (R/W)

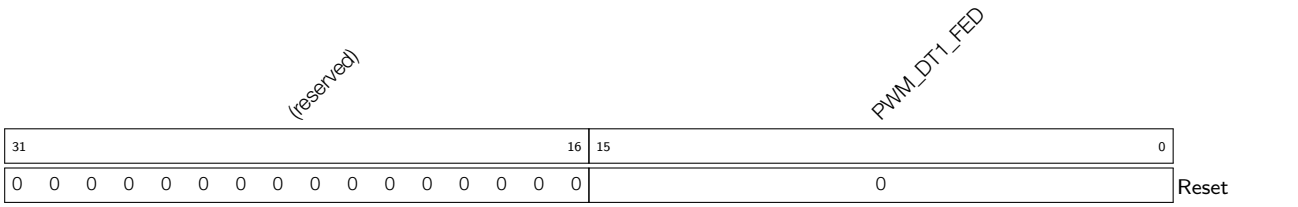
PWM_DT1_A_OUTSWAP S6 in Table 72. (R/W)

PWM_DT1_DEB_MODE S8 in Table 72; dual-edge B mode. 0: FED/RED take effect on different paths separately; 1: FED (falling edge delay)/RED (rising edge delay) take effect on B path. (R/W)

PWM_DT1_RED_UPMETHOD Updating method for RED active register. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

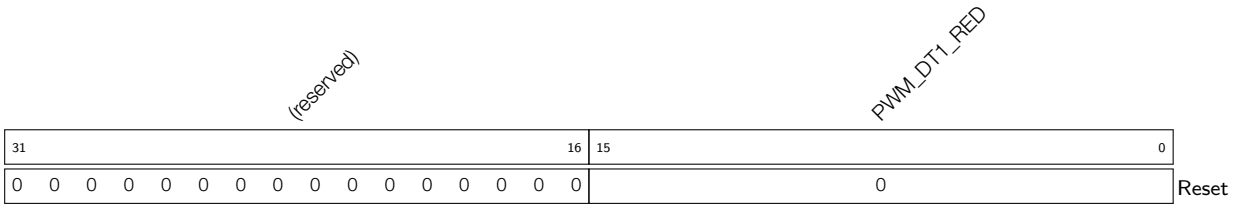
PWM_DT1_FED_UPMETHOD Updating method for FED active register. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

Register 16.38: PWM_DT1_FED_CFG_REG (0x0094)



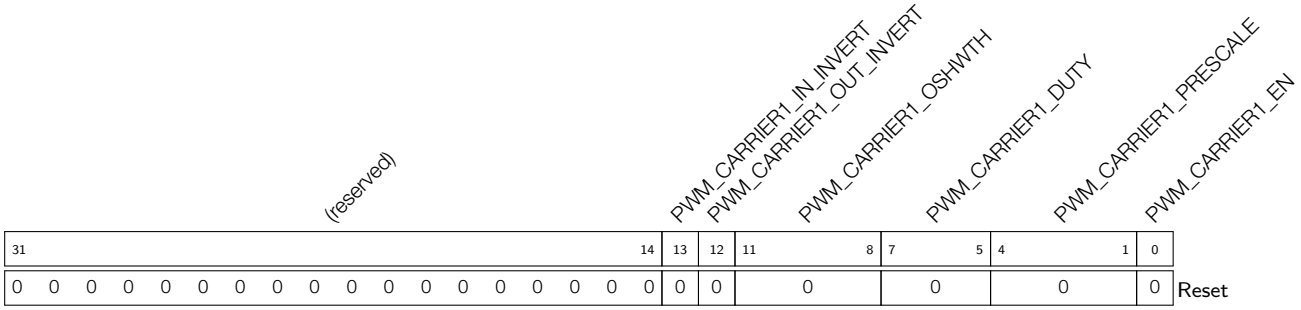
PWM_DT1_FED Shadow register for FED. (R/W)

Register 16.39: PWM_DT1_RED_CFG_REG (0x0098)



PWM_DT1_RED Shadow register for RED. (R/W)

Register 16.40: PWM_CARRIER1_CFG_REG (0x009c)



PWM_CARRIER1_IN_INVERT When set, invert the input of PWM1A and PWM1B for this submodule. (R/W)

PWM_CARRIER1_OUT_INVERT When set, invert the output of PWM1A and PWM1B for this submodule. (R/W)

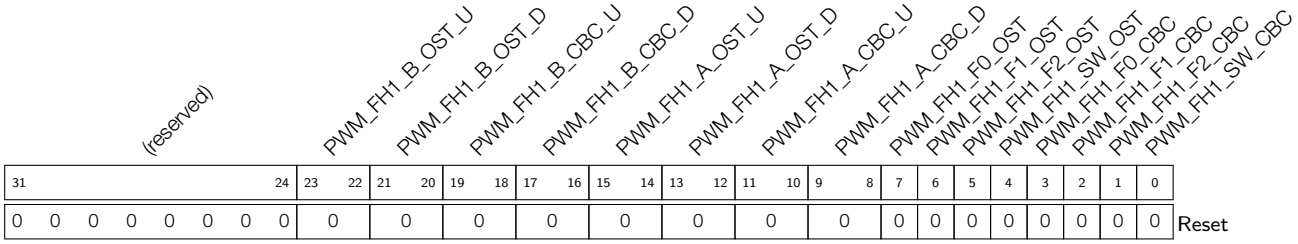
PWM_CARRIER1_OSHWTH Width of the first pulse in number of periods of the carrier. (R/W)

PWM_CARRIER1_DUTY Carrier duty selection. Duty = PWM_CARRIER1_DUTY/8. (R/W)

PWM_CARRIER1_PRESCALE PWM carrier1 clock (PC_clk) prescale value. Period of PC_clk = period of PWM_clk * (PWM_CARRIER1_PRESCALE + 1). (R/W)

PWM_CARRIER1_EN When set, carrier1 function is enabled. When cleared, carrier1 is bypassed. (R/W)

Register 16.41: PWM_FH1_CFG0_REG (0x00a0)



PWM_FH1_B_OST_U One-shot mode action on PWM1B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH1_B_OST_D One-shot mode action on PWM1B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH1_B_CBC_U Cycle-by-cycle mode action on PWM1B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH1_B_CBC_D Cycle-by-cycle mode action on PWM1B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH1_A_OST_U One-shot mode action on PWM1A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH1_A_OST_D One-shot mode action on PWM1A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH1_A_CBC_U Cycle-by-cycle mode action on PWM1A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH1_A_CBC_D Cycle-by-cycle mode action on PWM1A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH1_F0_OST Enable event_f0 to trigger one-shot mode action. 0: disable, 1: enable. (R/W)

PWM_FH1_F1_OST Enable event_f1 to trigger one-shot mode action. 0: disable, 1: enable. (R/W)

PWM_FH1_F2_OST Enable event_f2 to trigger one-shot mode action. 0: disable, 1: enable. (R/W)

PWM_FH1_SW_OST Enable the register for software-forced one-shot mode action. 0: disable, 1: enable. (R/W)

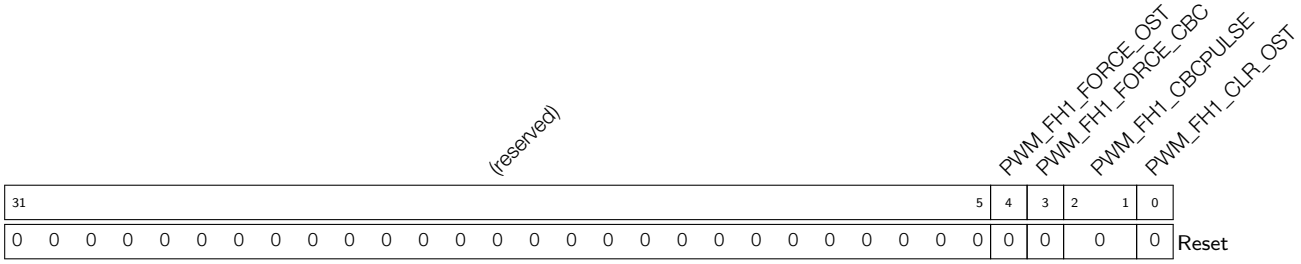
PWM_FH1_F0_CBC Enable event_f0 to trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

PWM_FH1_F1_CBC Enable event_f1 to trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

PWM_FH1_F2_CBC Enable event_f2 to will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

PWM_FH1_SW_CBC Enable the register for software-forced cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

Register 16.42: PWM_FH1_CFG1_REG (0x00a4)



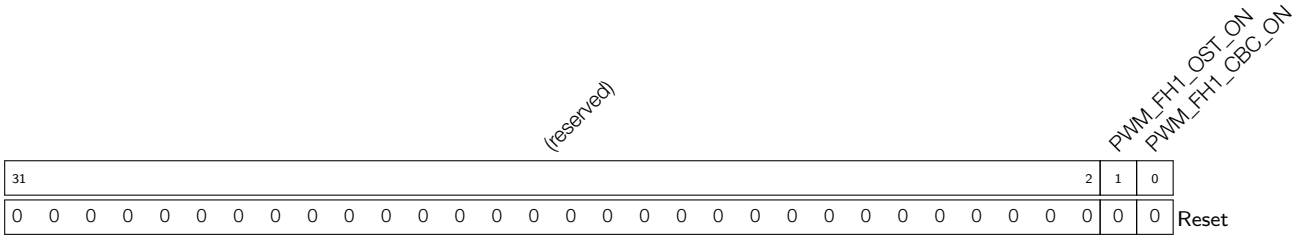
PWM_FH1_FORCE_OST A toggle (software negation of this bit's value) triggers a one-shot mode action. (R/W)

PWM_FH1_FORCE_CBC A toggle triggers a cycle-by-cycle mode action. (R/W)

PWM_FH1_CBCPULSE The cycle-by-cycle mode action refresh moment selection. When bit0 is set to 1: TEZ; when bit1 is set to 1: TEP. (R/W)

PWM_FH1_CLR_OST A toggle will clear on-going one-shot mode action. (R/W)

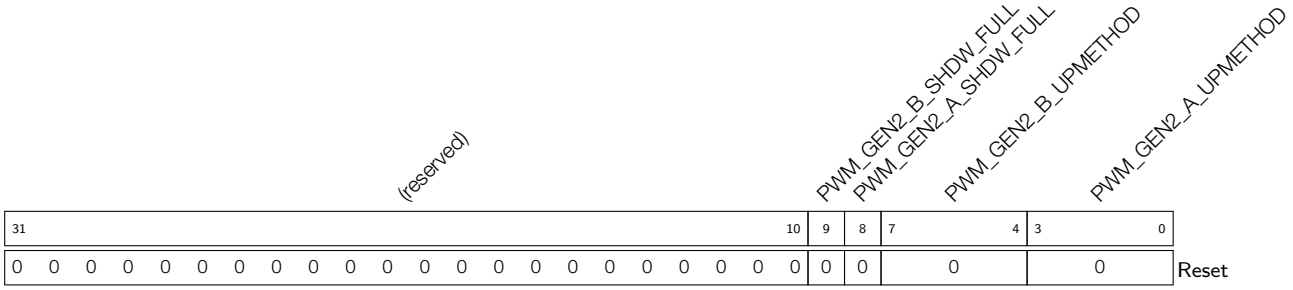
Register 16.43: PWM_FH1_STATUS_REG (0x00a8)



PWM_FH1_OST_ON Set and reset by hardware. If set, a one-shot mode action is on-going. (RO)

PWM_FH1_CBC_ON Set and reset by hardware. If set, a cycle-by-cycle mode action is on-going. (RO)

Register 16.44: PWM_GEN2_STMP_CFG_REG (0x00ac)



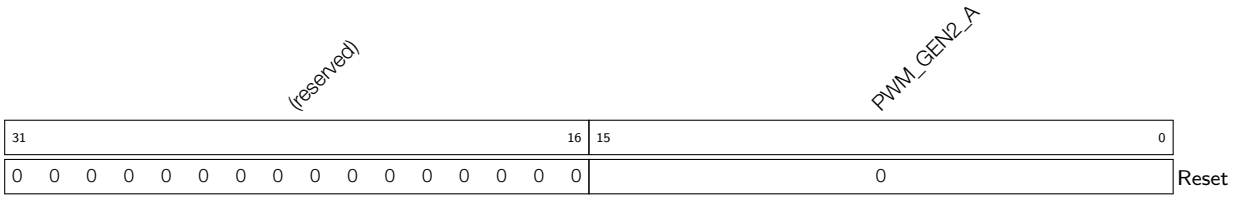
PWM_GEN2_B_SHDW_FULL Set and reset by hardware. If set, PWM generator 2 time stamp B's shadow register is filled and to be transferred to time stamp B's active register. If cleared, time stamp B's active register has been updated with shadow register's latest value. (RO)

PWM_GEN2_A_SHDW_FULL Set and reset by hardware. If set, PWM generator 2 time stamp A's shadow register is filled and to be transferred to time stamp A's active register. If cleared, time stamp A's active register has been updated with shadow register's latest value. (RO)

PWM_GEN2_B_UPMETHOD Updating method for PWM generator 2 time stamp B's active register. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

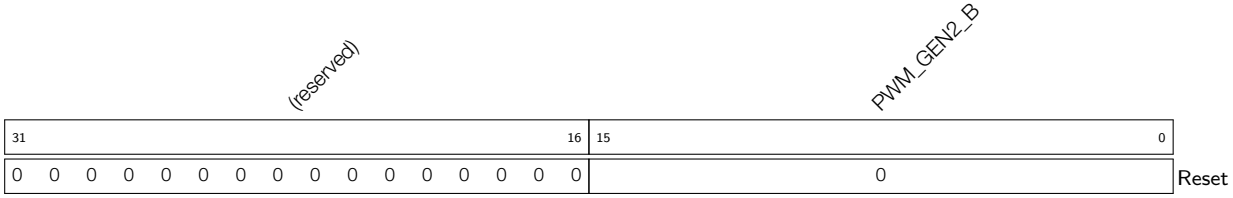
PWM_GEN2_A_UPMETHOD Updating method for PWM generator 2 time stamp A's active register. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

Register 16.45: PWM_GEN2_TSTMP_A_REG (0x00b0)



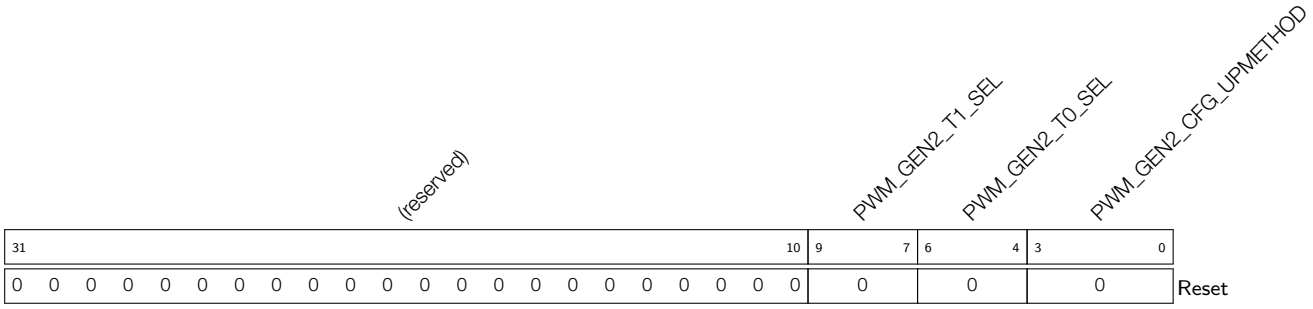
PWM_GEN2_A PWM generator 2 time stamp A's shadow register. (R/W)

Register 16.46: PWM_GEN2_TSTMP_B_REG (0x00b4)



PWM_GEN2_B PWM generator 2 time stamp B's shadow register. (R/W)

Register 16.47: PWM_GEN2_CFG0_REG (0x00b8)

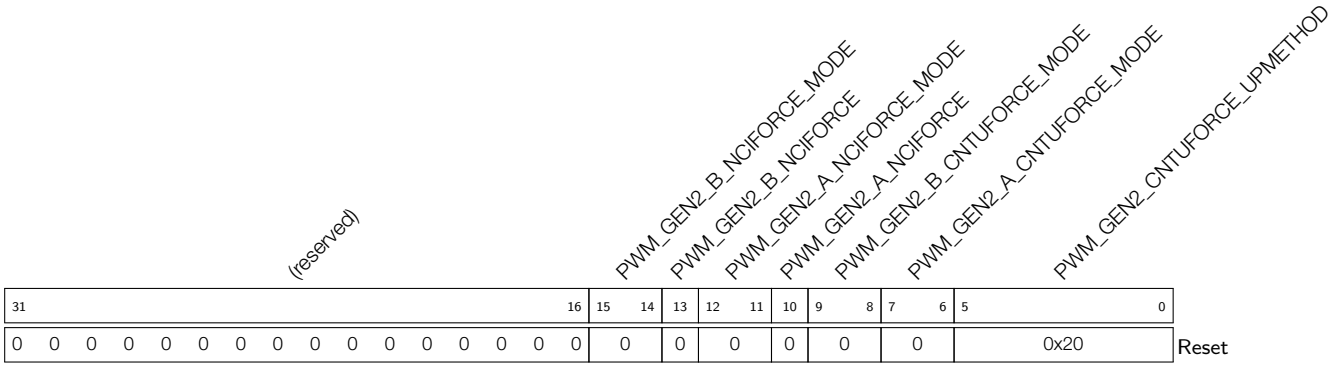


PWM_GEN2_T1_SEL Source selection for PWM generator2 event_t1, take effect immediately, 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

PWM_GEN2_T0_SEL Source selection for PWM generator2 event_t0, take effect immediately, 0: fault_event0, 1: fault_event1, 2: fault_event2, 3: sync_taken, 4: none. (R/W)

PWM_GEN2_CFG_UPMETHOD Updating method for PWM generator2's active register of configuration. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync. bit3: disable the update. (R/W)

Register 16.48: PWM_GEN2_FORCE_REG (0x00bc)



PWM_GEN2_B_NCIFORCE_MODE Non-continuous immediate software-force mode for PWM2B, 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

PWM_GEN2_B_NCIFORCE Trigger of non-continuous immediate software-force event for PWM2B, a toggle will trigger a force event. (R/W)

PWM_GEN2_A_NCIFORCE_MODE Non-continuous immediate software-force mode for PWM2A, 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

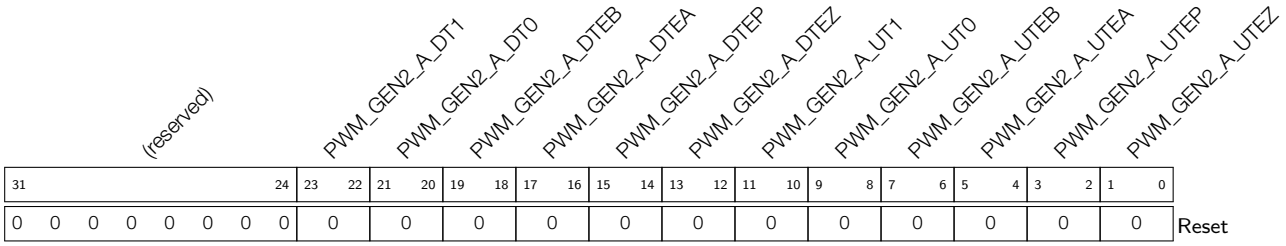
PWM_GEN2_A_NCIFORCE Trigger of non-continuous immediate software-force event for PWM2A, a toggle will trigger a force event. (R/W)

PWM_GEN2_B_CNTUFORCE_MODE Continuous software-force mode for PWM2B. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

PWM_GEN2_A_CNTUFORCE_MODE Continuous software-force mode for PWM2A. 0: disabled, 1: low, 2: high, 3: disabled. (R/W)

PWM_GEN2_CNTUFORCE_UPMETHOD Updating method for continuous software force of PWM generator2. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: TEA; when bit3 is set to 1: TEB; when bit4 is set to 1: sync; when bit5 is set to 1: disable update. (TEA/B here and below means an event generated when the timer value equals that of register A/B.) (R/W)

Register 16.49: PWM_GEN2_A_REG (0x00c0)



PWM_GEN2_A_DT1 Action on PWM2A triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

PWM_GEN2_A_DT0 Action on PWM2A triggered by event_t0 when the timer decreases. (R/W)

PWM_GEN2_A_DTEB Action on PWM2A triggered by event TEB when the timer decreases. (R/W)

PWM_GEN2_A_DTEA Action on PWM2A triggered by event TEA when the timer decreases. (R/W)

PWM_GEN2_A_DTEP Action on PWM2A triggered by event TEP when the timer decreases. (R/W)

PWM_GEN2_A_DTEZ Action on PWM2A triggered by event TEZ when the timer decreases. (R/W)

PWM_GEN2_A_UT1 Action on PWM2A triggered by event_t1 when the timer increases. (R/W)

PWM_GEN2_A_UT0 Action on PWM2A triggered by event_t0 when the timer increases. (R/W)

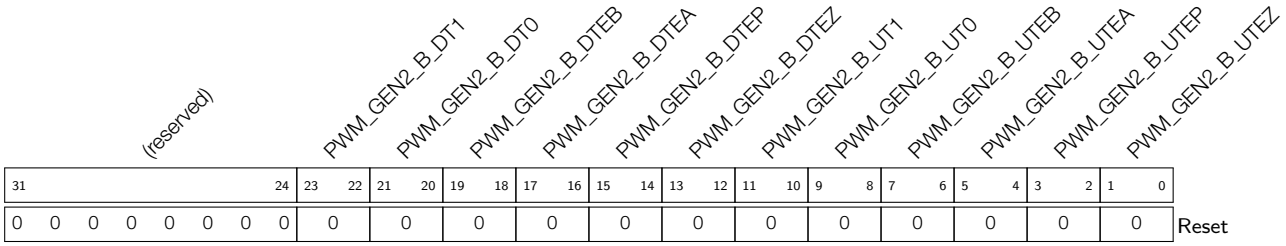
PWM_GEN2_A_UTEB Action on PWM2A triggered by event TEB when the timer increases. (R/W)

PWM_GEN2_A_UTEA Action on PWM2A triggered by event TEA when the timer increases. (R/W)

PWM_GEN2_A_UTEP Action on PWM2A triggered by event TEP when the timer increases. (R/W)

PWM_GEN2_A_UTEZ Action on PWM2A triggered by event TEZ when the timer increases. (R/W)

Register 16.50: PWM_GEN2_B_REG (0x00c4)



PWM_GEN2_B_DT1 Action on PWM2B triggered by event_t1 when the timer decreases. 0: no change, 1: low, 2: high, 3: toggle. (R/W)

PWM_GEN2_B_DT0 Action on PWM2B triggered by event_t0 when the timer decreases. (R/W)

PWM_GEN2_B_DTEB Action on PWM2B triggered by event TEB when the timer decreases. (R/W)

PWM_GEN2_B_DTEA Action on PWM2B triggered by event TEA when the timer decreases. (R/W)

PWM_GEN2_B_DTEP Action on PWM2B triggered by event TEP when the timer decreases. (R/W)

PWM_GEN2_B_DTEZ Action on PWM2B triggered by event TEZ when the timer decreases. (R/W)

PWM_GEN2_B_UT1 Action on PWM2B triggered by event_t1 when the timer increases. (R/W)

PWM_GEN2_B_UT0 Action on PWM2B triggered by event_t0 when the timer increases. (R/W)

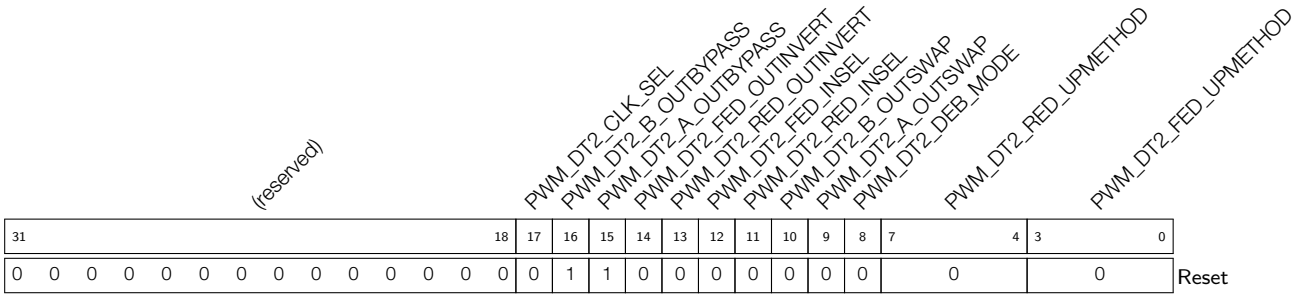
PWM_GEN2_B_UTEB Action on PWM2B triggered by event TEB when the timer increases. (R/W)

PWM_GEN2_B_UTEA Action on PWM2B triggered by event TEA when the timer increases. (R/W)

PWM_GEN2_B_UTEP Action on PWM2B triggered by event TEP when the timer increases. (R/W)

PWM_GEN2_B_UTEZ Action on PWM2B triggered by event TEZ when the timer increases. (R/W)

Register 16.51: PWM_DT2_CFG_REG (0x00c8)



PWM_DT2_CLK_SEL Dead time generator 1 clock selection. 0: PWM_clk; 1: PT_clk. (R/W)

PWM_DT2_B_OUTBYPASS S0 in Table 72. (R/W)

PWM_DT2_A_OUTBYPASS S1 in Table 72. (R/W)

PWM_DT2_FED_OUTINVERT S3 in Table 72. (R/W)

PWM_DT2_RED_OUTINVERT S2 in Table 72. (R/W)

PWM_DT2_FED_INSEL S5 in Table 72. (R/W)

PWM_DT2_RED_INSEL S4 in Table 72. (R/W)

PWM_DT2_B_OUTSWAP S7 in Table 72. (R/W)

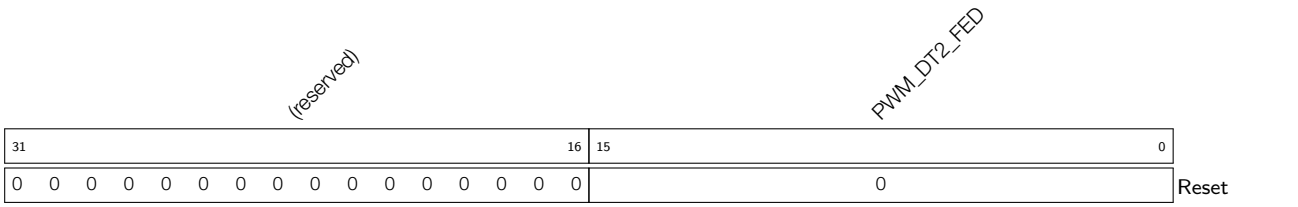
PWM_DT2_A_OUTSWAP S6 in Table 72. (R/W)

PWM_DT2_DEB_MODE S8 in Table 72, dual-edge B mode, 0: FED/RED take effect on different path separately, 1: FED/RED take effect on B path. (R/W)

PWM_DT2_RED_UPMETHOD Updating method for RED (rising edge delay) active register. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

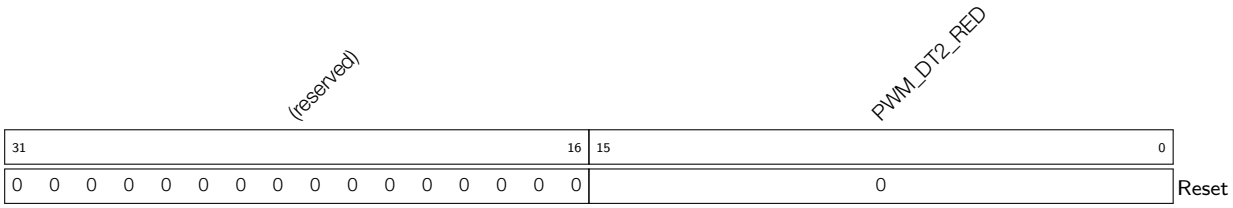
PWM_DT2_FED_UPMETHOD Updating method for FED (falling edge delay) active register. 0: immediately; when bit0 is set to 1: TEZ; when bit1 is set to 1: TEP; when bit2 is set to 1: sync; when bit3 is set to 1: disable the update. (R/W)

Register 16.52: PWM_DT2_FED_CFG_REG (0x00cc)



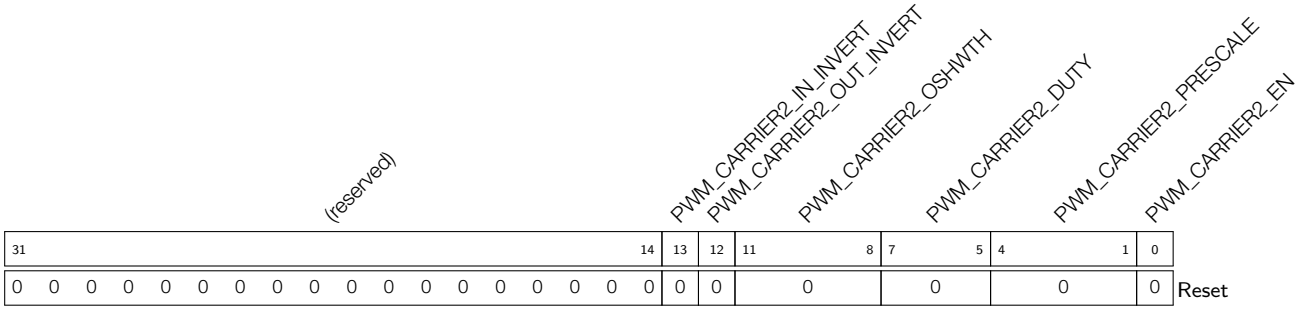
PWM_DT2_FED Shadow register for FED. (R/W)

Register 16.53: PWM_DT2_RED_CFG_REG (0x00d0)



PWM_DT2_RED Shadow register for RED. (R/W)

Register 16.54: PWM_CARRIER2_CFG_REG (0x00d4)



PWM_CARRIER2_IN_INVERT When set, invert the input of PWM2A and PWM2B for this submodule. (R/W)

PWM_CARRIER2_OUT_INVERT When set, invert the output of PWM2A and PWM2B for this submodule. (R/W)

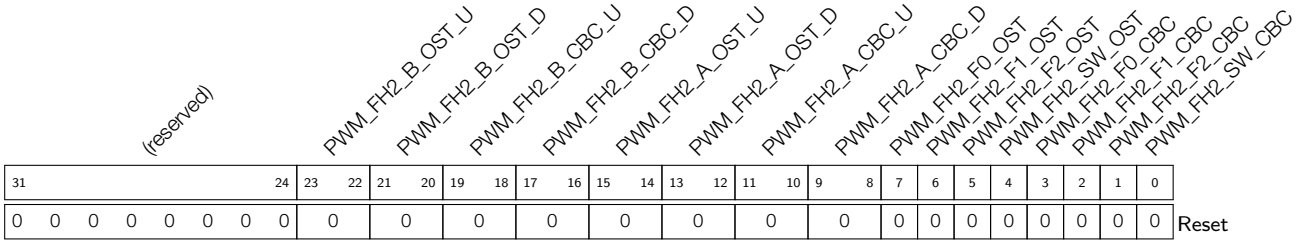
PWM_CARRIER2_OSHWTH Width of the first pulse in number of periods of the carrier. (R/W)

PWM_CARRIER2_DUTY Carrier duty selection. Duty = PWM_CARRIER2_DUTY / 8. (R/W)

PWM_CARRIER2_PRESCALE PWM carrier2 clock (PC_clk) prescale value. Period of PC_clk = period of PWM_clk * (PWM_CARRIER2_PRESCALE + 1). (R/W)

PWM_CARRIER2_EN When set, carrier2 function is enabled. When cleared, carrier2 is bypassed. (R/W)

Register 16.55: PWM_FH2_CFG0_REG (0x00d8)



PWM_FH2_B_OST_U One-shot mode action on PWM2B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH2_B_OST_D One-shot mode action on PWM2B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH2_B_CBC_U Cycle-by-cycle mode action on PWM2B when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH2_B_CBC_D Cycle-by-cycle mode action on PWM2B when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH2_A_OST_U One-shot mode action on PWM2A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH2_A_OST_D One-shot mode action on PWM2A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH2_A_CBC_U Cycle-by-cycle mode action on PWM2A when a fault event occurs and the timer is increasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH2_A_CBC_D Cycle-by-cycle mode action on PWM2A when a fault event occurs and the timer is decreasing. 0: do nothing, 1: force low, 2: force high, 3: toggle. (R/W)

PWM_FH2_F0_OST event_f0 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

PWM_FH2_F1_OST event_f1 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

PWM_FH2_F2_OST event_f2 will trigger one-shot mode action. 0: disable, 1: enable. (R/W)

PWM_FH2_SW_OST Enable register for software-forced one-shot mode action. 0: disable, 1: enable. (R/W)

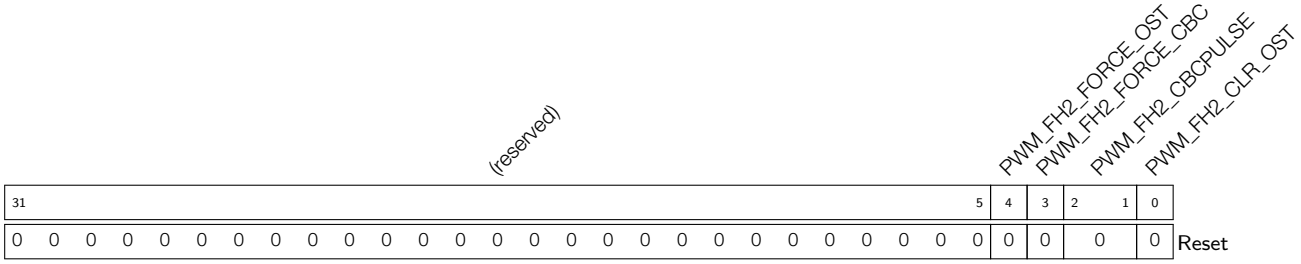
PWM_FH2_F0_CBC event_f0 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

PWM_FH2_F1_CBC event_f1 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

PWM_FH2_F2_CBC event_f2 will trigger cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

PWM_FH2_SW_CBC Enable register for software-forced cycle-by-cycle mode action. 0: disable, 1: enable. (R/W)

Register 16.56: PWM_FH2_CFG1_REG (0x00dc)



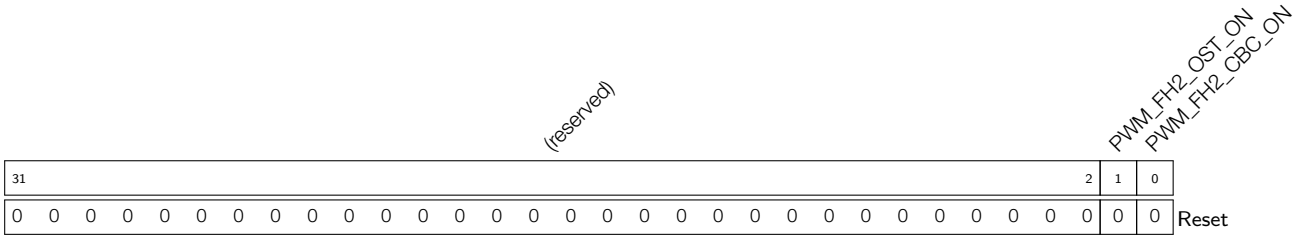
PWM_FH2_FORCE_OST A toggle (software negation of this bit's value) triggers a one-shot mode action. (R/W)

PWM_FH2_FORCE_CBC A toggle triggers a cycle-by-cycle mode action. (R/W)

PWM_FH2_CBCPULSE The cycle-by-cycle mode action refresh moment selection. When bit0 is set to 1: TEZ; when bit1 is set to 1:TEP. (R/W)

PWM_FH2_CLR_OST A toggle will clear on-going one-shot mode action. (R/W)

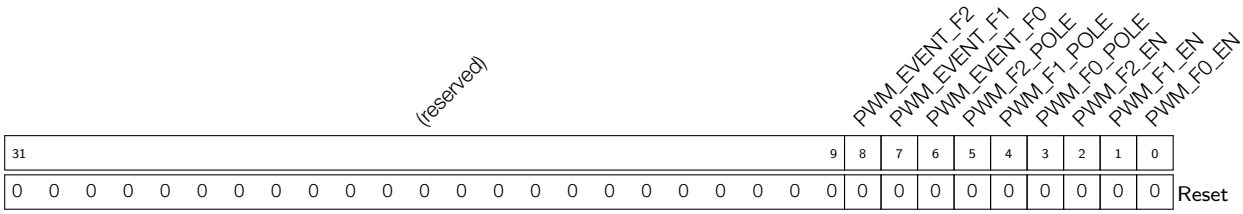
Register 16.57: PWM_FH2_STATUS_REG (0x00e0)



PWM_FH2_OST_ON Set and reset by hardware. If set, a one-shot mode action is on-going. (RO)

PWM_FH2_CBC_ON Set and reset by hardware. If set, a cycle-by-cycle mode action is on-going. (RO)

Register 16.58: PWM_FAULT_DETECT_REG (0x00e4)



PWM_EVENT_F2 Set and reset by hardware. If set, event_f2 is on-going. (RO)

PWM_EVENT_F1 Set and reset by hardware. If set, event_f1 is on-going. (RO)

PWM_EVENT_F0 Set and reset by hardware. If set, event_f0 is on-going. (RO)

PWM_F2_POLE Set event_f2 trigger polarity on FAULT2 source from GPIO matrix. 0: level low, 1: level high. (R/W)

PWM_F1_POLE Set event_f1 trigger polarity on FAULT2 source from GPIO matrix. 0: level low, 1: level high. (R/W)

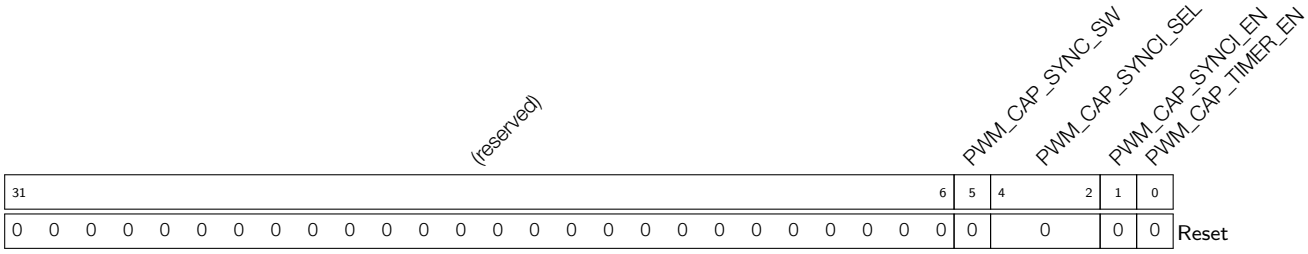
PWM_F0_POLE Set event_f0 trigger polarity on FAULT2 source from GPIO matrix. 0: level low, 1: level high. (R/W)

PWM_F2_EN Set to enable the generation of event_f2. (R/W)

PWM_F1_EN Set to enable the generation of event_f1. (R/W)

PWM_F0_EN Set to enable the generation of event_f0. (R/W)

Register 16.59: PWM_CAP_TIMER_CFG_REG (0x00e8)



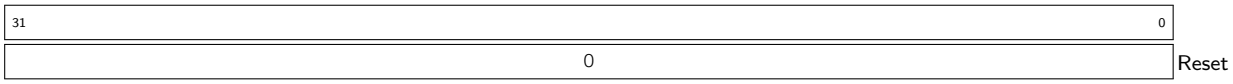
PWM_CAP_SYNC_SW Set this bit to force a capture timer sync; the capture timer is loaded with the value in the phase register. (WO)

PWM_CAP_SYNCI_SEL Capture module sync input selection. 0: none, 1: timer0 sync_out, 2: timer1 sync_out, 3: timer2 sync_out, 4: SYNC0 from GPIO matrix, 5: SYNC1 from GPIO matrix, 6: SYNC2 from GPIO matrix. (R/W)

PWM_CAP_SYNCI_EN When set, the capture timer sync is enabled. (R/W)

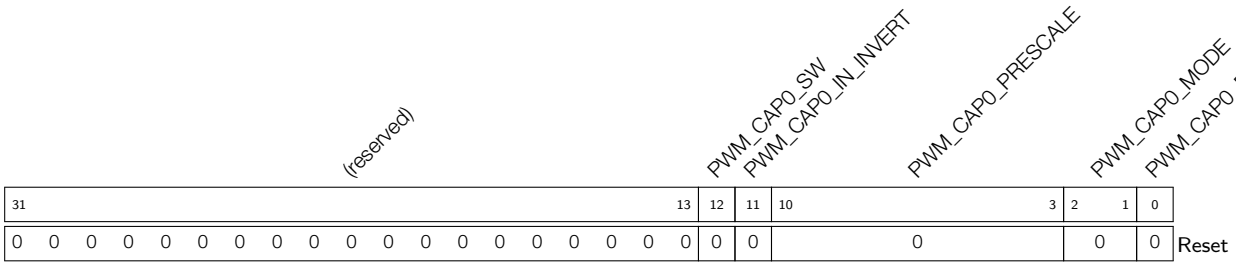
PWM_CAP_TIMER_EN When set, the capture timer incrementing under APB_clk is enabled. (R/W)

Register 16.60: PWM_CAP_TIMER_PHASE_REG (0x00ec)



PWM_CAP_TIMER_PHASE_REG Phase value for the capture timer sync operation. (R/W)

Register 16.61: PWM_CAP_CH0_CFG_REG (0x00f0)



PWM_CAP0_SW When set, a software-forced capture on channel 0 is triggered. (WO)

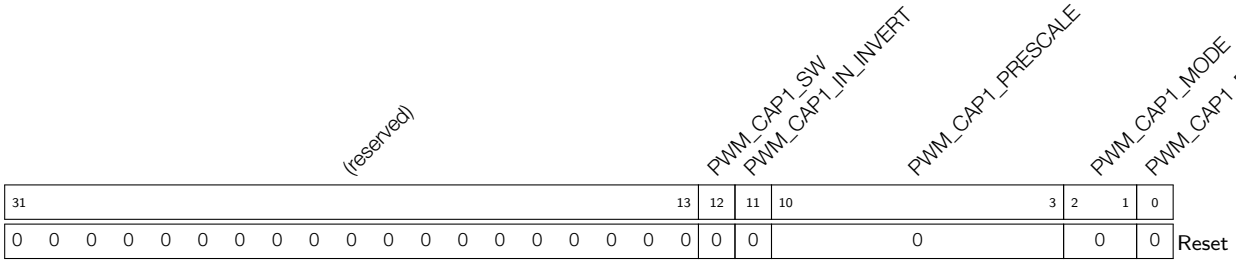
PWM_CAP0_IN_INVERT When set, CAP0 form GPIO matrix is inverted before prescaling. (R/W)

PWM_CAP0_PRESCALE Prescaling value on the positive edge of CAP0. Prescaling value = PWM_CAP0_PRESCALE + 1. (R/W)

PWM_CAP0_MODE Edge of capture on channel 0 after prescaling. When bit0 is set to 1: enable capture on the negative edge; When bit1 is set to 1: enable capture on the positive edge. (R/W)

PWM_CAP0_EN When set, capture on channel 0 is enabled. (R/W)

Register 16.62: PWM_CAP_CH1_CFG_REG (0x00f4)



PWM_CAP1_SW Write 1 will trigger a software-forced capture on channel 1. (WO)

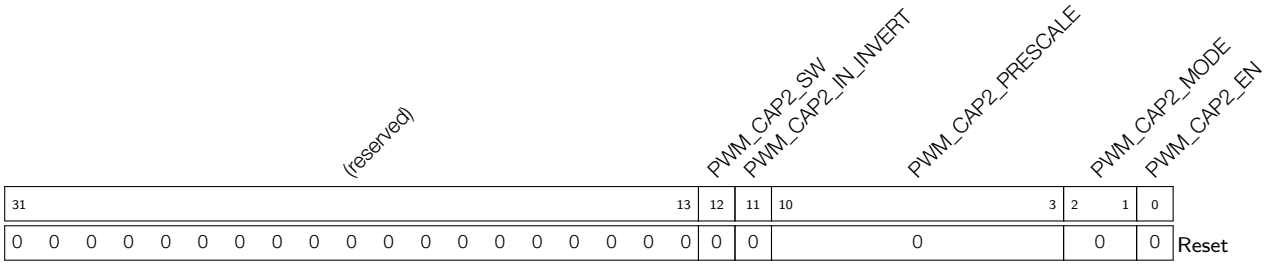
PWM_CAP1_IN_INVERT When set, CAP1 form GPIO matrix is inverted before prescaling. (R/W)

PWM_CAP1_PRESCALE Value of prescale on the positive edge of CAP1. Prescale value = PWM_CAP1_PRESCALE + 1. (R/W)

PWM_CAP1_MODE Edge of capture on channel 1 after prescaling. When bit0 is set to 1: enable capture on the negative edge; When bit1 is set to 1: enable capture on the positive edge. (R/W)

PWM_CAP1_EN When set, capture on channel 1 is enabled. (R/W)

Register 16.63: PWM_CAP_CH2_CFG_REG (0x00f8)



PWM_CAP2_SW When set, a software-forced capture on channel 2 is triggered. (WO)

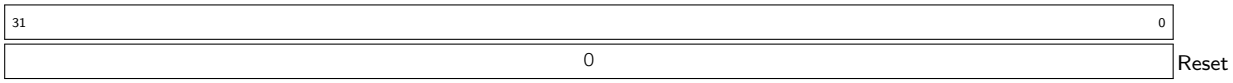
PWM_CAP2_IN_INVERT When set, CAP2 form GPIO matrix is inverted before prescaling. (R/W)

PWM_CAP2_PRESCALE Prescaling value on the positive edge of CAP2. Prescaling value = PWM_CAP2_PRESCALE + 1. (R/W)

PWM_CAP2_MODE Edge of capture on channel 2 after prescaling. When bit0 is set to 1: enable capture on the negative edge; when bit1 is set to 1: enable capture on the positive edge. (R/W)

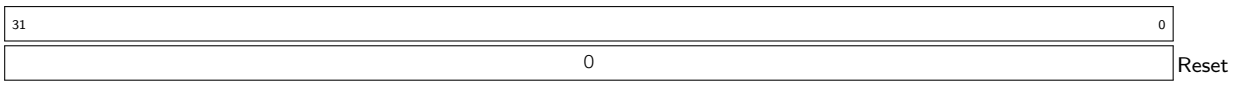
PWM_CAP2_EN When set, capture on channel 2 is enabled. (R/W)

Register 16.64: PWM_CAP_CH0_REG (0x00fc)



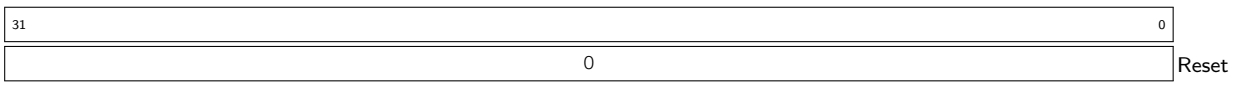
PWM_CAP_CH0_REG Value of the last capture on channel 0. (RO)

Register 16.65: PWM_CAP_CH1_REG (0x0100)



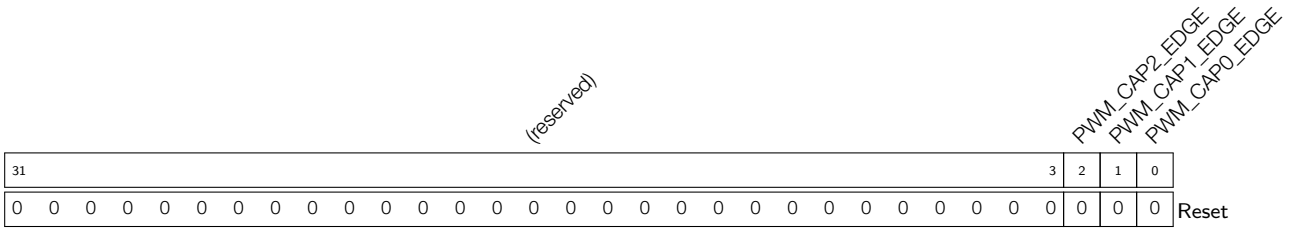
PWM_CAP_CH1_REG Value of the last capture on channel 1. (RO)

Register 16.66: PWM_CAP_CH2_REG (0x0104)



PWM_CAP_CH2_REG Value of the last capture on channel 2. (RO)

Register 16.67: PWM_CAP_STATUS_REG (0x0108)

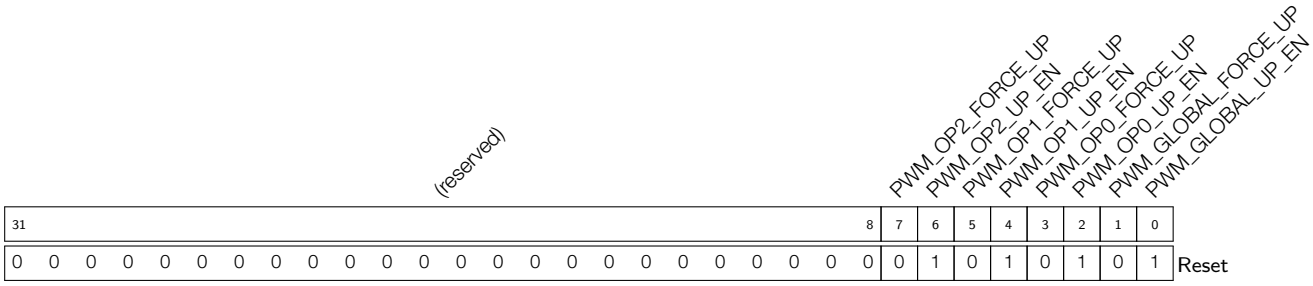


PWM_CAP2_EDGE Edge of the last capture trigger on channel 2. 0: posedge; 1: negedge. (RO)

PWM_CAP1_EDGE Edge of the last capture trigger on channel 1. 0: posedge; 1: negedge. (RO)

PWM_CAP0_EDGE Edge of the last capture trigger on channel 0. 0: posedge; 1: negedge. (RO)

Register 16.68: PWM_UPDATE_CFG_REG (0x010c)



PWM_OP2_FORCE_UP A toggle (software negation of this bit's value) will trigger a forced update of active registers in PWM operator 2. (R/W)

PWM_OP2_UP_EN When set and PWM_GLOBAL_UP_EN is set, update of active registers in PWM operator 2 are enabled (R/W)

PWM_OP1_FORCE_UP A toggle (software negation of this bit's value) will trigger a forced update of active registers in PWM operator 1. (R/W)

PWM_OP1_UP_EN When set and PWM_GLOBAL_UP_EN is set, update of active registers in PWM operator 1 are enabled. (R/W)

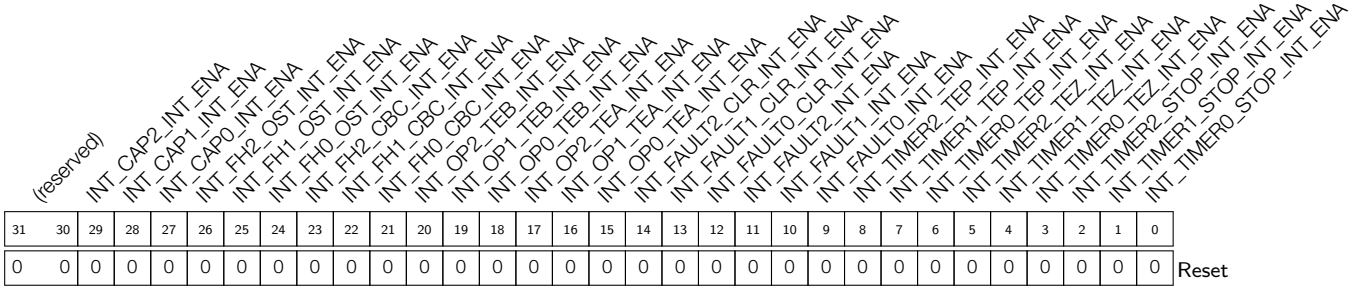
PWM_OP0_FORCE_UP A toggle (software negation of this bit's value) will trigger a forced update of active registers in PWM operator 0. (R/W)

PWM_OP0_UP_EN When set and PWM_GLOBAL_UP_EN is set, update of active registers in PWM operator 0 are enabled. (R/W)

PWM_GLOBAL_FORCE_UP A toggle (software negation of this bit's value) will trigger a forced update of all active registers in the MCPWM module. (R/W)

PWM_GLOBAL_UP_EN The global enable of update of all active registers in the MCPWM module. (R/W)

Register 16.69: INT_ENA_PWM_REG (0x0110)



INT_CAP2_INT_ENA The enable bit for the interrupt triggered by capture on channel 2. (R/W)

INT_CAP1_INT_ENA The enable bit for the interrupt triggered by capture on channel 1. (R/W)

INT_CAP0_INT_ENA The enable bit for the interrupt triggered by capture on channel 0. (R/W)

INT_FH2_OST_INT_ENA The enable bit for the interrupt triggered by a one-shot mode action on PWM2. (R/W)

INT_FH1_OST_INT_ENA The enable bit for the interrupt triggered by a one-shot mode action on PWM0. (R/W)

INT_FH0_OST_INT_ENA The enable bit for the interrupt triggered by a one-shot mode action on PWM0. (R/W)

INT_FH2_CBC_INT_ENA The enable bit for the interrupt triggered by a cycle-by-cycle mode action on PWM2. (R/W)

INT_FH1_CBC_INT_ENA The enable bit for the interrupt triggered by a cycle-by-cycle mode action on PWM1. (R/W)

INT_FH0_CBC_INT_ENA The enable bit for the interrupt triggered by a cycle-by-cycle mode action on PWM0. (R/W)

INT_OP2_TEB_INT_ENA The enable bit for the interrupt triggered by a PWM operator 2 TEB event (R/W)

INT_OP1_TEB_INT_ENA The enable bit for the interrupt triggered by a PWM operator 1 TEB event (R/W)

INT_OP0_TEB_INT_ENA The enable bit for the interrupt triggered by a PWM operator 0 TEB event (R/W)

INT_OP2_TEA_INT_ENA The enable bit for the interrupt triggered by a PWM operator 2 TEA event (R/W)

INT_OP1_TEA_INT_ENA The enable bit for the interrupt triggered by a PWM operator 1 TEA event (R/W)

INT_OP0_TEA_INT_ENA The enable bit for the interrupt triggered by a PWM operator 0 TEA event (R/W)

INT_FAULT2_CLR_INT_ENA The enable bit for the interrupt triggered when event_f2 ends. (R/W)

INT_FAULT1_CLR_INT_ENA The enable bit for the interrupt triggered when event_f1 ends. (R/W)

INT_FAULT0_CLR_INT_ENA The enable bit for the interrupt triggered when event_f0 ends. (R/W)

INT_FAULT2_INT_ENA The enable bit for the interrupt triggered when event_f2 starts. (R/W)

INT_FAULT1_INT_ENA The enable bit for the interrupt triggered when event_f1 starts. (R/W)

INT_FAULT0_INT_ENA The enable bit for the interrupt triggered when event_f0 starts. (R/W)

INT_TIMER2_TEP_INT_ENA The enable bit for the interrupt triggered by a PWM timer 2 TEP event. (R/W)

INT_TIMER1_TEP_INT_ENA The enable bit for the interrupt triggered by a PWM timer 1 TEP event. (R/W)

INT_TIMER0_TEP_INT_ENA The enable bit for the interrupt triggered by a PWM timer 0 TEP event. (R/W)

INT_TIMER2_TEZ_INT_ENA The enable bit for the interrupt triggered by a PWM timer 2 TEZ event. (R/W)

INT_TIMER1_TEZ_INT_ENA The enable bit for the interrupt triggered by a PWM timer 1 TEZ event. (R/W)

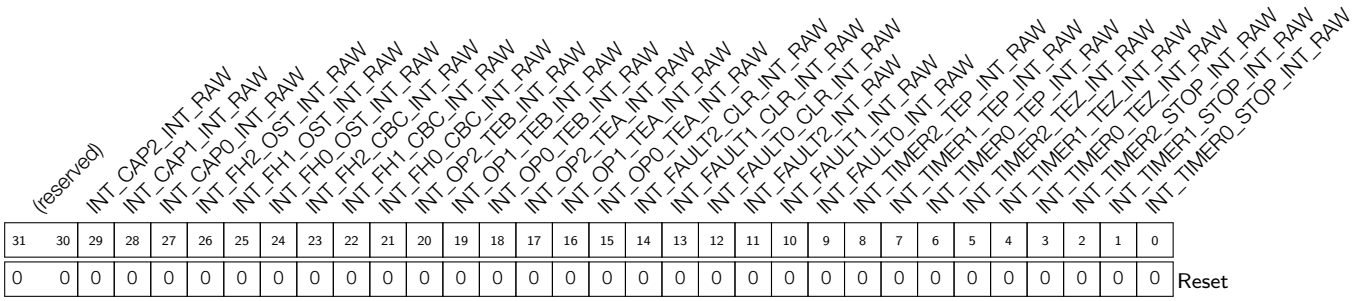
INT_TIMER0_TEZ_INT_ENA The enable bit for the interrupt triggered by a PWM timer 0 TEZ event. (R/W)

INT_TIMER2_STOP_INT_ENA The enable bit for the interrupt triggered when the timer 2 stops. (R/W)

INT_TIMER1_STOP_INT_ENA The enable bit for the interrupt triggered when the timer 1 stops. (R/W)

INT_TIMER0_STOP_INT_ENA The enable bit for the interrupt triggered when the timer 0 stops. (R/W)

Register 16.70: INT_RAW_PWM_REG (0x0114)



INT_CAP2_INT_RAW The raw status bit for the interrupt triggered by capture on channel 2. (RO)

INT_CAP1_INT_RAW The raw status bit for the interrupt triggered by capture on channel 1. (RO)

INT_CAP0_INT_RAW The raw status bit for the interrupt triggered by capture on channel 0. (RO)

INT_FH2_OST_INT_RAW The raw status bit for the interrupt triggered by a one-shot mode action on PWM2. (RO)

INT_FH1_OST_INT_RAW The raw status bit for the interrupt triggered by a one-shot mode action on PWM0. (RO)

INT_FH0_OST_INT_RAW The raw status bit for the interrupt triggered by a one-shot mode action on PWM0. (RO)

INT_FH2_CBC_INT_RAW The raw status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM2. (RO)

INT_FH1_CBC_INT_RAW The raw status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM1. (RO)

INT_FH0_CBC_INT_RAW The raw status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM0. (RO)

INT_OP2_TEB_INT_RAW The raw status bit for the interrupt triggered by a PWM operator 2 TEB event. (RO)

INT_OP1_TEB_INT_RAW The raw status bit for the interrupt triggered by a PWM operator 1 TEB event. (RO)

INT_OP0_TEB_INT_RAW The raw status bit for the interrupt triggered by a PWM operator 0 TEB event. (RO)

INT_OP2_TEA_INT_RAW The raw status bit for the interrupt triggered by a PWM operator 2 TEA event. (RO)

INT_OP1_TEA_INT_RAW The raw status bit for the interrupt triggered by a PWM operator 1 TEA event. (RO)

INT_OP0_TEA_INT_RAW The raw status bit for the interrupt triggered by a PWM operator 0 TEA event. (RO)

INT_FAULT2_CLR_INT_RAW The raw status bit for the interrupt triggered when event_f2 ends. (RO)

INT_FAULT1_CLR_INT_RAW The raw status bit for the interrupt triggered when event_f1 ends. (RO)

INT_FAULT0_CLR_INT_RAW The raw status bit for the interrupt triggered when event_f0 ends. (RO)

INT_FAULT2_INT_RAW The raw status bit for the interrupt triggered when event_f2 starts. (RO)

INT_FAULT1_INT_RAW The raw status bit for the interrupt triggered when event_f1 starts. (RO)

INT_FAULT0_INT_RAW The raw status bit for the interrupt triggered when event_f0 starts. (RO)

INT_TIMER2_TEP_INT_RAW The raw status bit for the interrupt triggered by a PWM timer 2 TEP event. (RO)

INT_TIMER1_TEP_INT_RAW The raw status bit for the interrupt triggered by a PWM timer 1 TEP event. (RO)

INT_TIMER0_TEP_INT_RAW The raw status bit for the interrupt triggered by a PWM timer 0 TEP event. (RO)

INT_TIMER2_TEZ_INT_RAW The raw status bit for the interrupt triggered by a PWM timer 2 TEZ event. (RO)

INT_TIMER1_TEZ_INT_RAW The raw status bit for the interrupt triggered by a PWM timer 1 TEZ event. (RO)

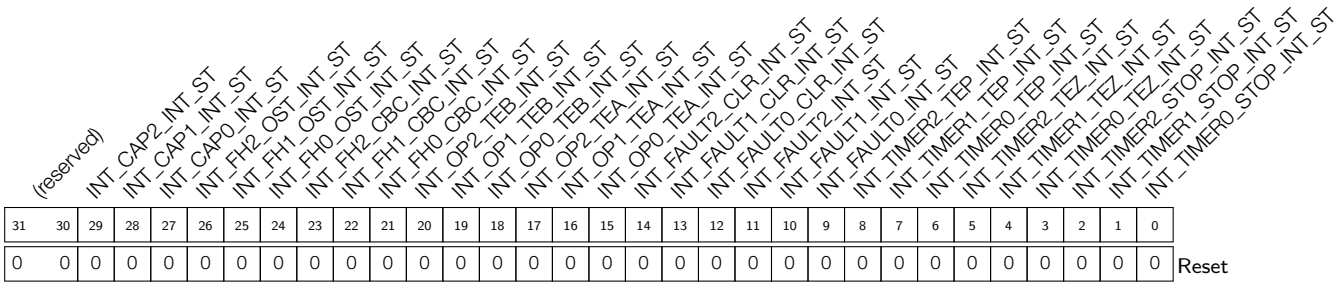
INT_TIMER0_TEZ_INT_RAW The raw status bit for the interrupt triggered by a PWM timer 0 TEZ event. (RO)

INT_TIMER2_STOP_INT_RAW The raw status bit for the interrupt triggered when the timer 2 stops. (RO)

INT_TIMER1_STOP_INT_RAW The raw status bit for the interrupt triggered when the timer 1 stops. (RO)

INT_TIMER0_STOP_INT_RAW The raw status bit for the interrupt triggered when the timer 0 stops. (RO)

Register 16.71: INT_ST_PWM_REG (0x0118)



INT_CAP2_INT_ST The masked status bit for the interrupt triggered by capture on channel 2. (RO)

INT_CAP1_INT_ST The masked status bit for the interrupt triggered by capture on channel 1. (RO)

INT_CAP0_INT_ST The masked status bit for the interrupt triggered by capture on channel 0. (RO)

INT_FH2_OST_INT_ST The masked status bit for the interrupt triggered by a one-shot mode action on PWM2. (RO)

INT_FH1_OST_INT_ST The masked status bit for the interrupt triggered by a one-shot mode action on PWM1. (RO)

INT_FH0_OST_INT_ST The masked status bit for the interrupt triggered by a one-shot mode action on PWM0. (RO)

INT_FH2_CBC_INT_ST The masked status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM2. (RO)

INT_FH1_CBC_INT_ST The masked status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM1. (RO)

INT_FH0_CBC_INT_ST The masked status bit for the interrupt triggered by a cycle-by-cycle mode action on PWM0. (RO)

INT_OP2_TEB_INT_ST The masked status bit for the interrupt triggered by a PWM operator 2 TEB event. (RO)

INT_OP1_TEB_INT_ST The masked status bit for the interrupt triggered by a PWM operator 1 TEB event. (RO)

INT_OP0_TEB_INT_ST The masked status bit for the interrupt triggered by a PWM operator 0 TEB event. (RO)

INT_OP2_TEA_INT_ST The masked status bit for the interrupt triggered by a PWM operator 2 TEA event. (RO)

INT_OP1_TEA_INT_ST The masked status bit for the interrupt triggered by a PWM operator 1 TEA event. (RO)

INT_OP0_TEA_INT_ST The masked status bit for the interrupt triggered by a PWM operator 0 TEA event. (RO)

INT_FAULT2_CLR_INT_ST The masked status bit for the interrupt triggered when event_f2 ends. (RO)

INT_FAULT1_CLR_INT_ST The masked status bit for the interrupt triggered when event_f1 ends. (RO)

INT_FAULT0_CLR_INT_ST The masked status bit for the interrupt triggered when event_f0 ends. (RO)

INT_FAULT2_INT_ST The masked status bit for the interrupt triggered when event_f2 starts. (RO)

INT_FAULT1_INT_ST The masked status bit for the interrupt triggered when event_f1 starts. (RO)

INT_FAULT0_INT_ST The masked status bit for the interrupt triggered when event_f0 starts. (RO)

INT_TIMER2_TEP_INT_ST The masked status bit for the interrupt triggered by a PWM timer 2 TEP event. (RO)

INT_TIMER1_TEP_INT_ST The masked status bit for the interrupt triggered by a PWM timer 1 TEP event. (RO)

INT_TIMER0_TEP_INT_ST The masked status bit for the interrupt triggered by a PWM timer 0 TEP event. (RO)

INT_TIMER2_TEZ_INT_ST The masked status bit for the interrupt triggered by a PWM timer 2 TEZ event. (RO)

INT_TIMER1_TEZ_INT_ST The masked status bit for the interrupt triggered by a PWM timer 1 TEZ event. (RO)

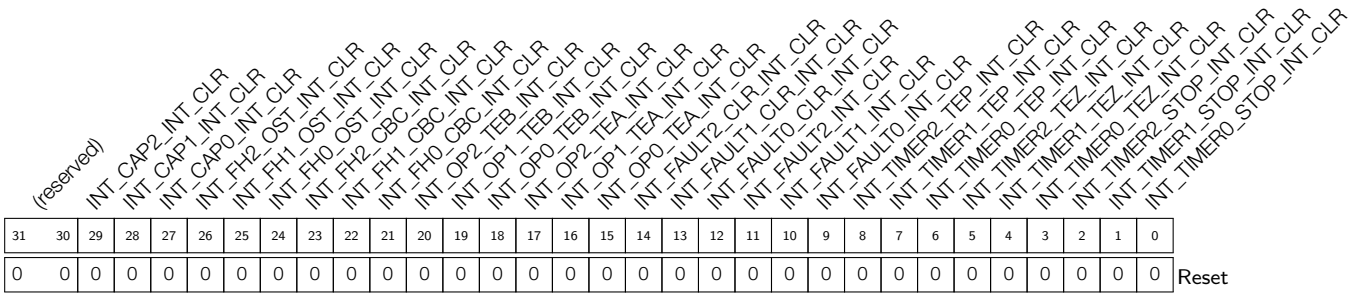
INT_TIMER0_TEZ_INT_ST The masked status bit for the interrupt triggered by a PWM timer 0 TEZ event. (RO)

INT_TIMER2_STOP_INT_ST The masked status bit for the interrupt triggered when the timer 2 stops. (RO)

INT_TIMER1_STOP_INT_ST The masked status bit for the interrupt triggered when the timer 1 stops. (RO)

INT_TIMER0_STOP_INT_ST The masked status bit for the interrupt triggered when the timer 0 stops. (RO)

Register 16.72: INT_CLR_PWM_REG (0x011c)



INT_CAP2_INT_CLR Set this bit to clear interrupt triggered by capture on channel 2. (WO)

INT_CAP1_INT_CLR Set this bit to clear interrupt triggered by capture on channel 1. (WO)

INT_CAP0_INT_CLR Set this bit to clear interrupt triggered by capture on channel 0. (WO)

INT_FH2_OST_INT_CLR Set this bit to clear interrupt triggered by a one-shot mode action on PWM2. (WO)

INT_FH1_OST_INT_CLR Set this bit to clear interrupt triggered by a one-shot mode action on PWM1. (WO)

INT_FH0_OST_INT_CLR Set this bit to clear interrupt triggered by a one-shot mode action on PWM0. (WO)

INT_FH2_CBC_INT_CLR Set this bit to clear interrupt triggered by a cycle-by-cycle mode action on PWM2. (WO)

INT_FH1_CBC_INT_CLR Set this bit to clear interrupt triggered by a cycle-by-cycle mode action on PWM1. (WO)

INT_FH0_CBC_INT_CLR Set this bit to clear interrupt triggered by a cycle-by-cycle mode action on PWM0. (WO)

INT_OP2_TEB_INT_CLR Set this bit to clear interrupt triggered by a PWM operator 2 TEB event. (WO)

INT_OP1_TEB_INT_CLR Set this bit to clear interrupt triggered by a PWM operator 1 TEB event. (WO)

INT_OP0_TEB_INT_CLR Set this bit to clear interrupt triggered by a PWM operator 0 TEB event. (WO)

INT_OP2_TEA_INT_CLR Set this bit to clear interrupt triggered by a PWM operator 2 TEA event. (WO)

INT_OP1_TEA_INT_CLR Set this bit to clear interrupt triggered by a PWM operator 1 TEA event. (WO)

INT_OP0_TEA_INT_CLR Set this bit to clear interrupt triggered by a PWM operator 0 TEA event. (WO)

INT_FAULT2_CLR_INT_CLR Set this bit to clear interrupt triggered when event_f2 ends. (WO)

INT_FAULT1_CLR_INT_CLR Set this bit to clear interrupt triggered when event_f1 ends. (WO)

INT_FAULT0_CLR_INT_CLR Set this bit to clear interrupt triggered when event_f0 ends. (WO)

INT_FAULT2_INT_CLR Set this bit to clear interrupt triggered when event_f2 starts. (WO)

INT_FAULT1_INT_CLR Set this bit to clear interrupt triggered when event_f1 starts. (WO)

INT_FAULT0_INT_CLR Set this bit to clear interrupt triggered when event_f0 starts. (WO)

INT_TIMER2_TEP_INT_CLR Set this bit to clear interrupt triggered by a PWM timer 2 TEP event. (WO)

INT_TIMER1_TEP_INT_CLR Set this bit to clear interrupt triggered by a PWM timer 1 TEP event. (WO)

INT_TIMER0_TEP_INT_CLR Set this bit to clear interrupt triggered by a PWM timer 0 TEP event. (WO)

INT_TIMER2_TEZ_INT_CLR Set this bit to clear interrupt triggered by a PWM timer 2 TEZ event. (WO)

INT_TIMER1_TEZ_INT_CLR Set this bit to clear interrupt triggered by a PWM timer 1 TEZ event. (WO)

INT_TIMER0_TEZ_INT_CLR Set this bit to clear interrupt triggered by a PWM timer 0 TEZ event. (WO)

INT_TIMER2_STOP_INT_CLR Set this bit to clear interrupt triggered when the timer 2 stops. (WO)

INT_TIMER1_STOP_INT_CLR Set this bit to clear interrupt triggered when the timer 1 stops. (WO)

INT_TIMER0_STOP_INT_CLR Set this bit to clear interrupt triggered when the timer 0 stops. (WO)

17. PULSE_CNT

17.1 Overview

The pulse counter module is designed to count the number of rising and/or falling edges of an input signal. Each pulse counter unit has a 16-bit signed counter register and two channels that can be configured to either increment or decrement the counter. Each channel has a signal input that accepts signal edges to be detected, as well as a control input that can be used to enable or disable the signal input. The inputs have optional filters that can be used to discard unwanted glitches in the signal.

The pulse counter has eight independent units, referred to as PULSE_CNT_U*n*.

17.2 Functional Description

17.2.1 Architecture

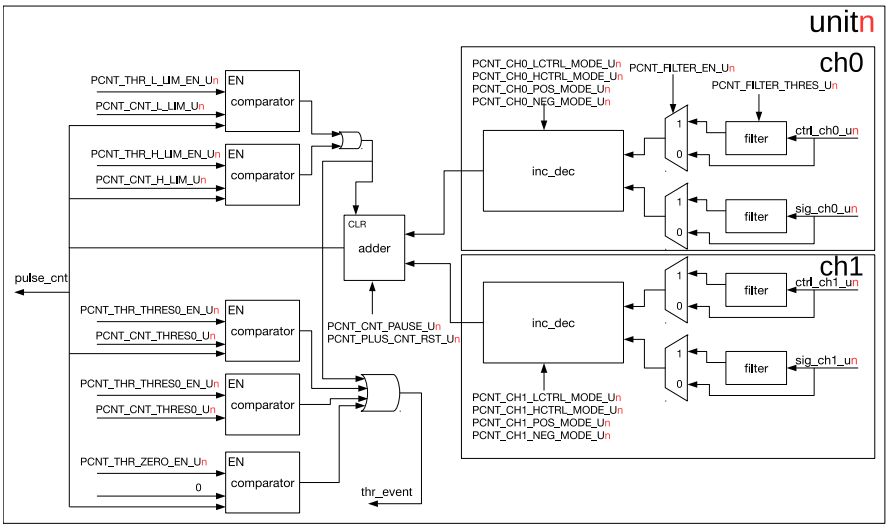


Figure 120: PULSE_CNT Architecture

The architecture of a pulse counter unit is illustrated in Figure 120. Each unit has two channels: ch0 and ch1, which are functionally equivalent. Each channel has a signal input, as well as a control input, which can both be connected to I/O pads. The counting behavior on both the positive and negative edge can be configured separately to increase, decrease, or do nothing to the counter value. Separately, for both control signal levels, the hardware can be configured to modify the edge action: invert it, disable it, or do nothing. The counter itself is a 16-bit signed up/down counter. Its value can be read by software directly, but is also monitored by a set of comparators which can trigger an interrupt.

17.2.2 Counter Channel Inputs

As stated before, the two inputs of a channel can affect the pulse counter in various ways. The specifics of this behaviour are set by LCTRL_MODE and HCTRL_MODE in this case when the control signal is low or high, respectively, and POS_MODE and NEG_MODE for positive and negative edges of the input signal. Setting POS_MODE and NEG_MODE to 1 will increase the counter when an edge is detected, setting them to 2 will decrease the counter and setting at any other value will neutralize the effect of the edge on the counter. LCTR_MODE and HCTR_MODE modify this behaviour, when the control input has the corresponding low or high

value: 0 does not modify the NEG_MODE and POS_MODE behaviour, 1 inverts it (setting POS_MODE/NEG_MODE to increase the counter should now decrease the counter and vice versa) and any other value disables counter effects for that signal level.

To summarize, a few examples have been considered. In this table, the effect on the counter for a rising edge is shown for both a low and a high control signal, as well as various other configuration options. For clarity, a short description in brackets is added after the values. Note: x denotes 'do not care'.

POS_ MODE	LCTRL_ MODE	HCTRL_ MODE	sig l→h when ctrl=0	sig l→h when ctrl=1
1 (inc)	0 (-)	0 (-)	Inc ctr	Inc ctr
2 (dec)	0 (-)	0 (-)	Dec ctr	Dec ctr
0 (-)	x	x	No action	No action
1 (inc)	0 (-)	1 (inv)	Inc ctr	Dec ctr
1 (inc)	1 (inv)	0 (-)	Dec ctr	Inc ctr
2 (dec)	0 (-)	1 (inv)	Dec ctr	Inc ctr
1 (inc)	0 (-)	2 (dis)	Inc ctr	No action
1 (inc)	2 (dis)	0 (-)	No action	Inc ctr

This table is also valid for negative edges (sig h→l) on substituting NEG_MODE for POS_MODE.

Each pulse counter unit also features a filter on each of the four inputs, adding the option to ignore short glitches in the signals. If a PCNT_FILTER_EN_U*n* can be set to filter the four input signals of the unit. If this filter is enabled, any pulses shorter than REG_FILTER_THRES_U*n* number of APB_CLK clock cycles will be filtered out and will have no effect on the counter. With the filter disabled, in theory infinitely small glitches could possibly trigger pulse counter action. However, in practice the signal inputs are sampled on APB_CLK edges and even with the filter disabled, pulse widths lasting shorter than one APB_CLK cycle may be missed.

Apart from the input channels, software also has some control over the counter. In particular, the counter value can be frozen to the current value by configuring PCNT_CNT_PAUSE_U*n*. It can also be reset to 0 by configuring PCNT_PLUS_CNT_RST_U*n*.

17.2.3 Watchpoints

The pulse counters have five watchpoints that share one interrupt. Interrupt generation can be enabled or disabled for each individual watchpoint. The watchpoints are:

- Maximum count value: Triggered when PULSE_CNT >= PCNT_CNT_H_LIM_U*n*. Additionally, this will reset the counter to 0. PCNT_CNT_H_LIM_U*n* should be a positive number.
- Minimum count value: Triggered when PULSE_CNT <= PCNT_CNT_L_LIM_U*n*. Additionally, this will reset the counter to 0. PCNT_CNT_L_LIM_U*n* should be a negative number.
- Two threshold values: Triggered when PULSE_CNT = PCNT_THR_THRES0_U*n* or PCNT_THR_THRES1_U*n*.
- Zero: Triggered when PULSE_CNT = 0.

17.2.4 Examples

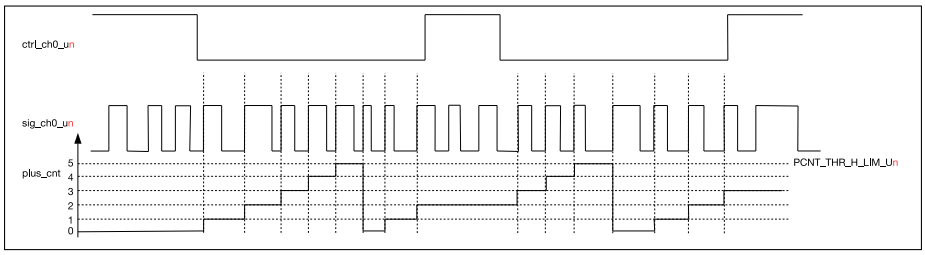


Figure 121: PULSE_CNT Upcounting Diagram

Figure 121 shows channel 0 being used as an up-counter. The configuration of channel 0 is shown below.

- CNT_CH0_POS_MODE_U*n* = 1: increase counter on the rising edge of sig_ch0_u*n*.
- PCNT_CH0_NEG_MODE_U*n* = 0: no counting on the falling edge of sig_ch0_u*n*.
- PCNT_CH0_LCTRL_MODE_U*n* = 0: Do not modify counter mode when sig_ch0_u*n* is low.
- PCNT_CH0_HCTRL_MODE_U*n* = 2: Do not allow counter increments/decrements when sig_ch0_u*n* is high.
- PCNT_CNT_H_LIM_U*n* = 5: PULSE_CNT resets to 0 when the count value increases to 5.

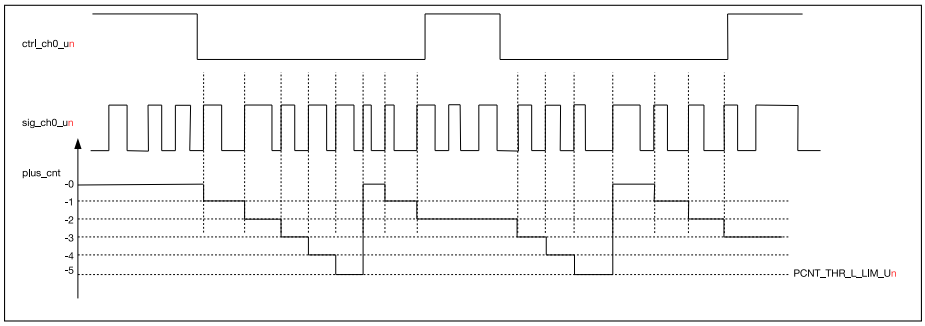


Figure 122: PULSE_CNT Downcounting Diagram

Figure 122 shows channel 0 decrementing the counter. The configuration of channel 0 differs from that in Figure 121 in the following two aspects:

- PCNT_CH0_LCTRL_MODE_U*n* = 1: invert counter mode when ctrl_ch0_u*n* is at low level, so it will decrease, rather than increase, the counter.
- PCNT_CNT_H_LIM_U*n* = –5: PULSE_CNT resets to 0 when the count value decreases to –5.

17.2.5 Interrupts

PCNT_CNT_THR_EVENT_U*n*_INT: This interrupt gets triggered when one of the five channel comparators detects a match.

17.3 Register Summary

Name	Description	Address	Access
Configuration registers			

Name	Description	Address	Access
PCNT_U0_CONF0_REG	Configuration register 0 for unit 0	0x3FF57000	R/W
PCNT_U1_CONF0_REG	Configuration register 0 for unit 1	0x3FF5700C	R/W
PCNT_U2_CONF0_REG	Configuration register 0 for unit 2	0x3FF57018	R/W
PCNT_U3_CONF0_REG	Configuration register 0 for unit 3	0x3FF57024	R/W
PCNT_U4_CONF0_REG	Configuration register 0 for unit 4	0x3FF57030	R/W
PCNT_U5_CONF0_REG	Configuration register 0 for unit 5	0x3FF5703C	R/W
PCNT_U6_CONF0_REG	Configuration register 0 for unit 6	0x3FF57048	R/W
PCNT_U7_CONF0_REG	Configuration register 0 for unit 7	0x3FF57054	R/W
PCNT_U0_CONF1_REG	Configuration register 1 for unit 0	0x3FF57004	R/W
PCNT_U1_CONF1_REG	Configuration register 1 for unit 1	0x3FF57010	R/W
PCNT_U2_CONF1_REG	Configuration register 1 for unit 2	0x3FF5701C	R/W
PCNT_U3_CONF1_REG	Configuration register 1 for unit 3	0x3FF57028	R/W
PCNT_U4_CONF1_REG	Configuration register 1 for unit 4	0x3FF57034	R/W
PCNT_U5_CONF1_REG	Configuration register 1 for unit 5	0x3FF57040	R/W
PCNT_U6_CONF1_REG	Configuration register 1 for unit 6	0x3FF5704C	R/W
PCNT_U7_CONF1_REG	Configuration register 1 for unit 7	0x3FF57058	R/W
PCNT_U0_CONF2_REG	Configuration register 2 for unit 0	0x3FF57008	R/W
PCNT_U1_CONF2_REG	Configuration register 2 for unit 1	0x3FF57014	R/W
PCNT_U2_CONF2_REG	Configuration register 2 for unit 2	0x3FF57020	R/W
PCNT_U3_CONF2_REG	Configuration register 2 for unit 3	0x3FF5702C	R/W
PCNT_U4_CONF2_REG	Configuration register 2 for unit 4	0x3FF57038	R/W
PCNT_U5_CONF2_REG	Configuration register 2 for unit 5	0x3FF57044	R/W
PCNT_U6_CONF2_REG	Configuration register 2 for unit 6	0x3FF57050	R/W
PCNT_U7_CONF2_REG	Configuration register 2 for unit 7	0x3FF5705C	R/W
Counter values			
PCNT_U0_CNT_REG	Counter value for unit 0	0x3FF57060	RO
PCNT_U1_CNT_REG	Counter value for unit 1	0x3FF57064	RO
PCNT_U2_CNT_REG	Counter value for unit 2	0x3FF57068	RO
PCNT_U3_CNT_REG	Counter value for unit 3	0x3FF5706C	RO
PCNT_U4_CNT_REG	Counter value for unit 4	0x3FF57070	RO
PCNT_U5_CNT_REG	Counter value for unit 5	0x3FF57074	RO
PCNT_U6_CNT_REG	Counter value for unit 6	0x3FF57078	RO
PCNT_U7_CNT_REG	Counter value for unit 7	0x3FF5707C	RO
Control registers			
PCNT_CTRL_REG	Control register for all counters	0x3FF570B0	R/W
Interrupt registers			
PCNT_INT_RAW_REG	Raw interrupt status	0x3FF57080	RO
PCNT_INT_ST_REG	Masked interrupt status	0x3FF57084	RO
PCNT_INT_ENA_REG	Interrupt enable bits	0x3FF57088	R/W
PCNT_INT_CLR_REG	Interrupt clear bits	0x3FF5708C	WO

17.4 Registers

Register 17.1: PCNT_U*n*_CONF0_REG (*n*: 0-7) (0x0+0x0C*n*)



PCNT_CH1_LCTRL_MODE_U*n* This register configures how the CH1_POS_MODE/CH1_NEG_MODE settings will be modified when the control signal is low. (R/W) 0: No modification; 1: Invert behaviour (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification

PCNT_CH1_HCTRL_MODE_U*n* This register configures how the CH1_POS_MODE/CH1_NEG_MODE settings will be modified when the control signal is low. (R/W) 0: No modification; 1: Invert behaviour (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification

PCNT_CH1_POS_MODE_U*n* This register sets the behaviour when the signal input of channel 1 detects a positive edge. (R/W) 1: Increment the counter; 2: Decrement the counter; 0, 3: No effect on counter

PCNT_CH1_NEG_MODE_U*n* This register sets the behaviour when the signal input of channel 1 detects a negative edge. (R/W) 1: Increment the counter; 2: Decrement the counter; 0, 3: No effect on counter

PCNT_CH0_LCTRL_MODE_U*n* This register configures how the CH0_POS_MODE/CH0_NEG_MODE settings will be modified when the control signal is low. (R/W) 0: No modification; 1: Invert behaviour (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification

PCNT_CH0_HCTRL_MODE_U*n* This register configures how the CH0_POS_MODE/CH0_NEG_MODE settings will be modified when the control signal is low. (R/W) 0: No modification; 1: Invert behaviour (increase -> decrease, decrease -> increase); 2, 3: Inhibit counter modification

PCNT_CH0_POS_MODE_U*n* This register sets the behaviour when the signal input of channel 0 detects a positive edge. (R/W) 1: Increase the counter; 2: Decrease the counter; 0, 3: No effect on counter

PCNT_CH0_NEG_MODE_U*n* This register sets the behaviour when the signal input of channel 0 detects a negative edge. (R/W) 1: Increase the counter; 2: Decrease the counter; 0, 3: No effect on counter

PCNT_THR_THRES1_EN_U*n* This is the enable bit for unit *n*'s thres1 comparator. (R/W)

PCNT_THR_THRES0_EN_U*n* This is the enable bit for unit *n*'s thres0 comparator. (R/W)

PCNT_THR_L_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_l_lim comparator. (R/W)

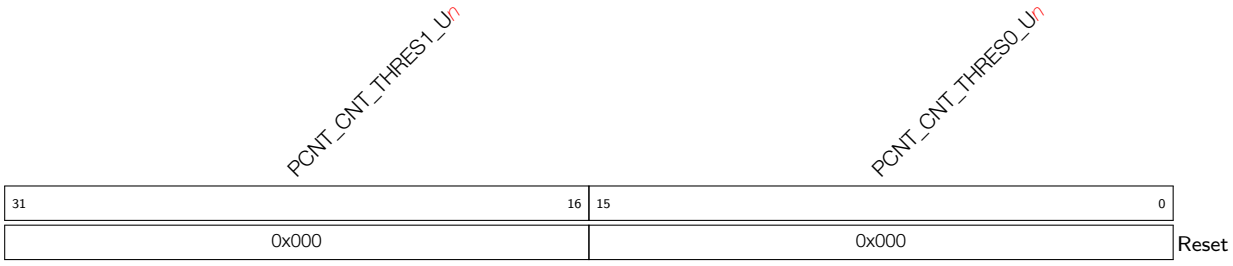
PCNT_THR_H_LIM_EN_U*n* This is the enable bit for unit *n*'s thr_h_lim comparator. (R/W)

PCNT_THR_ZERO_EN_U*n* This is the enable bit for unit *n*'s zero comparator. (R/W)

PCNT_FILTER_EN_U*n* This is the enable bit for unit *n*'s input filter. (R/W)

PCNT_FILTER_THRES_U*n* This sets the maximum threshold, in APB_CLK cycles, for the filter. Any pulses lasting shorter than this will be ignored when the filter is enabled. (R/W)

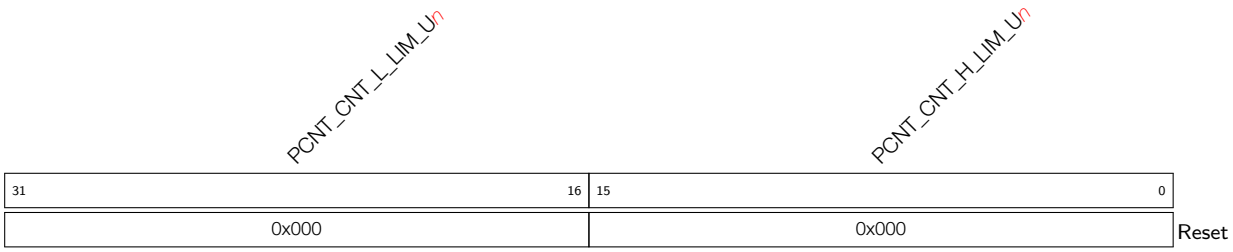
Register 17.2: PCNT_U*n*_CONF1_REG (*n*: 0-7) (0x4+0x0C*n)



PCNT_CNT_THRES1_U*n* This register is used to configure the thres1 value for unit *n*. (R/W)

PCNT_CNT_THRES0_U*n* This register is used to configure the thres0 value for unit *n*. (R/W)

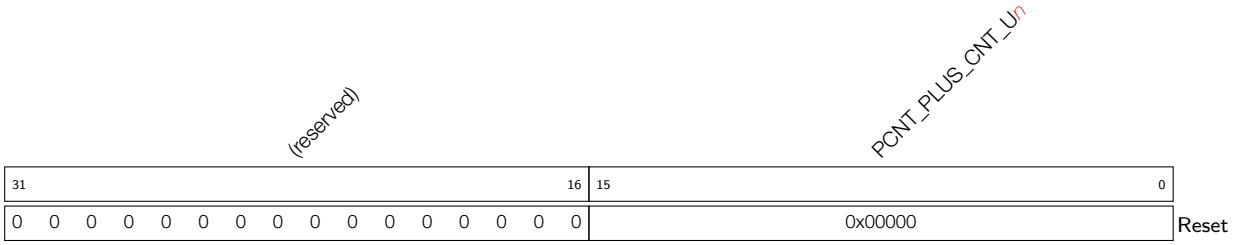
Register 17.3: PCNT_U*n*_CONF2_REG (*n*: 0-7) (0x8+0x0C*n)



PCNT_CNT_L_LIM_U*n* This register is used to configure the thr_l_lim value for unit *n*. (R/W)

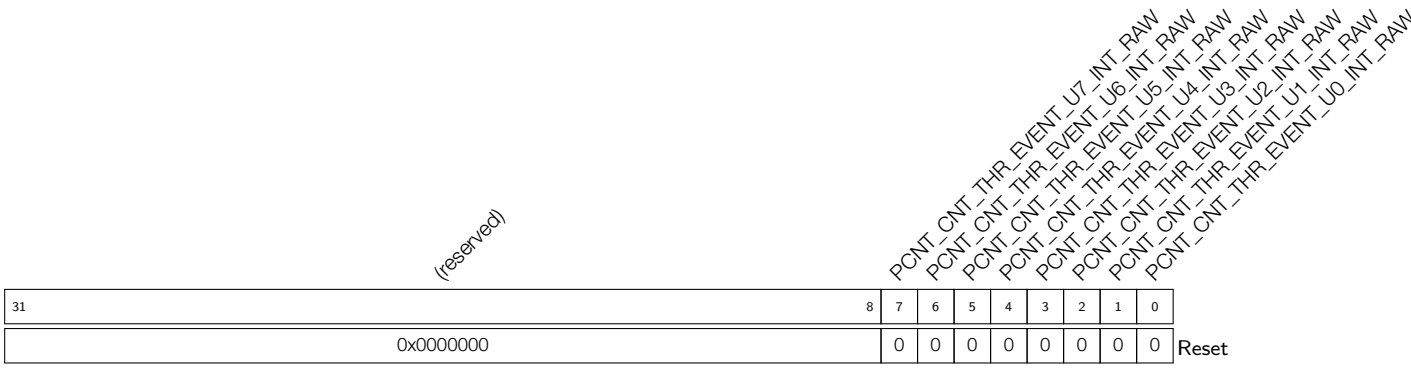
PCNT_CNT_H_LIM_U*n* This register is used to configure the thr_h_lim value for unit *n*. (R/W)

Register 17.4: PCNT_U*n*_CNT_REG (*n*: 0-7) (0x28+0x0C*n)



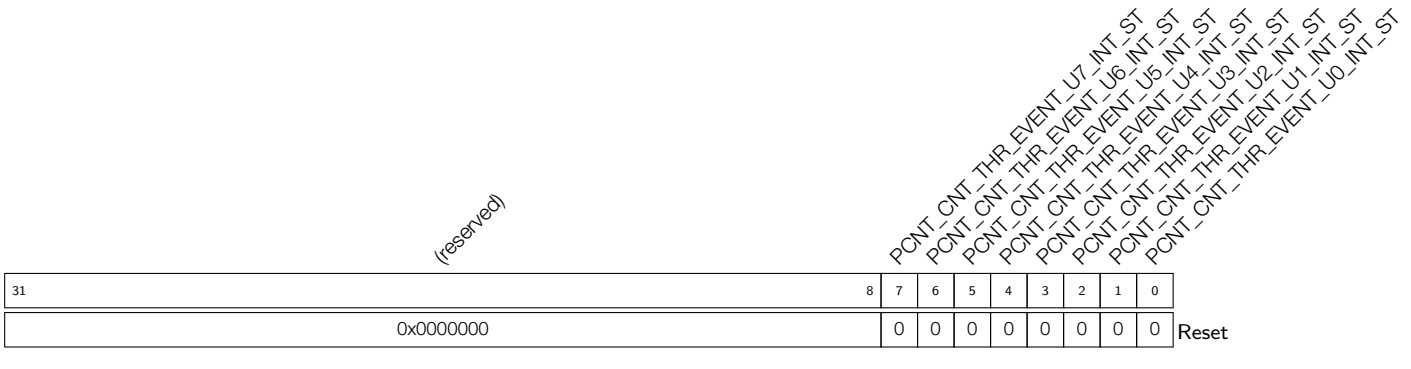
PCNT_PLUS_CNT_U*n* This register stores the current pulse count value for unit *n*. (RO)

Register 17.5: PCNT_INT_RAW_REG (0x0080)



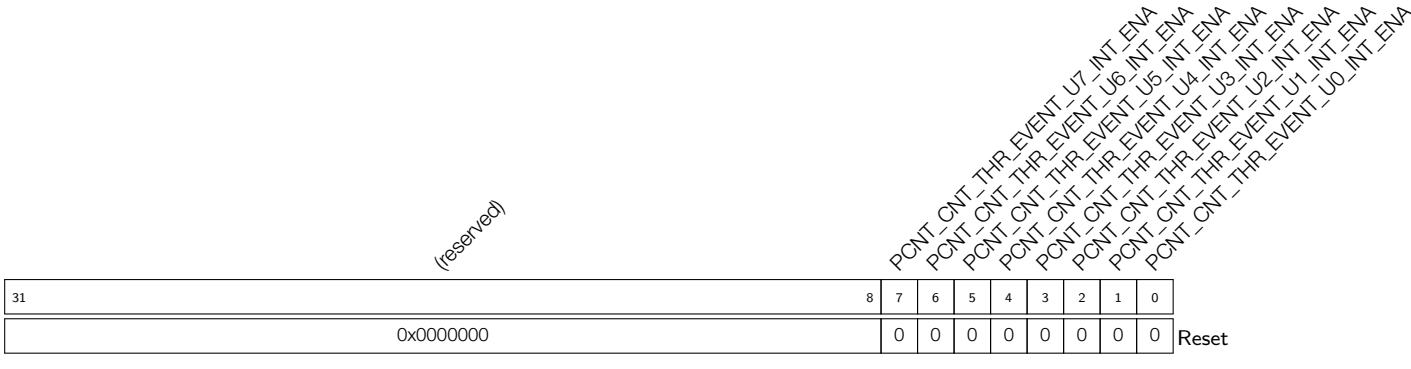
PCNT_CNT_THR_EVENT_U*n*_INT_RAW The raw interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 17.6: PCNT_INT_ST_REG (0x0084)



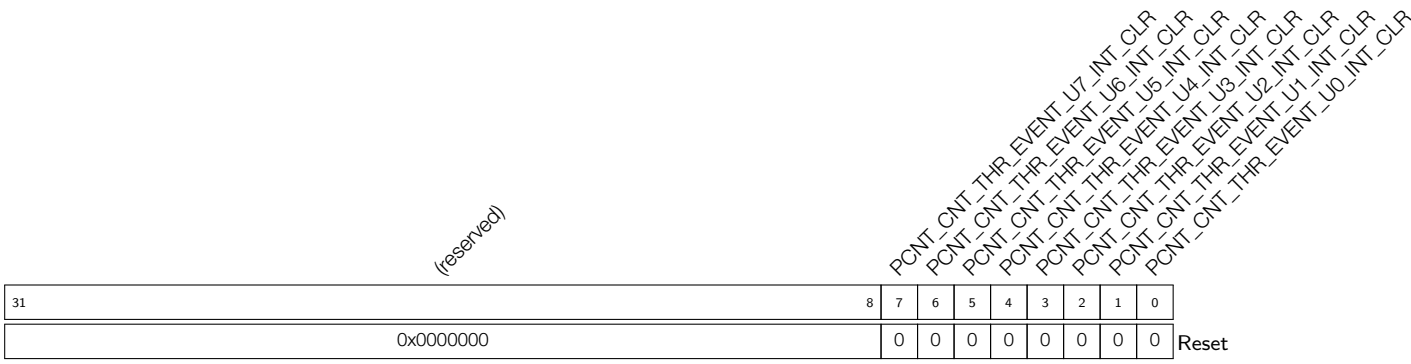
PCNT_CNT_THR_EVENT_U*n*_INT_ST The masked interrupt status bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (RO)

Register 17.7: PCNT_INT_ENA_REG (0x0088)



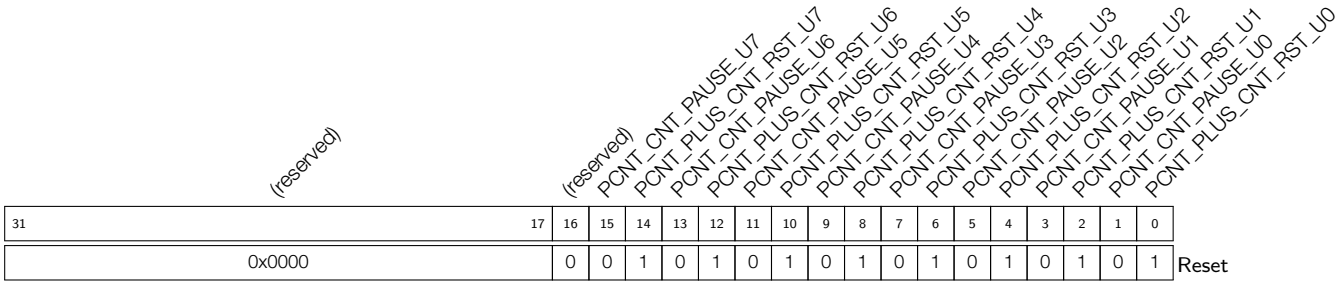
PCNT_CNT_THR_EVENT_U*n*_INT_ENA The interrupt enable bit for the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (R/W)

Register 17.8: PCNT_INT_CLR_REG (0x008c)



PCNT_CNT_THR_EVENT_U*n*_INT_CLR Set this bit to clear the PCNT_CNT_THR_EVENT_U*n*_INT interrupt. (WO)

Register 17.9: PCNT_CTRL_REG (0x00b0)



PCNT_CNT_PAUSE_U*n* Set this bit to freeze unit *n*'s counter. (R/W)

PCNT_PLUS_CNT_RST_U*n* Set this bit to clear unit *n*'s counter. (R/W)

18. 64-bit Timers

18.1 Introduction

There are four general-purpose timers embedded in the ESP32. They are all 64-bit generic timers based on 16-bit prescalers and 64-bit auto-reload-capable up/downcounters.

The ESP32 contains two timer modules, each containing two timers. The two timers in a block are indicated by an *x* in TIMG*n*_T*x*; the blocks themselves are indicated by an *n*.

The timers feature:

- A 16-bit clock prescaler, from 2 to 65536
- A 64-bit time-base counter
- Configurable up/down time-base counter: incrementing or decrementing
- Halt and resume of time-base counter
- Auto-reload at alarm
- Software-controlled instant reload
- Level and edge interrupt generation

18.2 Functional Description

18.2.1 16-bit Prescaler

Each timer uses the APB clock (APB_CLK, normally 80 MHz) as the basic clock. This clock is then divided down by a 16-bit precaler which generates the time-base counter clock (TB_clk). Every cycle of TB_clk causes the time-base counter to increment or decrement by one. The timer must be disabled (TIMG*n*_T*x*_EN is cleared) before changing the prescaler divisor which is configured by TIMG*n*_T*x*_DIVIDER register; changing it on an enabled timer can lead to unpredictable results. The prescaler can divide the APB clock by a factor from 2 to 65536. Specifically, when TIMG*n*_T*x*_DIVIDER is either 1 or 2, the clock divisor is 2; when TIMG*n*_T*x*_DIVIDER is 0, the clock divisor is 65536. Any other value will cause the clock to be divided by exactly that value.

18.2.2 64-bit Time-base Counter

The 64-bit time-base counter can be configured to count either up or down, depending on whether TIMG*n*_T*x*_INCREASE is set or cleared, respectively. It supports both auto-reload and software instant reload. An alarm event can be set when the counter reaches a value specified by the software.

Counting can be enabled and disabled by setting and clearing TIMG*n*_T*x*_EN. Clearing this bit essentially freezes the counter, causing it to neither count up nor count down; instead, it retains its value until TIMG*n*_T*x*_EN is set again. Reloading the counter when TIMG*n*_T*x*_EN is cleared will change its value, but counting will not be resumed until TIMG*n*_T*x*_EN is set.

Software can set a new counter value by setting registers TIMG*n*_T*x*_LOAD_LO and TIMG*n*_T*x*_LOAD_HI to the intended new value. The hardware will ignore these register settings until a reload; a reload will cause the contents of these registers to be copied to the counter itself. A reload event can be triggered by an alarm (auto-reload at alarm) or by software (software instant reload). To enable auto-reload at alarm, the register

TIMG*n*_T*x*_AUTORELOAD should be set. If auto-reload at alarm is not enabled, the time-base counter will continue incrementing or decrementing after the alarm. To trigger a software instant reload, any value can be written to the register TIMG*n*_T*x*_LOAD_REG; this will cause the counter value to change instantly. Software can also change the direction of the time-base counter instantly by changing the value of TIMG*n*_T*x*_INCREASE.

The time-base counter can also be read by software, but because the counter is 64-bit, the CPU can only get the value as two 32-bit values, the counter value needs to be latched onto TIMG*n*_T*x*LO_REG and TIMG*n*_T*x*HI_REG first. This is done by writing any value to TIMG*n*_T*x*UPDATE_REG; this will instantly latch the 64-bit timer value onto the two registers. Software can then read them at any point in time. This approach stops the timer value being read erroneously when a carry-over happens between reading the low and high word of the timer value.

18.2.3 Alarm Generation

The timer can trigger an alarm, which can cause a reload and/or an interrupt to occur. The alarm is triggered when the alarm registers TIMG*n*_T*x*_ALARMLO_REG and TIMG*n*_T*x*_ALARMHI_REG match the current timer value. In order to simplify the scenario where these registers are set 'too late' and the counter has already passed these values, the alarm also triggers when the current timer value is higher (for an up-counting timer) or lower (for a down-counting timer) than the current alarm value: if this is the case, the alarm will be triggered immediately upon loading the alarm registers.

18.2.4 MWDT

Each timer module also contains a Main System Watchdog Timer and its associated registers. While these registers are described here, their functional description can be found in the chapter entitled Watchdog Timer.

18.2.5 Interrupts

- TIMG*n*_T*x*_INT_WDT_INT: Generated when a watchdog timer interrupt stage times out.
- TIMG*n*_T*x*_INT_T1_INT: An alarm event on timer 1 generates this interrupt.
- TIMG*n*_T*x*_INT_T0_INT: An alarm event on timer 0 generates this interrupt.

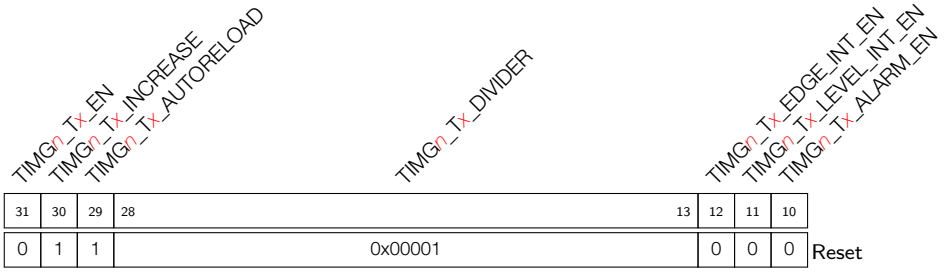
18.3 Register Summary

Name	Description	TIMG0	TIMG1	Acc
Timer 0 configuration and control registers				
TIMG*n*_T0CONFIG_REG	Timer 0 configuration register	0x3FF5F000	0x3FF60000	R/W
TIMG*n*_T0LO_REG	Timer 0 current value, low 32 bits	0x3FF5F004	0x3FF60004	RO
TIMG*n*_T0HI_REG	Timer 0 current value, high 32 bits	0x3FF5F008	0x3FF60008	RO
TIMG*n*_T0UPDATE_REG	Write to copy current timer value to TIMG*n*_T0_(LO/HI)_REG	0x3FF5F00C	0x3FF6000C	WO
TIMG*n*_T0ALARMLO_REG	Timer 0 alarm value, low 32 bits	0x3FF5F010	0x3FF60010	R/W
TIMG*n*_T0ALARMHI_REG	Timer 0 alarm value, high bits	0x3FF5F014	0x3FF60014	R/W
TIMG*n*_T0LOADLO_REG	Timer 0 reload value, low 32 bits	0x3FF5F018	0x3FF60018	R/W

Name	Description	TIMG0	TIMG1	Acc
TIMG*n*_T0LOAD_REG	Write to reload timer from TIMG*n*_T0_(LOADLOLOADHI)_REG	0x3FF5F020	0x3FF60020	WO
Timer 1 configuration and control registers				
TIMG*n*_T1CONFIG_REG	Timer 1 configuration register	0x3FF5F024	0x3FF60024	R/W
TIMG*n*_T1LO_REG	Timer 1 current value, low 32 bits	0x3FF5F028	0x3FF60028	RO
TIMG*n*_T1HI_REG	Timer 1 current value, high 32 bits	0x3FF5F02C	0x3FF6002C	RO
TIMG*n*_T1UPDATE_REG	Write to copy current timer value to TIMG*n*_T1_(LO/HI)_REG	0x3FF5F030	0x3FF60030	WO
TIMG*n*_T1ALARMLO_REG	Timer 1 alarm value, low 32 bits	0x3FF5F034	0x3FF60034	R/W
TIMG*n*_T1ALARMHI_REG	Timer 1 alarm value, high 32 bits	0x3FF5F038	0x3FF60038	R/W
TIMG*n*_T1LOADLO_REG	Timer 1 reload value, low 32 bits	0x3FF5F03C	0x3FF6003C	R/W
TIMG*n*_T1LOAD_REG	Write to reload timer from TIMG*n*_T1_(LOADLOLOADHI)_REG	0x3FF5F044	0x3FF60044	WO
System watchdog timer configuration and control registers				
TIMG*n*_T*x*_WDTCONFIG0_REG	Watchdog timer configuration register	0x3FF5F048	0x3FF60048	R/W
TIMG*n*_T*x*_WDTCONFIG1_REG	Watchdog timer prescaler register	0x3FF5F04C	0x3FF6004C	R/W
TIMG*n*_T*x*_WDTCONFIG2_REG	Watchdog timer stage 0 timeout value	0x3FF5F050	0x3FF60050	R/W
TIMG*n*_T*x*_WDTCONFIG3_REG	Watchdog timer stage 1 timeout value	0x3FF5F054	0x3FF60054	R/W
TIMG*n*_T*x*_WDTCONFIG4_REG	Watchdog timer stage 2 timeout value	0x3FF5F058	0x3FF60058	R/W
TIMG*n*_T*x*_WDTCONFIG5_REG	Watchdog timer stage 3 timeout value	0x3FF5F05C	0x3FF6005C	R/W
TIMG*n*_T*x*_WDTFEED_REG	Write to feed the watchdog timer	0x3FF5F060	0x3FF60060	WO
TIMG*n*_T*x*_WDTWPROTECT_REG	Watchdog write protect register	0x3FF5F064	0x3FF60064	R/W
Interrupt registers				
TIMG*n*_T*x*_INT_RAW_REG	Raw interrupt status	0x3FF5F09C	0x3FF6009C	RO
TIMG*n*_T*x*_INT_ST_REG	Masked interrupt status	0x3FF5F0A0	0x3FF600A0	RO
TIMG*n*_T*x*_INT_ENA_REG	Interrupt enable bits	0x3FF5F098	0x3FF60098	R/W
TIMG*n*_T*x*_INT_CLR_REG	Interrupt clear bits	0x3FF5F0A4	0x3FF600A4	WO

18.4 Registers

Register 18.1: TIMG*n*_T*x*CONFIG_REG (*x*: 0-1) (0x0+0x24x*)**



TIMG*n*_T*x*_EN When set, the timer *x* time-base counter is enabled. (R/W)

TIMG*n*_T*x*_INCREASE When set, the timer *x* time-base counter will increment every clock tick. When cleared, the timer *x* time-base counter will decrement. (R/W)

TIMG*n*_T*x*_AUTORELOAD When set, timer *x* auto-reload at alarm is enabled. (R/W)

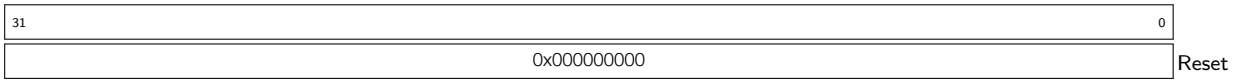
TIMG*n*_T*x*_DIVIDER Timer *x* clock (T*x*_clk) prescale value. (R/W)

TIMG*n*_T*x*_EDGE_INT_EN When set, an alarm will generate an edge type interrupt. (R/W)

TIMG*n*_T*x*_LEVEL_INT_EN When set, an alarm will generate a level type interrupt. (R/W)

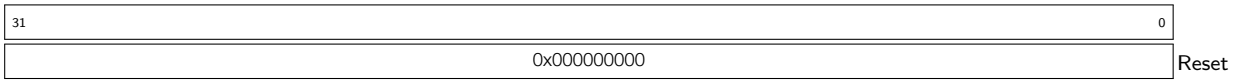
TIMG*n*_T*x*_ALARM_EN When set, the alarm is enabled. (R/W)

Register 18.2: TIMG*n*_T*x*LO_REG (*x*: 0-1) (0x4+0x24x*)**



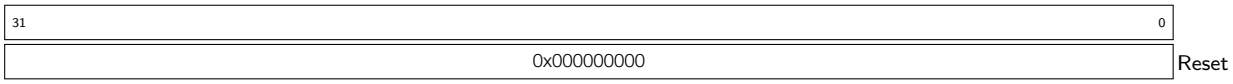
TIMG*n*_T*x*LO_REG After writing to TIMG*n*_T*x*UPDATE_REG, the low 32 bits of the time-base counter of timer *x* can be read here. (RO)

Register 18.3: TIMG*n*_T*x*HI_REG (*x*: 0-1) (0x8+0x24x*)**



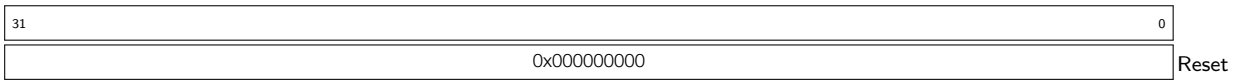
TIMG*n*_T*x*HI_REG After writing to TIMG*n*_T*x*UPDATE_REG, the high 32 bits of the time-base counter of timer *x* can be read here. (RO)

Register 18.4: TIMG*n*_T*x*UPDATE_REG (*x*: 0-1) (0xC+0x24**x*)



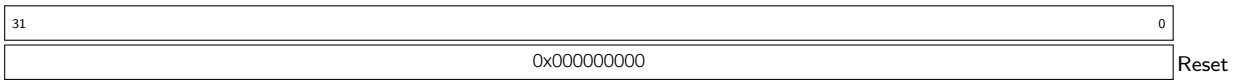
TIMG*n*_T*x*UPDATE_REG Write any value to trigger a timer *x* time-base counter value update (timer *x* current value will be stored in registers above). (WO)

Register 18.5: TIMG*n*_T*x*ALARMLO_REG (*x*: 0-1) (0x10+0x24**x*)



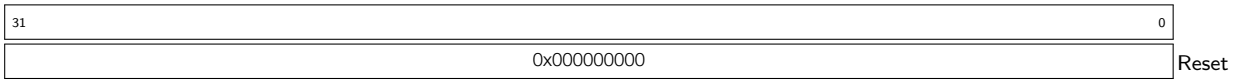
TIMG*n*_T*x*ALARMLO_REG Timer *x* alarm trigger time-base counter value, low 32 bits. (R/W)

Register 18.6: TIMG*n*_T*x*ALARMHI_REG (*x*: 0-1) (0x14+0x24**x*)



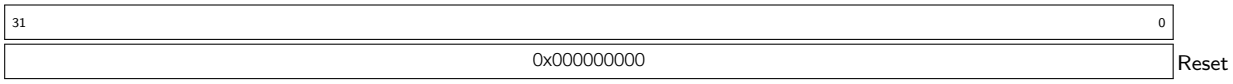
TIMG*n*_T*x*ALARMHI_REG Timer *x* alarm trigger time-base counter value, high 32 bits. (R/W)

Register 18.7: TIMG*n*_T*x*LOADLO_REG (*x*: 0-1) (0x18+0x24**x*)



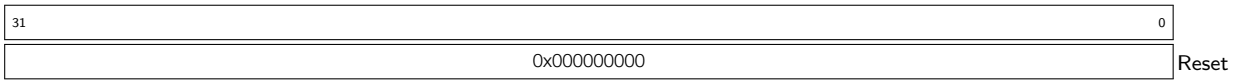
TIMG*n*_T*x*LOADLO_REG Low 32 bits of the value that a reload will load onto timer *x* time-base counter. (R/W)

Register 18.8: TIMG*n*_T*x*LOADHI_REG (*x*: 0-1) (0x1C+0x24**x*)



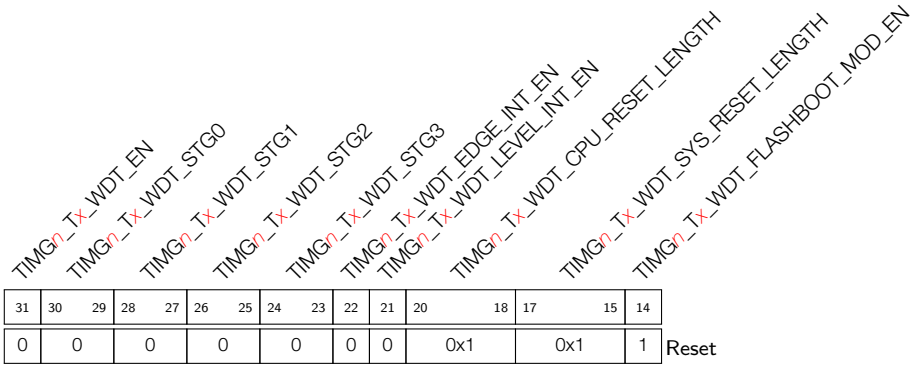
TIMG*n*_T*x*LOADHI_REG High 32 bits of the value that a reload will load onto timer *x* time-base counter. (R/W)

Register 18.9: TIMG*n*_T*x*LOAD_REG (*x*: 0-1) (0x20+0x24x*)**



TIMG*n*_T*x*LOAD_REG Write any value to trigger a timer *x* time-base counter reload. (WO)

Register 18.10: TIMG*n*_T*x*_WDTCONFIG0_REG (0x0048)



TIMG*n*_T*x*_WDT_EN When set, MWDT is enabled. (R/W)

TIMG*n*_T*x*_WDT_STG0 Stage 0 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG*n*_T*x*_WDT_STG1 Stage 1 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG*n*_T*x*_WDT_STG2 Stage 2 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG*n*_T*x*_WDT_STG3 Stage 3 configuration. 0: off, 1: interrupt, 2: reset CPU, 3: reset system. (R/W)

TIMG*n*_T*x*_WDT_EDGE_INT_EN When set, an edge type interrupt will occur at the timeout of a stage configured to generate an interrupt. (R/W)

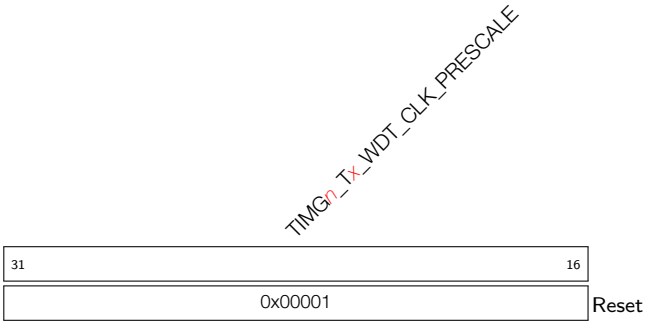
TIMG*n*_T*x*_WDT_LEVEL_INT_EN When set, a level type interrupt will occur at the timeout of a stage configured to generate an interrupt. (R/W)

TIMG*n*_T*x*_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*n*_T*x*_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)

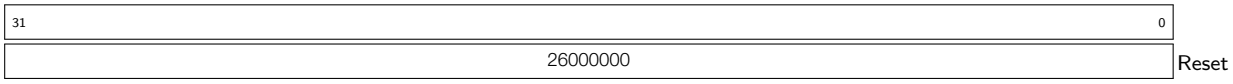
TIMG*n*_T*x*_WDT_FLASHBOOT_MOD_EN When set, Flash boot protection is enabled. (R/W)

Register 18.11: TIMG*n*_T*x*_WDTCONFIG1_REG (0x004c)



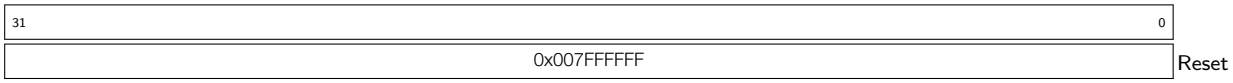
TIMG*n*_T*x*_WDT_CLK_PRESCALE MWDT clock prescale value. MWDT clock period = 12.5 ns * TIMG*n*_T*x*_WDT_CLK_PRESCALE. (R/W)

Register 18.12: TIMG*n*_T*x*_WDTCONFIG2_REG (0x0050)



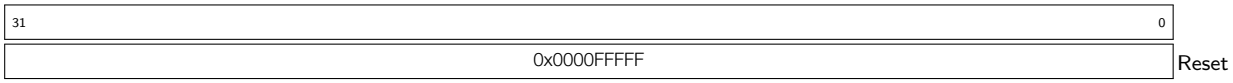
TIMG*n*_T*x*_WDTCONFIG2_REG Stage 0 timeout value, in MWDT clock cycles. (R/W)

Register 18.13: TIMG*n*_T*x*_WDTCONFIG3_REG (0x0054)



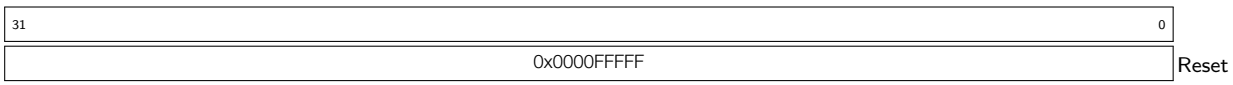
TIMG*n*_T*x*_WDTCONFIG3_REG Stage 1 timeout value, in MWDT clock cycles. (R/W)

Register 18.14: TIMG*n*_T*x*_WDTCONFIG4_REG (0x0058)



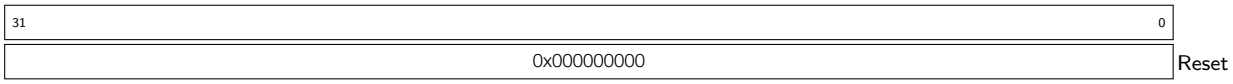
TIMG*n*_T*x*_WDTCONFIG4_REG Stage 2 timeout value, in MWDT clock cycles. (R/W)

Register 18.15: TIMG*n*_T*x*_WDTCONFIG5_REG (0x005c)



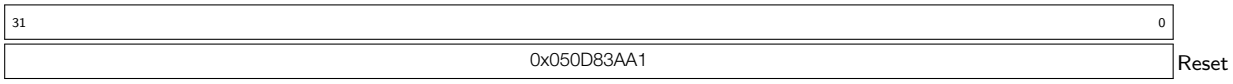
TIMG*n*_T*x*_WDTCONFIG5_REG Stage 3 timeout value, in MWDT clock cycles. (R/W)

Register 18.16: TIMG*n*_T*x*_WDTFEED_REG (0x0060)



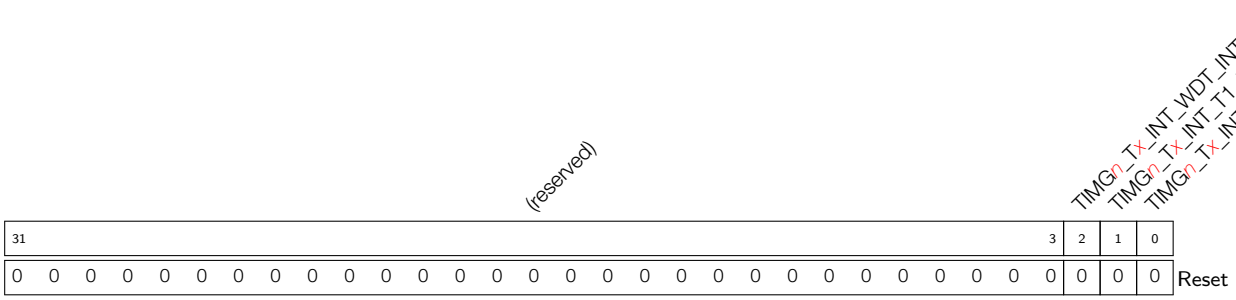
TIMG*n*_T*x*_WDTFEED_REG Write any value to feed the MWDT. (WO)

Register 18.17: TIMG*n*_T*x*_WDTWPROTECT_REG (0x0064)



TIMG*n*_T*x*_WDTWPROTECT_REG If the register contains a different value than its reset value, write protection is enabled. (R/W)

Register 18.18: TIMG*n*_T*x*_INT_ENA_REG (0x0098)

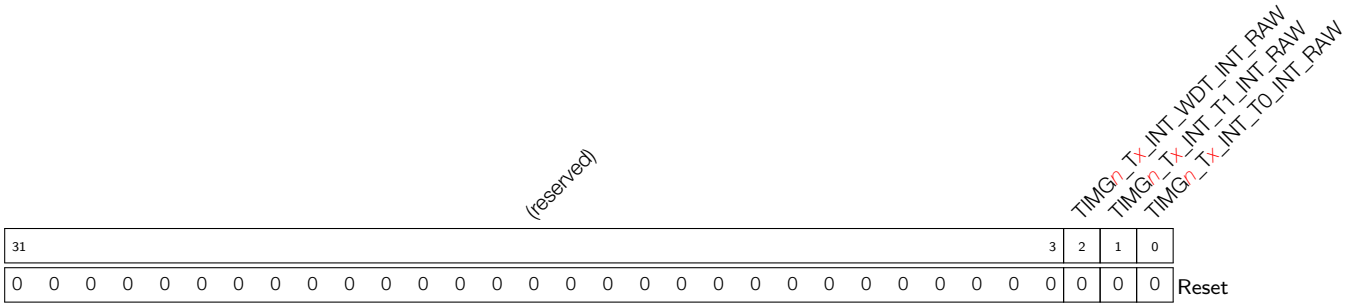


TIMG*n*_T*x*_INT_WDT_INT_ENA The interrupt enable bit for the TIMG*n*_T*x*_INT_WDT_INT interrupt. (R/W) (R/W)

TIMG*n*_T*x*_INT_T1_INT_ENA The interrupt enable bit for the TIMG*n*_T*x*_INT_T1_INT interrupt. (R/W) (R/W)

TIMG*n*_T*x*_INT_T0_INT_ENA The interrupt enable bit for the TIMG*n*_T*x*_INT_T0_INT interrupt. (R/W) (R/W)

Register 18.19: TIMG*n*_T*x*_INT_RAW_REG (0x009c)

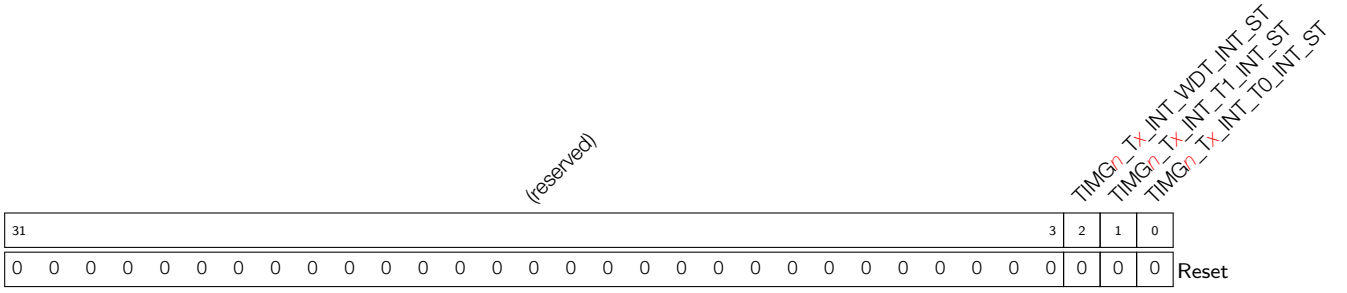


TIMG*n*_T*x*_INT_WDT_INT_RAW The raw interrupt status bit for the TIMG*n*_T*x*_INT_WDT_INT interrupt. (RO)

TIMG*n*_T*x*_INT_T1_INT_RAW The raw interrupt status bit for the TIMG*n*_T*x*_INT_T1_INT interrupt. (RO)

TIMG*n*_T*x*_INT_T0_INT_RAW The raw interrupt status bit for the TIMG*n*_T*x*_INT_T0_INT interrupt. (RO)

Register 18.20: TIMG*n*_T*x*_INT_ST_REG (0x00a0)

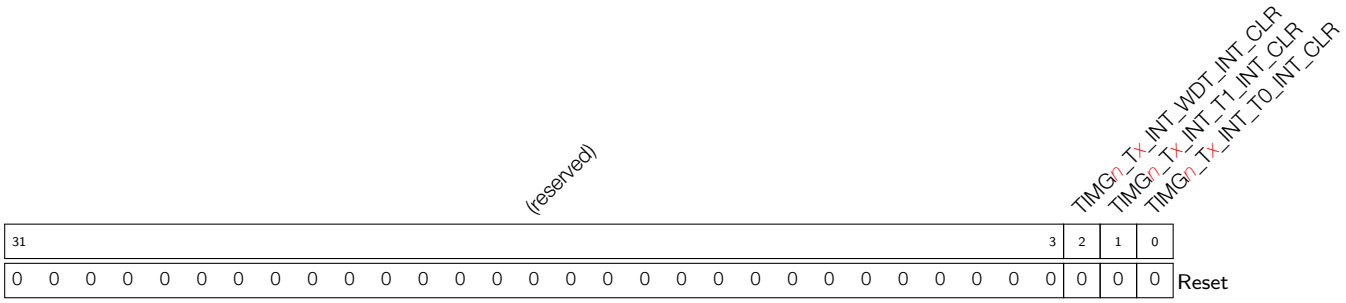


TIMG*n*_T*x*_INT_WDT_INT_ST The masked interrupt status bit for the TIMG*n*_T*x*_INT_WDT_INT interrupt. (RO)

TIMG*n*_T*x*_INT_T1_INT_ST The masked interrupt status bit for the TIMG*n*_T*x*_INT_T1_INT interrupt. (RO)

TIMG*n*_T*x*_INT_T0_INT_ST The masked interrupt status bit for the TIMG*n*_T*x*_INT_T0_INT interrupt. (RO)

Register 18.21: TIMG*n*_T*x*_INT_CLR_REG (0x00a4)



TIMG*n*_T*x*_INT_WDT_INT_CLR Set this bit to clear the TIMG*n*_T*x*_INT_WDT_INT interrupt. (WO)

TIMG*n*_T*x*_INT_T1_INT_CLR Set this bit to clear the TIMG*n*_T*x*_INT_T1_INT interrupt. (WO)

TIMG*n*_T*x*_INT_T0_INT_CLR Set this bit to clear the TIMG*n*_T*x*_INT_T0_INT interrupt. (WO)

19. Watchdog Timers

19.1 Introduction

The ESP32 has three watchdog timers: one in each of the two timer modules (called Main System Watchdog Timer, or MWDT) and one in the RTC module (which is called the RTC Watchdog Timer, or RWDT). These watchdog timers are intended to recover from an unforeseen fault, causing the application program to abandon its normal sequence. A watchdog timer has four stages. Each stage may take one out of three or four actions upon the expiry of a programmed period of time for this stage, unless the watchdog is fed or disabled. The actions are: interrupt, CPU reset, core reset and system reset. Only the RWDT can trigger the system reset, and is able to reset the entire chip and the main system including the RTC itself. A timeout value can be set for each stage individually.

During flash boot, the RWDT and the first MWDT start automatically in order to detect and recover from booting problems.

19.2 Features

- Four stages, each of which can be configured or disabled separately
- Programmable time period for each stage
- One out of three or four possible actions (interrupt, CPU reset, core reset and system reset) upon the expiry of each stage
- 32-bit expiry counter
- Write protection, to prevent the RWDT and MWDT configuration from being inadvertently altered.
- Flash boot protection
  If the boot process from an SPI flash does not complete within a predetermined period of time, the watchdog will reboot the entire main system.

19.3 Functional Description

19.3.1 Clock

The RWDT is clocked from the RTC slow clock, which usually will be 32 KHz. The MWDT clock source is derived from the APB clock via a pre-MWDT 16-bit configurable prescaler. For either watchdog, the clock source is fed into the 32-bit expiry counter. When this counter reaches the timeout value of the current stage, the action configured for the stage will execute, the expiry counter will be reset and the next stage will become active.

19.3.1.1 Operating Procedure

When a watchdog timer is enabled, it will proceed in loops from stage 0 to stage 3, then back to stage 0 and start again. The expiry action and time period for each stage can be configured individually.

Every stage can be configured for one of the following actions when the expiry timer reaches the stage's timeout value:

- Trigger an interrupt
  When the stage expires an interrupt is triggered.
- Reset a CPU core
  When the stage expires the designated CPU core will be reset. MWDT0 CPU reset only resets the PRO CPU. MWDT1 CPU reset only resets the APP CPU. The RWDT CPU reset can reset either of them, or both, or none, depending on configuration.
- Reset the main system
  When the stage expires, the main system, including the MWDTs, will be reset. In this article, the main system includes the CPU and all peripherals. The RTC is an exception to this, and it will not be reset.
- Reset the main system and RTC
  When the stage expires the main system and the RTC will both be reset. This action is only available in the RWDT.
- Disabled
  This stage will have no effects on the system.

When software feeds the watchdog timer, it returns to stage 0 and its expiry counter restarts from 0.

19.3.1.2 Write Protection

Both the MWDTs, as well as the RWDT, can be protected from accidental writing. To accomplish this, they have a write-key register (TIMERS_WDT_WKEY for the MWDT, RTC_CNTL_WDT_WKEY for the RWDT.) On reset, these registers are initialized to the value 0x50D83AA1. When the value in this register is changed from 0x50D83AA1, write protection is enabled. Writes to any WDT register, including the feeding register (but excluding the write-key register itself), are ignored. The recommended procedure for accessing a WDT is:

1. Disable the write protection
2. Make the required modification or feed the watchdog
3. Re-enable the write protection

19.3.1.3 Flash Boot Protection

During flash booting, the MWDT in timer group 0 (TIMG0), as well as the RWDT, are automatically enabled. Stage 0 for the enabled MWDT is automatically configured to reset the system upon expiry; stage 0 for the RWDT resets the RTC when it expires. After booting, the register TIMERS_WDT_FLASHBOOT_MOD_EN should be cleared to stop the flash boot protection procedure for the MWDT, and RTC_CNTL_WDT_FLASHBOOT_MOD_EN should be cleared to do the same for the RWDT. After this, the MWDT and RWDT can be configured by software.

19.3.1.4 Registers

The MWDT registers are part of the timer submodule and are described in the Timer Registers section. The RWDT registers are part of the RTC submodule and are described in the RTC Registers section.

20. eFuse Controller

20.1 Introduction

The ESP32 has a number of eFuses which store system parameters. Fundamentally, an eFuse is a single bit of non-volatile memory with the restriction that once an eFuse bit is programmed to 1, it can never be reverted to 0. Software can instruct the eFuse Controller to program each bit for each system parameter as needed.

Some of these system parameters can be read by software using the eFuse Controller. Some of the system parameters are also directly used by hardware modules.

20.2 Features

- Configuration of 27 system parameters
- Optional write-protection
- Optional software-read-protection

20.3 Functional Description

20.3.1 Structure

Twenty-seven system parameters with different bit width are stored in the eFuses. The name of each system parameter and the corresponding bit width are shown in Table 78. Among those parameters, efuse_wr_disable, efuse_rd_disable, BLK3_part_reserve and coding_scheme are directly used by the eFuse Controller.

Table 78: System Parameter

Name	Bit width	Program -Protection by efuse_wr_disable	Software-Read -Protection by efuse_rd_disable	Description
efuse_wr_disable	16	1	-	controls the eFuse Controller
efuse_rd_disable	4	0	-	controls the eFuse Controller
flash_crypt_cnt	8	2	-	governs the flash encryption/ decryption
WIFI_MAC_Address	56	3	-	Wi-Fi MAC address and CRC
SPI_pad_config_hd	5	3	-	configures the SPI I/O to a certain pad
chip_version	4	3	-	chip version
XPD_SDIO_REG	1	5	-	powers up the flash regulator
SDIO_TIEH	1	5	-	configures the flash regulator voltage: set to 1 for 3.3 V and set to 0 for 1.8 V
sdio_force	1	5	-	determines whether XPD_SDIO_REG and SDIO_TIEH can control the flash regulator

Name	Bit width	Program -Protection by efuse_wr_disable	Software-Read -Protection by efuse_rd_disable	Description
BLK3_part_reseve	2	10	3	controls the eFuse controller
SPI_pad_config_clk	5	6	-	configures the SPI I/O to a certain pad
SPI_pad_config_q	5	6	-	configures the SPI I/O to a certain pad
SPI_pad_config_d	5	6	-	configures the SPI I/O to a certain pad
SPI_pad_config_cs0	5	6	-	configures the SPI I/O to a certain pad
flash_crypt_config	4	10	3	governs flash encryption/ decryption
coding_scheme*	2	10	3	controls the eFuse Controller
console_debug_disable	1	15	-	Disables the ROM BASIC debug console fallback mode when set to 1
abstract_done_0	1	12	-	determines the status of Secure Boot
abstract_done_1	1	13	-	determines the status of Secure Boot
JTAG_disable	1	14	-	disables access to the JTAG controllers so as to effectively disable external use of JTAG
download_dis_encrypt	1	15	-	governs flash encryption/ decryption
download_dis_decrypt	1	15	-	governs flash encryption/ decryption
download_dis_cache	1	15	-	disables cache when boot mode is the Download Mode
key_status	1	10	3	determines whether BLOCK3 is deployed for user purposes
BLOCK1*	256/192/128	7	0	governs flash encryption/ decryption
BLOCK2*	256/192/128	8	1	key for Secure Boot
BLOCK3*	256/192/128	9	2	key for user purposes

20.3.1.1 System Parameter efuse_wr_disable

The system parameter efuse_wr_disable determines whether all of the system parameters are write-protected. Since efuse_wr_disable is a system parameter as well, it also determines whether it itself is write-protected.

If a system parameter is not write-protected, its unprogrammed bits can be programmed from 0 to 1. The bits

previously programmed to 1 will remain 1. When a system parameter is write-protected, none of its bits can be programmed: The unprogrammed bits will always remain 0 and the programmed bits will always remain 1.

The write-protection status of each system parameter corresponds to a bit in efuse_wr_disable. When the corresponding bit is set to 0, the system parameter is not write-protected. When the corresponding bit is set to 1, the system parameter is write-protected. If a system parameter is already write-protected, it will remain write-protected. The column entitled "Program-Protection by efuse_wr_disable" in Table 78 lists the corresponding bits that determine the write-protection status of each system parameter.

20.3.1.2 System Parameter efuse_rd_disable

Of the 26 system parameters, 20 are not constrained by software-read-protection. These are marked by "-" in the column entitled "Software-Read-Protection by efuse_rd_disable" in Table 78. Those system parameters, some of which are used by software and hardware modules at the same time, can be read by software via the eFuse Controller at any time.

When not software-read-protected, the other six system parameters can both be read by software and used by hardware modules. When they are software-read-protected, they can only be used by the hardware modules.

The column "Software-Read-Protection by efuse_rd_disable" in Table 78 lists the corresponding bits in efuse_rd_disable that determine the software read-protection status of the six system parameters. If a bit in the system parameter efuse_rd_disable is 0, the system parameter controlled by the bit is not software-read-protected. If a bit in the system parameter efuse_rd_disable is 1, the system parameter controlled by the bit is software-read-protected. If a system parameter is software-read-protected, it will remain in this state.

20.3.1.3 System Parameter coding_scheme

As Table 78 shows, only three system parameters, BLOCK1, BLOCK2, and BLOCK3, have variable bit width. Their bit width is controlled by another system parameter, coding_scheme. Despite their variable bit width, BLOCK1, BLOCK2, and BLOCK3 are assigned a fixed number of bits in eFuse. There is an encoding mapping between these three system parameters and their corresponding stored values in eFuse. For details please see Table 79.

Table 79: BLOCK1/2/3 Encoding

coding_scheme[1:0]	Width of BLOCK1/2/3	Coding scheme	Number of bits in eFuse
00/11	256	None	256
01	192	3/4	256
10	128	Repeat	256

The three coding schemes are explained as follows:

- $BLOCKN$ represents any of the following three system parameters: BLOCK1, BLOCK2 or BLOCK3.
- $BLOCKN[255:0]$, $BLOCKN[191:0]$, and $BLOCKN[127:0]$ represent each bit of the three system parameters in the three encoding schemes.

- ${}^{e}BLOCKN[255:0]$ represents each corresponding bit of those system parameters in eFuse after being encoded.

None

$$ {}^{e}BLOCKN[255:0] = BLOCKN[255:0] $$

3/4

$$ BLOCKN_i^j[7:0] = BLOCKN[48i+8j+7:48i+8j] \qquad i \in \{0,1,2,3\} \qquad j \in \{0,1,2,3,4,5\} $$
$$ {}^{e}BLOCKN_i^j[7:0] = {}^{e}BLOCKN[64i+8j+7:64i+8j] \qquad i \in \{0,1,2,3\} \qquad j \in \{0,1,2,3,4,5,6,7\} $$

$$ {}^{e}BLOCKN_i^j[7:0] = \begin{cases} BLOCKN_i^j[7:0] & j \in \{0,1,2,3,4,5\} \\ BLOCKN_i^0[7:0] \oplus BLOCKN_i^1[7:0] \\ \quad \oplus BLOCKN_i^2[7:0] \oplus BLOCKN_i^3[7:0] & j \in \{6\} \\ \quad \oplus BLOCKN_i^4[7:0] \oplus BLOCKN_i^5[7:0] \\ \sum_{l=0}^{5}(l+1)\sum_{k=0}^{7} BLOCKN_i^l[k] & j \in \{7\} \end{cases} \qquad i \in \{0,1,2,3\} $$

$\oplus$ means bitwise XOR

$\sum$ and $+$ mean summation

Repeat

$$ {}^{e}BLOCKN[255:128] = {}^{e}BLOCKN[127:0] = BLOCKN[127:0] $$

20.3.1.4 BLK3_part_reserve

System parameters coding_scheme, BLOCK1, BLOCK2, and BLOCK3 are controlled by the parameter BLK3_part_reserve.

When the value of BLK3_part_reserve is 0, coding_scheme, BLOCK1, BLOCK2, and BLOCK3 can be set to any value.

When the value of BLK3_part_reserve is 1, coding_scheme BLOCK1 BLOCK2 and BLOCK3 are controlled by 3/4 coding scheme. Meanwhile, $BLOCK3[143:96]$, namely, ${}^{e}BLOCK3[191:128]$ is unavailable.

20.3.2 Programming of System Parameters

The programming of variable-length system parameters BLOCK1, BLOCK2, and BLOCK3 is different from that of the fixed-length system parameters. **We program the ${}^{e}BLOCKN[255:0]$ value of encoded system parameters BLOCK1, BLOCK2, and BLOCK3 instead of directly programming the system parameters. The bit width of ${}^{e}BLOCKN[255:0]$ is always 256**. Fixed-length system parameters, in contrast, are programmed without encoding them first.

Each bit of the 24 fixed-length system parameters and the three encoded variable-length system parameters corresponds to a program register bit, as shown in Table 80. The register bits will be used when programming system parameters.

Table 80: Program Register

System parameter			Register	
Name	Width	Bit	Name	Bit
efuse_wr_disable	16	[15:0]	EFUSE_BLK0_WDATA0_REG	[15:0]
efuse_rd_disable	4	[3:0]		[19:16]
flash_crypt_cnt	8	[7:0]		[27:20]
WIFI_MAC_Address	56	[31:0]	EFUSE_BLK0_WDATA1_REG	[31:0]
		[55:32]	EFUSE_BLK0_WDATA2_REG	[23:0]
SPI_pad_config_hd	5	[4:0]	EFUSE_BLK0_WDATA3_REG	[8:4]
chip_version	4	[3:0]		[12:9]
BLK3_part_reserve	1	[0]		[14]
XPD_SDIO_REG	1	[0]	EFUSE_BLK0_WDATA4_REG	[14]
SDIO_TIEH	1	[0]		[15]
sdio_force	1	[0]		[16]
SPI_pad_config_clk	5	[4:0]	EFUSE_BLK0_WDATA5_REG	[4:0]
SPI_pad_config_q	5	[4:0]		[9:5]
SPI_pad_config_d	5	[4:0]		[14:10]
SPI_pad_config_cs0	5	[4:0]		[19:15]
flash_crypt_config	4	[3:0]		[31:28]
coding_scheme	2	[1:0]	EFUSE_BLK0_WDATA6_REG	[1:0]
console_debug_disable	1	[0]		[2]
abstract_done_0	1	[0]		[4]
abstract_done_1	1	[0]		[5]
JTAG_disable	1	[0]		[6]
download_dis_encrypt	1	[0]		[7]
download_dis_decrypt	1	[0]		[8]
download_dis_cache	1	[0]		[9]
key_status	1	[0]		[10]
BLOCK1	256/192/128	[31:0]	EFUSE_BLK1_WDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK1_WDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK1_WDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK1_WDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK1_WDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK1_WDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK1_WDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK1_WDATA7_REG	[31:0]
BLOCK2	256/192/128	[31:0]	EFUSE_BLK2_WDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK2_WDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK2_WDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK2_WDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK2_WDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK2_WDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK2_WDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK2_WDATA7_REG	[31:0]

System parameter			Register	
Name	Width	Bit	Name	Bit
BLOCK3	256/192/128	[31:0]	EFUSE_BLK3_WDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK3_WDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK3_WDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK3_WDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK3_WDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK3_WDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK3_WDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK3_WDATA7_REG	[31:0]

The process of programming system parameters is as follows:

1. Configure EFUSE_CLK_SEL0 bit, EFUSE_CLK_SEL1 bit of register EFUSE_CLK, and EFUSE_DAC_CLK_DIV bit of register EFUSE_DAC_CONF.
2. Set the corresponding register bit of the system parameter bit to be programmed to 1.
3. Write 0x5A5A into register EFUSE_CONF.
4. Write 0x2 into register EFUSE_CMD.
5. Poll register EFUSE_CMD until it is 0x0, or wait for a program-done interrupt.
6. Write 0x5AA5 into register EFUSE_CONF.
7. Write 0x1 into register EFUSE_CMD.
8. Poll register EFUSE_CMD until it is 0x0, or wait for a read-done interrupt.
9. Set the corresponding register bit of the programmed bit to 0.

The configuration values of the EFUSE_CLK_SEL0 bit, EFUSE_CLK_SEL1 bit of register EFUSE_CLK, and the EFUSE_DAC_CLK_DIV bit of register EFUSE_DAC_CONF are based on the current APB_CLK frequency, as is shown in Table 81.

Table 81: Timing Configuration

Configuration Value / APB_CLK Frequency Register		26 MHz	40 MHz	80 MHz
EFUSE_CLK	EFUSE_CLK_SEL0[7:0]	8'd250	8'd160	8'd80
	EFUSE_CLK_SEL1[7:0]	8'd255	8'd255	8'd128
EFUSE_DAC_CONF	EFUSE_DAC_CLK_DIV[7:0]	8'd52	8'd80	8'd160

The two methods to identify the generation of program/read-done interrupts are as follows:

Method One:

1. Poll bit 1/0 in register EFUSE_INT_RAW until bit 1/0 is 1, which represents the generation of an program/read-done interrupt.
2. Set the bit 1/0 in register EFUSE_INT_CLR to 1 to clear the program/read-done interrupts.

Method Two:

1. Set bit 1/0 in register EFUSE_INT_ENA to 1 to enable eFuse Controller to post a program/read-done interrupt.
2. Configure Interrupt Matrix to enable the CPU to respond to an EFUSE_INT interrupt.
3. A program/read-done interrupt is generated.
4. Read bit 1/0 in register EFUSE_INT_ST to identify the generation of the program/read-done interrupt.
5. Set bit 1/0 in register EFUSE_INT_CLR to 1 to clear the program/read-done interrupt.

The programming of different system parameters and even the programming of different bits of the same system parameter can be completed separately in multiple programmings. It is, however, recommended that users minimize programming cycles, and program all the bits that need to be programmed in a system parameter in one programming action. In addition, after all system parameters controlled by a certain bit of efuse_wr_disable are programmed, that bit should be immediately programmed. The programming of system parameters controlled by a certain bit of efuse_wr_disable, and the programming of that bit can even be completed at the same time. **Repeated programming of programmed bits is strictly forbidden.**

20.3.3 Software Reading of System Parameters

Each bit of the 24 fixed-length system parameters and the three variable-length system parameters corresponds to a software-read register bit, as shown in Table 82. Software can use the value of each system parameter by reading the value in the corresponding register.

The bit width of system parameters BLOCK1, BLOCK2, and BLOCK3 is variable. Although 256 register bits have been assigned to each of the three parameters, as shown in Table 82, some of the 256 register bits are useless in the 3/4 coding and the Repeat coding scheme. In the None coding scheme, the corresponding register bit of each bit of $BLOCKN[255:0]$ is used. In the 3/4 coding scheme, only the corresponding register bits of $BLOCKN[191:0]$ are useful. In Repeat coding scheme, only the corresponding bits of $BLOCKN[127:0]$ are useful. In different coding schemes, the values of useless register bits read by software are invalid. **The values of useful register bits read by software are the system parameters BLOCK1, BLOCK2, and BLOCK3 themselves instead of their values after being encoded.**

Table 82: Software Read Register

System parameter			Register	
Name	Bit Width	Bit	Name	Bit
efuse_wr_disable	16	[15:0]	EFUSE_BLK0_RDATA0_REG	[15:0]
efuse_rd_disable	4	[3:0]		[19:16]
flash_crypt_cnt	8	[7:0]		[27:20]
WIFI_MAC_Address	56	[31:0]	EFUSE_BLK0_RDATA1_REG	[31:0]
		[55:32]	EFUSE_BLK0_RDATA2_REG	[23:0]
SPI_pad_config_hd	5	[4:0]	EFUSE_BLK0_WDATA3_REG	[8:4]
chip_version	4	[3:0]		[12:9]
BLK3_part_reserve	1	[0]		[14]
XPD_SDIO_REG	1	[0]	EFUSE_BLK0_RDATA4_REG	[14]
SDIO_TIEH	1	[0]		[15]
sdio_force	1	[0]		[16]
SPI_pad_config_clk	5	[4:0]	EFUSE_BLK0_RDATA5_REG	[4:0]
SPI_pad_config_q	5	[4:0]		[9:5]

System parameter			Register	
Name	Bit Width	Bit	Name	Bit
SPI_pad_config_d	5	[4:0]		[14:10]
SPI_pad_config_cs0	5	[4:0]		[19:15]
flash_crypt_config	4	[3:0]		[31:28]
coding_scheme	2	[1:0]	EFUSE_BLK0_RDATA6_REG	[1:0]
console_debug_disable	1	[0]		[2]
abstract_done_0	1	[0]		[4]
abstract_done_1	1	[0]		[5]
JTAG_disable	1	[0]		[6]
download_dis_encrypt	1	[0]		[7]
download_dis_decrypt	1	[0]		[8]
download_dis_cache	1	[0]		[9]
key_status	1	[0]		[10]
BLOCK1	256/192/128	[31:0]	EFUSE_BLK1_RDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK1_RDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK1_RDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK1_RDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK1_RDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK1_RDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK1_RDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK1_RDATA7_REG	[31:0]
BLOCK2	256/192/128	[31:0]	EFUSE_BLK2_RDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK2_RDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK2_RDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK2_RDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK2_RDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK2_RDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK2_RDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK2_RDATA7_REG	[31:0]
BLOCK3	256/192/128	[31:0]	EFUSE_BLK3_RDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK3_RDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK3_RDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK3_RDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK3_RDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK3_RDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK3_RDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK3_RDATA7_REG	[31:0]

20.3.4 The Use of System Parameters by Hardware Modules

Hardware modules are directly hardwired to the ESP32 in order to use the system parameters. Software cannot change this behaviour. **Hardware modules use the decoded values of system parameters BLOCK1, BLOCK2, and BLOCK3, not their encoded values.**

20.3.5 Interrupts

- EFUSE_PGM_DONE_INT: Triggered when eFuse programming has finished.
- EFUSE_READ_DONE_INT: Triggered when eFuse reading has finished.

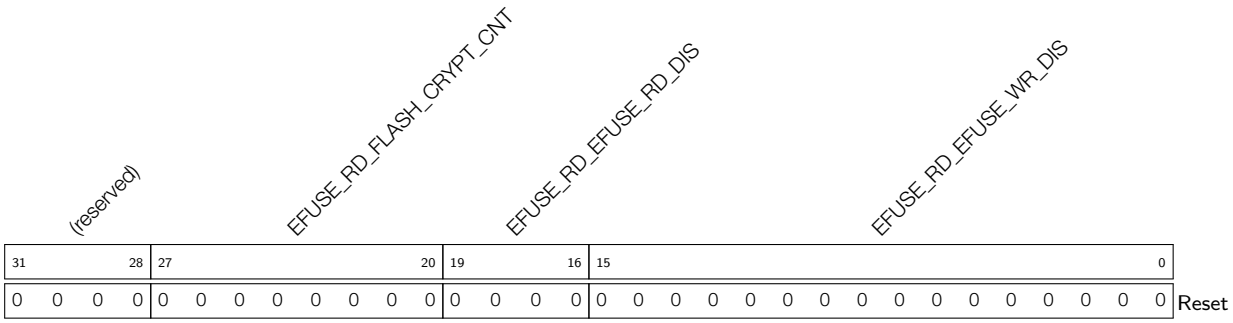
20.4 Register Summary

Name	Description	Address	Access
eFuse data read registers			
EFUSE_BLK0_RDATA0_REG	Returns data word 0 in eFuse BLOCK 0	0x3FF5A000	RO
EFUSE_BLK0_RDATA1_REG	Returns data word 1 in eFuse BLOCK 0	0x3FF5A004	RO
EFUSE_BLK0_RDATA2_REG	Returns data word 2 in eFuse BLOCK 0	0x3FF5A008	RO
EFUSE_BLK0_RDATA3_REG	Returns data word 3 in eFuse BLOCK 0	0x3FF5A00C	RO
EFUSE_BLK0_RDATA4_REG	Returns data word 4 in eFuse BLOCK 0	0x3FF5A010	RO
EFUSE_BLK0_RDATA5_REG	Returns data word 5 in eFuse BLOCK 0	0x3FF5A014	RO
EFUSE_BLK0_RDATA6_REG	Returns data word 6 in eFuse BLOCK 0	0x3FF5A018	RO
EFUSE_BLK1_RDATA0_REG	Returns data word 0 in eFuse BLOCK 1	0x3FF5A038	RO
EFUSE_BLK1_RDATA1_REG	Returns data word 1 in eFuse BLOCK 1	0x3FF5A03C	RO
EFUSE_BLK1_RDATA2_REG	Returns data word 2 in eFuse BLOCK 1	0x3FF5A040	RO
EFUSE_BLK1_RDATA3_REG	Returns data word 3 in eFuse BLOCK 1	0x3FF5A044	RO
EFUSE_BLK1_RDATA4_REG	Returns data word 4 in eFuse BLOCK 1	0x3FF5A048	RO
EFUSE_BLK1_RDATA5_REG	Returns data word 5 in eFuse BLOCK 1	0x3FF5A04C	RO
EFUSE_BLK1_RDATA6_REG	Returns data word 6 in eFuse BLOCK 1	0x3FF5A050	RO
EFUSE_BLK1_RDATA7_REG	Returns data word 7 in eFuse BLOCK 1	0x3FF5A054	RO
EFUSE_BLK2_RDATA0_REG	Returns data word 0 in eFuse BLOCK 2	0x3FF5A058	RO
EFUSE_BLK2_RDATA1_REG	Returns data word 1 in eFuse BLOCK 2	0x3FF5A05C	RO
EFUSE_BLK2_RDATA2_REG	Returns data word 2 in eFuse BLOCK 2	0x3FF5A060	RO
EFUSE_BLK2_RDATA3_REG	Returns data word 3 in eFuse BLOCK 2	0x3FF5A064	RO
EFUSE_BLK2_RDATA4_REG	Returns data word 4 in eFuse BLOCK 2	0x3FF5A068	RO
EFUSE_BLK2_RDATA5_REG	Returns data word 5 in eFuse BLOCK 2	0x3FF5A06C	RO
EFUSE_BLK2_RDATA6_REG	Returns data word 6 in eFuse BLOCK 2	0x3FF5A070	RO
EFUSE_BLK2_RDATA7_REG	Returns data word 7 in eFuse BLOCK 2	0x3FF5A074	RO
EFUSE_BLK3_RDATA0_REG	Returns data word 0 in eFuse BLOCK 3	0x3FF5A078	RO
EFUSE_BLK3_RDATA1_REG	Returns data word 1 in eFuse BLOCK 3	0x3FF5A07C	RO
EFUSE_BLK3_RDATA2_REG	Returns data word 2 in eFuse BLOCK 3	0x3FF5A080	RO
EFUSE_BLK3_RDATA3_REG	Returns data word 3 in eFuse BLOCK 3	0x3FF5A084	RO
EFUSE_BLK3_RDATA4_REG	Returns data word 4 in eFuse BLOCK 3	0x3FF5A088	RO
EFUSE_BLK3_RDATA5_REG	Returns data word 5 in eFuse BLOCK 3	0x3FF5A08C	RO
EFUSE_BLK3_RDATA6_REG	Returns data word 6 in eFuse BLOCK 3	0x3FF5A090	RO
EFUSE_BLK3_RDATA7_REG	Returns data word 7 in eFuse BLOCK 3	0x3FF5A094	RO
eFuse data write registers			
EFUSE_BLK0_WDATA0_REG	Writes data to word 0 in eFuse BLOCK 0	0x3FF5A01c	R/W
EFUSE_BLK0_WDATA1_REG	Writes data to word 1 in eFuse BLOCK 0	0x3FF5A020	R/W
EFUSE_BLK0_WDATA2_REG	Writes data to word 2 in eFuse BLOCK 0	0x3FF5A024	R/W
EFUSE_BLK0_WDATA3_REG	Writes data to word 3 in eFuse BLOCK 0	0x3FF5A028	R/W

Name	Description	Address	Access
EFUSE_BLK0_WDATA4_REG	Writes data to word 4 in eFuse BLOCK 0	0x3FF5A02c	R/W
EFUSE_BLK0_WDATA5_REG	Writes data to word 5 in eFuse BLOCK 0	0x3FF5A030	R/W
EFUSE_BLK0_WDATA6_REG	Writes data to word 6 in eFuse BLOCK 0	0x3FF5A034	R/W
EFUSE_BLK1_WDATA0_REG	Writes data to word 0 in eFuse BLOCK 1	0x3FF5A098	R/W
EFUSE_BLK1_WDATA1_REG	Writes data to word 1 in eFuse BLOCK 1	0x3FF5A09c	R/W
EFUSE_BLK1_WDATA2_REG	Writes data to word 2 in eFuse BLOCK 1	0x3FF5A0a0	R/W
EFUSE_BLK1_WDATA3_REG	Writes data to word 3 in eFuse BLOCK 1	0x3FF5A0a4	R/W
EFUSE_BLK1_WDATA4_REG	Writes data to word 4 in eFuse BLOCK 1	0x3FF5A0a8	R/W
EFUSE_BLK1_WDATA5_REG	Writes data to word 5 in eFuse BLOCK 1	0x3FF5A0ac	R/W
EFUSE_BLK1_WDATA6_REG	Writes data to word 6 in eFuse BLOCK 1	0x3FF5A0b0	R/W
EFUSE_BLK1_WDATA7_REG	Writes data to word 7 in eFuse BLOCK 1	0x3FF5A0b4	R/W
EFUSE_BLK2_WDATA0_REG	Writes data to word 0 in eFuse BLOCK 2	0x3FF5A0b8	R/W
EFUSE_BLK2_WDATA1_REG	Writes data to word 1 in eFuse BLOCK 2	0x3FF5A0bc	R/W
EFUSE_BLK2_WDATA2_REG	Writes data to word 2 in eFuse BLOCK 2	0x3FF5A0c0	R/W
EFUSE_BLK2_WDATA3_REG	Writes data to word 3 in eFuse BLOCK 2	0x3FF5A0c4	R/W
EFUSE_BLK2_WDATA4_REG	Writes data to word 4 in eFuse BLOCK 2	0x3FF5A0c8	R/W
EFUSE_BLK2_WDATA5_REG	Writes data to word 5 in eFuse BLOCK 2	0x3FF5A0cc	R/W
EFUSE_BLK2_WDATA6_REG	Writes data to word 6 in eFuse BLOCK 2	0x3FF5A0d0	R/W
EFUSE_BLK2_WDATA7_REG	Writes data to word 7 in eFuse BLOCK 2	0x3FF5A0d4	R/W
EFUSE_BLK3_WDATA0_REG	Writes data to word 0 in eFuse BLOCK 3	0x3FF5A0d8	R/W
EFUSE_BLK3_WDATA1_REG	Writes data to word 1 in eFuse BLOCK 3	0x3FF5A0dc	R/W
EFUSE_BLK3_WDATA2_REG	Writes data to word 2 in eFuse BLOCK 3	0x3FF5A0e0	R/W
EFUSE_BLK3_WDATA3_REG	Writes data to word 3 in eFuse BLOCK 3	0x3FF5A0e4	R/W
EFUSE_BLK3_WDATA4_REG	Writes data to word 4 in eFuse BLOCK 3	0x3FF5A0e8	R/W
EFUSE_BLK3_WDATA5_REG	Writes data to word 5 in eFuse BLOCK 3	0x3FF5A0ec	R/W
EFUSE_BLK3_WDATA6_REG	Writes data to word 6 in eFuse BLOCK 3	0x3FF5A0f0	R/W
EFUSE_BLK3_WDATA7_REG	Writes data to word 7 in eFuse BLOCK 3	0x3FF5A0f4	R/W
Control registers			
EFUSE_CLK_REG	Timing configuration register	0x3FF5A0F8	R/W
EFUSE_CONF_REG	Opcode register	0x3FF5A0FC	R/W
EFUSE_CMD_REG	Read/write command register	0x3FF5A104	R/W
Interrupt registers			
EFUSE_INT_RAW_REG	Raw interrupt status	0x3FF5A108	RO
EFUSE_INT_ST_REG	Masked interrupt status	0x3FF5A10C	RO
EFUSE_INT_ENA_REG	Interrupt enable bits	0x3FF5A110	R/W
EFUSE_INT_CLR_REG	Interrupt clear bits	0x3FF5A114	WO
Misc registers			
EFUSE_DAC_CONF_REG	Efuse timing configuration	0x3FF5A118	R/W
EFUSE_DEC_STATUS_REG	Status of 3/4 coding scheme	0x3FF5A11C	RO

20.5 Registers

Register 20.1: EFUSE_BLK0_RDATA0_REG (0x000)

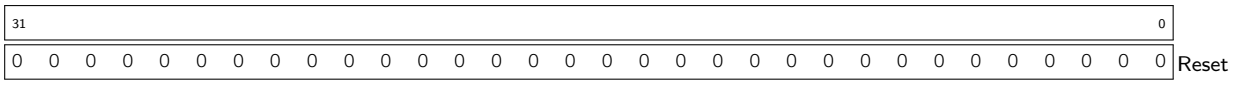


EFUSE_RD_FLASH_CRYPT_CNT This field returns the value of flash_crypt_cnt. (RO)

EFUSE_RD_EFUSE_RD_DIS This field returns the value of efuse_rd_disable. (RO)

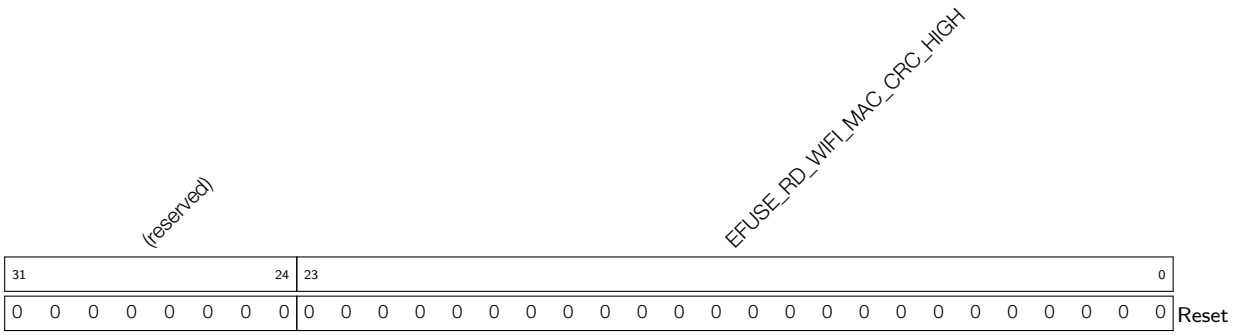
EFUSE_RD_EFUSE_WR_DIS This field returns the value of efuse_wr_disable. (RO)

Register 20.2: EFUSE_BLK0_RDATA1_REG (0x004)



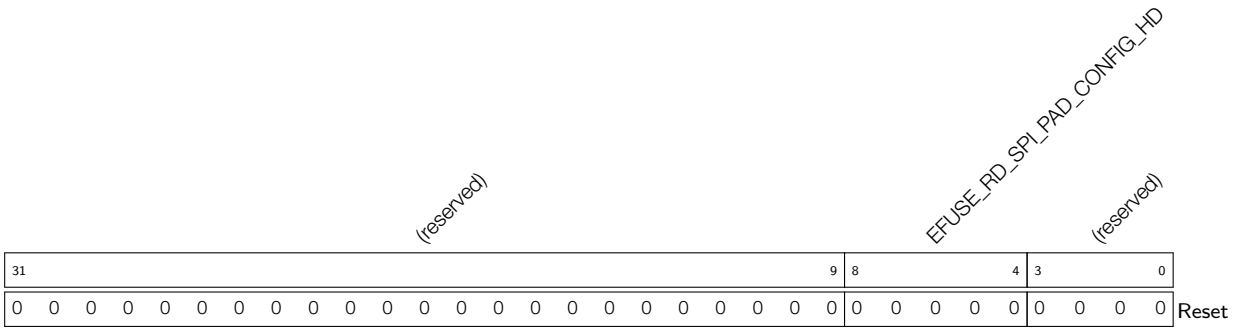
EFUSE_BLK0_RDATA1_REG This field returns the value of the lower 32 bits of WIFI_MAC_Address. (RO)

Register 20.3: EFUSE_BLK0_RDATA2_REG (0x008)



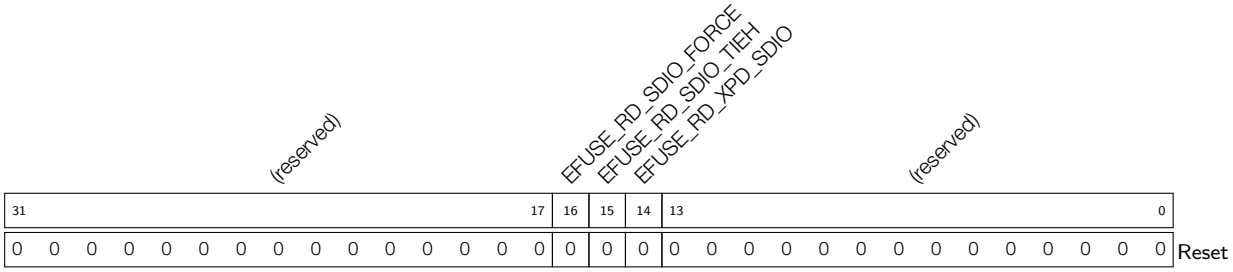
EFUSE_RD_WIFI_MAC_CRC_HIGH This field returns the value of the higher 24 bits of WIFI_MAC_Address. (RO)

Register 20.4: EFUSE_BLK0_RDATA3_REG (0x00c)



EFUSE_RD_SPI_PAD_CONFIG_HD This field returns the value of SPI_pad_config_hd. (RO)

Register 20.5: EFUSE_BLK0_RDATA4_REG (0x010)

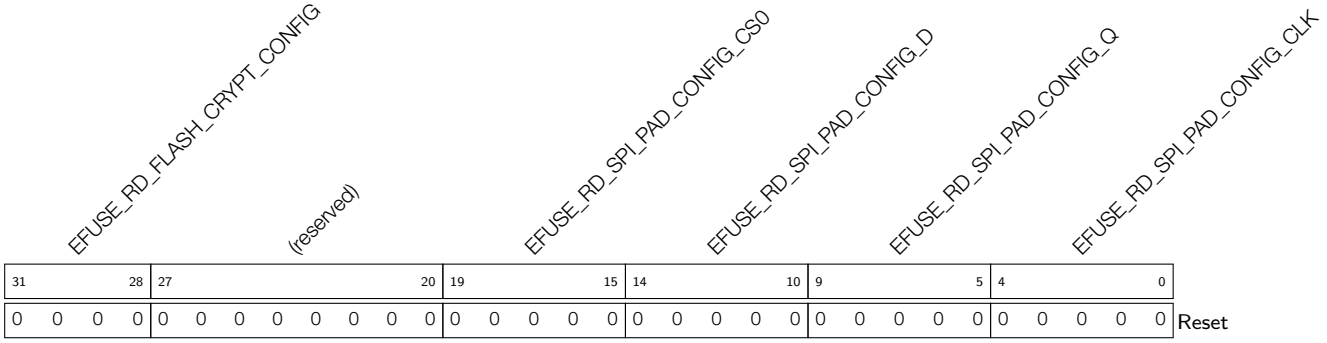


EFUSE_RD_SDIO_FORCE This field returns the value of sdio_force. (RO)

EFUSE_RD_SDIO_TIEH This field returns the value of SDIO_TIEH. (RO)

EFUSE_RD_XPD_SDIO This field returns the value of XPD_SDIO_REG. (RO)

Register 20.6: EFUSE_BLK0_RDATA5_REG (0x014)



EFUSE_RD_FLASH_CRYPT_CONFIG This field returns the value of flash_crypt_config. (RO)

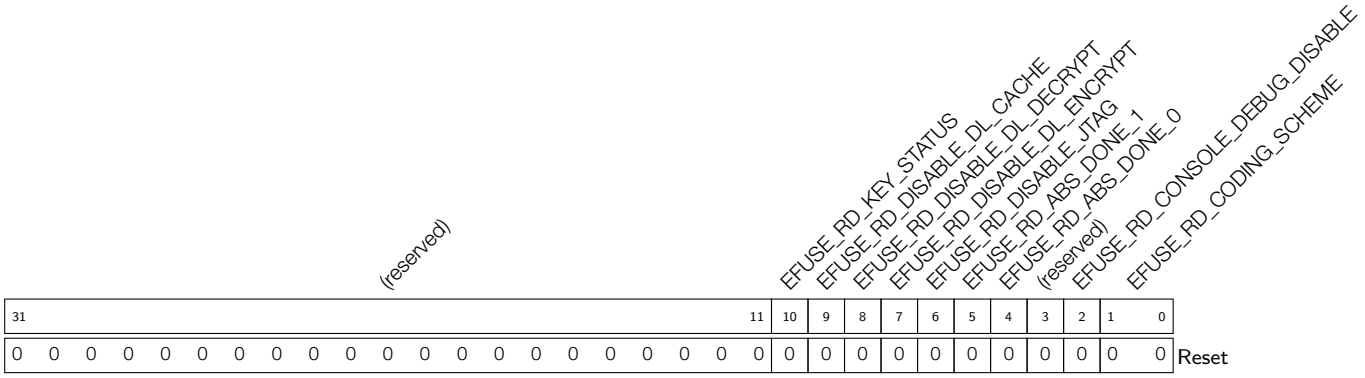
EFUSE_RD_SPI_PAD_CONFIG_CS0 This field returns the value of SPI_pad_config_cs0. (RO)

EFUSE_RD_SPI_PAD_CONFIG_D This field returns the value of SPI_pad_config_d. (RO)

EFUSE_RD_SPI_PAD_CONFIG_Q This field returns the value of SPI_pad_config_q. (RO)

EFUSE_RD_SPI_PAD_CONFIG_CLK This field returns the value of SPI_pad_config_clk. (RO)

Register 20.7: EFUSE_BLK0_RDATA6_REG (0x018)



EFUSE_RD_KEY_STATUS This field returns the value of key_status. (RO)

EFUSE_RD_DISABLE_DL_CACHE This field returns the value of download_dis_cache. (RO)

EFUSE_RD_DISABLE_DL_DECRYPT This field returns the value of download_dis_decrypt. (RO)

EFUSE_RD_DISABLE_DL_ENCRYPT This field returns the value of download_dis_encrypt. (RO)

EFUSE_RD_DISABLE_JTAG This field returns the value of JTAG_disable. (RO)

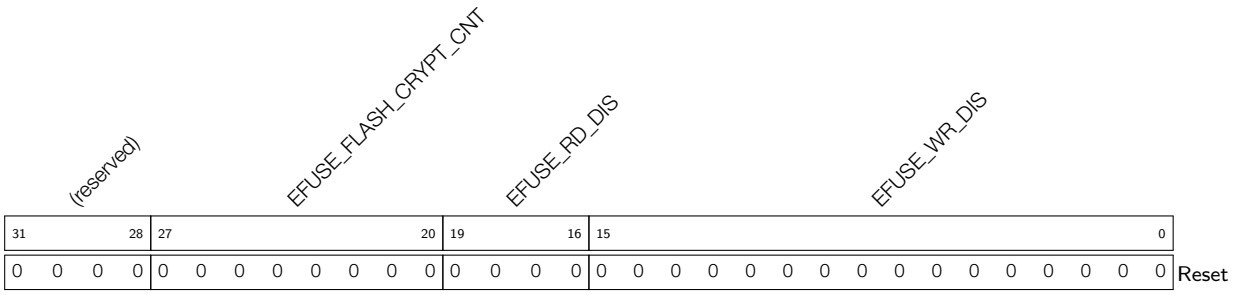
EFUSE_RD_ABS_DONE_1 This field returns the value of abstract_done_1. (RO)

EFUSE_RD_ABS_DONE_0 This field returns the value of abstract_done_0. (RO)

EFUSE_RD_CONSOLE_DEBUG_DISABLE This field returns the value of console_debug_disable. (RO)

EFUSE_RD_CODING_SCHEME This field returns the value of coding_scheme. (RO)

Register 20.8: EFUSE_BLK0_WDATA0_REG (0x01c)

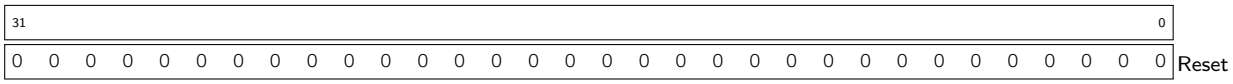


EFUSE_FLASH_CRYPT_CNT This field programs the value of flash_crypt_cnt. (R/W)

EFUSE_RD_DIS This field programs the value of efuse_rd_disable. (R/W)

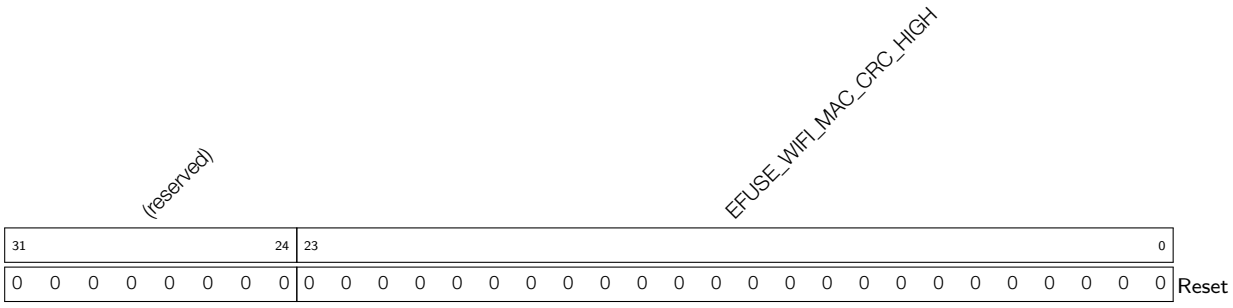
EFUSE_WR_DIS This field programs the value of efuse_wr_disable. (R/W)

Register 20.9: EFUSE_BLK0_WDATA1_REG (0x020)



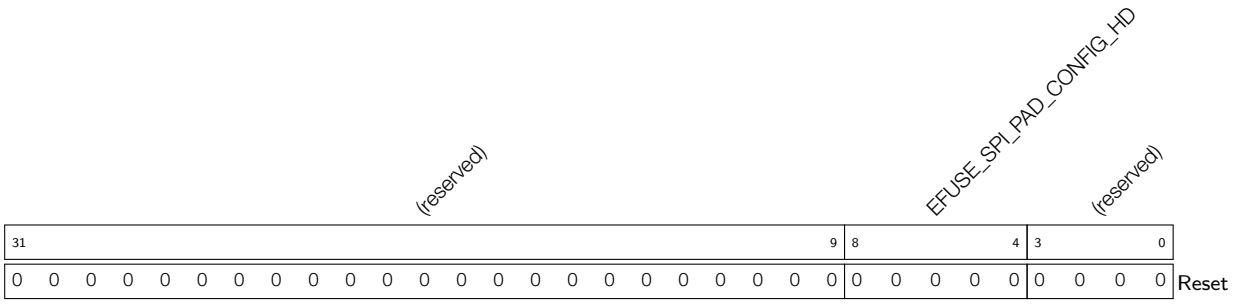
EFUSE_BLK0_WDATA1_REG This field programs the value of lower 32 bits of WIFI_MAC_Address. (R/W)

Register 20.10: EFUSE_BLK0_WDATA2_REG (0x024)



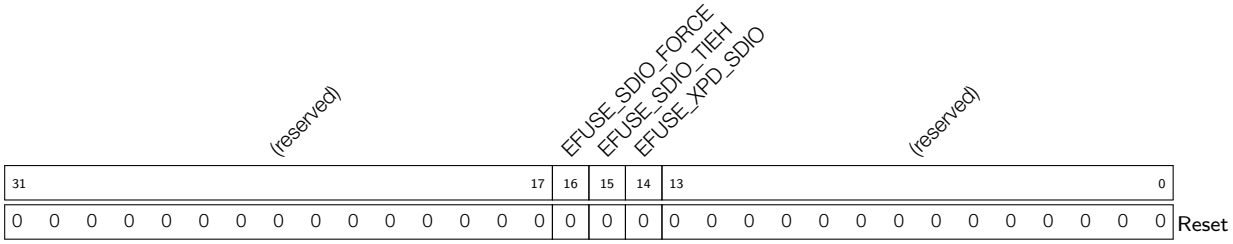
EFUSE_WIFI_MAC_CRC_HIGH This field programs the value of higher 24 bits of WIFI_MAC_Address. (R/W)

Register 20.11: EFUSE_BLK0_WDATA3_REG (0x028)



EFUSE_SPI_PAD_CONFIG_HD This field programs the value of SPI_pad_config_hd. (R/W)

Register 20.12: EFUSE_BLK0_WDATA4_REG (0x02c)

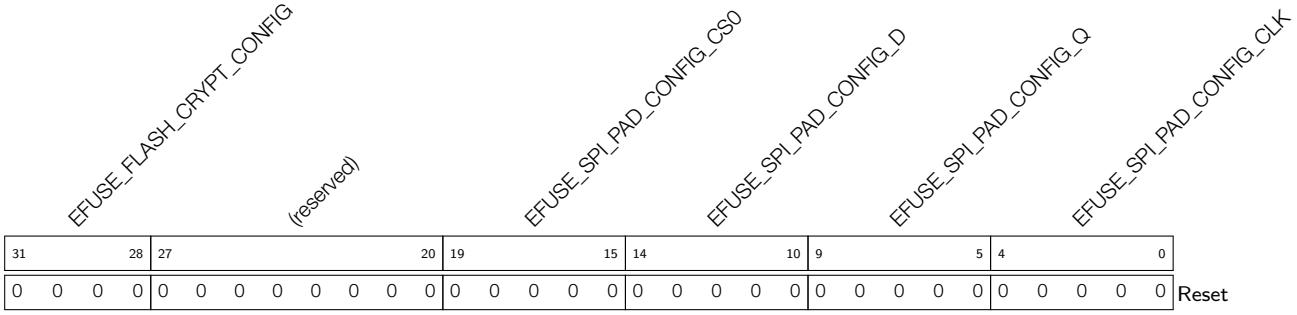


EFUSE_SDIO_FORCE This field programs the value of SDIO_TIEH. (R/W)

EFUSE_SDIO_TIEH This field programs the value of SDIO_TIEH. (R/W)

EFUSE_XPD_SDIO This field programs the value of XPD_SDIO_REG. (R/W)

Register 20.13: EFUSE_BLK0_WDATA5_REG (0x030)



EFUSE_FLASH_CRYPT_CONFIG This field programs the value of flash_crypt_config. (R/W)

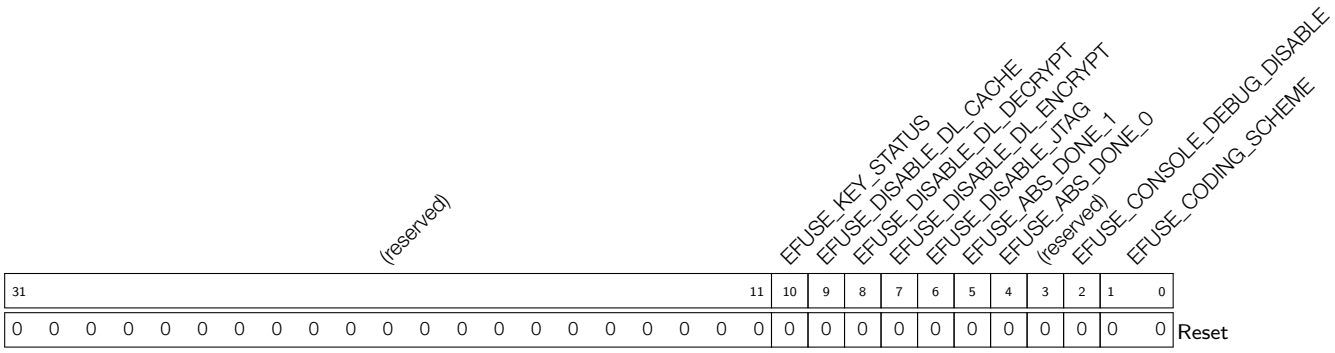
EFUSE_SPI_PAD_CONFIG_CS0 This field programs the value of SPI_pad_config_cs0. (R/W)

EFUSE_SPI_PAD_CONFIG_D This field programs the value of SPI_pad_config_d. (R/W)

EFUSE_SPI_PAD_CONFIG_Q This field programs the value of SPI_pad_config_q. (R/W)

EFUSE_SPI_PAD_CONFIG_CLK This field programs the value of SPI_pad_config_clk. (R/W)

Register 20.14: EFUSE_BLK0_WDATA6_REG (0x034)



EFUSE_KEY_STATUS This field programs the value of key_status. (R/W)

EFUSE_DISABLE_DL_CACHE This field programs the value of download_dis_cache. (R/W)

EFUSE_DISABLE_DL_DECRYPT This field programs the value of download_dis_decrypt. (R/W)

EFUSE_DISABLE_DL_ENCRYPT This field programs the value of download_dis_encrypt. (R/W)

EFUSE_DISABLE_JTAG This field programs the value of JTAG_disable. (R/W)

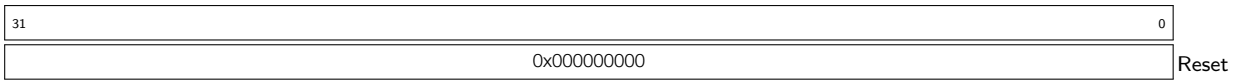
EFUSE_ABS_DONE_1 This field programs the value of abstract_done_1. (R/W)

EFUSE_ABS_DONE_0 This field programs the value of abstract_done_0. (R/W)

EFUSE_CONSOLE_DEBUG_DISABLE This field programs the value of console_debug_disable. (R/W)

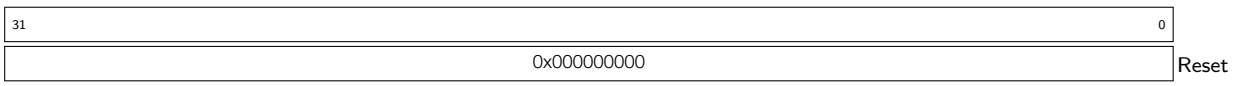
EFUSE_CODING_SCHEME This field programs the value of coding_scheme. (R/W)

Register 20.15: EFUSE_BLK1_RDATA*n*_REG (*n*: 0-7) (0x38+4n*)**



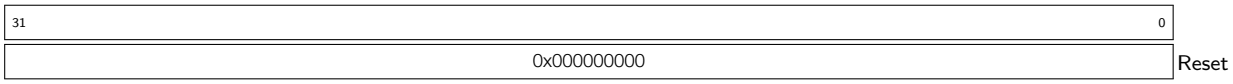
EFUSE_BLK1_RDATA*n*_REG This field returns the value of word *n* in BLOCK1. (RO)

Register 20.16: EFUSE_BLK2_RDATA*n*_REG (*n*: 0-7) (0x58+4n*)**



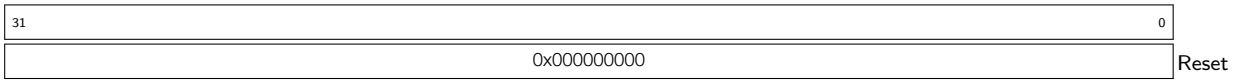
EFUSE_BLK2_RDATA*n*_REG This field returns the value of word *n* in BLOCK2. (RO)

Register 20.17: EFUSE_BLK3_RDATA*n*_REG (*n*: 0-7) (0x78+4n*)**



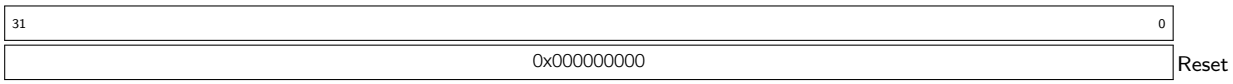
EFUSE_BLK3_RDATA*n*_REG This field returns the value of word *n* in BLOCK3. (RO)

Register 20.18: EFUSE_BLK1_WDATA*n*_REG (*n*: 0-7) (0x98+4n*)**



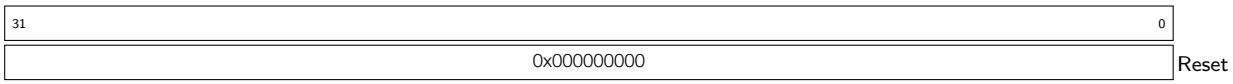
EFUSE_BLK1_WDATA*n*_REG This field programs the value of word *n* in of BLOCK1. (R/W)

Register 20.19: EFUSE_BLK2_WDATA*n*_REG (*n*: 0-7) (0xB8+4n*)**



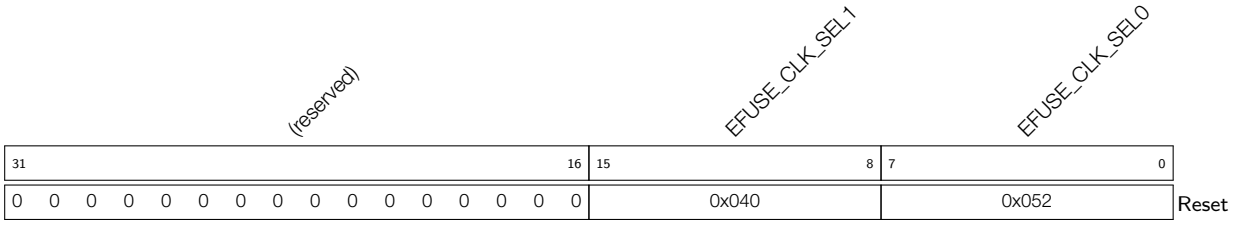
EFUSE_BLK2_WDATA*n*_REG This field programs the value of word *n* in of BLOCK2. (R/W)

Register 20.20: EFUSE_BLK3_WDATA*n*_REG (*n*: 0-7) (0xD8+4n*)**



EFUSE_BLK3_WDATA*n*_REG This field programs the value of word *n* in of BLOCK3. (R/W)

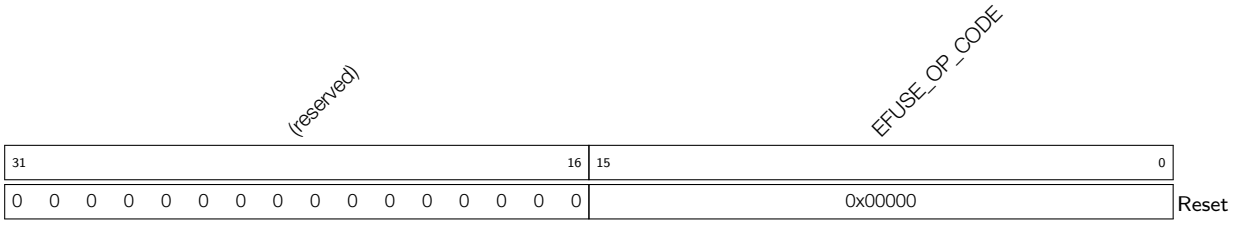
Register 20.21: EFUSE_CLK_REG (0x0f8)



EFUSE_CLK_SEL1 eFuse clock configuration field. (R/W)

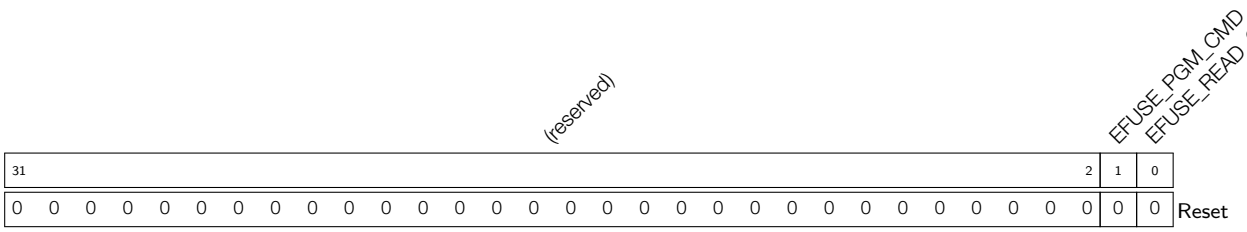
EFUSE_CLK_SEL0 eFuse clock configuration field. (R/W)

Register 20.22: EFUSE_CONF_REG (0x0fc)



EFUSE_OP_CODE eFuse operation code register. (R/W)

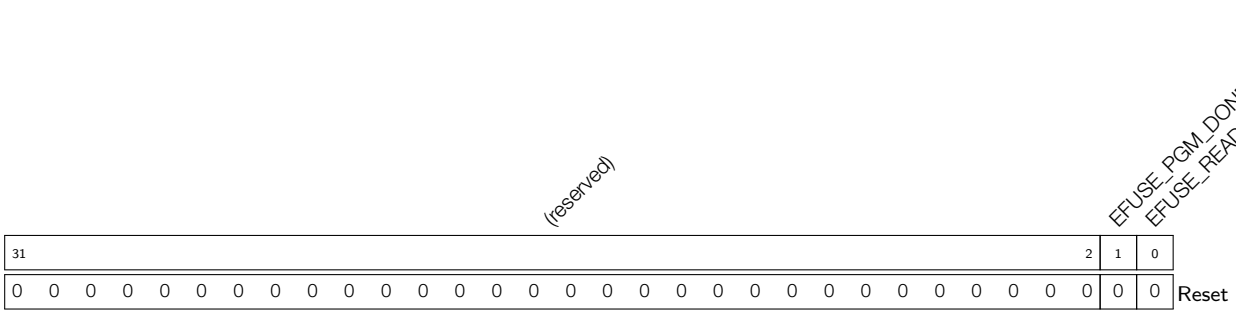
Register 20.23: EFUSE_CMD_REG (0x104)



EFUSE_PGM_CMD Set this to 1 to start a program operation. Reverts to 0 when the program operation is done. (R/W)

EFUSE_READ_CMD Set this to 1 to start a read operation. Reverts to 0 when the read operation is done. (R/W)

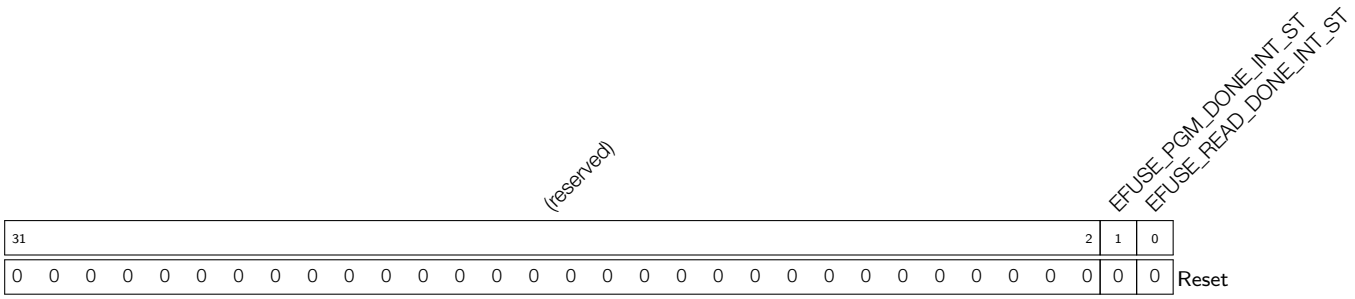
Register 20.24: EFUSE_INT_RAW_REG (0x108)



EFUSE_PGM_DONE_INT_RAW The raw interrupt status bit for the EFUSE_PGM_DONE_INT interrupt. (RO)

EFUSE_READ_DONE_INT_RAW The raw interrupt status bit for the EFUSE_READ_DONE_INT interrupt. (RO)

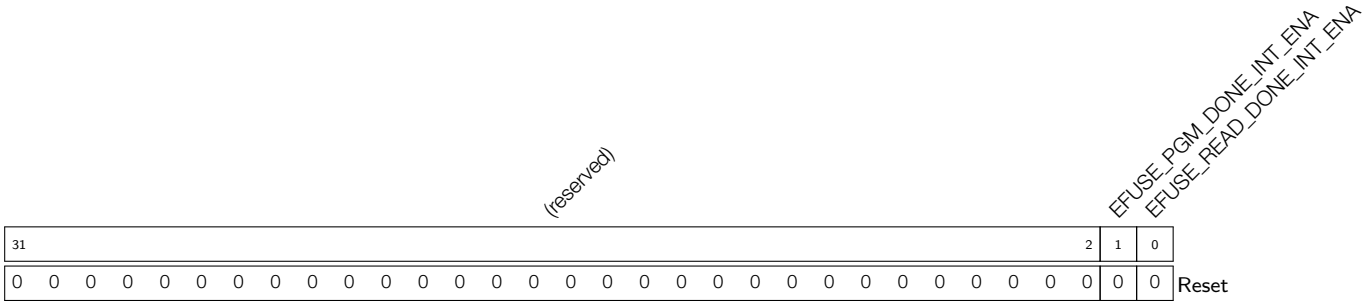
Register 20.25: EFUSE_INT_ST_REG (0x10c)



EFUSE_PGM_DONE_INT_ST The masked interrupt status bit for the EFUSE_PGM_DONE_INT interrupt. (RO)

EFUSE_READ_DONE_INT_ST The masked interrupt status bit for the EFUSE_READ_DONE_INT interrupt. (RO)

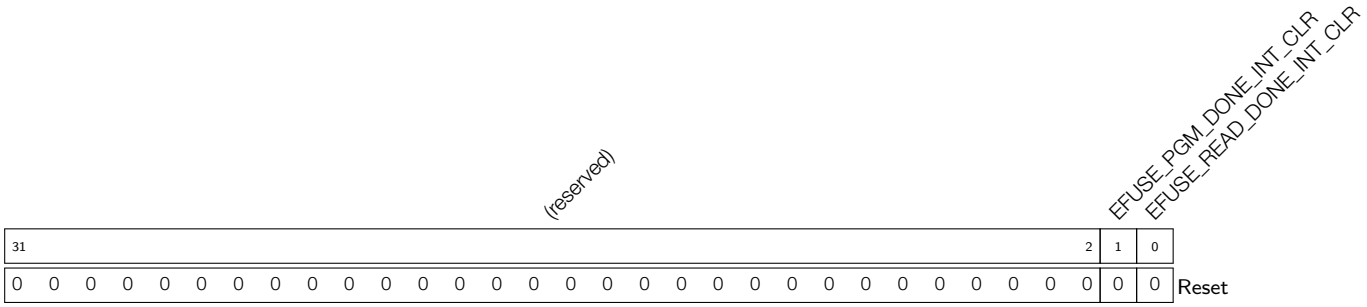
Register 20.26: EFUSE_INT_ENA_REG (0x110)



EFUSE_PGM_DONE_INT_ENA The interrupt enable bit for the EFUSE_PGM_DONE_INT interrupt. (R/W)

EFUSE_READ_DONE_INT_ENA The interrupt enable bit for the EFUSE_READ_DONE_INT interrupt. (R/W)

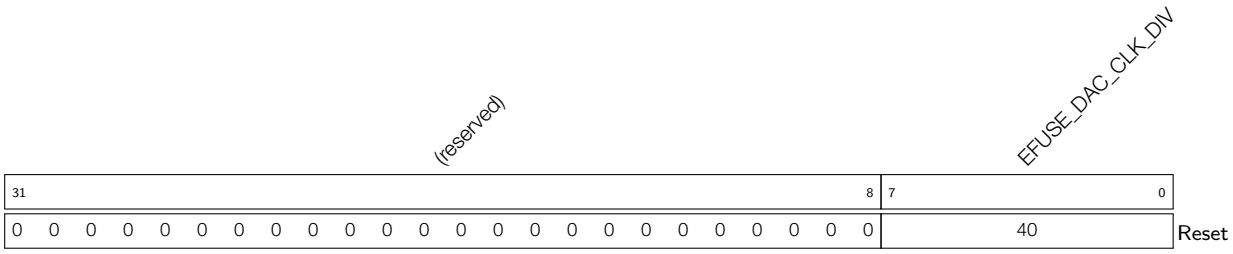
Register 20.27: EFUSE_INT_CLR_REG (0x114)



EFUSE_PGM_DONE_INT_CLR Set this bit to clear the EFUSE_PGM_DONE_INT interrupt. (WO)

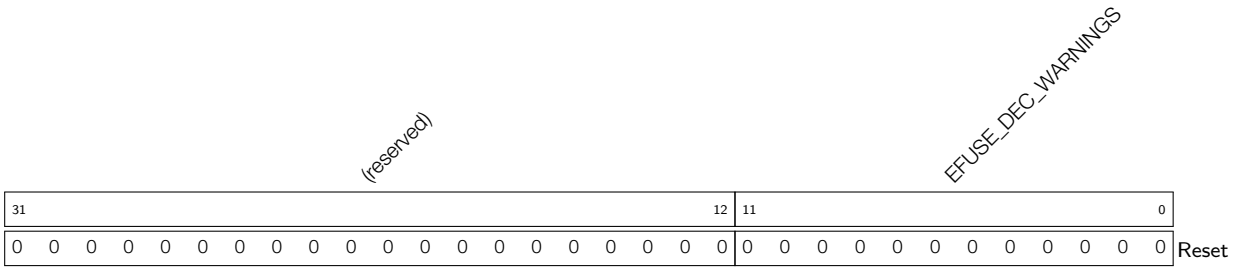
EFUSE_READ_DONE_INT_CLR Set this bit to clear the EFUSE_READ_DONE_INT interrupt. (WO)

Register 20.28: EFUSE_DAC_CONF_REG (0x118)



EFUSE_DAC_CLK_DIV eFuse timing configuration register. (R/W)

Register 20.29: EFUSE_DEC_STATUS_REG (0x11c)



EFUSE_DEC_WARNINGS If a bit is set in this register, it means some errors were corrected while decoding the 3/4 encoding scheme. (RO)

21. AES Accelerator

21.1 Introduction

The AES Accelerator speeds up AES operations significantly, compared to AES algorithms implemented solely in software. The AES Accelerator supports six algorithms of FIPS PUB 197, specifically AES-128, AES-192 and AES-256 encryption and decryption.

21.2 Features

- Supports AES-128 encryption and decryption
- Supports AES-192 encryption and decryption
- Supports AES-256 encryption and decryption
- Supports four variations of key endianness and four variations of text endianness

21.3 Functional Description

21.3.1 AES Algorithm Operations

The AES Accelerator supports six algorithms of FIPS PUB 197, specifically AES-128, AES-192 and AES-256 encryption and decryption. The AES_MODE_REG register can be configured to different values to enable different algorithm operations, as shown in Table 84.

Table 84: Operation Mode

AES_MODE_REG[2:0]	Operation
0	AES-128 Encryption
1	AES-192 Encryption
2	AES-256 Encryption
4	AES-128 Decryption
5	AES-192 Decryption
6	AES-256 Decryption

21.3.2 Key, Plaintext and Ciphertext

The encryption or decryption key is stored in AES_KEY_*n*_REG, which is a set of eight 32-bit registers. For AES-128 encryption/decryption, the 128-bit key is stored in AES_KEY_0_REG ~ AES_KEY_3_REG. For AES-192 encryption/decryption, the 192-bit key is stored in AES_KEY_0_REG ~ AES_KEY_5_REG. For AES-256 encryption/decryption, the 256-bit key is stored in AES_KEY_0_REG ~ AES_KEY_7_REG.

Plaintext and ciphertext is stored in the AES_TEXT_*m*_REG registers. There are four 32-bit registers. To enable AES-128/192/256 encryption, initialize the AES_TEXT_*m*_REG registers with plaintext before encryption. When encryption is finished, the AES Accelerator will store back the resulting ciphertext in the AES_TEXT_*m*_REG registers. To enable AES-128/192/256 decryption, initialize the AES_TEXT_*m*_REG registers with ciphertext before decryption. When decryption is finished, the AES Accelerator will store back the resulting plaintext in the AES_TEXT_*m*_REG registers.

21.3.3 Endianness

Key Endianness

Bit 0 and bit 1 in AES_ENDIAN_REG define the key endianness. For detailed information, please see Table 86, Table 87 and Table 88. w[0] ~ w[3] in Table 86, w[0] ~ w[5] in Table 87 and w[0] ~ w[7] in Table 88 are "the first Nk words of the expanded key" as specified in "5.2: Key Expansion" of FIPS PUB 197. "Column Bit" specifies the bytes in the word from w[0] to w[7]. The bytes of AES_KEY_*n*_REG comprise "the first Nk words of the expanded key".

Text Endianness

Bit 2 and bit 3 in AES_ENDIAN_REG define the endianness of input text, while Bit 4 and Bit 5 define the endianness of output text. The input text refers to the plaintext in AES-128/192/256 encryption and the ciphertext in decryption. The output text refers to the ciphertext in AES-128/192/256 encryption and the plaintext in decryption. For details, please see Table 85. "State" in Table 85 is defined as that in "3.4: The State" of FIPS PUB 197: "The AES algorithm operations are performed on a two-dimensional array of bytes called the State". The ciphertext or plaintexts stored in each byte of AES_TEXT_*m*_REG comprise the State.

Table 85: AES Text Endianness

AES_ENDIAN_REG[3]/[5]	AES_ENDIAN_REG[2]/[4]	Plaintext/Ciphertext					
0	0	State		c			
				0	1	2	3
		r	0	AES_TEXT_3_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_0_REG[31:24]
			1	AES_TEXT_3_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_0_REG[23:16]
			2	AES_TEXT_3_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_0_REG[15:8]
			3	AES_TEXT_3_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_0_REG[7:0]
0	1	State		c			
				0	1	2	3
		r	0	AES_TEXT_3_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_0_REG[7:0]
			1	AES_TEXT_3_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_0_REG[15:8]
			2	AES_TEXT_3_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_0_REG[23:16]
			3	AES_TEXT_3_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_0_REG[31:24]
1	0	State		c			
				0	1	2	3
		r	0	AES_TEXT_0_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_3_REG[31:24]
			1	AES_TEXT_0_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_3_REG[23:16]
			2	AES_TEXT_0_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_3_REG[15:8]
			3	AES_TEXT_0_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_3_REG[7:0]
1	1	State		c			
				0	1	2	3
		r	0	AES_TEXT_0_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_3_REG[7:0]
			1	AES_TEXT_0_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_3_REG[15:8]
			2	AES_TEXT_0_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_3_REG[23:16]
			3	AES_TEXT_0_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_3_REG[31:24]

Table 86: AES-128 Key Endianness

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

Table 87: AES-192 Key Endianness

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit	w[0]	w[1]	w[2]	w[3]	w[4]
0	0	[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG
		[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG
		[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG
		[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG
0	1	[31:24]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG
		[23:16]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG
		[15:8]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG
		[7:0]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG
1	0	[31:24]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG
		[23:16]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG
		[15:8]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG
		[7:0]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG
1	1	[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG
		[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_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
		[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

Table 88: AES-256 Key Endianness

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

21.3.4 Encryption and Decryption Operations

Single Operation

1. Initialize AES_MODE_REG, AES_KEY_*n*_REG, AES_TEXT_*m*_REG and AES_ENDIAN_REG.
2. Write 1 to AES_START_REG.
3. Wait until AES_IDLE_REG reads 1.
4. Read results from AES_TEXT_*m*_REG.

Consecutive Operations

Every time an operation is completed, only AES_TEXT_*m*_REG is modified by the AES Accelerator. Initialization can, therefore, be simplified in a series of consecutive operations.

1. Update contents of AES_MODE_REG, AES_KEY_*n*_REG and AES_ENDIAN_REG, if required.
2. Load AES_TEXT_*m*_REG.
3. Write 1 to AES_START_REG.
4. Wait until AES_IDLE_REG reads 1.
5. Read results from AES_TEXT_*m*_REG.

21.3.5 Speed

The AES Accelerator requires 11 to 15 clock cycles to encrypt a message block, and 21 or 22 clock cycles to decrypt a message block.

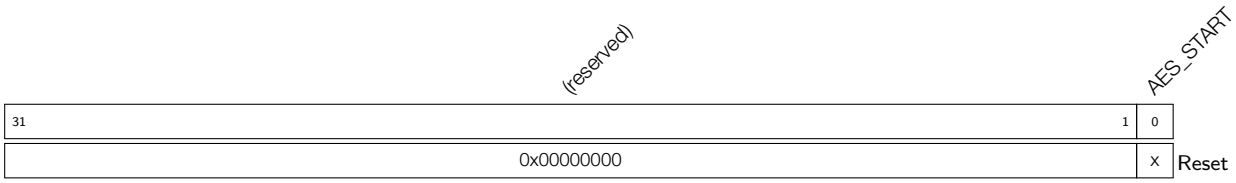
21.4 Register Summary

Name	Description	Address	Access
Configuration registers			
AES_MODE_REG	Mode of operation of the AES Accelerator	0x3FF01008	R/W
AES_ENDIAN_REG	Endianness configuration register	0x3FF01040	R/W
Key registers			
AES_KEY_0_REG	AES key material register 0	0x3FF01010	R/W
AES_KEY_1_REG	AES key material register 1	0x3FF01014	R/W
AES_KEY_2_REG	AES key material register 2	0x3FF01018	R/W
AES_KEY_3_REG	AES key material register 3	0x3FF0101C	R/W
AES_KEY_4_REG	AES key material register 4	0x3FF01020	R/W
AES_KEY_5_REG	AES key material register 5	0x3FF01024	R/W
AES_KEY_6_REG	AES key material register 6	0x3FF01028	R/W
AES_KEY_7_REG	AES key material register 7	0x3FF0102C	R/W
Encrypted/decrypted data registers			
AES_TEXT_0_REG	AES encrypted/decrypted data register 0	0x3FF01030	R/W
AES_TEXT_1_REG	AES encrypted/decrypted data register 1	0x3FF01034	R/W
AES_TEXT_2_REG	AES encrypted/decrypted data register 2	0x3FF01038	R/W
AES_TEXT_3_REG	AES encrypted/decrypted data register 3	0x3FF0103C	R/W
Control/status registers			

Name	Description	Address	Access
AES_START_REG	AES operation start control register	0x3FF01000	WO
AES_IDLE_REG	AES idle status register	0x3FF01004	RO

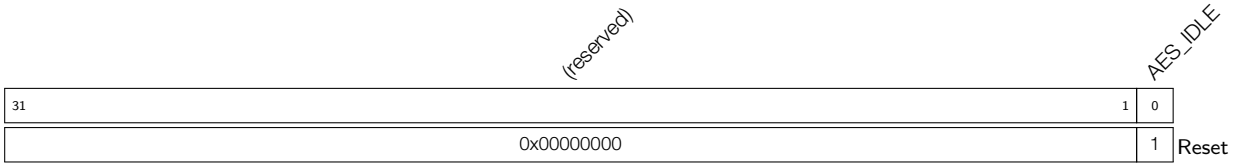
21.5 Registers

Register 21.1: AES_START_REG (0x000)



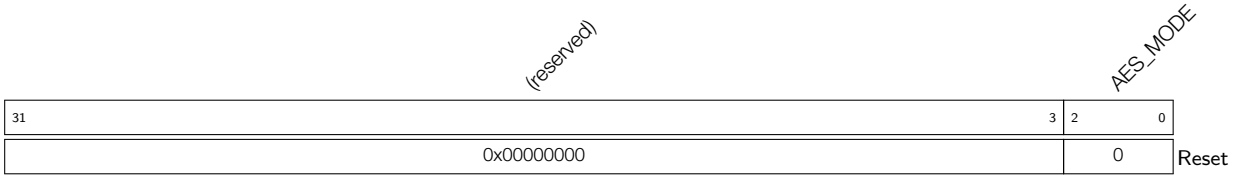
AES_START Write 1 to start the AES operation. (WO)

Register 21.2: AES_IDLE_REG (0x004)



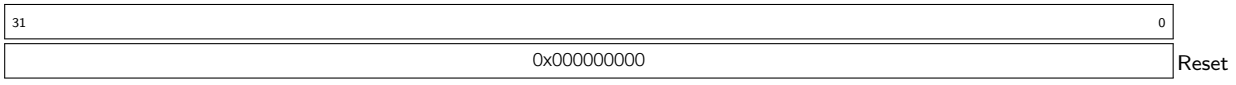
AES_IDLE AES Idle register. Reads 'zero' while the AES Accelerator is busy processing; reads 'one' otherwise. (RO)

Register 21.3: AES_MODE_REG (0x008)



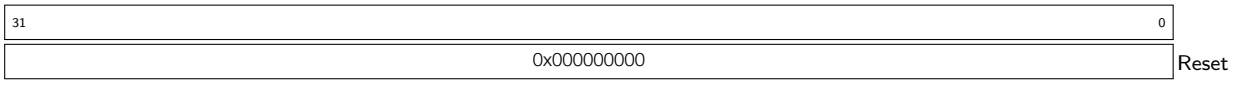
AES_MODE Selects the AES accelerator mode of operation. See Table 84 for details. (R/W)

Register 21.4: AES_KEY_*n*_REG (*n*: 0-7) (0x10+4n*)**



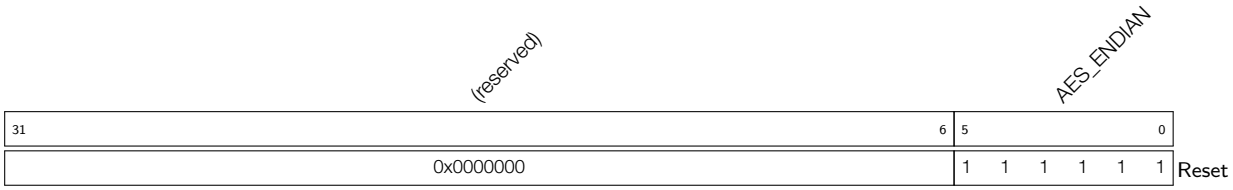
AES_KEY_*n*_REG (*n*: 0-7) AES key material register. (R/W)

Register 21.5: AES_TEXT_*m*_REG (*m*: 0-3) (0x30+4m*)**



AES_TEXT_*m*_REG (*m*: 0-3) Plaintext and ciphertext register. (R/W)

Register 21.6: AES_ENDIAN_REG (0x040)



AES_ENDIAN Endianness selection register. See Table 85 for details. (R/W)

22. SHA Accelerator

22.1 Introduction

The SHA Accelerator is included to speed up SHA hashing operations significantly, compared to SHA hashing algorithms implemented solely in software. The SHA Accelerator supports four algorithms of FIPS PUB 180-4, specifically SHA-1, SHA-256, SHA-384 and SHA-512.

22.2 Features

Hardware support for popular secure hashing algorithms:

- SHA-1
- SHA-256
- SHA-384
- SHA-512

22.3 Functional Description

22.3.1 Padding and Parsing the Message

The SHA Accelerator can only accept one message block at a time. Software divides the message into blocks according to "5.2 Parsing the Message" in FIPS PUB 180-4 and writes one block to the SHA_TEXT_*n*_REG registers each time. For SHA-1 and SHA-256, software writes a 512-bit message block to SHA_TEXT_0_REG ~ SHA_TEXT_15_REG each time. For SHA-384 and SHA-512, software writes a 1024-bit message block to SHA_TEXT_0_REG ~ SHA_TEXT_31_REG each time.

The SHA Accelerator is unable to perform the padding operation of "5.1 Padding the Message" in FIPS PUB 180-4; Note that the user software is expected to pad the message before feeding it into the accelerator.

As described in "2.2.1: Parameters" in FIPS PUB 180-4, "$M_0^{(i)}$ is the leftmost word of message block i". $M_0^{(i)}$ is stored in SHA_TEXT_0_REG. In the same fashion, the SHA_TEXT_1_REG register stores the second left-most word of a message block $M_1^{(N)}$, etc.

22.3.2 Message Digest

When the hashing operation is finished, the message digest will be refreshed by SHA Accelerator and will be stored in SHA_TEXT_*n*_REG. SHA-1 produces a 160-bit message digest and stores it in SHA_TEXT_0_REG ~ SHA_TEXT_4_REG. SHA-256 produces a 256-bit message digest and stores it in SHA_TEXT_0_REG ~ SHA_TEXT_7_REG. SHA-384 produces a 384-bit message digest and stores it in SHA_TEXT_0_REG ~ SHA_TEXT_11_REG. SHA-512 produces a 512-bit message digest and stores it in SHA_TEXT_0_REG ~ SHA_TEXT_15_REG.

As described in "2.2.1 Parameters" in FIPS PUB 180-4, "$H^{(N)}$ is the final hash value, and is used to determine the message digest", while "$H_0^{(i)}$ is the leftmost word of hash value i", so the leftmost word $H_0^{(N)}$ in the message digest is stored in SHA_TEXT_0_REG. In the same fashion, the second leftmost word $H_1^{(N)}$ in the message digest is stored in SHA_TEXT_1_REG, etc.

22.3.3 Hash Operation

There is a set of control registers for SHA-1, SHA-256, SHA-384 and SHA-512, respectively; different hashing algorithms use different control registers.
SHA-1 uses SHA_SHA1_START_REG, SHA_SHA1_CONTINUE_REG, SHA_SHA1_LOAD_REG and SHA_SHA1_BUSY_REG.
SHA-256 uses SHA_SHA256_START_REG, SHA_SHA256_CONTINUE_REG, SHA_SHA256_LOAD_REG and SHA_SHA256_BUSY_REG. SHA-384 uses SHA_SHA384_START_REG, SHA_SHA384_CONTINUE_REG, SHA_SHA384_LOAD_REG and SHA_SHA384_BUSY_REG.
SHA-512 uses SHA_SHA512_START_REG, SHA_SHA512_CONTINUE_REG, SHA_SHA512_LOAD_REG and SHA_SHA512_BUSY_REG. The following steps describe the operation in a detailed manner.

1. Feed the accelerator with the first message block:
   (a) Use the first message block to initialize SHA_TEXT_*n*_REG.
   (b) Write 1 to SHA_*X*_START_REG.
   (c) Wait for SHA_*X*_BUSY_REG to read 0, indicating that the operation is completed.
2. Similarly, feed the accelerator with subsequent message blocks:
   (a) Initialize SHA_TEXT_*n*_REG using the subsequent message block.
   (b) Write 1 to SHA_*X*_CONTINUE_REG.
   (c) Wait for SHA_*X*_BUSY_REG to read 0, indicating that the operation is completed.
3. Get message digest:
   (a) Write 1 to SHA_*X*_LOAD_REG.
   (b) Wait for SHA_*X*_BUSY_REG to read 0, indicating that operation is completed.
   (c) Read message digest from SHA_TEXT_*n*_REG.

22.3.4 Speed

The SHA Accelerator requires 60 to 100 clock cycles to process a message block and 8 to 20 clock cycles to calculate the final digest.

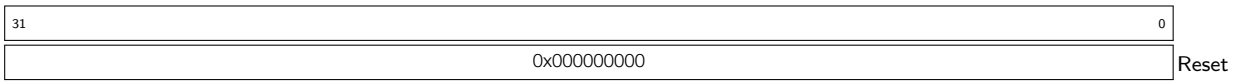
22.4 Register Summary

Name	Description	Address	Access
Encrypted/decrypted data registers			
SHA_TEXT_0_REG	SHA encrypted/decrypted data register 0	0x3FF03000	R/W
SHA_TEXT_1_REG	SHA encrypted/decrypted data register 1	0x3FF03004	R/W
SHA_TEXT_2_REG	SHA encrypted/decrypted data register 2	0x3FF03008	R/W
SHA_TEXT_3_REG	SHA encrypted/decrypted data register 3	0x3FF0300C	R/W
SHA_TEXT_4_REG	SHA encrypted/decrypted data register 4	0x3FF03010	R/W
SHA_TEXT_5_REG	SHA encrypted/decrypted data register 5	0x3FF03014	R/W
SHA_TEXT_6_REG	SHA encrypted/decrypted data register 6	0x3FF03018	R/W
SHA_TEXT_7_REG	SHA encrypted/decrypted data register 7	0x3FF0301C	R/W

Name	Description	Address	Access
SHA_TEXT_8_REG	SHA encrypted/decrypted data register 8	0x3FF03020	R/W
SHA_TEXT_9_REG	SHA encrypted/decrypted data register 9	0x3FF03024	R/W
SHA_TEXT_10_REG	SHA encrypted/decrypted data register 10	0x3FF03028	R/W
SHA_TEXT_11_REG	SHA encrypted/decrypted data register 11	0x3FF0302C	R/W
SHA_TEXT_12_REG	SHA encrypted/decrypted data register 12	0x3FF03030	R/W
SHA_TEXT_13_REG	SHA encrypted/decrypted data register 13	0x3FF03034	R/W
SHA_TEXT_14_REG	SHA encrypted/decrypted data register 14	0x3FF03038	R/W
SHA_TEXT_15_REG	SHA encrypted/decrypted data register 15	0x3FF0303C	R/W
SHA_TEXT_16_REG	SHA encrypted/decrypted data register 16	0x3FF03040	R/W
SHA_TEXT_17_REG	SHA encrypted/decrypted data register 17	0x3FF03044	R/W
SHA_TEXT_18_REG	SHA encrypted/decrypted data register 18	0x3FF03048	R/W
SHA_TEXT_19_REG	SHA encrypted/decrypted data register 19	0x3FF0304C	R/W
SHA_TEXT_20_REG	SHA encrypted/decrypted data register 20	0x3FF03050	R/W
SHA_TEXT_21_REG	SHA encrypted/decrypted data register 21	0x3FF03054	R/W
SHA_TEXT_22_REG	SHA encrypted/decrypted data register 22	0x3FF03058	R/W
SHA_TEXT_23_REG	SHA encrypted/decrypted data register 23	0x3FF0305C	R/W
SHA_TEXT_24_REG	SHA encrypted/decrypted data register 24	0x3FF03060	R/W
SHA_TEXT_25_REG	SHA encrypted/decrypted data register 25	0x3FF03064	R/W
SHA_TEXT_26_REG	SHA encrypted/decrypted data register 26	0x3FF03068	R/W
SHA_TEXT_27_REG	SHA encrypted/decrypted data register 27	0x3FF0306C	R/W
SHA_TEXT_28_REG	SHA encrypted/decrypted data register 28	0x3FF03070	R/W
SHA_TEXT_29_REG	SHA encrypted/decrypted data register 29	0x3FF03074	R/W
SHA_TEXT_30_REG	SHA encrypted/decrypted data register 30	0x3FF03078	R/W
SHA_TEXT_31_REG	SHA encrypted/decrypted data register 31	0x3FF0307C	R/W
Control/status registers			
SHA_SHA1_START_REG	Control register to initiate SHA1 operation	0x3FF03080	WO
SHA_SHA1_CONTINUE_REG	Control register to continue SHA1 operation	0x3FF03084	WO
SHA_SHA1_LOAD_REG	Control register to calculate the final SHA1 hash	0x3FF03088	WO
SHA_SHA1_BUSY_REG	Status register for SHA1 operation	0x3FF0308C	RO
SHA_SHA256_START_REG	Control register to initiate SHA256 operation	0x3FF03090	WO
SHA_SHA256_CONTINUE_REG	Control register to continue SHA256 operation	0x3FF03094	WO
SHA_SHA256_LOAD_REG	Control register to calculate the final SHA256 hash	0x3FF03098	WO
SHA_SHA256_BUSY_REG	Status register for SHA256 operation	0x3FF0309C	RO
SHA_SHA384_START_REG	Control register to initiate SHA384 operation	0x3FF030A0	WO
SHA_SHA384_CONTINUE_REG	Control register to continue SHA384 operation	0x3FF030A4	WO
SHA_SHA384_LOAD_REG	Control register to calculate the final SHA384 hash	0x3FF030A8	WO
SHA_SHA384_BUSY_REG	Status register for SHA384 operation	0x3FF030AC	RO
SHA_SHA512_START_REG	Control register to initiate SHA512 operation	0x3FF030B0	WO
SHA_SHA512_CONTINUE_REG	Control register to continue SHA512 operation	0x3FF030B4	WO
SHA_SHA512_LOAD_REG	Control register to calculate the final SHA512 hash	0x3FF030B8	WO
SHA_SHA512_BUSY_REG	Status register for SHA512 operation	0x3FF030BC	RO

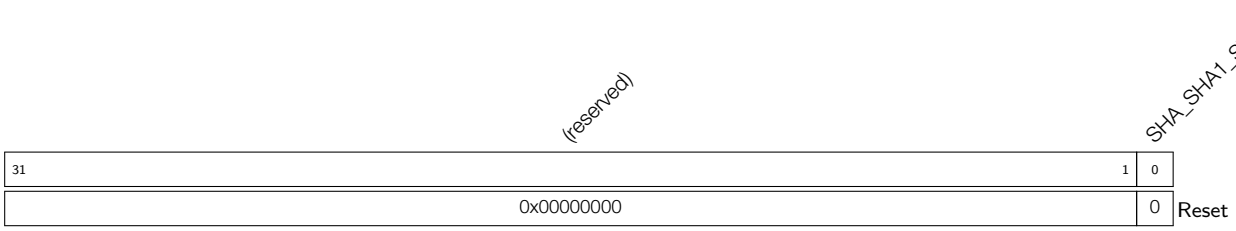
22.5 Registers

Register 22.1: SHA_TEXT_*n*_REG (*n*: 0-31) (0x0+4*n)



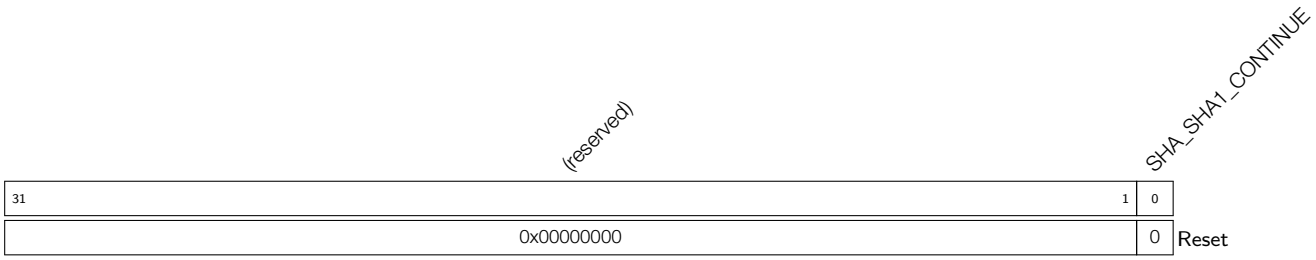
SHA_TEXT_*n*_REG (*n*: 0-31) SHA Message block and hash result register. (R/W)

Register 22.2: SHA_SHA1_START_REG (0x080)



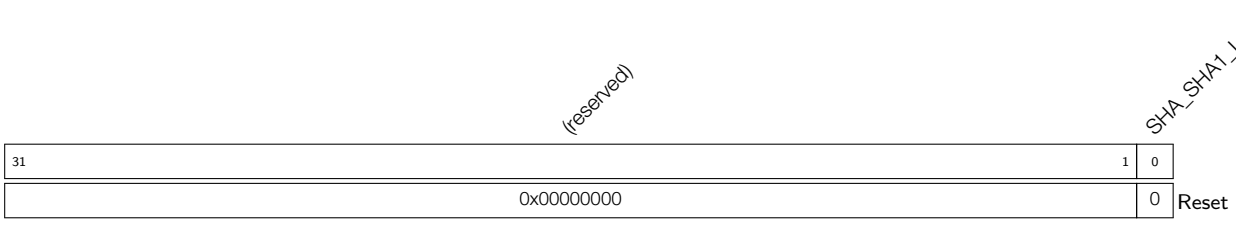
SHA_SHA1_START Write 1 to start an SHA-1 operation on the first message block. (WO)

Register 22.3: SHA_SHA1_CONTINUE_REG (0x084)



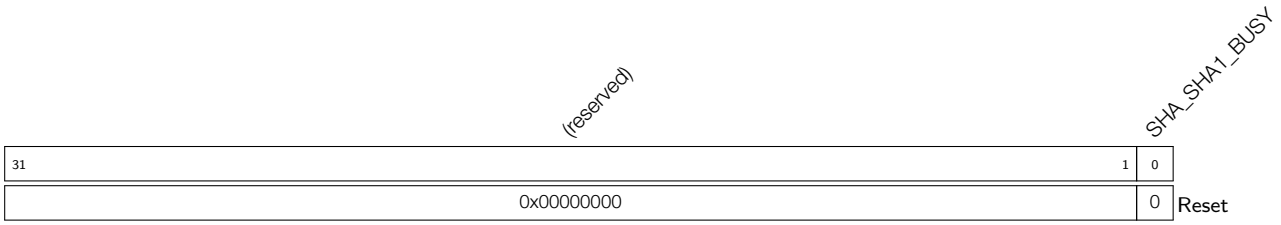
SHA_SHA1_CONTINUE Write 1 to continue the SHA-1 operation with subsequent blocks. (WO)

Register 22.4: SHA_SHA1_LOAD_REG (0x088)



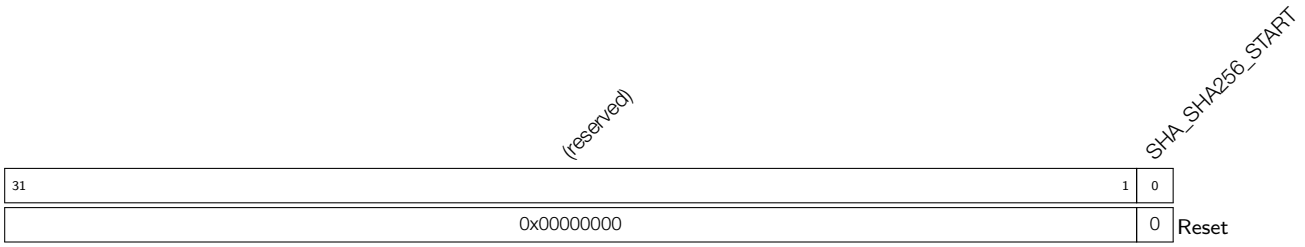
SHA_SHA1_LOAD Write 1 to finish the SHA-1 operation to calculate the final message hash. (WO)

Register 22.5: SHA_SHA1_BUSY_REG (0x08C)



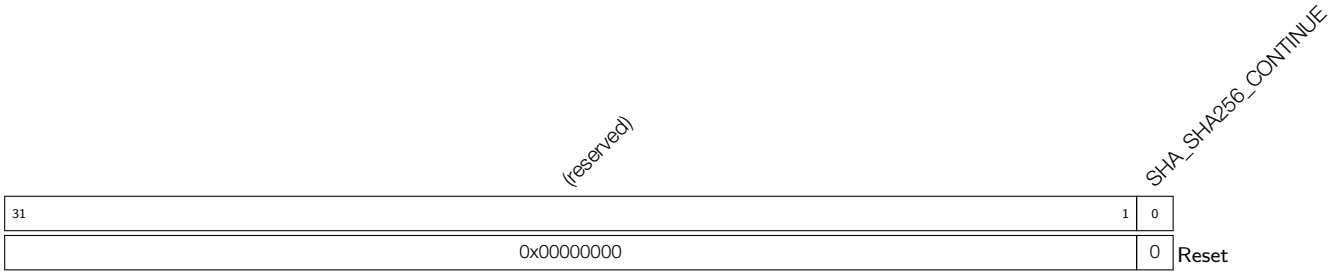
SHA_SHA1_BUSY SHA-1 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

Register 22.6: SHA_SHA256_START_REG (0x090)



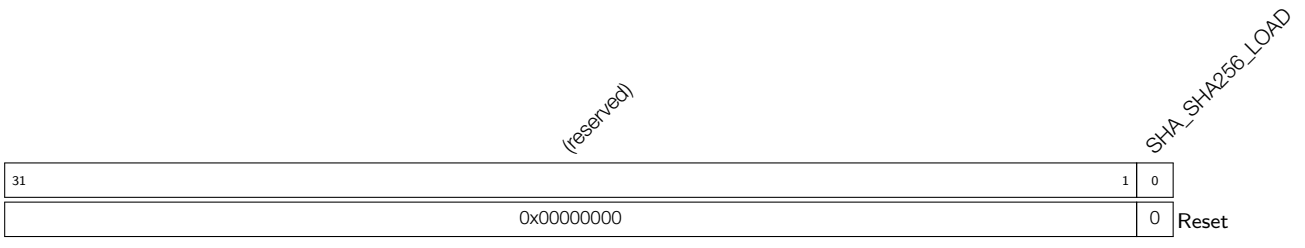
SHA_SHA256_START Write 1 to start an SHA-256 operation on the first message block. (WO)

Register 22.7: SHA_SHA256_CONTINUE_REG (0x094)



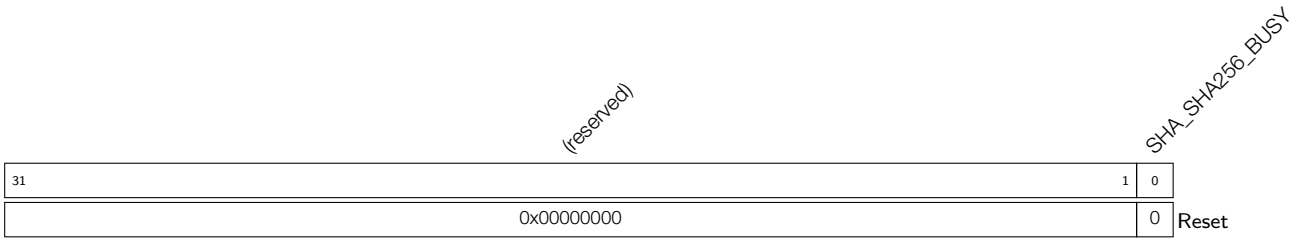
SHA_SHA256_CONTINUE Write 1 to continue the SHA-256 operation with subsequent blocks. (WO)

Register 22.8: SHA_SHA256_LOAD_REG (0x098)



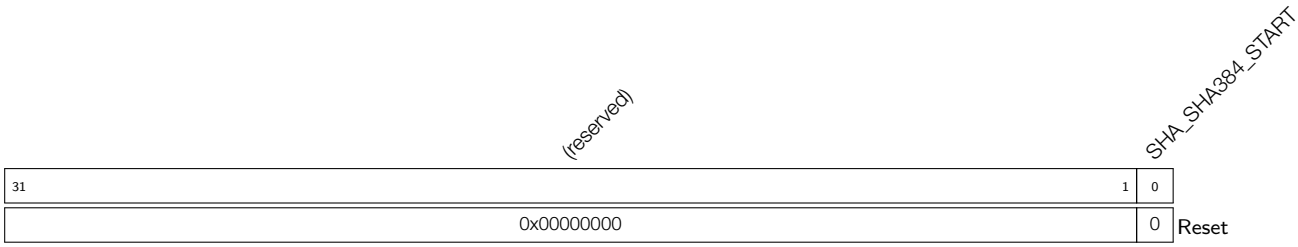
SHA_SHA256_LOAD Write 1 to finish the SHA-256 operation to calculate the final message hash. (WO)

Register 22.9: SHA_SHA256_BUSY_REG (0x09C)



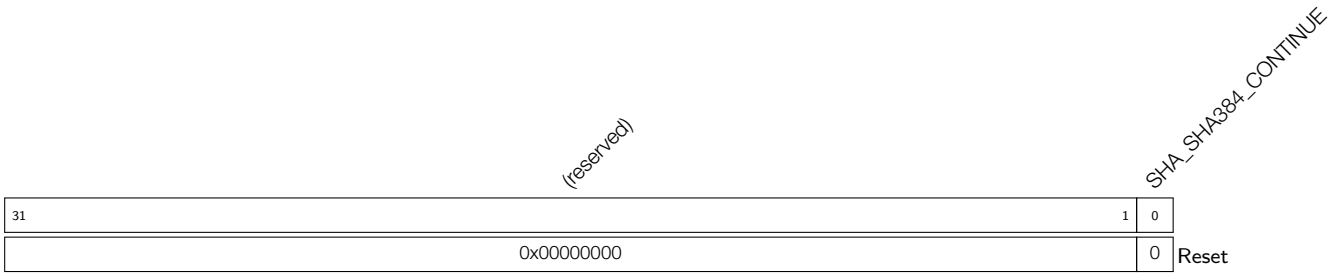
SHA_SHA256_BUSY SHA-256 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

Register 22.10: SHA_SHA384_START_REG (0x0A0)



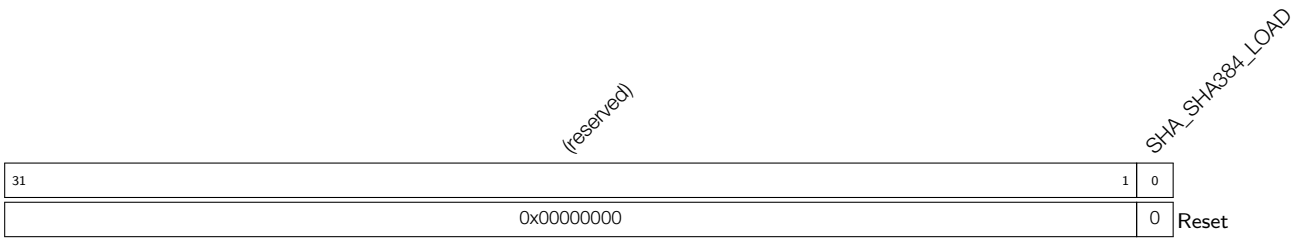
SHA_SHA384_START Write 1 to start an SHA-384 operation on the first message block. (WO)

Register 22.11: SHA_SHA384_CONTINUE_REG (0x0A4)



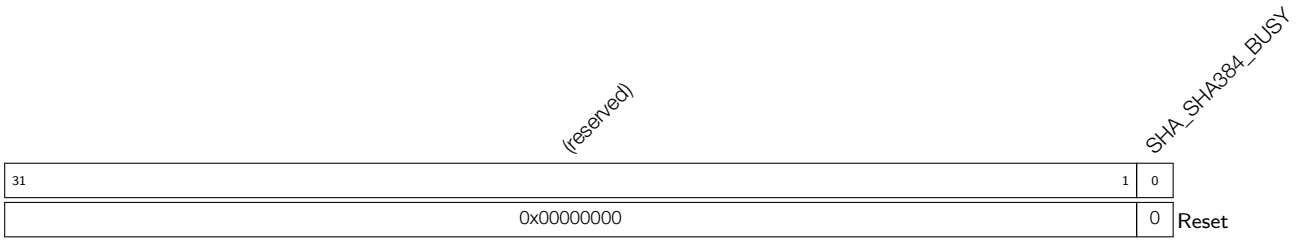
SHA_SHA384_CONTINUE Write 1 to continue the SHA-384 operation with subsequent blocks. (WO)

Register 22.12: SHA_SHA384_LOAD_REG (0x0A8)



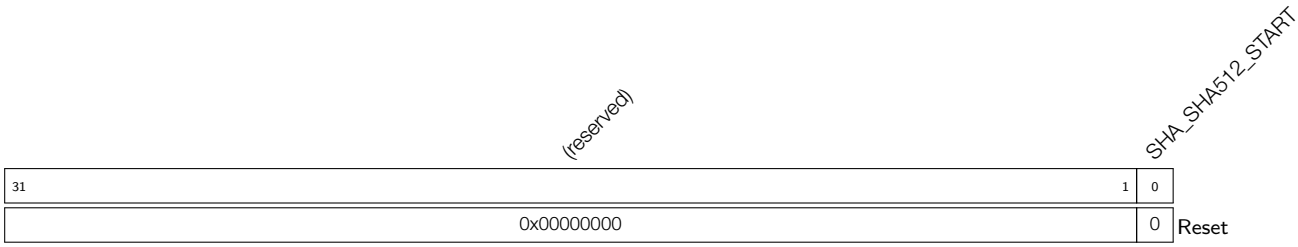
SHA_SHA384_LOAD Write 1 to finish the SHA-384 operation to calculate the final message hash. (WO)

Register 22.13: SHA_SHA384_BUSY_REG (0x0AC)



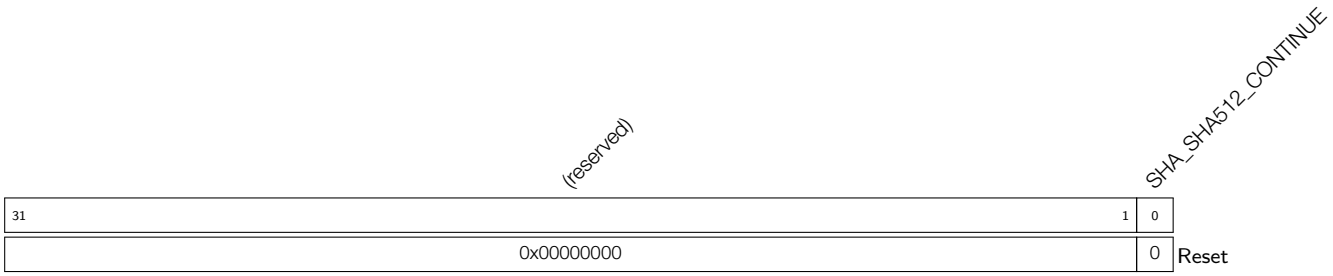
SHA_SHA384_BUSY SHA-384 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

Register 22.14: SHA_SHA512_START_REG (0x0B0)



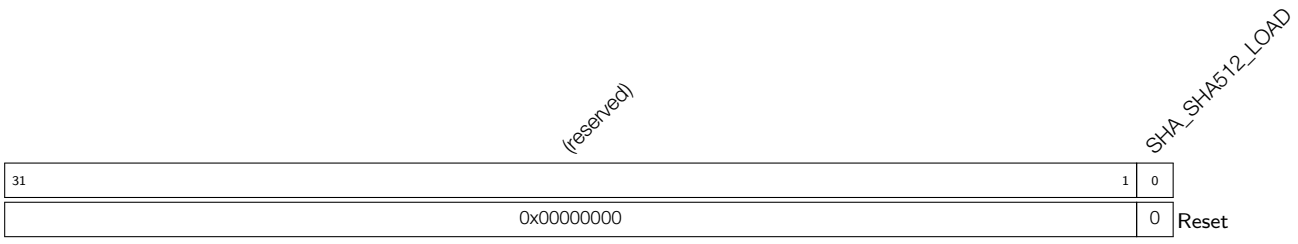
SHA_SHA512_START Write 1 to start an SHA-512 operation on the first message block. (WO)

Register 22.15: SHA_SHA512_CONTINUE_REG (0x0B4)



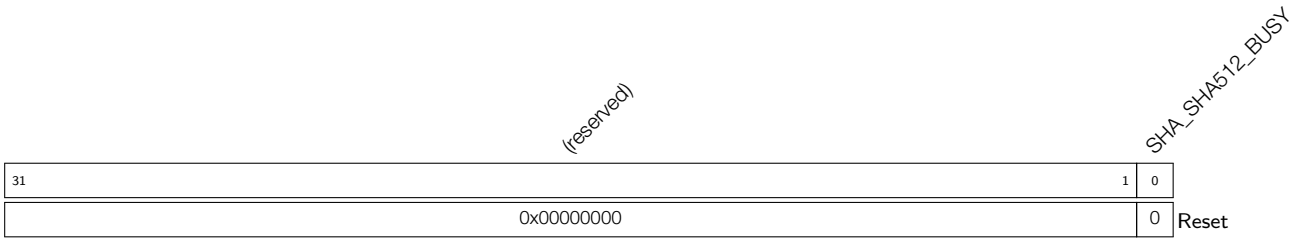
SHA_SHA512_CONTINUE Write 1 to continue the SHA-512 operation with subsequent blocks. (WO)

Register 22.16: SHA_SHA512_LOAD_REG (0x0B8)



SHA_SHA512_LOAD Write 1 to finish the SHA-512 operation to calculate the final message hash. (WO)

Register 22.17: SHA_SHA512_BUSY_REG (0x0BC)



SHA_SHA512_BUSY SHA-512 operation status: 1 if the SHA accelerator is processing data, 0 if it is idle. (RO)

23. RSA Accelerator

23.1 Introduction

The RSA Accelerator provides hardware support for multiple precision arithmetic operations used in RSA asymmetric cipher algorithms.

Sometimes, multiple precision arithmetic is also called ”bignum arithmetic”, ”bigint arithmetic” or ”arbitrary precision arithmetic”.

23.2 Features

- Support for large-number modular exponentiation
- Support for large-number modular multiplication
- Support for large-number multiplication
- Support for various lengths of operands

23.3 Functional Description

23.3.1 Initialization

The RSA Accelerator is activated by enabling the corresponding peripheral clock, and by clearing the DPORT_RSA_PD bit in the DPORT_RSA_PD_CTRL_REG register. This releases the RSA Accelerator from reset.

When the RSA Accelerator is released from reset, the register RSA_CLEAN_REG reads 0 and an initialization process begins. Hardware initializes the four memory blocks by setting them to 0. After initialization is complete, RSA_CLEAN_REG reads 1. For this reason, software should query RSA_CLEAN_REG after being released from reset, and before writing to any RSA Accelerator memory blocks or registers for the first time.

23.3.2 Large Number Modular Exponentiation

Large-number modular exponentiation performs $Z = X^Y \bmod M$. The operation is based on Montgomery multiplication. Aside from the arguments X, Y, and M, two additional ones are needed — $\overline{r}$ and M'. These arguments are calculated in advance by software.

The RSA Accelerator supports operand lengths of $N \in \{512, 1024, 1536, 2048, 2560, 3072, 3584, 4096\}$ bits. The bit length of arguments Z, X, Y, M, and $\overline{r}$ can be any one from the N set, but all numbers in a calculation must be of the same length. The bit length of M' is always 32.

To represent the numbers used as operands, define a base-b positional notation, as follows:

$$b = 2^{32}$$

In this notation, each number is represented by a sequence of base-b digits, where each base-b digit is a 32-bit word. Representing an N-bit number requires n base-b digits (all of the possible N lengths are multiples of 32).

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1}Z_{n-2}\cdots Z_0)_b$$
$$X = (X_{n-1}X_{n-2}\cdots X_0)_b$$
$$Y = (Y_{n-1}Y_{n-2}\cdots Y_0)_b$$
$$M = (M_{n-1}M_{n-2}\cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1}\overline{r}_{n-2}\cdots \overline{r}_0)_b$$

Each of the n values in $Z_{n-1} \sim Z_0$, $X_{n-1} \sim X_0$, $Y_{n-1} \sim Y_0$, $M_{n-1} \sim M_0$, $\overline{r}_{n-1} \sim \overline{r}_0$ 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

$$R = b^n$$

then, we can calculate the additional arguments, as follows:

$$\overline{r} = R^2 \bmod M \tag{1}$$
$$\begin{cases} M'' \times M + 1 = R \times R^{-1} \\ M' = M'' \bmod b \end{cases} \tag{2}$$

(Equation 2 is written in a form suitable for calculations using the extended binary GCD algorithm.)

Software can implement large-number modular exponentiations in the following order:

1. Write $(\frac{N}{512} - 1)$ to RSA_MODEXP_MODE_REG.
2. Write X_i, Y_i, M_i and $\overline{r}_i$ ($i \in [0, n) \cap \mathbb{N}$) to memory blocks RSA_X_MEM, RSA_Y_MEM, RSA_M_MEM and RSA_Z_MEM. The capacity of each memory block is 128 words. Each word of each memory block can store one base-b digit. The memory blocks use the little endian format for storage, i.e. the least significant digit of each number is in the lowest address.

   Users need to write data to each memory block only according to the length of the number; data beyond this length are ignored.
3. Write M' to RSA_M_PRIME_REG.
4. Write 1 to RSA_MODEXP_START_REG.
5. Wait for the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.
6. Read the result Z_i ($i \in [0, n) \cap \mathbb{N}$) from RSA_Z_MEM.
7. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.

After the operation, the RSA_MODEXP_MODE_REG register, memory blocks RSA_Y_MEM and RSA_M_MEM, as well as the RSA_M_PRIME_REG will not have changed. However, X_i in RSA_X_MEM and $\overline{r}_i$ in RSA_Z_MEM

will have been overwritten. In order to perform another operation, refresh the registers and memory blocks, as required.

23.3.3 Large Number Modular Multiplication

Large-number modular multiplication performs $Z = X \times Y \bmod M$. This operation is based on Montgomery multiplication. The same values $\overline{r}$ and M' are derived by software using the formulas 1 and 2 shown above.

The RSA Accelerator supports large-number modular multiplication with eight different operand lengths, which are the same as in the large-number modular exponentiation. The operation is performed by a combination of software and hardware. The software performs two hardware operations in sequence.

The software process is as follows:

1. Write $(\frac{N}{512} - 1)$ to RSA_MULT_MODE_REG.
2. Write X_i, M_i and $\overline{r}_i$ ($i \in [0, n) \cap \mathbb{N}$) to registers RSA_X_MEM, RSA_M_MEM and RSA_Z_MEM. Write data to each memory block only according to the length of the number. Data beyond this length are ignored.
3. Write M' to RSA_M_PRIME_REG.
4. Write 1 to RSA_MULT_START_REG.
5. Wait for the first round of the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.
6. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.
7. Write Y_i ($i \in [0, n) \cap \mathbb{N}$) to RSA_X_MEM.

   Users need to write to the memory block only according to the length of the number. Data beyond this length are ignored.
8. Write 1 to RSA_MULT_START_REG.
9. Wait for the second round of the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.
10. Read the result Z_i ($i \in [0, n) \cap \mathbb{N}$) from RSA_Z_MEM.
11. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.

After the operation, the RSA_MULT_MODE_REG register, and memory blocks RSA_M_MEM and RSA_M_PRIME_REG remain unchanged. Users do not need to refresh these registers or memory blocks if the values remain the same.

23.3.4 Large Number Multiplication

Large-number multiplication performs $Z = X \times Y$. The length of Z is twice that of X and Y. Therefore, the RSA Accelerator supports large-number multiplication with only four operand lengths of $N \in \{512, 1024, 1536, 2048\}$ bits. The length $\hat{N}$ of the result Z is $2 \times N$ bits.

Operands X and Y need to be extended to form arguments $\hat{X}$ and $\hat{Y}$ which have the same length ($\hat{N}$ bits) as

the result Z. X is left-extended and Y is right-extended, and defined as follows:

$$n = \frac{N}{32}$$

$$\hat{N} = 2 \times N$$

$$\hat{n} = \frac{\hat{N}}{32} = 2n$$

$$\hat{X} = (\hat{X}_{\hat{n}-1}\hat{X}_{\hat{n}-2}\cdots\hat{X}_0)_b = (\underbrace{00\cdots0}_{n}X)_b = (\underbrace{00\cdots0}_{n}X_{n-1}X_{n-2}\cdots X_0)_b$$

$$\hat{Y} = (\hat{Y}_{\hat{n}-1}\hat{Y}_{\hat{n}-2}\cdots\hat{Y}_0)_b = (Y\underbrace{00\cdots0}_{n})_b = (Y_{n-1}Y_{n-2}\cdots Y_0\underbrace{00\cdots0}_{n})_b$$

Software performs the operation in the following order:

1. Write $(\frac{\hat{N}}{512} - 1 + 8)$ to RSA_MULT_MODE_REG.
2. Write $\hat{X}_i$ and $\hat{Y}_i$ $(i \in [0, \hat{n}) \cap \mathbb{N})$ to RSA_X_MEM and RSA_Z_MEM, respectively.

   Write the valid data into each number's memory block, according to their lengths. Values beyond this length are ignored. Half of the base-b positional notations written to the memory are zero (using the derivations shown above). These zero values are indispensable.
3. Write 1 to RSA_MULT_START_REG.
4. Wait for the operation to be completed. Poll RSA_INTERRUPT_REG until it reads 1, or until the RSA_INTR interrupt is generated.
5. Read the result Z_i $(i \in [0, \hat{n}) \cap \mathbb{N})$ from RSA_Z_MEM.
6. Write 1 to RSA_INTERRUPT_REG to clear the interrupt.

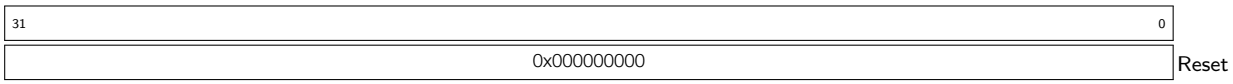
After the operation, only the RSA_MULT_MODE_REG register remains unmodified.

23.4 Register Summary

Name	Description	Address	Access
Configuration registers			
RSA_M_PRIME_REG	Register to store M'	0x3FF02800	R/W
Modular exponentiation registers			
RSA_MODEXP_MODE_REG	Modular exponentiation mode	0x3FF02804	R/W
RSA_MODEXP_START_REG	Start bit	0x3FF02808	WO
Modular multiplication registers			
RSA_MULT_MODE_REG	Modular multiplication mode	0x3FF0280C	R/W
RSA_MULT_START_REG	Start bit	0x3FF02810	WO
Misc registers			
RSA_INTERRUPT_REG	RSA interrupt register	0x3FF02814	R/W
RSA_CLEAN_REG	RSA clean register	0x3FF02818	RO

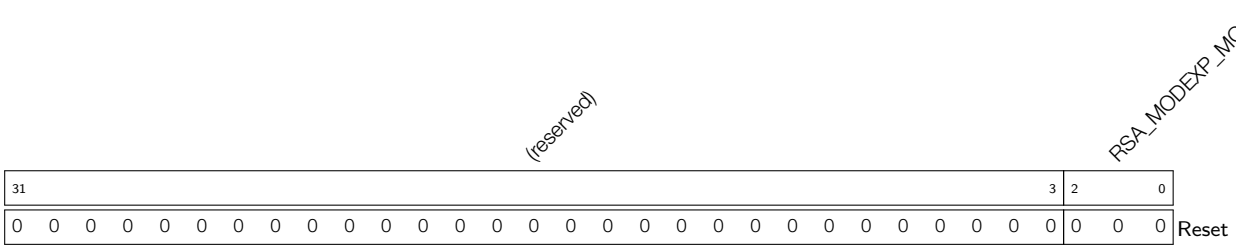
23.5 Registers

Register 23.1: RSA_M_PRIME_REG (0x800)



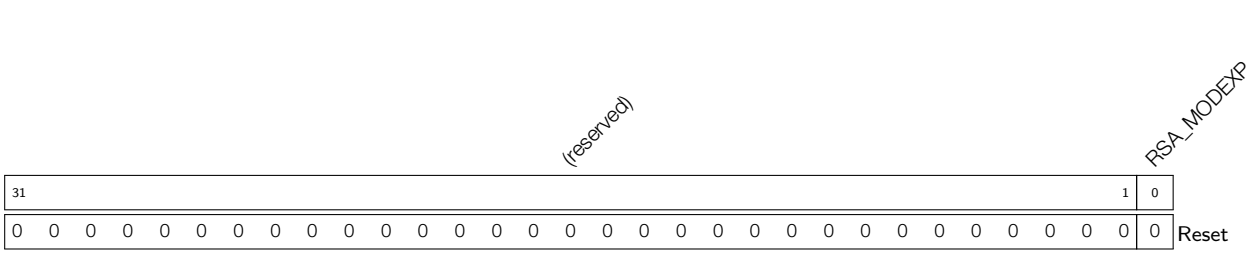
RSA_M_PRIME_REG This register contains M'. (R/W)

Register 23.2: RSA_MODEXP_MODE_REG (0x804)



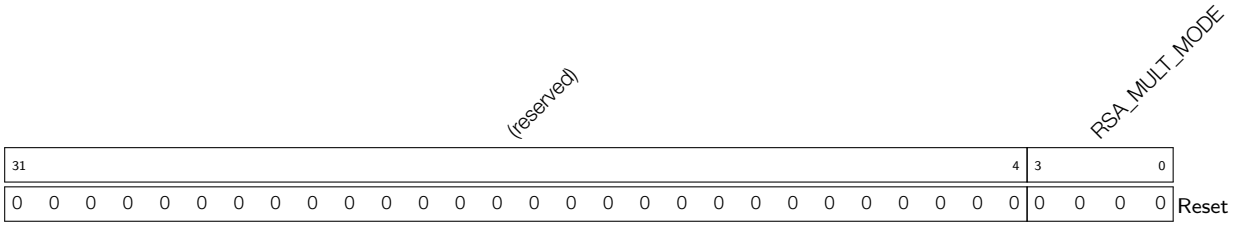
RSA_MODEXP_MODE This register contains the mode of modular exponentiation. (R/W)

Register 23.3: RSA_MODEXP_START_REG (0x808)



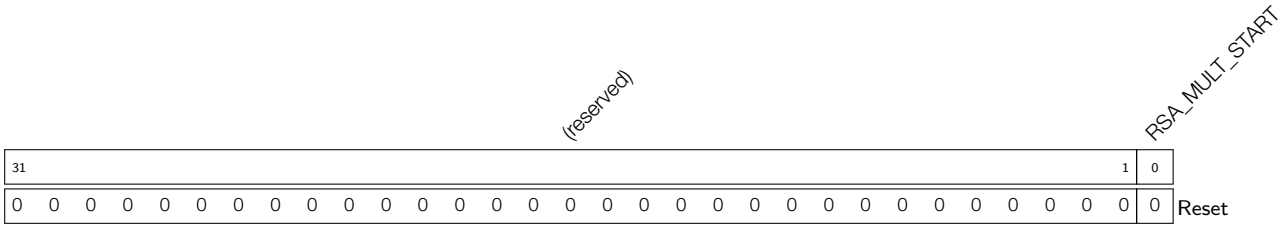
RSA_MODEXP_START Write 1 to start modular exponentiation. (WO)

Register 23.4: RSA_MULT_MODE_REG (0x80C)



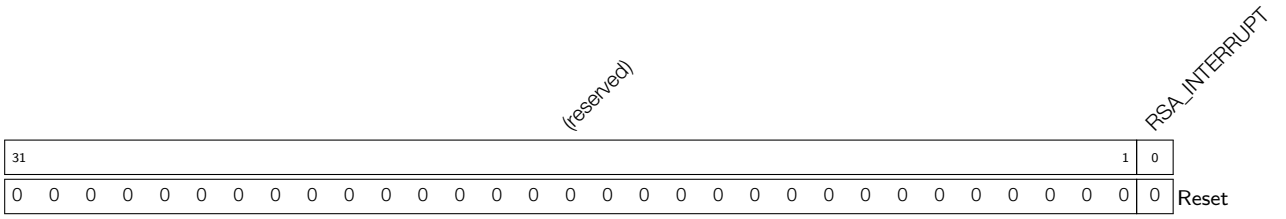
RSA_MULT_MODE This register contains the mode of modular multiplication and multiplication. (R/W)

Register 23.5: RSA_MULT_START_REG (0x810)



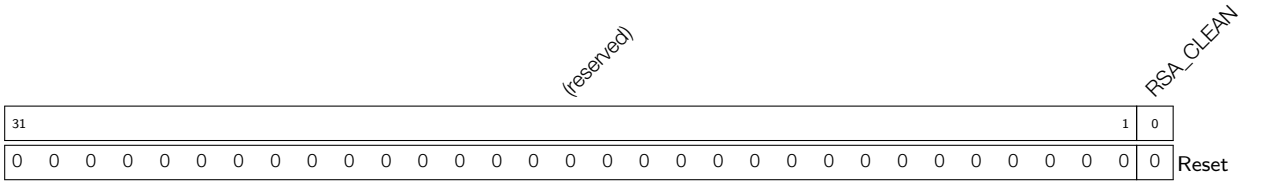
RSA_MULT_START Write 1 to start modular multiplication or multiplication. (WO)

Register 23.6: RSA_INTERRUPT_REG (0x814)



RSA_INTERRUPT RSA interrupt status register. Will read 1 once an operation has completed. (R/W)

Register 23.7: RSA_CLEAN_REG (0x818)



RSA_CLEAN This bit will read 1 once the memory initialization is completed. (RO)

24. Random Number Generator

24.1 Introduction

The ESP32 contains a true random number generator, whose values can be used as a basis for cryptographical operations, among other things.

24.2 Feature

It can generate true random numbers.

24.3 Functional Description

When used correctly, every 32-bit value 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 noise in the Wi-Fi/BT RF system. When Wi-Fi and BT are disabled, the random number generator will give out pseudo-random numbers.

When Wi-Fi or BT is enabled, the random number generator is fed two bits of entropy every APB clock cycle (normally 80 MHz). Thus, for the maximum amount of entropy, it is advisable to read the random register at a maximum rate of 5 MHz.

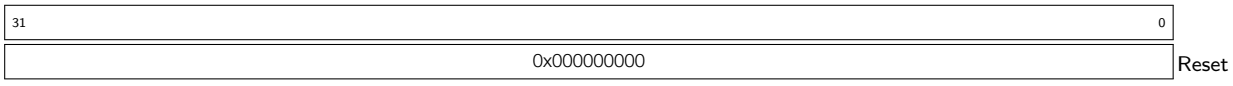
A data sample of 2 GB, read from the random number generator with Wi-Fi enabled and the random register read at 5 MHz, has been tested using the Dieharder Random Number Testsuite (version 3.31.1). The sample passed all tests.

24.4 Register Summary

Name	Description	Address	Access
RNG_DATA_REG	Random number data	0x3FF75144	RO

24.5 Register

Register 24.1: RNG_DATA_REG (0x144)



RNG_DATA_REG Random number source. (RO)

25. Flash Encryption/Decryption

25.1 Overview

Many variants of the ESP32 must store programs and data in external flash memory. The external flash memory chip is likely to contain proprietary firmware and sensitive user data, such as credentials for gaining access to a private network. The Flash Encryption block can encrypt code and write encrypted code to off-chip flash memory for enhanced hardware security. When the CPU reads off-chip flash through the cache, the Flash Decryption block can automatically decrypt instructions and data read from the off-chip flash, thus providing hardware-based security for application code.

25.2 Features

- Various key generation methods
- Software-based encryption
- High-speed, hardware decryption
- Register configuration, system parameters and boot mode jointly determine the flash encryption/decryption function.

25.3 Functional Description

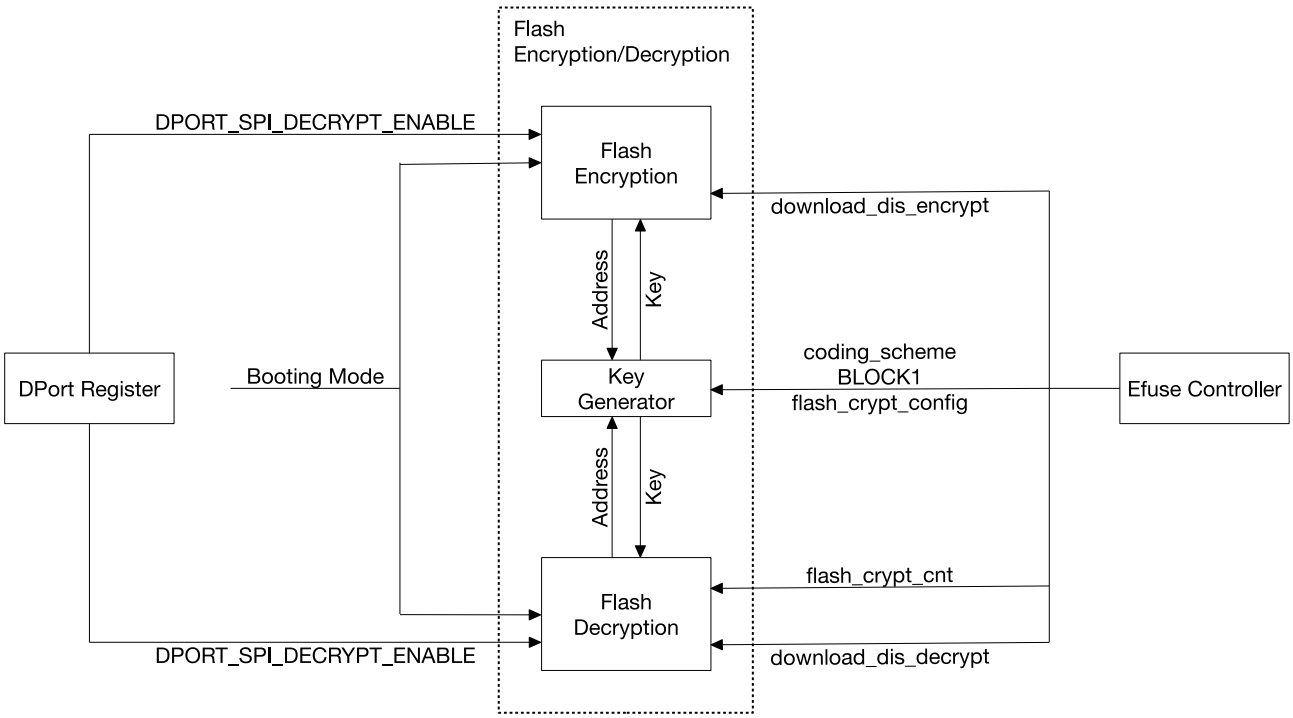


Figure 123: Flash Encryption/Decryption Module Architecture

The Flash Encryption/Decryption module consists of three parts, namely the Key Generator, Flash Encryption block and Flash Decryption block. The structure of these parts is shown in Figure 123. The Key Generator is

shared by both the Flash Encryption block and the Flash Decryption block, which can function simultaneously.

In the peripheral DPort Register, the register relevant to Flash Encryption/Decryption is DPORT_SPI_ENCRYPT_ENABLE bit and DPORT_SPI_DECRYPT_ENABLE bit in DPORT_SLAVE_SPI_CONFIG_REG. The Flash Encryption/Decryption module will fetch six system parameters from the peripheral eFuse Controller. These parameters are: coding_scheme, BLOCK1, flash_crypt_config, download_dis_encrypt, flash_crypt_cnt, and download_dis_decrypt.

25.3.1 Key Generator

According to system parameters coding_scheme and BLOCK1, the Key Generator will first generate $Key_o = f(coding\_scheme, BLOCK1)$.

Then, according to system parameter flash_crypt_config, and off-chip flash physical addresses $Addr_e$ and $Addr_d$ accessed by the Flash Encryption block and the Flash Decryption block, the Key Generator will respectively figure out that:
$Key_e = g(Key_o, flash\_crypt\_config, Addr_e)$,
$Key_d = g(Key_o, flash\_crypt\_config, Addr_d)$.

When all values of system parameter flash_crypt_config are 0, Key_e and Key_d are not relevant to the physical address of the off-chip flash. When all values of system parameter flash_crypt_config are not 0, every 8-word block on the off-chip flash has a dedicated Key_e and Key_d.

25.3.2 Flash Encryption Block

The Flash Encryption block is equipped with registers that can be accessed by the CPU directly. Registers embedded in the Flash Encryption block, registers in the peripheral DPort Register, system parameters and Boot Mode jointly configure and control this block.

The Flash Encryption block requires software intervention during operation. The steps are as follows:

1. Set the DPORT_SPI_ENCRYPT_ENABLE bit of register DPORT_SLAVE_SPI_CONFIG_REG.
2. Write the physical address prepared for the off-chip flash on register FLASH_ENCRYPT_ADDRESS_REG. The address must be 8-word boundary aligned.
3. The Flash Encryption block must encrypt 8-word long code segments. Write the lowest word to register FLASH_ENCRYPT_BUFFER_0_REG, the second-lowest word into FLASH_ENCRYPT_BUFFER_1_REG, and so on, up to FLASH_ENCRYPT_BUFFER_7_REG.
4. Set the FLASH_START bit in FLASH_ENCRYPT_START_REG.
5. Wait for the FLASH_DONE bit to be set in FLASH_ENCRYPT_DONE_REG.
6. Use this function and write any 8-word code to the 8-word aligned address on the off-chip flash via the peripheral SPI0.

In Steps 1 to 5, the Flash Encryption block encrypts 8-word long codes. The key encryption algorithm uses Key_e. The encryption result will also be 8-word long. In Step 6, the peripheral SPI0 writes encrypted results of the Flash Encryption block to the off-chip flash. One parameter of the function used in Step 6 will be the physical address of the off-chip flash. The physical address must be 8-word boundary aligned. Also, the value must be the same as the value written into register FLASH_ENCRYPT_ADDRESS_REG during Step 2. Even though the function used in Step 6 still has a parameter with an 8-word long code, the parameter will be meaningless if

Steps 1 to 5 are executed. The Peripheral SPI0 will use the encrypted result instead. If the Flash Encryption block is not operating, or has not executed Steps 1 to 5, Step 6 will not use the encrypted result. Instead, the function parameter will be used.

Flash Encryption Operating Conditions:

- During SPI Flash Boot

  If the DPORT_SPI_ENCRYPT_ENABLE bit of register DPORT_SLAVE_SPI_CONFIG_REG is 1, the Flash Encryption block is operational. Otherwise, it is not.

- During Download Boot

  If the DPORT_SPI_ENCRYPT_ENABLE bit of register DPORT_SLAVE_SPI_CONFIG_REG is 1, and system parameter download_dis_encrypt is 0, the Flash Encryption block is operational. Otherwise, it is not.

Even though software participates in the whole process, it cannot directly read the encrypted codes. Instead, the encrypted codes are integrated into the off-chip flash. Even though the CPU can skip the cache and get the encrypted code directly by reading the off-chip flash, the software can by no means access Key_e.

25.3.3 Flash Decryption Block

Flash Decryption is not a conventional peripheral, and is not equipped with registers. Therefore, the CPU cannot directly access the Flash Decryption block. The Peripheral DPort Register, system parameters and Booting Mode jointly control and configure the Flash Decryption block.

When the Flash Decryption block is operating, the CPU will read instructions and data from the off-chip flash via the cache. The Flash Decryption block automatically decrypts the instructions and data in the cache. The entire decryption process does not need software intervention and is transparent to the cache. The decryption algorithm can decrypt the code that has been encrypted by the Flash Encryption block. Software cannot access the key algorithm Key_d used.

When the Flash Encryption block is not operating, it does not have any effect on the contents stored in the off-chip flash, be they encrypted or unencrypted. What the CPU reads via the cache is the original information stored in the off-chip flash.

Flash Encryption Operating Conditions:

- During SPI Flash Boot

  In the low 7 bits of flash_crypt_cnt, if the number of value 1 is odd, the Flash Decryption block is operational. Otherwise, it is not.

- During Download Boot

  If the DPORT_SPI_DECRYPT_ENABLE bit in DPORT_SLAVE_SPI_CONFIG_REG is 1, and system parameter download_dis_decrypt is 0, the Flash Decryption block is operational. Otherwise, it is not.

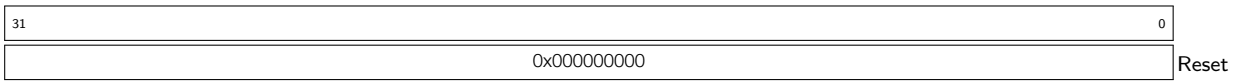
25.4 Register Summary

Name	Description	Address	Access
FLASH_ENCRYPTION_BUFFER_0_REG	Flash encryption buffer register 0	0x3FF5B000	WO
FLASH_ENCRYPTION_BUFFER_1_REG	Flash encryption buffer register 1	0x3FF5B004	WO
FLASH_ENCRYPTION_BUFFER_2_REG	Flash encryption buffer register 2	0x3FF5B008	WO

Name	Description	Address	Access
FLASH_ENCRYPTION_BUFFER_3_REG	Flash encryption buffer register 3	0x3FF5B00C	WO
FLASH_ENCRYPTION_BUFFER_4_REG	Flash encryption buffer register 4	0x3FF5B010	WO
FLASH_ENCRYPTION_BUFFER_5_REG	Flash encryption buffer register 5	0x3FF5B014	WO
FLASH_ENCRYPTION_BUFFER_6_REG	Flash encryption buffer register 6	0x3FF5B018	WO
FLASH_ENCRYPTION_BUFFER_7_REG	Flash encryption buffer register 7	0x3FF5B01C	WO
FLASH_ENCRYPTION_START_REG	Encrypt operation control register	0x3FF5B020	WO
FLASH_ENCRYPTION_ADDRESS_REG	External flash address register	0x3FF5B024	WO
FLASH_ENCRYPTION_DONE_REG	Encrypt operation status register	0x3FF5B028	RO

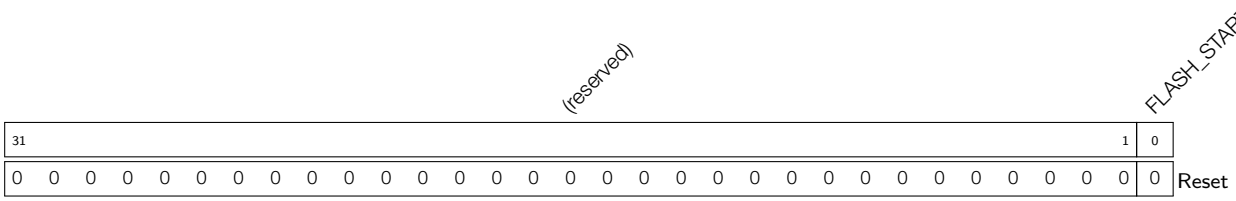
25.5 Register

Register 25.1: FLASH_ENCRYPTION_BUFFER_*n*_REG (*n*: 0-7) (0x0+4**n*)



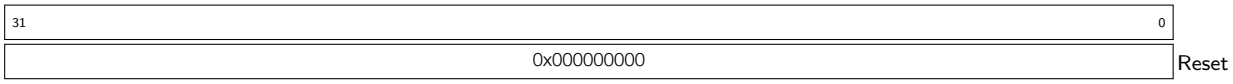
FLASH_ENCRYPTION_BUFFER_*n*_REG Data buffers for encryption. (WO)

Register 25.2: FLASH_ENCRYPTION_START_REG (0x020)



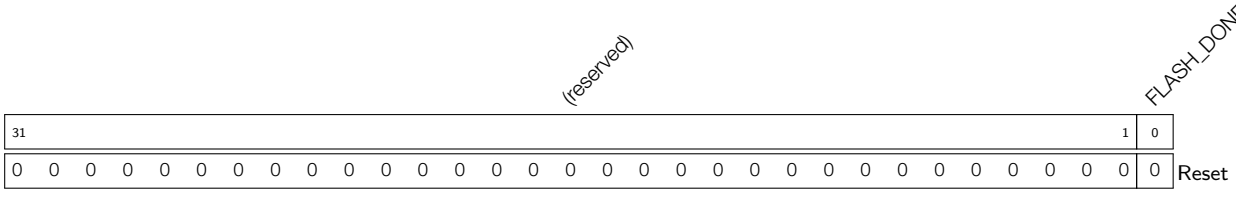
FLASH_START Set this bit to start encryption operation on data buffer. (WO)

Register 25.3: FLASH_ENCRYPTION_ADDRESS_REG (0x024)



FLASH_ENCRYPTION_ADDRESS_REG The physical address on the off-chip flash must be 8-word boundary aligned. (WO)

Register 25.4: FLASH_ENCRYPTION_DONE_REG (0x028)



FLASH_DONE Set this bit when encryption operation is complete. (RO)

26. PID/MPU/MMU

26.1 Introduction

Every peripheral and memory section in the ESP32 is accessed through either an MMU (Memory Management Unit) or an MPU (Memory Protection Unit). An MPU can allow or disallow the access of an application to a memory range or peripheral, depending on what kind of permission the OS has given to that particular application. An MMU can perform the same operation, as well as a virtual-to-physical memory address translation. This can be used to map an internal or external memory range to a certain virtual memory area. These mappings can be application-specific. Therefore, each application can be adjusted and have the memory configuration that is necessary for it to run properly. To differentiate between the OS and applications, there are eight Process Identifiers (or PIDs) that each application, or OS, can run. Furthermore, each application, or OS, is equipped with their own sets of mappings and rights.

26.2 Features

- Eight processes in each of the PRO_CPU and APP_CPU
- MPU/MMU management of on-chip memories, off-chip memories, and peripherals, based on process ID
- On-chip memory management by MPU/MMU
- Off-chip memory management by MMU
- Peripheral management by MPU

26.3 Functional Description

26.3.1 PID Controller

In the ESP32, a PID controller acts as an indicator that signals the MMU/MPU the owner PID of the code that is currently running. The intention is that the OS updates the PID in the PID controller every time it switches context to another application. The PID controller can detect interrupts and automatically switch PIDs to that of the OS, if so configured.

There are two peripheral PID controllers in the system, one for each of the two CPUs in the ESP32. Having a PID controller per CPU allows running different processes on different CPUs, if so desired.

26.3.2 MPU/MMU

The MPU and MMU manage on-chip memories, off-chip memories, and peripherals. To do this they are based on the process of accessing the peripheral or memory region. More specifically, when a code tries to access a MMU/MPU-protected memory region or peripheral, the MMU or MPU will receive the PID from the PID generator that is associated with the CPU on which the process is running.

For on-chip memory and peripherals, the decisions the MMU and MPU make are only based on this PID, whereas the specific CPU the code is running on is not taken into account. Subsequently, the MMU/MPU configuration for the internal memory and peripherals allows entries only for the eight different PIDs. In contrast, the MMU moderating access to the external memory takes not only the PID into account, but also the CPU the request is coming from. This means that MMUs have configuration options for every PID when running on the APP_CPU, as well as every PID when running on the PRO_CPU. While, in practice, accesses from both CPUs will be configured to have the same result for a specific process, doing so is not a hardware requirement.

The decision an MPU can make, based on this information, is to allow or deny a process to access the memory region or peripheral. An MMU has the same function, but additionally it redirects the virtual memory access, which the process acquired, into a physical memory access that can possibly reach out an entirely different physical memory region. This way, MMU-governed memory can be remapped on a process-by-process basis.

26.3.2.1 Embedded Memory

The on-chip memory is governed by fixed-function MPUs, configurable MPUs, and MMUs:

Table 94: MPU and MMU Structure for Internal Memory

Name	Size	Address range		Governed by
		From	To	
ROM0	384 KB	0x4000_0000	0x4005_FFFF	Static MPU
ROM1	64 KB	0x3FF9_0000	0x3FF9_FFFF	Static MPU
SRAM0	64 KB 128 KB	0x4007_0000 0x4008_0000	0x4007_FFFF 0x4009_FFFF	Static MPU SRAM0 MMU
SRAM1 (aliases)	128 KB 128 KB 32 KB	0x3FFE_0000 0x400A_0000 0x4000_0000	0x3FFF_FFFF 0x400B_FFFF 0x4000_7FFF	Static MPU Static MPU Static MPU
SRAM2	72 KB 128 KB	0x3FFA_E000 0x3FFC_0000	0x3FFB_FFFF 0x3FFD_FFFF	Static MPU SRAM2 MMU
RTC FAST (aliases)	8 KB 8 KB	0x3FF8_0000 0x400C_0000	0x3FF8_1FFF 0x400C_1FFF	RTC FAST MPU RTC FAST MPU
RTC SLOW	8 KB	0x5000_0000	0x5000_1FFF	RTC SLOW MPU

Static MPUs

ROM0, ROM1, the lower 64 KB of SRAM0, SRAM1 and the lower 72 KB of SRAM2 are governed by a static MPU. The behaviour of these MPUs are hardwired and cannot be configured by software. They moderate access to the memory region solely through the PID of the current process. When the PID of the process is 0 or 1, the memory can be read (and written when it is RAM) using the addresses specified in Table 94. When it is 2 ~ 7, the memory cannot be accessed.

RTC FAST & RTC SLOW MPU

The 8 KB RTC FAST Memory as well as the 8 KB of RTC SLOW Memory are governed by two configurable MPUs. The MPUs can be configured to allow or deny access to each individual PID, using the RTC_CNTL_RTC_PID_CONFIG_REG and DPORT_AHBLITE_MPU_TABLE_RTC_REG registers. Setting a bit in these registers will allow the corresponding PID to read or write from the memory; clearing the bit disallows access. Access for PID 0 and 1 to RTC SLOW memory cannot be configured and is always enabled. Table 95 and 96 define the bit-to-PID mappings of the registers.

Table 95: MPU for RTC FAST Memory

Size	Boundary address		Authority
	Low	High	PID RTC_CNTL_RTC_PID_CONFIG bit
8 KB	0x3FF8_0000	0x3FF8_1FFF	0 1 2 3 4 5 6 7
8 KB	0x400C_0000	0x400C_1FFF	0 1 2 3 4 5 6 7

Table 96: MPU for RTC SLOW Memory

Size	Boundary address		Authority	
	Low	High	PID = 0/1	PID DPORT_AHBLITE_MPU_TABLE_RTC_REG bit
8 KB	0x5000_0000	0x5000_1FFF	Read/Write	2 3 4 5 6 7 0 1 2 3 4 5

Register RTC_CNTL_RTC_PID_CONFIG_REG is part of the RTC peripheral and can only be modified by processes with a PID of 0; register DPORT_AHBLITE_MPU_TABLE_RTC_REG is a Dport register and can be changed by processes with a PID of 0 or 1.

SRAM0 and SRAM2 upper 128 KB MMUs

Both the upper 128 KB of SRAM0 and the upper 128 KB of SRAM2 are governed by an MMU. Not only can these MMUs allow or deny access to the memory they govern (just like the MPUs do), but they are also capable of translating the address a CPU reads from or writes to (which is a virtual address) to a possibly different address in memory (the physical address).

In order to accomplish this, the internal RAM MMUs divide the memory range they govern into 16 pages. The page size is configurable as 8 KB, 4 KB and 2 KB. When the page size is 8 KB, the 16 pages span the entire 128 KB memory region; when the page size is 4 KB or 2 KB, a non-MMU-covered region of 64 or 96 KB, respectively, will exist at the end of the memory space. Similar to the virtual and physical addresses, it is also possible to imagine the pages as having a virtual and physical component. The MMU can convert an address within a virtual page to an address within a physical page.

For PID 0 and 1, this mapping is 1-to-1, meaning that a read from or write to a certain virtual page will always be converted to a read from or write to the exact same physical page. This allows an operating system, running under PID 0 and/or 1, to always have access to the entire physical memory range.

For PID 2 to 7, however, every virtual page can be reconfigured, on a per-PID basis, to map to a different physical page. This way, reads and writes to an offset within a virtual page get translated into reads and writes to the

same offset within a different physical page. This is illustrated in Figure 124: the CPU (running a process with a PID between 2 to 7) tries to access memory address 0x3FFC_2345. This address is within the virtual Page 1 memory region, at offset 0x0345. The MMU is instructed that for this particular PID, it should translate an access to virtual page 1 into physical Page 2. This causes the memory access to be redirected to the same offset as the virtual memory access, yet in Page 2, which results in the effective access of physical memory address 0x3FFC_**4**345. The page size in this example is 8 KB.

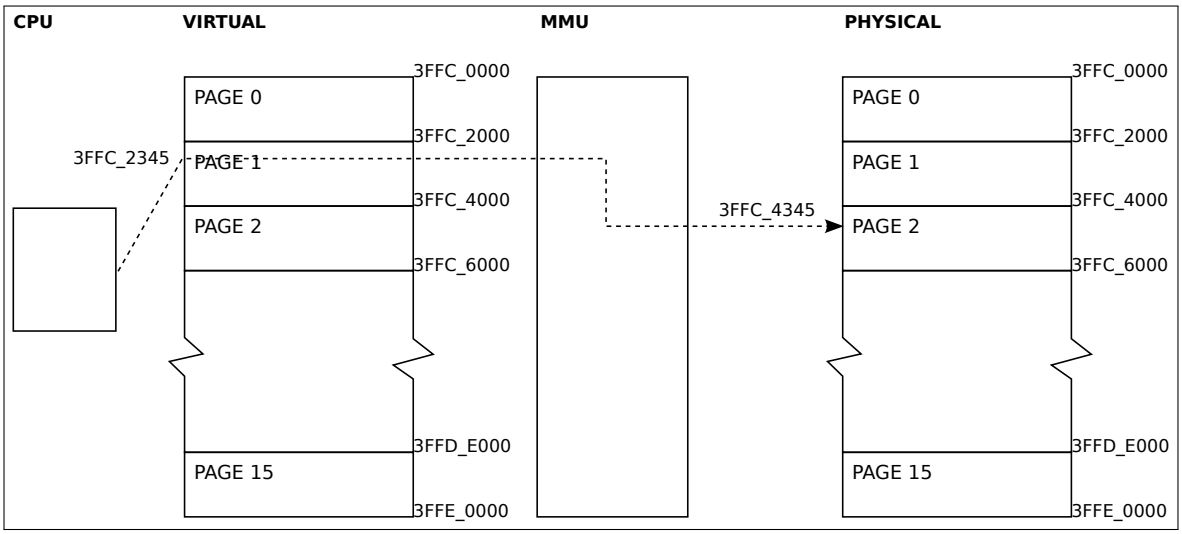


Figure 124: MMU Access Example

Table 97: Page Mode of MMU for the Remaining 128 KB of Internal SRAM0 and SRAM2

DPORT_IMMU_PAGE_MODE	DPORT_DMMU_PAGE_MODE	Page size
0	0	8 KB
1	1	4 KB
2	2	2 KB

Non-MMU Governed Memory

For the MMU-managed region of SRAM0 and SRAM2, the page size is configurable as 8 KB, 4 KB and 2 KB. The configuration is done by setting the DPORT_IMMU_PAGE_MODE (for SRAM0) and DPORT_DMMU_PAGE_MODE (for SRAM2) bits in registers DPORT_IMMU_PAGE_MODE_REG and DPORT_DMMU_PAGE_MODE_REG, as detailed in Table 97. Because the number of pages for either region is fixed at 16, the total amount of memory covered by these pages is 128 KB when 8 KB pages are selected, 64 KB when 4 KB pages are selected, and 32 KB when 2 KB pages are selected. This implies that for 8 KB pages, the entire MMU-managed range is used, but for the other page sizes there will be a part of the 128 KB memory that will not be governed by the MMU settings. Concretely, for a page size of 4 KB, these regions are 0x4009_0000 to 0x4009_FFFF and 0x3FFD_0000 to 0x3FFD_FFFF; for a page size of 2 KB, the regions are 0x4008_8000 to 0x4009_FFFF and 0x3FFC_8000 to 0x3FFD_FFFF. These ranges are readable and writable by processes with a PID of 0 or 1; processes with other PIDs cannot access this memory.

The layout of the pages in memory space is linear, namely, an SRAM0 MMU page n covers address space $0x40080000 + (pagesize * n)$ to $0x40080000 + (pagesize * (n+1) - 1)$; similarily, an SRAM2 MMU page n covers $0x3FFC0000 + (pagesize * n)$ to $0x3FFC0000 + (pagesize * (n+1) - 1)$. Tables 98 and 99 show the resulting addresses in full.

Table 98: Page Boundaries for SRAM0 MMU

Page	8 KB Pages		4 KB Pages		2 KB Pages	
	Bottom	Top	Bottom	Top	Bottom	Top
0	40080000	40081FFF	40080000	40080FFF	40080000	400807FF
1	40082000	40083FFF	40081000	40081FFF	40080800	40080FFF
2	40084000	40085FFF	40082000	40082FFF	40081000	400817FF
3	40086000	40087FFF	40083000	40083FFF	40081800	40081FFF
4	40088000	40089FFF	40084000	40084FFF	40082000	400827FF
5	4008A000	4008BFFF	40085000	40085FFF	40082800	40082FFF
6	4008C000	4008DFFF	40086000	40086FFF	40083000	400837FF
7	4008E000	4008FFFF	40087000	40087FFF	40083800	40083FFF
8	40090000	40091FFF	40088000	40088FFF	40084000	400847FF
9	40092000	40093FFF	40089000	40089FFF	40084800	40084FFF
10	40094000	40095FFF	4008A000	4008AFFF	40085000	400857FF
11	40096000	40097FFF	4008B000	4008BFFF	40085800	40085FFF
12	40098000	40099FFF	4008C000	4008CFFF	40086000	400867FF
13	4009A000	4009BFFF	4008D000	4008DFFF	40086800	40086FFF
14	4009C000	4009DFFF	4008E000	4008EFFF	40087000	400877FF
15	4009E000	4009FFFF	4008F000	4008FFFF	40087800	40087FFF
Rest	-	-	40090000	4009FFFF	4008800	4009FFFF

Table 99: Page Boundaries for SRAM2 MMU

Page	8 KB Pages		4 KB Pages		2 KB Pages	
	Bottom	Top	Bottom	Top	Bottom	Top
0	3FFC0000	3FFC1FFF	3FFC0000	3FFC0FFF	3FFC0000	3FFC07FF
1	3FFC2000	3FFC3FFF	3FFC1000	3FFC1FFF	3FFC0800	3FFC0FFF
2	3FFC4000	3FFC5FFF	3FFC2000	3FFC2FFF	3FFC1000	3FFC17FF
3	3FFC6000	3FFC7FFF	3FFC3000	3FFC3FFF	3FFC1800	3FFC1FFF
4	3FFC8000	3FFC9FFF	3FFC4000	3FFC4FFF	3FFC2000	3FFC27FF
5	3FFCA000	3FFCBFFF	3FFC5000	3FFC5FFF	3FFC2800	3FFC2FFF
6	3FFCC000	3FFCDFFF	3FFC6000	3FFC6FFF	3FFC3000	3FFC37FF
7	3FFCE000	3FFCFFFF	3FFC7000	3FFC7FFF	3FFC3800	3FFC3FFF
8	3FFD0000	3FFD1FFF	3FFC8000	3FFC8FFF	3FFC4000	3FFC47FF
9	3FFD2000	3FFD3FFF	3FFC9000	3FFC9FFF	3FFC4800	3FFC4FFF
10	3FFD4000	3FFD5FFF	3FFCA000	3FFCAFFF	3FFC5000	3FFC57FF
11	3FFD6000	3FFD7FFF	3FFCB000	3FFCBFFF	3FFC5800	3FFC5FFF
12	3FFD8000	3FFD9FFF	3FFCC000	3FFCCFFF	3FFC6000	3FFC67FF
13	3FFDA000	3FFDBFFF	3FFCD000	3FFCDFFF	3FFC6800	3FFC6FFF
14	3FFDC000	3FFDDFFF	3FFCE000	3FFCEFFF	3FFC7000	3FFC77FF
15	3FFDE000	3FFDFFFF	3FFCF000	3FFCFFFF	3FFC7800	3FFC7FFF
Rest	-	-	3FFD0000	3FFDFFFF	3FFC8000	3FFDFFFF

MMU Mapping

For each of the SRAM0 and SRAM2 MMUs, access rights and virtual to physical page mapping are done by a set of 16 registers. In contrast to most of the other MMUs, each register controls a physical page, not a virtual one. These registers control which of the PIDs have access to the physical memory, as well as which virtual page maps to this physical page. The bits in the register are described in Table 100. Keep in mind that these registers only govern accesses from processes with PID 2 to 7; PID 0 and 1 always have full read and write access to all pages and no virtual-to-physical mapping is done. In other words, if a process with a PID of 0 or 1 accesses virtual page *x*, the access will always go to physical page *x*, regardless of these register settings. These registers, as well as the page size selection registers DPORT_IMMU_PAGE_MODE_REG and DPORT_DMMU_PAGE_MODE_REG, are only writable from a process with PID 0 or 1.

Table 100: DPORT_DMMU_TABLE*n*_REG & DPORT_IMMU_TABLE*n*_REG

[6:4]	Access rights for PID 2 ~ 7
0	None of PIDs 2 ~ 7 have access.
1	All of PIDs 2 ~ 7 have access.
2	Only PID 2 has access.
3	Only PID 3 has access.
4	Only PID 4 has access.
5	Only PID 5 has access.
6	Only PID 6 has access.
7	Only PID 7 has access.

[3:0]	Address authority
0x00	Virtual page 0 accesses this physical page.
0x01	Virtual page 1 accesses this physical page.
0x02	Virtual page 2 accesses this physical page.
0x03	Virtual page 3 accesses this physical page.
0x04	Virtual page 4 accesses this physical page.
0x05	Virtual page 5 accesses this physical page.
0x06	Virtual page 6 accesses this physical page.
0x07	Virtual page 7 accesses this physical page.
0x08	Virtual page 8 accesses this physical page.
0x09	Virtual page 9 accesses this physical page.
0x10	Virtual page 10 accesses this physical page.
0x11	Virtual page 11 accesses this physical page.
0x12	Virtual page 12 accesses this physical page.
0x13	Virtual page 13 accesses this physical page.
0x14	Virtual page 14 accesses this physical page.
0x15	Virtual page 15 accesses this physical page.

Differences Between SRAM0 and SRAM2 MMU

The memory governed by the SRAM0 MMU is accessed through the processors I-bus, while the processor accesses the memory governed by the SRAM2 MMU through the D-bus. Thus, the normal envisioned use is for the code to be stored in the SRAM0 MMU pages and data in the MMU pages of SRAM2. In general, applications running under a PID of 2 to 7 are not expected to modify their own code, because for these PIDs access to the MMU pages of SRAM0 is read-only. These applications must, however, be able to modify their data section, so that they are allowed to read as well as write MMU pages located in SRAM2. As stated before, processes running under PID 0 or 1 always have full read-and-write access to both memory ranges.

DMA MPU

Applications may want to configure the DMA to send data straight from or to the peripherals they can control. With access to DMA, a malicious process may also be able to copy data from or to a region it cannot normally

access. In order to be secure against that scenario, there is a DMA MPU which can be used to disallow DMA transfers from memory regions with sensitive data in them.

For each 8 KB region in the SRAM1 and SRAM2 regions, there is a bit in the DPORT_AHB_MPU_TABLE_*n*_REG registers which tells the MPU to either allow or disallow DMA access to this region. The DMA MPU uses only these bits to decide if a DMA transfer can be started; the PID of the process is not a factor. This means that when the OS wants to restrict its processes in a heterogenous fashion, it will need to re-load these registers with the values applicable to the process to be run on every context switch.

The register bits that govern access to the 8 KB regions are detailed in Table 101. When a register bit is set, DMA can read/write the corresponding 8 KB memory range. When the bit is cleared, access to that memory range is denied.

Table 101: MPU for DMA

Size	Boundary address		Authority	
	Low	High	Register	Bit
Internal SRAM 2				
8 KB	0x3FFA_E000	0x3FFA_FFFF	DPORT_AHB_MPU_TABLE_0_REG	0
8 KB	0x3FFB_0000	0x3FFB_1FFF	DPORT_AHB_MPU_TABLE_0_REG	1
8 KB	0x3FFB_2000	0x3FFB_3FFF	DPORT_AHB_MPU_TABLE_0_REG	2
8 KB	0x3FFB_4000	0x3FFB_5FFF	DPORT_AHB_MPU_TABLE_0_REG	3
8 KB	0x3FFB_6000	0x3FFB_7FFF	DPORT_AHB_MPU_TABLE_0_REG	4
8 KB	0x3FFB_8000	0x3FFB_9FFF	DPORT_AHB_MPU_TABLE_0_REG	5
8 KB	0x3FFB_A000	0x3FFB_BFFF	DPORT_AHB_MPU_TABLE_0_REG	6
8 KB	0x3FFB_C000	0x3FFB_DFFF	DPORT_AHB_MPU_TABLE_0_REG	7
8 KB	0x3FFB_E000	0x3FFB_FFFF	DPORT_AHB_MPU_TABLE_0_REG	8
8 KB	0x3FFC_0000	0x3FFC_1FFF	DPORT_AHB_MPU_TABLE_0_REG	9
8 KB	0x3FFC_2000	0x3FFC_3FFF	DPORT_AHB_MPU_TABLE_0_REG	10
8 KB	0x3FFC_4000	0x3FFC_5FFF	DPORT_AHB_MPU_TABLE_0_REG	11
8 KB	0x3FFC_6000	0x3FFC_7FFF	DPORT_AHB_MPU_TABLE_0_REG	12
8 KB	0x3FFC_8000	0x3FFC_9FFF	DPORT_AHB_MPU_TABLE_0_REG	13
8 KB	0x3FFC_A000	0x3FFC_BFFF	DPORT_AHB_MPU_TABLE_0_REG	14
8 KB	0x3FFC_C000	0x3FFC_DFFF	DPORT_AHB_MPU_TABLE_0_REG	15
8 KB	0x3FFC_E000	0x3FFC_FFFF	DPORT_AHB_MPU_TABLE_0_REG	16
8 KB	0x3FFD_0000	0x3FFD_1FFF	DPORT_AHB_MPU_TABLE_0_REG	17
8 KB	0x3FFD_2000	0x3FFD_3FFF	DPORT_AHB_MPU_TABLE_0_REG	18
8 KB	0x3FFD_4000	0x3FFD_5FFF	DPORT_AHB_MPU_TABLE_0_REG	19
8 KB	0x3FFD_6000	0x3FFD_7FFF	DPORT_AHB_MPU_TABLE_0_REG	20
8 KB	0x3FFD_8000	0x3FFD_9FFF	DPORT_AHB_MPU_TABLE_0_REG	21
8 KB	0x3FFD_A000	0x3FFD_BFFF	DPORT_AHB_MPU_TABLE_0_REG	22
8 KB	0x3FFD_C000	0x3FFD_DFFF	DPORT_AHB_MPU_TABLE_0_REG	23
8 KB	0x3FFD_E000	0x3FFD_FFFF	DPORT_AHB_MPU_TABLE_0_REG	24
Internal SRAM 1				
8 KB	0x3FFE_0000	0x3FFE_1FFF	DPORT_AHB_MPU_TABLE_0_REG	25
8 KB	0x3FFE_2000	0x3FFE_3FFF	DPORT_AHB_MPU_TABLE_0_REG	26
8 KB	0x3FFE_4000	0x3FFE_5FFF	DPORT_AHB_MPU_TABLE_0_REG	27
8 KB	0x3FFE_6000	0x3FFE_7FFF	DPORT_AHB_MPU_TABLE_0_REG	28

Size	Boundary address		Authority	
	Low	High	Register	Bit
8 KB	0x3FFE_8000	0x3FFE_9FFF	DPORT_AHB_MPU_TABLE_0_REG	29
8 KB	0x3FFE_A000	0x3FFE_BFFF	DPORT_AHB_MPU_TABLE_0_REG	30
8 KB	0x3FFE_C000	0x3FFE_DFFF	DPORT_AHB_MPU_TABLE_0_REG	31
8 KB	0x3FFE_E000	0x3FFE_FFFF	DPORT_AHB_MPU_TABLE_1_REG	0
8 KB	0x3FFF_0000	0x3FFF_1FFF	DPORT_AHB_MPU_TABLE_1_REG	1
8 KB	0x3FFF_2000	0x3FFF_3FFF	DPORT_AHB_MPU_TABLE_1_REG	2
8 KB	0x3FFF_4000	0x3FFF_5FFF	DPORT_AHB_MPU_TABLE_1_REG	3
8 KB	0x3FFF_6000	0x3FFF_7FFF	DPORT_AHB_MPU_TABLE_1_REG	4
8 KB	0x3FFF_8000	0x3FFF_9FFF	DPORT_AHB_MPU_TABLE_1_REG	5
8 KB	0x3FFF_A000	0x3FFF_BFFF	DPORT_AHB_MPU_TABLE_1_REG	6
8 KB	0x3FFF_C000	0x3FFF_DFFF	DPORT_AHB_MPU_TABLE_1_REG	7
8 KB	0x3FFF_E000	0x3FFF_FFFF	DPORT_AHB_MPU_TABLE_1_REG	8

Registers DPROT_AHB_MPU_TABLE_0_REG DPROT_AHB_MPU_TABLE_1_REG are located in the DPort address space. Only processes with a PID of 0 or 1 can modify these two registers.

26.3.2.2 External Memory

Accesses to the external flash and external SPI RAM are done through a cache and are also handled by an MMU. This Cache MMU can apply different mappings, depending on the PID of the process as well as the CPU the process is running on. The MMU does this in a way that is similar to the internal memory MMU, that is, for every page of virtual memory, it has a register detailing which physical page this virtual page should map to. There are differences between the MMUs governing the internal memory and the Cache MMU, though. First of all, the Cache MMU has a fixed page size (which is 64 KB for external flash and 32 KB for external RAM) and secondly, instead of specifying access rights in the MMU entries, the Cache MMU has explicit mapping tables for each PID and processor core. The MMU mapping configuration registers will be referred to as 'entries' in the rest of this chapter. These registers are only accessible from processes with a PID of 0 or 1; processes with a PID of 2 to 7 will have to delegate to one of the above-mentioned processes to change their MMU settings.

The MMU entries, as stated before, are used for mapping a virtual memory page access to a physical memory page access. The MMU controls five regions of virtual address space, detailed in Table 102. $VAddr_1$ to $VAddr_4$ are used for accessing external flash, whereas $VAddr_{RAM}$ is used for accessing external RAM. Note that $VAddr_4$ is a subset of $VAddr_0$.

Table 102: Virtual Address for External Memory

Name	Size	Boundary address		Page quantity
		Low	High	
$VAddr_0$	4 MB	0x3F40_0000	0x3F7F_FFFF	64
$VAddr_1$	4 MB	0x4000_0000	0x403F_FFFF	64*
$VAddr_2$	4 MB	0x4040_0000	0x407F_FFFF	64
$VAddr_3$	4 MB	0x4080_0000	0x40BF_FFFF	64
$VAddr_4$	1 MB	0x3F40_0000	0x3F4F_FFFF	16
$VAddr_{RAM}$	4 MB	0x3F80_0000	0x3FBF_FFFF	128

* The configuration entries for address range 0x4000_0000 ~ 0x403F_FFFF are implemented and documented as if it were a full 4 MB address range, but it is not accessible as such. Instead, the address range 0x4000_0000 ~ 0x400C_1FFF accesses on-chip memory. This means that some of the configuration entries for $VAddr_1$ will not be used.

External Flash

For flash, the relationships among entry numbers, virtual memory ranges, and PIDs are detailed in Tables 103 and 104, which for every memory region and PID combination specify the first MMU entry governing the mapping. This number refers to the MMU entry governing the very first page; the entire region is described by the amount of pages specified in the 'count' column.

These two tables are essentially the same, with the sole difference being that the APP_CPU entry numbers are 2048 higher than the corresponding PRO_CPU numbers. Note that memory regions $VAddr_0$ and $VAddr_1$ are only accessible using PID 0 and 1, while $VAddr_4$ can only be accessed by PID 2 ~ 7.

Table 103: MMU Entry Numbers for PRO_CPU

VAddr	Count	First MMU entry for PID						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	0	-	-	-	-	-	-
$VAddr_1$	64	64	-	-	-	-	-	-
$VAddr_2$	64	128	256	384	512	640	768	896
$VAddr_3$	64	192	320	448	576	704	832	960
$VAddr_4$	16	-	1056	1072	1088	1104	1120	1136

Table 104: MMU Entry Numbers for APP_CPU

VAddr	Count	First MMU entry for PID						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	2048	-	-	-	-	-	-
$VAddr_1$	64	2112	-	-	-	-	-	-
$VAddr_2$	64	2176	2304	2432	2560	2688	2816	2944
$VAddr_3$	64	2240	2368	2496	2624	2752	2880	3008
$VAddr_4$	16	-	3104	3120	3136	3152	3168	3184

As these tables show, virtual address $VAddr_1$ can only be used by processes with a PID of 0 or 1. There is a

special mode to allow processes with a PID of 2 to 7 to read the External Flash via address $VAddr_1$. When the DPORT_PRO_SINGLE_IRAM_ENA bit of register DPORT_PRO_CACHE_CTRL_REG is 1, the MMU enters this special mode for PRO_CPU memory accesses. Similarily, when the DPORT_APP_SINGLE_IRAM_ENA bit of register DPORT_APP_CACHE_CTRL_REG is 1, the APP_CPU accesses memory using this special mode. In this mode, the process and virtual address page supported by each configuration entry of MMU are different. For details please see Table 105 and 106. As shown in these tables, in this special mode $VAddr_2$ and $VAddr_3$ cannot be used to access External Flash.

Table 105: MMU Entry Numbers for PRO_CPU (Special Mode)

VAddr	Count	First MMU entry for PID						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	0	-	-	-	-	-	-
$VAddr_1$	64	64	256	384	512	640	768	896
$VAddr_2$	64	-	-	-	-	-	-	-
$VAddr_3$	64	-	-	-	-	-	-	-
$VAddr_4$	16	-	1056	1072	1088	1104	1120	1136

Table 106: MMU Entry Numbers for APP_CPU (Special Mode)

VAddr	Count	First MMU entry for PID						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	2048	-	-	-	-	-	-
$VAddr_1$	64	2112	2304	2432	2560	2688	2816	2944
$VAddr_2$	64	-	-	-	-	-	-	-
$VAddr_3$	64	-	-	-	-	-	-	-
$VAddr_4$	16	-	3104	3120	3136	3152	3168	3184

Every configuration entry of MMU maps a virtual address page of a CPU process to a physical address page. An entry is 32 bits wide. Of these, bits 0~7 indicate the physical page the virtual page is mapped to. Bit 8 should be cleared to indicate that the MMU entry is valid; entries with this bit set will not map any physical address to the virtual address. Bits 10 to 32 are unused and should be written as zero. Because there are eight address bits in an MMU entry, and the page size for external flash is 64 KB, a maximum of 256 * 64 KB = 16 MB of external flash is supported.

Examples

Example 1. A PRO_CPU process, with a PID of 1, needs to read external flash address 0x07_2375 via virtual address 0x3F70_2375. The MMU is not in the special mode.

- According to Table 102, virtual address 0x3F70_2375 resides in the 0x30'th page of $VAddr_0$.
- According to Table 103, the MMU entry for $VAddr_0$ for PID 0/1 for the PRO_CPU starts at 0.
- The modified MMU entry is 0 + 0x30 = 0x30.
- Address 0x07_2375 resides in the 7'th 64 KB-sized page.
- MMU entry 0x30 needs to be set to 7 and marked as valid by setting the 8'th bit to 0. Thus, 0x007 is written to MMU entry 0x30.

Example 2. An APP_CPU process, with a PID of 4, needs to read external flash address 0x44_048C via virtual address 0x4044_048C. The MMU is not in special mode.

- According to Table 102, virtual address 0x4044_048C resides in the 0x4'th page of $VAddr_2$.
- According to Table 104, the MMU entry for $VAddr_2$ for PID 4 for the APP_CPU starts at 2560.
- The modified MMU entry is 2560 + 0x4 = 2564.
- Address 0x44_048C resides in the 0x44'th 64 KB-sized page.
- MMU entry 2564 needs to be set to 0x44 and marked as valid by setting the 8'th bit to 0. Thus, 0x044 is written to MMU entry 2564.

External RAM

Processes running on PRO_CPU and APP_CPU can read and write External SRAM via the Cache at virtual address range $VAddr_{RAM}$, which is 0x3F80_0000 ~ 0x3FBF_FFFF. As with the flash MMU, the address space and the physical memory are divided into pages. For the External RAM MMU, the page size is 32 KB and the MMU is able to map 256 physical pages into the virtual address space, allowing for 32 KB * 256 = 8 MB of physical external RAM to be mapped.

The mapping of virtual pages into this memory range depends on the mode this MMU is in: Low-High mode, Even-Odd mode, or Normal mode. In all cases, the DPORT_PRO_DRAM_HL bit and DPORT_PRO_DRAM_SPLIT bit in register DPORT_PRO_CACHE_CTRL_REG, the DPORT_APP_DRAM_HL bit and DPORT_APP_DRAM_SPLIT bit in register DPORT_APP_CACHE_CTRL_REG determine the virtual address mode for External SRAM. For details, please see Table 107. If a different mapping for the PRO_CPU and APP_CPU is required, the Normal Mode should be selected, as it is the only mode that can provide this. If it is allowable for the PRO_CPU and the APP_CPU to share the same mapping, using either High-Low or Even-Odd mode can give a speed gain when both CPUs access memory frequently.

In case the APP_CPU cache is disabled, which renders the region of 0x4007_8000 to 0x4007_FFFF usable as normal internal RAM, the usability of the various cache modes changes. Normal mode will allow PRO_CPU access to external RAM to keep functioning, but the APP_CPU will be unable to access the external RAM. High-Low mode allows both CPUs to use external RAM, but only for the 2 MB virtual memory addresses from 0x3F80_0000 to 0x3F9F_FFFF. It is not advised to use Even-Odd mode with the APP_CPU cache region disabled.

Table 107: Virtual Address Mode for External SRAM

Mode	DPORT_PRO_DRAM_HL DPORT_APP_DRAM_HL	DPORT_PRO_DRAM_SPLIT DPORT_APP_DRAM_SPLIT
Low-High	1	0
Even-Odd	0	1
Normal	0	0

In normal mode, the virtual-to-physical page mapping can be different for both CPUs. Page mappings for PRO_CPU are set using the MMU entries for ${}^{L}VAddr_{RAM}$, and page mappings for the APP_CPU can be configured using the MMU entries for ${}^{R}VAddr_{RAM}$. In this mode, all 128 pages of both ${}^{L}VAddr$ and ${}^{R}VAddr$ are fully used, allowing a maximum of 8 MB of memory to be mapped; 4 MB into PRO_CPU address space and a possibly different 4 MB into the APP_CPU address space, as can be seen in Table 108.

Table 108: Virtual Address for External SRAM (Normal Mode)

Virtual address	Size	PRO_CPU address	
		Low	High
$^{L}VAddr_{RAM}$	4 MB	0x3F80_0000	0x3FBF_FFFF
Virtual address	Size	APP_CPU address	
		Low	High
$^{R}VAddr_{RAM}$	4 MB	0x3F80_0000	0x3FBF_FFFF

In Low-High mode, both the PRO_CPU and the APP_CPU use the same mapping entries. In this mode $^{L}VAddr_{RAM}$ is used for the lower 2 MB of the virtual address space, while $^{R}VAddr_{RAM}$ is used for the upper 2 MB. This also means that the upper 64 MMU entries for $^{L}VAddr_{RAM}$, as well as the lower 64 entries for $^{R}VAddr_{RAM}$, are unused. Table 109 details these address ranges.

Table 109: Virtual Address for External SRAM (Low-High Mode)

Virtual address	Size	PRO_CPU/APP_CPU address	
		Low	High
$^{L}VAddr_{RAM}$	2 MB	0x3F80_0000	0x3F9F_FFFF
$^{R}VAddr_{RAM}$	2 MB	0x3FA0_0000	0x3FBF_FFFF

In Even-Odd memory, the VRAM is split into 32-byte chunks. The even chunks are resolved through the MMU entries for $^{L}VAddr_{RAM}$, the odd chunks through the entries for $^{R}VAddr_{RAM}$. Generally, the MMU entries for $^{L}VAddr_{RAM}$ and $^{R}VAddr_{RAM}$ are set to the same values, so that the virtual pages map to a contiguous region of physical memory. Table 110 details this mode.

Table 110: Virtual Address for External SRAM (Even-Odd Mode)

Virtual address	Size	PRO_CPU/APP_CPU address	
		Low	High
$^{L}VAddr_{RAM}$	32 Bytes	0x3F80_0000	0x3F80_001F
$^{R}VAddr_{RAM}$	32 Bytes	0x3F80_0020	0x3F80_003F
$^{L}VAddr_{RAM}$	32 Bytes	0x3F80_0040	0x3F80_005F
$^{R}VAddr_{RAM}$	32 Bytes	0x3F80_0060	0x3F80_007F
...			
$^{L}VAddr_{RAM}$	32 Bytes	0x3FBF_FFC0	0x3FBF_FFDF
$^{R}VAddr_{RAM}$	32 Bytes	0x3FBF_FFE0	0x3FBF_FFFF

The bit configuration of the External RAM MMU entries is the same as for the flash memory: the entries are 32-bit registers, with the lower nine bits being used. Bits 0~7 contain the physical page the entry should map its associate virtual page address to, while bit 8 is cleared when the entry is valid and set when it is not. Table 111 details the first MMU entry number for $^{L}VAddr_{RAM}$ and $^{R}VAddr_{RAM}$ for all PIDs.

Table 111: MMU Entry Numbers for External RAM

VAddr	Count	First MMU entry for PID						
		0/1	2	3	4	5	6	7
${}^{L}VAddr_{RAM}$	128	1152	1280	1408	1536	1664	1792	1920
${}^{R}VAddr_{RAM}$	128	3200	3328	3456	3584	3712	3840	3968

Examples

Example 1. A PRO_CPU process, with a PID of 7, needs to read or write external RAM address 0x7F_A375 via virtual address 0x3FA7_2375. The MMU is in Low-High mode.

- According to Table 102, virtual address 0x3FA7_2375 resides in the 0x4E'th 32-KB-page of $VAddr_{RAM}$.
- According to Table 109, virtual address 0x3FA7_2375 is governed by ${}^{R}VAddr_{RAM}$.
- According to Table 111, the MMU entry for ${}^{R}VAddr_{RAM}$ for PID 7 for the PRO_CPU starts at 3968.
- The modified MMU entry is 3968 + 0x4E = 4046.
- Address 0x7F_A375 resides in the 255'th 32 KB-sized page.
- MMU entry 4046 needs to be set to 255 and marked as valid by clearing the 8'th bit. Thus, 0x0FF is written to MMU entry 4046.

Example 2. An APP_CPU process, with a PID of 5, needs to read or write external RAM address 0x55_5805 up to 0x55_5823 starting at virtual address 0x3F85_5805. The MMU is in Even-Odd mode.

- According to Table 102, virtual address 0x3F85_5805 resides in the 0x0A'th 32-KB-page of $VAddr_{RAM}$.
- According to Table 110, the range to be read/written spans both a 32-byte region in ${}^{R}VAddr_{RAM}$ and ${}^{L}VAddr_{RAM}$.
- According to Table 111, the MMU entry for ${}^{L}VAddr_{RAM}$ for PID 5 starts at 1664.
- According to Table 111, the MMU entry for ${}^{R}VAddr_{RAM}$ for PID 5 starts at 3712.
- The modified MMU entries are 1664 + 0x0A = 1674 and 3712 + 0x0A = 3722.
- The addresses 0x55_5805 to 0x55_5823 reside in the 0xAA'th 32 KB-sized page.
- MMU entries 1674 and 3722 need to be set to 0xAA and marked as valid by setting the 8'th bit to 0. Thus, 0x0AA is written to MMU entries 1674 and 3722. This mapping applies to both the PRO_CPU and the APP_CPU.

Example 3. A PRO_CPU process, with a PID of 1, and an APP_CPU process whose PID is also 1, need to read or write external RAM using virtual address 0x3F80_0876. The PRO_CPU needs this region to access physical address 0x10_0876, while the APP_CPU wants to access physical address 0x20_0876 through this virtual address. The MMU is in Normal mode.

- According to Table 102, virtual address 0x3F80_0876 resides in the 0'th 32-KB-page of $VAddr_{RAM}$.
- According to Table 111, the MMU entry for PID 1 for the PRO_CPU starts at 1152.
- According to Table 111, the MMU entry for PID 1 for the APP_CPU starts at 3200.
- The MMU entries that are modified are 1152 + 0 = 1152 for the PRO_CPU and 3200 + 0 = 3200 for the APP_CPU.
- Address 0x10_0876 resides in the 0x20'th 32 KB-sized page.
- Address 0x20_0876 resides in the 0x40'th 32 KB-sized page.
- For the PRO_CPU, MMU entry 1152 needs to be set to 0x20 and marked as valid by clearing the 8'th bit. Thus, 0x020 is written to MMU entry 1152.

- For the APP_CPU, MMU entry 3200 needs to be set to 0x40 and marked as valid by clearing the 8'th bit. Thus, 0x040 is written to MMU entry 3200.
- Now, the PRO_CPU and the APP_CPU can access different physical memory regions through the same virtual address.

26.3.2.3 Peripheral

The Peripheral MPU manages the 41 peripheral modules. This MMU can be configured per peripheral to only allow access from a process with a certain PID. The registers to configure this are detailed in Table 112.

Table 112: MPU for Peripheral

Peripheral	Authority	
	PID = 0/1	PID = 2 ~ 7
DPort Register	Access	Forbidden
AES Accelerator	Access	Forbidden
RSA Accelerator	Access	Forbidden
SHA Accelerator	Access	Forbidden
Secure Boot	Access	Forbidden
Cache MMU Table	Access	Forbidden
PID Controller	Access	Forbidden
UART0	Access	DPORT_AHBLITE_MPU_TABLE_UART_REG
SPI1	Access	DPORT_AHBLITE_MPU_TABLE_SPI1_REG
SPI0	Access	DPORT_AHBLITE_MPU_TABLE_SPI0_REG
GPIO	Access	DPORT_AHBLITE_MPU_TABLE_GPIO_REG
RTC	Access	DPORT_AHBLITE_MPU_TABLE_RTC_REG
IO MUX	Access	DPORT_AHBLITE_MPU_TABLE_IO_MUX_REG
SDIO Slave	Access	DPORT_AHBLITE_MPU_TABLE_HINF_REG
UDMA1	Access	DPORT_AHBLITE_MPU_TABLE_UHCI1_REG
I2S0	Access	DPORT_AHBLITE_MPU_TABLE_I2S0_REG
UART1	Access	DPORT_AHBLITE_MPU_TABLE_UART1_REG
I2C0	Access	DPORT_AHBLITE_MPU_TABLE_I2C_EXT0_REG
UDMA0	Access	DPORT_AHBLITE_MPU_TABLE_UHCI0_REG
SDIO Slave	Access	DPORT_AHBLITE_MPU_TABLE_SLCHOST_REG
RMT	Access	DPORT_AHBLITE_MPU_TABLE_RMT_REG
PCNT	Access	DPORT_AHBLITE_MPU_TABLE_PCNT_REG
SDIO Slave	Access	DPORT_AHBLITE_MPU_TABLE_SLC_REG
LED PWM	Access	DPORT_AHBLITE_MPU_TABLE_LEDC_REG
Efuse Controller	Access	DPORT_AHBLITE_MPU_TABLE_EFUSE_REG
Flash Encryption	Access	DPORT_AHBLITE_MPU_TABLE_SPI_ENCRYPT_REG
PWM0	Access	DPORT_AHBLITE_MPU_TABLE_PWM0_REG
TIMG0	Access	DPORT_AHBLITE_MPU_TABLE_TIMERGROUP_REG
TIMG1	Access	DPORT_AHBLITE_MPU_TABLE_TIMERGROUP1_REG
SPI2	Access	DPORT_AHBLITE_MPU_TABLE_SPI2_REG
SPI3	Access	DPORT_AHBLITE_MPU_TABLE_SPI3_REG
SYSCON	Access	DPORT_AHBLITE_MPU_TABLE_APB_CTRL_REG

Peripheral	Authority	
	PID = 0/1	PID = 2 ~ 7
I2C1	Access	DPORT_AHBLITE_MPU_TABLE_I2C_EXT1_REG
SDMMC	Access	DPORT_AHBLITE_MPU_TABLE_SDIO_HOST_REG
EMAC	Access	DPORT_AHBLITE_MPU_TABLE_EMAC_REG
PWM1	Access	DPORT_AHBLITE_MPU_TABLE_PWM1_REG
I2S1	Access	DPORT_AHBLITE_MPU_TABLE_I2S1_REG
UART2	Access	DPORT_AHBLITE_MPU_TABLE_UART2_REG
PWM2	Access	DPORT_AHBLITE_MPU_TABLE_PWM2_REG
PWM3	Access	DPORT_AHBLITE_MPU_TABLE_PWM3_REG
RNG	Access	DPORT_AHBLITE_MPU_TABLE_PWR_REG

Each bit of register DPORT_AHBLITE_MPU_TABLE_*X*_REG determines whether each process can access the peripherals managed by the register. For details please see Table 113. When a bit of register DPORT_AHBLITE_MPU_TABLE_*X*_REG is 1, it means that a process with the corresponding PID can access the corresponding peripheral of the register. Otherwise, the process cannot access the corresponding peripheral.

Table 113: DPORT_AHBLITE_MPU_TABLE_*X*_REG

PID	2 3 4 5 6 7
DPORT_AHBLITE_MPU_TABLE_*X*_REG bit	0 1 2 3 4 5

All the DPORT_AHBLITE_MPU_TABLE_*X*_REG registers are in peripheral DPort Register. Only processes with PID 0/1 can modify these registers.

27. PID Controller

27.1 Overview

The ESP32 is a dual core device and is capable of running and managing multiple processes. The PID Controller supports switching of PID when a process switch occurs. In addition to PID management, the PID Controller also facilitates management of nested interrupts by recording execution status just before an interrupt service routine is executed. This enables the user application to manage process switches and nested interrupts more efficiently.

27.2 Features

The PID Controller features:

- Process management and priority
- Process PID switch
- Interrupt information recording
- Nested interrupt management

27.3 Functional Description

Eight processes run on the CPU, and are assigned with PID of 0 ~ 7 respectively. Among the eight processes, processes with PID of 0 or 1 are elevated processes with higher authority compared to processes with PID ranging from 2 ~ 7.

A CPU process switch may occur in two cases:

- An interrupt occurs and the CPU fetches an instruction from the interrupt vector. Instruction fetch or execution from interrupt vector is always treated as a process with PID of 0, irrespective of which process was being executed on the CPU when the interrupt occurred.
- A currently active process explicitly performs a process switch. Only elevated processes with PID of 0 or 1 may perform a process switch.

27.3.1 Interrupt Identification

Interrupts are classified into seven priority levels: Level 1, Level 2, Level 3, Level 4, Level 5, Level 6 (Debug), and NMI. Each level of interrupt is assigned an interrupt vector entry address. The PID Controller recognizes CPU instruction fetch from an interrupt vector entry address and automatically switches PID to 0. If CPU only accesses the interrupt vector entry address, PID Controller performs no action.

PIDCTRL_INTERRUPT_ENABLE_REG determines whether the PID Controller identifies and registers an interrupt of certain priority. When a bit of register PIDCTRL_INTERRUPT_ENABLE_REG is 1, PID Controller will take action when CPU fetches instruction from the interrupt vector entry address of the corresponding interrupt. Otherwise, PID Controller performs no action. The registers PIDCTRL_INTERRUPT_ADDR_1_REG ~ PIDCTRL_INTERRUPT_ADDR_7_REG define the interrupt vector entry address for all the interrupt priority levels. For details please refer to Table 114.

Table 114: Interrupt Vector Entry Address

Priority level	PIDCTRL_INTERRUPT_ENABLE_REG bit controlling interrupt identification	Interrupt vector entry address
Level 1	1	PIDCTRL_INTERRUPT_ADDR_1_REG
Level 2	2	PIDCTRL_INTERRUPT_ADDR_2_REG
Level 3	3	PIDCTRL_INTERRUPT_ADDR_3_REG
Level 4	4	PIDCTRL_INTERRUPT_ADDR_4_REG
Level 5	5	PIDCTRL_INTERRUPT_ADDR_5_REG
Level 6 (Debug)	6	PIDCTRL_INTERRUPT_ADDR_6_REG
NMI	7	PIDCTRL_INTERRUPT_ADDR_7_REG

27.3.2 Information Recording

When PID Controller identifies an interrupt, it records three items of information in addition to switching PID to 0. The recorded information includes the priority level of current interrupt, previous interrupt status of the system and the previous process running on the CPU.

PID Controller records the priority level of the current interrupt in register PIDCTRL_LEVEL_REG. For details please refer to Table 115.

Table 115: Configuration of PIDCTRL_LEVEL_REG

Value	Priority level of the current interrupt
0	No interrupt
1	Level 1
2	Level 2
3	Level 3
4	Level 4
5	Level 5
6	Level 6
7	NMI

PID Controller also records in register PIDCTRL_FROM_*n*_REG the status of the system before the interrupt occurred. The bit width of register PIDCTRL_FROM_*n*_REG is 7. The highest four bits represent the interrupt

status of the system before the interrupt indicated by the register occurred. The lowest three bits represent the process running on the CPU before the interrupt indicated by the register occurred. For details please refer to Table 116.

Table 116: Configuration of PIDCTRL_FROM_*n*_REG

[6:3]	Previous interrupt	[2:0]	Previous process
0	No interrupt	0	Process with PID of 0
1	Level 1 Interrupt	1	Process with PID of 1
2	Level 2 Interrupt	2	Process with PID of 2
3	Level 3 Interrupt	3	Process with PID of 3
4	Level 4 Interrupt	4	Process with PID of 4
5	Level 5 Interrupt	5	Process with PID of 5
6	Level 6 Interrupt	6	Process with PID of 6
7	Level 7 Interrupt	7	Process with PID of 7

PID Controller possesses registers PIDCTRL_FROM_1_REG ~ PIDCTRL_FROM_7_REG, which correspond to the interrupts of Level 1, Level 2, Level 3, Level 4, Level 5, Level 6 (Debug), and NMI respectively. This enables the system to implement interrupt nesting. Please refer to Table 125 for examples.

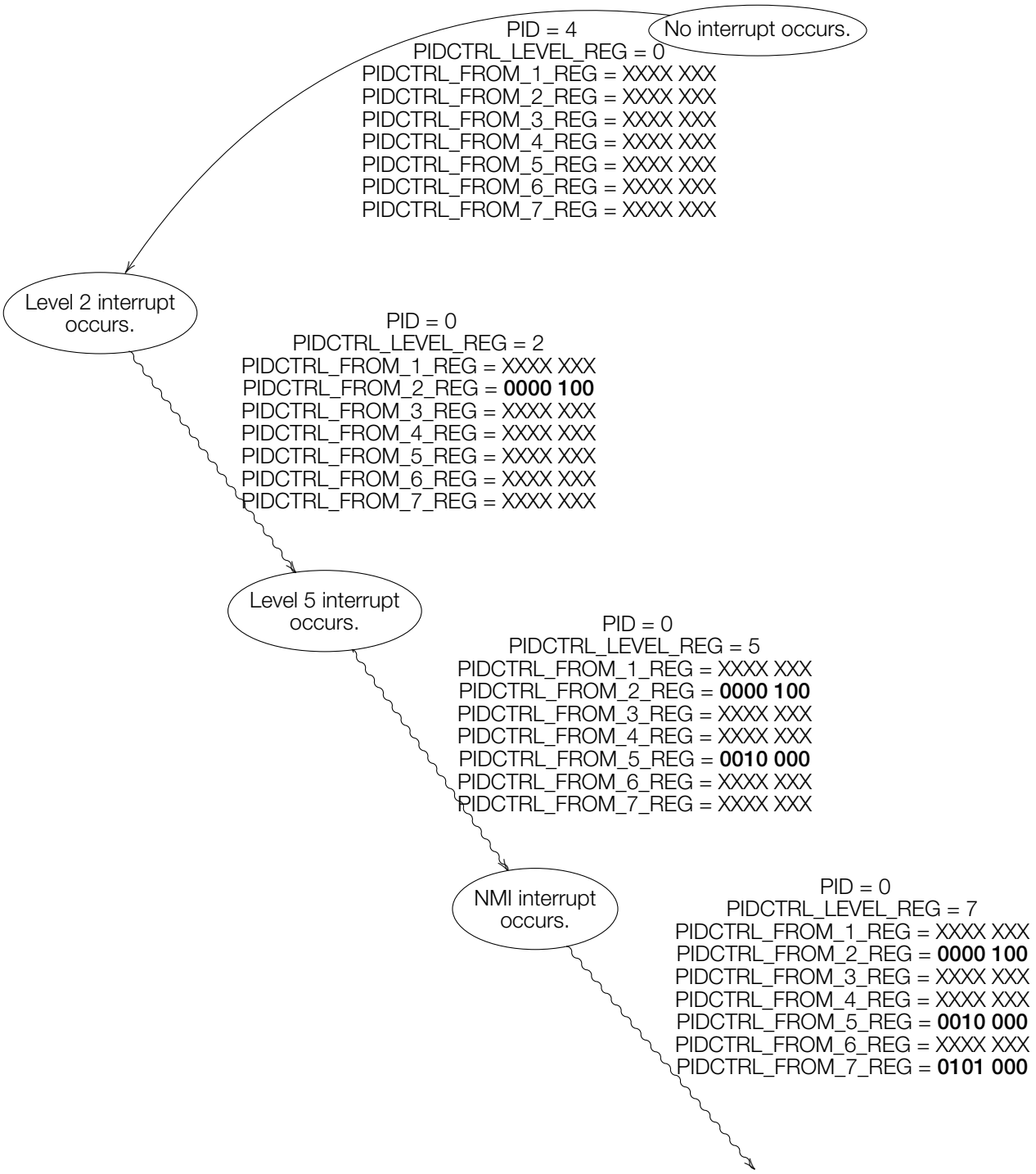


Figure 125: Interrupt Nesting

If the configuration of register PIDCTRL_INTERRUPT_ENABLE_REG prevents PID Controller from identifying an interrupt, PID Controller will not record any information, and PIDCTRL_LEVEL_REG and PIDCTRL_FROM_*n*_REG will remain unchanged.

27.3.3 Proactive Process Switching

As mentioned before, only an elevated process with PID of 0/1 can initiate a process switch. The new process may have any PID from 0 ~ 7 after the process switch. The key for successful proactive process switching is that when the last command of the current process switches to the first command of the new process, PID should

switch from 0/1 to that of the new process.

The software procedure for proactive process switching is as follows:

1. Mask all the interrupts except NMI by using software.
2. Set register PIDCTRL_NMI_MASK_ENABLE_REG to 1 to generate a CPU NMI Interrupt Mask signal.
3. Configure registers PIDCTRL_PID_DELAY_REG and PIDCTRL_NMI_DELAY_REG.
4. Configure register PIDCTRL_PID_NEW_REG.
5. Configure register PIDCTRL_LEVEL_REG and PIDCTRL_FROM_*n*_REG.
6. Set register PIDCTRL_PID_CONFIRM_REG and register PIDCTRL_NMI_MASK_DISABLE_REG to 1.
7. Revoke the masking of all interrupts but NMI.
8. Switch to the new process and fetch instruction.

Though we can deal with interrupt nesting, an elevated process should not be interrupted during the process switching, and therefore the interrupts have been masked in step 1 and step 2.

In step 3, the configured values of registers PIDCTRL_PID_DELAY_REG and PIDCTRL_NMI_DELAY_REG will affect step 6.

In step 4, the configured value of register PIDCTRL_PID_NEW_REG will be the new PID after step 6.

If the system is currently in a nested interrupt and needs to revert to the previous interrupt, register PIDCTRL_LEVEL_REG must be restored based on the information recorded in register PIDCTRL_FROM_*n*_REG in step 5.

In step 6, after the values of register PIDCTRL_PID_CONFIRM_REG and register PIDCTRL_NMI_MASK_DISABLE_REG are set to 1, PID Controller will not immediately switch PID to the value of register PIDCTRL_PID_NEW_REG, nor disable CPU NMI Interrupt Mask signal at once. Instead, PID Controller performs each task after a different number of clock cycles. The numbers of clock cycles are the values specified in register PIDCTRL_PID_DELAY_REG and PIDCTRL_NMI_DELAY_REG respectively.

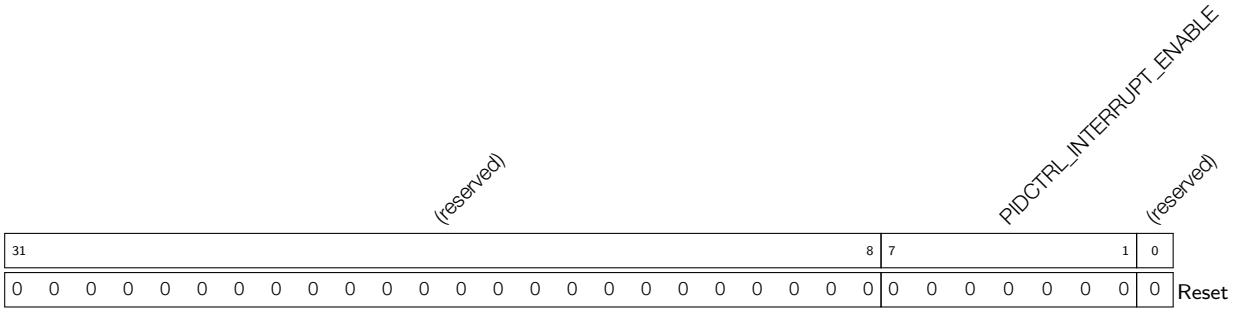
In step 7, other tasks can be implemented as well. To do this, the cost of those tasks should be included when configuring registers PIDCTRL_PID_DELAY_REG and PIDCTRL_NMI_DELAY_REG in step 3.

27.4 Register Summary

Name	Description	Address	Access
PIDCTRL_INTERRUPT_ENABLE_REG	PID interrupt identification enable	0x3FF1F000	R/W
PIDCTRL_INTERRUPT_ADDR_1_REG	Level 1 interrupt vector address	0x3FF1F004	R/W
PIDCTRL_INTERRUPT_ADDR_2_REG	Level 2 interrupt vector address	0x3FF1F008	R/W
PIDCTRL_INTERRUPT_ADDR_3_REG	Level 3 interrupt vector address	0x3FF1F00C	R/W
PIDCTRL_INTERRUPT_ADDR_4_REG	Level 4 interrupt vector address	0x3FF1F010	R/W
PIDCTRL_INTERRUPT_ADDR_5_REG	Level 5 interrupt vector address	0x3FF1F014	R/W
PIDCTRL_INTERRUPT_ADDR_6_REG	Level 6 interrupt vector address	0x3FF1F018	R/W
PIDCTRL_INTERRUPT_ADDR_7_REG	NMI interrupt vector address	0x3FF1F01C	R/W
PIDCTRL_PID_DELAY_REG	New PID valid delay	0x3FF1F020	R/W
PIDCTRL_NMI_DELAY_REG	NMI mask signal disable delay	0x3FF1F024	R/W
PIDCTRL_LEVEL_REG	Current interrupt priority	0x3FF1F028	R/W
PIDCTRL_FROM_1_REG	System status before Level 1 interrupt	0x3FF1F02C	R/W
PIDCTRL_FROM_2_REG	System status before Level 2 interrupt	0x3FF1F030	R/W
PIDCTRL_FROM_3_REG	System status before Level 3 interrupt	0x3FF1F034	R/W
PIDCTRL_FROM_4_REG	System status before Level 4 interrupt	0x3FF1F038	R/W
PIDCTRL_FROM_5_REG	System status before Level 5 interrupt	0x3FF1F03C	R/W
PIDCTRL_FROM_6_REG	System status before Level 6 interrupt	0x3FF1F040	R/W
PIDCTRL_FROM_7_REG	System status before NMI	0x3FF1F044	R/W
PIDCTRL_PID_NEW_REG	New PID configuration register	0x3FF1F048	R/W
PIDCTRL_PID_CONFIRM_REG	New PID confirmation register	0x3FF1F04C	WO
PIDCTRL_NMI_MASK_ENABLE_REG	NMI mask enable register	0x3FF1F054	WO
PIDCTRL_NMI_MASK_DISABLE_REG	NMI mask disable register	0x3FF1F058	WO

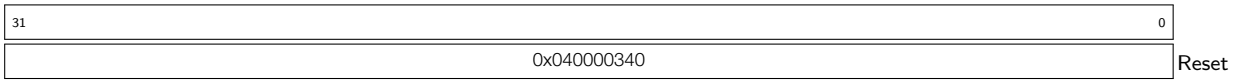
27.5 Registers

Register 27.1: PIDCTRL_INTERRUPT_ENABLE_REG (0x000)



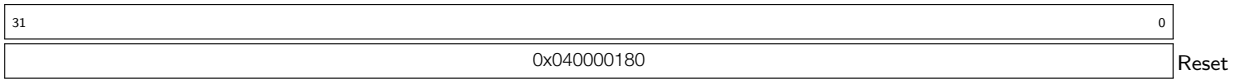
PIDCTRL_INTERRUPT_ENABLE These bits are used to enable interrupt identification and processing. (R/W)

Register 27.2: PIDCTRL_INTERRUPT_ADDR_1_REG (0x004)



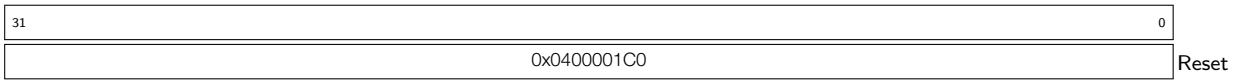
PIDCTRL_INTERRUPT_ADDR_1_REG Level 1 interrupt vector entry address. (R/W)

Register 27.3: PIDCTRL_INTERRUPT_ADDR_2_REG (0x008)



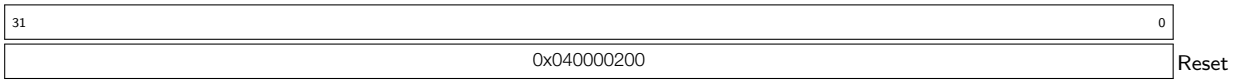
PIDCTRL_INTERRUPT_ADDR_2_REG Level 2 interrupt vector entry address. (R/W)

Register 27.4: PIDCTRL_INTERRUPT_ADDR_3_REG (0x00C)



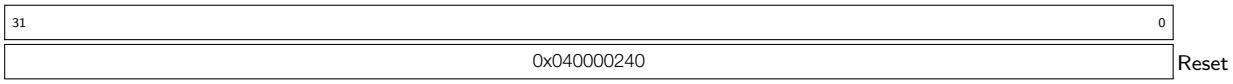
PIDCTRL_INTERRUPT_ADDR_3_REG Level 3 interrupt vector entry address. (R/W)

Register 27.5: PIDCTRL_INTERRUPT_ADDR_4_REG (0x010)



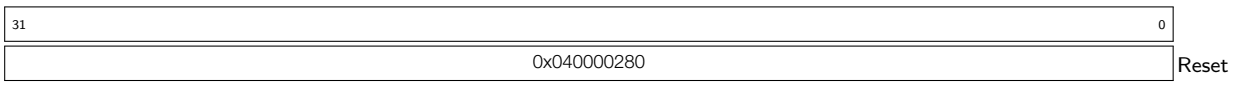
PIDCTRL_INTERRUPT_ADDR_4_REG Level 4 interrupt vector entry address. (R/W)

Register 27.6: PIDCTRL_INTERRUPT_ADDR_5_REG (0x014)



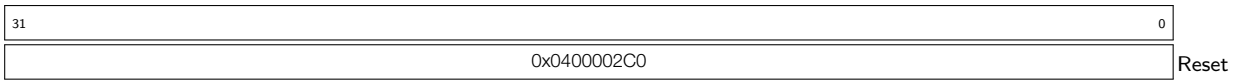
PIDCTRL_INTERRUPT_ADDR_5_REG Level 5 interrupt vector entry address. (R/W)

Register 27.7: PIDCTRL_INTERRUPT_ADDR_6_REG (0x018)



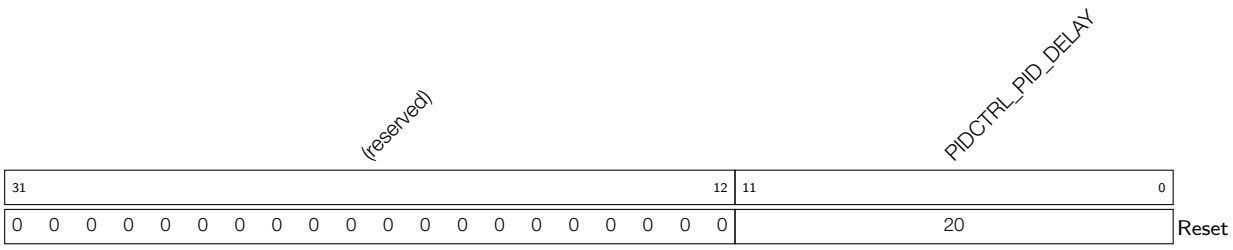
PIDCTRL_INTERRUPT_ADDR_6_REG Level 6 interrupt vector entry address. (R/W)

Register 27.8: PIDCTRL_INTERRUPT_ADDR_7_REG (0x01C)



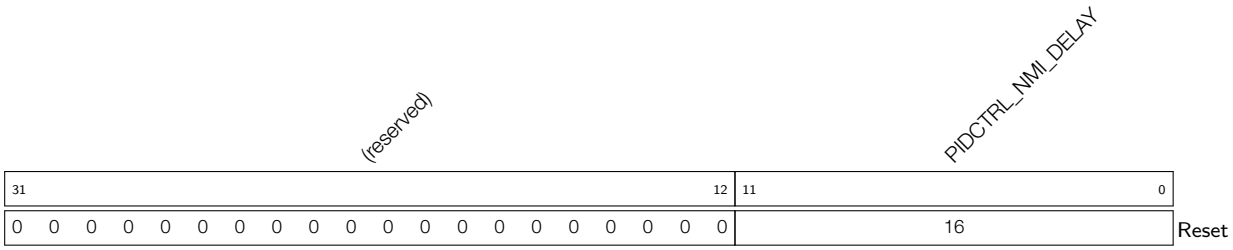
PIDCTRL_INTERRUPT_ADDR_7_REG NMI interrupt vector entry address. (R/W)

Register 27.9: PIDCTRL_PID_DELAY_REG (0x020)



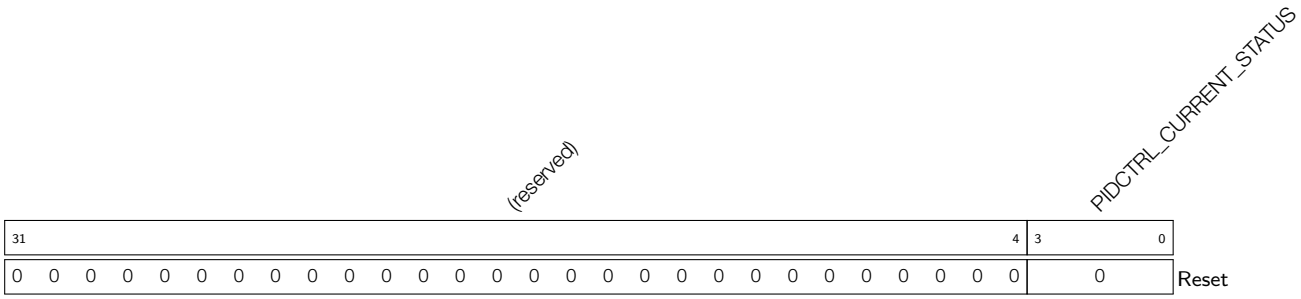
PIDCTRL_PID_DELAY Delay until newly assigned PID is valid. (R/W)

Register 27.10: PIDCTRL_NMI_DELAY_REG (0x024)



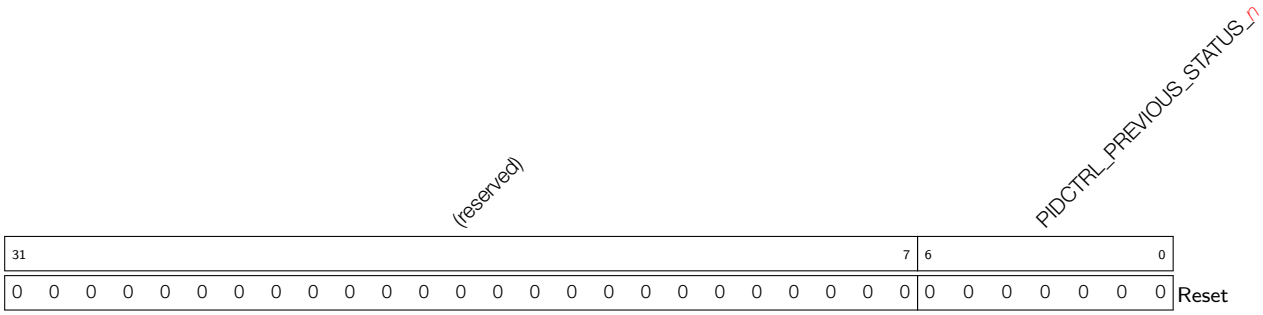
PIDCTRL_NMI_DELAY Delay for disabling CPU NMI interrupt mask signal. (R/W)

Register 27.11: PIDCTRL_LEVEL_REG (0x028)



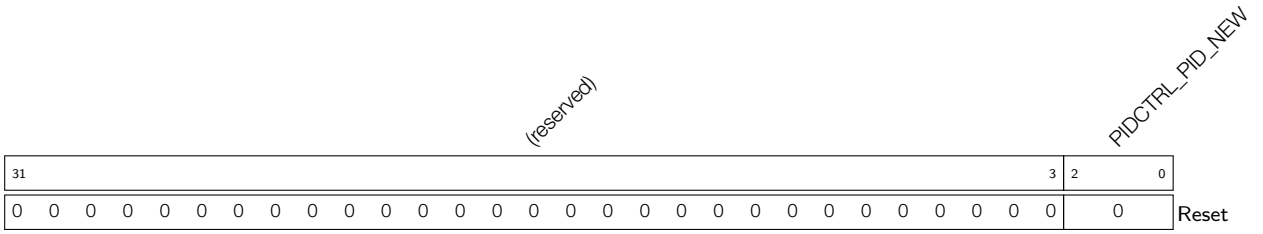
PIDCTRL_CURRENT_STATUS The current status of the system. (R/W)

Register 27.12: PIDCTRL_FROM_*n*_REG (*n*: 1-7) (0x28+0x4n*)**



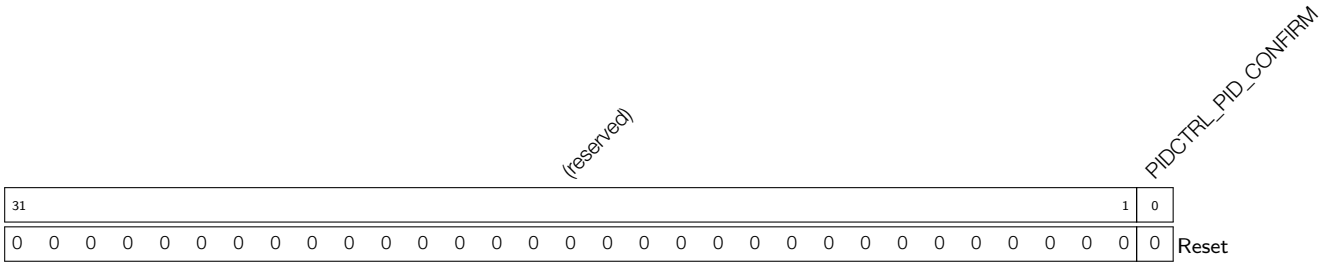
PIDCTRL_PREVIOUS_STATUS_*n* System status before any of Level 1 to Level 6, NMI interrupts occurs. (R/W)

Register 27.13: PIDCTRL_PID_NEW_REG (0x048)



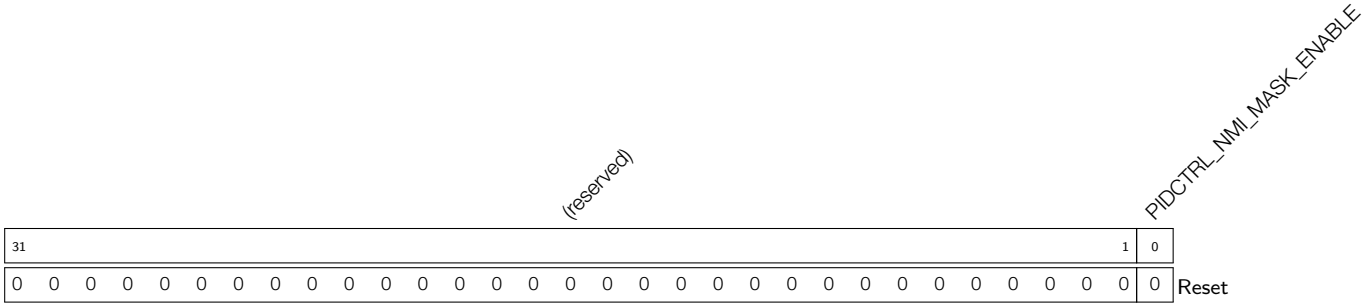
PIDCTRL_PID_NEW New PID. (R/W)

Register 27.14: PIDCTRL_PID_CONFIRM_REG (0x04C)



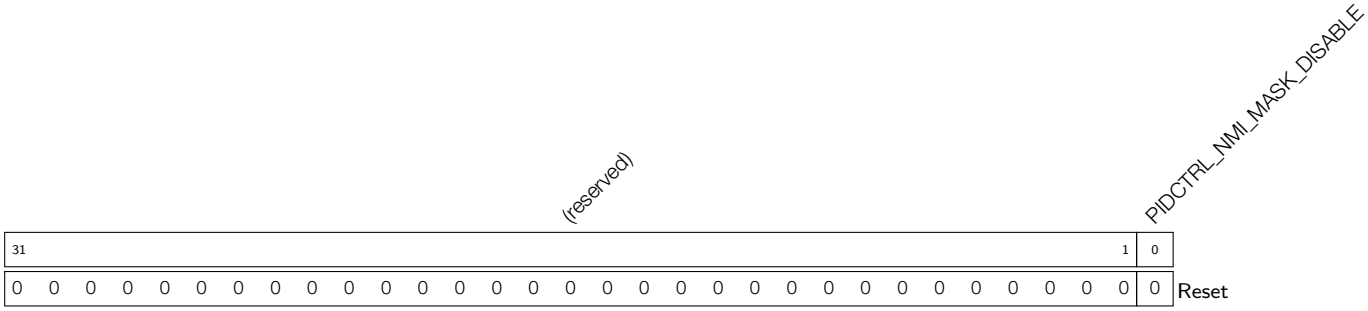
PIDCTRL_PID_CONFIRM This bit is used to confirm the switch of PID. (WO)

Register 27.15: PIDCTRL_NMI_MASK_ENABLE_REG (0x054)



PIDCTRL_NMI_MASK_ENABLE This bit is used to enable CPU NMI interrupt mask signal. (WO)

Register 27.16: PIDCTRL_NMI_MASK_DISABLE_REG (0x058)



PIDCTRL_NMI_MASK_DISABLE This bit is used to disable CPU NMI interrupt mask signal. (WO)

28. On-Chip Sensors and Analog Signal Processing

28.1 Introduction

ESP32 has three types of built-in sensors for various applications: a capacitive touch sensor with up to 10 inputs, and a Hall effect sensor.

The processing of analog signals is done by two successive approximation ADCs (SAR ADC). There are five controllers dedicated to operating ADCs. This provides flexibility when it comes to converting analog inputs in both high-performance and low-power modes, with minimum processor overhead.

ESP32 is also capable of generating analog signals, using two independent DACs and a cosine waveform generator.

28.2 Capacitive Touch Sensor

28.2.1 Introduction

A touch-sensor system is built on a substrate which carries electrodes and relevant connections under a protective flat surface; see Figure 126. When a user touches the surface, the capacitance variation is triggered and a binary signal is generated to indicate whether the touch is valid.

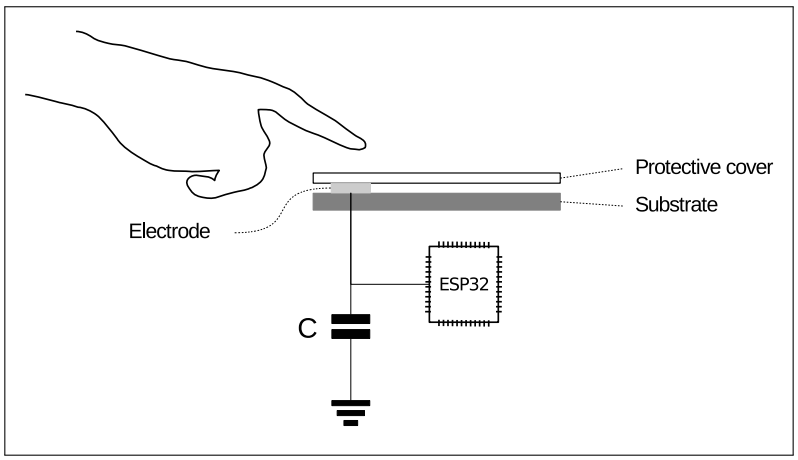


Figure 126: Touch Sensor

28.2.2 Features

- Up to 10 capacitive touch pads / GPIOs
- The sensing pads can be arranged in different combinations, so that a larger area or more points can be detected.
- The touch pad sensing process is under the control of a hardware-implemented finite-state machine (FSM) which is initiated by software or a dedicated hardware timer.
- Information that a pad has been touched can be obtained:
  - by checking touch-sensor registers directly through software,
  - from an interrupt triggered by a touch detection,

  - by waking up the CPU from deep sleep upon touch detection.

- Support for low-power operation in the following scenarios:
  - CPU waiting in deep sleep and saving power until touch detection and subsequent wake up
  - Touch detection managed by the ULP coprocessor
    The user program in ULP coprocessor can trigger a scanning process by checking and writing into specific registers, in order to verify whether the touch threshold is reached.

28.2.3 Available GPIOs

All 10 available sensing GPIOs (pads) are listed in Table 118.

Table 118: ESP32 Capacitive Sensing Touch Pads

Touch Sensing Signal Name	Pin Name
T0	GPIO4
T1	GPIO0
T2	GPIO2
T3	MTDO
T4	MTCK
T5	MTDI
T6	MTMS
T7	GPIO27
T8	32K_XN
T9	32K_XP

28.2.4 Functional Description

The internal structure of the touch sensor is shown in Figure 127. The operating flow is shown in Figure 128.

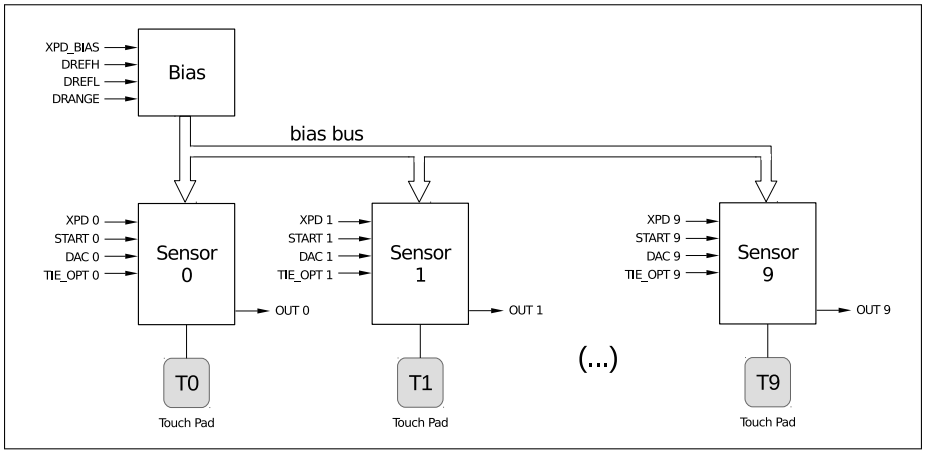


Figure 127: Touch Sensor Structure

The capacitance of a touch pad is periodically charged and discharged. The chart ”Pad Voltage” shows the charge/discharge voltage that swings from DREFH (reference voltage high) to DREFL (reference voltage low). During each swing, the touch sensor generates an output pulse, shown in the chart as ”OUT”. The swing slope is different when the pad is touched (high capacitance) and when it is not (low capacitance). By comparing the

difference between the output pulse counts during the same time interval, we can conclude whether the touch pad has been touched. TIE_OPT is used to establish the initial voltage level that starts the charge/discharge cycle.

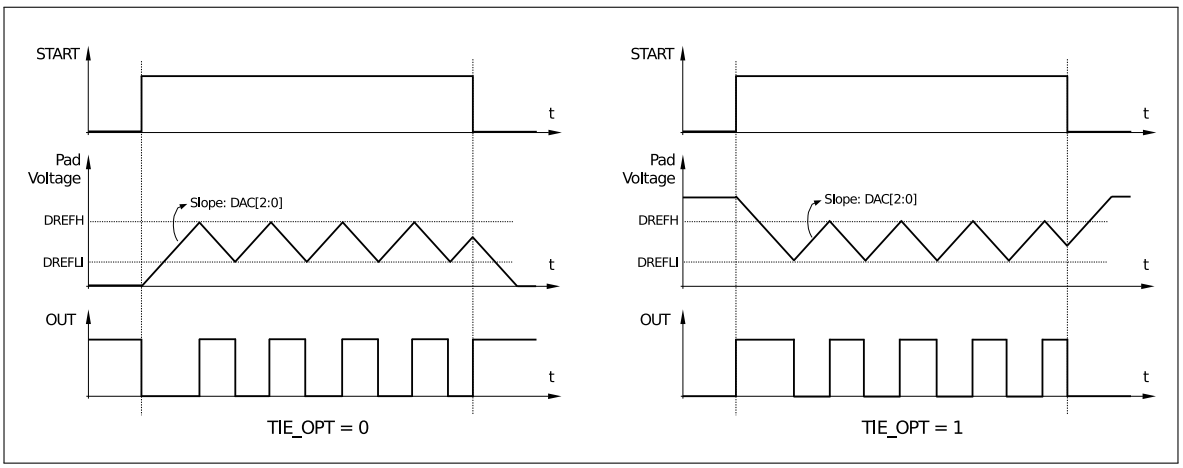


Figure 128: Touch Sensor Operating Flow

28.2.5 Touch FSM

The Touch FSM performs a measurement sequence described in section 28.2.4. Software can operate the Touch FSM through dedicated registers. The internal structure of a touch FSM is shown in Figure 129.

The functions of Touch FSM include:

- Receipt of a start signal, either from software or a timer
  - when SENS_SAR_TOUCH_START_FORCE=1, SENS_SAR_TOUCH_START_EN is used to initiate a single measurement
  - when SENS_SAR_TOUCH_START_FORCE=0, measurement is triggered periodically with a timer.

  The Touch FSM can be active in sleep mode. The SENS_SAR_TOUCH_SLEEP_CYCLES register can be used to set the cycles. The sensor is operated by FAST_CLK, which normally runs at 8 MHz. More information on that can be found in chapter Reset and Clock.
- Generation of XPD_TOUCH_BIAS / TOUCH_XPD / TOUCH_START with adjustable timing sequence
  To select enabled pads, TOUCH_XPD / TOUCH_START is masked by the 10-bit register SENS_SAR_TOUCH_PAD_WORKEN.
- Counting of pulses on TOUCH0_OUT ~ TOUCH9_OUT
  The result can be read from SENS_SAR_TOUCH_MEAS_OUT*n*. All ten touch sensors can work simultaneously.
- Generation of a wakeup interrupt
  The FSM regards the touch pads as “touched”, if the number of counted pulses is below the threshold. The 10-bit registers SENS_TOUCH_PAD_OUTEN1 & SENS_TOUCH_PAD_OUTEN2 define two sets of touch pads, i.e. SET1 & SET2. If at least one of the pads in SET1 is “touched”, the wakeup interrupt will be generated by default. It is also possible to configure the wakeup interrupt to be generated only when pads from both sets are “touched”.

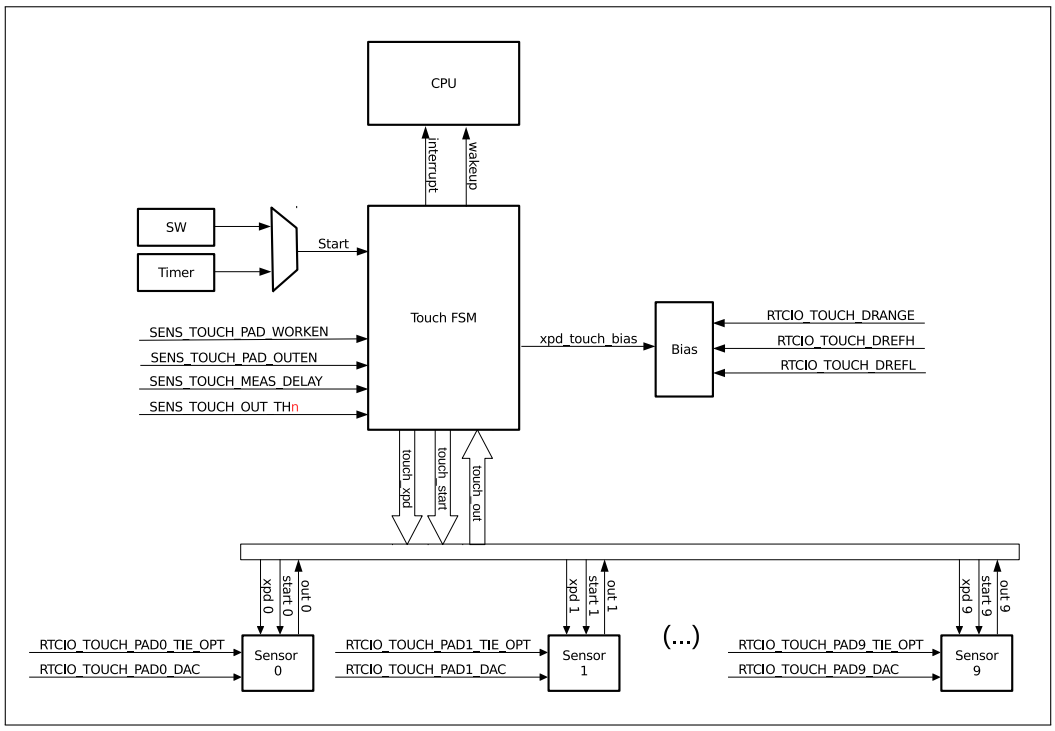


Figure 129: Touch FSM Structure

28.3 SAR ADC

28.3.1 Introduction

ESP32 integrates two 12-bit SAR ADCs. They are managed by five SAR ADC controllers, and are able to measure signals from one to 18 analog pads. It is also possible to measure internal signals, such as vdd33. Some of the pads can be used to build a programmable gain-amplifier which measures small analog signals.

The SAR ADC controllers have specialized uses. Two of them support high-performance multiple-channel scanning. Another two are used for low-power operation during deep sleep, and the last one is dedicated to PWDET / PKDET (power and peak detection). A diagram of the SAR ADCs is shown in Figure 130.

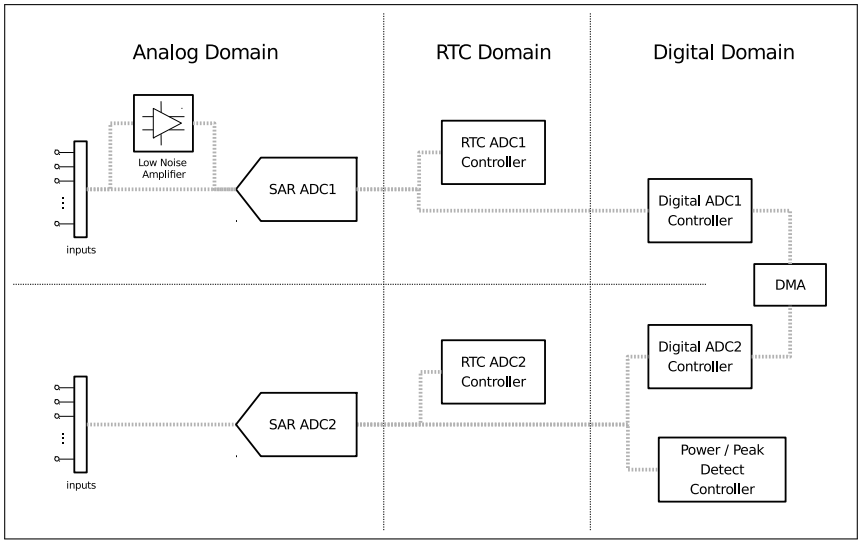


Figure 130: SAR ADC Depiction

28.3.2 Features

- Two SAR ADCs, with simultaneous sampling and conversion
- Up to five SAR ADC controllers for different purposes (e.g. high performance, low power or PWDET / PKDET).
- Up to 18 analog input pads
- One channel for internal voltage vdd33, two for pa_pkdet (available on selected controllers)
- 12-bit, 11-bit, 10-bit, 9-bit configurable resolution
- DMA support (available on one controller)
- Multiple channel-scanning modes (available on two controllers)
- Operation during deep sleep (available on one controller)
- Controlled by a ULP coprocessor (available on two controllers)

28.3.3 Outline of Function

The SAR ADC module's major components, and their interconnections, are shown in Figure 131.

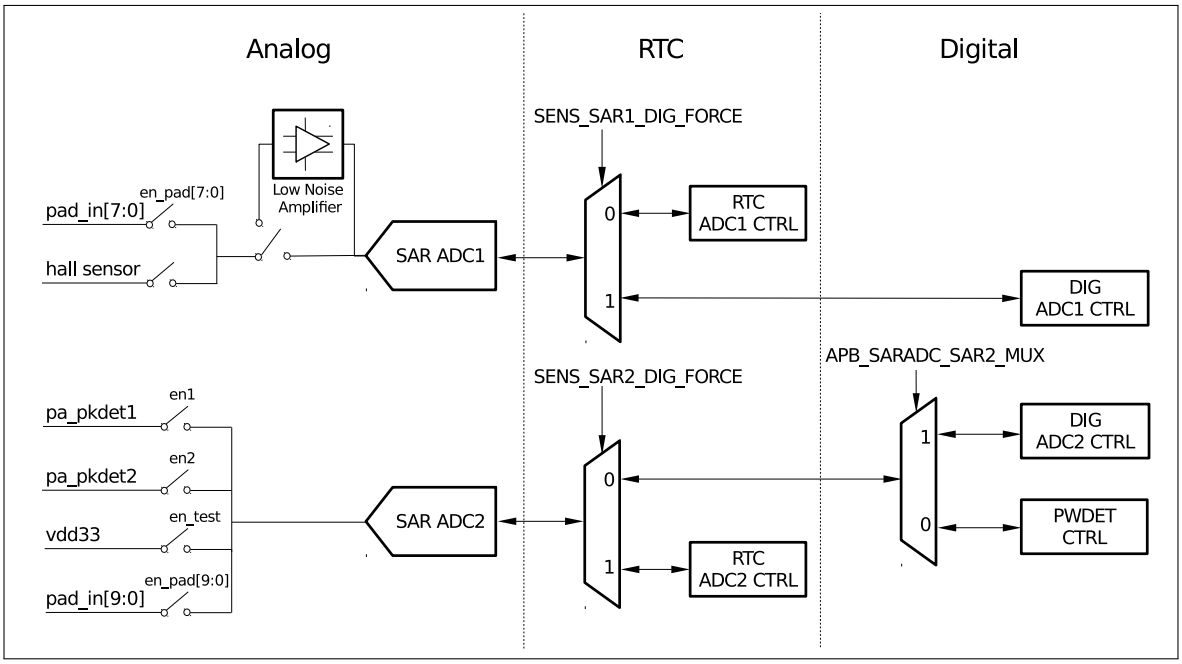


Figure 131: SAR ADC Outline of Function

A summary of all the analog signals that may be sent to the SAR ADC module for processing by either ADC1 or ADC2 is presented in Table 119.

Table 119: Inputs of SAR ADC module

Signal Name	Pad #	Processed by
VDET_2	7	SAR ADC1
VDET_1	6	
32K_XN	5	
32K_XP	4	

Signal Name	Pad #	Processed by
SENSOR_VN	3	
SENSOR_CAPN	2	
SENSOR_CAPP	1	
SENSOR_VP	0	
Hall sensor	n/a	
GPIO26	9	SAR ADC2
GPIO25	8	
GPIO27	7	
MTMS	6	
MTDI	5	
MTCK	4	
MTDO	3	
GPIO2	2	
GPIO0	1	
GPIO4	0	
pa_pkdet1	n/a	
pa_pkdet2	n/a	
vdd33	n/a	

There are five ADC controllers in ESP32: RTC ADC1 CTRL, RTC ADC2 CTRL, DIG ADC1 CTRL, DIG ADC2 CTRL and PWDET CTRL. The differences between them are summarized in Table 120.

Table 120: ESP32 SAR ADC Controllers

	RTC ADC1	RTC ADC2	DIG ADC1	DIG ADC2	PWDET
DAC	Y	-	-	-	-
Support deep sleep	Y	Y	-	-	-
ULP coprocessor	Y	Y	-	-	-
vdd33	-	Y	-	Y	-
PWDET/PKDET	-	-	-	-	Y
Hall sensor	Y	-	-	-	-
DMA	-	-	Y	-	-

28.3.4 RTC SAR ADC Controllers

The purpose of SAR ADC controllers in the RTC power domain – RTC ADC1 CTRL and RTC ADC2 CTRL – is to provide ADC measurement with minimal power consumption in a low frequency.

The outline of a single controller's function is shown in Figure 132. For each controller, the start of analog-to-digital conversion can be triggered by register SENS_SAR_MEAS*n*_START_SAR. The measurement's result can be obtained from register SENS_SAR_MEAS*n*_DATA_SAR.

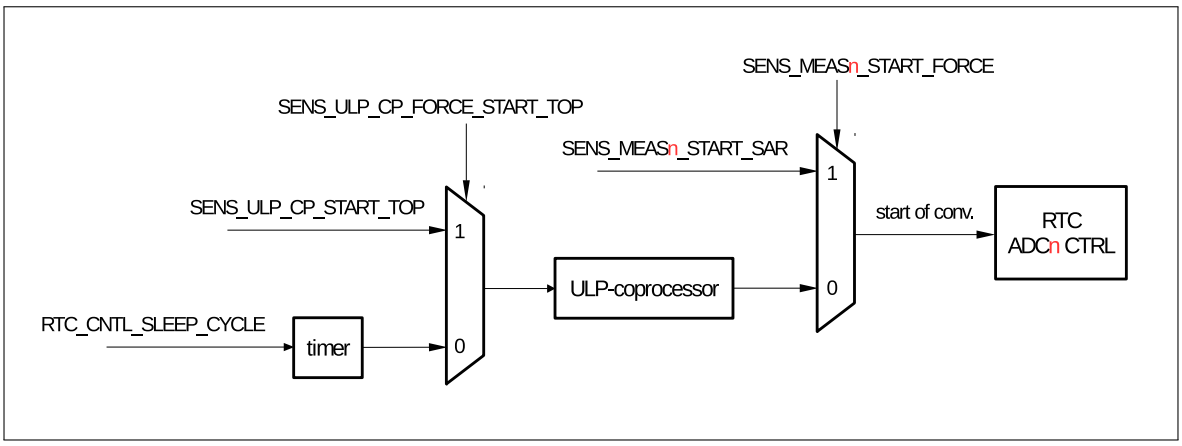


Figure 132: RTC SAR ADC Outline of Function

The controllers are intertwined with the ULP coprocessor, as the ULP coprocessor has a built-in instruction to start an ADC measurement. In many cases, the controllers need to cooperate with the ULP coprocessor, e.g.:

- when periodically monitoring a channel during deep sleep, where the ULP coprocessor is the only trigger source during this mode;
- when scanning channels continuously in a sequence. Continuous scanning or DMA is not supported by the controllers. However, it is possible with the help of the ULP coprocessor.

28.3.5 DIG SAR ADC Controllers

Compared to RTC SAR ADC controllers, DIG SAR ADC controllers have optimized performance and throughput. Some of their features are:

- High performance; the clock is much faster, therefore, the sample rate is highly increased.
- Multiple-channel scanning mode; there is a pattern table that defines the measurement rule for each SAR ADC. The scanning mode can be configured as a single mode, double mode, or alternate mode.
- The scanning can be started by software or I2S.
- DMA support; an interrupt will be generated when scanning is finished.

Note:
We do not use the term “start of conversion” in this section, because there is no direct access to starting a single SAR analog-to-digital conversion. We use “start of scan” instead, which implies that we expect to scan a sequence of channels with DIG ADC controllers.

Figure 133 shows a diagram of DIG SAR ADC controllers.

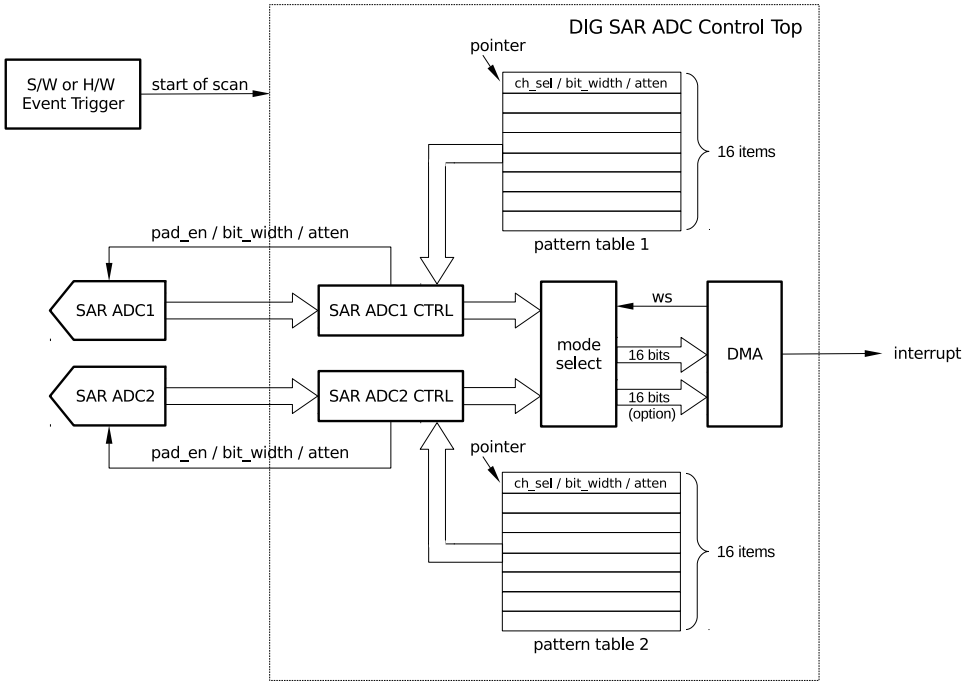


Figure 133: Diagram of DIG SAR ADC Controllers

The pattern tables contain the measurement rules mentioned above. Each table has 16 items which store information on channel selection, resolution and attenuation. When scanning starts, the controller reads measurement rules one-by-one from a pattern table. For each controller the scanning sequence includes 16 different rules at most, before repeating itself.

The 8-bit item (the pattern table register) is composed of three fields that contain channel, resolution and attenuation information, as shown in Table 121.

Table 121: Fields of the Pattern Table Register

Pattern Table Register [7:0]		
ch_sel[3:0]	bit_width[1:0]	atten[1:0]
channel to be scanned	resolution	attenuation

There are three scanning modes: single mode, double mode and alternate mode.

- Single mode: channels of either SAR ADC1 or SAR ADC2 will be scanned.
- Double mode: channels of SAR ADC1 and SAR ADC2 will be scanned simultaneously.
- Alternate mode: channels of SAR ADC1 and SAR ADC2 will be scanned alternately.

ESP32 supports up to a 12-bit SAR ADC resolution. The 16-bit data in DMA is composed of the ADC result and some necessary information related to the scanning mode:

- For single mode, only 4-bit information on channel selection is added.
- For double mode or alternate mode, 4-bit information on channel selection is added plus one extra bit indicating which SAR ADC was selected.

For each scanning mode there is a corresponding data format, called Type I and Type II. Both data formats are described in Tables 122 and 123.

Table 122: Fields of Type I DMA Data Format

Type I DMA Data Format [15:0]	
ch_sel[3:0]	data[11:0]
channel	SAR ADC data

Table 123: Fields of Type II DMA Data Format

Type II DMA Data Format [15:0]		
sar_sel	ch_sel[3:0]	SAR ADC data[10:0]
SAR ADCn	channel	SAR ADC data

For Type I the resolution of SAR ADC is up to 12 bits, while for Type II the resolution is 11 bits at most.

DIG SAR ADC Controllers allow the use of I2S for direct memory access. The WS signal of I2S acts as a measurement-trigger signal. The DATA signal provides the information that the measurement result is ready. Software can configure APB_SARADC_DATA_TO_I2S, in order to connect ADC to I2S.

28.4 Hall Sensor

28.4.1 Introduction

The Hall effect is the generation of a voltage difference across an n-type semiconductor passing electrical current, when a magnetic field is applied to it in a direction perpendicular to that of the flow of the current. The voltage is proportional to the product of the magnetic field's strength and current value. A Hall-effect sensor could be used to measure the strength of a magnetic field, when constant current flows through it, or when the current is in the presence of a constant magnetic field. As the heart of many applications, the Hall-effect sensors provide proximity detection, positioning, speed measurement, and current sensing.

Inside of ESP32 there is a Hall sensor for magnetic field-sensing applications, which is designed to feed voltage signals to the SAR ADC. It can be controlled by the ULP coprocessor, when low-power operation is required. Such functionality, which enhances the power-processing and flexibility of ESP32, makes it an attractive solution for position sensing, proximity detection, speed measurement, etc.

28.4.2 Features

- Built-in Hall element with amplifier
- Designed to operate with ADC
- Capable of outputting both analog voltage and digital signals related to the strength of the magnetic field
- Powerful and easy-to-implement functionality, due to its integration with built-in ULP coprocessor, GPIOs, CPU, Wi-Fi, etc.

28.4.3 Functional Description

The Hall sensor converts the magnetic field into voltage, feeds it into an amplifier, and then outputs it through pin SENSOR_VP and pin SENSOR_VN. ESP32's built-in ADC converts the voltage into a digital value for processing by the CPU in the digital domain.

The inner structure of a Hall sensor is shown in Figure 134.

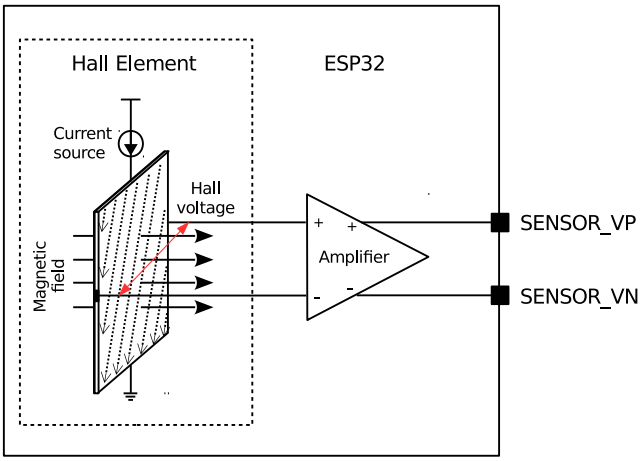


Figure 134: Hall Sensor

The configuration of a Hall sensor for reading is done with registers SENS_SAR_TOUCH_CTRL1_REG and RTCIO_HALL_SENS_REG, which are used to power up the Hall sensor. The subsequent processing is done by SAR ADC1. The result is obtained from the RTC ADC1 controller. For more details, please refer to section 28.3.

28.5 DAC

28.5.1 Introduction

Two 8-bit DAC channels can be used to convert digital values into analog output signals (up to two of them). The design structure is composed of integrated resistor strings and a buffer. This dual DAC supports power supply and uses it as input voltage reference. The dual DAC also supports independent or simultaneous signal conversions inside of its channels.

28.5.2 Features

The features of DAC are as follows:

- Two 8-bit DAC channels
- Independent or simultaneous conversion in channels
- Voltage reference from the VDD3P3_RTC pin
- Cosine waveform (CW) generator
- DMA capability

- Start of conversion can be triggered by software or SAR ADC FSM (please refer to the SAR ADC chapter for more details)
- Can be fully controlled by the ULP coprocessor

A diagram showing the DAC channel's function is presented in Figure 135. For a detailed description, see the sections below.

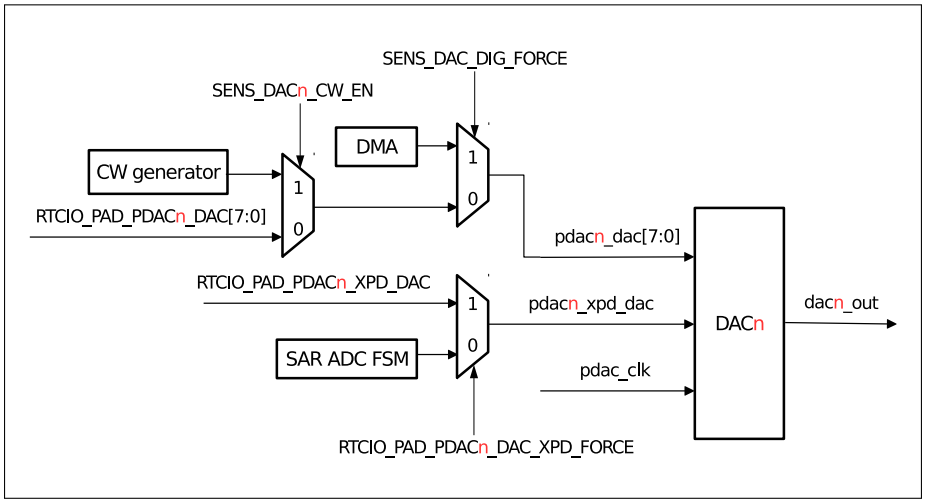


Figure 135: Diagram of DAC Function

28.5.3 Structure

The two 8-bit DAC channels can be configured independently. For each DAC channel, the output analog voltage can be calculated as follows:

$$DAC\textit{n}\_OUT = VDD3P3\_RTC \cdot PDAC\textit{n}\_DAC/256$$

- VDD3P3_RTC is the voltage on pin VDD3P3_RTC (typically 3.3V).
- PDAC*n*_DAC has multiple sources: CW generator, register RTCIO_PAD_DAC*n*_REG, and DMA.

The start of conversion is determined by register RTCIO_PAD_PDAC*n*_XPD_DAC. The conversion process itself is controlled by software or SAR ADC FSM; see Figure 135.

28.5.4 Cosine Waveform Generator

The cosine waveform (CW) generator can be used to generate a cosine / sine tone. A diagram showing cosine waveform generator's function is presented in Figure 136.

The CW generator has the following features:

- Adjustable frequency
  The frequency of CW can be adjusted by register SENS_SAR_SW_FSTEP[15:0]:

  $$freq = dig\_clk\_rtc\_freq \cdot SENS\_SAR\_SW\_FSTEP/65536$$

  The frequency of dig_clk_rtc is typically 8 MHz.
- Scaling
  Configuring register SENS_SAR_DAC_SCALE*n*[1:0]; the amplitude of a CW can be multiplied by 1, 1/2, 1/4 or 1/8.

- DC offset
  The offset may be introduced by register SENS_SAR_DAC_DC*n*[7:0]. The result will be saturated.
- Phase shift
  A phase-shift of 0 / 90 / 180 / 270 degrees can be added by setting register SENS_SAR_DAC_INV*n*[1:0].

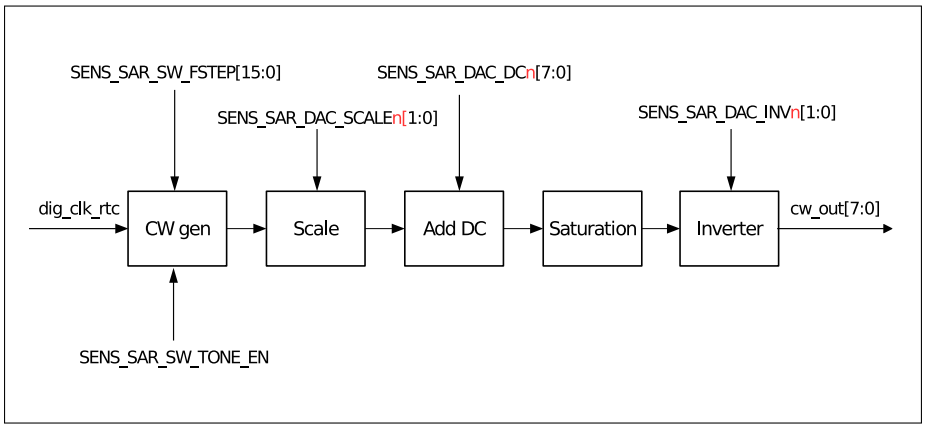


Figure 136: Cosine Waveform (CW) Generator

28.5.5 DMA support

A DMA controller with dual DMA channels can be used to set the output of two DAC channels. By configuring SENS_SAR_DAC_DIG_FORCE, I2S_clk can be connected to DAC clk, and I2S_DATA_OUT can be connected to DAC_DATA for direct memory access.

For details, please refer to chapter DMA.

28.6 Register Summary

Note: The registers listed below have been grouped, according to their functionality. This particular grouping does not reflect the exact sequential order of their place in memory.

28.6.1 Sensors

Name	Description	Address	Access
Touch pad setup and control registers			
SENS_SAR_TOUCH_CTRL1_REG	Touch pad control	0x3FF48858	R/W
SENS_SAR_TOUCH_CTRL2_REG	Touch pad control and status	0x3FF48884	RO
SENS_SAR_TOUCH_ENABLE_REG	Wakeup interrupt control and working set	0x3FF4888C	R/W
SENS_SAR_TOUCH_THRES1_REG	Threshold setup for pads 0 and 1	0x3FF4885C	R/W
SENS_SAR_TOUCH_THRES2_REG	Threshold setup for pads 2 and 3	0x3FF48860	R/W
SENS_SAR_TOUCH_THRES3_REG	Threshold setup for pads 4 and 5	0x3FF48864	R/W
SENS_SAR_TOUCH_THRES4_REG	Threshold setup for pads 6 and 7	0x3FF48868	R/W
SENS_SAR_TOUCH_THRES5_REG	Threshold setup for pads 8 and 9	0x3FF4886C	R/W
SENS_SAR_TOUCH_OUT1_REG	Counters for pads 0 and 1	0x3FF48870	RO
SENS_SAR_TOUCH_OUT2_REG	Counters for pads 2 and 3	0x3FF48874	RO
SENS_SAR_TOUCH_OUT3_REG	Counters for pads 4 and 5	0x3FF48878	RO
SENS_SAR_TOUCH_OUT4_REG	Counters for pads 6 and 6	0x3FF4887C	RO
SENS_SAR_TOUCH_OUT5_REG	Counters for pads 8 and 9	0x3FF48880	RO
SAR ADC control register			
SENS_SAR_START_FORCE_REG	SAR ADC1 and ADC2 control	0x3FF4882C	R/W
SAR ADC1 control registers			
SENS_SAR_READ_CTRL_REG	SAR ADC1 data and sampling control	0x3FF48800	R/W
SENS_SAR_MEAS_START1_REG	SAR ADC1 conversion control and status	0x3FF48854	RO
SAR ADC2 control registers			
SENS_SAR_READ_CTRL2_REG	SAR ADC2 data and sampling control	0x3FF48890	R/W
SENS_SAR_MEAS_START2_REG	SAR ADC2 conversion control and status	0x3FF48894	RO
ULP coprocessor configuration register			
SENS_ULP_CP_SLEEP_CYC0_REG	Sleep cycles for ULP coprocessor	0x3FF48818	R/W
Pad attenuation configuration registers			
SENS_SAR_ATTEN1_REG	2-bit attenuation for each pad	0x3FF48834	R/W
SENS_SAR_ATTEN2_REG	2-bit attenuation for each pad	0x3FF48838	R/W
DAC control registers			
SENS_SAR_DAC_CTRL1_REG	DAC control	0x3FF48898	R/W
SENS_SAR_DAC_CTRL2_REG	DAC output control	0x3FF4889C	R/W

28.6.2 Advanced Peripheral Bus

Name	Description	Address	Access
SAR ADC1 and ADC2 common configuration registers			
APB_SARADC_CTRL_REG	SAR ADC common configuration	0x06002610	R/W
APB_SARADC_CTRL2_REG	SAR ADC common configuration	0x06002614	R/W
APB_SARADC_FSM_REG	SAR ADC FSM sample cycles configuration	0x06002618	R/W

SAR ADC1 pattern table registers			
APB_SARADC_SAR1_PATT_TAB1_REG	Items 0 - 3 of pattern table	0x0600261C	R/W
APB_SARADC_SAR1_PATT_TAB2_REG	Items 4 - 7 of pattern table	0x06002620	R/W
APB_SARADC_SAR1_PATT_TAB3_REG	Items 8 - 11 of pattern table	0x06002624	R/W
APB_SARADC_SAR1_PATT_TAB4_REG	Items 12 - 15 of pattern table	0x06002628	R/W
SAR ADC2 pattern table registers			
APB_SARADC_SAR2_PATT_TAB1_REG	Items 0 - 3 of pattern table	0x0600262C	R/W
APB_SARADC_SAR2_PATT_TAB2_REG	Items 4 - 7 of pattern table	0x06002630	R/W
APB_SARADC_SAR2_PATT_TAB3_REG	Items 8 - 11 of pattern table	0x06002634	R/W
APB_SARADC_SAR2_PATT_TAB4_REG	Items 12 - 15 of pattern table	0x06002638	R/W

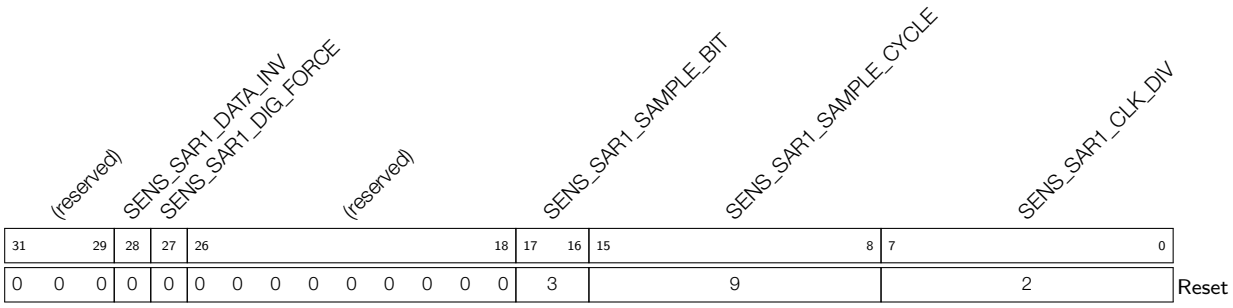
28.6.3 RTC I/O

For details, please refer to Section Register Summary in Chapter IO_MUX and GPIO Matrix.

28.7 Registers

28.7.1 Sensors

Register 28.1: SENS_SAR_READ_CTRL_REG (0x0000)



SENS_SAR1_DATA_INV Invert SAR ADC1 data. (R/W)

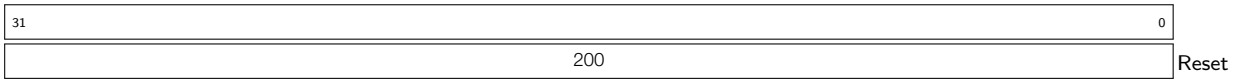
SENS_SAR1_DIG_FORCE 1: SAR ADC1 controlled by DIG ADC1 CTR, 0: SAR ADC1 controlled by RTC ADC1 CTRL. (R/W)

SENS_SAR1_SAMPLE_BIT Bit width of SAR ADC1, 00: for 9-bit, 01: for 10-bit, 10: for 11-bit, 11: for 12-bit. (R/W)

SENS_SAR1_SAMPLE_CYCLE Sample cycles for SAR ADC1. (R/W)

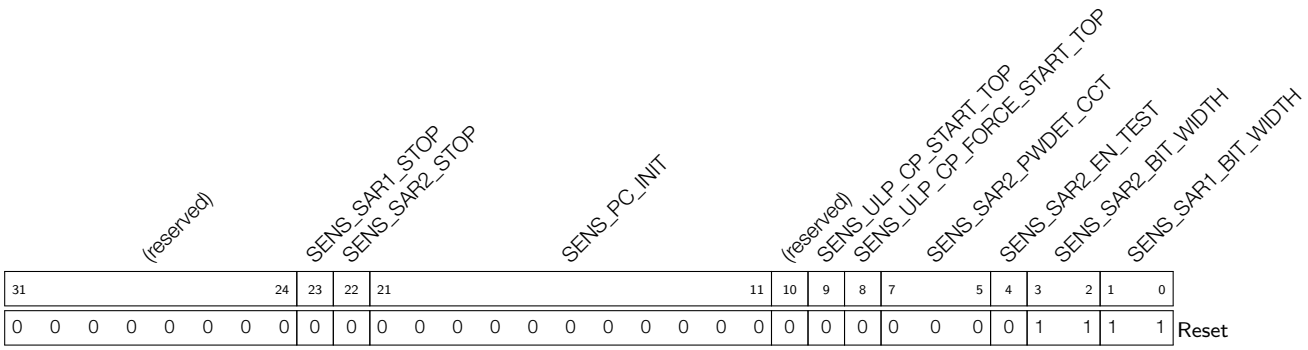
SENS_SAR1_CLK_DIV Clock divider. (R/W)

Register 28.2: SENS_ULP_CP_SLEEP_CYC0_REG (0x0018)



SENS_ULP_CP_SLEEP_CYC0_REG Sleep cycles for ULP coprocessor timer. (R/W)

Register 28.3: SENS_SAR_START_FORCE_REG (0x002c)



SENS_SAR1_STOP Stop SAR ADC1 conversion. (R/W)

SENS_SAR2_STOP Stop SAR ADC2 conversion. (R/W)

SENS_PC_INIT Initialized PC for ULP coprocessor. (R/W)

SENS_ULP_CP_START_TOP Write 1 to start ULP coprocessor; it is active only when reg_ulp_cp_force_start_top = 1. (R/W)

SENS_ULP_CP_FORCE_START_TOP 1: ULP coprocessor is started by SW, 0: ULP coprocessor is started by timer. (R/W)

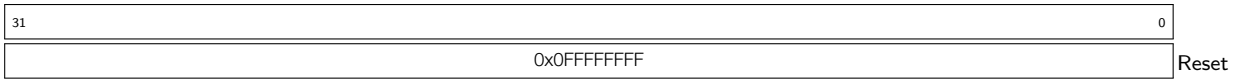
SENS_SAR2_PWDET_CCT SAR2_PWDET_CCT, PA power detector capacitance tuning. (R/W)

SENS_SAR2_EN_TEST SAR2_EN_TEST is active only when reg_sar2_dig_force = 0. (R/W)

SENS_SAR2_BIT_WIDTH Bit width of SAR ADC1, 00: 9 bits, 01: 10 bits, 10: 11 bits, 11: 12 bits. (R/W)

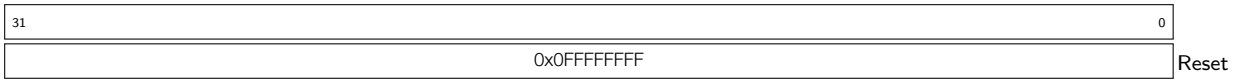
SENS_SAR1_BIT_WIDTH Bit width of SAR ADC2, 00: 9 bits, 01: 10 bits, 10: 11 bits, 11: 12 bits. (R/W)

Register 28.4: SENS_SAR_ATTEN1_REG (0x0034)



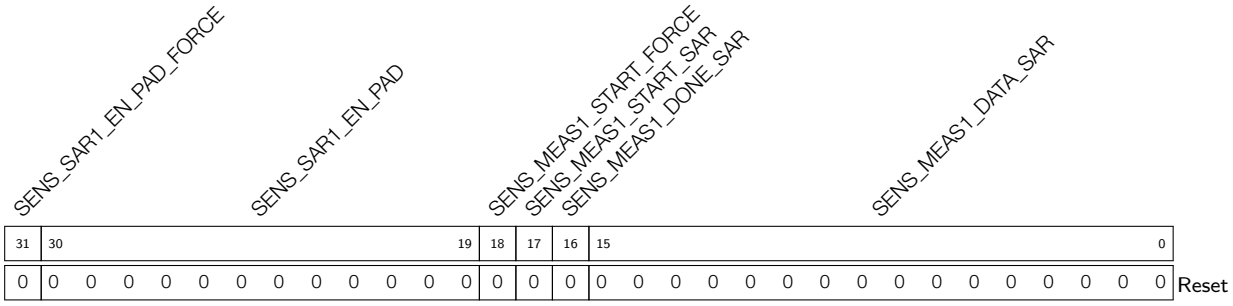
SENS_SAR_ATTEN1_REG 2-bit attenuation for each pad, 11: 1 dB, 10: 6 dB, 01: 3 dB, 00: 0 dB, [1:0] is used for ADC1_CH0, [3:2] is used for ADC1_CH1, etc. (R/W)

Register 28.5: SENS_SAR_ATTEN2_REG (0x0038)



SENS_SAR_ATTEN2_REG 2-bit attenuation for each pad, 11: 1 dB, 10: 6 dB, 01: 3 dB, 00: 0 dB, [1:0] is used for ADC2_CH0, [3:2] is used for ADC2_CH1, etc (R/W)

Register 28.6: SENS_SAR_MEAS_START1_REG (0x0054)



SENS_SAR1_EN_PAD_FORCE 1: SAR ADC1 pad enable bitmap is controlled by SW, 0: SAR ADC1 pad enable bitmap is controlled by ULP coprocessor. (R/W)

SENS_SAR1_EN_PAD SAR ADC1 pad enable bitmap; active only when reg_sar1_en_pad_force = 1. (R/W)

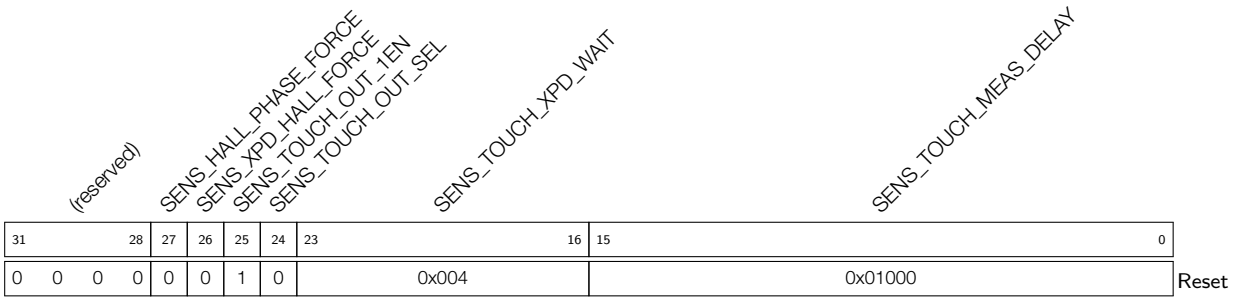
SENS_MEAS1_START_FORCE 1: SAR ADC1 controller (in RTC) is started by SW, 0: SAR ADC1 controller is started by ULP coprocessor. (R/W)

SENS_MEAS1_START_SAR SAR ADC1 controller (in RTC) starts conversion; active only when reg_meas1_start_force = 1. (R/W)

SENS_MEAS1_DONE_SAR SAR ADC1 conversion-done indication. (RO)

SENS_MEAS1_DATA_SAR SAR ADC1 data. (RO)

Register 28.7: SENS_SAR_TOUCH_CTRL1_REG (0x0058)



SENS_HALL_PHASE_FORCE 1: HALL PHASE is controlled by SW, 0: HALL PHASE is controlled by FSM in ULP coprocessor. (R/W)

SENS_XPD_HALL_FORCE 1: XPD HALL is controlled by SW, 0: XPD HALL is controlled by FSM in ULP coprocessor. (R/W)

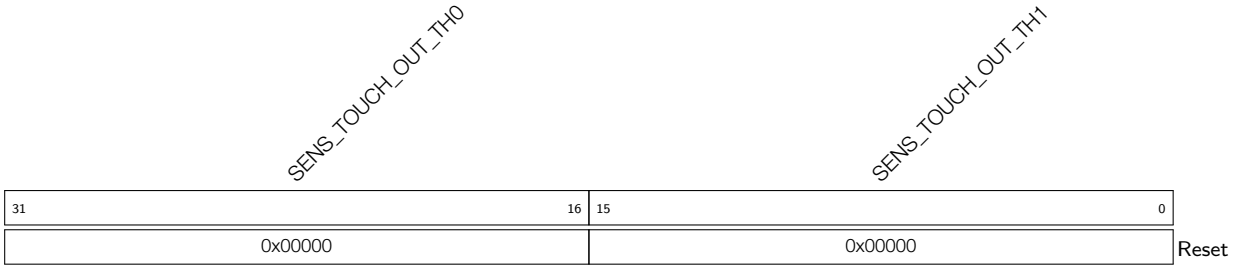
SENS_TOUCH_OUT_1EN 1: wakeup interrupt is generated if SET1 is touched, 0: wakeup interrupt is generated only if both SET1 & SET2 are touched. (R/W)

SENS_TOUCH_OUT_SEL 1: the touch pad is considered touched when the value of the counter is greater than the threshold, 0: the touch pad is considered touched when the value of the counter is less than the threshold. (R/W)

SENS_TOUCH_XPD_WAIT The waiting time (in 8 MHz cycles) between TOUCH_START and TOUCH_XPD. (R/W)

SENS_TOUCH_MEAS_DELAY The measurement's duration (in 8 MHz cycles). (R/W)

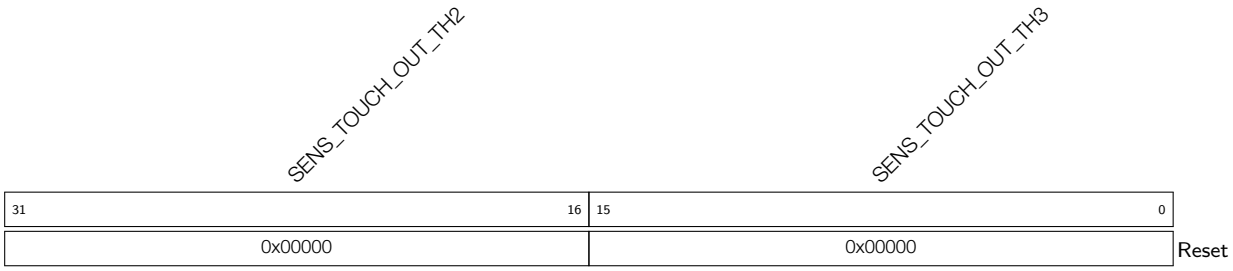
Register 28.8: SENS_SAR_TOUCH_THRES1_REG (0x005c)



SENS_TOUCH_OUT_TH0 The threshold for touch pad 0. (R/W)

SENS_TOUCH_OUT_TH1 The threshold for touch pad 1. (R/W)

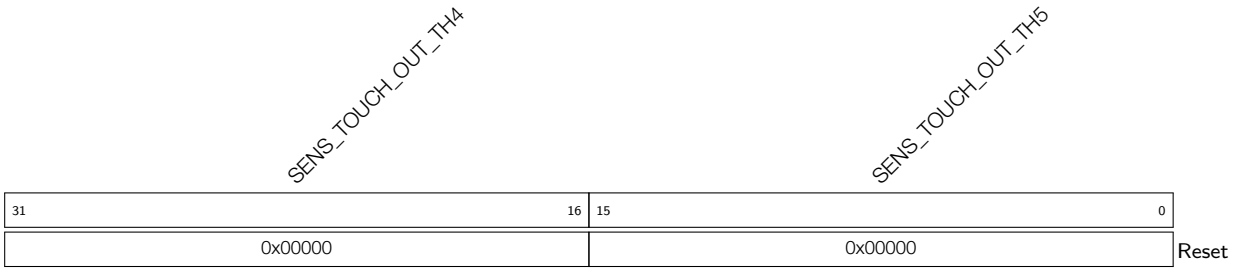
Register 28.9: SENS_SAR_TOUCH_THRES2_REG (0x0060)



SENS_TOUCH_OUT_TH2 The threshold for touch pad 2. (R/W)

SENS_TOUCH_OUT_TH3 The threshold for touch pad 3. (R/W)

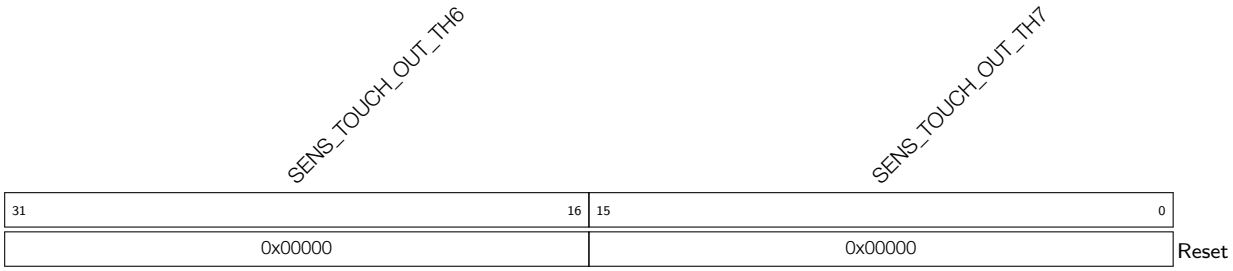
Register 28.10: SENS_SAR_TOUCH_THRES3_REG (0x0064)



SENS_TOUCH_OUT_TH4 The threshold for touch pad 4. (R/W)

SENS_TOUCH_OUT_TH5 The threshold for touch pad 5. (R/W)

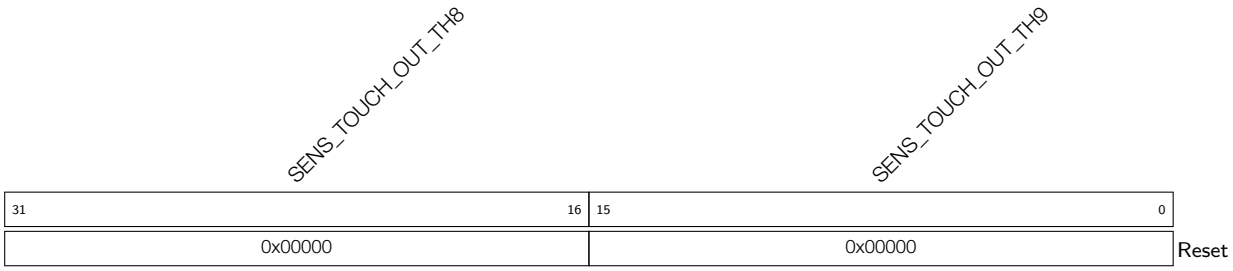
Register 28.11: SENS_SAR_TOUCH_THRES4_REG (0x0068)



SENS_TOUCH_OUT_TH6 The threshold for touch pad 6. (R/W)

SENS_TOUCH_OUT_TH7 The threshold for touch pad 7. (R/W)

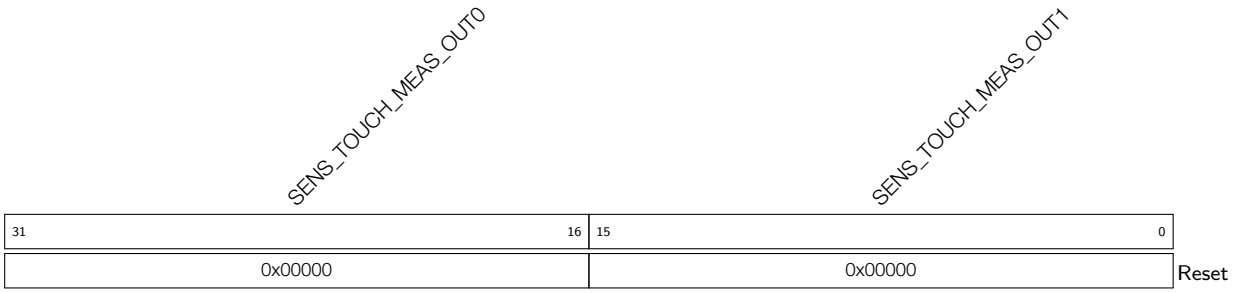
Register 28.12: SENS_SAR_TOUCH_THRES5_REG (0x006c)



SENS_TOUCH_OUT_TH8 The threshold for touch pad 8. (R/W)

SENS_TOUCH_OUT_TH9 The threshold for touch pad 9. (R/W)

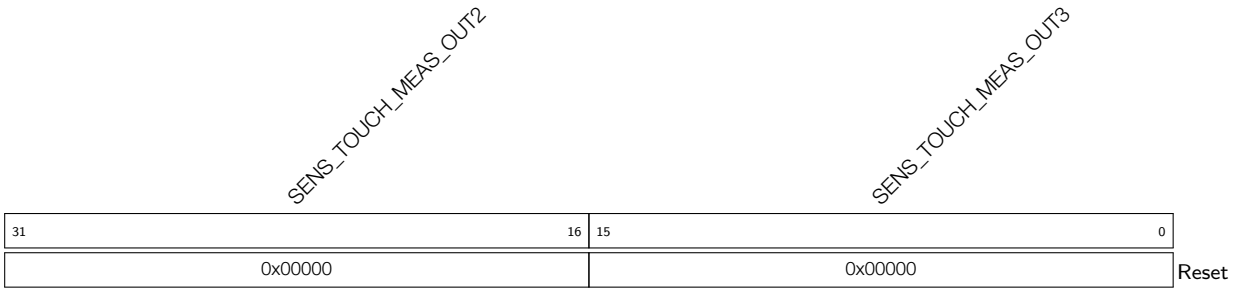
Register 28.13: SENS_SAR_TOUCH_OUT1_REG (0x0070)



SENS_TOUCH_MEAS_OUT0 The counter for touch pad 0. (RO)

SENS_TOUCH_MEAS_OUT1 The counter for touch pad 1. (RO)

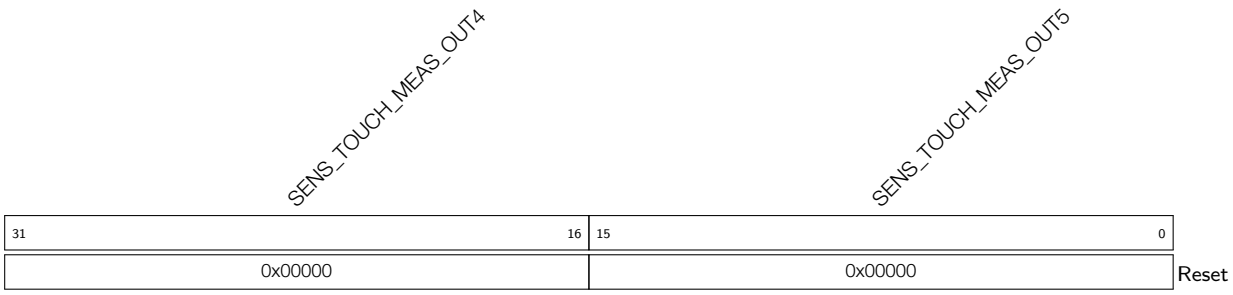
Register 28.14: SENS_SAR_TOUCH_OUT2_REG (0x0074)



SENS_TOUCH_MEAS_OUT2 The counter for touch pad 2. (RO)

SENS_TOUCH_MEAS_OUT3 The counter for touch pad 3. (RO)

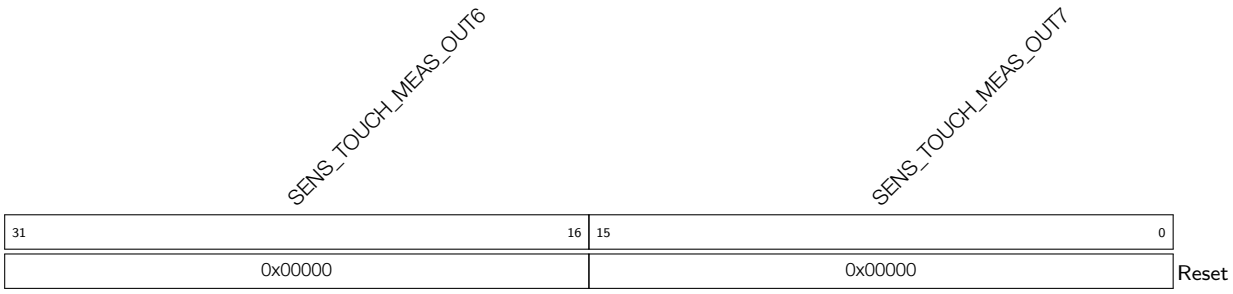
Register 28.15: SENS_SAR_TOUCH_OUT3_REG (0x0078)



SENS_TOUCH_MEAS_OUT4 The counter for touch pad 4. (RO)

SENS_TOUCH_MEAS_OUT5 The counter for touch pad 5. (RO)

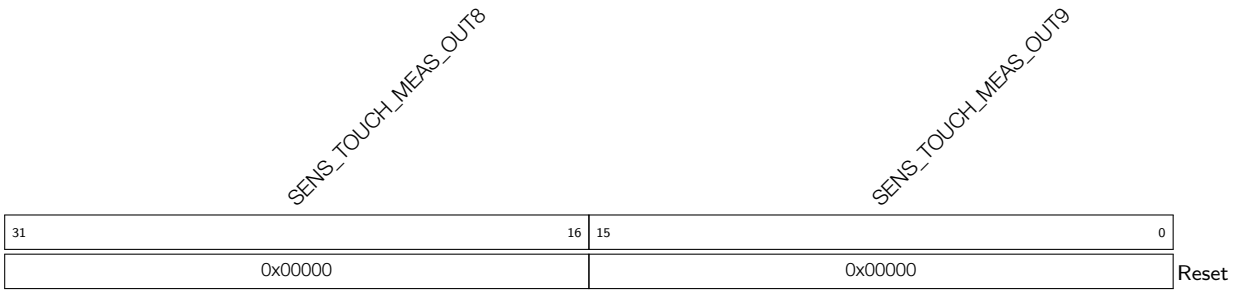
Register 28.16: SENS_SAR_TOUCH_OUT4_REG (0x007c)



SENS_TOUCH_MEAS_OUT6 The counter for touch pad 6. (RO)

SENS_TOUCH_MEAS_OUT7 The counter for touch pad 7. (RO)

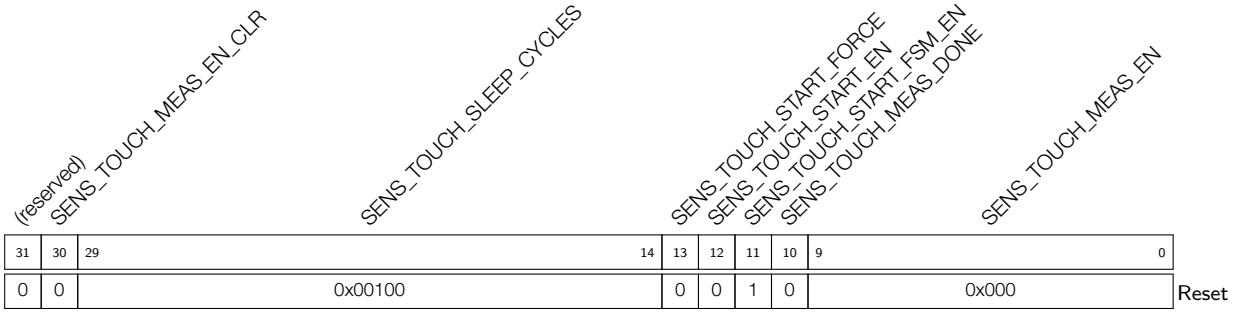
Register 28.17: SENS_SAR_TOUCH_OUT5_REG (0x0080)



SENS_TOUCH_MEAS_OUT8 The counter for touch pad 8. (RO)

SENS_TOUCH_MEAS_OUT9 The counter for touch pad 9. (RO)

Register 28.18: SENS_SAR_TOUCH_CTRL2_REG (0x0084)



SENS_TOUCH_MEAS_EN_CLR Set to clear reg_touch_meas_en. (WO)

SENS_TOUCH_SLEEP_CYCLES Sleep cycles for timer. (R/W)

SENS_TOUCH_START_FORCE 1: starts the Touch FSM via software; 0: starts the Touch FSM via timer. (R/W)

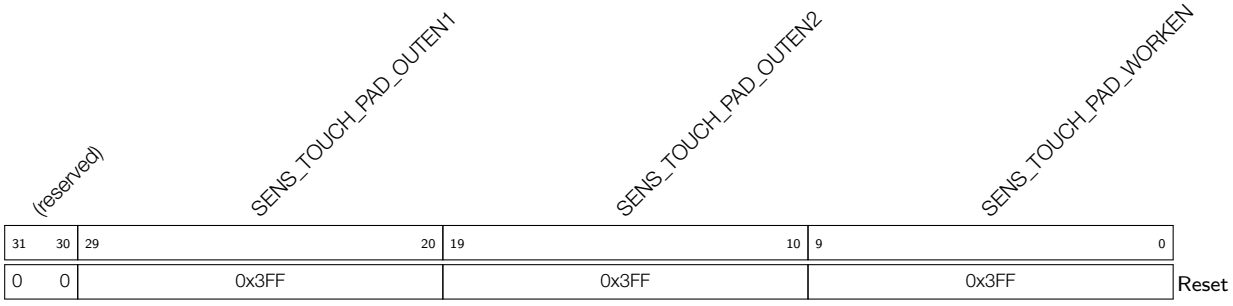
SENS_TOUCH_START_EN 1: starts the Touch FSM; this is valid when reg_touch_start_force is set. (R/W)

SENS_TOUCH_START_FSM_EN 1: TOUCH_START & TOUCH_XPD are controlled by the Touch FSM; 0: TOUCH_START & TOUCH_XPD are controlled by registers. (R/W)

SENS_TOUCH_MEAS_DONE Set to 1 by FSM, indicating that touch measurement is done. (RO)

SENS_TOUCH_MEAS_EN 10-bit register indicating which pads are touched. (RO)

Register 28.19: SENS_SAR_TOUCH_ENABLE_REG (0x008c)

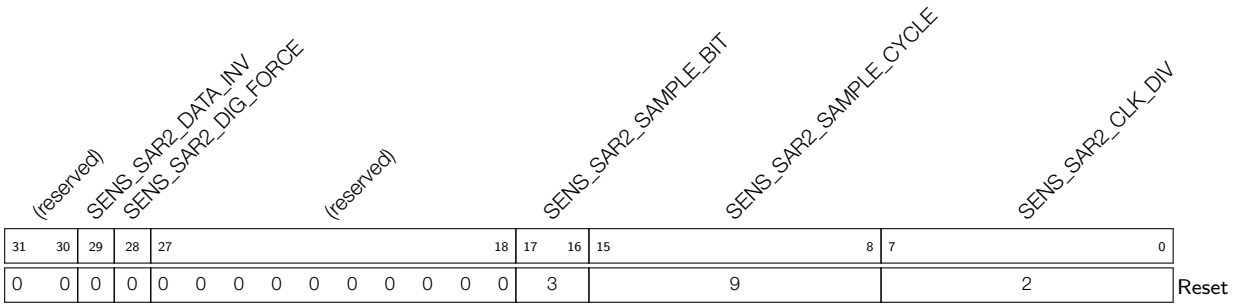


SENS_TOUCH_PAD_OUTEN1 Bitmap defining SET1 for generating a wakeup interrupt; SET1 is considered touched if at least one of the touch pads in SET1 is touched. (R/W)

SENS_TOUCH_PAD_OUTEN2 Bitmap defining SET2 for generating a wakeup interrupt; SET2 is considered touched if at least one of the touch pads in SET2 is touched. (R/W)

SENS_TOUCH_PAD_WORKEN Bitmap defining the working set during measurement. (R/W)

Register 28.20: SENS_SAR_READ_CTRL2_REG (0x0090)



SENS_SAR2_DATA_INV Invert SAR ADC2 data. (R/W)

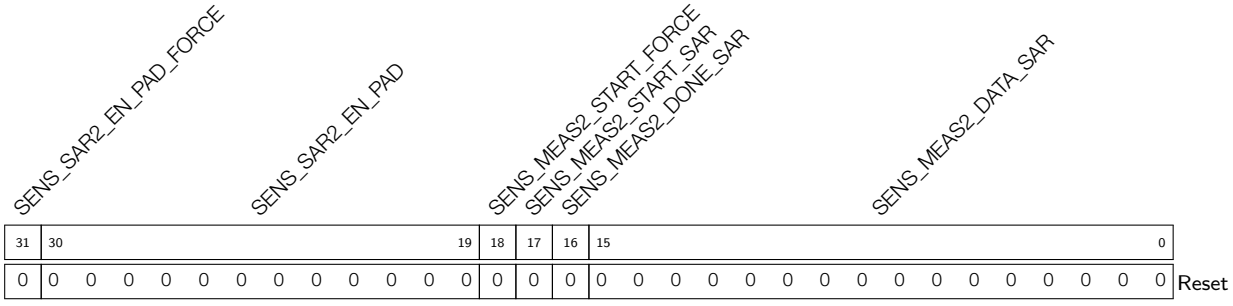
SENS_SAR2_DIG_FORCE 1: SAR ADC2 controlled by DIG ADC2 CTRL or PWDET CTRL, 0: SAR ADC2 controlled by RTC ADC2 CTRL (R/W)

SENS_SAR2_SAMPLE_BIT Bit width of SAR ADC2, 00: for 9-bit, 01: for 10-bit, 10: for 11-bit, 11: for 12-bit. (R/W)

SENS_SAR2_SAMPLE_CYCLE Sample cycles of SAR ADC2. (R/W)

SENS_SAR2_CLK_DIV Clock divider. (R/W)

Register 28.21: SENS_SAR_MEAS_START2_REG (0x0094)



SENS_SAR2_EN_PAD_FORCE 1: SAR ADC2 pad enable bitmap is controlled by SW, 0: SAR ADC2 pad enable bitmap is controlled by ULP coprocessor. (R/W)

SENS_SAR2_EN_PAD SAR ADC2 pad enable bitmap; active only when reg_sar2_en_pad_force = 1. (R/W)

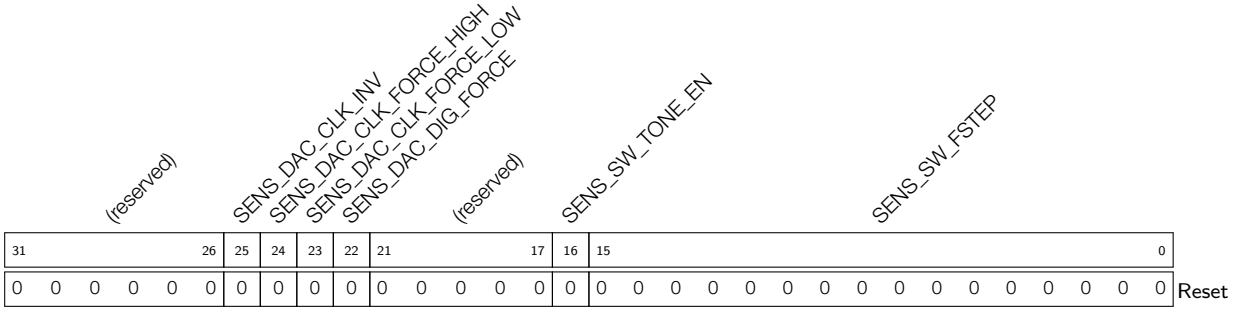
SENS_MEAS2_START_FORCE 1: SAR ADC2 controller (in RTC) is started by SW, 0: SAR ADC2 controller is started by ULP coprocessor. (R/W)

SENS_MEAS2_START_SAR SAR ADC2 controller (in RTC) starts conversion; active only when reg_meas2_start_force = 1. (R/W)

SENS_MEAS2_DONE_SAR SAR ADC2-conversion-done indication. (RO)

SENS_MEAS2_DATA_SAR SAR ADC2 data. (RO)

Register 28.22: SENS_SAR_DAC_CTRL1_REG (0x0098)



SENS_DAC_CLK_INV 1: inverts PDAC_CLK, 0: no inversion. (R/W)

SENS_DAC_CLK_FORCE_HIGH forces PDAC_CLK to be 1. (R/W)

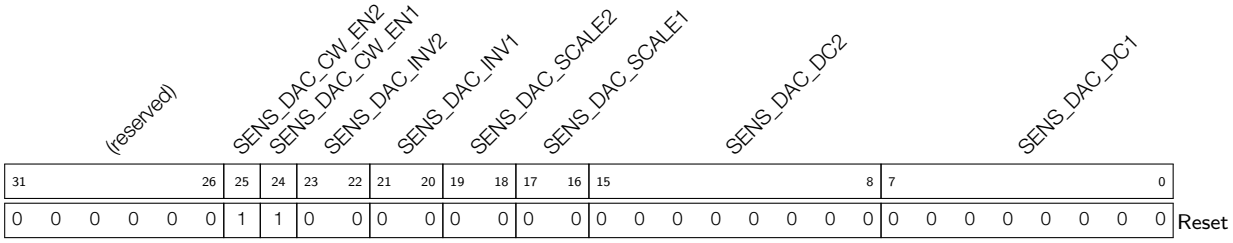
SENS_DAC_CLK_FORCE_LOW forces PDAC_CLK to be 0. (R/W)

SENS_DAC_DIG_FORCE 1: DAC1 & DAC2 use DMA, 0: DAC1 & DAC2 do not use DMA. (R/W)

SENS_SW_TONE_EN 1: enable CW generator, 0: disable CW generator. (R/W)

SENS_SW_FSTEP Frequency step for CW generator; can be used to adjust the frequency. (R/W)

Register 28.23: SENS_SAR_DAC_CTRL2_REG (0x009c)



SENS_DAC_CW_EN2 1: selects CW generator as source for PDAC2_DAC[7:0], 0: selects register reg_pdac2_dac[7:0] as source for PDAC2_DAC[7:0]. (R/W)

SENS_DAC_CW_EN1 1: selects CW generator as source for PDAC1_DAC[7:0], 0: selects register reg_pdac1_dac[7:0] as source for PDAC1_DAC[7:0]. (R/W)

SENS_DAC_INV2 DAC2, 00: does not invert any bits, 01: inverts all bits, 10: inverts MSB, 11: inverts all bits except for MSB. (R/W)

SENS_DAC_INV1 DAC1, 00: does not invert any bits, 01: inverts all bits, 10: inverts MSB, 11: inverts all bits except for MSB. (R/W)

SENS_DAC_SCALE2 DAC2, 00: no scale, 01: scale to 1/2, 10: scale to 1/4, scale to 1/8. (R/W)

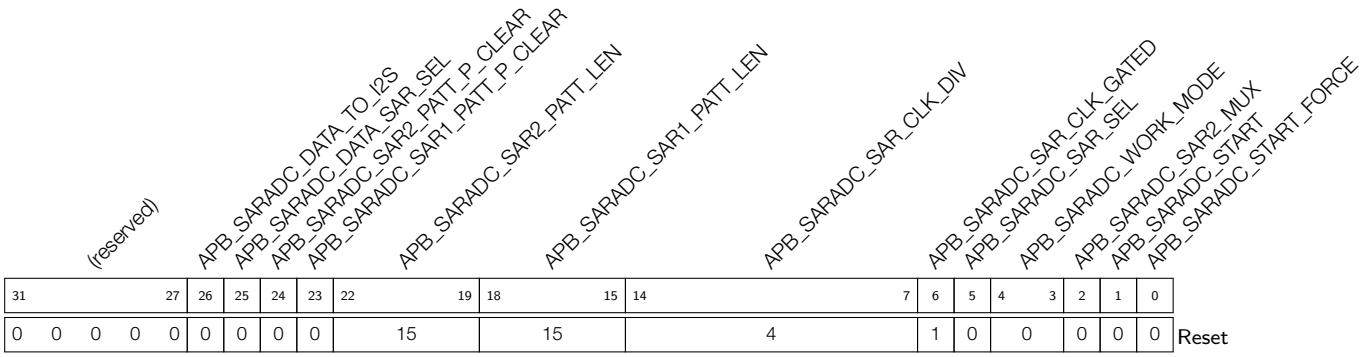
SENS_DAC_SCALE1 DAC1, 00: no scale, 01: scale to 1/2, 10: scale to 1/4, scale to 1/8. (R/W)

SENS_DAC_DC2 DC offset for DAC2 CW generator. (R/W)

SENS_DAC_DC1 DC offset for DAC1 CW generator. (R/W)

28.7.2 Advanced Peripheral Bus

Register 28.24: APB_SARADC_CTRL_REG (0x10)



APB_SARADC_DATA_TO_I2S 1: I2S input data is from SAR ADC (for DMA), 0: I2S input data is from GPIO matrix. (R/W)

APB_SARADC_DATA_SAR_SEL 1: sar_sel will be coded by the MSB of the 16-bit output data, in this case, the resolution should not contain more than 11 bits; 0: using 12-bit SAR ADC resolution. (R/W)

APB_SARADC_SAR2_PATT_P_CLEAR Clears the pointer of pattern table for DIG ADC2 CTRL. (R/W)

APB_SARADC_SAR1_PATT_P_CLEAR Clears the pointer of pattern table for DIG ADC1 CTRL. (R/W)

APB_SARADC_SAR2_PATT_LEN SAR ADC2, 0 - 15 means pattern table length of 1 - 16. (R/W)

APB_SARADC_SAR1_PATT_LEN SAR ADC1, 0 - 15 means pattern table length of 1 - 16. (R/W)

APB_SARADC_SAR_CLK_DIV SAR clock divider. (R/W)

APB_SARADC_SAR_CLK_GATED Reserved. Please initialize to 0b1 (R/W)

APB_SARADC_SAR_SEL 0: SAR1, 1: SAR2, this setting is applicable in the single SAR mode. (R/W)

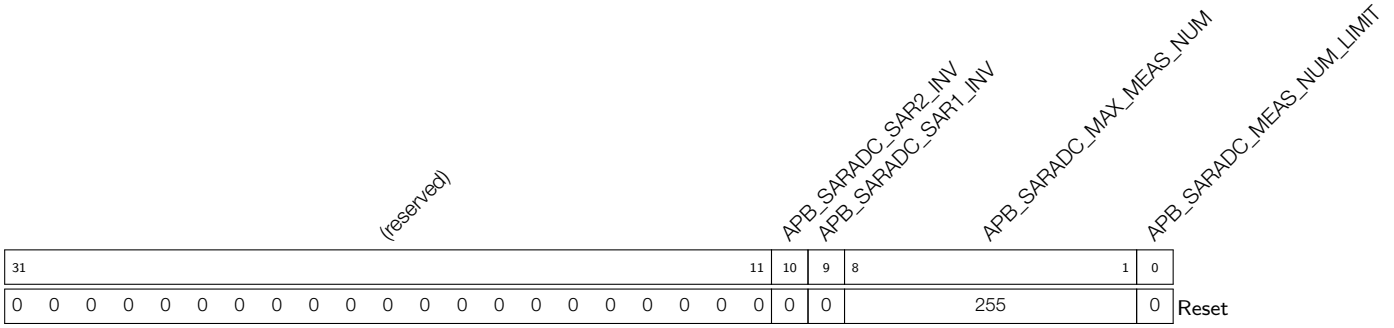
APB_SARADC_WORK_MODE 0: single mode, 1: double mode, 2: alternate mode. (R/W)

APB_SARADC_SAR2_MUX 1: SAR ADC2 is controlled by DIG ADC2 CTRL, 0: SAR ADC2 is controlled by PWDET CTRL. (R/W)

APB_SARADC_START Reserved. Please initialize to 0 (R/W)

APB_SARADC_START_FORCE Reserved. Please initialize to 0 (R/W)

Register 28.25: APB_SARADC_CTRL2_REG (0x14)



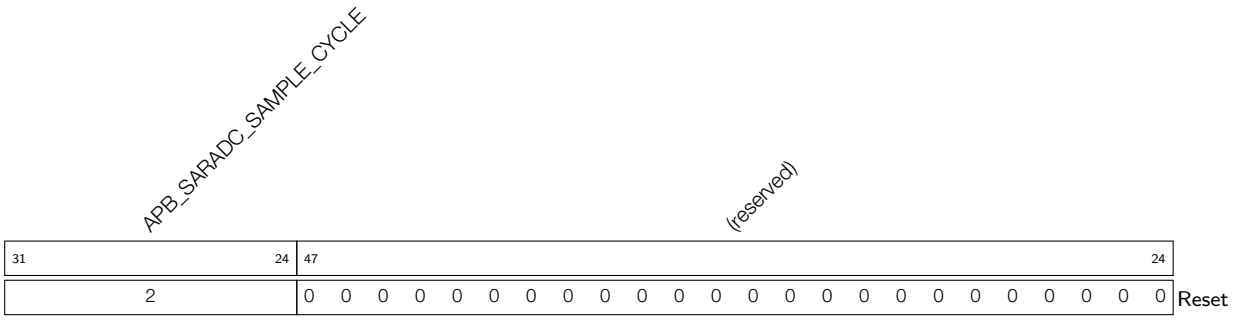
APB_SARADC_SAR2_INV 1: data to DIG ADC2 CTRL is inverted, 0: data is not inverted. (R/W)

APB_SARADC_SAR1_INV 1: data to DIG ADC1 CTRL is inverted, 0: data is not inverted. (R/W)

APB_SARADC_MAX_MEAS_NUM Max conversion number. (R/W)

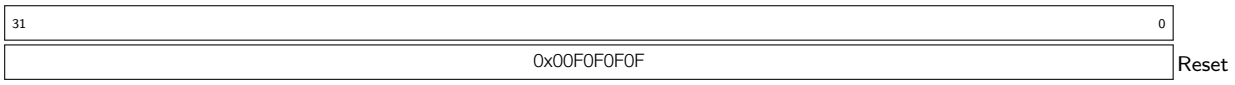
APB_SARADC_MEAS_NUM_LIMIT Reserved. Please initialize to 0b1 (R/W)

Register 28.26: APB_SARADC_FSM_REG (0x18)



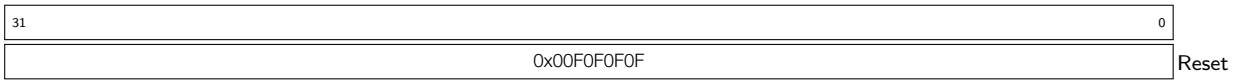
APB_SARADC_SAMPLE_CYCLE Sample cycles. (R/W)

Register 28.27: APB_SARADC_SAR1_PATT_TAB1_REG (0x1C)



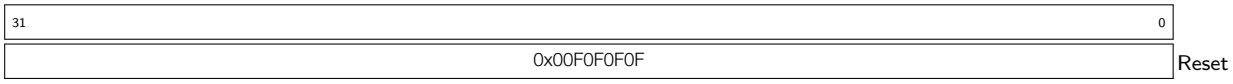
APB_SARADC_SAR1_PATT_TAB1_REG Pattern tables 0 - 3 for SAR ADC1, one byte for each pattern table: [31:28] pattern0_channel, [27:26] pattern0_bit_width, [25:24] pattern0_attenuation, [23:20] pattern1_channel, etc. (R/W)

Register 28.28: APB_SARADC_SAR1_PATT_TAB2_REG (0x20)



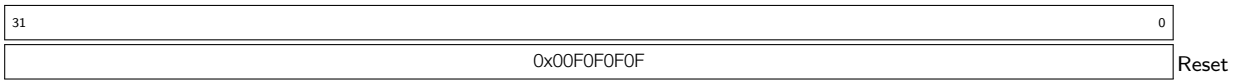
APB_SARADC_SAR1_PATT_TAB2_REG Pattern tables 4 - 7 for SAR ADC1, one byte for each pattern table: [31:28] pattern4_channel, [27:26] pattern4_bit_width, [25:24] pattern4_attenuation, [23:20] pattern5_channel, etc. (R/W)

Register 28.29: APB_SARADC_SAR1_PATT_TAB3_REG (0x24)



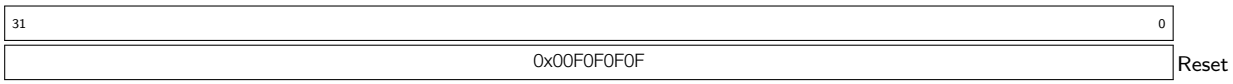
APB_SARADC_SAR1_PATT_TAB3_REG Pattern tables 8 - 11 for SAR ADC1, one byte for each pattern table: [31:28] pattern8_channel, [27:26] pattern8_bit_width, [25:24] pattern8_attenuation, [23:20] pattern9_channel, etc. (R/W)

Register 28.30: APB_SARADC_SAR1_PATT_TAB4_REG (0x28)



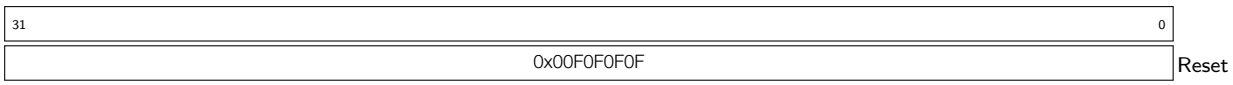
APB_SARADC_SAR1_PATT_TAB4_REG Pattern tables 12 - 15 for SAR ADC1, one byte for each pattern table: [31:28] pattern12_channel, [27:26] pattern12_bit_width, [25:24] pattern12_attenuation, [23:20] pattern13_channel, etc. (R/W)

Register 28.31: APB_SARADC_SAR2_PATT_TAB1_REG (0x2C)



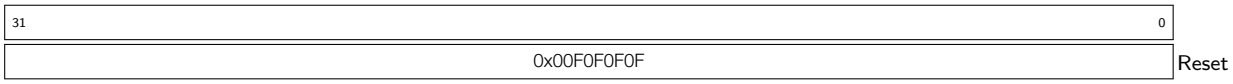
APB_SARADC_SAR2_PATT_TAB1_REG Pattern tables 0 - 3 for SAR ADC2, one byte for each pattern table: [31:28] pattern0_channel, [27:26] pattern0_bit_width, [25:24] pattern0_attenuation, [23:20] pattern1_channel, etc. (R/W)

Register 28.32: APB_SARADC_SAR2_PATT_TAB2_REG (0x30)



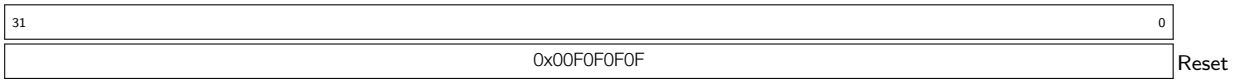
APB_SARADC_SAR2_PATT_TAB2_REG Pattern tables 4 - 7 for SAR ADC2, one byte for each pattern table: [31:28] pattern4_channel, [27:26] pattern4_bit_width, [25:24] pattern4_attenuation, [23:20] pattern5_channel, etc. (R/W)

Register 28.33: APB_SARADC_SAR2_PATT_TAB3_REG (0x34)



APB_SARADC_SAR2_PATT_TAB3_REG Pattern tables 8 - 11 for SAR ADC2, one byte for each pattern table: [31:28] pattern8_channel, [27:26] pattern8_bit_width, [25:24] pattern8_attenuation, [23:20] pattern9_channel, etc. (R/W)

Register 28.34: APB_SARADC_SAR2_PATT_TAB4_REG (0x38)



APB_SARADC_SAR2_PATT_TAB4_REG Pattern tables 12 - 15 for SAR ADC2, one byte for each pattern table: [31:28] pattern12_channel, [27:26] pattern12_bit_width, [25:24] pattern12_attenuation, [23:20] pattern13_channel, etc. (R/W)

28.7.3 RTC I/O

For details, please refer to Section Registers in Chapter IO_MUX and GPIO Matrix.

29. ULP Co-processor

29.1 Introduction

The ULP co-processor is an ultra-low-power processor that remains powered on during the Deep-sleep mode of the main SoC. Hence, the developer can store in the RTC memory a program for the ULP co-processor to access peripheral devices, internal sensors and RTC registers during deep sleep. This is useful for designing applications where the CPU needs to be woken up by an external event, or timer, or a combination of these, while maintaining minimal power consumption.

29.2 Features

- Contains up to 8 KB of SRAM for instructions and data
- Uses RTC_FAST_CLK, which is 8 MHz
- Works both in normal and deep sleep
- Is able to wake up the digital core or send an interrupt to the CPU
- Can access peripheral devices, internal sensors and RTC registers
- Contains four 16-bit general-purpose registers (R0, R1, R2, R3) for manipulating data and accessing memory
- Includes one 8-bit Stage_cnt register which can be manipulated by ALU and used in JUMP instructions

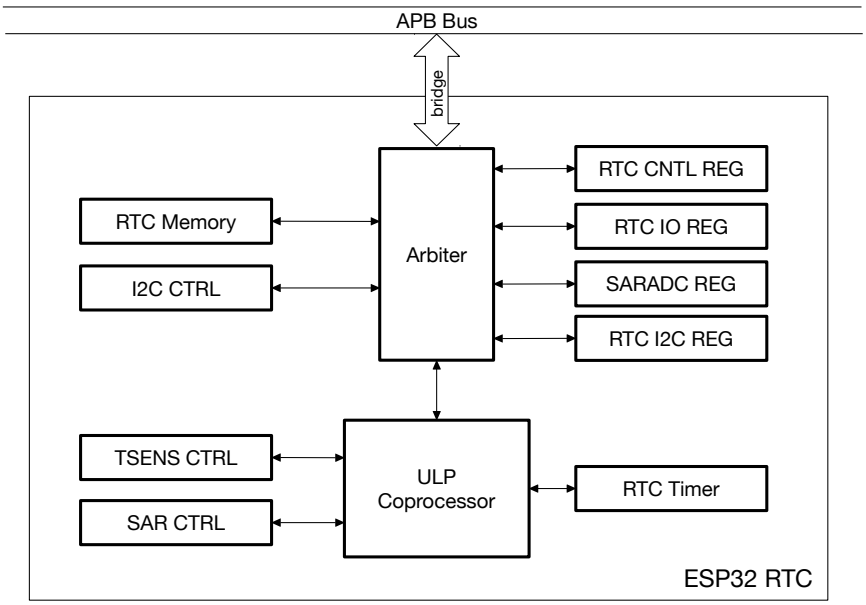


Figure 137: ULP Co-processor Diagram

29.3 Functional Description

The ULP co-processor is a programmable FSM (Finite State Machine) that can work during deep sleep. Like general-purpose CPUs, ULP co-processor also has some instructions which can be useful for a relatively complex logic, and also some special commands for RTC controllers/peripherals. The 8 KB of SRAM RTC slow memory can be accessed by both the ULP co-processor and the CPU; hence, it is usually used to store instructions and share data between the ULP co-processor and the CPU.

The ULP co-processor can be started by software or a periodically-triggered timer. The operation of the ULP co-processor is ended by executing the HALT instruction. Meanwhile, it can access almost every module in RTC domain, either through built-in instructions or RTC registers. In many cases the ULP co-processor can be a good supplement to, or replacement of, the CPU, especially for power-sensitive applications. Figure 137 shows the overall layout of a ULP co-processor.

29.4 Instruction Set

The ULP co-processor provides the following instructions:

- Perform arithmetic and logic operations - ALU
- Load and store data - LD, ST, REG_RD and REG_WR
- Jump to a certain address - JUMP
- Manage program execution - WAIT/HALT
- Control sleep period of ULP co-processor - SLEEP
- Wake up/communicate with SoC - WAKE
- Take measurements - ADC
- Communicate using I2C - I2C_RD/I2C_WR

The ULP co-processor's instruction format is shown in Figure 138.



Figure 138: The ULP Co-processor Instruction Format

An instruction, which has one *OpCode*, can perform various different operations, depending on the setting of *Operands* bits. A good example is the ALU instruction, which is able to perform ten arithmetic and logic operations; or the JUMP instruction, which may be conditional or unconditional, absolute or relative.

Each instruction has a fixed width of 32 bits. A series of instructions can make a program be executed by the ULP co-processor. The execution flow inside the program uses 32-bit addressing. The program is stored in a dedicated region called Slow Memory (RTC_SLOW_MEM), which is visible to the main CPUs as one that has an address range of 0x5000_0000 to 0x5000_1FFF (8 KB).

29.4.1 ALU - Perform Arithmetic/Logic Operations

The ALU (Arithmetic and Logic Unit) performs arithmetic and logic operations on values stored in ULP co-processor registers, and on immediate values stored in the instruction itself.
The following operations are supported:

- Arithmetic: ADD and SUB
- Logic: AND and OR
- Bit shifting: LSH and RSH
- Moving data to register: MOVE
- Stage count register manipulation: STAGE_RST, STAGE_INC and STAGE_DEC

The ALU instruction, which has one *OpCode*, can perform various different arithmetic and logic operations, depending on the setting of the instruction's bits [27:21] accordingly.

29.4.1.1 Operations among Registers

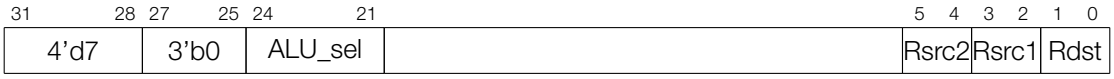


Figure 139: Instruction Type — ALU for Operations among Registers

When bits [27:25] of the instruction in Figure 139 are set to 3'b0, ALU performs operations, using the ULP co-processor register R[0-3]. The types of operations depend on the setting of the instruction's bits [24:21] presented in Table 126.

Operand	**Description** - see Figure 139
ALU_sel	Type of ALU operation
Rdst	Register R[0-3], destination
Rsrc1	Register R[0-3], source
Rsrc2	Register R[0-3], source

ALU_sel	Instruction	Operation	Description
0	ADD	*Rdst* = *Rsrc1* + *Rsrc2*	Add to register
1	SUB	*Rdst* = *Rsrc1* - *Rsrc2*	Subtract from register
2	AND	*Rdst* = *Rsrc1* & *Rsrc2*	Logical AND of two operands
3	OR	*Rdst* = *Rsrc1* | *Rsrc2*	Logical OR of two operands
4	MOVE	*Rdst* = *Rsrc1*	Move to register
5	LSH	*Rdst* = *Rsrc1* << *Rsrc2*	Logical Shift Left
6	RSH	*Rdst* = *Rsrc1* >> *Rsrc2*	Logical Shift Right

Table 126: ALU Operations among Registers

Note:

- ADD/SUB operations can be used to set/clear the overflow flag in ALU.
- All ALU operations can be used to set/clear the zero flag in ALU.

29.4.1.2 Operations with Immediate Value

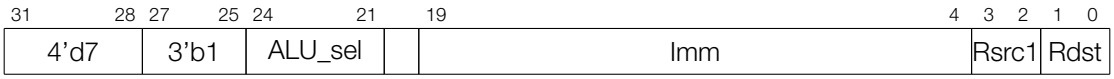


Figure 140: Instruction Type — ALU for Operations with Immediate Value

When bits [27:25] of the instruction in Figure 140 are set to 3'b1, ALU performs operations, using register R[0-3] and the immediate value stored in [19:4]. The types of operations depend on the setting of the instruction's bits [24:21] presented in Table 127.

Operand	**Description** - see Figure 140
ALU_sel	Type of ALU operation
Rdst	Register R[0-3], destination
Rsrc1	Register R[0-3], source
Imm	16-bit signed value

ALU_sel	Instruction	Operation	Description
0	ADD	*Rdst = Rsrc1 + Imm*	Add to register
1	SUB	*Rdst = Rsrc1 - Imm*	Subtract from register
2	AND	*Rdst = Rsrc1 & Imm*	Logical AND of two operands
3	OR	*Rdst = Rsrc1 | Imm*	Logical OR of two operands
4	MOVE	*Rdst = Imm*	Move to register
5	LSH	*Rdst = Rsrc1 << Imm*	Logical Shift to the Left
6	RSH	*Rdst = Rsrc1 >> Imm*	Logical Shift to the Right

Table 127: ALU Operations with Immediate Value

Note:

- ADD/SUB operations can be used to set/clear the overflow flag in ALU.
- All ALU operations can be used to set/clear the zero flag in ALU.

29.4.1.3 Operations with Stage Count Register

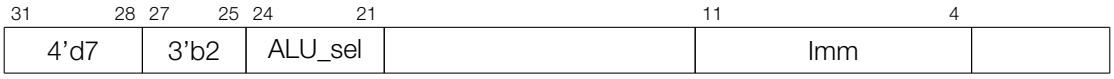


Figure 141: Instruction Type — ALU for Operations with Stage Count Register

ALU is also able to increment/decrement by a given value, or reset the 8-bit register Stage_cnt. To do so, bits [27:25] of instruction in Figure 141 should be set to 3'b2. The type of operation depends on the setting of the instruction's bits [24:21] presented in Table 128. The Stage_cnt is a separate register and is not a part of the instruction in Figure 141.

Operand	**Description** - see Figure 141
ALU_sel	Type of ALU operation
Stage_cnt	Stage count register, a separate register [7:0] used to store variables, such as loop index
Imm	8-bit value

ALU_sel	Instruction	Operation	Description
0	STAGE_INC	*Stage_cnt* = *Stage_cnt* + *Imm*	Increment stage count register
1	STAGE_DEC	*Stage_cnt* = *Stage_cnt* - *Imm*	Decrement stage count register
2	STAGE_RST	*Stage_cnt* = 0	Reset stage count register

Table 128: ALU Operations with Stage Count Register

29.4.2 ST – Store Data in Memory

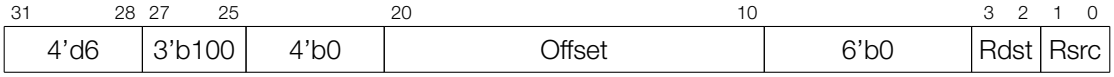


Figure 142: Instruction Type — ST

Operand	**Description** - see Figure 142
Offset	10-bit signed value, offset expressed in 32-bit words
Rsrc	Register R[0-3], 16-bit value to store
Rdst	Register R[0-3], address of the destination, expressed in 32-bit words

Description

The instruction stores the 16-bit value of *Rsrc* in the lower half-word of memory with address *Rdst* + *Offset*. The upper half-word is written with the current program counter (PC) expressed in words and shifted to the left by 5 bits:

Mem [Rdst + Offset]{31:0} = {PC[10:0], 5'b0, Rsrc[15:0]}

The application can use the higher 16 bits to determine which instruction in the ULP program has written any particular word into memory.

Note:

- This instruction can only access 32-bit memory words.
- Data from *Rsrc* is always stored in the lower 16 bits of a memory word. Differently put, it is not possible to store *Rsrc* in the upper 16 bits of memory.
- The "Mem" written is the RTC_SLOW_MEM memory. Address 0, as seen by the ULP co-processor, corresponds to address 0x50000000, as seen by the main CPUs.

29.4.3 LD – Load Data from Memory

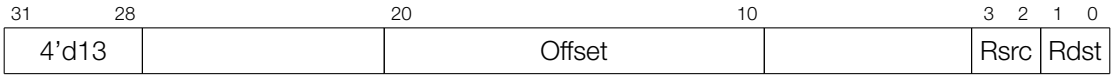


Figure 143: Instruction Type — LD

Operand	**Description** - see Figure 143
Offset	10-bit signed value, offset expressed in 32-bit words
Rsrc	Register R[0-3], address of destination memory, expressed in 32-bit words
Rdst	Register R[0-3], destination

Description

The instruction loads the lower 16-bit half-word from memory with address *Rsrc* + *offset* into the destination register *Rdst*:

Rdst[15:0] = Mem[Rsrc + Offset][15:0]

Note:

- This instruction can only access 32-bit memory words.
- In any case, it is always the lower 16 bits of a memory word that are loaded. Differently put, it is not possible to read the upper 16 bits.
- The "Mem" loaded is the RTC_SLOW_MEM memory. Address 0, as seen by the ULP co-processor, corresponds to address 0x50000000, as seen by the main CPUs.

29.4.4 JUMP – Jump to an Absolute Address

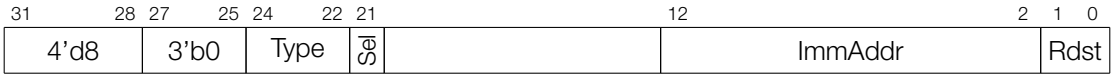


Figure 144: Instruction Type — JUMP

Operand	**Description** - see Figure 144
Rdst	Register R[0-3], address to jump to
ImmAddr	13-bit address, expressed in 32-bit words
Sel	Selects the address to jump to:
	0 - jump to the address contained in *ImmAddr*
	1 - jump to the address contained in *Rdst*
Type	Jump type:
	0 - make an unconditional jump
	1 - jump only if the last ALU operation has set the zero flag
	2 - jump only if the last ALU operation has set the overflow flag

Description

The instruction prompts a jump to the specified address. The jump can be either unconditional or based on the ALU flag.

Note:

All jump addresses are expressed in 32-bit words.

29.4.5 JUMPR – Jump to a Relative Offset (Conditional upon R0)

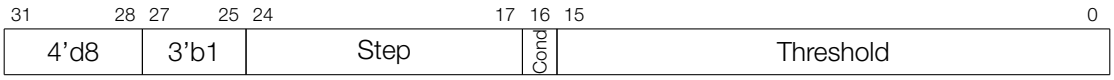


Figure 145: Instruction Type — JUMPR

Operand	**Description** - see Figure 145
Step	Relative shift from current position, expressed in 32-bit words:
	if Step[7] = 0 then PC = PC + Step[6:0]
	if Step[7] = 1 then PC = PC - Step[6:0]
Threshold	Threshold value for condition (see *Cond* below) to jump
Cond	Condition to jump:
	0 - jump if R0 < *Threshold*
	1 - jump if R0 >= *Threshold*

Description

The instruction prompts a jump to a relative address, if the above-mentioned condition is true. The condition itself is the result of comparing the R0 register value and the *Threshold* value.

Note:

All jump addresses are expressed in 32-bit words.

29.4.6 JUMPS – Jump to a Relative Address (Conditional upon Stage Count Register)

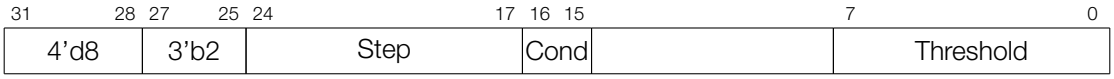


Figure 146: Instruction Type — JUMP

Operand	**Description** - see Figure 146
Step	Relative shift from current position, expressed in 32-bit words: if Step[7] = 0, then PC = PC + Step[6:0] if Step[7] = 1, then PC = PC - Step[6:0]
Threshold	Threshold value for condition (see *Cond* below) to jump
Cond	Condition of jump: 1X - jump if *Stage_cnt* == *Threshold* 00 - jump if *Stage_cnt* < *Threshold* 01 - jump if *Stage_cnt* > *Threshold*

Note:

- A description of how to set the stage count register is provided in section 29.4.1.3.
- All jump addresses are expressed in 32-bit words.

Description

The instruction prompts a jump to a relative address if the above-mentioned condition is true. The condition itself is the result of comparing the value of *Stage_cnt* (stage count register) and the *Threshold* value.

29.4.7 HALT – End the Program

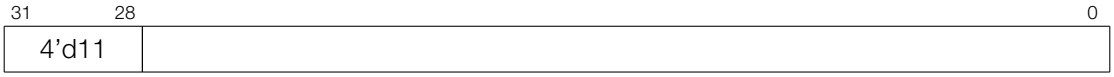


Figure 147: Instruction Type — HALT

Description

The instruction ends the operation of the processor and puts it into power-down mode.

Note:

After executing this instruction, the ULP co-processor timer gets started.

29.4.8 WAKE – Wake up the Chip

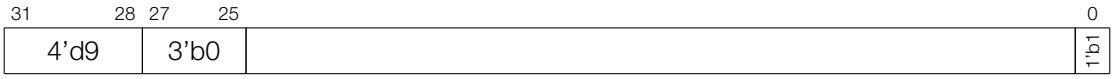


Figure 148: Instruction Type — WAKE

Description

This instruction sends an interrupt from the ULP co-processor to the RTC controller.

- If the SoC is in Deep-sleep mode, and the ULP wake-up is enabled, the above-mentioned interrupt will wake up the SoC.
- If the SoC is not in Deep-sleep mode, and the ULP interrupt bit (RTC_CNTL_ULP_CP_INT_ENA) is set in register RTC_CNTL_INT_ENA_REG, a RTC interrupt will be triggered.

29.4.9 Sleep – Set the ULP Timer's Wake-up Period

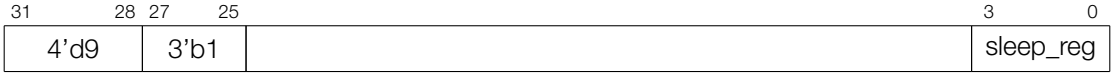


Figure 149: Instruction Type — SLEEP

Operand	**Description** - see Figure 149
sleep_reg	Selects one of five SENS_ULP_CP_SLEEP_CYC*n*_REG (*n*: 0-4) as the wake-up period of the ULP co-processor

Description

The instruction selects which one of the SENS_ULP_CP_SLEEP_CYC*n*_REG (*n*: 0-4) register values is to be used by the ULP timer as the wake-up period. By default, the value of SENS_ULP_CP_SLEEP_CYC0_REG is used.

29.4.10 WAIT – Wait for a Number of Cycles

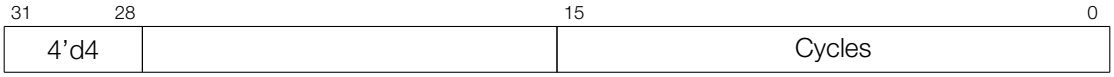


Figure 150: Instruction Type — WAIT

Operand	**Description** - see Figure 150
Cycles	the number of cycles to wait between sleeps

Description

The instruction will delay the ULP co-processor from getting into sleep for a certain number of *Cycles*.

29.4.11 ADC – Take Measurement with ADC

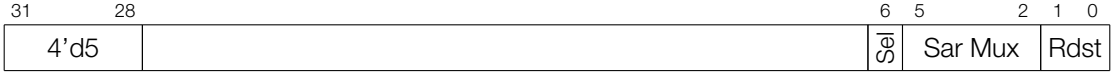


Figure 151: Instruction Type — ADC

Operand	**Description** - see Figure 151
Rdst	Destination Register R[0-3], results will be stored in this register.
Sel	Selected ADC : 0 = SAR ADC1, 1 = SAR ADC2, see Table 129.
Sar Mux	SARADC Pad [Sar_Mux - 1] is enabled, see Table 129.

Table 129: Input Signals Measured using the ADC Instruction

Pad Name/Signal/GPIO	*Sar_Mux*	Processed by /*Sel*
SENSOR_VP (GPIO36)	1	SAR ADC1/*Sel* = 0
SENSOR_CAPP (GPIO37)	2	
SENSOR_CAPN (GPIO38)	3	
SENSOR_VN (GPIO39)	4	
32K_XP (GPIO33)	5	
32K_XN (GPIO32)	6	
VDET_1 (GPIO34)	7	
VDET_2 (GPIO35)	8	
Hall phase 1	9	
Hall phase 0	10	
GPIO4	1	SAR ADC2/*Sel* = 1
GPIO0	2	
GPIO2	3	
MTDO (GPIO15)	4	
MTCK (GPIO13)	5	
MTDI (GPIO12)	6	
MTMS (GPIO14)	7	
GPIO27	8	
GPIO25	9	
GPIO26	10	

Description

The instruction prompts the taking of measurements with the use of ADC. Pads/signals available for ADC measurement are provided in Table 129.

29.4.12 I2C_RD/I2C_WR – Read/Write I2C

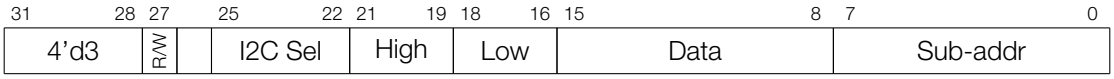


Figure 152: Instruction Type — I2C

Operand	**Description** - see Figure 152
Sub-addr	Slave register address
Data	Data to write in I2C_WR operation (not used in I2C_RD operation)
Low	High part of bit mask
High	Low part of bit mask
I2C Sel	Select register *n* of SENS_I2C_SLAVE_ADDR*n* (*n*: 0-7), which contains the I2C slave address.
R/W	I2C communication direction: 1 - I2C write 0 - I2C read

Description

Communicate (read/write) with external I2C slave devices. Details on using the RTC I2C peripheral are provided in section 29.6.

Note:

When working in master mode, RTC_I2C samples the SDA input on the negative edge of SCL.

29.4.13 REG_RD – Read from Peripheral Register

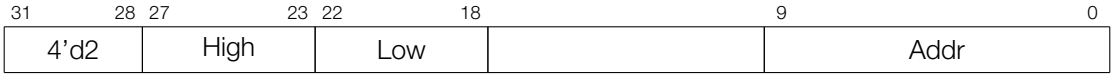


Figure 153: Instruction Type — REG_RD

Operand	**Description** - see Figure 153
Addr	Register address, expressed in 32-bit words
High	High part of R0
Low	Low part of R0

Description

The instruction prompts a read of up to 16 bits from a peripheral register into a general-purpose register:

$$R0 = REG[Addr][High:Low]$$

In case of more than 16 bits being requested, i.e. *High* - *Low* + 1 > 16, then the instruction will return [Low+15:Low].

Note:

- This instruction can access registers in RTC_CNTL, RTC_IO, SENS and RTC_I2C peripherals. The address of the register, as seen from the ULP co-processor, can be calculated from the address of the same register on the DPORT bus, as follows:

$$addr\_ulp = (addr\_dport - DR\_REG\_RTCCNTL\_BASE)/4$$

- The *addr_ulp* is expressed in 32-bit words (not in bytes), and value 0 maps onto the DR_REG_RTCCNTL_BASE (as seen from the main CPUs). Thus, 10 bits of address cover a 4096-byte range of peripheral register space, including regions DR_REG_RTCCNTL_BASE, DR_REG_RTCIO_BASE, DR_REG_SENS_BASE and DR_REG_RTC_I2C_BASE.

29.4.14 REG_WR – Write to Peripheral Register

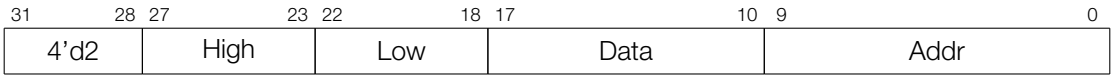


Figure 154: Instruction Type — REG_WR

Operand	**Description** - see Figure 154
Addr	Register address, expressed in 32-bit words
High	High part of R0
Low	Low part of R0
Data	Value to write, 8 bits

Description

The instruction prompts the writing of up to 8 bits from a general-purpose register into a peripheral register.

$$REG[Addr][High:Low] = Data$$

If more than 8 bits are requested, i.e. *High* - *Low* + 1 > 8, then the instruction will pad with zeros the bits above the eighth bit.

Note:

See notes regarding *addr_ulp* in section 29.4.13 above.

29.5 ULP Program Execution

The ULP co-processor is designed to operate independently of the main CPUs, while they are either in deep sleep or running.

In a typical power-saving scenario, the ULP co-processor operates while the main CPUs are in deep sleep. To save power even further, the ULP co-processor can get into sleep mode, as well. In such a scenario, there is a specific hardware timer in place to wake up the ULP co-processor, since there is no software program running at the same time. This timer should be configured in advance by setting and then selecting one of the SENS_ULP_CP_SLEEP_CYC*n*_REG registers that contain the expiration period. This can be done either by the main program, or the ULP program with the REG_WR and SLEEP instructions. Then, the ULP timer should be enabled by setting bit RTC_CNTL_ULP_CP_SLP_TIMER_EN in the RTC_CNTL_STATE0_REG register.

The ULP co-processor puts itself into sleep mode by executing the HALT instruction. This also triggers the ULP timer to start counting RTC_SLOW_CLK ticks which, by default, originate from an internal 150 kHz RC oscillator. Once the timer expires, the ULP co-processor is powered up and runs a program with the program counter (PC) which is stored in register SENS_PC_INIT. The relationship between the described signals and registers is shown in Figure 155.

On reset or power-up the above-mentioned ULP program may start up only after the expiration of SENS_ULP_CP_SLEEP_CYC0_REG, which is the default selection period of the ULP timer.

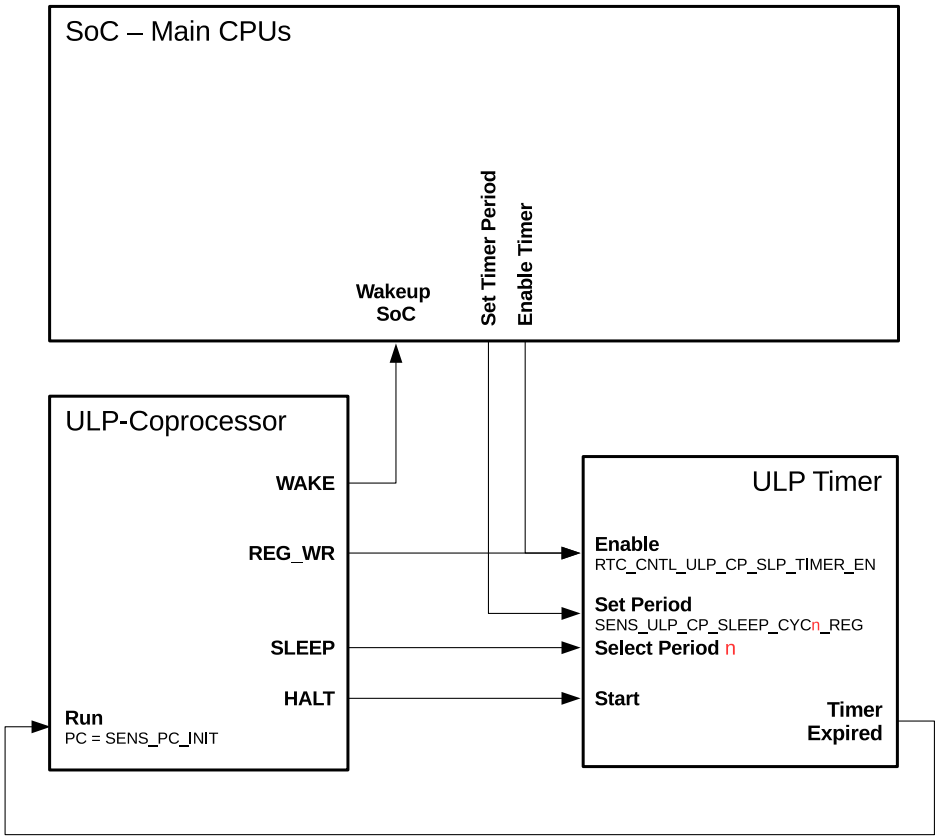


Figure 155: Control of ULP Program Execution

A sample operation sequence of the ULP program is shown in Figure 156, where the following steps are executed:

1. Software enables the ULP timer by using bit RTC_CNTL_ULP_CP_SLP_TIMER_EN.
2. The ULP timer expires and the ULP co-processor starts running the program at PC = SENS_PC_INIT.
3. The ULP program executes the HALT instruction; the ULP co-processor is halted and the timer gets restarted.
4. The ULP program executes the SLEEP instruction to change the sleep timer period register.
5. The ULP program, or software, disables the ULP timer by using bit RTC_CNTL_ULP_CP_SLP_TIMER_EN.

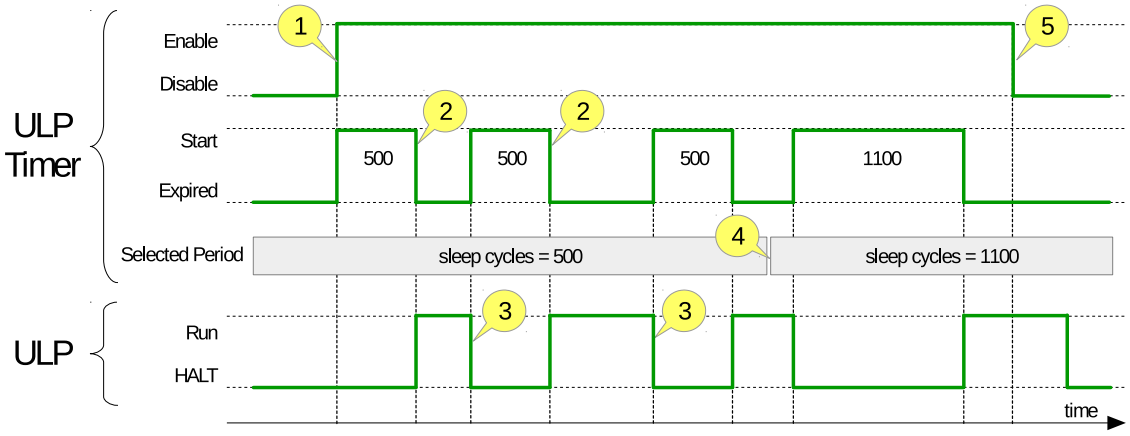


Figure 156: Sample of a ULP Operation Sequence

29.6 RTC_I2C Controller

The ULP co-processor can use a separate I2C controller, located in the RTC domain, to communicate with external I2C slave devices. RTC_I2C has a limited feature set, compared to I2C0/I2C1 peripherals.

29.6.1 Configuring RTC_I2C

Before the ULP co-processor can use the I2C instruction, certain parameters of the RTC_I2C need to be configured. This can be done by the program running on one of the main CPUs, or by the ULP co-processor itself. Configuration is performed by writing certain timing parameters into the RTC_I2C registers:

1. Set the low and high SCL half-periods by using RTC_I2C_SCL_LOW_PERIOD_REG and RTC_I2C_SCL_HIGH_PERIOD_REG in RTC_FAST_CLK cycles (e.g. RTC_I2C_SCL_LOW_PERIOD=40, RTC_I2C_SCL_HIGH_PERIOD=40 for 100 kHz frequency).
2. Set the number of cycles between the SDA switch and the falling edge of SCL by using RTC_I2C_SDA_DUTY_REG in RTC_FAST_CLK (e.g. RTC_I2C_SDA_DUTY=16).
3. Set the waiting time after the START condition by using RTC_I2C_SCL_START_PERIOD_REG (e.g. RTC_I2C_SCL_START_PERIOD=30).
4. Set the waiting time before the END condition by using RTC_I2C_SCL_STOP_PERIOD_REG (e.g. RTC_I2C_SCL_STOP_PERIOD=44).
5. Set the transaction timeout by using RTC_I2C_TIMEOUT_REG (e.g. RTC_I2C_TIMEOUT=200).
6. Enable the master mode (set the RTC_I2C_MS_MODE bit in RTC_I2C_CTRL_REG).
7. Write the address(es) of external slave(s) to SENS_I2C_SLAVE_ADDR*n* (*n*: 0-7). Up to eight slave addresses can be pre-programmed this way. One of these addresses can then be selected for each transaction as part of the ULP I2C instruction.

Once RTC_I2C is configured, instructions ULP I2C_RD and I2C_WR can be used.

29.6.2 Using RTC_I2C

The ULP co-processor supports two instructions (with a single OpCode) for using RTC_I2C: I2C_RD (read) and I2C_WR (write).

29.6.2.1 I2C_RD - Read a Single Byte

The I2C_RD instruction performs the following I2C transaction (see Figure 157):

1. Master generates a START condition.
2. Master sends slave address, with r/w bit set to 0 (“write”). Slave address is obtained from SENS_I2C_SLAVE_ADDR*n*, where *n* is given as an argument to the I2C_RD instruction.
3. Slave generates ACK.
4. Master sends slave register address (given as an argument to the I2C_RD instruction).
5. Slave generates ACK.
6. Master generates a repeated START condition.
7. Master sends slave address, with r/w bit set to 1 (“read”).
8. Slave sends one byte of data.
9. Master generates NACK.
10. Master generates a STOP condition.

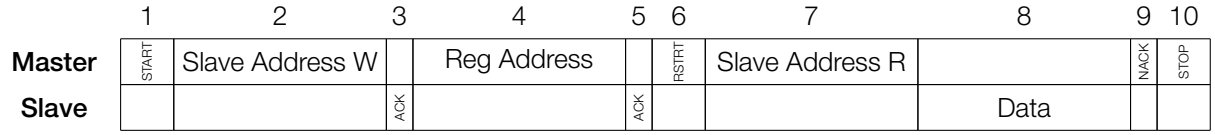


Figure 157: I2C Read Operation

Note:
The RTC_I2C peripheral samples the SDA signals on the falling edge of SCL. If the slave changes SDA in less than 0.38 microseconds, the master will receive incorrect data.

The byte received from the slave is stored into the R0 register.

29.6.2.2 I2C_WR - Write a Single Byte

The I2C_WR instruction performs the following I2C transaction (see Figure 158):

1. Master generates a START condition.
2. Master sends slave address, with r/w bit set to 0 (“write”). Slave address is obtained from SENS_I2C_SLAVE_ADDR*n*, where *n* is given as an argument to the I2C_WR instruction.
3. Slave generates ACK.
4. Master sends slave register address (given as an argument to the I2C_WR instruction).
5. Slave generates ACK.
6. Master generates a repeated START condition.
7. Master sends slave address, with r/w bit set to 0 (“write”).
8. Master sends one byte of data.
9. Slave generates ACK.

10. Master generates a STOP condition.

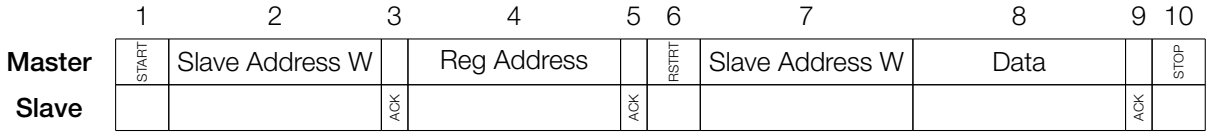


Figure 158: I2C Write Operation

29.6.2.3 Detecting Error Conditions

ULP I2C_RD and I2C_WR instructions will not report error conditions, such as a NACK from a slave, via ULP registers. Instead, applications can query specific bits in the RTC_I2C_INT_ST_REG register to determine if the transaction was successful. To enable checking for specific communication events, their corresponding bits should be set in register RTC_I2C_INT_EN_REG. Note that the bit map is shifted by 1. If a specific communication event is detected and set in register RTC_I2C_INT_ST_REG, it can then be cleared using RTC_I2C_INT_CLR_REG.

29.6.2.4 Connecting I2C Signals

SDA and SCL signals can be mapped onto two out of the four GPIO pins, which are identified in the ESP32 pin lists in *ESP32 Datasheet*, using the RTCIO_SAR_I2C_IO_REG register.

29.7 Register Summary

29.7.1 SENS_ULP Address Space

Name	Description	Address	Access
ULP Timer cycles select			
SENS_ULP_CP_SLEEP_CYC0_REG	Timer cycles setting 0	0x3FF48818	R/W
SENS_ULP_CP_SLEEP_CYC1_REG	Timer cycles setting 1	0x3FF4881C	R/W
SENS_ULP_CP_SLEEP_CYC2_REG	Timer cycles setting 2	0x3FF48820	R/W
SENS_ULP_CP_SLEEP_CYC3_REG	Timer cycles setting 3	0x3FF48824	R/W
SENS_ULP_CP_SLEEP_CYC4_REG	Timer cycles setting 4	0x3FF48828	R/W
RTC I2C slave address select			
SENS_SAR_SLAVE_ADDR1_REG	I2C addresses 0 and 1	0x3FF4883C	R/W
SENS_SAR_SLAVE_ADDR2_REG	I2C addresses 2 and 4	0x3FF48840	R/W
SENS_SAR_SLAVE_ADDR3_REG	I2C addresses 4 and 5	0x3FF48844	R/W
SENS_SAR_SLAVE_ADDR4_REG	I2C addresses 6 and 7, I2C control	0x3FF48848	R/W
RTC I2C control			
SENS_SAR_I2C_CTRL_REG	I2C control registers	0x3FF48850	R/W

29.7.2 RTC_I2C Address Space

Name	Description	Address	Access
RTC I2C control registers			
RTC_I2C_CTRL_REG	Transmission setting	0x3FF48C04	R/W
RTC_I2C_DEBUG_STATUS_REG	Debug status	0x3FF48C08	R/W
RTC_I2C_TIMEOUT_REG	Timeout setting	0x3FF48C0C	R/W
RTC_I2C_SLAVE_ADDR_REG	Local slave address setting	0x3FF48C10	R/W
RTC I2C signal setting registers			
RTC_I2C_SDA_DUTY_REG	Configures the SDA hold time after a negative SCL edge	0x3FF48C30	R/W
RTC_I2C_SCL_LOW_PERIOD_REG	Configures the low level width of SCL	0x3FF48C00	R/W
RTC_I2C_SCL_HIGH_PERIOD_REG	Configures the high level width of SCL	0x3FF48C38	R/W
RTC_I2C_SCL_START_PERIOD_REG	Configures the delay between the SDA and SCL negative edge for a start condition	0x3FF48C40	R/W
RTC_I2C_SCL_STOP_PERIOD_REG	Configures the delay between the SDA and SCL positive edge for a stop condition	0x3FF48C44	R/W
RTC I2C interrupt registers - listed only for debugging			
RTC_I2C_INT_CLR_REG	Clear status of I2C communication events	0x3FF48C24	R/W
RTC_I2C_INT_EN_REG	Enable capture of I2C communication status events	0x3FF48C28	R/W
RTC_I2C_INT_ST_REG	Status of captured I2C communication events	0x3FF48C2C	R/O

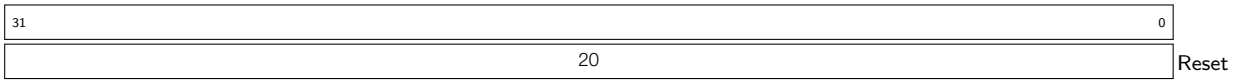
Note:

Interrupts from RTC_I2C are not connected. The interrupt registers above are listed only for debugging purposes.

29.8 Registers

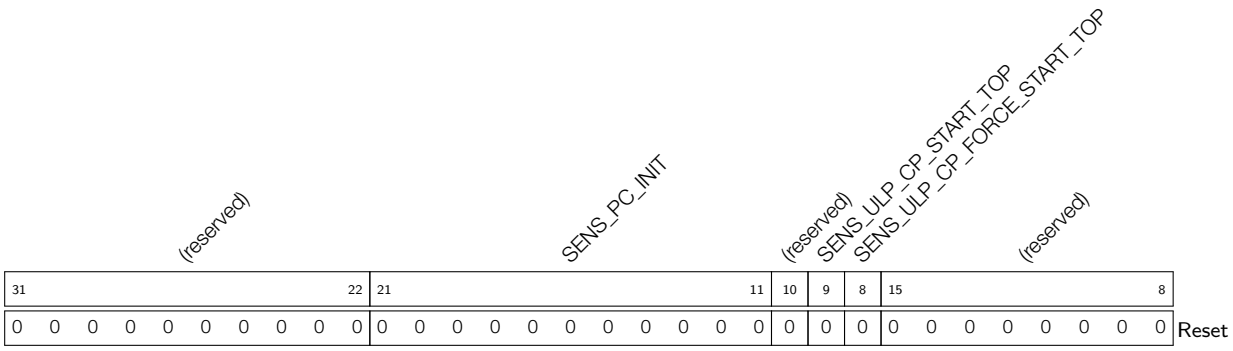
29.8.1 SENS_ULP Address Space

Register 29.1: SENS_ULP_CP_SLEEP_CYC*n*_REG (*n*: 0-4) (0x18+0x4n*)**



SENS_ULP_CP_SLEEP_CYC*n*_REG ULP timer cycles setting *n*; the ULP co-processor can select one of such registers by using the SLEEP instruction. (R/W)

Register 29.2: SENS_SAR_START_FORCE_REG (0x002c)

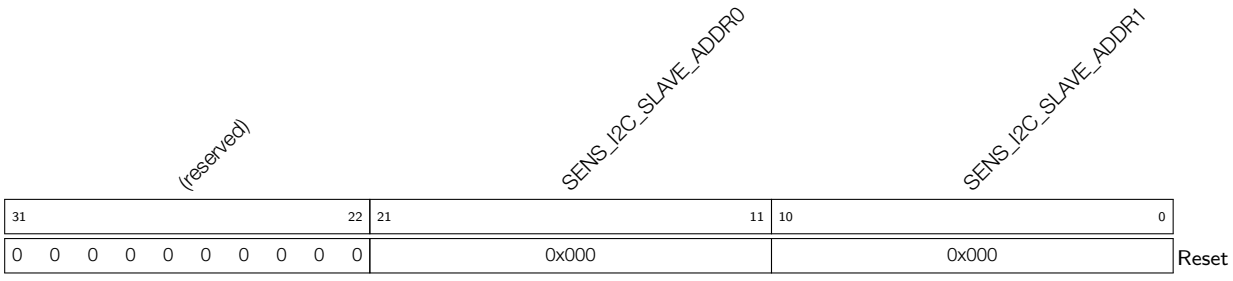


SENS_PC_INIT ULP PC entry address. (R/W)

SENS_ULP_CP_START_TOP Set this bit to start the ULP co-processor; it is active only when SENS_ULP_CP_FORCE_START_TOP = 1. (R/W)

SENS_ULP_CP_FORCE_START_TOP 1: ULP co-processor is started by SENS_ULP_CP_START_TOP; 0: ULP co-processor is started by timer. (R/W)

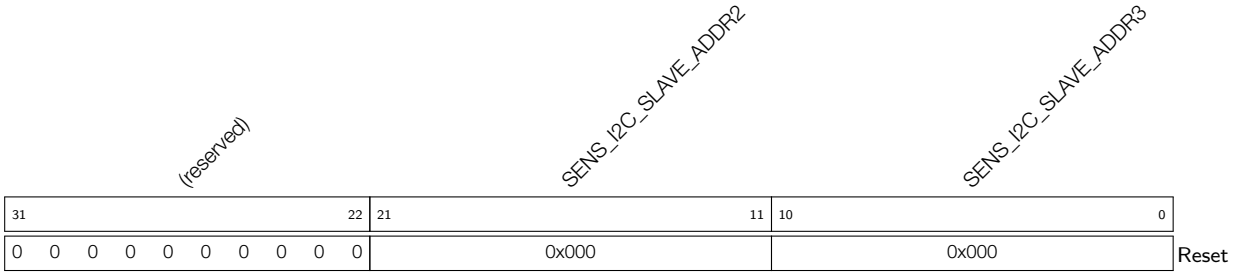
Register 29.3: SENS_SAR_SLAVE_ADDR1_REG (0x003c)



SENS_I2C_SLAVE_ADDR0 I2C slave address 0. (R/W)

SENS_I2C_SLAVE_ADDR1 I2C slave address 1. (R/W)

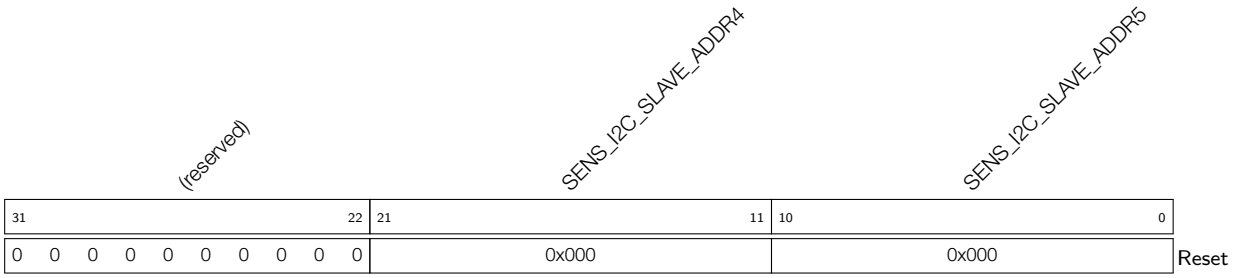
Register 29.4: SENS_SAR_SLAVE_ADDR2_REG (0x0040)



SENS_I2C_SLAVE_ADDR2 I2C slave address 2. (R/W)

SENS_I2C_SLAVE_ADDR3 I2C slave address 3. (R/W)

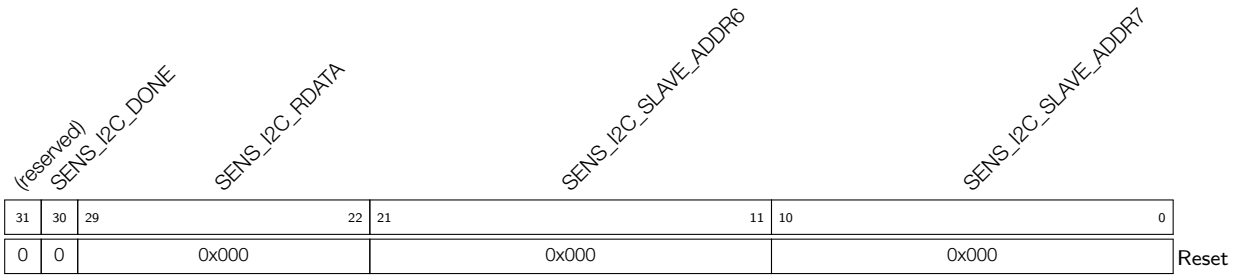
Register 29.5: SENS_SAR_SLAVE_ADDR3_REG (0x0044)



SENS_I2C_SLAVE_ADDR4 I2C slave address 4. (R/W)

SENS_I2C_SLAVE_ADDR5 I2C slave address 5. (R/W)

Register 29.6: SENS_SAR_SLAVE_ADDR4_REG (0x0048)



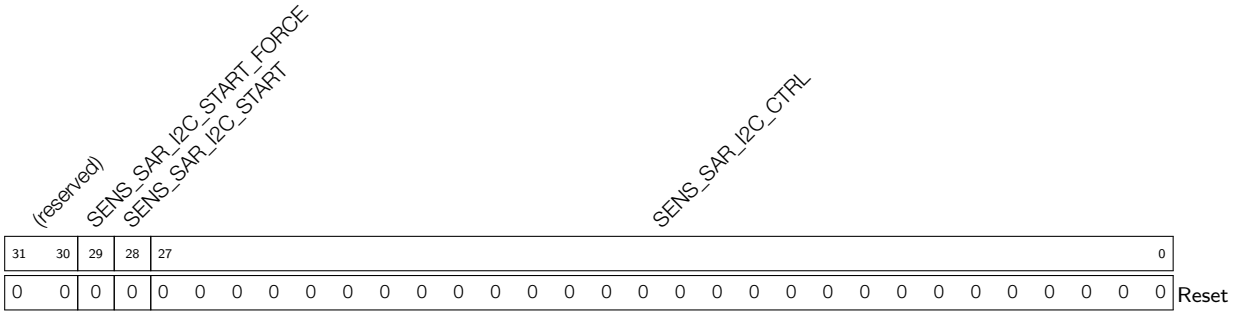
SENS_I2C_DONE Indicate I2C done. (RO)

SENS_I2C_RDATA I2C read data. (RO)

SENS_I2C_SLAVE_ADDR6 I2C slave address 6. (R/W)

SENS_I2C_SLAVE_ADDR7 I2C slave address 7. (R/W)

Register 29.7: SENS_SAR_I2C_CTRL_REG (0x0050)



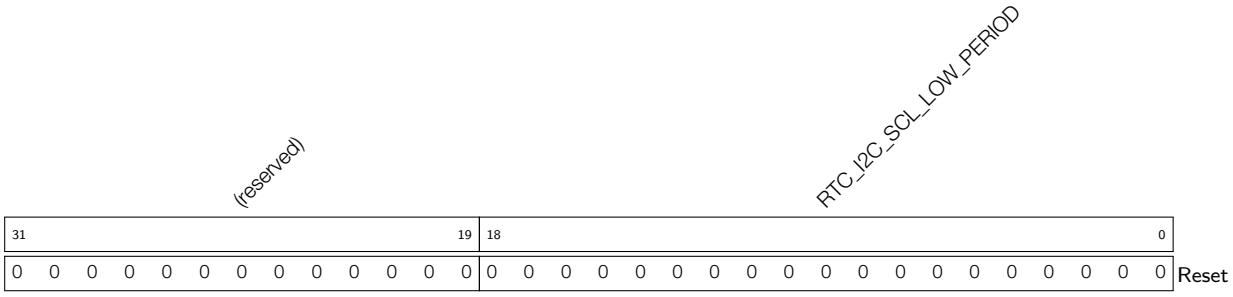
SENS_SAR_I2C_START_FORCE 1: I2C started by SW, 0: I2C started by FSM. (R/W)

SENS_SAR_I2C_START Start I2C; active only when SENS_SAR_I2C_START_FORCE = 1. (R/W)

SENS_SAR_I2C_CTRL I2C control data; active only when SENS_SAR_I2C_START_FORCE = 1. (R/W)

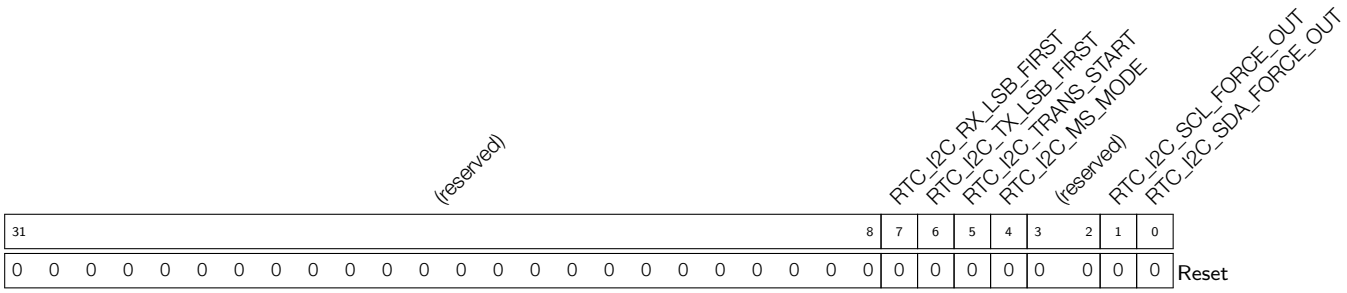
29.8.2 RTC_I2C Address Space

Register 29.8: RTC_I2C_SCL_LOW_PERIOD_REG (0x000)



RTC_I2C_SCL_LOW_PERIOD Number of FAST_CLK cycles when SCL == 0. (R/W)

Register 29.9: RTC_I2C_CTRL_REG (0x004)



RTC_I2C_RX_LSB_FIRST Send LSB first. (R/W)

RTC_I2C_TX_LSB_FIRST Receive LSB first. (R/W)

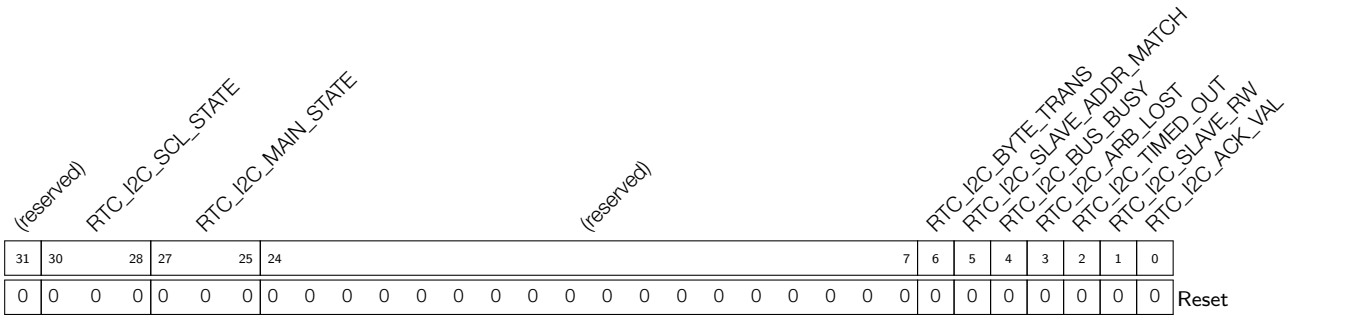
RTC_I2C_TRANS_START Force to generate a start condition. (R/W)

RTC_I2C_MS_MODE Master (1), or slave (0). (R/W)

RTC_I2C_SCL_FORCE_OUT SCL is push-pull (1) or open-drain (0). (R/W)

RTC_I2C_SDA_FORCE_OUT SDA is push-pull (1) or open-drain (0). (R/W)

Register 29.10: RTC_I2C_DEBUG_STATUS_REG (0x008)



RTC_I2C_SCL_STATE State of SCL machine. (R/W)

RTC_I2C_MAIN_STATE State of the main machine. (R/W)

RTC_I2C_BYTE_TRANS 8-bit transmit done. (R/W)

RTC_I2C_SLAVE_ADDR_MATCH Indicates whether the addresses are matched, when in slave mode. (R/W)

RTC_I2C_BUS_BUSY Operation is in progress. (R/W)

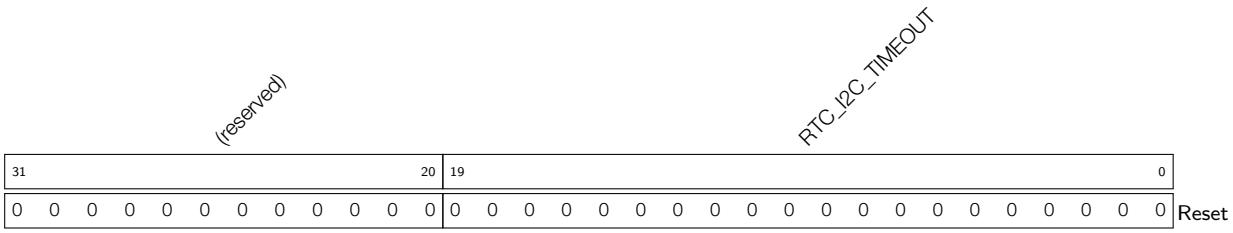
RTC_I2C_ARB_LOST Indicates the loss of I2C bus control, when in master mode. (R/W)

RTC_I2C_TIMED_OUT Transfer has timed out. (R/W)

RTC_I2C_SLAVE_RW Indicates the value of the received R/W bit, when in slave mode. (R/W)

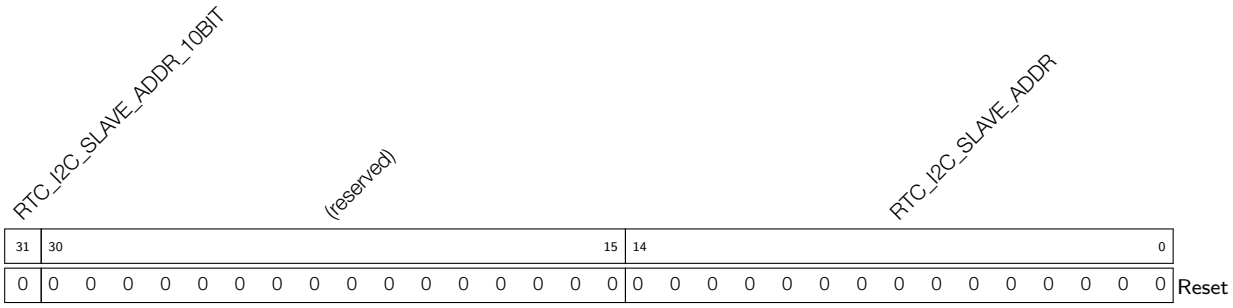
RTC_I2C_ACK_VAL The value of ACK signal on the bus. (R/W)

Register 29.11: RTC_I2C_TIMEOUT_REG (0x00c)



RTC_I2C_TIMEOUT Maximum number of FAST_CLK cycles that the transmission can take. (R/W)

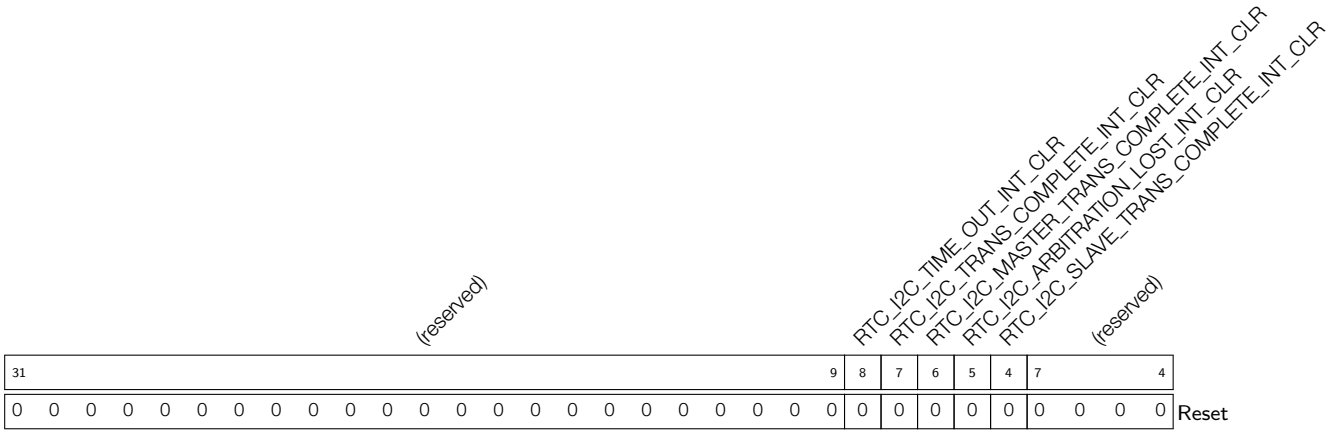
Register 29.12: RTC_I2C_SLAVE_ADDR_REG (0x010)



RTC_I2C_SLAVE_ADDR_10BIT Set if local slave address is 10-bit. (R/W)

RTC_I2C_SLAVE_ADDR Local slave address. (R/W)

Register 29.13: RTC_I2C_INT_CLR_REG (0x024)



RTC_I2C_TIME_OUT_INT_CLR Clear interrupt upon timeout. (R/W)

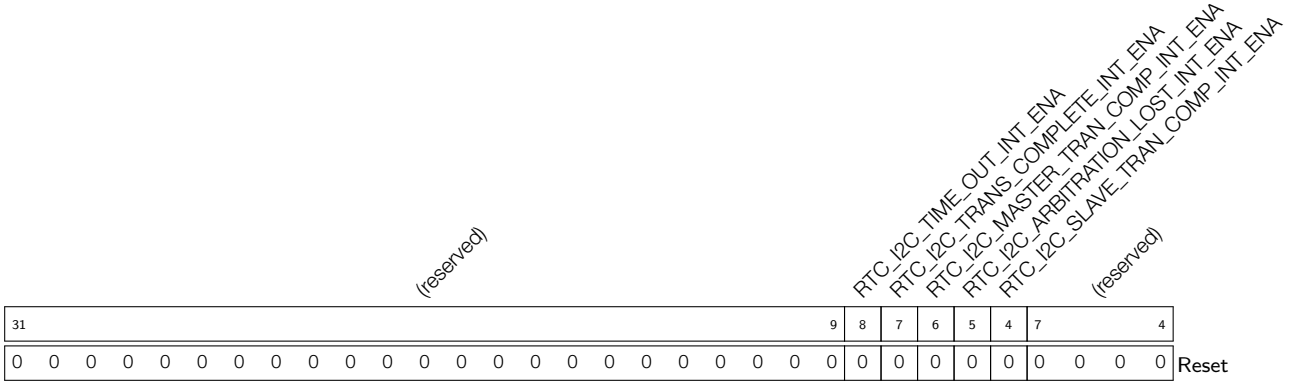
RTC_I2C_TRANS_COMPLETE_INT_CLR Clear interrupt upon detecting a stop pattern. (R/W)

RTC_I2C_MASTER_TRANS_COMPLETE_INT_CLR Clear interrupt upon completion of transaction, when in master mode. (R/W)

RTC_I2C_ARBITRATION_LOST_INT_CLR Clear interrupt upon losing control of the bus, when in master mode. (R/W)

RTC_I2C_SLAVE_TRANS_COMPLETE_INT_CLR Clear interrupt upon completion of transaction, when in slave mode. (R/W)

Register 29.14: RTC_I2C_INT_EN_REG (0x028)



RTC_I2C_TIME_OUT_INT_ENA Enable interrupt upon timeout. (R/W)

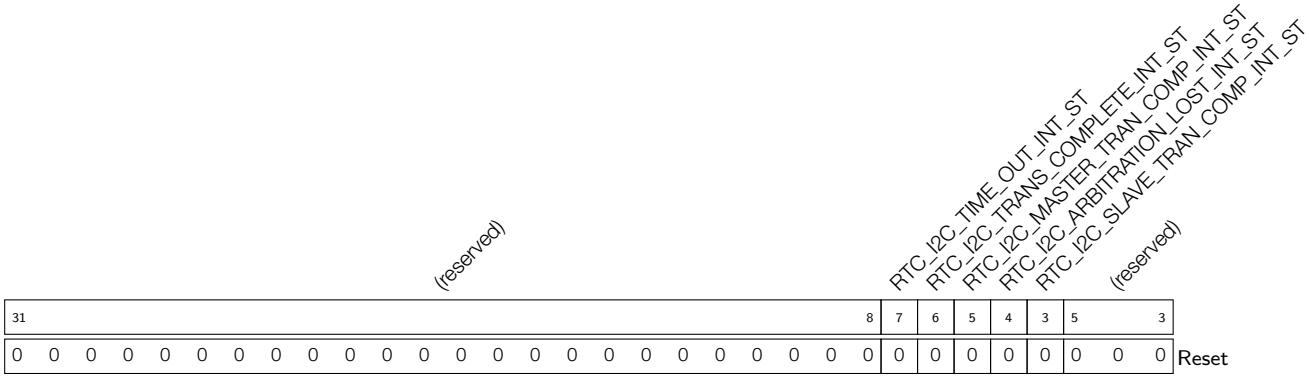
RTC_I2C_TRANS_COMPLETE_INT_ENA Enable interrupt upon detecting a stop pattern. (R/W)

RTC_I2C_MASTER_TRAN_COMP_INT_ENA Enable interrupt upon completion of transaction, when in master mode. (R/W)

RTC_I2C_ARBITRATION_LOST_INT_ENA Enable interrupt upon losing control of the bus, when in master mode. (R/W)

RTC_I2C_SLAVE_TRAN_COMP_INT_ENA Enable interrupt upon completion of transaction, when in slave mode. (R/W)

Register 29.15: RTC_I2C_INT_ST_REG (0x02c)



RTC_I2C_TIME_OUT_INT_ST Detected timeout. (R/O)

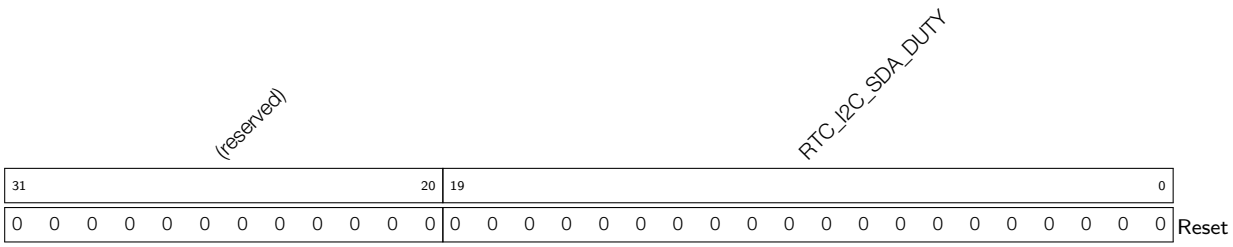
RTC_I2C_TRANS_COMPLETE_INT_ST Detected stop pattern on I2C bus. (R/O)

RTC_I2C_MASTER_TRAN_COMP_INT_ST Transaction completed, when in master mode. (R/O)

RTC_I2C_ARBITRATION_LOST_INT_ST Bus control lost, when in master mode. (R/O)

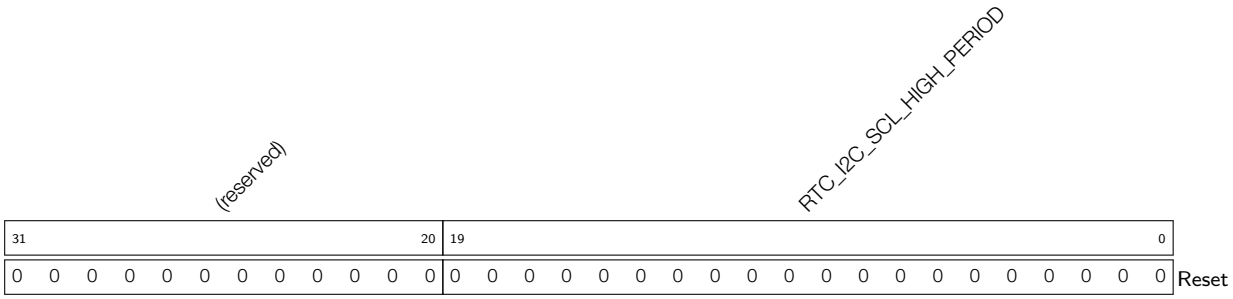
RTC_I2C_SLAVE_TRAN_COMP_INT_ST Transaction completed, when in slave mode. (R/O)

Register 29.16: RTC_I2C_SDA_DUTY_REG (0x030)



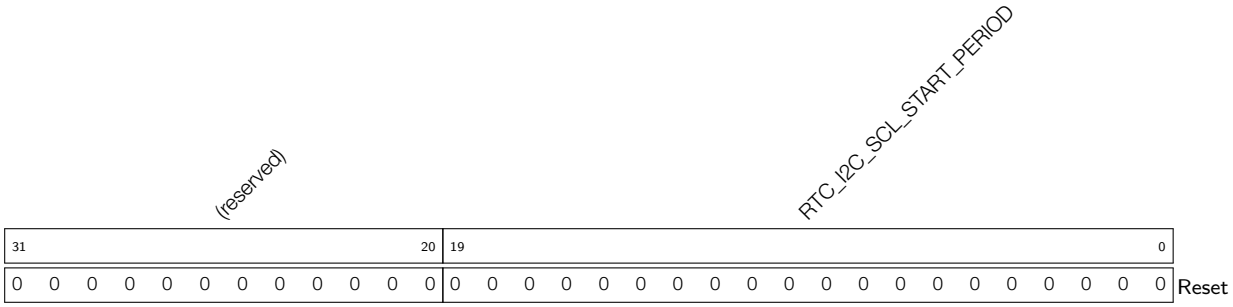
RTC_I2C_SDA_DUTY Number of FAST_CLK cycles between the SDA switch and the falling edge of SCL. (R/W)

Register 29.17: RTC_I2C_SCL_HIGH_PERIOD_REG (0x038)



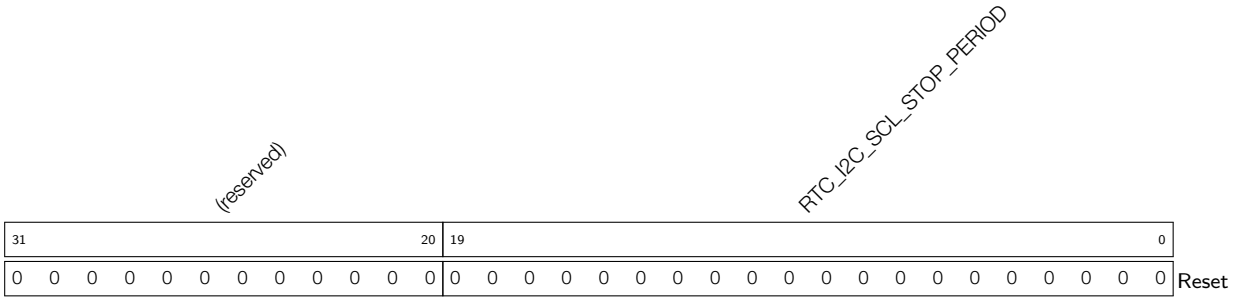
RTC_I2C_SCL_HIGH_PERIOD Number of FAST_CLK cycles when SCL == 1. (R/W)

Register 29.18: RTC_I2C_SCL_START_PERIOD_REG (0x040)



RTC_I2C_SCL_START_PERIOD Number of FAST_CLK cycles to wait before generating a start condition. (R/W)

Register 29.19: RTC_I2C_SCL_STOP_PERIOD_REG (0x044)



RTC_I2C_SCL_STOP_PERIOD Number of FAST_CLK cycles to wait before generating a stop condition. (R/W)

30. Low-Power Management

30.1 Introduction

ESP32 offers efficient and flexible power-management technology to achieve the best balance between power consumption, wakeup latency and available wakeup sources. Users can select out of five predefined power modes of the main processors to suit specific needs of the application. In addition, to save power in power-sensitive applications, control may be executed by the Ultra-Low-Power co-processor (ULP co-processor), while the main processors are in Deep-sleep mode.

30.2 Features

- Five predefined power modes to support various applications
- Up to 16 KB of retention memory
- 8 x 32 bits of retention registers
- ULP co-processor enabled in all low-power modes
- RTC boot supported to shorten the wakeup latency

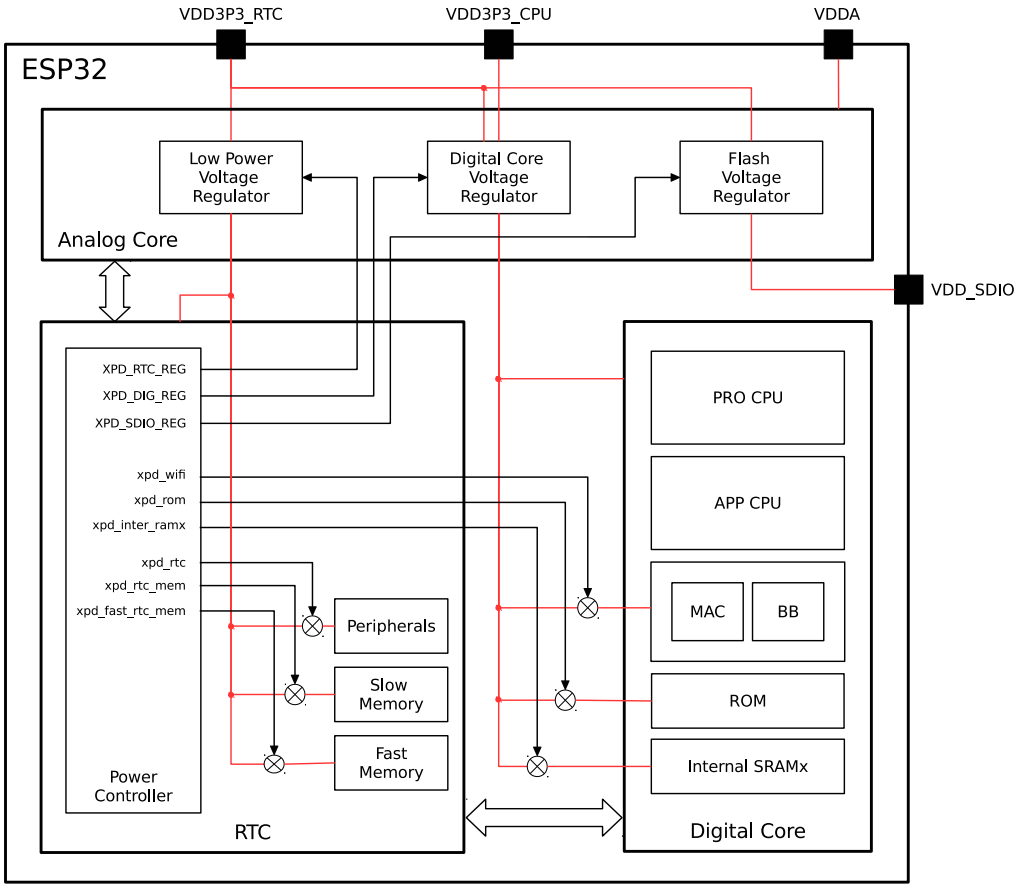


Figure 159: ESP32 Power Control

30.3 Functional Description

30.3.1 Overview

The low-power management unit includes voltage regulators, a power controller, power switch cells, power domain isolation cells, etc. Figure 159 shows the high-level architecture of ESP32's low-power management.

30.3.2 Digital Core Voltage Regulator

The built-in voltage regulator can convert the external power supply (typically 3.3V) to 1.1V to support the internal digital core. It receives a wide range of external power supply from 1.8V to 3.6V, and provides an output voltage from 0.90V to 1.25V.

1. When XPD_DIG_REG == 1, the regulator outputs a 1.1V voltage and the digital core is able to run; when XPD_DIG_REG == 0, both the regulator and the digital core stop running.
2. DIG_REG_DBIAS[2:0] tunes the supply voltage of the digital core:

$$\text{VDD\_DIG} = 0.90 + \text{DBIAS} \cdot 0.05\text{V}$$

3. The current to the digital core comes from pin VDD3P3_CPU and pin VDD3P3_RTC.

Figure 160 shows the structure of a digital core's voltage regulator.

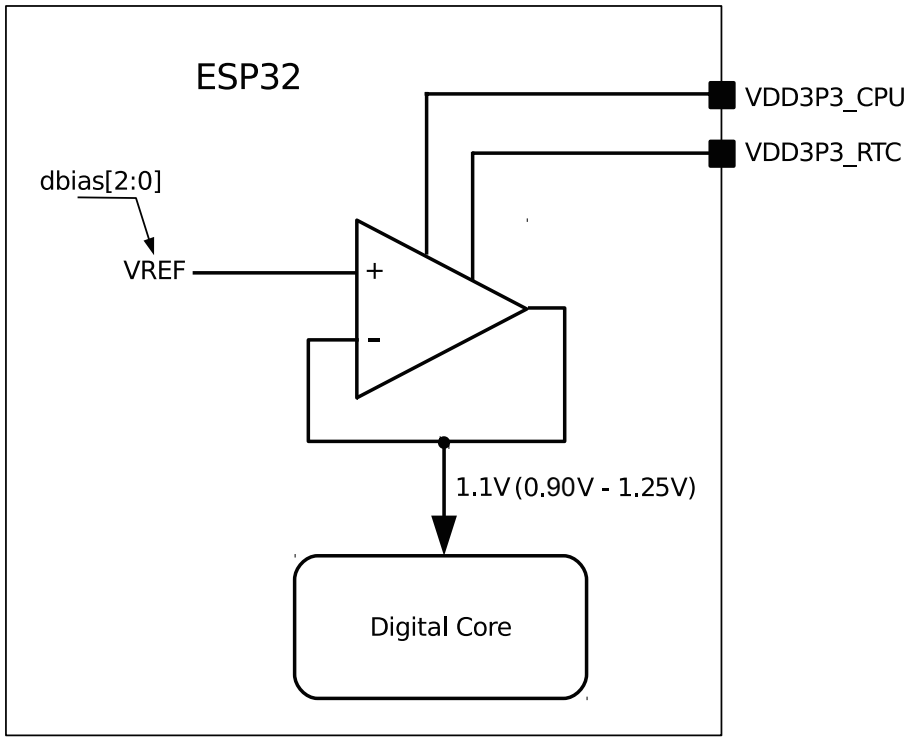


Figure 160: Digital Core Voltage Regulator

30.3.3 Low-Power Voltage Regulator

The built-in low-power voltage regulator can convert the external power supply (typically 3.3V) to 1.1V to support the internal RTC core. To save power, it receives a wide range of external power supply from 1.8V to 3.6V, and

supports an adjustable output voltage of 0.90V to 1.25V in normal work mode, a fixed output voltage of about 0.75V both in Deep-sleep mode and Hibernation mode.

1. When the pin CHIP_PU is at a high level, the low-power voltage regulator cannot be turned off. It should be switched only between normal-work mode and Deep-sleep mode.
2. In normal-work mode, RTC_DBIAS[2:0] can be used to tune the output voltage:

$$\text{VDD\_RTC} = 0.90 + \text{DBIAS} \cdot 0.05\text{V}$$

3. In Deep-sleep mode, the output voltage of the regulator is fixed at about 0.75V.
4. The current to the RTC core comes from pin VDD3P3_RTC.

Figure 161 shows the structure of a low-power voltage regulator.

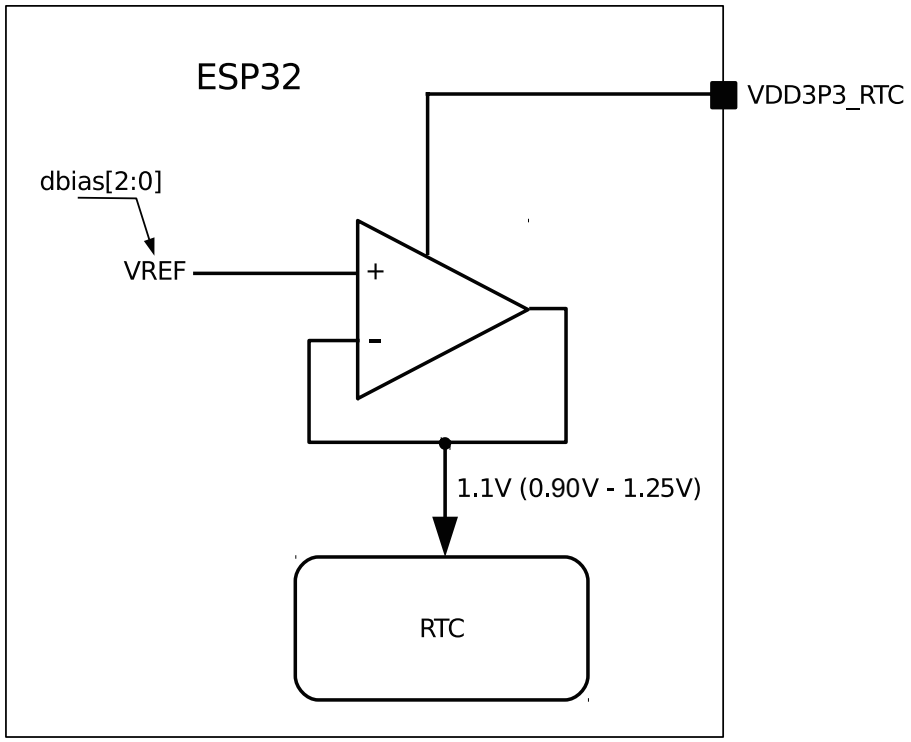


Figure 161: Low-Power Voltage Regulator

30.3.4 Flash Voltage Regulator

The built-in flash voltage regulator can supply a voltage of 3.3V or 1.8V to other devices (flash, for example) in the system, with a maximum output current of 40 mA.

1. When XPD_SDIO_VREG == 1, the regulator outputs a voltage of 3.3V or 1.8V; when XPD_SDIO_VREG == 0, the output is high-impedance and, in this case, the voltage is provided by the external power supply.
2. When SDIO_TIEH == 1, the regulator shorts pin VDD_SDIO to pin VDD3P3_RTC. The regulator then outputs a voltage of 3.3V which is the voltage of pin VDD3P3_RTC. When SDIO_TIEH == 0, the inner loop ties the regulator output to the voltage of VREF, which is typically 1.8V.
3. DREFH_SDIO, DREFM_SDIO and DREFL_SDIO could be used to tune the reference voltage VREF slightly. However, it is recommended that users do not change the value of these registers, since it may affect the stability of the inner loop.

4. When the regulator output is 3.3V or 1.8V, the output current comes from the pin VDD3P3_RTC.

Figure 162 shows the structure of a flash voltage regulator.

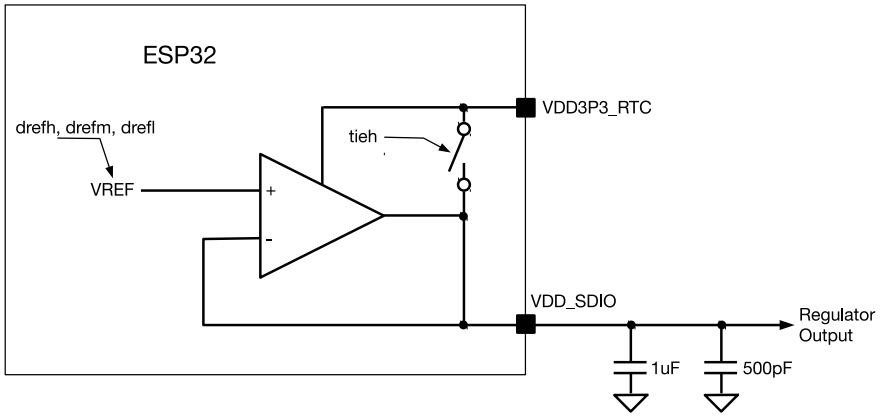


Figure 162: Flash Voltage Regulator

30.3.5 Brownout Detector

The brownout detector checks the voltage of pin VDD3P3_RTC. If the voltage drops rapidly and becomes too low, the detector would trigger a signal to shut down some power-consuming blocks (such as LNA, PA, etc.) to allow extra time for the digital block to save and transfer important data. The power consumption of the detector is ultra low. It remains enabled whenever the chip is powered on, with an adjustable trigger level calibrated around 2.5V.

1. As the output of the brownout detector, RTC_CNTL_BROWN_OUT_DET goes high when the voltage of pin VDD3P3_RTC is lower than the threshold value.
2. RTC_CNTL_DBROWN_OUT_THRES[2:0] is used to tune the threshold voltage, which is usually calibrated around 2.5V.

Figure 163 shows the structure of a brownout detector.

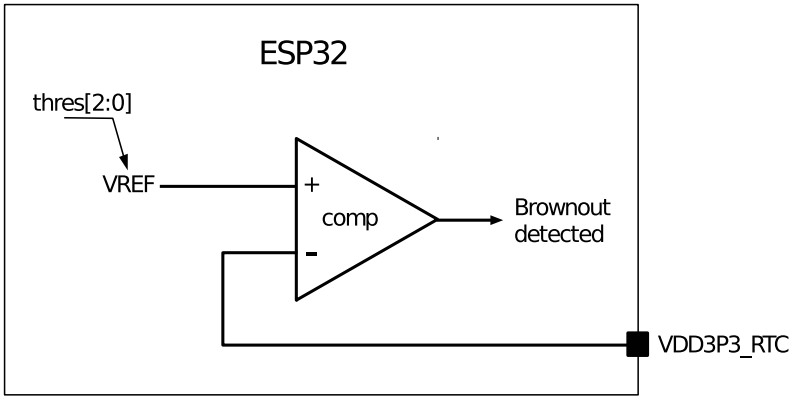


Figure 163: Brownout Detector

30.3.6 RTC Module

The RTC module is designed to handle the entry into, and exit from, the low-power mode, and control the clock sources, PLL, power switch and isolation cells to generate power-gating, clock-gating, and reset signals. As for

the low-power management, RTC is composed of the following modules (see Figure 164):

- RTC main state machine: records the power state.
- Digital & analog power controller: generates actual power-gating/clock-gating signals for digital parts and analog parts.
- Sleep & wakeup controller: handles the entry into & exit from the low-power mode.
- Timers: include RTC main timer, ULP co-processor timer and touch timer.
- Low-Power processor and sensor controllers: include ULP co-processor, touch controller, SAR ADC controller, etc.
- Retention memory:
  - RTC slow memory: an 8 KB SRAM, mostly used as retention memory or instruction & data memory for the ULP co-processor. The CPU accesses it through the APB, starting from address 0x50000000.
  - RTC fast memory: an 8 KB SRAM, mostly used as retention memory. The CPU accesses it through IRAM0/DRAM0. Fast RTC memory is about 10 times faster than the RTC slow memory.
- Retention registers: always-on registers of 8 x 32 bits, serving as data storage.
- RTC IO pads: 18 always-on analog pads, usually functioning as wake-up sources.

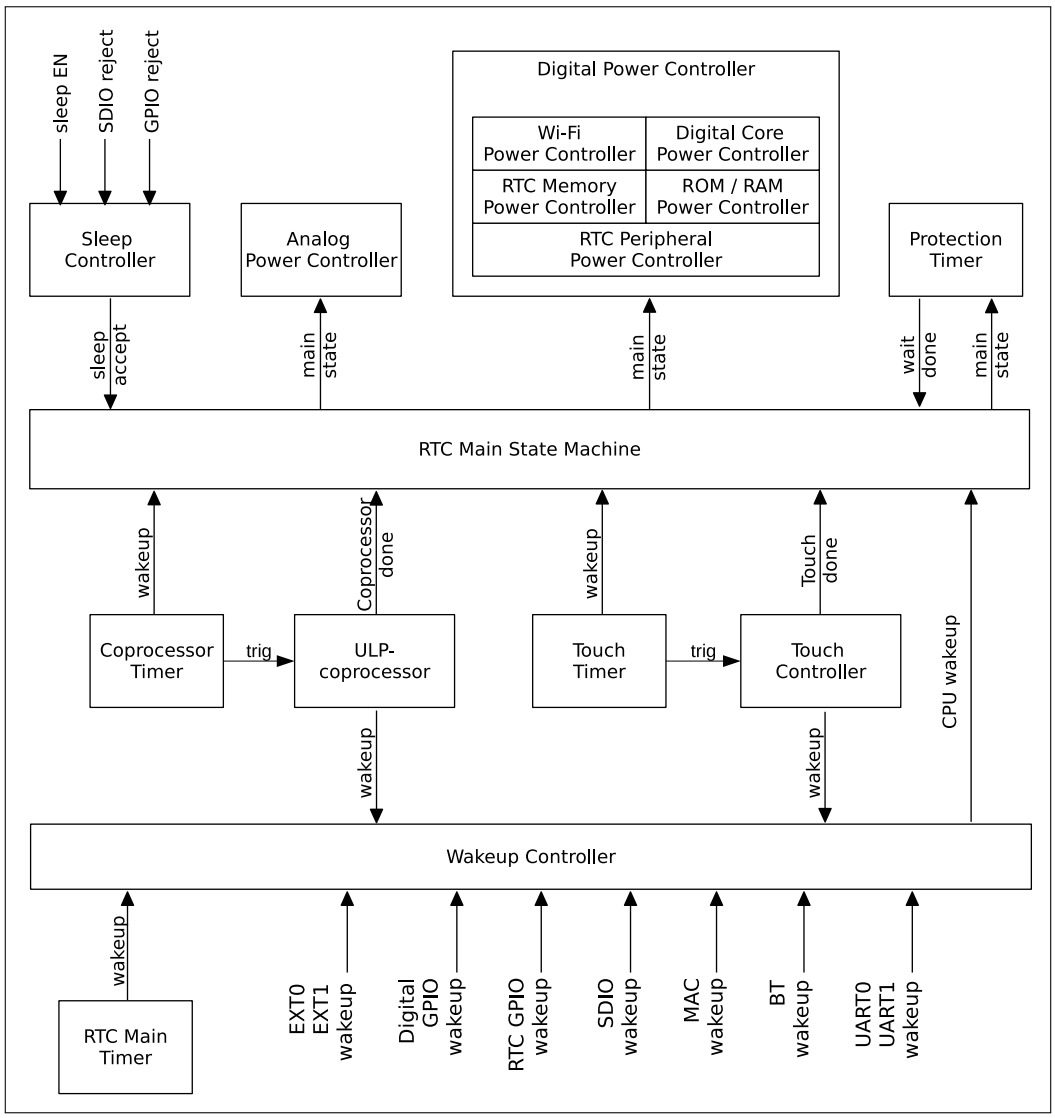


Figure 164: RTC Structure

30.3.7 Low-Power Clocks

In the low-power mode, the 40 MHz crystal and PLL are usually powered down to save power. But clocks are needed for the chip to remain active in the low-power mode.

For the RTC core, there are five possible clock sources:

- external low-speed (32.768 kHz) crystal clock CK_XTAL_32K,
- external high-speed (2 MHz ~ 40 MHz) crystal clock CK_40M_DIG,
- internal RC oscillator SLOW_CK (typically about 150 kHz and adjustable),
- internal 8-MHz oscillator CK8M_OUT, and
- internal 31.25-kHz clock CK8M_D256_OUT (derived from the internal 8-MHz oscillator divided by 256).

With these clocks, fast_rtc_clk and slow_rtc_clk is derived. By default, fast_rtc_clk is CK8M_OUT while slow_rtc_clk is SLOW_CK. For details, please see Figure 165.

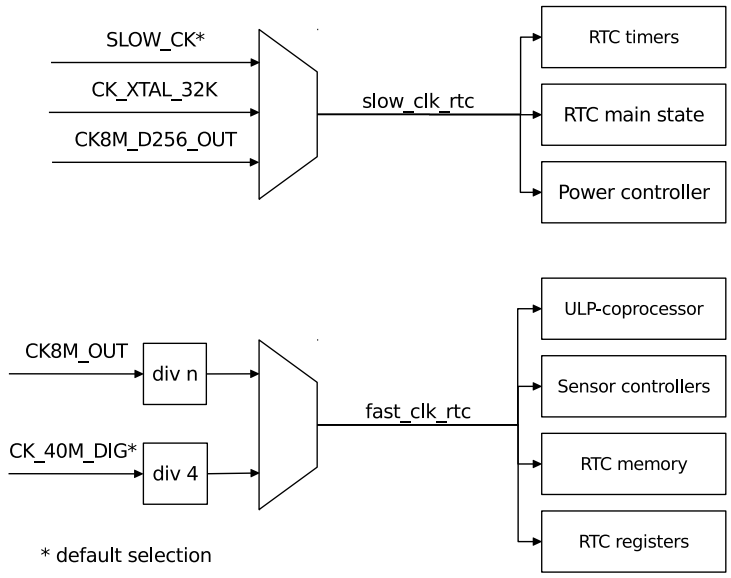


Figure 165: RTC Low-Power Clocks

For the digital core, low_power_clk is switched among four sources. For details, please see Figure 166.

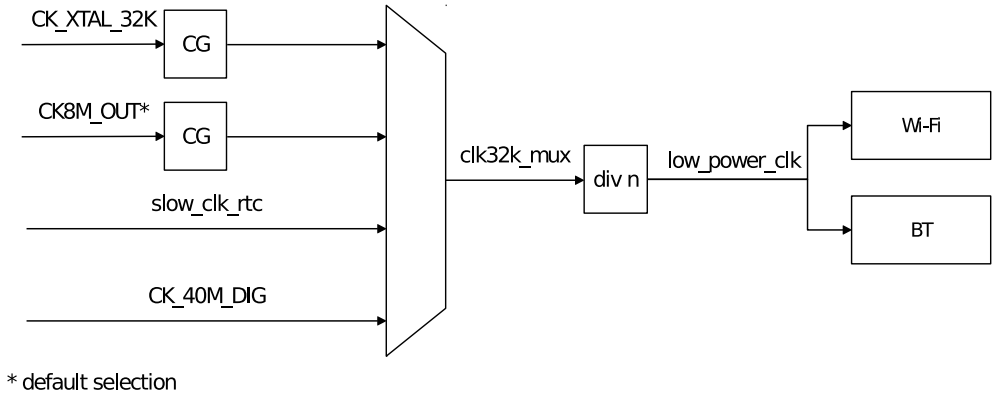


Figure 166: Digital Low-Power Clocks

30.3.8 Power-Gating Implementation

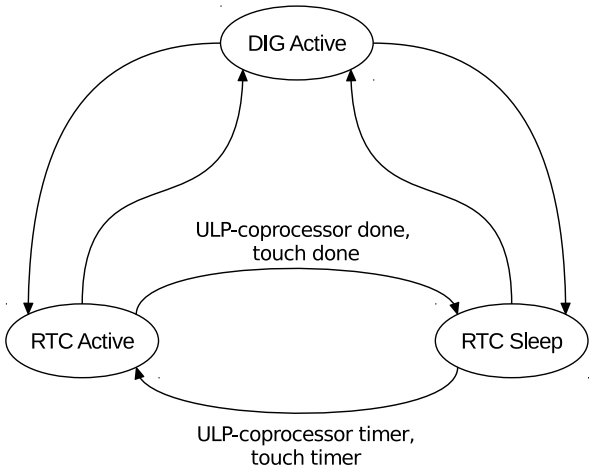


Figure 167: RTC States

The switch among power-gating states can be see in Figure 167. The actual power-control signals could also be set by software as force-power-up (FPU) or force-power-down (FPD). Since the power domains can be power-gated independently, there are many combinations for different applications. Table 132 shows how the power domains in ESP32 are controlled.

Table 132: RTC Power Domains

Power Domains		RTC Main State			S/W Options		Notes*
		DIG Active	RTC Active	RTC Sleep	FPU	FPD	
RTC	RTC Digital Core	ON	ON	ON	N	N	1
	RTC Peripherals	ON	ON	OFF	Y	Y	2
	RTC Slow Memory	ON	OFF	OFF	Y	Y	3
	RTC Fast Memory	ON	OFF	OFF	Y	Y	4
Digital	Digital Core	ON	OFF	OFF	Y	Y	5
	Wi-Fi	ON	OFF	OFF	Y	Y	6
	ROM	ON	OFF	OFF	Y	Y	-
	Internal SRAM	ON	OFF	OFF	Y	Y	7
Analog	40 MHz Crystal	ON	OFF	OFF	Y	Y	-
	PLL	ON	OFF	OFF	Y	Y	-
	8 MHz OSC	ON	OFF	OFF	Y	Y	-
	Radio	-	-	-	Y	Y	-

Notes*:

1. The power-domain RTC core is the “always-on” power domain, and the FPU/FPD option is not available.
2. The power-domain RTC peripherals include most of the fast logic in RTC, including the ULP co-processor, sensor controllers, etc.
3. The power-domain RTC slow memory should be forced to power on when it is used as retention memory, or when the ULP co-processor is working.
4. The power-domain RTC fast memory should be forced to power on, when it is used as retention memory.
5. When the power-domain digital core is powered down, all included in power domains are powered down.
6. The power-domain Wi-Fi includes the Wi-Fi MAC and BB.
7. Each internal SRAM can be power-gated independently.

30.3.9 Predefined Power Modes

In ESP32, we recommend that you always use the predefined power modes first, before trying to tune each power control signal. The predefined power modes should cover most scenarios:

- Active mode
  - The CPU is clocked at XTAL_DIV_N (40 MHz/26 MHz) or PLL (80 MHz/160 MHz/240 MHz).
  - The chip can receive, transmit, or listen.
- Modem-sleep mode
  - The CPU is operational and the clock is configurable.
  - The Wi-Fi/Bluetooth baseband is clock-gated or powered down. The radio is turned off.

  - Current consumption: ~30 mA with 80 MHz PLL.
  - Current consumption: ~3 mA with 2 MHz XTAL.
  - Immediate wake-up.
- Light-sleep mode
  - The internal 8 MHz oscillator, 40 MHz high-speed crystal, PLL, and radio are disabled.
  - The clock in the digital core is gated. The CPUs are stalled.
  - The ULP co-processor and touch controller can be periodically triggered by monitor sensors.
  - Current consumption: ~ 800 μA.
  - Wake-up latency: less than 1 ms.
- Deep-sleep mode
  - The internal 8 MHz oscillator, 40 MHz high-speed crystal, PLL and radio are disabled.
  - The digital core is powered down. The CPU context is lost.
  - The supply voltage to the RTC core drops to 0.7V.
  - 8 x 32 bits of data are kept in general-purpose retention registers.
  - The RTC memory and fast RTC memory can be retained.
  - Current consumption: ~ 6.5 μA.
  - Wake-up latency: less than 1 ms.
  - Recommended for ultra-low-power infrequently-connected Wi-Fi/Bluetooth applications.
- Hibernatation mode
  - The internal 8 MHz oscillator, 40 MHz high-speed crystal, PLL, and radio are disabled.
  - The digital core is powered down. The CPU context is lost.
  - The RTC peripheral domain is powered down.
  - The supply voltage to the RTC core drops to 0.7V.
  - 8 x 32 bits of data are kept in general-purpose retention registers.
  - The RTC memory and fast RTC memory are powered down.
  - Current consumption: ~ 4.5 μA.
  - Wake-up source: RTC timer only.
  - Wake-up latency: less than 1 ms.
  - Recommended for ultra-low-power infrequently-connected Wi-Fi/Bluetooth applications.

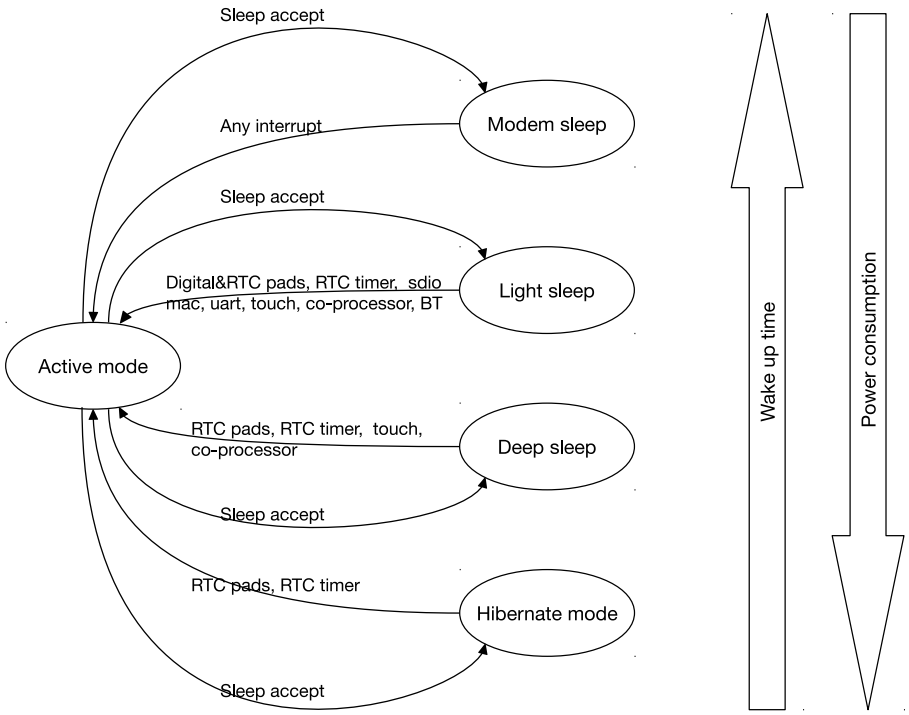


Figure 168: Power Modes

By default, the ESP32 is in active mode after a system reset.There are several low-power modes for saving power when the CPU does not need to be kept running, for example, when waiting for an external event. It is up to the user to select the mode that best balances power consumption, wake-up latency and available wake-up sources. For details, please see Figure 168.

Please note that the predefined power mode could be further optimized and adapted to any application.

30.3.10 Wakeup Source

The ESP32 supports various wake-up sources, which could wake up the CPU in different sleep modes. The wake-up source is determined by RTC_CNTL_WAKEUP_ENA, as shown in Table 133.

Table 133: Wake-up Source

WAKEUP_ENA	Wake-up Source	Light-sleep	Deep-sleep	Hibernation	Notes*
0x1	EXT0	Y	Y	-	1
0x2	EXT1	Y	Y	Y	2
0x4	GPIO	Y	Y	-	3
0x8	RTC timer	Y	Y	Y	-
0x10	SDIO	Y	-	-	4
0x20	Wi-Fi	Y	-	-	5
0x40	UART0	Y	-	-	6
0x80	UART1	Y	-	-	6
0x100	TOUCH	Y	Y	-	-
0x200	ULP co-proccesor	Y	Y	-	-
0x400	BT	Y	-	-	5

Notes*:

1. EXT0 can only wake up the chip in light-sleep/deep-sleep mode. If RTC_CNTL_EXT_WAKEUP0_LV is 1, it is

pad high-level triggered; otherwise, it is low-level triggered. Users can set RTCIO_EXT_WAKEUP0_SEL[4:0] to select one of the RTC PADs to be the wake-up source.

2. EXT1 is especially designed to wake up the chip from any sleep mode, and it also supports multiple pads' combinations. First, RTC_CNTL_EXT_WAKEUP1_SEL[17:0] should be configured with the bitmap of PADS selected as a wake-up source. Then, if RTC_CNTL_EXT_WAKEUP1_LV is 1, as long as one of the PADs is at high-voltage level, it can trigger a wake-up. However, if RTC_CNTL_EXT_WAKEUP1_LV is 0, it needs all selected PADs to be at low-voltage level to trigger a wake-up.

3. In Deep-sleep mode, only RTC GPIOs (not DIGITAL GPIOs) can work as wakeup source.

4. Wake-up is triggered by receiving any SDIO command.

5. To wake up the chip with a Wi-Fi or BT source, the power mode switches between the Active, Modem- and Light-sleep modes. The CPU, Wi-Fi, Bluetooth, and radio are woken up at predetermined intervals to keep Wi-Fi/BT connections active.

6. Wake-up is triggered when the number of RX pulses received is greater than the value stored in the threshold register.

30.3.11 RTC Timer

The RTC timer is a 48-bit counter that can be read. The clock is RTC_SLOW_CLK. Any reset/sleep mode, except for the power-up reset, will not stop or reset the RTC timer.

The RTC timer can be used to wake up the CPU at a designated time, and to wake up TOUCH or the ULP co-processor periodically.

30.3.12 RTC Boot

Since the CPU, ROM and RAM are powered down during Deep-sleep and Hibernation mode, the wake-up time is much longer than that in Light sleep/Modem sleep, because of the ROM unpacking and data-copying from the flash (SPI booting). There are two types of SRAM in the RTC, named slow RTC memory and fast RTC memory, which remain powered-on in Deep-sleep mode. For small-scale codes (less than 8 KB), there are two methods of speeding up the wake-up time, i.e. avoiding ROM unpacking and SPI booting.

The first method is to use the RTC slow memory:

1. Set register RTC_CNTL_PROCPU_STAT_VECTOR_SEL for PRO_CPU (or register RTC_CNTL_APPCPU_STAT_VECTOR_SEL for APP_CPU) to 0.
2. Put the chip into sleep.
3. When the CPU is powered up, the reset vector starts from 0x50000000, instead of 0x40000400. ROM unpacking & SPI boot are not needed. The code in RTC memory has to do itself some initialization for the C program environment.

The second method is to use the fast RTC memory:

1. Set register RTC_CNTL_PROCPU_STAT_VECTOR_SEL for PRO_CPU (or register RTC_CNTL_APPCPU_STAT_VECTOR_SEL for APP_CPU) to 1.
2. Calculate CRC for the fast RTC memory, and save the result in register RTC_CNTL_RTC_STORE6_REG[31:0].
3. Input register RTC_CNTL_RTC_STORE7_REG[31:0] with the entry address in the fast RTC memory.

4. Put the chip into sleep.
5. When the CPU is powered up, after ROM unpacking and some necessary initialization, the CRC is calculated again. If the result matches with register RTC_CNTL_RTC_STORE6_REG[31:0], the CPU will jump to the entry address.

The boot flow is shown in Figure 169.

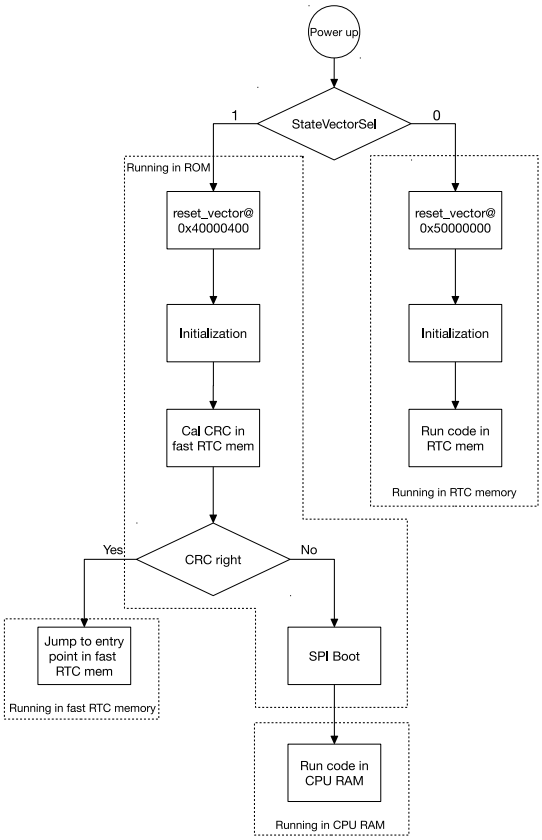


Figure 169: ESP32 Boot Flow

30.4 Register Summary

Notes:

- The registers listed below have been grouped according to their functionality. This particular grouping does not reflect the exact sequential order in which they are stored in memory.
- The base address for registers is 0x60008000 when accessed by AHB, and 0x3FF48000 when accessed by DPORT bus.

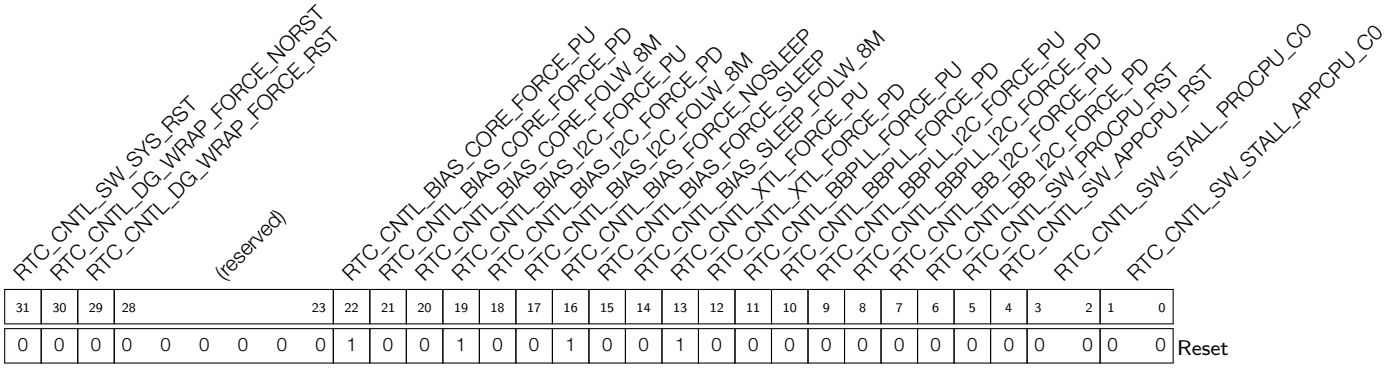
Name	Description	Address	Access
RTC option register			
RTC_CNTL_OPTIONS0_REG	Configure RTC options	0x3FF48000	R/W
Control and configuration of RTC timer registers			
RTC_CNTL_SLP_TIMER0_REG	RTC sleep timer	0x3FF48004	R/W
RTC_CNTL_SLP_TIMER1_REG	RTC sleep timer, alarm and control	0x3FF48008	R/W
RTC_CNTL_TIME_UPDATE_REG	Update control of RTC timer	0x3FF4800C	RO
RTC_CNTL_TIME0_REG	RTC timer low 32 bits	0x3FF48010	RO

Name	Description	Address	Access
RTC_CNTL_TIME1_REG	RTC timer high 16 bits	0x3FF48014	RO
RTC_CNTL_STATE0_REG	RTC sleep, SDIO and ULP control	0x3FF48018	R/W
RTC_CNTL_TIMER1_REG	CPU stall enable	0x3FF4801C	R/W
RTC_CNTL_TIMER2_REG	Slow clock and touch controller configuration	0x3FF48020	R/W
RTC_CNTL_TIMER5_REG	Minimal sleep cycles in slow clock	0x3FF4802C	R/W
Reset state and wakeup control registers			
RTC_CNTL_RESET_STATE_REG	Reset state control and cause of CPUs	0x3FF48034	RO
RTC_CNTL_WAKEUP_STATE_REG	Wake-up filter, enable and cause	0x3FF48038	RO
RTC_CNTL_EXT_WAKEUP_CONF_REG	Configuration of wake-up at low/high level	0x3FF48060	R/W
RTC_CNTL_EXT_WAKEUP1_REG	Selection of pads for external wake-up and wake-up clear bit	0x3FF480CC	R/W
RTC_CNTL_EXT_WAKEUP1_STATUS_REG	External wake-up status	0x3FF480D0	RO
RTC interrupt control and status registers			
RTC_CNTL_INT_ENA_REG	Interrupt enable bits	0x3FF4803C	R/W
RTC_CNTL_INT_RAW_REG	Raw interrupt status	0x3FF48040	RO
RTC_CNTL_INT_ST_REG	Masked interrupt status	0x3FF48044	RO
RTC_CNTL_INT_CLR_REG	Interrupt clear bits	0x3FF48048	WO
RTC general purpose retention registers			
RTC_CNTL_STORE0_REG	General purpose retention register 0	0x3FF4804C	R/W
RTC_CNTL_STORE1_REG	General purpose retention register 1	0x3FF48050	R/W
RTC_CNTL_STORE2_REG	General purpose retention register 2	0x3FF48054	R/W
RTC_CNTL_STORE3_REG	General purpose retention register 3	0x3FF48058	R/W
RTC_CNTL_STORE4_REG	General purpose retention register 4	0x3FF480B0	R/W
RTC_CNTL_STORE5_REG	General purpose retention register 5	0x3FF480B4	R/W
RTC_CNTL_STORE6_REG	General purpose retention register 6	0x3FF480B8	R/W
RTC_CNTL_STORE7_REG	General purpose retention register 7	0x3FF480BC	R/W
Internal power management registers			
RTC_CNTL_ANA_CONF_REG	Power-up/down configuration	0x3FF48030	R/W
RTC_CNTL_VREG_REG	Internal power distribution and control	0x3FF4807C	R/W
RTC_CNTL_PWC_REG	RTC domain power management	0x3FF48080	R/W
RTC_CNTL_DIG_PWC_REG	Digital domain power management	0x3FF48084	R/W
RTC_CNTL_DIG_ISO_REG	Digital domain isolation control	0x3FF48088	RO
RTC watchdog configuration and control registers			
RTC_CNTL_WDTCONFIG0_REG	WDT Configuration register 0	0x3FF4808C	R/W
RTC_CNTL_WDTCONFIG1_REG	WDT Configuration register 1	0x3FF48090	R/W
RTC_CNTL_WDTCONFIG2_REG	WDT Configuration register 2	0x3FF48094	R/W
RTC_CNTL_WDTCONFIG3_REG	WDT Configuration register 3	0x3FF48098	R/W
RTC_CNTL_WDTCONFIG4_REG	WDT Configuration register 4	0x3FF4809C	R/W
RTC_CNTL_WDTFEED_REG	Watchdog feed register	0x3FF480A0	WO
RTC_CNTL_WDTWPROTECT_REG	Watchdog write protect register	0x3FF480A4	R/W
Miscellaneous RTC configuration registers			
RTC_CNTL_EXT_XTL_CONF_REG	XTAL control by external pads	0x3FF4805C	R/W

Name	Description	Address	Access
RTC_CNTL_SLP_REJECT_CONF_REG	Reject cause and enable control	0x3FF48064	R/W
RTC_CNTL_CPU_PERIOD_CONF_REG	CPU period select	0x3FF48068	R/W
RTC_CNTL_CLK_CONF_REG	Configuration of RTC clocks	0x3FF48070	R/W
RTC_CNTL_SDIO_CONF_REG	SDIO configuration	0x3FF48074	R/W
RTC_CNTL_SW_CPU_STALL_REG	Stall of CPUs	0x3FF480AC	R/W
RTC_CNTL_HOLD_FORCE_REG	RTC pad hold register	0x3FF480C8	R/W
RTC_CNTL_BROWN_OUT_REG	Brownout management	0x3FF480D4	R/W

30.5 Registers

Register 30.1: RTC_CNTL_OPTIONS0_REG (0x3FF48000)



RTC_CNTL_SW_SYS_RST SW system reset. (WO)

RTC_CNTL_DG_WRAP_FORCE_NORST The digital core forces no reset in deep sleep. (R/W)

RTC_CNTL_DG_WRAP_FORCE_RST The digital core can force a reset in deep sleep. (R/W)

RTC_CNTL_BIAS_CORE_FORCE_PU BIAS_CORE force power up. (R/W)

RTC_CNTL_BIAS_CORE_FORCE_PD BIAS_CORE force power down. (R/W)

RTC_CNTL_BIAS_CORE_FOLW_8M BIAS_CORE follow CK8M. (R/W)

RTC_CNTL_BIAS_I2C_FORCE_PU BIAS_I2C force power up. (R/W)

RTC_CNTL_BIAS_I2C_FORCE_PD BIAS_I2C force power down. (R/W)

RTC_CNTL_BIAS_I2C_FOLW_8M BIAS_I2C follow CK8M. (R/W)

RTC_CNTL_BIAS_FORCE_NOSLEEP BIAS_SLEEP force no sleep. (R/W)

RTC_CNTL_BIAS_FORCE_SLEEP BIAS_SLEEP force sleep. (R/W)

RTC_CNTL_BIAS_SLEEP_FOLW_8M BIAS_SLEEP follow CK8M. (R/W)

RTC_CNTL_XTL_FORCE_PU Crystal force power up. (R/W)

RTC_CNTL_XTL_FORCE_PD Crystal force power down. (R/W)

RTC_CNTL_BBPLL_FORCE_PU BB_PLL force power up. (R/W)

RTC_CNTL_BBPLL_FORCE_PD BB_PLL force power down. (R/W)

RTC_CNTL_BBPLL_I2C_FORCE_PU BB_PLL_I2C force power up. (R/W)

RTC_CNTL_BBPLL_I2C_FORCE_PD BB_PLL _I2C force power down. (R/W)

RTC_CNTL_BB_I2C_FORCE_PU BB_I2C force power up. (R/W)

RTC_CNTL_BB_I2C_FORCE_PD BB_I2C force power down. (R/W)

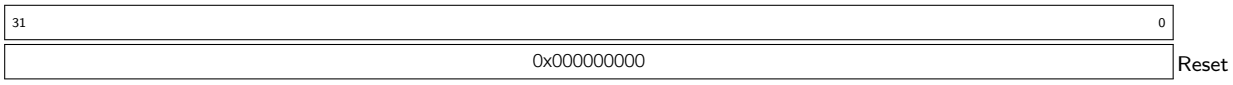
RTC_CNTL_SW_PROCPU_RST PRO_CPU SW reset. (WO)

RTC_CNTL_SW_APPCPU_RST APP_CPU SW reset. (WO)

RTC_CNTL_SW_STALL_PROCPU_C0 described under RTC_CNTL_SW_CPU_STALL_REG. (R/W)

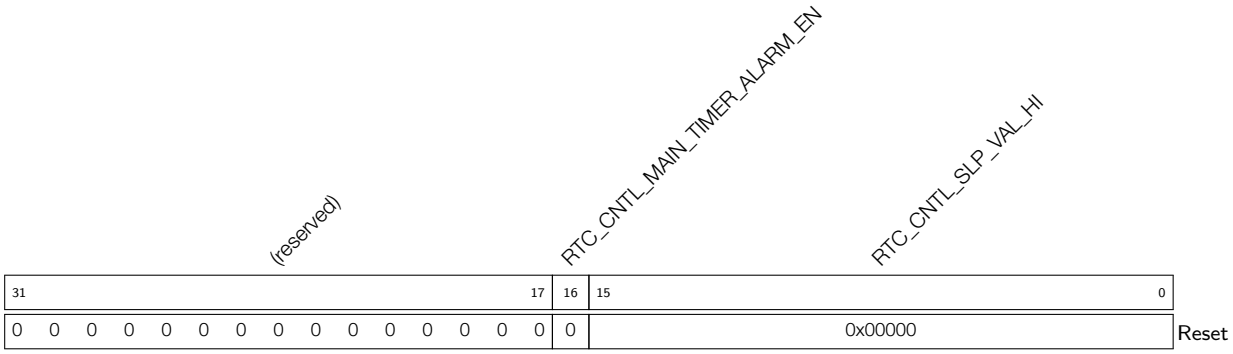
RTC_CNTL_SW_STALL_APPCPU_C0 described under RTC_CNTL_SW_CPU_STALL_REG. (R/W)

Register 30.2: RTC_CNTL_SLP_TIMER0_REG (0x3FF48004)



RTC_CNTL_SLP_TIMER0_REG RTC sleep timer low 32 bits. (R/W)

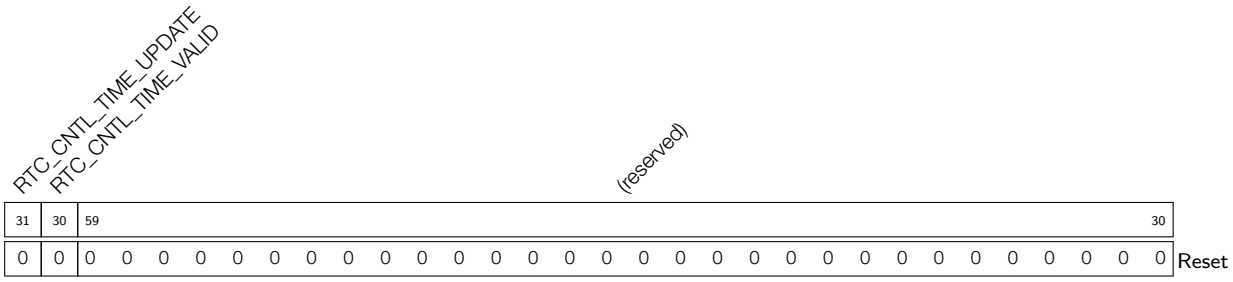
Register 30.3: RTC_CNTL_SLP_TIMER1_REG (0x3FF48008)



RTC_CNTL_MAIN_TIMER_ALARM_EN Timer alarm enable bit. (R/W)

RTC_CNTL_SLP_VAL_HI RTC sleep timer high 16 bits. (R/W)

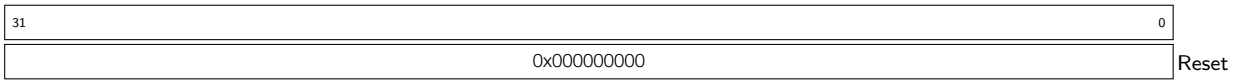
Register 30.4: RTC_CNTL_TIME_UPDATE_REG (0x3FF4800C)



RTC_CNTL_TIME_UPDATE Set 1: to update register with RTC timer. (WO)

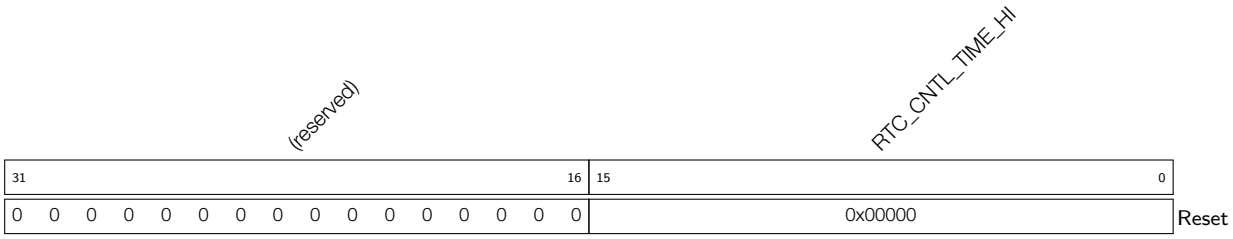
RTC_CNTL_TIME_VALID Indicates that the register is updated. (RO)

Register 30.5: RTC_CNTL_TIME0_REG (0x3FF48010)



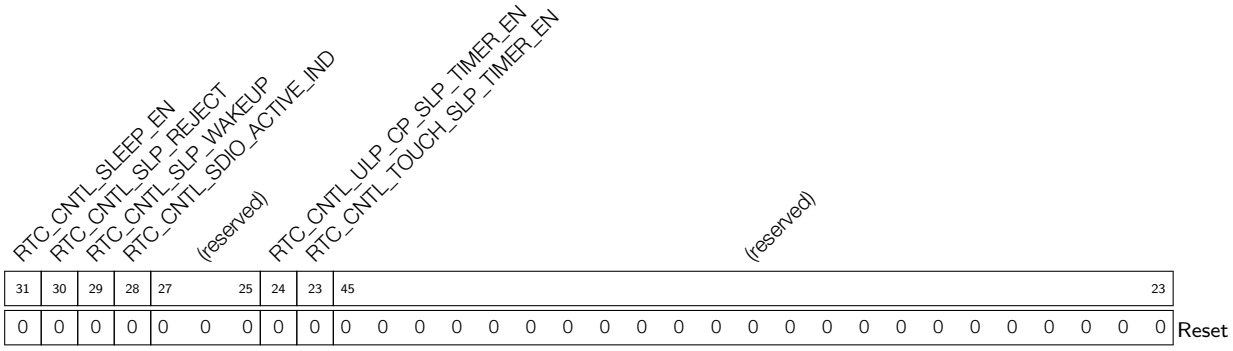
RTC_CNTL_TIME0_REG RTC timer low 32 bits. (RO)

Register 30.6: RTC_CNTL_TIME1_REG (0x3FF48014)



RTC_CNTL_TIME_HI RTC timer high 16 bits. (RO)

Register 30.7: RTC_CNTL_STATE0_REG (0x3FF48018)



RTC_CNTL_SLEEP_EN Sleep enable bit. (R/W)

RTC_CNTL_SLP_REJECT Sleep reject bit. (R/W)

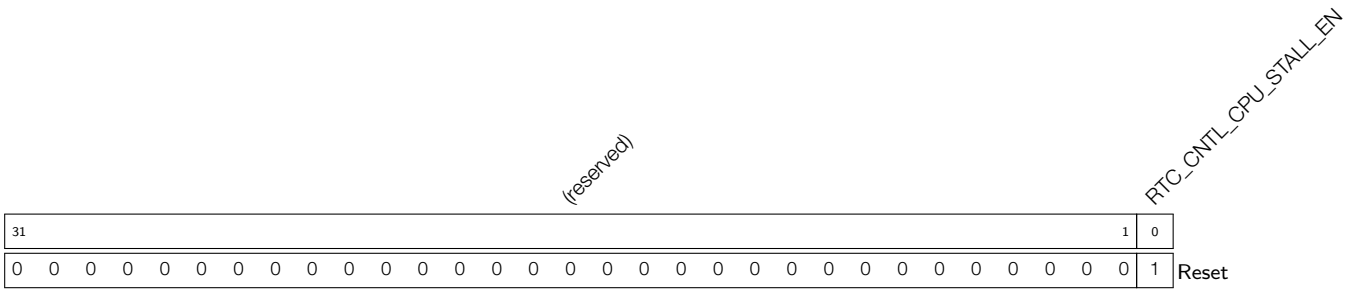
RTC_CNTL_SLP_WAKEUP Sleep wake-up bit. (R/W)

RTC_CNTL_SDIO_ACTIVE_IND SDIO active indication. (RO)

RTC_CNTL_ULP_CP_SLP_TIMER_EN ULP co-processor timer enable bit. (R/W)

RTC_CNTL_TOUCH_SLP_TIMER_EN Touch timer enable bit. (R/W)

Register 30.8: RTC_CNTL_TIMER1_REG (0x3FF4801C)



RTC_CNTL_CPU_STALL_EN CPU stall enable bit. (R/W)

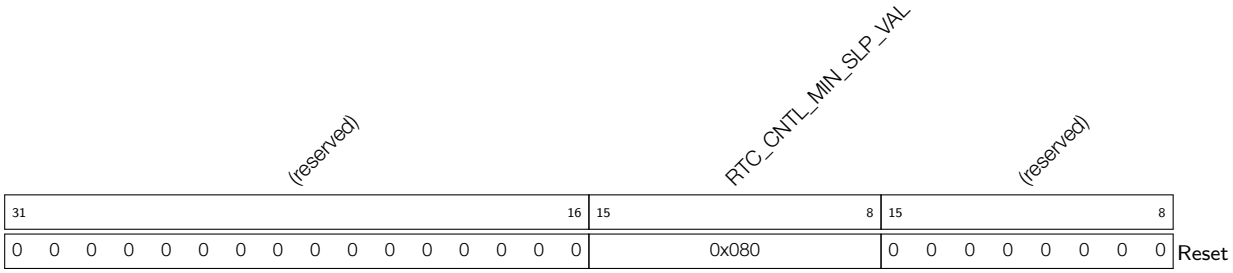
Register 30.9: RTC_CNTL_TIMER2_REG (0x3FF48020)



RTC_CNTL_MIN_TIME_CK8M_OFF Minimal amount of cycles in slow_clk_rtc to power down CK8M. (R/W)

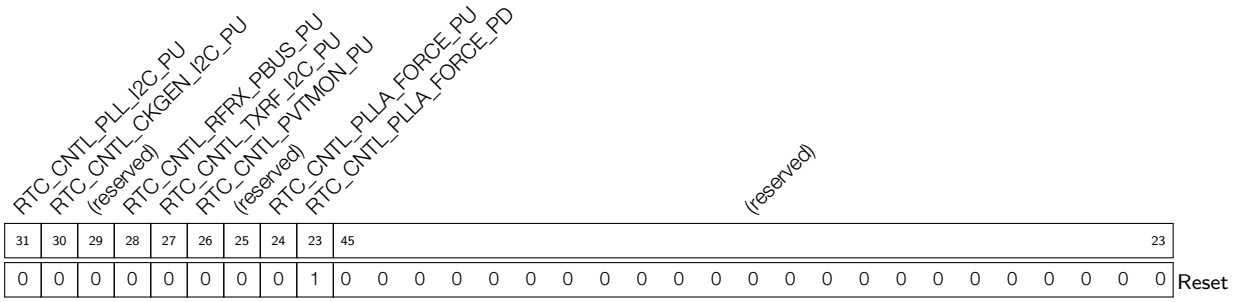
RTC_CNTL_ULPCP_TOUCH_START_WAIT Awaited cycles in slow_clk_rtc before ULP co-processor/touch controller starts working. (R/W)

Register 30.10: RTC_CNTL_TIMER5_REG (0x3FF4802C)



RTC_CNTL_MIN_SLP_VAL Minimal amount of sleep cycles in slow_clk_rtc. (R/W)

Register 30.11: RTC_CNTL_ANA_CONF_REG (0x3FF48030)



RTC_CNTL_PLL_I2C_PU 1: PLL_I2C power up, otherwise power down. (R/W)

RTC_CNTL_CKGEN_I2C_PU 1: CKGEN_I2C power up, otherwise power down. (R/W)

RTC_CNTL_RFRX_PBUS_PU 1: RFRX_PBUS power up, otherwise power down. (R/W)

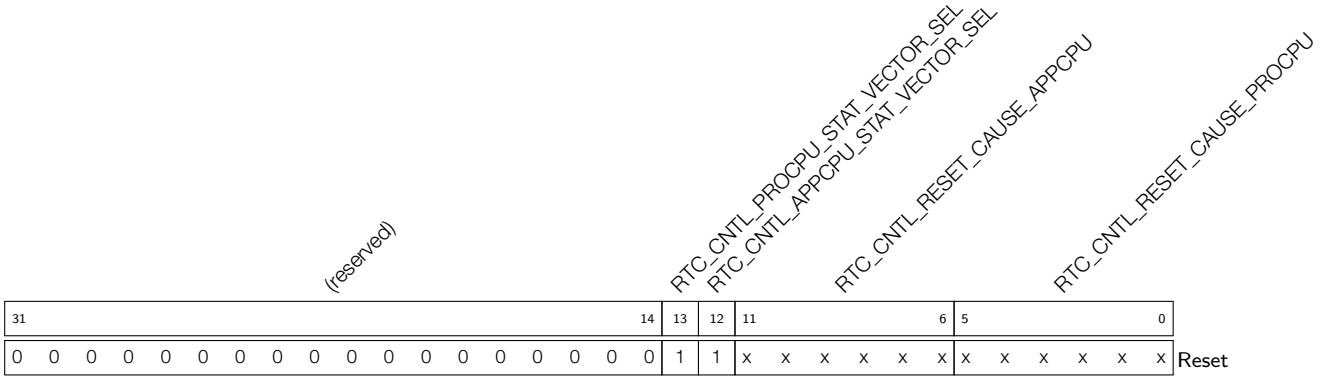
RTC_CNTL_TXRF_I2C_PU 1: TXRF_I2C power up, otherwise power down. (R/W)

RTC_CNTL_PVTMON_PU 1: PVTMON power up, otherwise power down. (R/W)

RTC_CNTL_PLLA_FORCE_PU PLLA force power up. (R/W)

RTC_CNTL_PLLA_FORCE_PD PLLA force power down. (R/W)

Register 30.12: RTC_CNTL_RESET_STATE_REG (0x3FF48034)



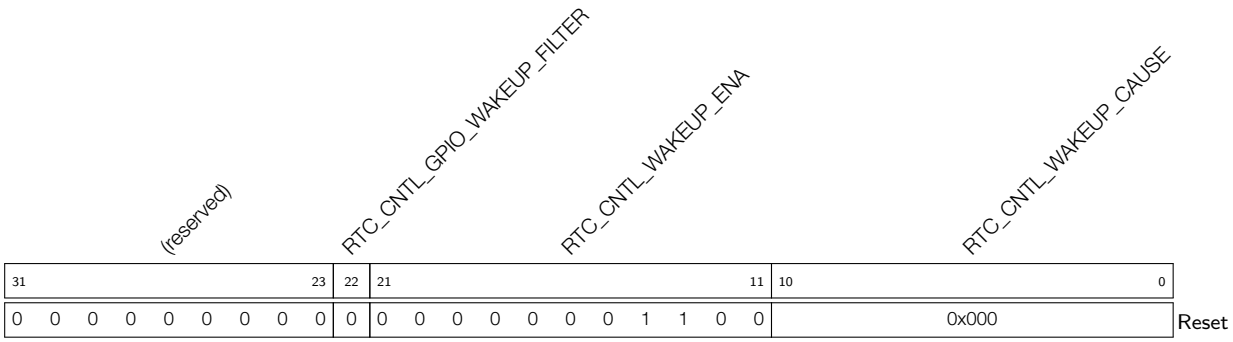
RTC_CNTL_PROCPU_STAT_VECTOR_SEL PRO_CPU state vector selection. (R/W)

RTC_CNTL_APPCPU_STAT_VECTOR_SEL APP_CPU state vector selection. (R/W)

RTC_CNTL_RESET_CAUSE_APPCPU Reset cause for APP_CPU. (RO)

RTC_CNTL_RESET_CAUSE_PROCPU Reset cause for PRO_CPU. (RO)

Register 30.13: RTC_CNTL_WAKEUP_STATE_REG (0x3FF48038)

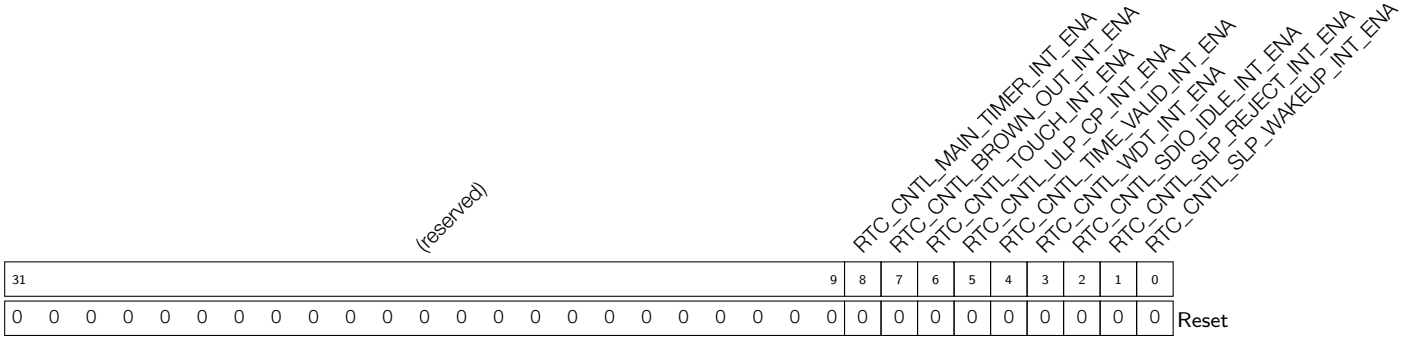


RTC_CNTL_GPIO_WAKEUP_FILTER Enable filter for GPIO wake-up event. (R/W)

RTC_CNTL_WAKEUP_ENA Wake-up enable bitmap. (R/W)

RTC_CNTL_WAKEUP_CAUSE Wake-up cause. (RO)

Register 30.14: RTC_CNTL_INT_ENA_REG (0x3FF4803C)



RTC_CNTL_MAIN_TIMER_INT_ENA The interrupt enable bit for the RTC_CNTL_MAIN_TIMER_INT interrupt. (R/W)

RTC_CNTL_BROWN_OUT_INT_ENA The interrupt enable bit for the RTC_CNTL_BROWN_OUT_INT interrupt. (R/W)

RTC_CNTL_TOUCH_INT_ENA The interrupt enable bit for the RTC_CNTL_TOUCH_INT interrupt. (R/W)

RTC_CNTL_ULP_CP_INT_ENA The interrupt enable bit for the RTC_CNTL_ULP_CP_INT interrupt. (R/W)

RTC_CNTL_TIME_VALID_INT_ENA The interrupt enable bit for the RTC_CNTL_TIME_VALID_INT interrupt. (R/W)

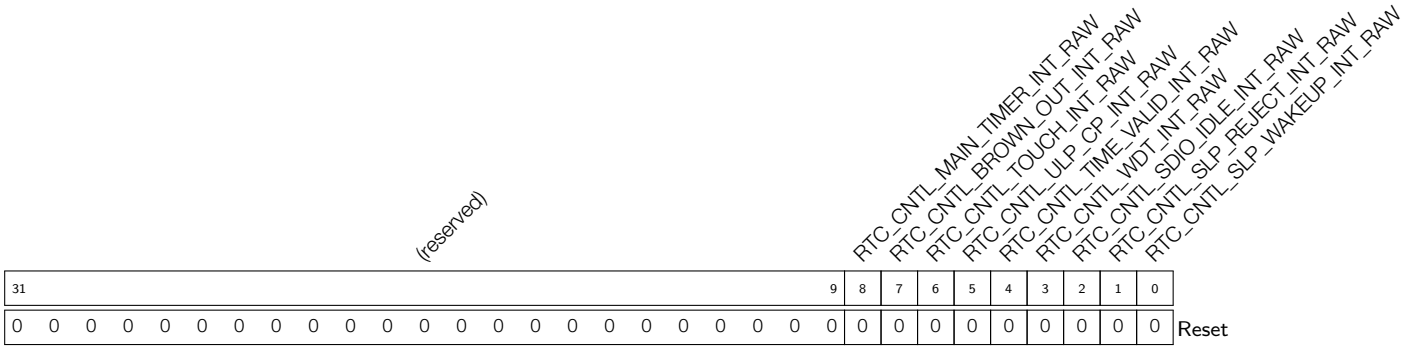
RTC_CNTL_WDT_INT_ENA The interrupt enable bit for the RTC_CNTL_WDT_INT interrupt. (R/W)

RTC_CNTL_SDIO_IDLE_INT_ENA The interrupt enable bit for the RTC_CNTL_SDIO_IDLE_INT interrupt. (R/W)

RTC_CNTL_SLP_REJECT_INT_ENA The interrupt enable bit for the RTC_CNTL_SLP_REJECT_INT interrupt. (R/W)

RTC_CNTL_SLP_WAKEUP_INT_ENA The interrupt enable bit for the RTC_CNTL_SLP_WAKEUP_INT interrupt. (R/W)

Register 30.15: RTC_CNTL_INT_RAW_REG (0x3FF48040)



RTC_CNTL_MAIN_TIMER_INT_RAW The raw interrupt status bit for the RTC_CNTL_MAIN_TIMER_INT interrupt. (RO)

RTC_CNTL_BROWN_OUT_INT_RAW The raw interrupt status bit for the RTC_CNTL_BROWN_OUT_INT interrupt. (RO)

RTC_CNTL_TOUCH_INT_RAW The raw interrupt status bit for the RTC_CNTL_TOUCH_INT interrupt. (RO)

RTC_CNTL_ULP_CP_INT_RAW The raw interrupt status bit for the RTC_CNTL_ULP_CP_INT interrupt. (RO)

RTC_CNTL_TIME_VALID_INT_RAW The raw interrupt status bit for the RTC_CNTL_TIME_VALID_INT interrupt. (RO)

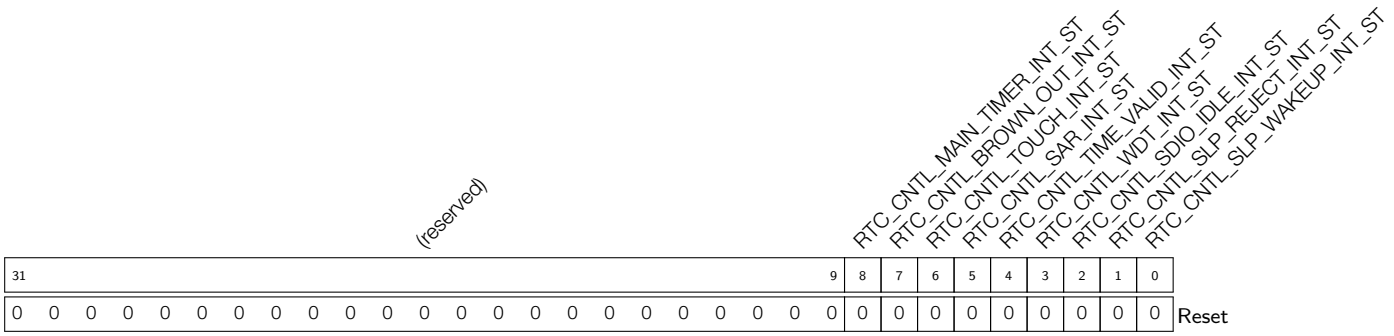
RTC_CNTL_WDT_INT_RAW The raw interrupt status bit for the RTC_CNTL_WDT_INT interrupt. (RO)

RTC_CNTL_SDIO_IDLE_INT_RAW The raw interrupt status bit for the RTC_CNTL_SDIO_IDLE_INT interrupt. (RO)

RTC_CNTL_SLP_REJECT_INT_RAW The raw interrupt status bit for the RTC_CNTL_SLP_REJECT_INT interrupt. (RO)

RTC_CNTL_SLP_WAKEUP_INT_RAW The raw interrupt status bit for the RTC_CNTL_SLP_WAKEUP_INT interrupt. (RO)

Register 30.16: RTC_CNTL_INT_ST_REG (0x3FF48044)



RTC_CNTL_MAIN_TIMER_INT_ST The masked interrupt status bit for the RTC_CNTL_MAIN_TIMER_INT interrupt. (RO)

RTC_CNTL_BROWN_OUT_INT_ST The masked interrupt status bit for the RTC_CNTL_BROWN_OUT_INT interrupt. (RO)

RTC_CNTL_TOUCH_INT_ST The masked interrupt status bit for the RTC_CNTL_TOUCH_INT interrupt. (RO)

RTC_CNTL_SAR_INT_ST The masked interrupt status bit for the RTC_CNTL_SAR_INT interrupt. (RO)

RTC_CNTL_TIME_VALID_INT_ST The masked interrupt status bit for the RTC_CNTL_TIME_VALID_INT interrupt. (RO)

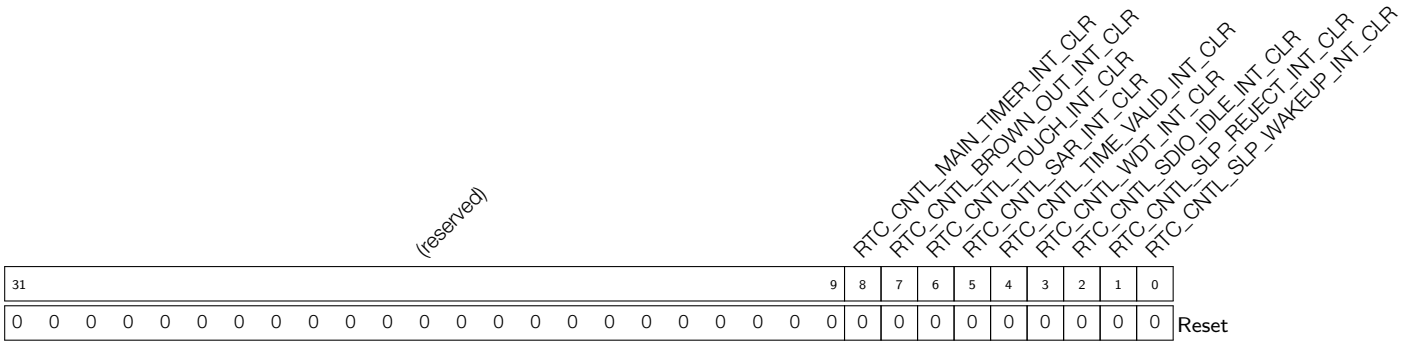
RTC_CNTL_WDT_INT_ST The masked interrupt status bit for the RTC_CNTL_WDT_INT interrupt. (RO)

RTC_CNTL_SDIO_IDLE_INT_ST The masked interrupt status bit for the RTC_CNTL_SDIO_IDLE_INT interrupt. (RO)

RTC_CNTL_SLP_REJECT_INT_ST The masked interrupt status bit for the RTC_CNTL_SLP_REJECT_INT interrupt. (RO)

RTC_CNTL_SLP_WAKEUP_INT_ST The masked interrupt status bit for the RTC_CNTL_SLP_WAKEUP_INT interrupt. (RO)

Register 30.17: RTC_CNTL_INT_CLR_REG (0x3FF48048)



RTC_CNTL_MAIN_TIMER_INT_CLR Set this bit to clear the RTC_CNTL_MAIN_TIMER_INT interrupt. (WO)

RTC_CNTL_BROWN_OUT_INT_CLR Set this bit to clear the RTC_CNTL_BROWN_OUT_INT interrupt. (WO)

RTC_CNTL_TOUCH_INT_CLR Set this bit to clear the RTC_CNTL_TOUCH_INT interrupt. (WO)

RTC_CNTL_SAR_INT_CLR Set this bit to clear the RTC_CNTL_SAR_INT interrupt. (WO)

RTC_CNTL_TIME_VALID_INT_CLR Set this bit to clear the RTC_CNTL_TIME_VALID_INT interrupt. (WO)

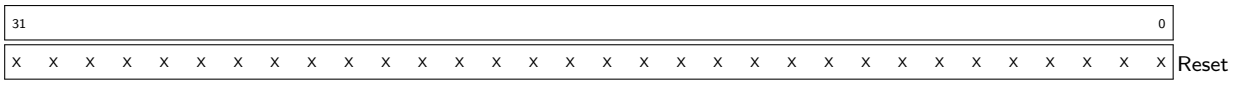
RTC_CNTL_WDT_INT_CLR Set this bit to clear the RTC_CNTL_WDT_INT interrupt. (WO)

RTC_CNTL_SDIO_IDLE_INT_CLR Set this bit to clear the RTC_CNTL_SDIO_IDLE_INT interrupt. (WO)

RTC_CNTL_SLP_REJECT_INT_CLR Set this bit to clear the RTC_CNTL_SLP_REJECT_INT interrupt. (WO)

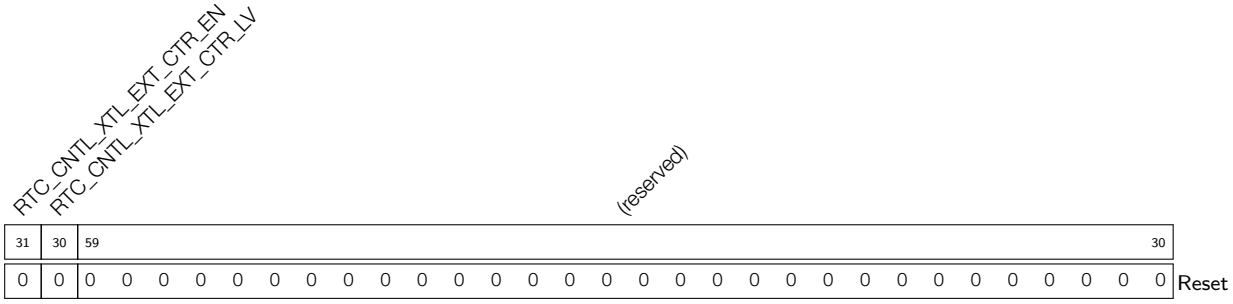
RTC_CNTL_SLP_WAKEUP_INT_CLR Set this bit to clear the RTC_CNTL_SLP_WAKEUP_INT interrupt. (WO)

Register 30.18: RTC_CNTL_STORE*n*_REG (*n*: 0-3) (0x3FF4804C+4n*)**



RTC_CNTL_STORE*n*_REG 32-bit general-purpose retention register. (R/W)

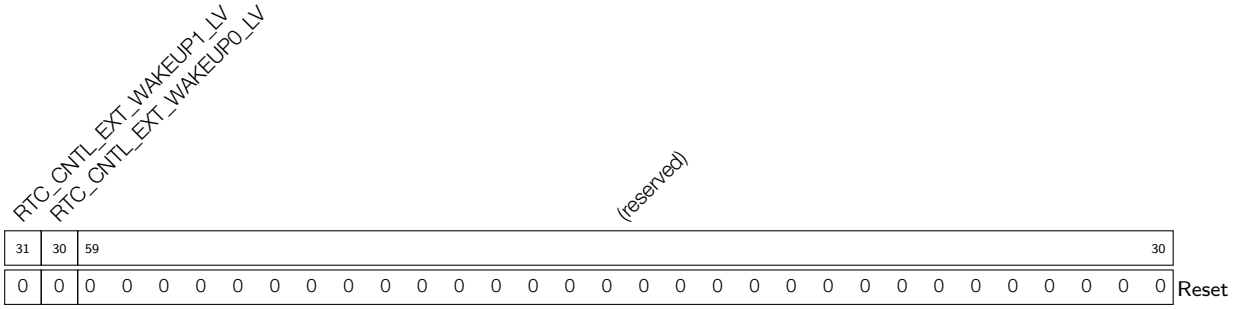
Register 30.19: RTC_CNTL_EXT_XTL_CONF_REG (0x3FF4805C)



RTC_CNTL_XTL_EXT_CTR_EN Enable control XTAL with external pads. (R/W)

RTC_CNTL_XTL_EXT_CTR_LV 0: power down XTAL at high level, 1: power down XTAL at low level. (R/W)

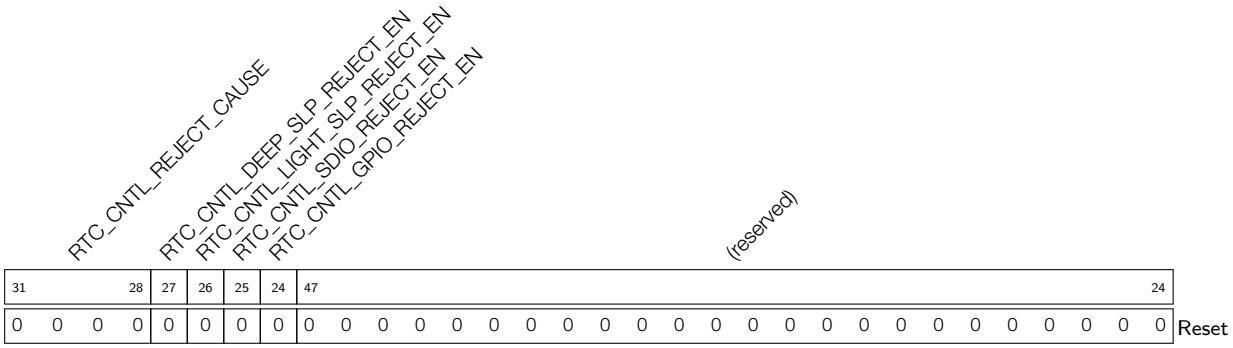
Register 30.20: RTC_CNTL_EXT_WAKEUP_CONF_REG (0x3FF48060)



RTC_CNTL_EXT_WAKEUP1_LV 0: external wake-up at low level, 1: external wake-up at high level. (R/W)

RTC_CNTL_EXT_WAKEUP0_LV 0: external wake-up at low level, 1: external wake-up at high level. (R/W)

Register 30.21: RTC_CNTL_SLP_REJECT_CONF_REG (0x3FF48064)



RTC_CNTL_REJECT_CAUSE Sleep reject cause. (RO)

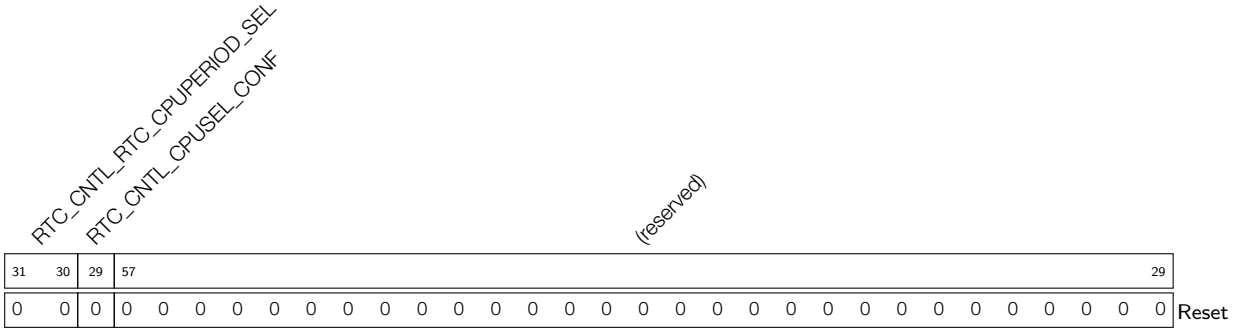
RTC_CNTL_DEEP_SLP_REJECT_EN Enable reject for deep sleep. (R/W)

RTC_CNTL_LIGHT_SLP_REJECT_EN Enable reject for light sleep. (R/W)

RTC_CNTL_SDIO_REJECT_EN Enable SDIO reject. (R/W)

RTC_CNTL_GPIO_REJECT_EN Enable GPIO reject. (R/W)

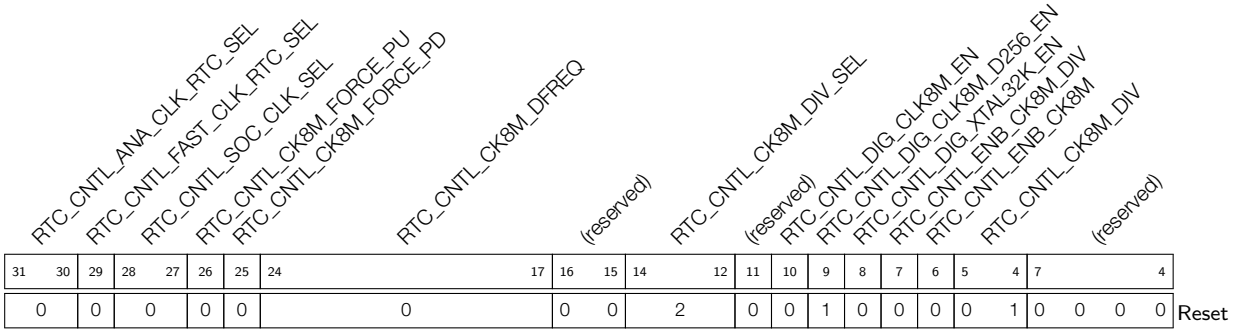
Register 30.22: RTC_CNTL_CPU_PERIOD_CONF_REG (0x3FF48068)



RTC_CNTL_RTC_CPUPERIOD_SEL CPU period selection. (R/W)

RTC_CNTL_CPUSEL_CONF CPU selection option. (R/W)

Register 30.23: RTC_CNTL_CLK_CONF_REG (0x3FF48070)



RTC_CNTL_ANA_CLK_RTC_SEL slow_clk_rtc sel. 0: SLOW_CK, 1: CK_XTAL_32K, 2: CK8M_D256_OUT. (R/W)

RTC_CNTL_FAST_CLK_RTC_SEL fast_clk_rtc sel. 0: XTAL div 4, 1: CK8M. (R/W)

RTC_CNTL_SOC_CLK_SEL SOC clock sel. 0: XTAL, 1: PLL, 2: CK8M, 3: APLL. (R/W)

RTC_CNTL_CK8M_FORCE_PU CK8M force power up. (R/W)

RTC_CNTL_CK8M_FORCE_PD CK8M force power down. (R/W)

RTC_CNTL_CK8M_DFREQ CK8M_DFREQ. (R/W)

RTC_CNTL_CK8M_DIV_SEL Divider = reg_rtc_cntl_ck8m_div_sel + 1. (R/W)

RTC_CNTL_DIG_CLK8M_EN Enable CK8M for digital core (no relation to RTC core). (R/W)

RTC_CNTL_DIG_CLK8M_D256_EN Enable CK8M_D256_OUT for digital core (no relation to RTC core). (R/W)

RTC_CNTL_DIG_XTAL32K_EN Enable CK_XTAL_32K for digital core (no relation to RTC core). (R/W)

RTC_CNTL_ENB_CK8M_DIV 1: CK8M_D256_OUT is actually CK8M, 0: CK8M_D256_OUT is CK8M divided by 256. (R/W)

RTC_CNTL_ENB_CK8M Disable CK8M and CK8M_D256_OUT. (R/W)

RTC_CNTL_CK8M_DIV CK8M_D256_OUT divider. 00: div128, 01: div256, 10: div512, 11: div1024. (R/W)

Register 30.24: RTC_CNTL_SDIO_CONF_REG (0x3FF48074)

RTC_CNTL_XPD_SDIO_VREG	RTC_CNTL_DREFH_SDIO	RTC_CNTL_DREFM_SDIO	RTC_CNTL_DREFL_SDIO	RTC_CNTL_REG1P8_READY	RTC_CNTL_SDIO_TIEH	RTC_CNTL_SDIO_FORCE	RTC_CNTL_SDIO_VREG_PD_EN	(reserved)	
31	30 29	28 27	26 25	24	23	22	21	41 21	
0	0 0	0 0	0 1	0	1	0	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_XPD_SDIO_VREG SW option for XPD_SDIO_VREG; active only when reg_rtc_cntl_sdio_force == 1. (R/W)

RTC_CNTL_DREFH_SDIO SW option for DREFH_SDIO; active only when reg_rtc_cntl_sdio_force == 1. (R/W)

RTC_CNTL_DREFM_SDIO SW option for DREFM_SDIO; active only when reg_rtc_cntl_sdio_force == 1. (R/W)

RTC_CNTL_DREFL_SDIO SW option for DREFL_SDIO; active only when reg_rtc_cntl_sdio_force == 1. (R/W)

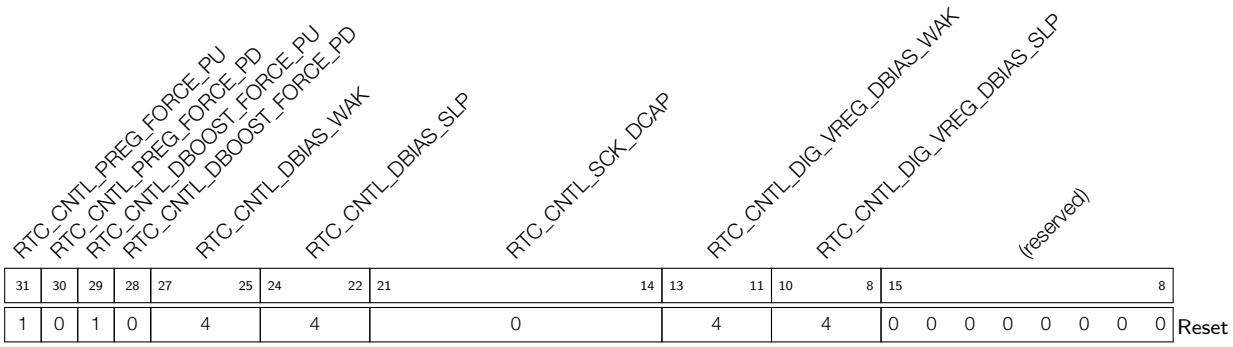
RTC_CNTL_REG1P8_READY Read-only register for REG1P8_READY. (RO)

RTC_CNTL_SDIO_TIEH SW option for SDIO_TIEH; active only when reg_rtc_cntl_sdio_force == 1. (R/W)

RTC_CNTL_SDIO_FORCE 1: use SW option to control SDIO_VREG; 0: use state machine to control SDIO_VREG. (R/W)

RTC_CNTL_SDIO_VREG_PD_EN Power down SDIO_VREG in sleep; active only when reg_rtc_cntl_sdio_force == 0. (R/W)

Register 30.25: RTC_CNTL_VREG_REG (0x3FF4807C)



RTC_CNTL_VREG_FORCE_PU RTC voltage regulator - force power up. (R/W)

RTC_CNTL_VREG_FORCE_PD RTC voltage regulator - force power down (in this case power down means decreasing the voltage to 0.8V or lower). (R/W)

RTC_CNTL_DBOOST_FORCE_PU RTC_DBOOST force power up. (R/W)

RTC_CNTL_DBOOST_FORCE_PD RTC_DBOOST force power down. (R/W)

RTC_CNTL_DBIAS_WAK RTC_DBIAS during wake-up. (R/W)

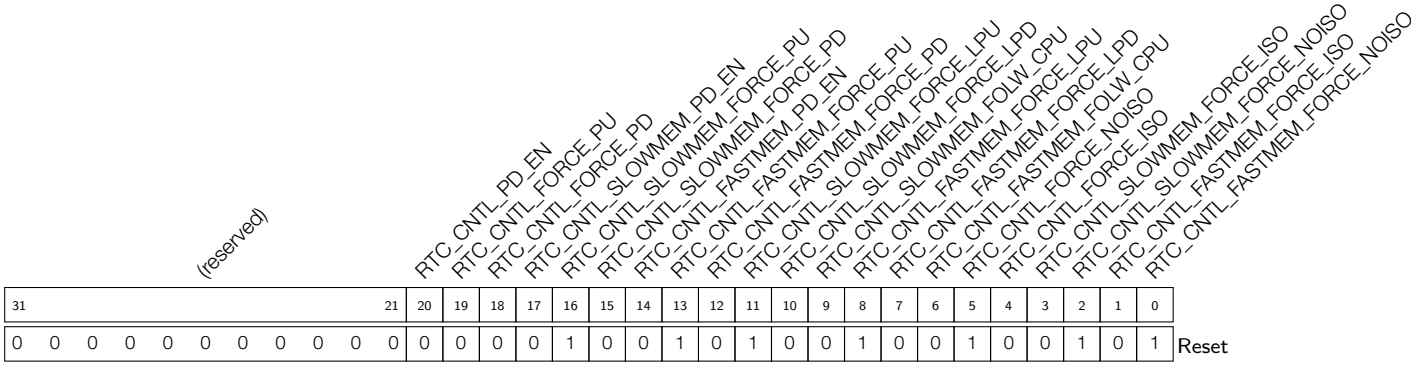
RTC_CNTL_DBIAS_SLP RTC_DBIAS during sleep. (R/W)

RTC_CNTL_SCK_DCAP Used to adjust the frequency of RTC slow clock. (R/W)

RTC_CNTL_DIG_VREG_DBIAS_WAK Digital voltage regulator DBIAS during wake-up. (R/W)

RTC_CNTL_DIG_VREG_DBIAS_SLP Digital voltage regulator DBIAS during sleep. (R/W)

Register 30.26: RTC_CNTL_PWC_REG (0x3FF48080)



RTC_CNTL_PD_EN Enable power down rtc_peri in sleep. (R/W)

RTC_CNTL_FORCE_PU rtc_peri force power up. (R/W)

RTC_CNTL_FORCE_PD rtc_peri force power down. (R/W)

RTC_CNTL_SLOWMEM_PD_EN Enable power down RTC memory in sleep. (R/W)

RTC_CNTL_SLOWMEM_FORCE_PU RTC memory force power up. (R/W)

RTC_CNTL_SLOWMEM_FORCE_PD RTC memory force power down. (R/W)

RTC_CNTL_FASTMEM_PD_EN Enable power down fast RTC memory in sleep. (R/W)

RTC_CNTL_FASTMEM_FORCE_PU Fast RTC memory force power up. (R/W)

RTC_CNTL_FASTMEM_FORCE_PD Fast RTC memory force power down. (R/W)

RTC_CNTL_SLOWMEM_FORCE_LPU RTC memory force power up in low-power mode. (R/W)

RTC_CNTL_SLOWMEM_FORCE_LPD RTC memory force power down in low-power mode. (R/W)

RTC_CNTL_SLOWMEM_FOLW_CPU 1: RTC memory low-power mode PD following CPU; 0: RTC memory low-power mode PD following RTC state machine. (R/W)

RTC_CNTL_FASTMEM_FORCE_LPU Fast RTC memory force power up in low-power mode. (R/W)

RTC_CNTL_FASTMEM_FORCE_LPD Fast RTC memory force power down in low-power mode. (R/W)

RTC_CNTL_FASTMEM_FOLW_CPU 1: Fast RTC memory low-power mode PD following CPU; 0: fast RTC memory low-power mode PD following RTC state machine. (R/W)

RTC_CNTL_FORCE_NOISO rtc_peri force no isolation. (R/W)

RTC_CNTL_FORCE_ISO rtc_peri force isolation. (R/W)

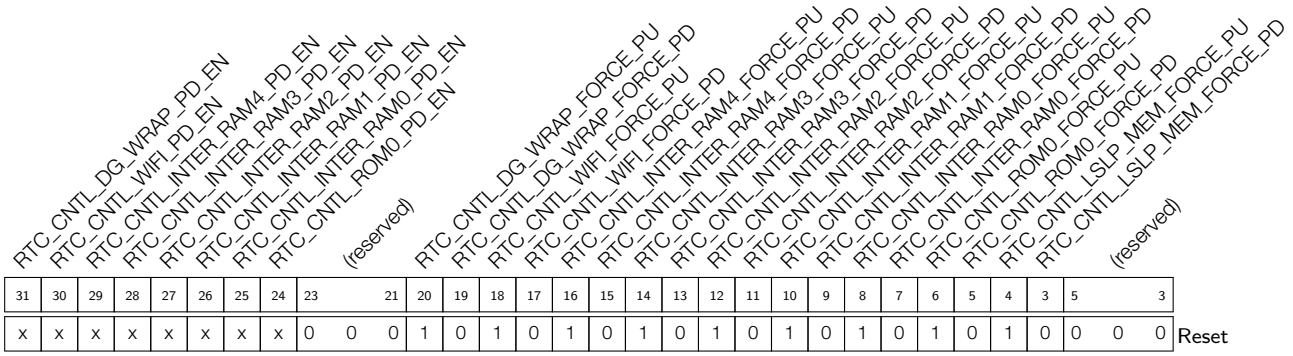
RTC_CNTL_SLOWMEM_FORCE_ISO RTC memory force isolation. (R/W)

RTC_CNTL_SLOWMEM_FORCE_NOISO RTC memory force no isolation. (R/W)

RTC_CNTL_FASTMEM_FORCE_ISO Fast RTC memory force isolation. (R/W)

RTC_CNTL_FASTMEM_FORCE_NOISO Fast RTC memory force no isolation. (R/W)

Register 30.27: RTC_CNTL_DIG_PWC_REG (0x3FF48084)



RTC_CNTL_DG_WRAP_PD_EN Enable power down digital core in sleep mode. (R/W)

RTC_CNTL_WIFI_PD_EN Enable power down Wi-Fi in sleep. (R/W)

RTC_CNTL_INTER_RAM4_PD_EN Enable power down internal SRAM 4 in sleep mode. (R/W)

RTC_CNTL_INTER_RAM3_PD_EN Enable power down internal SRAM 3 in sleep mode. (R/W)

RTC_CNTL_INTER_RAM2_PD_EN Enable power down internal SRAM 2 in sleep mode. (R/W)

RTC_CNTL_INTER_RAM1_PD_EN Enable power down internal SRAM 1 in sleep mode. (R/W)

RTC_CNTL_INTER_RAM0_PD_EN Enable power down internal SRAM 0 in sleep mode. (R/W)

RTC_CNTL_ROM0_PD_EN Enable power down ROM in sleep mode. (R/W)

RTC_CNTL_DG_WRAP_FORCE_PU Digital core force power up. (R/W)

RTC_CNTL_DG_WRAP_FORCE_PD Digital core force power down. (R/W)

RTC_CNTL_WIFI_FORCE_PU Wi-Fi force power up. (R/W)

RTC_CNTL_WIFI_FORCE_PD Wi-Fi force power down. (R/W)

RTC_CNTL_INTER_RAM4_FORCE_PU Internal SRAM 4 force power up. (R/W)

RTC_CNTL_INTER_RAM4_FORCE_PD Internal SRAM 4 force power down. (R/W)

RTC_CNTL_INTER_RAM3_FORCE_PU Internal SRAM 3 force power up. (R/W)

RTC_CNTL_INTER_RAM3_FORCE_PD Internal SRAM 3 force power down. (R/W)

RTC_CNTL_INTER_RAM2_FORCE_PU Internal SRAM 2 force power up. (R/W)

RTC_CNTL_INTER_RAM2_FORCE_PD Internal SRAM 2 force power down. (R/W)

RTC_CNTL_INTER_RAM1_FORCE_PU Internal SRAM 1 force power up. (R/W)

RTC_CNTL_INTER_RAM1_FORCE_PD Internal SRAM 1 force power down. (R/W)

RTC_CNTL_INTER_RAM0_FORCE_PU Internal SRAM 0 force power up. (R/W)

RTC_CNTL_INTER_RAM0_FORCE_PD Internal SRAM 0 force power down. (R/W)

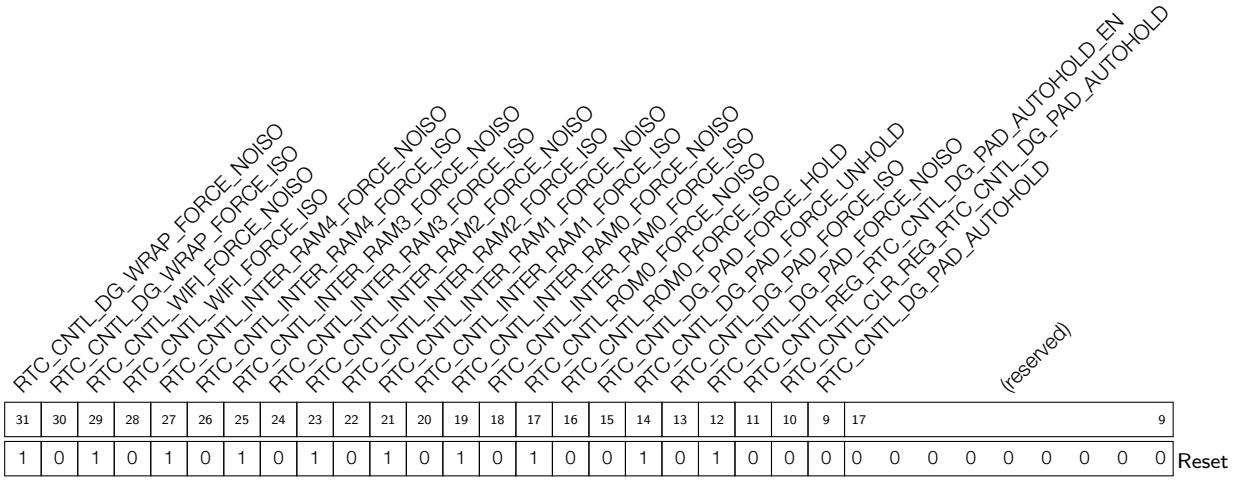
RTC_CNTL_ROM0_FORCE_PU ROM force power up. (R/W)

RTC_CNTL_ROM0_FORCE_PD ROM force power down. (R/W)

RTC_CNTL_LSLP_MEM_FORCE_PU Memories in digital core force power up in sleep mode. (R/W)

RTC_CNTL_LSLP_MEM_FORCE_PD Memories in digital core force power down in sleep mode. (R/W)

Register 30.28: RTC_CNTL_DIG_ISO_REG (0x3FF48088)



RTC_CNTL_DG_WRAP_FORCE_NOISO Digital core force no isolation. (R/W)

RTC_CNTL_DG_WRAP_FORCE_ISO Digital core force isolation. (R/W)

RTC_CNTL_WIFI_FORCE_NOISO Wi-Fi force no isolation. (R/W)

RTC_CNTL_WIFI_FORCE_ISO Wi-Fi force isolation. (R/W)

RTC_CNTL_INTER_RAM4_FORCE_NOISO Internal SRAM 4 force no isolation. (R/W)

RTC_CNTL_INTER_RAM4_FORCE_ISO Internal SRAM 4 force isolation. (R/W)

RTC_CNTL_INTER_RAM3_FORCE_NOISO Internal SRAM 3 force no isolation. (R/W)

RTC_CNTL_INTER_RAM3_FORCE_ISO Internal SRAM 3 force isolation. (R/W)

RTC_CNTL_INTER_RAM2_FORCE_NOISO Internal SRAM 2 force no isolation. (R/W)

RTC_CNTL_INTER_RAM2_FORCE_ISO Internal SRAM 2 force isolation. (R/W)

RTC_CNTL_INTER_RAM1_FORCE_NOISO Internal SRAM 1 force no isolation. (R/W)

RTC_CNTL_INTER_RAM1_FORCE_ISO Internal SRAM 1 force isolation. (R/W)

RTC_CNTL_INTER_RAM0_FORCE_NOISO Internal SRAM 0 force no isolation. (R/W)

RTC_CNTL_INTER_RAM0_FORCE_ISO Internal SRAM 0 force isolation. (R/W)

RTC_CNTL_ROM0_FORCE_NOISO ROM force no isolation. (R/W)

RTC_CNTL_ROM0_FORCE_ISO ROM force isolation. (R/W)

RTC_CNTL_DG_PAD_FORCE_HOLD Digital pad force hold. (R/W)

RTC_CNTL_DG_PAD_FORCE_UNHOLD Digital pad force un-hold. (R/W)

RTC_CNTL_DG_PAD_FORCE_ISO Digital pad force isolation. (R/W)

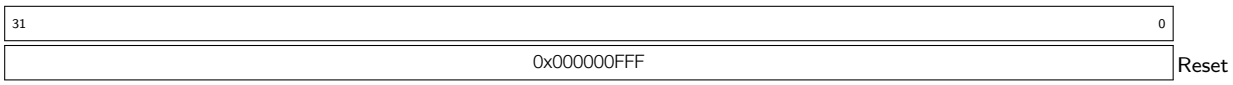
RTC_CNTL_DG_PAD_FORCE_NOISO Digital pad force no isolation. (R/W)

RTC_CNTL_REG_RTC_CNTL_DG_PAD_AUTOHOLD_EN Digital pad enable auto-hold. (R/W)

RTC_CNTL_CLR_REG_RTC_CNTL_DG_PAD_AUTOHOLD Write-only register clears digital pad auto-hold. (WO)

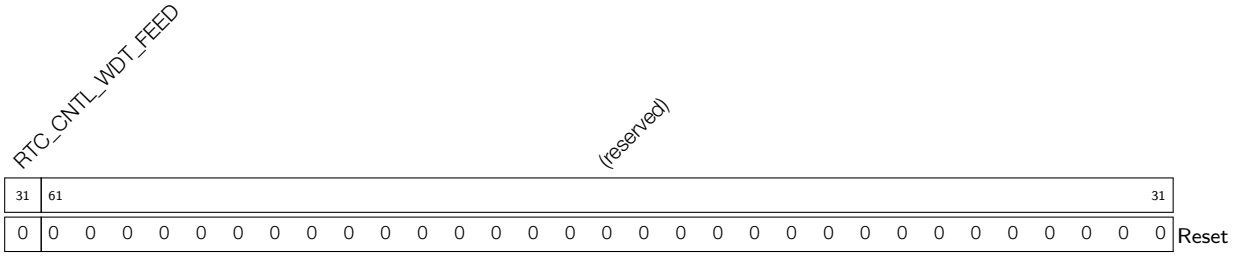
RTC_CNTL_DG_PAD_AUTOHOLD Read-only register indicates digital pad auto-hold status. (RO)

Register 30.29: RTC_CNTL_WDTCONFIG*n*_REG (*n*: 0-4) (0x3FF4808C+4n*)**



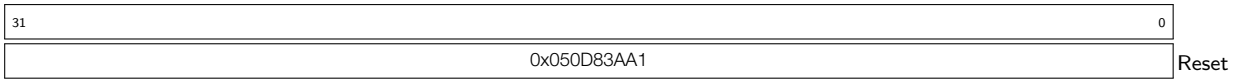
RTC_CNTL_WDTCONFIG*n*_REG Hold cycles for WDT stage*N* (*N* = *n*+1). (R/W)

Register 30.30: RTC_CNTL_WDTFEED_REG (0x3FF480A0)



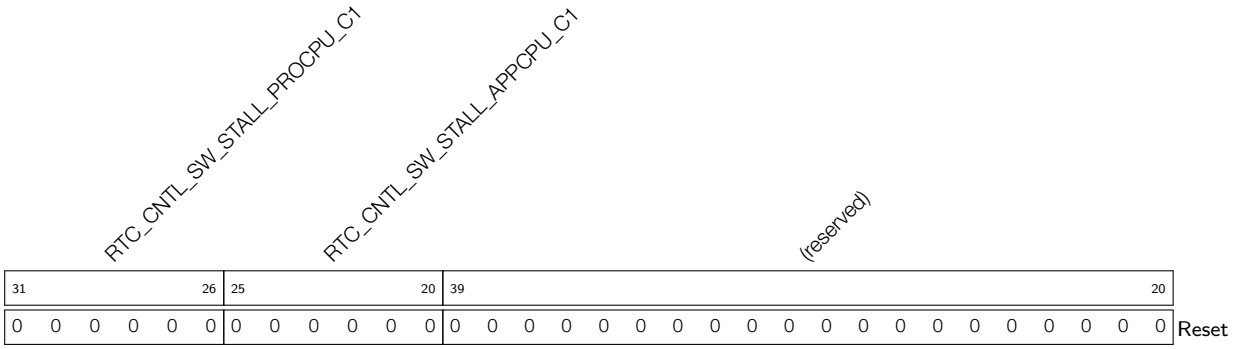
RTC_CNTL_WDT_FEED SW feeds WDT. (WO)

Register 30.31: RTC_CNTL_WDTWPROTECT_REG (0x3FF480A4)



RTC_CNTL_WDTWPROTECT_REG If RTC_CNTL_WDTWPROTECT is other than 0x50d83aa1, then the RTC watchdog will be in a write-protected mode and RTC_CNTL_WDTCONFIGn_REG will be locked for modifications. (R/W)

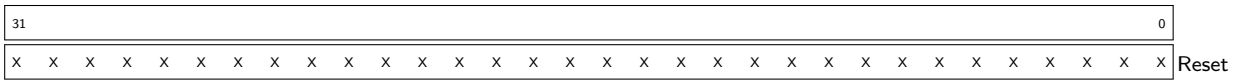
Register 30.32: RTC_CNTL_SW_CPU_STALL_REG (0x3FF480AC)



RTC_CNTL_SW_STALL_PROCPU_C1 reg_rtc_cntl_sw_stall_procpu_c1[5:0], reg_rtc_cntl_sw_stall_procpu_c0[1:0] == 0x86 (100001 10) will stall PRO_CPU, see also RTC_CNTL_OPTIONS0_REG. (R/W)

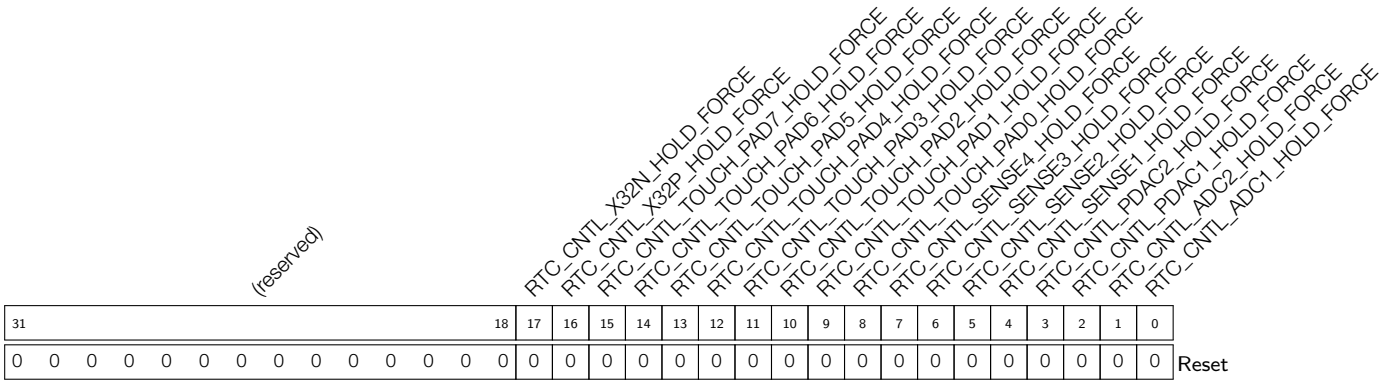
RTC_CNTL_SW_STALL_APPCPU_C1 reg_rtc_cntl_sw_stall_appcpu_c1[5:0], reg_rtc_cntl_sw_stall_appcpu_c0[1:0] == 0x86 (100001 10) will stall APP_CPU, see also RTC_CNTL_OPTIONS0_REG. (R/W)

Register 30.33: RTC_CNTL_STORE*n*_REG (*n*: 4-7) (0x3FF480B0+4n*)**



RTC_CNTL_STORE*n*_REG 32-bit general-purpose retention register. (R/W)

Register 30.34: RTC_CNTL_HOLD_FORCE_REG (0x3FF480C8)



RTC_CNTL_X32N_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_X32P_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_TOUCH_PAD7_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_TOUCH_PAD6_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_TOUCH_PAD5_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_TOUCH_PAD4_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_TOUCH_PAD3_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_TOUCH_PAD2_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_TOUCH_PAD1_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_TOUCH_PAD0_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_SENSE4_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_SENSE3_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_SENSE2_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_SENSE1_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

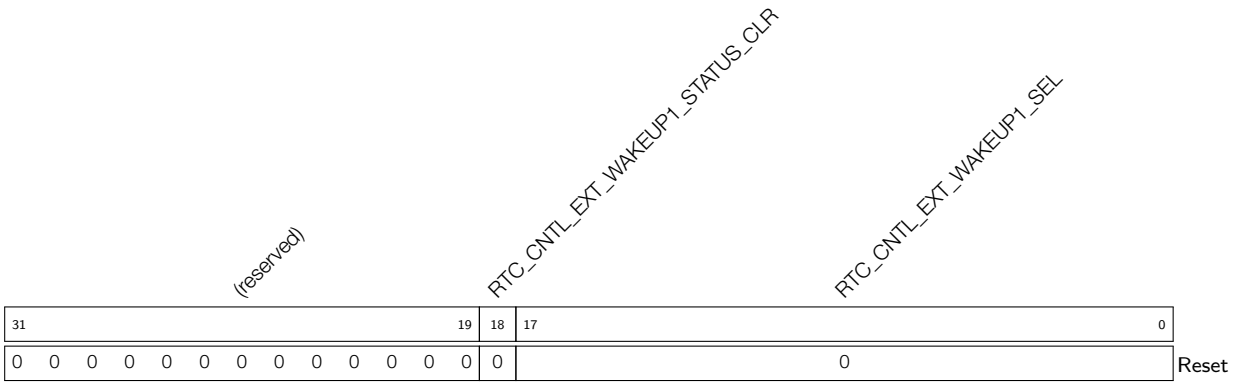
RTC_CNTL_PDAC2_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_PDAC1_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_ADC2_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

RTC_CNTL_ADC1_HOLD_FORCE Set to preserve pad's state during hibernation. (R/W)

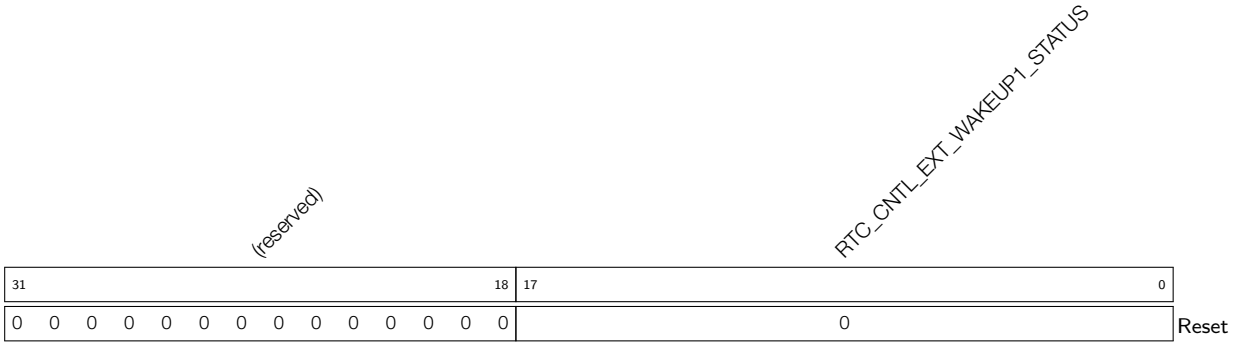
Register 30.35: RTC_CNTL_EXT_WAKEUP1_REG (0x3FF480CC)



RTC_CNTL_EXT_WAKEUP1_STATUS_CLR Clear external wakeup1 status. (WO)

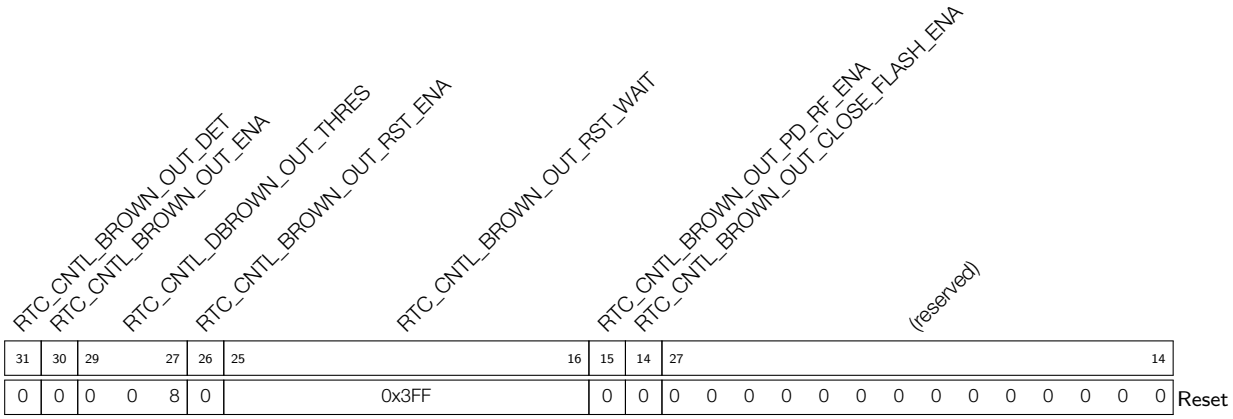
RTC_CNTL_EXT_WAKEUP1_SEL Bitmap to select RTC pads for external wakeup1. (R/W)

Register 30.36: RTC_CNTL_EXT_WAKEUP1_STATUS_REG (0x3FF480D0)



RTC_CNTL_EXT_WAKEUP1_STATUS External wakeup1 status. (RO)

Register 30.37: RTC_CNTL_BROWN_OUT_REG (0x3FF480D4)



RTC_CNTL_BROWN_OUT_DET Brownout detect. (RO)

RTC_CNTL_BROWN_OUT_ENA Enable brownout. (R/W)

RTC_CNTL_DBROWN_OUT_THRES Brownout threshold. (R/W)

RTC_CNTL_BROWN_OUT_RST_ENA Enable brownout reset. (R/W)

RTC_CNTL_BROWN_OUT_RST_WAIT Brownout reset wait cycles. (R/W)

RTC_CNTL_BROWN_OUT_PD_RF_ENA Enable power down RF when brownout happens. (R/W)

RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA Enable close flash when brownout happens. (R/W)

Revision History

Date	Version	Release notes
2018.07	V3.5	Changes to chapter 15 RMT: • Updated RAM start address in section 15.2.2: RMT RAM; • Corrected several wrong addresses of RMT registers; • Updated the description of register RMT_APB_CONF_REG. Updated the description of registers UART_RX_TOUT_THRHD, UART_RXFIFO_FULL_INT_CLR, UART_RXFIFO_FULL_INT_CLR.
2018.06	V3.4	Updated the images in Section 4.8: ESP32 I/O Pad Power Supplies; Updated Section 11.3.3: I^2C Bus Timing; Added notes to Section 14.2.3: LED_PWM Channels; Updated the "Maximum count value" in Section 17.2.3: Pulse Counter Watchpoints; Removed the description of the temperature sensor and LNA.
2018.05	V3.3	Updated the addresses of registers in the Register Summary Section and the Registers Section of Chapter Low-Power Management.
2018.04	V3.2	Updated Figure 23 CMD53 Content; Added six registers in Chapter Ethernet MAC: • DMAOPERATION_MODE_REG; • DMAIN_EN_REG, • DMAMISSEDFR_REG, • PMT_RWUFFR_REG, • PMT_CSR_REG, • EMACLPI_CSR_REG, and • EMACLPITIMERSCONTROL_REG.
2018.04	V3.1	Updated Figure 89 RMT Architecture; Added a note to Section 4.7; Added the function description for the bits of the register in Section 4.46.
2018.03	V3.0	Updated the instruction layout diagram of ST in Section 29.4.2; Added description of registers EMACADDR2HIGH_REG to EMACADDR7LOW_REG in Section 10.9 and Section 10.10.
2018.02	V2.9	Updated Sections 4.2.2, 4.2.3, 4.3.2; Added registers I2S_FIFO_WR_REG and I2S_FIFO_RD_REG in Section I2S Registers.
2018.01	V2.8	Added Chapter Ethernet MAC. Added the description of system parameter BLK3_part_reserve in Chapter eFuse Controller.
2017.12	V2.7	Added Subsection Cache in Section System and Memory; Updated Section Timers and the naming of several registers in LED_PWM; Updated the description of console_debug_disable in Chapter eFuse Controller.
2017.11	V2.6	Updated Chapter Remote Controller Peripheral:

Date	Version	Release notes
		• Updated Figure 89 RMT Architecture; • Updated section RMT RAM; • Updated section Transmitter; • Updated the description of RMT_CH*n*_TX_THR_EVENT_INT. Added notes in Section UART RAM and Register UART_CONF0_REG.
2017.11	V2.5	Updated the addresses for register SPI_CTRL_REG in Section SPI Register Summary; Added Section Clock Phase Selection in Chapter SD/MMC Host Controller, and a description of register CLK_EDGE_SEL; Major revision on Chapter I2C Controller.
2017.09	V2.4	Added the description of register SLC0HOST_TOKEN_RDATA in Chapter SDIO Slave; Added notes in Section The Clock of I2S Module; Added a note in Section GP-SPI Master Mode; Added Chapter DPort Register; Added Chapter DMA Controller.
2017.08	V2.3	Added Chapter Flash Encryption/Decryption.
2017.07	V2.2	Added Chapter Low-Power Management.
2017.07	V2.1	Updated the addresses of the GPIO configuration/data registers and the GPIO RTC function configuration registers in Chapter IO_MUX and GPIO Matrix; Added Chapter PID Controller.
2017.07	V2.0	Added Chapter SDIO Slave.
2017.06	V1.9	Updated Chapter IO_MUX and GPIO Matrix; Added Chapter MCPWM.
2017.06	V1.8	Added register I2S_STATE_REG in Chapter I2S; Updated Chapter IO_MUX and GPIO Matrix; Added Chapter ULP Co-processor.
2017.05	V1.7	Added Chapter On-Chip Sensors and Analog Signal Processing; Added Section Audio PLL; Updated Section eFuse Controller Register Summary; Updated Sections I2S PDM and LCD MODE; Updated Section Communication Format Supported by GP-SPI Slave.
2017.03	V1.6	Added Chapter SD/MMC Host Controller; Added register IO_MUX_PIN_CTRL in Chapter IO_MUX and GPIO Matrix.
2017.03	V1.5	Added Chapter I2S.
2017.01	V1.4	Added Chapter SPI; Added Chapter UART Controllers.
2016.12	V1.3	Added Chapter eFuse Controller; Added Chapter RSA Accelerator; Added Chapter Random Number Generator; Updated Section I2C Controller Interrupt and Section I2C Controller Registers.
2016.11	V1.2	Added Chapter PID/MPU/MMU; Updated Section IO_MUX and GPIO Matrix Register Summary; Updated Section LED_PWM Register Summary.
2016.09	V1.1	Added Chapter I2C Controller.

Date	Version	Release notes
2016.08	V1.0	Initial release.