CBU5000 UWB SoC Family

V210 User Manual V1.0

ShenZhen ChipsBank Technology Co., Ltd.

http://www.chipsbank.com/

7th Floor, Building 12, Phase II, Shenzhen Software Park,

Kejizhong 2nd Road, Nanshan District

Table of Content

List of Figures

List of Tables

1 Overview

This chapter gives an overview of the UWB SoC architecture and main features.

1.1 General description

The UWB Soc is a fully integrated low-power, single chip CMOS RF 3GHz-10GHz UWB SoC compliant with the IEEE 802.15.4a (HRP UWB PHY) and IEEE 802.15.4z (BPRF and HPRF mode) standards. It is suitable for real-time location system (RTLS) with high accuracy of 10 cm.

The chip integrates a 32-bit ARM Cortex-M33 processor with TrustZone security feature, a UWB transceiver for position system, and various peripherals such as UART, SPI, Ethernet PHY.

1.2 Key Features

1.2.1 General

- Compliant with IEEE 802.15.4z, IEEE 802.15.4-2015, and FiRa standards
- Support band group two from 6 to 9 GHz, i.e., channel 5, 6, 8, 9, 10
- Support packet length up to 4095 bytes
- Support various data rates from 850Kbps, 6.8Mbps, 7.8Mbps, 27.2Mbps, and 31.2Mbps
- Support STS packet configuration 0, 1 & 3, with integrated AES-128 for secure ranging.
- Support both SoC mode and transceiver mode
- Support two-way ranging (TWR), and time difference of arrival (TDOA)
- Support 3D angle of arrivals (AOA) with 1 TX and 3 RX
- Hardware based security architecture
- SPI master/slave interface with clock up to 32 MHz
- QSPI controller to support external flash up to 1Mbytes with speed of 64 MHz
- Low BOM cost due to the high integration of SoC
- I2C master interface
- UART interface

1.2.2 Special

- Fully integrated single-chip dual-transceivers SoC, i.e., IR-UWB and BLE
    - Integrated FiRa Framework API with both UWBS and OOB
- 32-Bit MCU at various rate: 64 and 128 MHz
    - Enable optimization between high performance & low power consumption
- Accurate estimation of crystal frequency offset
    - Enable accurate single-side two-way ranging (SS-TWR) to save time & power
- Configurable TX Pulse Shaping Module
    - Enable device to transmit at max regulatory power (-41dBm/MHz) for Channel 10 in China.

- RADAR mode to cater for the rising demand on various applications
    - Presence detection, fall detection, health monitoring
- Long range at max transmit power
    - The coverage for communication and ranging is able to reach 360 meters
- Short-frame Ranging Structure
    - 16-symbol preamble length to increase the ranging rate or reduce the power consumption

1.3 Applications

1.3.1 Target Device

- Smartphone
- Anchor/Tag
- RADAR Device

1.3.2 Use cases

- Smart city & mobilities
    - Indoor navigation
    - Ticket validation
    - Parking Garage Access Control
    - Vehicle digital key
- Smart retail
    - Targeted Marketing
    - Tap-free mobile payment
    - Foot traffic and shopping behaviour analysis
- Smart home
    - Gesture-based control
    - Residential access control
    - Presence based device activation
- Smart Industrial
    - Social distancing
    - Patient tracking
    - Geo-fencing

1.4 High Level Diagram

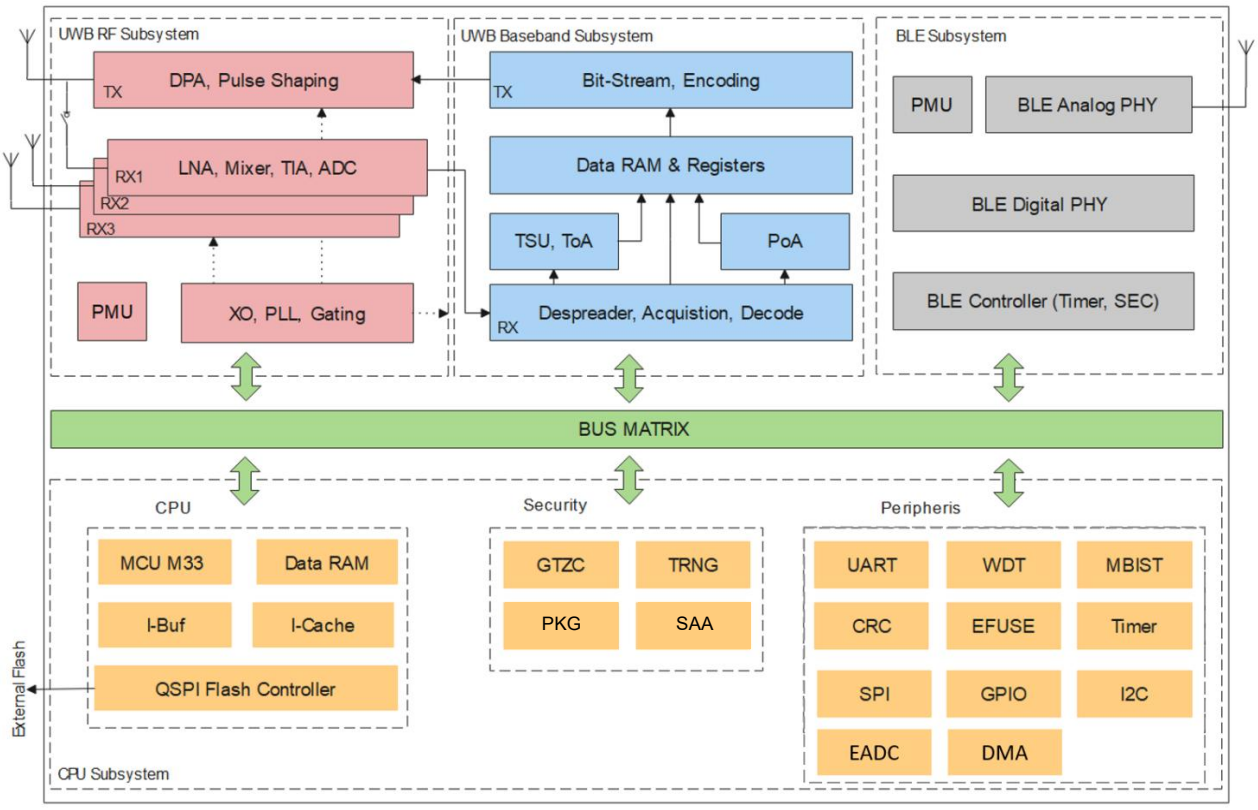


Figure 1-1: High Level SoC diagram

2 Block diagram

The chip integrates a 32-bit ARM Cortex-M33 processor with TrustZone security feature, a UWB transceiver for position system, BLE for device connection and various peripherals such as UART, SPI, I2C.

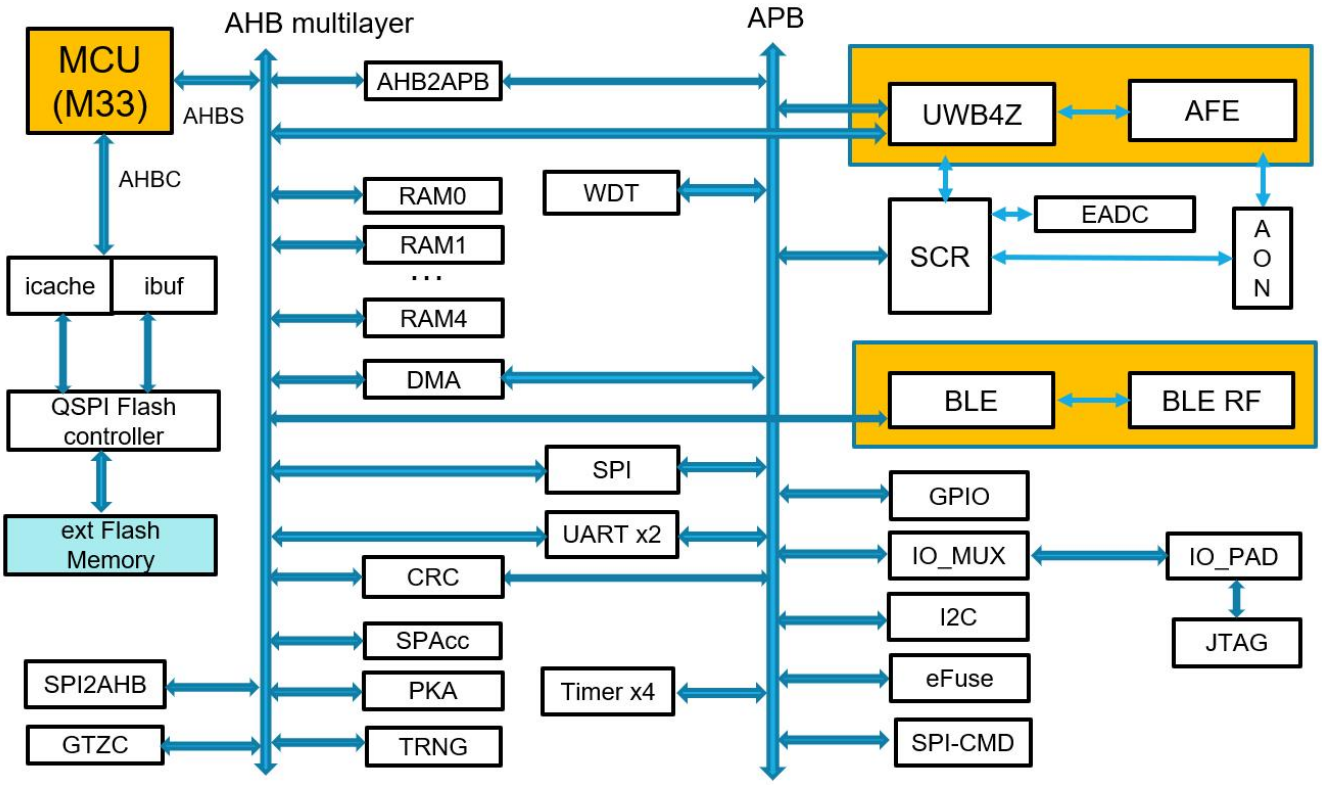


Figure 2-1: Block Diagram of CBU5000

The blocks integrated:

- 32-bit Microprocessor
  - 3-stages pipeline structure
  - Floating point unit
  - Hardware security platform
  - Little-Endian
  - Configurable operating frequency: 64, 128 MHz
- QSPI Flash controller
  - Support up to 1Mbytes flash memory
- Work RAM
  - 4 16-KB and 1 32-KB SRAM
- UART
  - Universal asynchronous receiver/transmitter
  - Communication baud rate can be set by software
  - 2 UART
- SPI
  - Can work as SPI master or slave
  - Configurable clock frequency with max frequency is 64 MHz
- I2C
  - Standard I2C Master
- SAA
  - Security algorithm accelerator:
  - Programmable AES 128/192/256 cryptographic function
- PKG

    - Public key generator
- TRNG
    - True Random number generator
- CRC
    - 8, 16, 32-bit CRC with configurable polynomial
    - A master bus interface is integrated to automatically read data from data memory
- EADC
    - Temperature sensor
    - External analog signal
- WDT
    - Watch-dog timer
- TIMER
    - 4 general purpose timers
    - Able to output PWM signals
- SCR
    - System control register
- ICACHE
    - 16-KB Instruction cache
- IBUF
    - 256-KB Instruction buffer
- GPIO
    - 12 General purpose inputs/outputs
    - Shared with ports of SPI, I2C, UART and internal events.
- BLE
    - BLE LL
    - BLE PHY 5.1
- UWB
    - UWB4Z

3 CPU

The Cortex-M33 processor is a low gate count, highly energy efficient processor that is intended for microcontroller and deeply embedded applications. The processor is based on the Armv8-M architecture and is primarily for use in environments where security is an important consideration.

The processor includes the following features:

• The Security Extension.

• The Floating-point Extension.

• The Digital Signal Processing (DSP) Extension.

• The Debug Extension.

• An in-order issue pipeline.

• Thumb-2 technology. See the *Arm®v8-M Architecture Reference Manual*.

• Data accesses performed as either big or little endian.

• A *Nested Vectored Interrupt Controller* (NVIC) closely integrated with the processor with up to 480 interrupts.

• An optional Floating Point Unit (FPU) supporting single-precision arithmetic.

• Support for exception-continuable instructions, such as LDM, LDMDB, STM, STMDB, PUSH, and POP. If the processor supports FPU, the VLDM, VSTM, VPUSH, VPOP exception-continuable instructions are also included.

• A low-cost debug solution with the optional ability to:

— Implement breakpoints.

— Implement watchpoints, tracing, and system profiling.

— Support printf() style debugging through an *Instrumentation Trace Macrocell* (ITM).

• Support for the instruction trace option:

— Embedded Trace Macrocell (ETM). See the Arm® CoreSight™ ETM-M33 Technical Reference Manual for more information.

• Optional coprocessor interface for external hardware accelerators.

• Support for the *Custom Datapath Extension* (CDE) which adds classes of *Arm Custom Instructions* (ACIs) in the coprocessor instruction space.

• Low-power features including architectural clock gating, sleep mode, and a power aware system with optional *Wake-up Interrupt Controller* (WIC).

• A memory system, which can include optional memory protection and security attribution.

3.1 Floating point unit

The floating-point unit (FPU) may generate exceptions when used due to e.g., overflow or underflow. These exceptions will trigger the FPU interrupt. To clear the IRQ line when an exception has occurred, the relevant exception bit within the FPSCR register needs to be cleared. For more information about the FPSCR or other FPU registers, see *Cortex-M33 Devices Generic User Guide*.

3.2 CPU and support module configuration

The ARM® Cortex®-M33 processor has several CPU options and modules implemented on the device.

Table 3-1: ARM M33 configuration

	Option / Module	Description	Implemented
Core options	NVIC	Nested Vector Interrupt Controller	40 vectors
	PRIORITIES	Priority bits	3
	WIC	Wakeup Interrupt Controller	YES
	Endianness	Memory system endianness	Little endian
	Bit Banding	Bit banded memory	NO
	DWT	Data Watchpoint and Trace	YES
	SysTick	System tick timer	YES
Modules	MPU	Memory protection unit	YES
	EAM	AHB5 Exclusive Access Monitor	NO
	MSC	AHB5 Master Security Controller	YES
	PPC	AHB5/APB peripheral protection controller	YES
	MPC	AHB5 memory protection controller	YES
	GTZC	Global trustzone controller	YES
	FPU	Floating point unit	YES
	DAP	Debug Access Port	YES
	ETM	Embedded Trace Macrocell	YES
	ITM	Instrumentation Trace Macrocell	YES
	TPIU	Trace Port Interface Unit	YES
	ETB	Embedded Trace Buffer	NO
	FPB	Flash Patch and Breakpoint Unit	YES
	HTM	AHB Trace Macrocell	NO

3.3 IRQ

There are total 39 interrupt signals generated from different function blocks.

Table 3-2: List of IRQ signals

Irq no	Signal	Function module
0	s_irq_tz_ctrl	GTZC
1	s_irq_qspi	QSPI
2	s_dma_irq	DMA
3	s_spa_irq	SPA
4	s_pka_irq	PKA
5	s_trng_irq	TRNG
6	s_irq_crc	CRC
7	s_irq_gpio	GPIO
8	s_irq_spims	SPIMS
9	s_irq_uart[0]	UART0
10	s_irq_uart[1]	UART1
11	s_irq_i2c[0]	I2C
12	s_irq_timer[0]	TIMER0
13	s_irq_timer[1]	TIMER1
14	s_irq_timer[2]	TIMER2
15	s_irq_timer[3]	TIMER3
16	s_ext_irq_in_syn[0]	EXT0
17	s_ext_irq_in_syn[1]	EXT1
18	i_ble_irq	BLE
19	s_event_irq_syn	EVENT
20	dsr_event	UWB - DSR
21	rx_event[0]	UWB – RX
22	rx_event[1]	UWB – RX
23	rx_event[2]	UWB – RX
24	rx_event[3]	UWB – RX
25	rx_event[4]	UWB – RX
26	rx_event[5]	UWB – RX
27	rx_event[6]	UWB – RX
28	rx_event[7]	UWB – RX
29	rx_event[8]	UWB – RX
30	rx_event[9]	UWB – RX
31	rx_event[10]	UWB – RX
32	rx_event[11]	UWB – RX
33	tx_event[0]	UWB – TX
34	tx_event[1]	UWB – TX
35	gp_event[0]	UWB - GP
36	gp_event[1]	UWB - GP
37	gp_event[2]	UWB - GP
38	gp_event[3]	UWB - GP

4 AHB multilayer

The CPU and all peripherals with AHBM are AHB bus masters on the AHB multilayer, while the RAM and various other modules are AHB slaves. The CPU has exclusive access to all AHB slaves and the RAM. Access rights to each of the RAM AHB slaves are resolved using the priority of the different bus masters in the system. All bus masters to have the same priority to access each AHB slaves. It is guaranteed by design that the related masters will never be able to access the same slave at the same time.

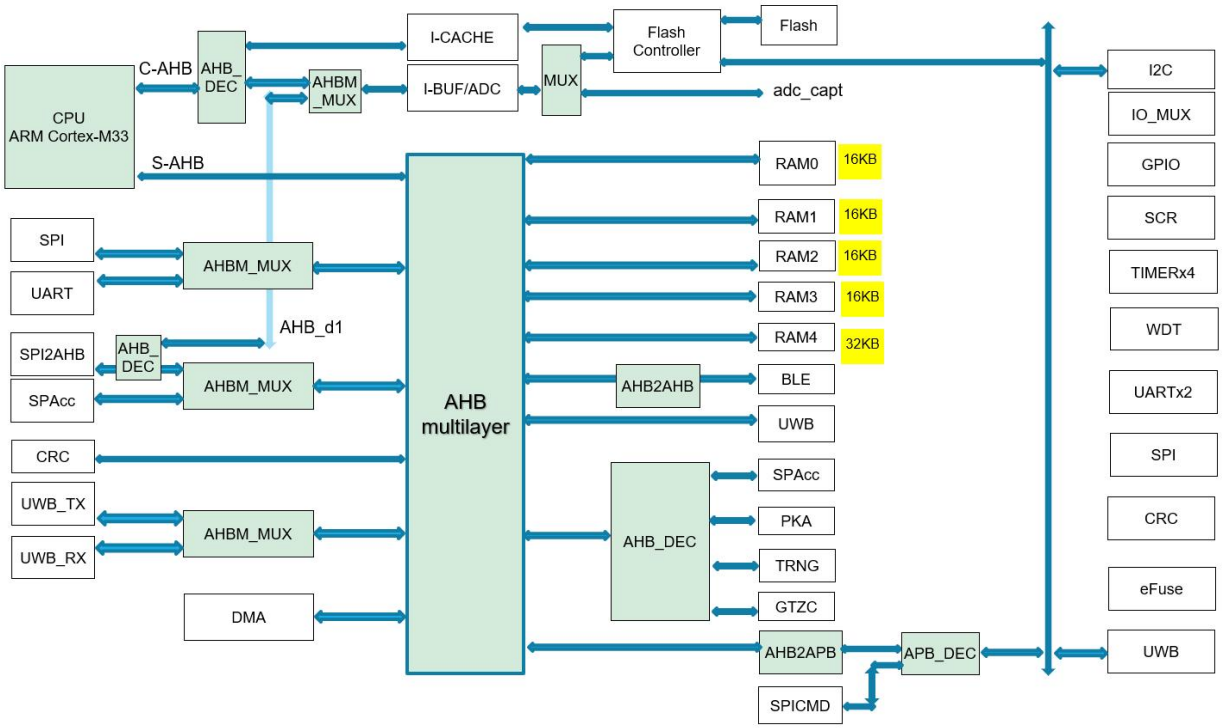


Figure 4-1: AHB multilayer interconnect

The SoC contains 96 KB RAM that can be used for data storage and a flash controller for code storage in an external flash device. The CPU and some peripherals with AHB master interface (AHBMI) can access memory via the AHB multilayer interconnect. The CPU is also able to access peripherals via the AHB multilayer interconnect.

4.1 Master devices

Table 4-1: List of Master devices

AHB Master index	AHB master	Notes
0	M33	
1	SAA	Shared by SAA and debug interface
2	SPIMS+UART	Shared by SPI and UART
3	CRC	
4	UWB	
5	DMA	

4.2 Slave devices

Table 4-2: List of slave devices

AHB slave index	AHB slave	Notes
0	Data ram 0	16KB
1	Data ram 1	16KB
2	Data ram 2	16KB
3	Data ram 3	16KB
4	Data ram 4	32KB
5	UWB CIR RAM	
6	BLE	
7	Security IP	Security IP consists of 4 slaves: SAA, PKG, TRNG, GTZC
8	APB bridge	Connects to 18 apb devices

4.3 Data RAM - Random access memory

The RAM interface is divided into 5 RAM AHB slaves. 4 RAM AHB slaves are connected to 4 16-kilobyte RAM. 1 RAM AHB slave connected to one 32-kilobyte RAM. The total RAM size is 96 Kbytes.

Each of the RAM sections have separate power control for System ON and System OFF mode operation, which is configured via SCR (system control register).

4.4 Flash - Non-volatile memory

The external flash is supported by SOC, which is used as code memory. The Flash can be read an unlimited number of times by the CPU, but it has restrictions on the number of times it can be written and erased and also on how it can be written. Writing to Flash is managed by the QSPI flash controller.

The Flash is divided into multiple pages (256 bytes per page) that can be accessed by the CPU via AHBC buses. Flash reserves some blocks start from address 0 for config data and system/user data, the sizes of which are defined in config data. The config data define the flash partition and some access commands.

4.5 Ibuf

There is one 256 KB instruction buffer used to load instruction codes from external flash starting from address 0 in boot load stage.

4.6 Icache

The 4-way instruction cache (ICACHE) is integrated in the cpu subsystem to improve the performance of reading instruction codes that is not in the ibuf. The icache can be enabled or bypassed according to the application requirement.

4.7 Memory map

The complete memory map is shown in *Figure 4-2*. Code RAM is implemented by external Flash memory device.

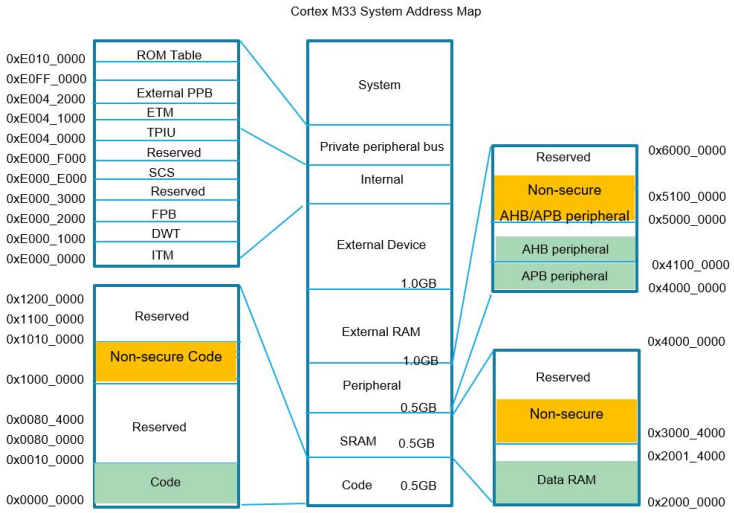


Figure 4-2: Memory map

The address map for each ahb and apb devices is listed in *Table 4-3*.

Table 4-3: Address map

Start Address	End Address	Devices	Notes
0x0000_0000	0x0003_FFFF	IBUF	AHB
0x2000_0000	0x2000_3FFF	DATA_RAM0	AHB
0x2000_4000	0x2000_7FFF	DATA_RAM1	AHB
0x2000_8000	0x2000_BFFF	DATA_RAM2	AHB
0x2000_C000	0x2000_FFFF	DATA_RAM3	AHB
0x2001_0000	0x2000_7FFF	DATA_RAM4	AHB
0x4100_0000	0x4100_1FFF	UWB4Z CIR RAM	AHB
0x4101_0000	0x4101_FFFF	BLE	AHB

0x4108_0000	0x410B_FFFF	SAA	AHB
0x410C_0000	0x410C_7FFF	PKG	AHB
0x410D_0000	0x410D_0FFF	TRNG	AHB
0x410F_0000	0x410F_0FFF	GTZC	AHB
0x4000_0000	0x400F_FFFF	APB devices	Stat with 0x40
0x4000_0000	0x4001_FFFF	UWB4Z	APB
0x4000_0000	0x4000_1FFF	UWB RX0 RFPLL/AFE	APB
0x4000_2000	0x4000_3FFF	UWB RX1 RFPLL/AFE	APB
0x4000_4000	0x4000_5FFF	UWB RX2 RFPLL/AFE	APB
0x4000_6000	0x4000_7FFF	UWB TX RFPLL/AFE	APB
0x4000_8000	0x4000_8FFF	UWB BBPLL	APB
0x4000_9000	0x4000_93FF	UWB ANN/DIG TX	APB
0x4000_9400	0x4000_97FF	UWB ANN/DIG RX0	APB
0x4000_9800	0x4000_9BFF	UWB ANN/DIG RX1	APB
0x4000_9C00	0x4000_9FFF	UWB ANN/DIG RX2	APB
0x4000_F000	0x4000_F0FF	UWB TOP CTRL	APB
0x4001_0000	0x4001_03FF	UWB Radio control	APB
0x4001_4000	0x4001_401F	UWB TRX control top	APB
0x4001_5000	0x4001_50FF	UWB TX DL	APB
0x4001_6000	0x4001_603F	UWB RX PHY/DL	APB
0x4001_7000	0x4001_71FF	UWB RX DFE/SYNC	APB
0x4001_B000	0x4001_B1FF	UWB TS SAMPLE	APB
0x4002_0000	0x4002_0FFF	SCR	APB
0x4002_1000	0x4002_11FF	DMA	APB
0x4002_2000	0x4002_2FFF	GPIO	APB
0x4002_3000	0x4002_307F	SPI 0	APB
0x4002_4000	0x4002_407F	SPI 1 (reserved)	APB
0x4002_5000	0x4002_503F	UART 0	APB
0x4002_6000	0x4002_603F	UART 1	APB
0x4002_7000	0x4002_703F	I2C	APB
0x4002_8000	0x4002_801F	WDT	APB
0x4002_9000	0x4002_903F	TIMER 0	APB
0x4002_A000	0x4002_A03F	TIMER 1	APB
0x4002_B000	0x4002_B03F	TIMER 2	APB
0x4002_C000	0x4002_C03F	TIMER 3	APB
0x4002_D000	0x4002_D0FF	IOMUX	APB
0x4002_E000	0x4002_E03F	CRC	APB
0x4002_F000	0x4002_F0FF	EFUSE	APB
0x4003_0000	0x4003_00FF	TIMER 0	APB

5 Power management

The chip has defined the following power states:

POWER-DOWN	The chip is power down, and can only be waked up by external pin.
DEEPSLEEP	AON is power on and the other digital circuits are power off. The 32KHz RC clock is enabled. The timer inside AON is used to wake up the chip to active state.
SLEEP	AON and basic CPU subsystem are power on. The CPU can be power on or off, and UWB, BLE and security modules are power off. The memory blocks ibuf, icache and data ram can be in retention mode or power off based on the system register setting. The timer inside AON is used to wake up the chip to active state.
ACTIVE	The SOC is power on and in active mode. The chip is ready to enter various function modes.
BLE	ACTIVE state and BLE is power on.
UWB	ACTIVE state and UWB is power on

Always-on module (AON) is responsible for the power management of the chip, which includes the sequence of power on, sleep and deep sleep. The chip will enter to ACTIVE state after power on. According to system command, the chip can switch from ACTIVE state to SLEEP or DEEPSLEEP state.

The chip enters ACTIVE state with MCU power on after wakeup from SLEEP, DEEPSLEEP or POWER-DOWN state. In this state, SOC automatically loads boot codes from flash and stores them in instruction buffer, then executes the codes from the instruction buffer. BLE and UWB can be activated or power off by firmware or software configuring registers in SCR.

In ACTIVE state, forcing the external chip enable pin to 0 is used to force the chip to POWER-DOWN state. Forcing this pin to 1 enable the chip back to ACTIVE state.

The chip can enter SLEEP or DEEPSLEEP state by configuring the corresponding SCR registers. The timer inside AON is used to wake up the chip to ACTIVE state.

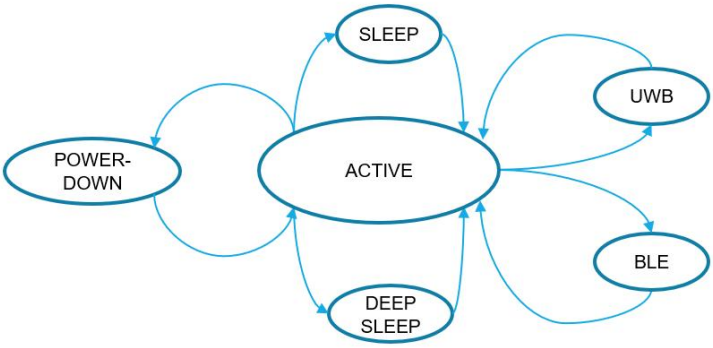


Figure 5-1: SOC Power State

5.1 AON power state

Always-on module (AON) is responsible for the power management of the chip, which includes the sequence of power on, sleep and deep sleep. The chip will enter to ACTIVE state after power on. According to system command, the chip can switch from ACTIVE state to SLEEP or DEEPSLEEP state.

RESET: after power on, AON is in reset state before power-on-reset (POR) is released.

EXTPEN: after POR is released, AON enters EXTPEN and output EXTEN to 1 to power on external DC-DC.

DIGPEN: In this state, GPIO is supplied and then internal VDD for digital circuit is supplied.

LDO_1v2: enable LDO for analog 1.2 V.

LDO_XO: enable LDO for XO.

ACTIVE: active state and ready to enter function modes.

SLEEP: XO is disabled. Internal digital VDD is available. IBUF, Data RAM and ICACHE can be configured to keep the memory contents in this state. MCU can also be switch off or switch on in this state.

DEEPSLEEP: only AON is power on. Other circuits are power off.

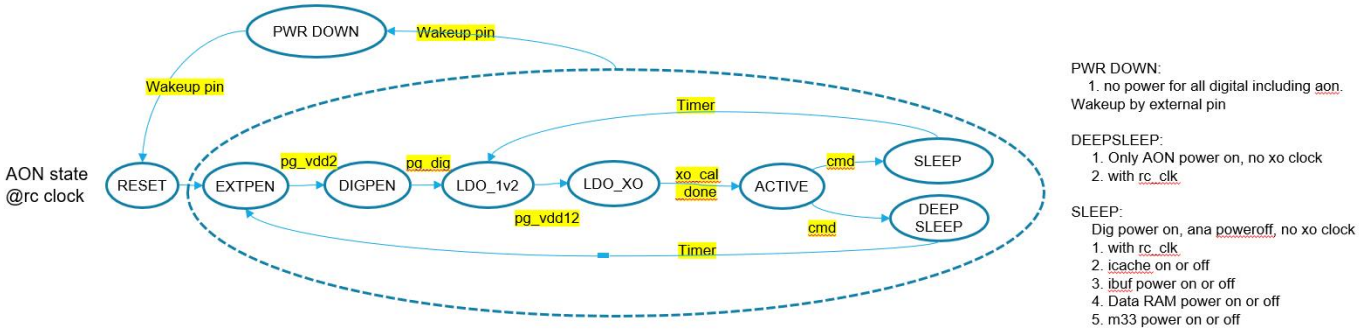


Figure 5-2: AON Power Management Unit

5.2 CPU power state

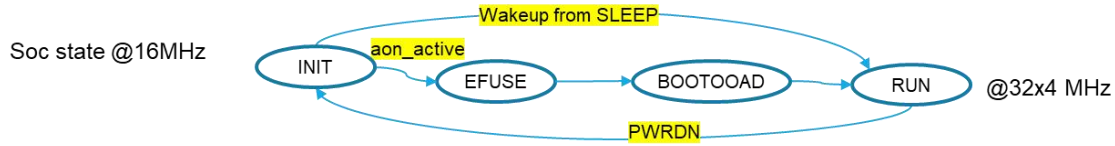


Figure 5-3: CPU Power state

The CPU subsystem undergoes a sequence of states starting with INIT state, which is progressed to when AON enters the active state. Then, it transitions to the eFuse state to load content from eFuse, accompanied by the release of certain control signals to ensure proper configuration loading. Once the eFuse content is loaded, the module enters the BOOT loading state. During this stage, instructions are loaded from the external flash memory to the instruction buffer (IBUF) using the QSPI controller. Once the boot loading is completed, the CPU subsystem proceeds to the RUN state, i.e., CPU_ON state of the chip. In the RUN state, it fetches and executes the code from the IBUF. The module remains in this state until the SoC undergoes a hard reset or entering POWER_DOWN state.

EFUSE and BOOTLOAD states can be skipped by writing 1 to SCR CPU_CTRL registers skip_init_efuse and skip_init_ibuf, respectively.

5.3 Power on control for individual modules

The following modules can be individually power on or off via SCR register.

- ✓ SAA
- ✓ PKG
- ✓ TRNG
- ✓ DMA
- ✓ UWB
- ✓ BLE

6 CLOCK — Clock generation

The following clocks are generated in the chip:

- 32 kHz RC oscillator
- 32 MHz on-chip oscillator, using external 32 MHz crystal
- 128MHz, 64MHz, 32MHz and 16 MHz clocks for CPU subsystem
- BBPLL to generate 499.2MHz clock for UWB
- Clock divider to generate clk_250m and clk_125m for UWB
- RFPLL to generate RF channel clock for UWB RX and TX

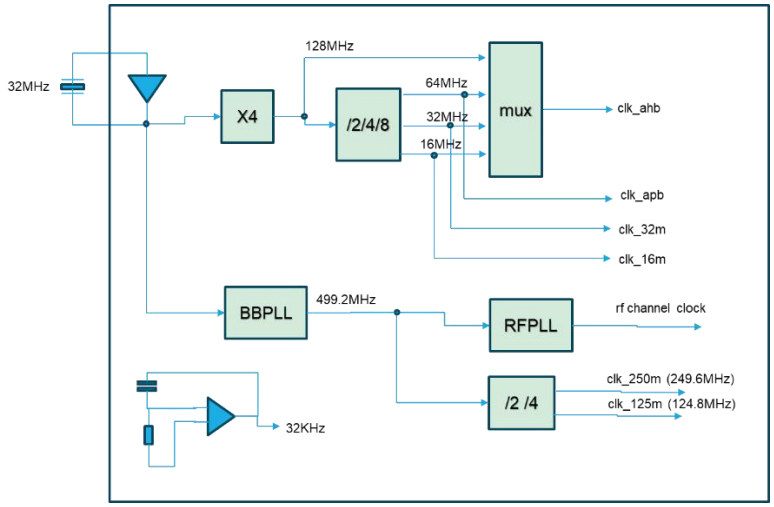


Figure 6-1: Clock generation

6.1 CPU subsystem clock

The different clock generated by the system clock generator modules are derived from the clock XO which operates at 128Mhz generated from the Analog module. The clock is then mux-ed with the scan clock if scan mode is selected and gated to generate the main xo clock used to generate the other clocks. A counter is employed to generate the subsequent clocks. Each clock corresponds to each bit of the counter for simpler clock generation as shown in *Figure 6-2*.

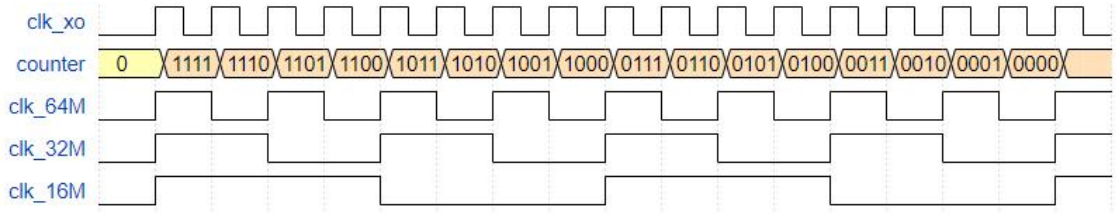


Figure 6-2: Clock generated from XO clock in CPU subsystem clock generation module

It also generates the AHB clock whose frequency can be chosen from 128MHz, 64MHz, 32MHz and 16 MHz by writing to the SCR registers offset 0x00 CPU_CTRL [24:23]. Refer section *SCR - System Configure Register.* The APB clock is fixed to 64Mhz.

6.2 UWB clock module

Uwb clock module generates 250MHz clock for rx and 125 MHz clock for tx and rx. The clocks to TX and RX can be enabled or disabled separately by writing uwb top control register for power saving purpose.

7 SCR - System Configure Register

The system operations and configurations are controlled by this register, which has a base address of 0x40020000. It governs reset, power, and various control signals for different modules. Notable registers within this system include:

1. The 'cpu_ctrl' register (offset: 0x00) manages the operations of the CPU and its associated modules.
2. The 'pson_ctrl' register (offset: 0x04) monitors and releases power controls for different power domains.
3. The 'rst_ctrl' register (offset: 0x08) releases reset signals to different modules, conserving power.
4. The 'sleep_mode' register (offset: 0x0C) handles sleep regulations for the AON (Always-On) module.
5. The 'status' register (offset: 0x10) monitors important flags.

Table 7-1: SCR register address

Offset	Reg	Range	Field	Access Type	Reset Value	Description
0x0	cpu_ctrl	[27]	icache_refresh	RW	1'b0	=1: Refresh icache. It will clear icache content.
		[26]	icache_bypass	RW	1'b0	=1: bypass icache
		[25]	ibuf_force_to_use	RW	1'b0	=1: force to use ibuf
		[24:23]	clk_sel	RW	2'd2	cpu amba/ahb clock selection 2'd0: 16MHz 2'd1: 32MHz 2'd2: 64 MHz 2'd3: 128MHz
		[20]	amba_rst_req	RW	1'b0	=1: Request to reset amba (ahb and apb register)
		[19]	ena_sysrst_amba	RW	1'b0	=1: enable system reset to reset ahb bus and apb bus
		[18]	sysrstn_soft	RW	1'b1	=0: generate sysreset_n signal for cpu =1: release sysreset_n
		[17]	strap_cpu_wait_disable	RW	1'b0	=1: Disable cpu_wait signal caused by strapping pin， also causes aon exit from dsleep mode
		[16]	ibuf_reload	RW	1'b0	=1: Reload instruction codes to ibuf
		[15]	eFuse_reload	RW	1'b0	=1: Reload eFuse data
		[14]	skip_init_eFuse	RW	1'b0	=1: skip eFuse loading before cpu run state =0: don't skip
		[13]	skip_init_ibuf	RW	1'b0	Skip ibuf loading and disable to use ibuf =1: skip ibuf initialization (i.e., loading) when wakeup from sleep =0: will not skip initialization
		[12]	pson_wic_in_sleep	RW	1'b0	=1: power on wic of cpu in sleep =0: power off wic of cpu in sleep
		[11]	pson_ram_in_sleep	RW	1'b0	=1: power on system ram in sleep =0: power off system ram in sleep
		[10]	pson_ibuf_in_sleep	RW	1'b0	=1: power on instruction buffer in sleep =0: power off instruction buffer in sleep
		[9]	pson_icache_in_sleep	RW	1'b0	=1: power on icache in sleep mode.
		[8]	pson_ram4	RW	1'b1	=1: power on ram_4
		[7]	pson_ram3	RW	1'b1	=1: power on ram_3
		[6]	pson_ram2	RW	1'b1	=1: power on ram_2
		[5]	pson_ram1	RW	1'b1	=1: power on ram_1
		[4]	pson_ram0	RW	1'b1	=1: power on ram_0
		[3]	pson_ibuf	RW	1'b1	=1: power on ibuf
		[2]	pson_icache	RW	1'b1	=1: power on for icache
		[1]	cpu_shutdown_en	RW	1'b1	=1: enable cpu shutdown in sleep mode
		[0]	cpu_maintain_power	RW	1'b0	=1: force cpu always power on
0x4	pson_ctrl	[25]	pson_ble_sec_ack	RO		
		[24]	pson_ble_ctrl_ack	RO		
		[23]	pson_ble_phy_ack	RO		

		[22]	pson_dma_ack	RO		
		[21]	pson_trng_ack	RO		
		[20]	pson_pkg_ack	RO		
		[19]	pson_saa_ack	RO		
		[18]	pson_ble_ack	RO		
		[17]	pson_uwb_ack	RO		
		[16]	pson_m33_ack	RO		
		[10]	pson_dma	RW	1'b0	=1: power on dma subsystem =0: power off dma subsystem
		[5]	pson_trng	RW	1'b0	=1: power on trng subsystem =0: power off trng subsystem
		[4]	pson_pka	RW	1'b0	=1: power on pka subsystem =0: power off pka subsystem
		[3]	pson_spacc	RW	1'b0	=1: power on spacc subsystem =0: power off spacc subsystem
		[2]	pson_ble	RW	1'b0	=1: power on ble subsystem =0: power off ble subsystem
		[1]	pson_uwb	RW	1'b0	=1: power on uwb subsystem =0: power off uwb subsystem
		[0]	pson_m33_en	RW	1'b1	=1: power on for M33 CPU =0: power off for M33 CPU
0x8	rst_ctrl	[24]	rst_dma	RW	1'b0	=1: reset release for DMA module
		[23]	rst_bist	RW	1'b0	=1: reset release for MBIST
		[22]	rst_eFuse	RW	1'b0	=1: reset release for EFUSE
		[20]	rst_timer3	RW	1'b0	=1: reset release for TIMER_3
		[19]	rst_timer2	RW	1'b0	=1: reset release for TIMER_2
		[18]	rst_timer1	RW	1'b0	=1: reset release for TIMER_1
		[17]	rst_timer0	RW	1'b0	=1: reset release for TIMER_0
		[16]	rst_gpio	RW	1'b0	=1: reset release for GPIO module
		[15]	rst_i2c	RW	1'b0	= 1: reset release for I2C
		[14]	rst_spi	RW	1'b0	=1: reset release for SPI_MS
		[13]	rst_uart1	RW	1'b0	=1: reset release for UART_1
		[12]	rst_uart0	RW	1'b0	=1: reset release for UART_0
		[11]	rst_crc	RW	1'b0	=1: reset release for CRC
		[10]	rst_trng	RW	1'b0	=1: reset release for TRNG
		[9]	rst_pkg	RW	1'b0	=1: reset release for PKA
		[8]	rst_saa	RW	1'b0	=1: reset release for SPACC
		[4]	rst_ble	RW	1'b0	=1: release reset for ble
		[0]	rst_uwb	RW	1'b0	=1: release reset signal for uwb
0xc	sleep_mode	[20:1]	sleep_time	RW	20'h00FFF	~128ms Number of cycles in sleep mode before wakeup
		[0]	force_sleep	RW	1'b0	Force to deepsleep or sleep mode
0x10	status	[31:7]	init_ns_vtor	RO		init ns vtor defined in flash
		[6]	ibuf_load_done	RO		ibuf load done
		[5]	waked_fr_sleep	RO		=1: current wakeup is from sleep. =0: current wakeup is from deepsleep.
		[4]	trustzone	RO		=1: trustzone enabled
		[3]	cpu_currns	RO		cpu secrity state: =0: cpu in security state. =1: cpu is in non-security state

		[2:0]	cpu_state	RO		cpu state: eFuse_load, boot_load, run
0x14	reserved					
0x18	reserved					
0x1C	reserved					
0x20	reserved					
0x24	reserved					
0x28	reserved					
0x2C	reserved					
0x30	reserved					
0x34	reserved					
0x38	reserved					
0x3C	reserved					
0x40	reserved					
0x44	reserved					

Please refer to Table 25-1 Explanation of Access in register list for “Access” definition

8 GTZC — Global trust zone controller

TrustZone is a security technology developed by ARM that provides hardware isolation and security features in SoC designs. The Global TrustZone Controller (GTZC) is a component within the SoC that manages and controls the TrustZone security features. The TrustZone can be majorly divided into different regions as

1. Secure (S)
2. Non-secure (NS)
3. Non-secure callable (NSC)

The *secure* region houses the secure information accessible only by secure masters, peripherals or secure memory regions. The *Non-secure callable* is also a type of secure location. Here the M33 permits to hold the SG (Secure Gateway) instruction that enables the software transition from non-secure to secure state. This feature removes the need of software creators to allows for the accidental inclusion of SG instructions or data sharing. When a non-secure program calls for a secure side, the first instruction to be performed is the Secure gate. The *Non-secure* region is accessible by all the software running on the device.

8.1 Features

1. The GTZC configures the TrustZone features and sets up the secure & non-secure domains within the SoC, thereby establishing boundaries of with secure execution.
2. It enables secure boot process and initialized the secure world for the CPU.
3. It provides *MPC (Memory protection controller)* which manages the secure memory regions and ensures that access to these regions is limited to trusted entities or secure execution contexts.
4. It handles interrupts and manages the interrupt routing between secure and non-secure worlds. It ensures that only authorized entities in the secure world can handle secure interrupts.
5. It provides *PPC (Peripheral Protection controller)* which controls access to secure peripherals or secure resources within the SoC. It ensures that only trusted software in the secure world can access and utilize these resources.
6. It provides *MSC (Master security controller)* which controls the AHB bus interface signals when the individual device is acting as a master independent of the CPU.

MPC, MSC and PPC play a critical role in enforcing security boundaries within the TrustZone system. The AHB protocol supports TrustZone feature with the help of HNONSEC signal. The relationship between CPU security state, SAU/IDAU and AHB bus signal HNONSEC is that for instruction fetches, the HNONSEC is always determined by the security state associated with the address. For data accesses, the external attribute on AHB HNONSEC is based on the associated security state specified by the SAU/IDAU. For example, a region defined as Non-Secure in the SAU/IDAU always sets the HNONSEC of the AHB request to 1’b1. Both the system bus and code bus of the M33 reflect the HNONSEC access.

The HNONSEC & HPROT bus is controlled by the master, hence when the CPU is the master the SAU & IDAU are utilized by the CPU to judge what is the value of these signals. When a device is in master mode, then it is responsible to set the correct values for these signals. It can judge these based on the MSC register settings.

The CPU security state can be monitored by reading the CURRNS signal in the SCR register offset 0x10 (status_cpu_currns [3]). This is an indication of the CPU state switch. It is an active low signal. CURRNS =0 indicates secure state, CURRNS=1 indicates non-secure state.

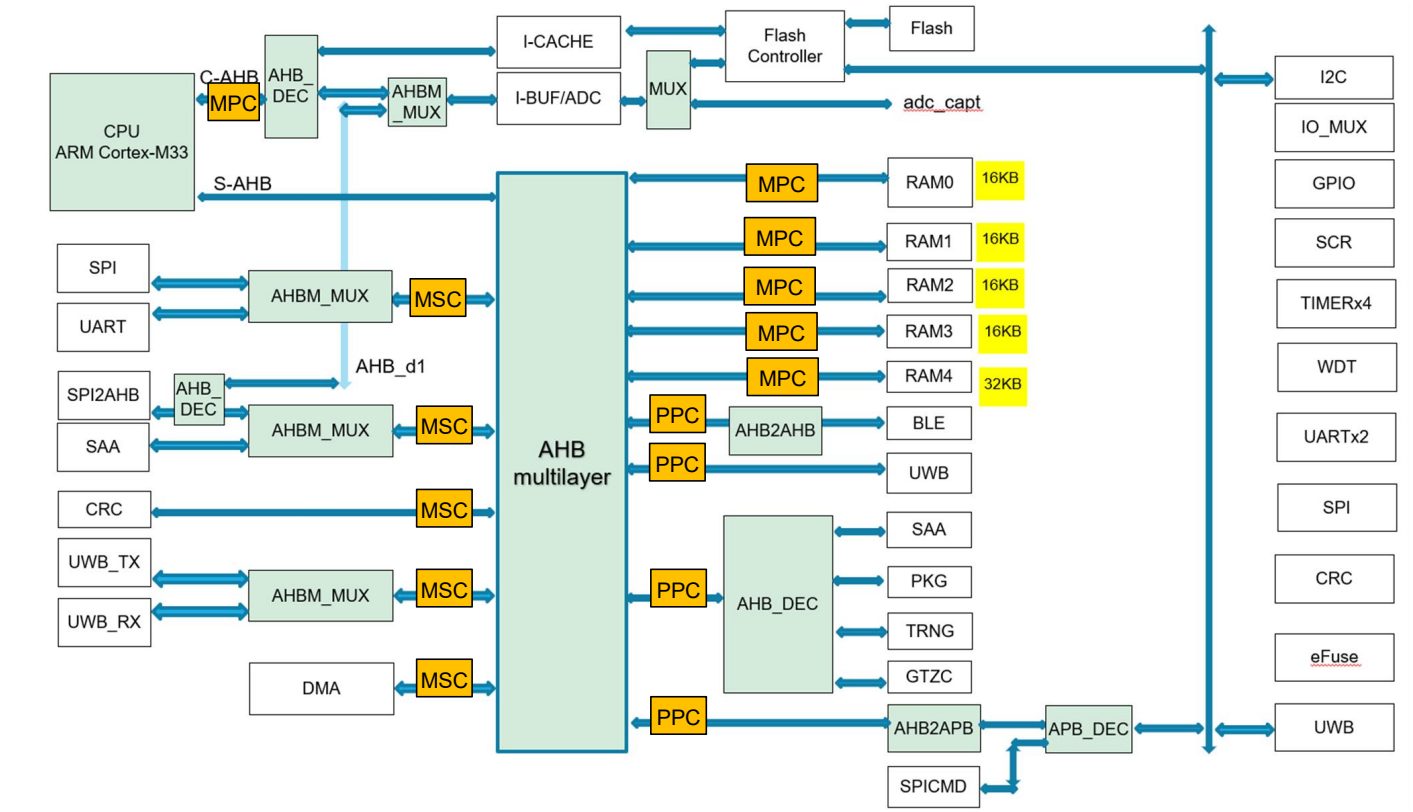


Figure 8- 1: Main architecture denoting MPC, MSC and PPC implementation

IDAU (Implementation Defined Attribution Unit) and SAU (Security Attribution Unit) are components that help enforce memory isolation and access control within the TrustZone-enabled system for the CPU. The IDAU provides coarse-grained control over memory attributes and permissions, while the SAU enables the assignment of security attributes to differentiate between secure and non-secure memory regions. The SAU is configurable through the firmware at run-time providing higher flexibility to overwrite configurations whereas the IDAU is hardcoded into the hardware.

9 EFUSE — eFuse controller

The eFuse or the Electrically programmable fuse allows to store permanent or one-time programmable data. It serves as a non-volatile memory element that can be electrically programmed. The eFuse model used here is TEF28HPCP16X32HD18_PHRM (512 bits High density Electrical Fuse). The controller acts as a bridge enabling the SoC to read, program and control the capabilities of the eFuse.

9.1 Functional description

1. The controller facilitates the programming of eFuse using the APB bus. It allows the SoC to write specific values or configurations to the cells in the eFuse for permanent storage.
2. It enabled reading of content in the eFuse, thereby accessing cryptographic keys and sensitive data.

3. The controller supports error checking mechanism to ensure the integrity of the programming & reading processes. It performs bit-level verification, redundancy check or error correction techniques to detect potential errors or inconsistencies.
4. The controller operates in different modes to program bits and repair wrong bits
5. The controller operates at 64Mhz clock frequency
6. Controller receives input data from the APB bus and saves in the array register

The structure of efuse_top and its related modules are shown *Figure 9-1*.

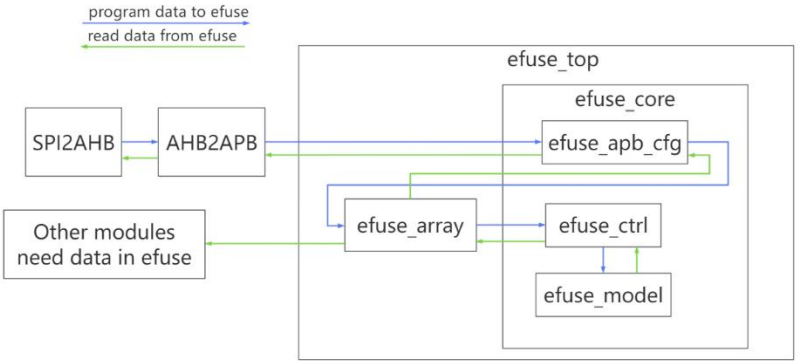


Figure 9-1: Block diagram of structure of efuse_top and its related modules

9.2 *Operation*

The eFuse controller mainly operates on a state machine as shown in Figure 9-2.

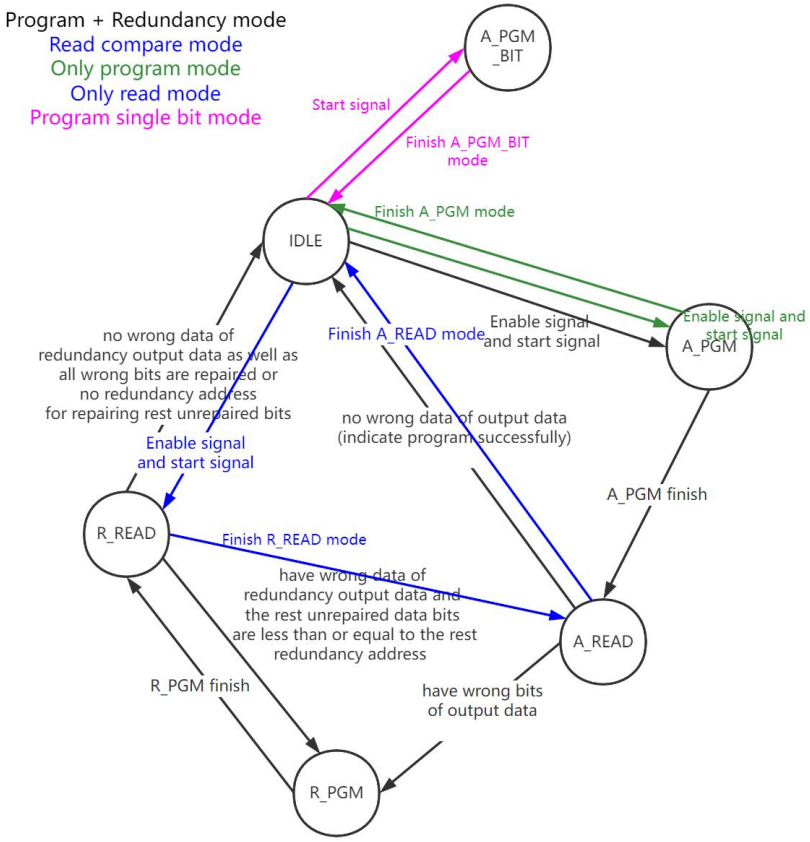


Figure 9-2: State Machine of eFuse controller

There are total 5 operation modes supported by the controller:

1. Program + Redundancy mode: IDLE → A_PGM → A_READ → IDLE
   → R_PGM → R_READ → IDLE
   → R_PGM → R_READ…
2. Read compare mode: IDLE → R_READ → A_READ → IDLE
3. Only program mode: IDLE → A_PGM → IDLE
4. Only read mode: IDLE → R_READ → A_READ → IDLE
5. Program single bit mode: IDLE → A_PGM_BIT

The eFuse address space can be explained as

1. 0~11 is used to save relevant parameters
2. 12~14 is used to repair wrong bits in address 0~11
3. 15 is used to map the repair information bits to the location spaces

In address 0, first bit indicates whether the eFuse can be used or not (0: can be used; 1: cannot be used), the second bit indicates whether the APB can read eFuse or not (0: can be read; 1: cannot be read).

(i). Program + Redundancy mode: Used to program data to eFuse as well as repair at most four wrong bits.

(a) IDLE state: Initial state, control each signal of eFuse to make it stay in power_down mode.

(b) A_PGM state: Turn to this state when both programs enable signal and start signal become high. Control each signal of eFuse work correctly in A_PGM mode. Program each bit(1'b1) one by one.

(c) A_READ state: Turn to this state when A_PGM is finished. Control each signal of eFuse work correctly in A_READ mode. Compare the output data bits from each address of eFuse to the original data bits program in the eFuse. If all output data bits are correct, turn to IDLE state and means programming successfully, else turn to R_PGM state for repairing wrong bits.

(d) R_PGM state: Control each signal of eFuse work correctly in R_PGM mode. Program each bit(1'b1) one by one. According to the redundancy working mechanism of eFuse, it can repair at most four wrong bits. The information (including the address, the location of wrong bit and the original correct value of bit) of wrong bits are need to program in the redundancy information row (RIR) of eFuse. The corresponding relation between redundancy bit and its address is shown in *Table 9-1*.

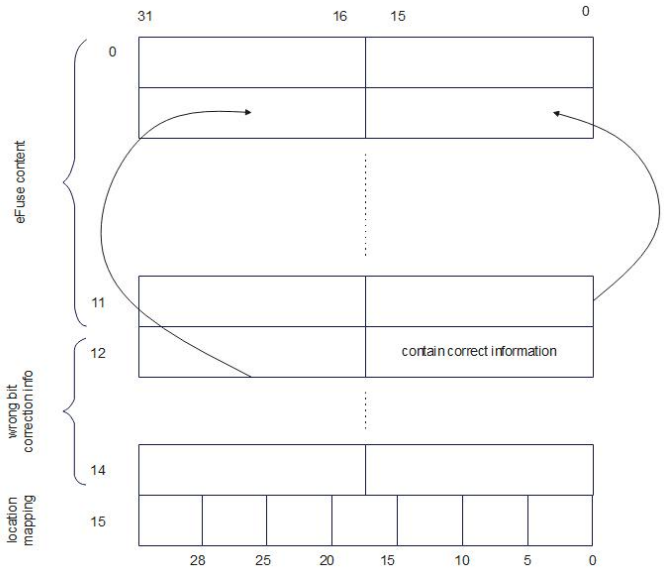


Figure 9-3: eFuse array control information

Table 9-1: The corresponding relation between redundancy bit and its address

	A8	A7	A6	A5	A4	A3	A2~A0

RF[0]	0	0	0	0	0	0	X
FB[0]_Data	0	0	0	0	1	0	X
FB[0]_A0	0	0	0	1	0	0	X
FB[0]_A1	0	0	0	1	1	0	X
FB[0]_A2	0	0	1	0	0	0	X
FB[0]_A3	0	0	1	0	1	0	X
FB[0]_A4	0	0	1	1	0	0	X
FB[0]_A5	0	0	1	1	1	0	X
FB[0]_A6	0	1	0	0	0	0	X
FB[0]_A7	0	1	0	0	1	0	X
FB[0]_A8	0	1	0	1	0	0	X
N/A	0	1	0	1	1	0	X
N/A	0	1	1	0	0	0	X
N/A	0	1	1	0	1	0	X
N/A	0	1	1	1	0	0	X
FB[0]_Disable	0	1	1	1	1	0	X
RF[1]	1	0	0	0	0	0	X
FB[1]_Data	1	0	0	0	1	0	X
FB[1]_A0	1	0	0	1	0	0	X
FB[1]_A1	1	0	0	1	1	0	X
FB[1]_A2	1	0	1	0	0	0	X
FB[1]_A3	1	0	1	0	1	0	X
FB[1]_A4	1	0	1	1	0	0	X
FB[1]_A5	1	0	1	1	1	0	X
FB[1]_A6	1	1	0	0	0	0	X
FB[1]_A7	1	1	0	0	1	0	X
FB[1]_A8	1	1	0	1	0	0	X
N/A	1	1	0	1	1	0	X
N/A	1	1	1	0	0	0	X
N/A	1	1	1	0	1	0	X
N/A	1	1	1	1	0	0	X
FB[1]_Disable	1	1	1	1	1	0	X

	A8	**A7**	**A6**	**A5**	**A4**	**A3**	**A2~A0**
RF[2]	0	0	0	0	0	1	X
FB[2]_Data	0	0	0	0	1	1	X
FB[2]_A0	0	0	0	1	0	1	X
FB[2]_A1	0	0	0	1	1	1	X
FB[2]_A2	0	0	1	0	0	1	X
FB[2]_A3	0	0	1	0	1	1	X
FB[2]_A4	0	0	1	1	0	1	X
FB[2]_A5	0	0	1	1	1	1	X
FB[2]_A6	0	1	0	0	0	1	X
FB[2]_A7	0	1	0	0	1	1	X
FB[2]_A8	0	1	0	1	0	1	X
N/A	0	1	0	1	1	1	X
N/A	0	1	1	0	0	1	X
N/A	0	1	1	0	1	1	X
N/A	0	1	1	1	0	1	X
FB[2]_Disable	0	1	1	1	1	1	X
RF[3]	1	0	0	0	0	1	X
FB[3]_Data	1	0	0	0	1	1	X
FB[3]_A0	1	0	0	1	0	1	X
FB[3]_A1	1	0	0	1	1	1	X
FB[3]_A2	1	0	1	0	0	1	X
FB[3]_A3	1	0	1	0	1	1	X
FB[3]_A4	1	0	1	1	0	1	X
FB[3]_A5	1	0	1	1	1	1	X
FB[3]_A6	1	1	0	0	0	1	X
FB[3]_A7	1	1	0	0	1	1	X
FB[3]_A8	1	1	0	1	0	1	X

N/A	1	1	0	1	1	1	X
N/A	1	1	1	0	0	1	X
N/A	1	1	1	0	1	1	X
N/A	1	1	1	1	0	1	X
FB[3]_Disable	1	1	1	1	1	1	X

Note:

(1). A8~A0 is the address of RIR(Redundancy Information Row).

(2). RF[n] is the flag to record whether the redundancy bit has been used or not.

(3). FB[n]_Data is the correct data of n^{th} redundancy bit, which will be stored in RIR.

(4). FB[n]_A8~A0 is the address data of n^{th} redundancy bit, which will be stored in RIR.

(5). FB[n]_Disable is used to disregard the n^{th} redundancy bits. When set to "1", this repaired bit will be ineffective and disregarded.

If the number of wrong bits is more than one, program the first wrong bit information at first. Then turn to R_READ state, if the output from eFuse in R_READ mode is correct, next program the second wrong bit information, else it is not correct, program its information again until the output is correct or there is no redundancy address left.

(e) R_READ state: Turn to this state when R_PGM is finished. Control each signal of eFuse work correctly in R_READ mode. Compare the output data from address ([A3] = [0] or [1] depends on which address was programmed in R_PGM state) of eFuse RIR to the original data program in the eFuse. If output data is correct and all wrong data bits have been repaired, turn to IDLE state and means programming successfully, else output data is correct but still has other wrong bits that are not repaired, turn to R_PGM state for repairing other wrong bits. If output data is wrong and the rest redundancy address is enough to the rest unrepaired bits, turn to R_PGM state for repairing this bit again, else turn to IDLE state and means programming unsuccessfully.

(ii). Read compare mode: Used to get information of remaining wrong bits after eFuse redundancy repairing

(a) IDLE state: Initial state, control each signal of eFuse to make it stay in power_down mode.

(b) R_READ state: Turn to this state when both enable read compare signal and start signal become high. Control each signal of eFuse work correctly in R_READ mode. Load the repaired bits information in eFuse.

(c) A_READ state: Turn to this state when R_READ is finished. Control each signal of eFuse work correctly in A_READ mode. Compare the output data bits from each address of eFuse to the original data bits program in the eFuse. After finishing A_READ mode, turn to IDLE state and output the information of wrong bits(if has).

(iii). Only program mode: Used to program correct data of wrong bits to the empty word of eFuse for further repairing

(a) IDLE state: Initial state, control each signal of eFuse to make it stay in power_down mode.

(b) A_PGM state: Turn to this state when both enable program only signal and start signal become high. Control each signal of eFuse work correctly in A_PGM mode. Program each bit(1'b1) one by one. After finishing A_PGM mode, turn to IDLE state.

(iv). Only read mode: Used to read data from eFuse

(a) IDLE state: Initial state, control each signal of eFuse to make it stay in power_down mode.

(b) R_READ state: Turn to this state when both enable read only signal and start signal become high. Control each signal of eFuse work correctly in R_READ mode. Load the repaired bits information in eFuse.

(c) A_READ state: Turn to this state when R_READ is finished. Control each signal of eFuse work correctly in A_READ mode. Output the data in eFuse. After finishing A_READ mode, turn to IDLE state.

(v). Program single bit mode: Used to program single bit to the eFuse after all other modes for indicating whether the eFuse can be used and read

(a) IDLE state: Initial state, control each signal of eFuse to make it stay in power_down mode.

(b) A_PGM_BIT state: Turn to this state when start signal becomes high. Control each signal of eFuse work correctly in A_PGM mode. Program single bit(1'b1). After finishing A_PGM mode, turn to IDLE state.

Definition of first word of array(a[0]): a[0][1]: eFuse still has wrong bits; a[0][2]: apb read disable.

9.3 eFuse array content description

Table 9-2: eFuse array description

Word	Description
Word 0	Efuse_programmed, efuse readable
Word 1	Trim value
Word 2	Trim value
Word 3	CHIP_ID (version, slot, package, pkg year, pkg week)
Word 4	aes upper 32bits to form 96 bits input
Word 5	128 bits key for flash: bit [31:0]
Word 6	128 bits key for flash: bit [63:32]
Word 7	128 bits key for flash: bit [95:64]
Word 8	128 bits key for flash: bit [127:96]
Word 9	32 bits random passcode for debug authentication
Word 10	CHIP_ID (wafer id, row, column)
Word 11	Reserved for user
Word 12	Reserved for repairing
Word 13	Reserved for repairing
Word 14	Reserved for repairing
Word 15	***Not Used***

N.B.: Word1, word2, word3 and word10 are used for trim value, chip id. They shall not be changed by user. Word 5 to word 8 are used for storing AES key of flash controller. Word 4 is also used for AES input, which can be used as a key for end user. Word 9 is reserved for custom to enable/disable the debugging features such as SWD debugging. Word 12 to word 15 are reserved for efuse repairing, should be not used by custom.

Table 9-3: eFuse array word 0 description

Word 0	Description
0	Word 0 read disable
1	Word 1 read disable
2	Word 2 read disable
3	Word 3 read disable
4	Word 4 read disable
5	Word 5 read disable
6	Word 6 read disable
7	Word 7 read disable
8	Word 8 read disable
9	Word 9 read disable
10	Word 10 read disable
11	Word 11 read disable
12	aes_enable
13	Disable spi2ahb interface
14	Disable spi2ahb access ibuf
15	Dsiable spi command

Table 9-4: eFuse array word 1 description

Word 1	Description
[3:0]	Reserved
[8:4]	trim_ibg_res
[9]	trim_ibg_res enable
[14:10]	trim_vbg_res
[15]	trim_vbg_res enable
[20:16]	rx0_set_lpf_ibias
[25:21]	rx1_set_lpf_ibias
[30:26]	rx2_set_lpf_ibias
[31]	rx_set_lpf_ibias_en

Table 9- 5: eFuse array word 2 description

Word 1	Description
[3:0]	trim_aon_0v8
[4]	trim_aon_0v8 enable
[31:5]	Reserved

9.4 Efuse functions based on working mechanism

1. Program data to efuse word by word:
   Using apb to set start programming address and end programming address of efuse, as well as specific data content with corresponding word address. Trigger start after finishing all operations.
2. Program single bit data to efuse(make this bit change from 0 to 1):
   Using apb to set address of single bit, the address(A8~A0) format: A3~A0: address of word bank; A8~A4: each bit location of word(e.g. Word 5 bit 2: A[3:0]=4'b0101, A[8:4]=5'b00010). Trigger start after finishing all operations.
3. Read data from efuse word by word:
   Using apb to set start reading address and end reading address of efuse. Trigger start after finishing all operations. Then read data from register by setting corresponding word address.
4. Read data from efuse word by word and compare whether data are correct:

Using this function after program data to efuse. Using apb to set start reading address and end reading address of efuse. Trigger start after finishing all operations. Then read data from fail related registers to check the fail programming flag and specific fail programming bit address.

9.5 Registers

The registers for the efuse_top module are as listed in *Table 9-6*.

Base address of efuse_top module: 0x4002F000.

Table 9- 6: The registers for the efuse top module

Offset	Reg	Range	Field	Access	Reset Value	Note
0x0	EFUSE_MODE	[3]	EFUSE_READ_COMP_EN	RW	1'b0	Enable efuse_ctrl read data from efuse word by word and compare that to the original data in array register for checking whether it has fail programming bits
		[2]	EFUSE_PGM_ONLY_EN	RW	1'b0	Enable efuse_ctrl program data to efuse word by word
		[1]	EFUSE_READ_ONLY_EN	RW	1'b0	Enable efuse_ctrl read data from efuse word by word.
0x4	EFUSE_CTRL	[0]	START	WP	1'b0	Start of doing operation to efuse
0x8	EFUSE_ADDR	[3:0]	EFUSE_ADDR_STA	RW	4'b0	Start address of data written to the array register in efuse_ctrl
		[7:4]	EFUSE_ADDR_END	RW	4'd11	End address of data written to the array register in efuse_ctrl
0xc	EFUSE_REG_ADDR	[3:0]	ADDR	RW	4'b0	Address of data written to the array register
0x10	EFUSE_REG_WR	[31:0]	WDATA	WE	32'b0	Data written to the array register corresponding address
0x14	EFUSE_REG_RD	[31:0]	RDATA	RO		Efuse word read from register array
0x18	EFUSE_PGM_BIT	[10:2]	BIT_ADDR	RW	9'd0	Address of data bit written to the array register in bit program mode
		[1]	BIT_VALUE	RW	1'b1	

		[0]	START	WP	1'b0	Indicate start of doing operation to efuse in bit program mode
0x1c	EFUSE_STATUS	[1]	EFUSE_FAIL_RESULT	RO		Indicate whether all bits are programmed correctly or not
0x20	EFUSE_FAIL_BIT00	[8:0]	INFO	RO		The information of wrong bits after read compare mode
0x24	EFUSE_FAIL_BIT01	[8:0]	INFO	RO		
0x28	EFUSE_FAIL_BIT02	[8:0]	INFO	RO		
0x2c	EFUSE_FAIL_BIT03	[8:0]	INFO	RO		
0x30	EFUSE_FAIL_BIT04	[8:0]	INFO	RO		
0x34	EFUSE_FAIL_BIT05	[8:0]	INFO	RO		
0x38	EFUSE_FAIL_BIT06	[8:0]	INFO	RO		
0x3c	EFUSE_FAIL_BIT07	[8:0]	INFO	RO		

Please refer to Table 25-1 Explanation of Access in register list for “Access” definition

10 QSPI flash controller

The Quad serial peripheral interface (QSPI) controller acts as the bridge between the external Flash memory and the user which aids to program, erase and retrieve information via APB interface. This provides faster data transfer compared to traditional SPI interface.

10.1 Functional Features

1. The controller handles the QSPI commands for NOR flash memory and controls the overall operation of the interface. It manages the timing, sequencing, and configuration of data transfers.
2. The operating clock frequency of QSPI controller is 64Mhz for the input clock and APB bus
3. It has internal registers to convert parallel data to serial for data transmission based on the command being configured to the flash.
4. It generates the clock, CS and data lines which in turn connects to the external flash through the IO pads.
5. Supports access of the external flash memory through AHB bus, APB bus and memory bus.
6. It holds registers which can be configured to send different commands based on the user applications.
7. It also handles the addressing complexity when fetching or programming the flash.

The *Figure 10-1* shows internal blocks and external peripheral interaction of the QSPI controller with other modules.

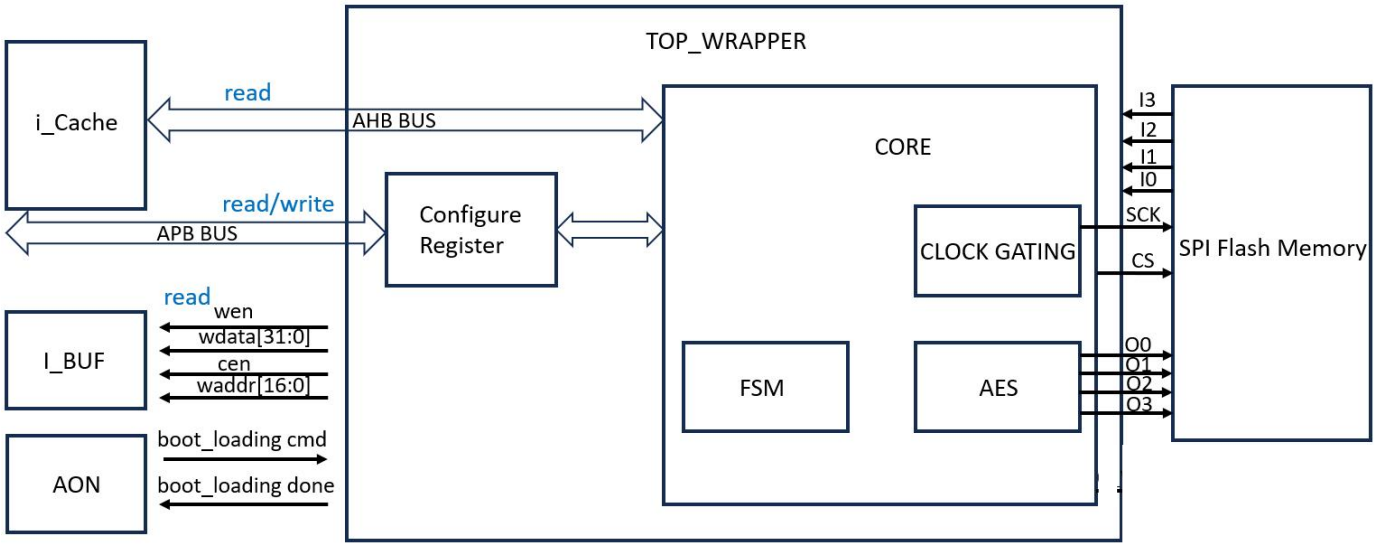


Figure 10-1: Block diagram of the QSPI controller and its peripheral interfaces

For operation reasons, the flash memory is divided as three regions, as shown in *Figure 10-2*.

- User configurations
- System Configurations
- Data storage

The user configuration area is used to indicate some useful information relating to the flash and its operations. The system configuration space is used to contain any special system information. The size of this page can be modified or if found not useful can be set to 0. The actual data storage contains the FW code or instructions.

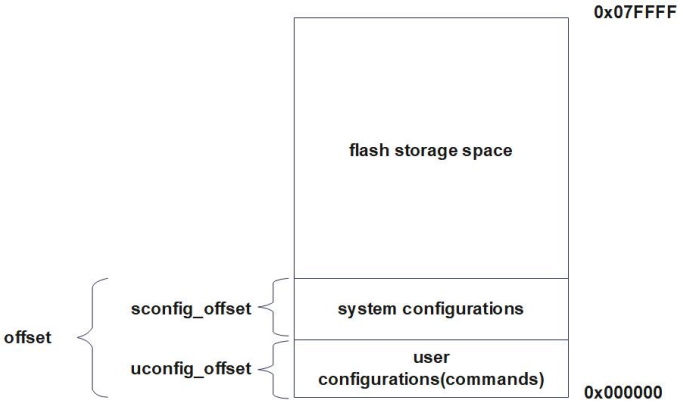


Figure 10-2: Flash memory space segregation

The bytes in the user configurations space are organized as shown in *Table 10-1*.

Table 10-1: User configuration byte with their description

Byte_offset	Name	Description
0x00	uconfig_flag	==A2 denotes that the user configuration area has been configured (flag is reflected in APB register)
0x01	code_config_flag	==A1 denotes that code has been configured (flag is reflected in APB register)
0x02	uconfig_page	Denotes the number of pages allocated for vendor specific information
0x03	sconfig page	Denotes the number of pages allocated for system data
0x04	sts_cmd_byte	Number of bytes to be used for write status register (min value 1; cannot be 0)
0x05	sts_write_cmd	Command to write the status register
0x06	sts_write_byte1	1st byte for the write status register command
0x07	sts_write_byte2	2nd byte for write status register command
0x08	sts_read_cmd	Command to read the status register
0x09	sts_write_mask_qe	Mask to know the quad bit
0x0A	sts_read_mask_wip	Mask to know the WIP bit
0x0B	burst_ndummy	[3:0] - Denotes the number of dummy cycles in the burst_read command [7:4] - Denotes the mode in which the command and wrap code needs to be sent =0 Standard SPI =1 QSPI mode
0x0C	burst_cmd	Command to set the BURST_READ mode
0x0D	burst_set_16	Wrap code for BURST_READ (WRAP_4)
0x0E	burst_set_32	Wrap code for BURST_READ (WRAP_8)
0x0F	burst_reset_code	Burst reset code for the BURST_READ
0x10	trustzone_en	==A3 trustzone is enabled
0x11	init_ns_vtor[7:0]	Address [7:0] for NS VTOR table

0x12	init_ns_vtor[15:8]	Address [15:8] for NS VTOR table
0x13	init_ns_vtor[23:16]	Address [23:16] for NS VTOR table
0x14	init_ns_vtor[31:24]	Address [31:24] for NS VTOR table
0x15	ubuf_base_ucf	Address in U_buffer
0x16	ubuf_size_ucf	Size to be loaded in U_buffer
0x17	ubuf_flash_addr_ucf	Flash address from which we want to load the ubuf [07:0]
0x18	ubuf_flash_addr_ucf	Flash address from which we want to load the ubuf [15:8]
0x19	ubuf_ucf_vld	=A4 indicates that the provided size and address for the Boot loading is valid (bytes 0x15-0x18)

The QSPI controller operates mainly on a state machine. The state machine has been coded in a way to optimize and support the major operations of the controller with respect to the flash.

State	Description
IDLE	The controller waits for new incoming commands
CONFIG_LOAD	Load the user configuration bytes from flash
WREN_EN	Set the write enable latch in the flash
QUAD	Send the quad enable command to flash
RDSR	Continuously reads the status register for the WIP pin
BURST_SET	Sets/resets the burst mode depending on incoming command
AHB_READ	Performs incoming command from icache using AHB bus
APB_CMD	Performs incoming command through APB bus
BOOT_LOAD	Loads the i_buffer or u_buffer content from the flash

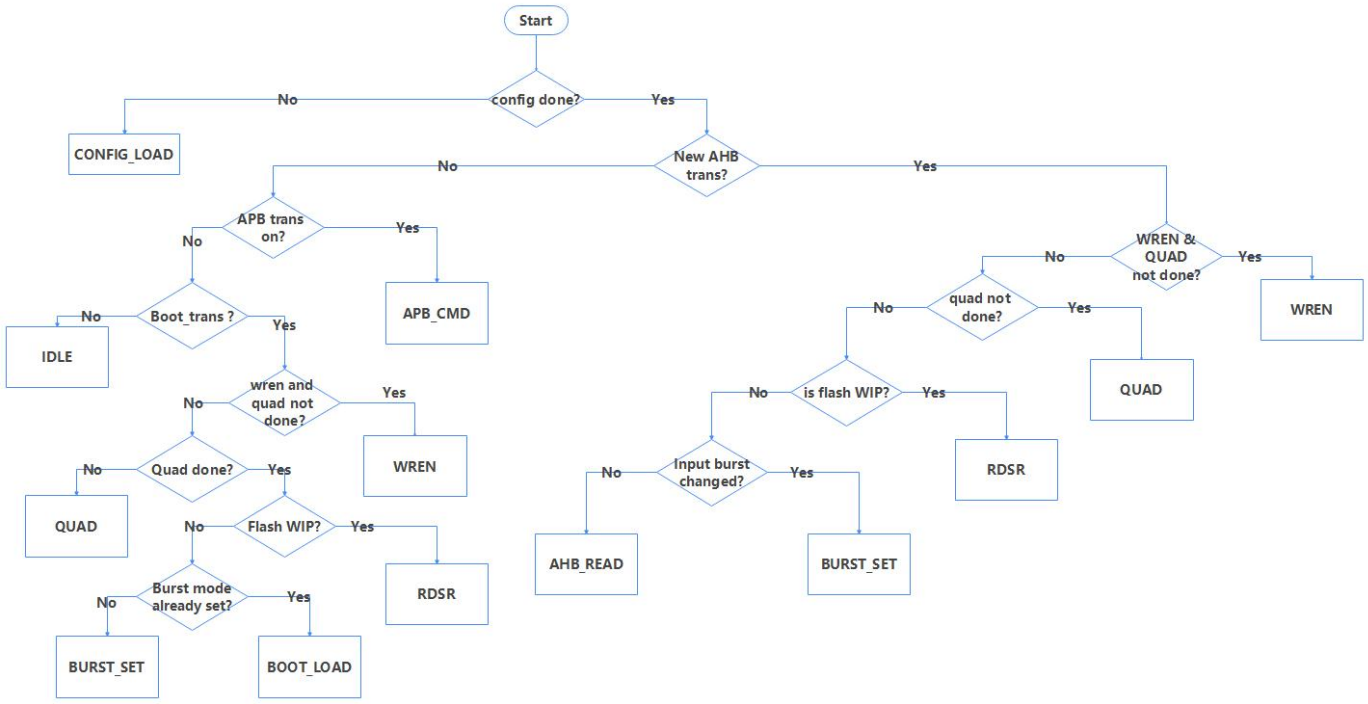


Figure 10- 3: FSM of QSPI state machine

11 GPIO — General purpose input/output

11.1 Function description

In this chip there are 20 IO pins are defined. These IO pins are used Inputs/Outputs for external data communication. Out of these 20 pins there are:

- 12 GPIO pins – GPIO00, GPIO01, GPIO02, GPIO03, GPIO04, GPIO05, GPIO6, GPIO07, GPIO08, GPIO09, GPIO10, GPIO11
- 6 QSPI pins – QSPI_CSn, QSPI_CLK, QSPI_SIO0, QSPI_SIO1, QSPI_SIO2, QSPI_SIO3
- TEST pin, EXTEN pin.

Using IOMUX, these GPIO's can be configured to use for different peripheral interfaces such as SPI, UART, MBIST, JTAG, SWD. These pins can also use to output some key event signals.

As the name indicates, IOMUX is a selection mux which enables flexible and configurable connectivity between different peripheral signals or events and GPIO Pins.

11.1.1 Strapping Pins

The GPIO00, GPIO01, GPIO02, GPIO08 Pins are used as strapping pins to define the different function modes after power ON. The TEST pin should be set to 0 and EXTEN pin is not configured. The function modes which can be configured are given in the below table. In strapping mode (all mode except GPIO mode), GPIO 8, 9, 10 and 11 are reserved for debugging, JTAG and SWD. In normal GPIO mode, GPIO 8, 9, 10, 11 can be used for GPIO or device IO.

Table 11-1: GPIO Pins configuration for strapping modes

Function mode	GPIO00	GPIO01	GPIO02	GPIO08	TEST
Debug mode 1	-	0	0	1	0
Debug mode 2	-	1	0	1	0
JTAG	0	1	1	1	0
JTAG (cpu_run)	1	1	1	1	0
SWD (9,11)	1	1	1	0	0
SWD (6,7)	0	1	1	0	0
GPIO mode	0	0	0	0	0

11.1.1.1 JTAG Interface

If the strapping pins are set to JTAG, JTAG (cpu_run) function mode then either GPIO08 ~ GPIO11 or GPIO04 ~ GPIO07 pins are used for JTAG Interface based on IOMUX.DEBUG_IO reg config.

If IOMUX.DEBUG_IO[3:2] = 2'b01

Table 11-2: GPIO Config for JTAG Interface

Pins	Interface signals
GPIO04	JTAG_TMS
GPIO05	JTAG_CLK
GPIO06	JTAG_TDI
GPIO07	JTAG_TDO

If IOMUX.DEBUG_IO[1:0] = 2'b11

Pins	Interface signals
GPIO08	JTAG_TMS
GPIO09	JTAG_CLK
GPIO10	JTAG_TDI
GPIO11	JTAG_TDO

11.1.1.2 SWD Interface

If the strapping pins are set to SWD (9,11) then GPIO09, GPIO11 are used for SWD Interface. In this mode GPIO 10 can be input only.

Table 11-3: GPIO Config for SWD Interface

Pins	Interface signals
GPIO09	SWDCK
GPIO11	SWDIO

If the strapping pins are set to SWD (6,7) then GPIO06, GPIO07 are used for SWD Interface. In this mode GPIO 10 can be input only.

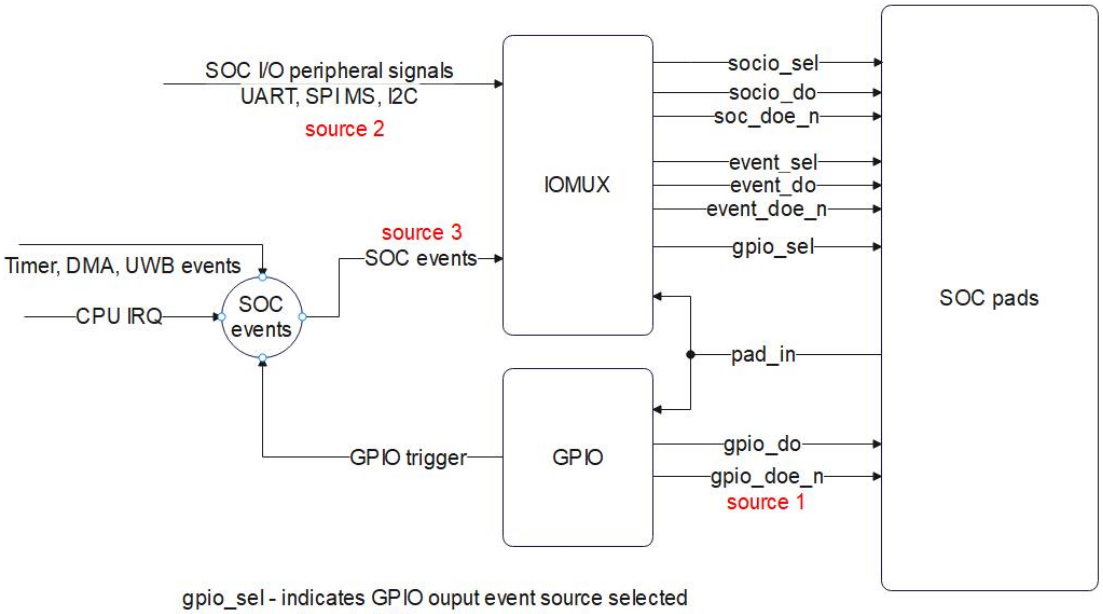
Pins	Interface signals
GPIO06	SWDCK
GPIO07	SWDIO

11.1.2 Input/Output Event Signals

The GPIO pins can be configured to input or output some event signals. Using IOMUX and GPIO registers, we can choose GPIO events, IO peripheral signals, SOC events to connect to GPIO pins. If the strapping pins are used to set any peripheral interfaces (SPIC, JTAG, SWD) then the respective GPIO pins cannot be used to configure to connect the event signals. The GPIO pads are defined in SOC pads module.

The Events to the GPIO are from three sources (see *Table 11-2*, *Table 11-3*, *Table 11-4* for complete list):

1. GPIO output signal from GPIO module
2. SOC IO Peripheral signals from SPI MS, UART, I2C modules
3. SOC Events – CPU_IRQ, Timer events, GPIO Trigger events, DMA and UWB events



gpio_sel - indicates GPIO ouput event source selected
gpio_do - selected GPIO ouput event signal
gpio_doe_n - enable signal for event
pad_in - GPIO input signal

socio_sel - indicates SOC IO peripherl signal source selected
socio_do - selected SOC output peripheral signal
socio_doe_n - enable signal for event
pad_in - when socio selected then the inout signal corresponds to SOC input peripheral signal

event_sel - indicates SOC event signal source selected
event_do - selected SOC event signal
event_doe_n - enable signal for event

Figure 11-1: Event select circuit

11.2 Registers

11.2.1 IOMUX Registers

Table 11-4: List of IOMUX registers

Offset	Reg	Range	Field	Access	Reset Value	Description
0x00	debug_io	[3:2]	gpio4_7	RW	2'd0	Force gpio4~7 to be debugging interface to JTAG or SWD
		[1:0]	gpio8_11_sel	RW	2'd0	Force gpio8~11 to be debugging related interface.
0x04	gpio00	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd0	Configure index of the event signals to map to GPIO00

0x08	gpio01	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd1	Configure index of the event signals to map to GPIO01
0x0c	gpio02	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd2	Configure index of the event signals to map to GPIO02
0x10	gpio03	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd3	Configure index of the event signals to map to GPIO03
0x14	gpio04	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd12	Configure index of the event signals to map to GPIO04
0x18	gpio05	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd13	Configure index of the event signals to map to GPIO05
0x1c	gpio06	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd14	Configure index of the event signals to map to GPIO06
0x20	gpio07	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd15	Configure index of the event signals to map to GPIO07
0x24	gpio08	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd16	Configure index of the event signals to map to GPIO08
0x28	gpio09	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd9	Configure index of the event signals to map to GPIO09
0x2c	gpio10	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd18	Configure index of the event signals to map to GPIO10
0x30	gpio11	[9:8]	mode_sel	RW	2'd0	Configure Event mode select
		[5:0]	src_sel	RW	6'd19	Configure index of the event signals to map to GPIO11
0x34	spi_trig	[6:0]	spi_start	RW	7'd127	To trigger spi_start signal based on any SOC events
0x38	uart0_trig	[14:8]	tx_start	RW	7'd127	To trigger uart0_tx start signal based on any SOC events
		[6:0]	rx_start	RW	7'd127	To trigger uart0_rx start signal based on any SOC events
0x3c	uart1_trig	[14:8]	tx_start	RW	7'd127	To trigger uart1_tx start signal based on any SOC events
		[6:0]	rx_start	RW	7'd127	To trigger uart1_rx start signal based on any SOC events
0x40	time0_trig	[14:8]	capture0	RW	7'd127	To capture timer0 based on any SOC events
		[6:0]	start	RW	7'd127	To trigger timer0 start signal based on any SOC events
0x44	time1_trig	[14:8]	capture0	RW	7'd127	To capture timer1 based on any SOC events
		[6:0]	start	RW	7'd127	To trigger timer1 start signal based on any SOC events
0x48	time2_trig	[14:8]	capture0	RW	7'd127	To capture timer2 based on any SOC events
		[6:0]	start	RW	7'd127	To trigger timer2 start signal based on any SOC events

0x4c	time3_trig	[14:8]	capture0	RW	7'd127	To capture timer3 based on any SOC events
		[6:0]	start	RW	7'd127	To trigger timer3 start signal based on any SOC events
0x50	i2c_trig	[6:0]	i2c_start	RW	7'd127	To trigger I2C start signal based on any SOC events
0x54	crc_trig	[6:0]	crc_start	RW	7'd127	To trigger CRC start signal based on any SOC events
0x58	events_irq	[6:0]	sel	RW	7'd127	Select one event as IRQ to CPU

Please refer to Table 25-1 Explanation of Access in register list for “Access” definition

11.2.1.1 DEBUG_IO Register

This register is used to force GPIO pins to set to debugging interfaces like JTAG, SWD, SPI_CMD, SPI2AHB.

Bits	Reg Name	Access	Reset Value	Description
[3:2]	gpio4_7	RW	2’b00	Force GPIO04 to GPIO07 to set to debugging interfaces – JTAG, SWD • 2’b00 – The pins will be set based on strapping pin config • 2’b01 – Set the pins to JTAG Interface • 2’b10 – Set the pins to SWD Interface
[1:0]	gpio8_11_sel	RW	2’b00	Force GPIO08 to GPIO11 to set to debugging interfaces – SPI_CMD, SPI2AHB, JTAG • 2’b00 – The pins will be set based on strapping pin config • 2’b01 – Reserved for debugging ports • 2’b10 – Reserved for debugging ports • 2’b11 – Set the pins to JTAG Interface

11.2.1.2 GPIOXX Register

This set of registers are used to configure the GPIO pins to either input or output event signals.

Bits	Reg Name	Access	Description
[9:8]	mode_sel	RW	Configure this register to select event mode • 2’b00 – To select GPIO events (set through GPIO_OUT reg) to connect to GPIO pins • 2’b01 – To select SOC peripherals signals to input / output through GPIO pins • 2’b10 – To select SOC events (list0) to output through GPIO pins • 2’b11 – To select SOC events (list1) to output through GPIO pins
[5:0]	src_sel	RW	Configure the event signal index to map them to GPIO pins.

Please see the tables below for the list of all event signals.

Table 11-5: List of GPIO Events

Event idx	GPIO Events
0	GPIO_OUT[0]
1	GPIO_OUT[1]
2	GPIO_OUT[2]
3	GPIO_OUT[3]
4	GPIO_OUT[4]
5	GPIO_OUT[5]
6	GPIO_OUT[6]
7	GPIO_OUT[7]
8	GPIO_OUT[8]
9	GPIO_OUT[9]
10	GPIO_OUT[10]
11	GPIO_OUT[11]

Table 11-6: List of SOC Peripheral Events

Event idx	SOC Peripheral Events	Event idx	SOC peripheral
0	spim_cs0	15	uart0_rtsn
1	spim_clk	16	uart1_txd
2	spim_mosi	17	uart1_rxd
3	spim_miso	18	uart1_ctsn

4	spim_cs1	19	uart1_rtsn
5	spim_cs2	20	i2c_sck
6	spim_cs3	21	i2c_sda
7	spim_cs4	22	ext_irq_i0
8	spis_cs	23	ext_irq_i1
9	spis_clk	24	uwb_i0
10	spis_mosi	25	uwb_i1
11	spis_miso	26	uwb_i2
12	uart0_txd	27	uwb_i3
13	uart0_rxd	[31:28]	others: reserved.
14	uart0_ctsn		

Table 11-7: Event list 0 of SOC Events

Event idx	SOC Events	Event idx	SOC Events
0	tz_ctrl_irq	32	RX done irq
1	qspi_irq	33	Tx done irq
2	dma_irq	34	TX SFD irq
3	spacc_irq	35	General Purpose event 0
4	pka_irq	36	General Purpose event 1
5	trng_irq	37	General Purpose event 2
6	crc_irq	38	General Purpose event 3
7	gpio_irq	39	Timer 0 Timeout 0 event
8	spims_irq	40	Timer 0 Timeout 1 event
9	uart0_irq	41	Timer 0 Timeout 2 event
10	uart1_irq	42	Timer 0 Timeout 3 event
11	I2c_irq	43	Timer 0 Timeout common
12	Timer0_irq	44	Timer 1 Timeout 0 event
13	Timer1_irq	45	Timer 1 Timeout 1 event
14	Timer2_irq	46	Timer 1 Timeout 2 event
15	Timer3_irq	47	Timer 1 Timeout 3 event
16	Ext_irq_in_0	48	Timer 1 Timeout common
17	Ext_irq_in_1	49	Timer 2 Timeout 0 event
18	Ble_irq	50	Timer 2 Timeout 1 event
19	Event_irq	51	Timer 2 Timeout 2 event
20	DSR RX buffer overflow irq	52	Timer 2 Timeout 3 event
21	RX0 Done irq	53	Timer 2 Timeout common
22	RX0 PD irq	54	Timer 3 Timeout 0 event
23	RX0 SFD irq	55	Timer 3 Timeout 1 event
24	RX1 Done irq	56	Timer 3 Timeout 2 event
25	RX1 PD irq	57	Timer 3 Timeout 3 event
26	RX1 SFD irq	58	Timer 3 Timeout common
27	RX2 Done irq	59	gpio_trig_val [0]
28	RX2 PD irq	60	gpio_trig_val [1]
29	RX2 SFD irq	61	gpio_trig_val [2]
30	STS CIR End irq	62	gpio_trig_val [3]
31	PHY PHR Done irq	63	gpio_trig_val [4]

Table 11-8: Event list 1 of SOC Events

Event idx	SOC Events	Event idx	SOC Events
64	gpio_trig_val [5]	90	RX1 SFD Det
65	gpio_trig_val [6]	91	RX2 Start
66	gpio_trig_val [7]	92	RX2 Done
67	gpio_trig_val [8]	93	RX2 Pmbl Det
68	gpio_trig_val [9]	94	RX2 SFD Det
69	gpio_trig_val [10]	95	STS Seg0 Enable
70	gpio_trig_val [11]	96	STS Seg1 Enable
71	DMA Channel 0 ready	97	STS CIR End
72	DMA Channel 1 ready	98	PHY PHR Done
73	DMA Channel 2 ready	99	RX Done

74	DMA Channel 3 ready	100	PHY PHR Fail
75	tx_str_req	101	Rx Error event
76	event_debug	102	ADC debug event
77	DSR rx buffer overflow	103	ADC capture active
78	General Purpose event 0	104	AGC gain step [0]
79	General Purpose event 1	105	AGC gain step [1]
80	General Purpose event 2	106	AGC gain step [2]
81	General Purpose event 3	107	TX debug event
82	RX debug event	108	TX done
83	RX0 start	109	TX STS 2 Mark
84	RX0 Done	110	TX STS 1 Mark
85	RX0 Pmbl Det	111	TX SFD Mark
86	RX0 SFD Det	112	TX PLD EOM
87	RX1 start	113	TX STS EOM
88	RX1 Done	114	TX SYNC EOM
89	RX1 Pmbl Det	115	TX start

11.2.1.3 SPI_TRIG_START Register

This register is configured to start the SPI master-slave module based on any SOC event signal.

Bits	Reg Name	Access	Reset Value	Description
[6:0]	SPI_TRIG_START	RW	7'd127	To start SPI module based on any SOC event signal. Configure the Event index of the SOC event signal

11.2.1.4 UART0_TRIG Register

This register is configured to start the UART_0 TX/RX module based on any SOC event signal.

Bits	Reg Name	Access	Reset Value	Description
[6:0]	UART0_TRIG_RX_START	RW	7'd127	To start UART0 RX module based on any SOC event signal. Configure the Event index of the SOC event signal
[14:8]	UART0_TRIG_TX_START	RW	7'd127	To start UART0 TX module based on any SOC event signal. Configure the Event index of the SOC event signal

11.2.1.5 UART1_TRIG Register

This register is configured to start the UART_1 TX/RX module based on any SOC event signal.

Bits	Reg Name	Access	Reset Value	Description
[6:0]	UART1_TRIG_RX_START	RW	7'd127	To start UART1 RX module based on any SOC event signal. Configure the Event index of the SOC event signal
[14:8]	UART1_TRIG_TX_START	RW	7'd127	To start UART1 TX module based on any SOC event signal. Configure the Event index of the SOC event signal

11.2.1.6 TIME0_TRIG Register

This register is configured to start the SPI master-slave module based on any SOC event signal.

Bits	Reg Name	Access	Reset Value	Description
[6:0]	TIME0_TRIG_START	RW	7'd127	To start Timer0 module based on any SOC event signal. Configure the Event index of the SOC event signal
[14:8]	TIME0_TRIG_CAPTURE	RW	7'd127	To capture Timer0 timeout 0 event. Configure the Event index of the SOC event signal

11.2.1.7 TIME1_TRIG Register

This register is configured to start the SPI master-slave module based on any SOC event signal.

Bits	Reg Name	Access	Reset Value	Description
[6:0]	TIME1_TRIG_START	RW	7'd127	To start Timer1 module based on any SOC event signal. Configure the Event index of the SOC event signal
[14:8]	TIME1_TRIG_CAPTURE	RW	7'd127	To capture Timer1 timeout 0 event. Configure the Event index of the SOC event signal

11.2.1.8 TIME2_TRIG Register

This register is configured to start the SPI master-slave module based on any SOC event signal.

Bits	Reg Name	Access	Reset Value	Description
[6:0]	TIME2_TRIG_START	RW	7’d127	To start Timer2 module based on any SOC event signal. Configure the Event index of the SOC event signal
[14:8]	TIME2_TRIG_CAPTURE	RW	7’d127	To capture Timer2 timeout 0 event. Configure the Event index of the SOC event signal

11.2.1.9 TIME3_TRIG Register

This register is configured to start the SPI master-slave module based on any SOC event signal.

Bits	Reg Name	Access	Reset Value	Description
[6:0]	TIME3_TRIG_START	RW	7’d127	To start Timer3 module based on any SOC event signal. Configure the Event index of the SOC event signal
[14:8]	TIME3_TRIG_CAPTURE	RW	7’d127	To capture Timer3 timeout 0 event. Configure the Event index of the SOC event signal

11.2.1.10 I2C_TRIG_START Register

This register is configured to start the SPI master-slave module based on any SOC event signal.

Bits	Reg Name	Access	Reset Value	Description
[6:0]	I2C_TRIG_START	RW	7’d127	To start I2C module based on any SOC event signal. Configure the Event index of the SOC event signal

11.2.1.11 CRC_TRIG_START Register

This register is configured to start the SPI master-slave module based on any SOC event signal.

Bits	Reg Name	Access	Reset Value	Description
[6:0]	CRC_TRIG_START	RW	7’d127	To start CRC module based on any SOC event signal. Configure the Event index of the SOC event signal

11.2.1.12 EVENTS_IRQ_SEL Register

This register is configured to any SOC event signal to be selected as IRQ signal.

Bits	Reg Name	Access	Reset Value	Description
[6:0]	EVENTS_IRQ_SEL	RW	7’d127	Configure the Event index of the SOC event signal which is selected as IRQ

11.2.2 GPIO Registers

Table 11-9: List of GPIO registers

Offset	Reg	Range	Field	Access	Reset Value	Description
0x0	GPIO_MODE	[0]	enable	RW	1'b0	GPIO mode enable
0x4	GPIO	[11:0]	out	RW	12'h0	Output register to GPIO port
0x8	GPIO_OUT	[11:0]	en	RW	12'h0	GPIO output enable
0xc	GPIO_IN	[11:0]	data	RO		Read GPIO input port
0x10	TRIG	[11:0]	en	RW	12'h0	Trigger mode enable
0x14	TRIG_CFG	[23:0]	set	RW	12'h0	Trigger Set register for each pin
0x18	TRIG_VAL	[11:0]	reg	RO		Trigger status of the pin
0x1c	TRIG_CLEAR	[11]	pin11	WP	1'b0	Clear the trigger value
		[10]	pin10	WP	1'b0	Clear the trigger
		[9]	pin9	WP	1'b0	Clear the trigger
		[8]	pin8	WP	1'b0	Clear the trigger
		[7]	pin7	WP	1'b0	Clear the trigger
		[6]	pin6	WP	1'b0	Clear the trigger
		[5]	pin5	WP	1'b0	Clear the trigger
		[4]	pin4	WP	1'b0	Clear the trigger
		[3]	pin3	WP	1'b0	Clear the trigger

		[2]	pin2	WP	1'b0	Clear the trigger
		[1]	pin1	WP	1'b0	Clear the trigger
		[0]	pin0	WP	1'b0	Clear the trigger
0x20	TRIG_OUT	[0]	sel	RW	1'b0	trigger output selection
0x24	GPIO_PIN0	[4:0]	cfg	RW	5'h0	GPIO Pin 0 config (PE, PS, DS1, DS0, SL)
0x28	GPIO_PIN1	[4:0]	cfg	RW	5'h0	GPIO Pin 1
0x2c	GPIO_PIN2	[4:0]	cfg	RW	5'h0	GPIO Pin 2
0x30	GPIO_PIN3	[4:0]	cfg	RW	5'h0	GPIO Pin 3
0x34	GPIO_PIN4	[4:0]	cfg	RW	5'h0	GPIO Pin 4
0x38	GPIO_PIN5	[4:0]	cfg	RW	5'h0	GPIO Pin 5
0x3c	GPIO_PIN6	[4:0]	cfg	RW	5'h0	GPIO Pin 6
0x40	GPIO_PIN7	[4:0]	cfg	RW	5'h0	GPIO Pin 7
0x44	GPIO_PIN8	[4:0]	cfg	RW	5'h0	GPIO Pin 8
0x48	GPIO_PIN9	[4:0]	cfg	RW	5'h0	GPIO Pin 9
0x4c	GPIO_PIN10	[4:0]	cfg	RW	5'h0	GPIO Pin 10
0x50	GPIO_PIN11	[4:0]	cfg	RW	5'h0	GPIO Pin 11

Please refer to Table 25-1 Explanation of Access in register list for “Access” definition

11.2.2.1 GPIO_MODE_ENABLE Register

This register is set to enable the GPIO module.

Bits	Reg Name	Access	Reset Value	Description
[0]	GPIO_MODE_ENABLE	RW	1’b0	Configure the register to enable the GPIO module • 1’b1 – Enable GPIO Module • 1’b0 – Disable GPIO Module

11.2.2.2 GPIO_OUT Register

This register is set to output GPIO events. The size of register is 12 bits. Each bit represents one GPIO pin.

Bits	Reg Name	Access	Reset Value	Description
[11:0]	GPIO_OUT	RW	12’h0	Each bit corresponds to one GPIO Pin. • 1’b1 – To set gpio output event to 1 • 1’b0 – To set gpio output event to 0

11.2.2.3 GPIO_OUT_EN Register

This register is configured to enable the gpio_output events to map to GPIO pins. The size of the register is 12 bits. Each bit represents one GPIO pin.

Bits	Reg Name	Access	Reset Value	Description
[11:0]	GPIO_OUT_EN	RW	12’h0	Each bit corresponds to one GPIO pin. • 1’b1 – To enable GPIO output event • 1’b0 – To disable GPIO output event

11.2.2.4 GPIO_IN_DATA Register

This is a Read-Only (RO) register. It returns the value of input GPIO pins.

Bits	Reg Name	Access	Reset Value	Description
[11:0]	GPIO_IN_DATA	RO	-	To read the GPIO pin values. Each bit corresponds to one GPIO pin.

11.2.2.5 TRIG_EN Register

The register enables the GPIO Trigger mode for all the GPIO pins.

Bits	Reg Name	Access	Reset Value	Description
[11:0]	TRIG_EN	RW	12’h0	To enable the Trigger mode for GPIO pins. Each bit represents one GPIO • Set bit to 1 – Enable trigger mode • Set bit to 0 – Disable trigger mode

11.2.2.6 TRIG_CFG_SET Register

The register is configured to set the type of trigger each GPIO pin encounters.

Bits	Reg Name	Access	Reset Value	Description
[23:0]	TRIG_CFG_SET	RW	23'h0	Set the type of GPIO trigger. Every two bits represents each GPIO pin • 2'b00 – no trig • 2'b01 – Rising Edge • 2'b10 – Falling Edge • 2'b11 – Rising or Falling Edge

11.2.2.7 TRIG_VAL Register

This register is Read-only (RO). The register returns the Trigger status of GPIO pins.

Bits	Reg Name	Access	Reset Value	Description
[11:0]	TRIG_VAL	RO	-	Returns the trigger status of the GPIO pins. Each bit represents one GPIO pin.

11.2.2.8 TRIG_CLEAR Register

The register is configured to clear the trigger value of the GPIO pins. This is a write-pulse (WP) register. By configurinf this register a pulse signal is generated and it is used to clear the trigger value of the pin.

Bits	Reg Name	Access	Reset Value	Description
[11:0]	TRIG_CLEAR	WP	12'h0	To clear the trigger value of the GPIO pin. Each bit represents one GPIO pin • 1'b1 – Write 1 to clear the trigger value

11.2.2.9 TRIG_OUT Register

The register is configured to select the trigger output from the TRIG_VAL register or directly when the GPIO is triggered.

Bits	Reg Name	Access	Reset Value	Description
[0]	TRIG_OUT	RW	1'b0	To select the GPIO trigger output source. • 1'b1 – select the trigger output from TRIG_VAL register • 1'b0 – select the trigger output directly when GPIO is triggered

11.2.2.10 GPIO_PINx Register

The register sets the GPIO pad configurations such as Pull up/ Pull down, drive strength.

Bits	Reg Name	Access	Reset Value	Description
[0]	GPIO_PINx	RW	1'b0	To set the Pull up / Pull down enable for GPIO pad • 1'b1 – Pull up/Pull down Enable • 1'b0 – Pull up/ Pull down Disable
[1]	GPIO_PINx	RW	1'b0	To select the Pull up/ Pull down config for GPIO pad • 1'b0 – Pull down select • 1'b1 – Pull up select
[2]	GPIO_PINx	RW	1'b0	Not used reserved bit
[3]	GPIO_PINx	RW	1'b0	To set the drive strength of the GPIO PAD
[4]	GPIO_PINx	RW	1'b0	Not used reserved bit

12 DMA — Direct Memory Access

12.1 Functional description

12.1.1 Module description

DMA, direct memory access module is used in the system to facilitate the data connection of the communication peripherals (PKA mem, BLE mem, UWB mem and I2C) to the system memory. The current version of DMA module integrated has a total of 4 channels for memory transfers from defined source address to destination address and 8 flow control channels per channel. This allows for data transfers from e.g. a memory bank to a register or memory in a peripheral module. One example is DMA's usage with I2C module where once I2C module's txdata register is empty, a dma_ctrl signal will be asserted.

12.1.2 Configurable Features

DMA module consists of 4 configurable data channels with 8 flow control channels each, the dma module features:

- ➔ Programmable data channel source and destination
- ➔ Programmable data transfer size, source/destination data width
- ➔ Programmable source/destination address update
- ➔ Programmable linked list configuration
- ➔ Programmable wait states between transfers

12.1.3 Dma channels and flow control channel

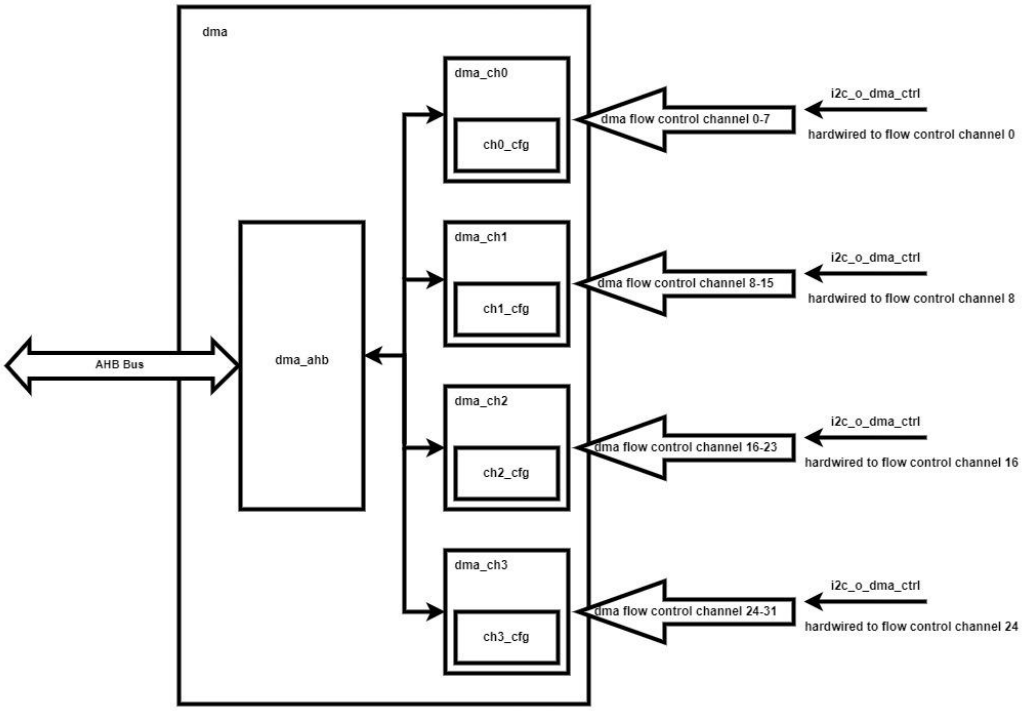


Figure 12- 1: High level block diagram of dma module

The dma module consists of 4 configurable channels. All dma channels has 8 flow control channels each and share one ahb submodule. Therefore, only one channel can perform a data transfer at a time. If two channels were to be active at the same time, the dma ahb will be alternatively used by both channels with priority given to the channel with priority set.

The flow control channels of each channel can be forced by writing into the DMA_REQ_REG.

Meanwhile the source and destination flow control select is set by configuring the ChFcSrcSel & ChFcDestSel with an offset of 1, meaning 0 disables flow control and 1 corresponds to flow control channel 0. For example, if flow control is to be enabled and the 1st channel is to be used for the dma_ch1's source, the channel 1 source flow control select should be set to 1. The flow control serves as a ready signal for the dma to either perform read or write towards source. While if the 1st flow control channel of channel 1 is to be asserted to 1 by writing to DMA->DMA_REQ_REG, the 1st flow control channel of channel 1 corresponds to the 9th bit of DMA->DMA_REQ_REG being DMA_REQ_REG[8].

If the flow control for either the source or destination is not set, the dma channel will automatically assume that the source or destination is always ready to be read or write.

N.B.: In UW1000 V210 chip, I2C dma_ctrl signal is hardwired to the 1st flow control channel of each dma channel. However, if I2C module is not in used the 1st flow control channel will still be free and can be controlled by forcing the flow control channel to be asserted by writing to dma_req register.

12.1.4 Basic transfer mode

In basic transfer mode, the data transfer starts whenever channel flow control is asserted. The transfer continues up until the defined number of bytes to be transferred in ChCtrl_TransferSize is reached and ch_ready signal will be asserted to signal completion of transfer. If flow control is enabled, after each read write sequence a flow control check will be done.

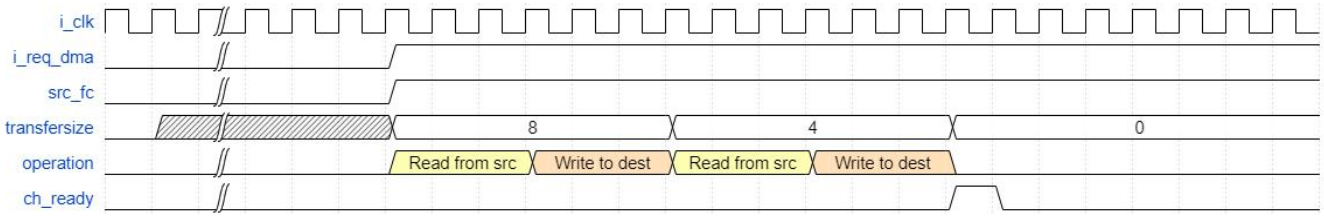


Figure 12-2: Timing diagram of basic mode

12.1.5 Continuous transfer mode

In continuous mode, the source and destination address are fixed and the data transfer will be done continuously once started, ignoring the transfer size configured for the channel. Data transfers will be continuously made as long as flow control of the channel is asserted.

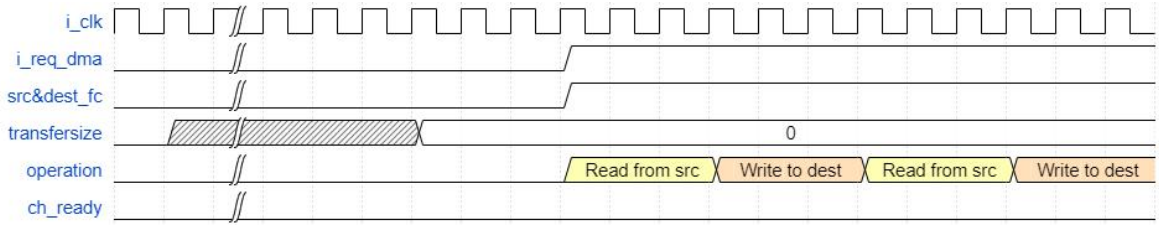


Figure 12-3: Timing diagram of continuous transfer mode

12.1.6 Linked List configuration packet (Lli)

- Data channel configuration

The dma module supports list packet transfers where if Lli is configured, once the current transfer is done, the dma channel will retrieve the next channel configuration from the Lli address and reconfigure itself to setup for the next data channel. If the ChLliAddr is set to 0, the dma channel will assume that there is no Lli configured and the transfer will end and ch_ready will be asserted. Else if a list configuration packet is set, the channel will retrieve the channel configuration from the set address and continue with the next data transfer up until there are no more list configuration set, then it'll assert the ch_ready signal to signal the completion of the transfers.

Table 12-1: Linked list format

LLI Format	
0x00	ChSrcAddr
0x04	ChDestAddr
0x08	ChLLIAddr
0x0C	ChCtrl
0x10	ChFc

All linked list configurations should follow this format where the 1st data the ChLliAddress points to contains the ChSrcAddr, followd by the ChdestAddr, ChLliAddr, ChCtrl and ChFc for the next configuration.

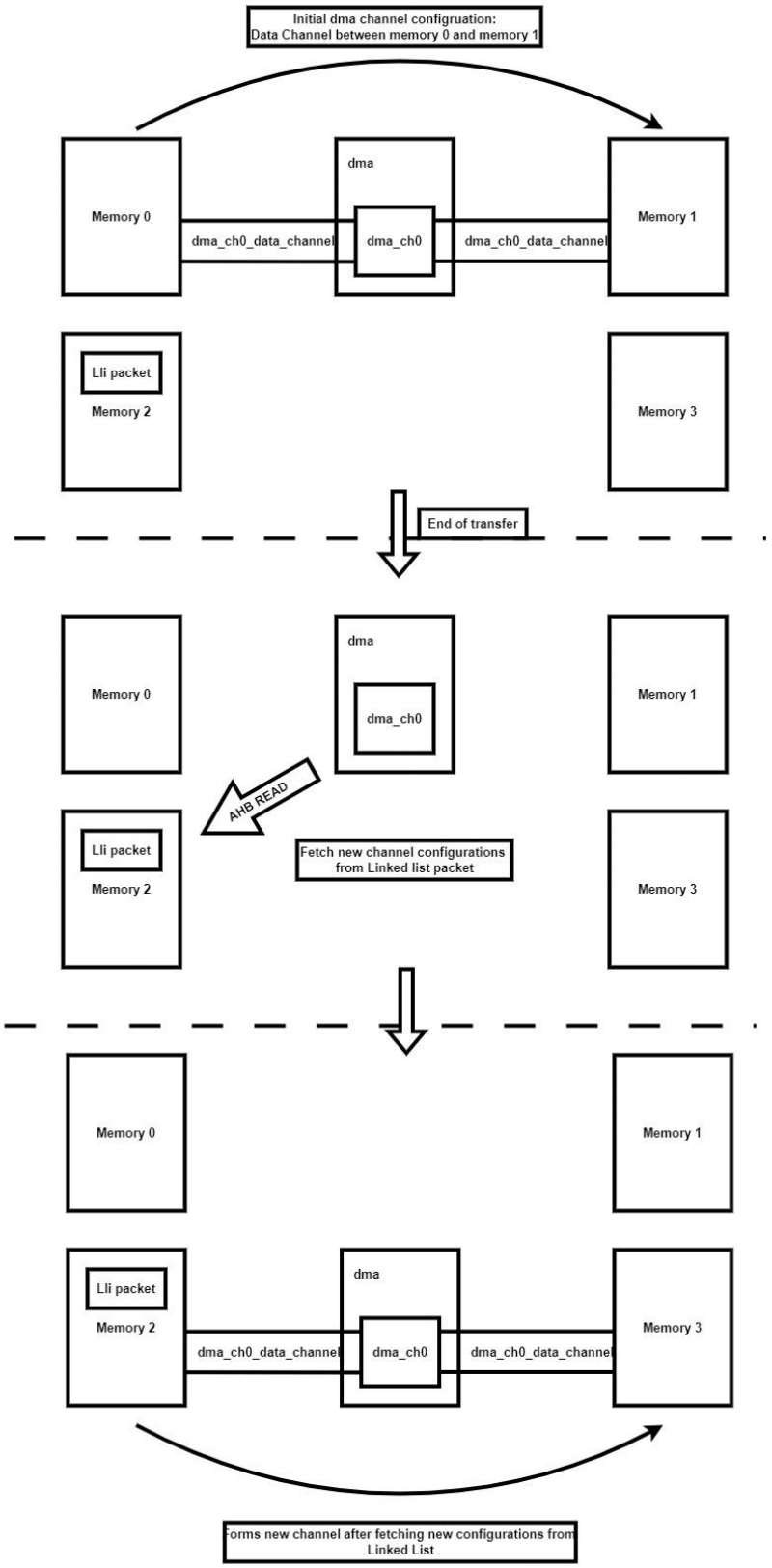


Figure 12-4: **DMA data transfer between memory 0 and memory 1**

Above shows an example of a linked list configuration done after finishing the 1st data transfer from memory 0 to memory 1. Once the 1st transfer is done, dma_ch0 will fetch the linked list packet from the configured ChLliAddr to get the 2nd configuration.

- Delay packet configuration

If the Lli configuration received is a delay Lli packet type, the dma_ch will decrement ChSrcAddr up until 0 before asserting ch_ready or fetching the next Lli packet. This can be used as a delay function after the 1st data transfer is done.

12.1.7 Usage examples

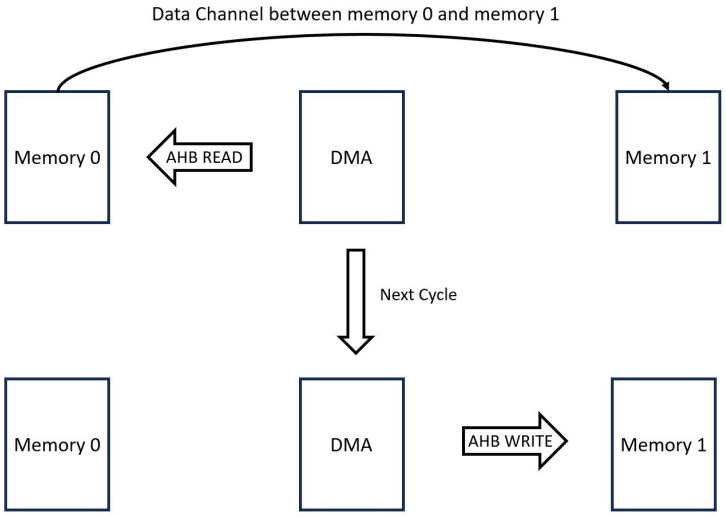


Figure 12-5: **DMA data transfer between memory 0 and memory 1**

An example of the usage of the dma module is to transfer a large amount of data from memory A to memory B. This decreases the workload of the CPU that is supposedly responsible to move the data. For example, a load of 4K bytes can be transferred by just having the CPU configure one of the dma channels to setup the data channel. Then ch_ready can be configured to assert the interrupt once the transfer is done to signal the completion of the transfer.

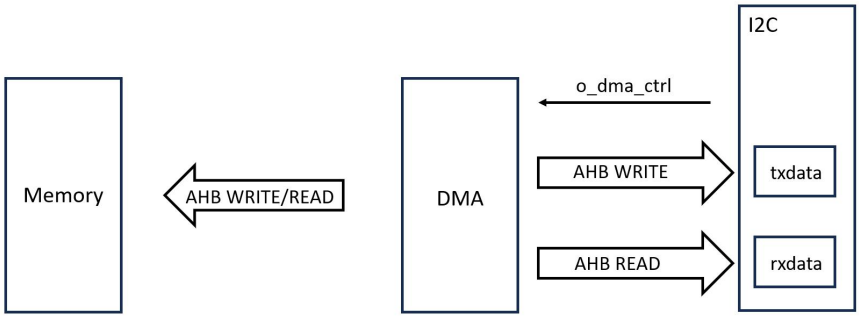


Figure 12-6: **DMA data transfer between i2c data registers and memory**

Another example is for the data transfer of i2c module that works in conjunction with the dma module. In the i2c write case, whenever txdata is empty, the i2c module will assert o_dma_ctrl which is hardwired to the flow control channel 0 of the dma module to signal that it is ready for the dma module to transfer the next txdata to be sent via i2c. On the other hand, for i2c read whenever rxdata is full o_dma_ctrl will be asserted and data will be transferred away from the i2c rxdata buffer.

12.2 Registers

The base address for the DMA registers is 32'h40021000.

Table 12-2: DMA module registers

Offset	Reg	Range	Field	Access	ResetValue	Note
0x000	DMAENABLE	[0]	DmaEnable	RW	1'b0	1: Enable DMA
0x004	CHIRQEN	[3:0]	ChReadyIRQEn	RW	4'b0	1: Enable channel ready interrupt
		[19:16]	ChErrIRQEn	RW	4'b0	1: Enable channel error interrupt
0x008	CHIRQSTAT	[3:0]	ChReadyIRQStat	RO	4'b0	Read to clear module IRQ
		[19:16]	ChErrIRQStat	RO	4'b0	
0x00C	CHFCWS	[1:0]	ChFcWS	RW	2'b0	00: No additional wait state 01: 1 clock cycle wait state 10: 2 clock cycle wait state 11: 3 clock cycle wait state

0x010	CH_IDLE	[3:0]	ch_idle	RO	4'b0	
0x014	DMA_REQ_REG	[7:0]	dma_req_reg[0]	RW	8'b0	1: Force flow control to 1
		[15:8]	dma_req_reg[1]	RW	8'b0	1: Force flow control to 2
		[23:16]	dma_req_reg[2]	RW	8'b0	1: Force flow control to 3
		[31:24]	dma_req_reg[3]	RW	8'b0	1: Force flow control to 4

Please refer to Table 25-1 Explanation of Access in register list for “Access” definition

Table 12-3: DMA_ch0 submodule registers

0x100	CH0_CHSRCADDR	[31:0]	ChSrcAddr	RW	32'b0	Channel source address
0x104	CH0_CHDESTADDR	[31:0]	ChDestAddr	RW	32'b0	Channel destination address
0x108	CH0_CHLLIADDR	[31:0]	ChLLIAddr	RW	32'b0	Channel Lli address
0x10C	CH0_CHCTRL	[17:0]	ChCtrl_TransferSize	RW	18'b0	Number of bytes to be transferred
		[19:18]	ChCtrl_Type	RW	2'b0	00: Default mode 01: Delay Lli packet by decrementing ChSrcAddr until 0 then begin next Lli read
		[21:20]	ChCtrl_Swidth	RW	2'b0	00: Byte access 01: Half word access 10: Word access
		[23:22]	ChCtrl_Dwidth	RW	2'b0	00: Byte access 01: Half word access 10: Word access
		[24]	ChCtrl_IfcSel	RW	1'b0	0: AHB access transaction 1: APB access transaction
		[25]	ChCtrl_Sinc	RW	1'b0	1: Update source address after each transaction
		[26]	ChCtrl_Dinc	RW	1'b0	1: Update destination address after each transaction
		[31]	ChCtrl_Continous	RW	1'b0	1: Enable Continuous transfers; only from/to fixed src/dest addresses and flow control (must set ChCtrl_Sinc and ChCtrl_Dinc to 0)
0x110	CH0_CHFC	[3:0]	ChFcSrcSel	RW	4'b0	Indicate index of flow control for source
		[11:8]	ChFcDestSel	RW	4'b0	Indicate index of flow control for destination
		[19:16]	ChFcLliSel	RW	4'b0	Indicate index of flow control for Lli
0x114	CH0_CHCFG	[0]	ChCfg_En	RW	1'b0	1: Enable channel
		[1]	ChCfg_Pause	RW	1'b0	1: Pause channel
		[2]	ChCfg_Prio	RW	1'b0	1: Set priority to channel
		[3]	ChCfg_HaltOnErr	RW	1'b0	1: Set to stop on error by clearing ChCfg
		[4]	ChCfg_IRQonNextLLI	RW	1'b0	1: Enable channel ready response when Lli read is done

Table 12-4: DMA_ch1 submodule registers

0x120	CH1_CHSRCADDR	[31:0]	ChSrcAddr	RW	32'b0	Channel source address
0x124	CH1_CHDESTADDR	[31:0]	ChDestAddr	RW	32'b0	Channel destination address
0x128	CH1_CHLLIADDR	[31:0]	ChLLIAddr	RW	32'b0	Channel Lli address
0x12	CH1_CHCTRL	[17:0]	ChCtrl_TransferSize	R	18'b	Number of bytes to be transferred

C				W	0	
		[19:18]	ChCtrl_Type	RW	2'b0	00: Default mode 01: Delay Lli packet by decrementing ChSrcAddr until 0 then begin next Lli read
		[21:20]	ChCtrl_Swidth	RW	2'b0	00: Byte access 01: Half word access 10: Word access
		[23:22]	ChCtrl_Dwidth	RW	2'b0	00: Byte access 01: Half word access 10: Word access
		[24]	ChCtrl_IfcSel	RW	1'b0	0: AHB access transaction 1: APB access transaction
		[25]	ChCtrl_Sinc	RW	1'b0	1: Update source address after each transaction
		[26]	ChCtrl_Dinc	RW	1'b0	1:Update destination address after each transaction
		[31]	ChCtrl_Continous	RW	1'b0	1: Enable Continuous transfers; only from/to fixed src/dest addresses and flow control (must set ChCtrl_Sinc and ChCtrl_Dinc to 0)
0x130	CH1_CHFC	[3:0]	ChFcSrcSel	RW	4'b0	Indicate index of flow control for source
		[11:8]	ChFcDestSel	RW	4'b0	Indicate index of flow control for destination
		[19:16]	ChFcLliSel	RW	4'b0	Indicate index of flow control for Lli
0x134	CH1_CHCFG	[0]	ChCfg_En	RW	1'b0	1: Enable channel
		[1]	ChCfg_Pause	RW	1'b0	1: Pause channel
		[2]	ChCfg_Prio	RW	1'b0	1: Set priority to channel
		[3]	ChCfg_HaltOnErr	RW	1'b0	1: Set to stop on error by clearing ChCfg
		[4]	ChCfg_IRQonNextLLI	RW	1'b0	1: Enable channel ready response when Lli read is done

Table 12- 5: DMA_ch2 submodule registers

0x140	CH2_CHSRCADDR	[31:0]	ChSrcAddr	RW	32'b0	Channel source address
0x144	CH2_CHDESTADDR	[31:0]	ChDestAddr	RW	32'b0	Channel destination address
0x148	CH2_CHLLIADDR	[31:0]	ChLLIAddr	RW	32'b0	Channel Lli address
0x14C	CH2_CHCTRL	[17:0]	ChCtrl_TransferSize	RW	18'b0	Number of bytes to be transferred
		[19:18]	ChCtrl_Type	RW	2'b0	00: Default mode 01: Delay Lli packet by decrementing ChSrcAddr until 0 then begin next Lli read
		[21:20]	ChCtrl_Swidth	RW	2'b0	00: Byte access 01: Half word access 10: Word access
		[23:22]	ChCtrl_Dwidth	RW	2'b0	00: Byte access 01: Half word access 10: Word access
		[24]	ChCtrl_IfcSel	RW	1'b0	0: AHB access transaction 1: APB access transaction
		[25]	ChCtrl_Sinc	RW	1'b0	1: Update source address after each transaction
		[26]	ChCtrl_Dinc	RW	1'b0	1:Update destination address after each transaction
		[31]	ChCtrl_Continous	RW	1'b0	1: Enable Continuous transfers; only from/to fixed src/dest addresses

						and flow control (must set ChCtrl_Sinc and ChCtrl_Dinc to 0)
0x150	CH2_CHFC	[3:0]	ChFcSrcSel	RW	4'b0	Indicate index of flow control for source
		[11:8]	ChFcDestSel	RW	4'b0	Indicate index of flow control for destination
		[19:16]	ChFcLliSel	RW	4'b0	Indicate index of flow control for Lli
0x154	CH2_CHCFG	[0]	ChCfg_En	RW	1'b0	1: Enable channel
		[1]	ChCfg_Pause	RW	1'b0	1: Pause channel
		[2]	ChCfg_Prio	RW	1'b0	1: Set priority to channel
		[3]	ChCfg_HaltOnErr	RW	1'b0	1: Set to stop on error by clearing ChCfg
		[4]	ChCfg_IRQonNextLLI	RW	1'b0	1: Enable channel ready response when Lli read is done

Table 12- 6: DMA_ch3 submodule registers

0x160	CH3_CHSRCADDR	[31:0]	ChSrcAddr	RW	32'b0	Channel source address
0x164	CH3_CHDESTADDR	[31:0]	ChDestAddr	RW	32'b0	Channel destination address
0x168	CH3_CHLLIADDR	[31:0]	ChLLIAddr	RW	32'b0	Channel Lli address
0x16C	CH3_CHCTRL	[17:0]	ChCtrl_TransferSize	RW	18'b0	Number of bytes to be transferred
		[19:18]	ChCtrl_Type	RW	2'b0	00: Default mode 01: Delay Lli packet by decrementing ChSrcAddr until 0 then begin next Lli read
		[21:20]	ChCtrl_Swidth	RW	2'b0	00: Byte access 01: Half word access 10: Word access
		[23:22]	ChCtrl_Dwidth	RW	2'b0	00: Byte access 01: Half word access 10: Word access
		[24]	ChCtrl_IfcSel	RW	1'b0	0: AHB access transaction 1: APB access transaction
		[25]	ChCtrl_Sinc	RW	1'b0	1: Update source address after each transaction
		[26]	ChCtrl_Dinc	RW	1'b0	1:Update destination address after each transaction
		[31]	ChCtrl_Continous	RW	1'b0	1: Enable Continuous transfers; only from/to fixed src/dest addresses and flow control (must set ChCtrl_Sinc and ChCtrl_Dinc to 0)
0x170	CH3_CHFC	[3:0]	ChFcSrcSel	RW	4'b0	Indicate index of flow control for source
		[11:8]	ChFcDestSel	RW	4'b0	Indicate index of flow control for destination
		[19:16]	ChFcLliSel	RW	4'b0	Indicate index of flow control for Lli
0x174	CH3_CHCFG	[0]	ChCfg_En	RW	1'b0	1: Enable channel
		[1]	ChCfg_Pause	RW	1'b0	1: Pause channel
		[2]	ChCfg_Prio	RW	1'b0	1: Set priority to channel
		[3]	ChCfg_HaltOnErr	RW	1'b0	1: Set to stop on error by clearing ChCfg
		[4]	ChCfg_IRQonNextLLI	RW	1'b0	1: Enable channel ready response when Lli read is done

12.2.1.1 DMAENABLE

This register enables dma module, this should be done as the 1st step in the application flow for dma to enable the module.

12.2.1.2 CHIRQEN & CHIRQSTAT

ChIRQEn enables the interrupt of the dma module to be triggered by certain events defined. (ch_ready & ch_error)

Reading from ChIRQStat clears the interrupt of the dma module.

12.2.1.3 CHFCWS

This register adds additional wait states between each data transfers up to 3 clock cycles. This acts as a delay in between each AHB data transfer initiated by the DMA module.

12.2.1.4 DMA_REQ_REG

This register forces the flow control channel into the value written into the register. This can be used as flow control for memory modules where it's always ready to receive or send data.

12.2.2 DMA_ch0 register

All dma_ch submodules has the same configuration registers but with different addresses. For this part, channel 0 will be used as an example for the description of the registers.

12.2.2.1 CH0_CHSRCADDR & CH0_CHDESTADDR & CH0_CHLLIADDR

This register denotes the source, destination and Lli (linked list) address of the data channel.

12.2.2.2 CH0_CHCTRTL

This register configures the current data transfer.

- CHCTRl_TRANSFERSIZE
  This bit space denotes the transfer size of the current data channel.
- CHCTRL_TYPE
  This bit space is used to denote delay used in between Lli packets, if a delay type is set the channel will use ChSrcAddr as a counter and decrement ChSrcAddr every clock cycle up until 0 before fetching the next Lli packet.
- CHCTRL_SWIDTH
  This bit space denotes the size of read data from the source address. Usually, it should be set as the same as the destination address.
- CHCTRL_DWIDTH
  This bit space denotes the size of write data towards the destination address.
- CHCTRL_SINC
  This bit space is used to toggle whether the source address should be updated according the source data width defined after each transaction.
- CHCTRL_DINC
  This bit space is used to toggle whether the source address should be updated according the source data width defined after each transaction.
- CHCTRL_COTINUOUS
  This bit space is used to enable or disable continuous transfer mode.

12.2.2.3 CH0_CHFC

This register enables as well as configures the flow control channel used for source, destination and Lli of the current data channel. Flow control will be enabled as long as ChFcSrcSel / ChFcDestSel / ChFcLliSel is not 0. It is to be noted that 1 corresponds to the 1st flow control channel of the dma channel. Can refer to previous section as a guideline on how to configure the flow control.

12.2.2.4 CH0_CHCFG

This register initiates the start of the transfer as well as set the priority of the channel

- CHCFG_EN

This bit space enables the dma channel. This can be used as a start for the data transfer. Once the data channel is enabled, the dma channel will initiate data transfers depending on the state of the flow control channels.

- CHCFG_PAUSE
  This bit space puts the dma channel on pause.
- CHCFG_PRIO
  This bit space sets this dma channel as a higher priority compared to other channels. This can be useful when multiple dma channels are working simultaneously since only one AHB submodule is shared between all 4 channels.
- CHCFG_HALTONERR
  This configures the dma channel to stop whenever it encounters an error.
- CHCFG_IRQONNEXTLLI
  This configures the dma channel to assert the interrupt signal upon receiving a new Lli configuration.

13 SAA— Security Algorithm Accelerator

The Security Algorithm Accelerator (SAA) cores perform parallel hashing and encryption operations to allow acceleration support of a variety of security algorithms.

The SAA optimizes cryptographic offload for bulk processing of cipher and hash algorithms, as well as combined mode algorithms (for example, GCM, CCM). All ciphers supported by the SAA are symmetric-key algorithms.

The *Figure 13-1* shows the flow across the various top-level modules for a configuration with the dual-clock domain enabled. In our architecture both the clock domains are in hclk.

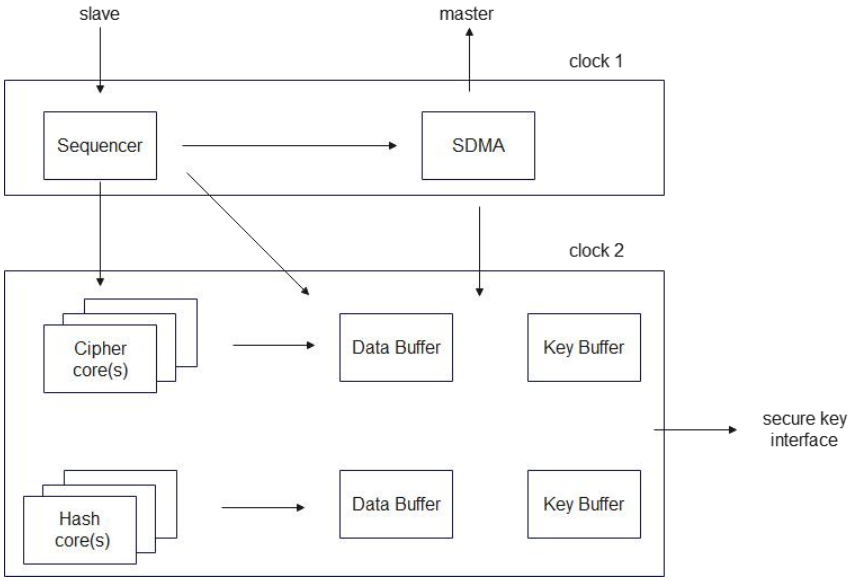


Figure 13-1: System Level Block Diagram of SAA

Context (also called key context) stands for cryptographic information. Each cryptographic algorithm requires unique information for processing. Contexts are stored in context memory (also called context buffer or key buffer). There are two context memories: one each for cipher and hash operations. The SAA module can support the message length up to 16KB with 16 contexts.

Algorithms and modes supported by SAA are individually configurable. Below is the list of cipher and hash algorithms and modes.

- **Cipher Algorithms**
  - AES (Advanced Encryption Standard) – Supports 128 and 256-bit key size
  - DES (Data Encryption Standard)

- **Cipher modes**
  - ECB (Electronic CodeBook ciphering mode)
  - CBC (Cipher Block Chaining mode)
  - CFB (Cipher FeedBack ciphering mode)
  - OFB (Output FeedBack ciphering mode)
  - CTR (CounTeR ciphering mode)
  - XTS (XEX encryption mode with Tweak and cipher test Stealing)
- **Hash Algorithms**
  - MD-5 (Message Digest 5)
  - SHA-1 (Secure Hash Algorithm)
  - SHA-2 : SHA-224, SHA-256
  - AES Based MACs : XCBC (eXtended CBC-MAC mode), CMAC (Cipher based MAC mode)
  - CRC-32 IEEE 802.3 (Cyclic Redundancy Check)
- **Hash modes**
  - HMAC (Hashed Message Authentication Code)
- **Combined Modes (AEAD mode)**
  - AES-CCM (Counter with Cipher Block Chaining-Message Authentication Code)
  - AES-GCM (Galois/Counter and GAMC modes)

Figure 13-2 shows the packet format.

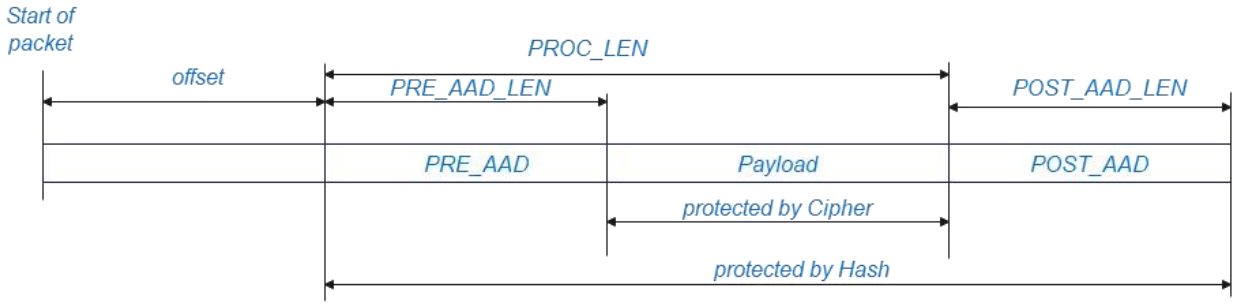


Figure 13-2: Generic Packet Format

14 PKG — Public Key Generator

The Public Key Generator (PKG) is a co-processor dedicated to enhance the performance and efficiency of public-key cryptographic operations such as RSA operations and Elliptic Curve Cryptography (ECC). These cryptographic operations require complex mathematical operations on very large numbers (from 160 to 4096 bits). The majority of embedded CPUs are limited to operations on 32 or 64 bits so PKGs are implemented to handle these operations more quickly and deliver higher performance levels that are not achievable in software-only solutions.

The various PKG functions implemented are mainly divided into three categories

- Base Modular Arithmetic Library Functions
- RSA Functions
- Base ECC Functions
- ECC Shamir Functions
- ECC – 521 Functions
- ECC – 521 Shamir Functions

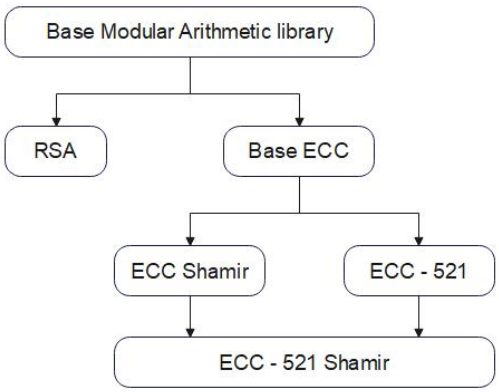


Figure 14- 1: PKG Functions Dependency Graph

The engine of PKG always uses data internally in little-endian format the least significant byte is in the right most byte lane of the data parameter and the least significant word is at the lower relative address. In cases where the host has data in big-endian format (least significant byte is in the left most byte lane), the host should set the ENDIAN_SWAP bit in the CONFIG register prior to transferring data into or out of the PKG.

15 TRNG — True random number generator

15.1 Overview

The Random number generator (RNG) generates true non-deterministic random numbers based on internal thermal noise that are suitable for cryptographic purposes.
The TRNG generates random numbers that are intended to be statistically equivalent to a uniformly distributed random data stream. The circuit includes a NIST SP800-90B compliant noise source, a NIST SP800-90B vetted conditioning component, and a NIST SP800-90A approved Deterministic Random Bit Generator (DRBG). The noise source sends Independent and Identically Distributed (IID) noise stream to the conditioning component to produce full-entropy seed which is then for generating random numbers.

The TRNG can generate random seeds from the internal ring-oscillator based noise source or can be manually seeded through a host-provided nonce. Host-provided nonce are fed into the conditioning component to increase the entropy rate using a NIST SP800-90B vetted conditioning function.

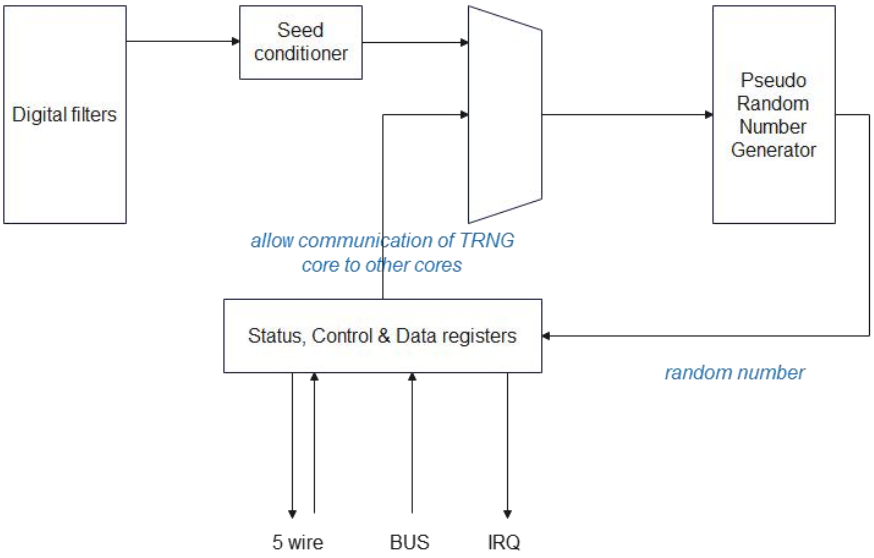


Figure 15-1: Simplified block diagram of TRNG

15.2 Functional description

The following *Figure 15-2* shows the legal command state transition for the TRNG NIST. This sequence must be followed by the application software. A Zeroize command can interrupt any other command. This means that move to the zeroize state from any other state is possible. For the simplicity's sake, arcs from other states to Zeroize are not shown. A KAT command can only be performed after a Zeroize operation (command or I_zeroize pin) since it changes the internal state information and it also must be followed by a Zeroize operation.

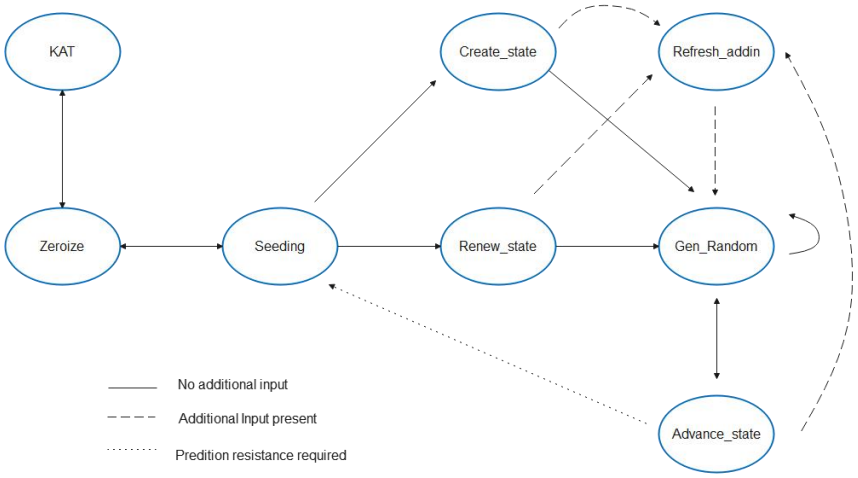


Figure 15-2: Legal Command State Transition

The following are the supported features:

- Internal random (re)seed operation
  i. 128 or 256-bit random number generation (build-time configuration option)
- Two separate reseed reminder schedules provide autonomous background reseeding (build-time configuration option)
- Shift register compatible output stream for auxiliary uses such as, DPA, TA, IPsec, and so on (for TRNG product only)
- Status/control I/O for allowing external monitoring of internal actions and states (for TRNG product only)
- Glue-less interface to ESM family entropy port (TRNG for ESM product only)

16 EADC — External ADC

16.1 Function description

An EADC refers to an analog-to-digital converter, which is used to convert external low speed analog signals and on-chip temperature into digital format. The EADC controller receives voltage, temperature or low speed signals from the analog circuit which is stored in respective FIFOs. EADC controller is controlled by using an APB register in SCR. The content in the FIFOs is read using the APB bus. In addition to storing data, the controller also generates clock based on the APB register configurations. The block diagram of the EADC connection is shown in *Figure 16-1*.

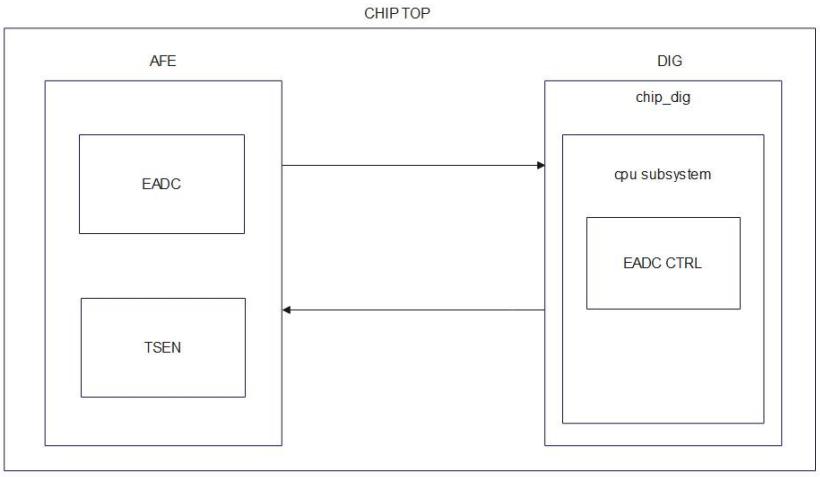


Figure 16-1: Block diagram of control flow for EADC

Figure 16-2 shows the clock divider logic circuit. Here the clock select signal determines the divide rate of the input 16Mhz clock signal is the manual clock mode is not selected. If the manual mode is selected then we use the register to generate the clock by writing 0 or 1 to the register.

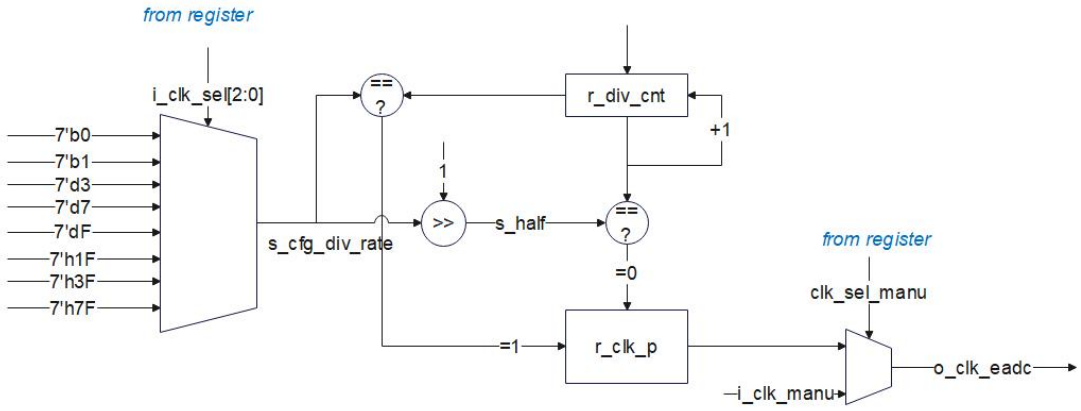


Figure 16-2: Clock Divider logic of EADC

16.2 Register

The registers for the EADC controller are provided in the SCR register under offsets 0x2C and 0x30. Registers in offset 0x2c are used to configure the analog EADC circuit, while the offset 0x30 is used to read the FIFO content.

Table 16-1: Register list of EADC

Offset	Reg	Range	Field	Access Type	Reset Value	Description

0x2C	eadc	[29]	vin_sel_dft	RW	1'b0	Analog control signal
		[28]	manual_clk	RW	1'b0	Write 1 to generate the rising edge of manual clock. Write 0 to generate the falling edge of manual clock
		[27:25]	clk_sel	RW	3'b0	Clock selection =0: manual clock =N: 16MHz divided by 2^N.
		[24:18]	eadc_vin_mid	RW	7'd65	Analog control signal
		[17:13]	eadc_vin_mid_cal	RW	5'd16	Analog control signal
		[12]	eadc_output_sign	RW	1'b0	Analog control signal
		[11:9]	eadc_gain	RW	3'b0	Analog control signal
		[8]	tsen_output_sign	RW	1'b0	Analog control signal
		[7:3]	tsen_eadc_vref	RW	5'd16	Analog control signal
		[2]	tsen_eadc_cal_mode	RW	1'b0	Analog control signal
		[1]	tsen_eadc_en	RW	1'b0	=1: enable temperature sensor
		[0]	rst_n	RW	1'b0	=1: release reset of eadc module
0x30	eadc_read	[31]	ts_fifo_full	RO		=1: TS FIFO is full
		[30:27]	ts_fifo_ndata	RO		Number of data in TS FIFO
		[26]	ts_fifo_empty	RO		=1: TS FIFO is empty
		[25:16]	ts_out	RO		TS data
		[15]	eadc_fifo_full	RO		=1: eadc FIFO is full
		[14:11]	eadc_fifo_ndata	RO		Number of data in eadc FIFO
		[10]	eadc_fifo_empty	RO		=1: eadc FIFO is empty
		[9:0]	eadc_out	RO		eadc data

Please refer to Table 25-1 Explanation of Access in register list for “Access” definition

17 SPI — Serial peripheral interface master/slave with AHBM

There is a general-purpose SPI module integrated in the chip for the communication with external SPI devices. It can be configured as a master or slave device. Here are the primary features of the SPI module:

- SPI mode 0 and mode 2 are supported for flexible communication.
- There are 5 chip select signals in master mode for multiple external slave device communication.
- The AHBM facilitates direct data transfers to and from RAM, supporting both SPI Slave and SPI Master functionalities.
- Each SPI signal can be individually selected, enabling customization of the IO pin for enhanced flexibility and control.

17.1 Functional description

17.1.1 SPI_MS Protocol

The block diagram of the SPI within the v210 chip is shown in Figure 1 below. The SPI_MS module supports traditional 4-line SPI operations and can be configured as either a master or slave to communicate with other SPI devices.

The SPI_MS module communicates with the microcontroller through both the APB and AHB interfaces. The APB bus is used to configure the module and read he status from registers, while the AHB bus is typically used by the DMA module to access memory for data transmission. An interrupt signal is generated by the SPI_MS module to notify the CPU to handle specific tasks.

Additionally, the IO_MUX is used to configure the GPIOs that connect to the SPI_MS module.

17.1.2 Configurable features

The SPI module support master or slave operation with the feature supported as below:

- ➔ Programmable working mode (master /slave)
- ➔ Programmable data length (up to 4095 bytes)
- ➔ Programmable clock polarities (spi MODE0 / MODE2)
- ➔ Programmable bit order / byte order
- ➔ Programmable clock rate for master mode (32MHz, 16 MHz, 8 MHz, 4MHz, 2 MHz, 1MHz, 500KHz, 250HKz)
- ➔ Programmable interrupts events
- ➔ Programmable FIFO mode or SDMA mode

17.1.3 I/Os setup

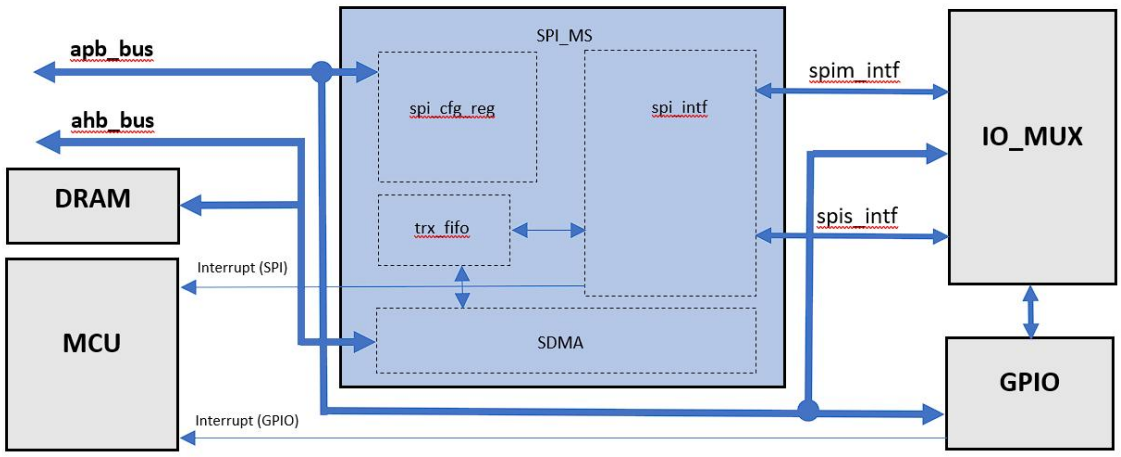


Figure 17-1: High-Level block diagram of SPI_MS GPIO pin configuration

The SPI_MS contains four main functional modules. The **spi_cfg_reg** module manages access to configuration and status registers. The **spi_intf** module converts data into the SPI format. The **trx_fifo** module acts as a data buffer for both transmitting and receiving. Finally, the **sdma** module facilitates automatic data transfer between the FIFO and memory via the AHB interface.

The SPI_MS module has four IOs which is need to connect to external GPIOs. It works either master or slave mode. The direction of each IO will be invert for master and slave.

Table 17-1: SPI GPIO signals

Name	Direction		Description
	Master	Slave	
SPICSn	Out	IN	The assertion low by SPI host indicates the beginning of a transaction. The de-assertion high to end of current transaction.
SPICLK	Out	IN	The clock from SPI host.
SPIMOSI	Out	IN	The data line from SPI host.
SPIMISO	IN	OUT	The data line to SPI host.

The user can also configure the GPIOs to be internally pulled-up to prevent High Z states if the connected line is not pulled-up.

N.B.: The output pin will only work as a driver during the SPI operation, once the operation transfer is done, the GPIO will go return to its High Z tristate state and will not drive the GPIO. If the user would want all output signals to continue driving the GPIO output, the GPIO can be set as internally pull-up.

The same goes for all input signal, if the driver is a tristate driver, it is recommended to setup the GPIO as internally pulled-up or given physical on-board pulled-up resistor.

17.1.4 SPI transaction Formatting

SPI interface supports **clock polarities** as defined in the Motorola SPI protocol as shown in the *Figure 20-2* below.

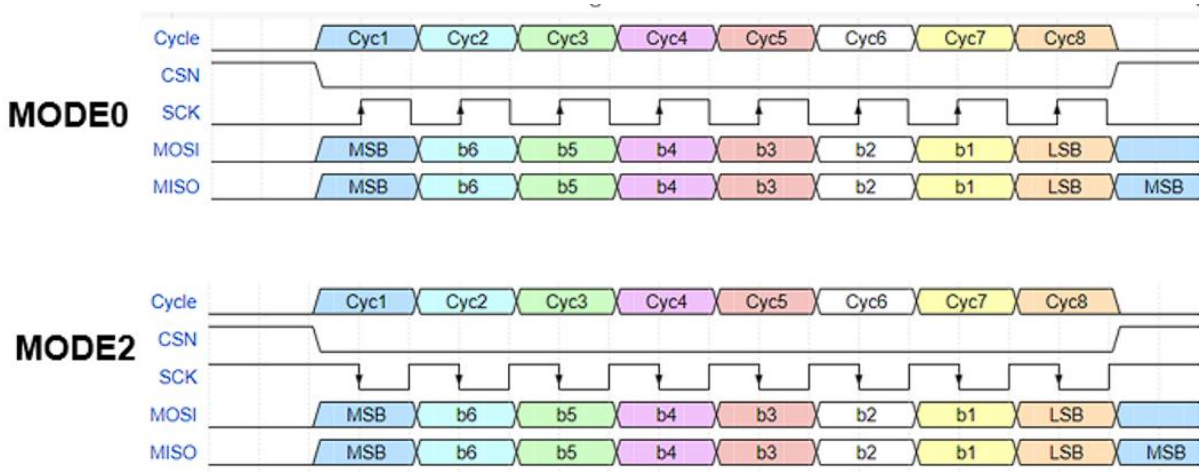


Figure 17-2: Mode 0 and Mode 2 of CPOL and CPHA for SPI clock

The module supports both bit-order and byte-order reversal. By default, it operates in traditional MSB-first mode, as shown in the figure above. However, by modifying the configuration register, LSB-first mode can also be enabled.

The **trx_fifo** module is an 8-byte buffer used to store data for both the transmitter and receiver. When the SPI command length exceeds 8 bytes in FIFO mode, users must manually fill or read the buffer. Alternatively, if the data is stored in memory and the data length is known, the **SDMA** module can handle the FIFO operations automatically. In SDMA mode, the module supports SPI transmissions of up to 4095 bytes.

17.1.5 SPI Timing

The SPI master supports eight different clock rates, which are derived from a 64 MHz clock source. The maximum operating speed for SCK is 32 MHz, and it can be halved in each step, down to a minimum speed of 250 kHz. The SPI slave module can support all clock rates provided by SPI host.

17.1.6 FIFO & SDMA Mode

The SPI_MS module can operate in two different working modes: FIFO mode and DMA mode. The FIFO mode is suitable of low speed SPI or data is not ready when start SPI operation. While the DMA mode is more suitable for high data rate operation.

- FIFO mode

There is an 8-byte FIFO buffer implemented for both TX and RX. FIFO mode can handle SPI transmissions of less than 8 bytes without CPU involvement. However, for transmissions exceeding 8 bytes, the CPU must manage FIFO read/write operations when the corresponding interrupt is triggered. This can introduce delays and potential errors if the CPU cannot complete the task in time. In FIFO mode, data can be prepared for transmission while the SPI is in operation.

- SDMA mode

In DMA mode, a simple DMA module automatically handles TX refill and RX readout without CPU intervention, making it more suitable for high-speed and large data operations. However, in this mode, the data RAM must be prepared before starting the SPI operation.

For both of the modes, the number of bytes sent or received can be checked from the spi_trx registers by reading the rxb_nbyte or txb_nbyte.

17.2 Registers

The APB interface is used to access the configuration register banks. *Table 20-2* provides the addresses and functional descriptions of these registers.

The base address of the SPI_MS module on the APB is located at 0x40023000.

Table 17-2: List of configuration registers in SPI_MS

Offset	Reg	Range	Field	Access	Reset Value	Description
0x0	SPI_EN	[0]	spi_enable	RW	1'b0	enable/disable SPI
0x4	SPI_START	[0]	spi_start	WP	1'b1	start spi transmission; =1: write 1 to start spi

		[1]	reserved			
0x8	SPI_INT_EN	[0]	spi_frame_end	RW	1'b0	enable/disable interrupt signal =1: enable interrupt =0: disable interrupt
		[1]	spi_tx_end	RW	1'b0	
		[2]	spi_rx_end	RW	1'b0	
		[3]	Txfifo_empty	RW	1'b0	
		[4]	Txfifo_full	RW	1'b0	
		[5]	Txbuf_empty	RW	1'b0	
		[6]	Rxfifo_empty	RW	1'b0	
		[7]	Rxfifo_full	RW	1'b0	
		[8]	Rxbuf_full	RW	1'b0	
		[9]	Txfifo_over_read_err	RW	1'b0	
		[10]	Rxfifo_overflow_err	RW	1'b0	
		[11]	Txbuf_read_err	RW	1'b0	
		[12]	Rxbuf_wr_err	RW	1'b0	
0xc	SPI_INT_CLR	[0]	clear_interrupt	WP	1'b1	clear interrupt =1: clear interrupt =0: no action
0x10	SPI_CFG	[1:0]	MODE	RW	2'd0	SPI mode, define SPI clock pol and phase (CPOL, CPHA) Only MODE0 /MODE2 is supported
		[2]	TYPE	RW	1'b0	SPI type: 1: SPI Master 0: SPI Slave
		[3]	BITORDER	RW	1'b1	bit order in a byte: 0: lsb first; 1: msb first
		[4]	BYTEORDER	RW	1'b1	byte order in a word [byte3 byte2 byte1 byte0] 0: Byte 0 first. 1: Byte 3 first.
		[7:5]	CLK_DIV_SEL	RW	3'h0	Spi clock diver mode select 1: 32MHz 2: 16MHz 3: 8MHz 4: 4MHz 5: 2MHz 6: 1MHz 7: 500KHz 0: 250KHz
		[8]	CSN0	RW	1'b1	Spi_cs0 is used
		[9]	CSN1	RW	1'b0	Spi_cs1 is used
		[10]	CSN2	RW	1'b0	Spi_cs2 is used
		[11]	CSN3	RW	1'b0	Spi_cs3 is used
		[12]	CSN4	RW	1'b0	Spi_cs4 is used
0x14	SPI_BUF_EN	[0]	EN	RW	1'b0	enable direct memory buffer access =1: enable direct mem buffer access (SDMA) =0: disable direct mem buffer access, data stored in FIFO
		[2:1]	Buf_en_min_bytes	RW	2'b00	Define the number of byte shall be received before transfer to rx buffer. n: n+1 byte
0x18	SPI_TXFIFO	[31:0]	Tx fifo write data	WE	--	TX data to be transferred write one word (32 bits) to TXD FIFO
0x1C	SPI_RXFIFO	[31:0]	RX fifo read data	RE	--	RX data received. Read one word (32 bits) from RXD FIFO
0x20	SPI_TRXFIFO	[1:0]	Trxfifo_read_bytes	RW	2'h3	Number of bytes read from fifo 0: 1 byte 1: 2 bytes 2: 3 bytes 3: 4 bytes
		[3:2]	Trxfifo_write_bytes	RW	2'h3	Number of bytes write to fifo 0: 1 byte 1: 2 bytes 2: 3 bytes 3: 4 bytes

0x28	SPI_TXBUF	[31:0]	SMDA TX memory start address	RW	32'h0	pointer to memory buffer for sending
0x2C	SPI_RXBUF	[31:0]	SMDA RX memory start address	RW	32'h0	pointer to memory buffer for receiving
0x30	SPI_BUF_SIZE	[11:0]	Tx_buf_max_size	RW	12'h0	Set max mem size for TX buffer
		[27:16]	Rx_buf_max_size	RW	12'h0	Set max mem size for RX buffer
0x34	SPI_TRX_ST	[11:0]	Txb_nbytes	RO		Number of bytes has send out
		[27:16]	Rxb_nbytes	RO		Number of bytes received
		[31:28]	Rxfifo_nBytes	RO		Number of bytes exist in rxfifo
0x38	SPI_EVENT	[0]	SPI_ON	RO		Spi is on
		[1]	TX_END	RO		tx is end
		[2]	RX_END	RO		rx is end
		[3]	TXFIFO_EMPTY	RO		txfifo is empty
		[4]	TXFIFO_FULL	RO		txfifo is full
		[5]	TXB_EMPTY	RO		Tx buffer is empty
		[6]	RXFIFO_EMPTY	RO		rxfifo is empty
		[7]	RXFIFO_FULL	RO		rxfifo is full
		[8]	RXB_FULL	RO		rx buffer is full
		[9]	TXFIFO_OverRead	RO		tx buffer is over read
		[10]	RXFIFO_Overflow	RO		rxfifo overflow
		[11]	TXB_RD_ERR	RO		TX buffer read error
		[12]	RXB_WR_ERR	RO		RX buffer write error

Please refer to Table 25-1 Explanation of Access in register list for “Access” definition

17.2.1 SPI_REG_SPI_EN

This register is used to enable SPI_MS. It can force the internal FSM back to IDEL mode at any time when the register is reset.

17.2.2 SPI_REG_SPI_START

Writing **1** to bit[0] starts the SPI operation in both master and slave modes. Ensure that a new SPI operation is initiated only after the previous operation is complete; otherwise, unpredictable errors may occur.

17.2.3 SPI_REG_INT_EN

This register is used to select the interrupt signals. There are 12 types of interrupt signals available, and any signal listed here can be used as the module's output interrupt. Write **1** to the corresponding register location to enable the desired interrupt.

17.2.4 SPI_REG_INT_CLR

Writing **1** to bit[0] clears the output interrupt.

17.2.5 SPI_REG_CFG

This register is used to configure the working mode of the SPI module:

- Write **0** to bits [1:0] to set the SPI to MODE0; write **2** to set it to MODE2.
- Write **1** to bit [2] to configure the SPI as a master, or **0** to configure it as a slave.
- Write **1** to bit [3] to enable the reverse order of each byte from LSB to MSB.
- Write **1** to bit [4] to enable the reverse order of bytes within a 32-bit word.
- Write a 3-bit value to bits [7:5] to adjust the master SPI clock rate.
- Write to bits [11:8] to select the corresponding chip enable.

17.2.6 SPI_BUF_EN

This register is used to enable DMA to access data ram as SPI_MS’s data buffer.

Writing **1** to bit[0] to enable the DMA module. Bit [2:1] are used to set the transfer threshold condition for the DMA when SPI data receiver. Setting this value to **0** means that when received data is available in the FIFO, the SDMA will immediately transfer it to the data RAM. If a value greater than **0** is set, the SDMA module will only start transferring data when the number of entries in the FIFO exceeds the specified threshold.

17.2.7 SPI_TXFIFO & SPI_RXFIFO

This register is used to write transmitter data to sending FIFO. It is a write only address. You should make sure there are enough free space to store the data before write.

This register is used to read received data from receiver FIFO. It is a read only address. You should make sure the received data are available.

17.2.8 SPI_TRXFIFO

This register is used to specify the number of bytes to write to the TXFIFO or read from the RXFIFO.

Bit[1:0] determine the number of bytes to read. For each APB read operation, **N + 1** bytes of data will be retrieved from the RXFIFO.

Bit[3:2] determine the number of bytes to write. For each APB write operation, **N + 1** bytes of data will be written to the TXFIFO.

17.2.9 SPI_TXBUF & SPI_RXBUF

This register is used to set the start address where the transmitter data are stored.

This register is used to set the start address where the received data are stored.

17.2.10 SPI_BUF_SIZE

This register is used to set the max number of byte that can be accept by SPI command.

Bit[11:0] is used to set TX buffer size.

Bit[27:16] is used to ser RX buffer size.

17.2.11 SPI_TRX_STATUS

This register is a read only register that stored the information of current SPI operation.

Bit[11:0] is used to read the number of bytes that has send through SPI.

Bit[27:16] is used to read the number of bytes that has received from SPI.

Bit [31:28] is used to check the data in RXFIFO which is ready for read.

17.2.12 SPI_EVENT

The register is read only register that is used to check the event happen when interrupt is happen. The bit arrangement is just corresponding the interrupt setup.

17.3 Application flow

17.3.1 SPI_MS common configuration flow

N.B. : The firmware would need to add NVIC_EnableIRQ (SPI_IRQ, GPIO_IRQ) to enable the CPU to accept spi interrupts & gpio interrupt

The common setting is used to configure the IO_MUX, GPIO, and SPI IRQ for SPI operation.

17.3.2 master mode

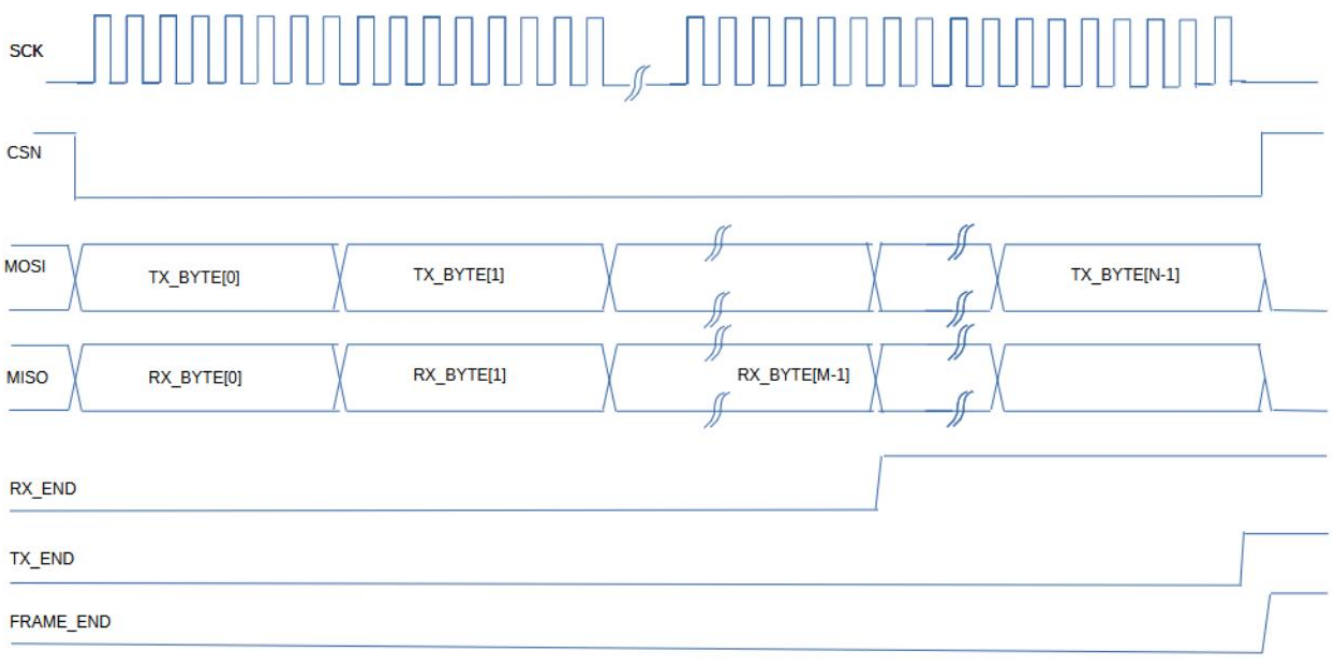


Figure 17-3: Master SPI Interface signal with Interrupts

The figure shows the signal on the SPI interface and the internally generated interrupts when SPI module works at master mode. In the example above, the master intends to transfer N bytes of TX data to the slave and receive M bytes of data from the slave simultaneously (where M ≤ N). The `RX_END`, `TX_END`, and `FRAME_END` interrupts are all generated in the system clock domain. The `RX_END` interrupt will be triggered when the `RX_BYTE[M-1]` is received and stored in the `RXFIFO`. Similarly, the `TX_END` interrupt will be triggered when the `TX_BYTE[N-1]` is sent. Finally, the `FRAME_END` interrupt will be triggered when `CSN` is set to 1

17.3.3 slave mode

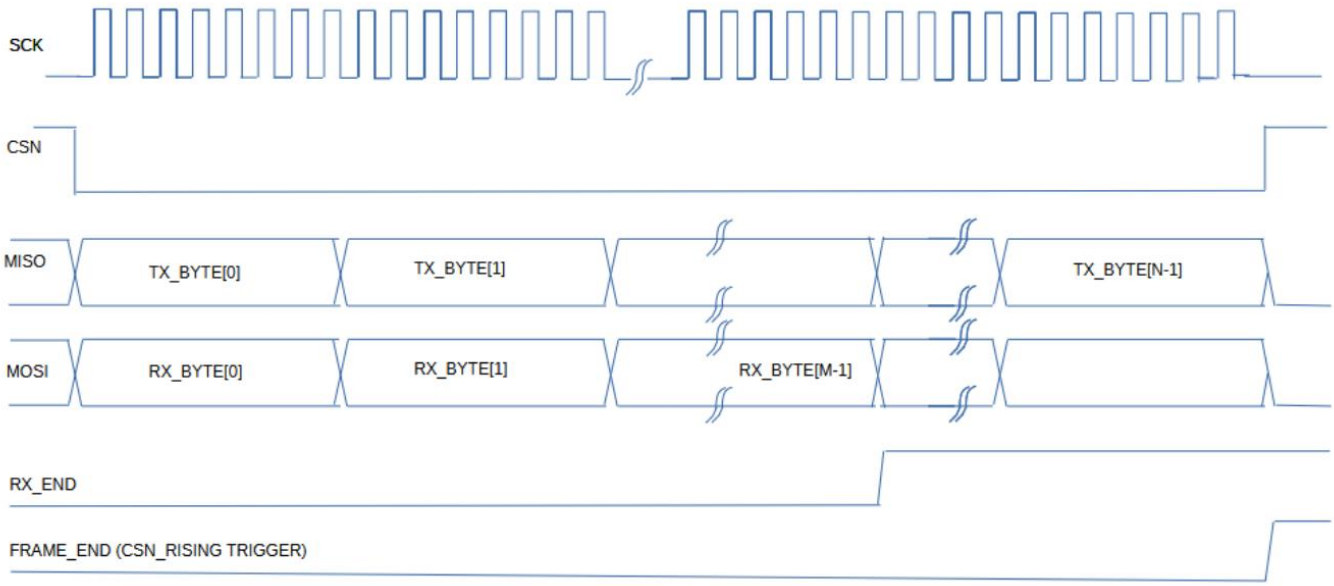


Figure 17-4: Slave SPI Interface signal with Interrupts

The figure shows the signals on the SPI interface and the internally generated interrupts when the SPI module operates in slave mode. The slave may or may not be aware when the CSN rises, depending on the master's signal and the configuration set in slave by the user. For example, if all setup is correct and the master intends to receive N bytes and transmit M bytes from the slave simultaneously (where M < N), the RX_END and GPIO_CSN_RISING triggers can be used as interrupts. The RX_END interrupt will be triggered when RX_BYTE[M-1] is received and stored in the RXFIFO. The GPIO csn rising is generated when the CSN set which means the host terminate current operation.

In the current chip, if the user sets N = M, the last byte (RX_BYTE[N-1]) will not be written to the RXFIFO. To ensure successful writing of the mentioned last byte, the master must provide one additional SCLK pulse.

18 UART — Universal asynchronous receiver

The Universal asynchronous receiver/transmitter with AHB master (AHBM) interface (UART-A) offers fast, full-duplex, asynchronous serial communication with built-in flow control (CTS, RTS) support in hardware at a rate up to 1 Mbps, and AHBM data transfer from/to RAM.

The mode of transmission is in the form of a packet, it consists of a start bit, data frame (8 bits), a parity bit, and stop bits (1/1.5/2/3 bits). It converts parallel data into serial data for transmission when sending data, and converts received serial data into parallel data when receiving data. The GPIOs used for each UART interface can be chosen from any GPIO on the device and are independently configurable. This enables great flexibility in device pinout and efficient use of board space and signal routing.

Base address	Peripheral	Instance	Description	Configuration
0x40002000	UART-A	UART-A0	Universal Asynchronous Receiver/ Transmitter with AHBM	
0x40003000	UART-A	UART-A1	Universal Asynchronous Receiver/ Transmitter with AHBM	

18.1 Functional description

18.1.1 UART Protocol

UART (Universal asynchronous receiver/transmitter) is an asynchronous serial communication protocol that transfers data byte by byte in the form of a packet. The packet consists of the data byte, a parity bit and encapsulated by the start bit and stop bit/bits. It converts parallel data into serial data for transmission when sending data and likewise received serial data into parallel data.

Start bit	Data Frame	Parity bit	Stop bit
1 bit	8 bit	0/1 bit	1/1.5/2/3 bit

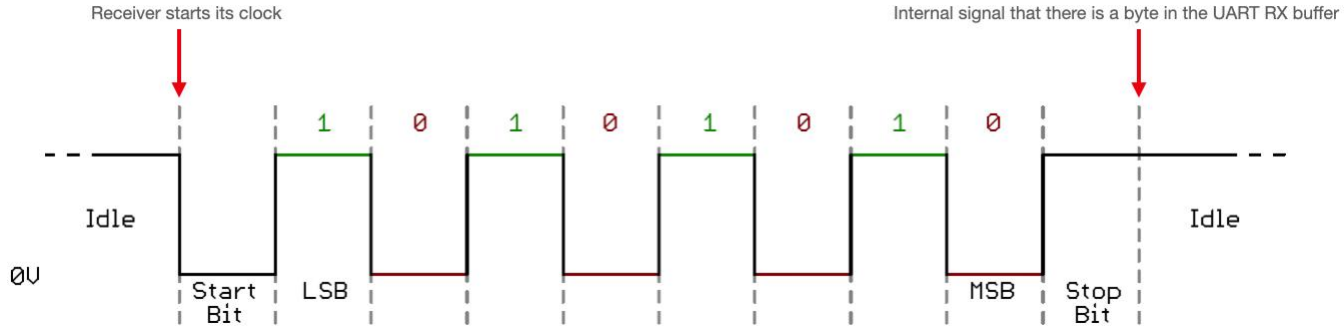


Figure 18- 1: UART transfer example (without parity)

18.1.2 Configurable features

The UART module support full-duplex operation with the feature supported as below:

- ➔ Programmable stop bit length (1, 1.5,2 and 3 bits)
- ➔ Programmable data length (8 bits)
- ➔ Programmable parity bit (even, odd, no-parity)
- ➔ Programmable hardware flow control
- ➔ Programmable interrupts events
- ➔ Programmable FIFO mode or SDMA mode

18.1.3 I/Os setup

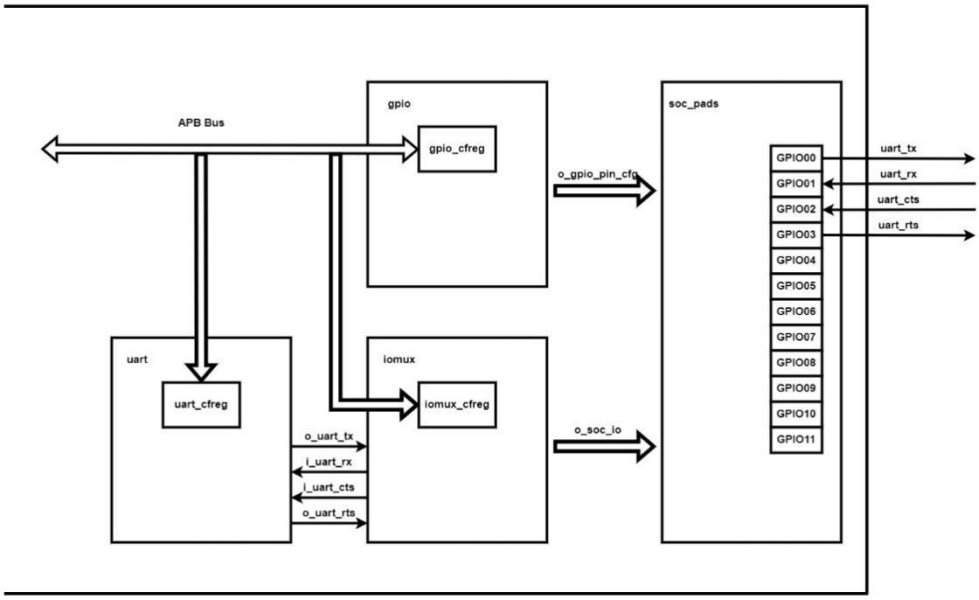


Figure 18- 2: High-Level block diagram of uart GPIO pin configuration

The I/O of the UART module will be configured to available GPIOs via the IOMUX to be connected to external UART devices. The I/O that will need to be configured are:

1. o_uart_tx
2. i_uart_rx
3. i_uart_cts (If hardware flow control is enabled for tx transfer)
4. o_uart_rts (if hardware flow control is enabled for rx transfer)

The user can also configure the GPIOs to be internally pulled-up to prevent High Z states if the connected line is not pulled-up.

N.B.: The o_uart_tx will only work as a driver during the uart tx transfer, once the tx transfer is done, the GPIO will go return to its High Z tristate state and will not drive the GPIO. If the user would want o_uart_tx to continue driving the GPIO output, the GPIO can be set as internally pull-up.

The same goes for uarts_rx, if the driver is a tristate driver, it is recommended to setup the GPIO as internally pulled-up to prevent high Z states.

18.1.4 FIFO & SDMA Mode

There are 2 configurable modes available for the UART module being:

- FIFO mode

Tx/Rx fifo will need to be operated according to the interrupts asserted. The max size of each fifo is 8 bytes. In the case of tx transfers, data will need to be written into the txd register for every transfer else the uart will be in idle state waiting for the next tx data to be written this cycle will be repeated up until the txb_max_nbytes is reached, any other data written into txd register will be ignored. Whereas for rx transfers, if the data is not read out from rxd register, the previous data will be skipped and rxfifo_overflow event will be asserted and can trigger the interrupt if enabled.

- SDMA mode

A part of the memory is allocated as uart tx and rx buffers and tx data will be prewritten and prepared in the tx buffers. Once the start is issued, for tx transfers, the sdma submodule will retrieve data directly from the memory and transfer it to the txfifo to be sent via the uart_tx submodule up until the txb_max_bytes defined is reached. Likewise, for rx tranfers, the sdma submodule will retrieve data from the rxfifo every time the current bytes in the rx fifo reaches the min_bytes defined and transfers it to the rxbuffer memory.

For both of the modes, the number of bytes sent or received can be checked from the uart_trx registers by reading the rxb_nbyte or txb_nbyte.

18.1.5 Hardware flow control

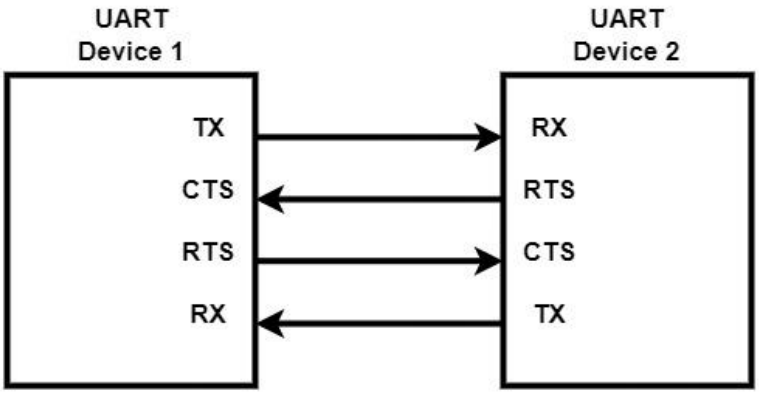


Figure 18-3: Illustration of two UART device connection with hardware flow control

The UART module supports hardware flow control dictated by Clear To Send (CTS) and Ready To Send (RTS) signals. The RTS signal of the Rx device will be connected to the CTS of the Tx device and the Tx device will wait for its CTS signal to be asserted before sending the data. For example, if the Tx device has started a Tx transfer with hardware flow control enabled, if the CTS signal received is not asserted, it will wait for the CTS signal to be asserted by the RTS of the Rx device before initiating the UART transfer.

N.B.: For the current version of the chip, CTS and RTS is configured as active high signals. UART0 and UART1 share the same CTS signal.

18.2 Registers

Table 18-1: Register list of UART

Offset	Reg	Range	Field	Access	Reset Value	Description
0x00	UART_EN	[1]	txen	RW	1'b0	Enable UART TX =0: disable TX =1: enable TX
		[0]	rxen	RW	1'b0	Enable UART RX =0: disable RX =1: enable RX
0x04	UART_TXCTRL	[1]	stop	WP	1'b0	1: To issue a tx stop pulse
		[0]	start	WP	1'b0	1: To issue a tx start pulse
0x08	UART_RXCTRL	[1]	stop	WP	1'b0	1: To issue a rx stop pulse
		[0]	start	WP	1'b0	1: To issue a rx start pulse
0x0c	UART_INT_EN	[15]	RXB_WR_ERR	RW	1'b0	enable interrupt by rxb_wr_err event
		[14]	TXB_RD_ERR	RW	1'b0	enable interrupt by txb_wr_err event
		[13]	BREAK_ERR	RW	1'b0	enable interrupt by rxb_ovf_err event
		[12]	FRAME_ERR	RW	1'b0	enable interrupt by frame_err event
		[11]	PARITY_ERR	RW	1'b0	enable interrupt by parity_err event
		[10]	RXFIFO_OVF_ERR	RW	1'b0	enable interrupt by rxfifo_ovf_err event
		[9]	CTS	RW	1'b0	enable interrupt by cts event
		[8]	RXB_FULL	RW	1'b0	enable interrupt by rxb_full event
		[7]	RXFIFO_FULL	RW	1'b0	enable interrupt by rxfifo_full event
		[6]	RXFIFO_EMTPY	RW	1'b0	enable interrupt by rxfifo_empty event
		[5]	RXFIFO_READY	RW	1'b0	enable interrupt by rxfifo_ready event
		[4]	RXD_READY	RW	1'b0	enable interrupt by rxd_ready event

		[3]	TXB_EMPTY	RW	1'b0	enable interrupt by txb_empty event
		[2]	TXFIFO_FULL	RW	1'b0	enable interrupt by txfifo_full event
		[1]	TXFIFO_EMPTY	RW	1'b0	enable interrupt by txfifo_empty event
		[0]	TXD_READY	RW	1'b0	enable interrupt by txd_ready event
0x10	UART_INT_CLR	[0]	int_clear	WP	1'b0	clear interrupt signal
0x14	UART_CFG	[31:12]	BAUDRATE	RW	20'd32	baud rate =n: baud rate is round(64MHz/n). n>=8: baud rate is from 100 baud to 8M
		[8]	RXBUF_WRAP	RW	1'b0	Reserved
		[7]	TRXBUF_EN	RW	1'b0	=0: txd/rxd FIFO mode, write/read to/from register TXD and RXD FIFO. =1: BUF mode enable. TX/RX Data in buffer that defined in register UART_TXBUF, UART_RXBUF.
		[6:5]	STOP_BITS	RW	2'd0	number of stop bits for transmission =0: 1 stop bits =1: 1.5 stop bits =2: 2 stop bits =3: 3 stop bits
		[4]	BYTE_ORDER	RW	1'b0	Reserved
		[3]	BIT_ORDER	RW	1'b0	bit order in a byte =0: lsb first =1: msb first
		[2:1]	PARITY	RW	2'd1	Parity bits =0: exclude parity bit =1: include even parity bit =2: include odd parity bit =3: reserved
		[0]	FLOW	RW	1'b0	Hardware flow control 0: disabled 1: enabled
0x18	UART_TXD	[31:0]	TXD	WE	--	Write one word (32 bits) to external TXD FIFO for transmission. Immediately start TX
0x1c	UART_RXD	[31:0]	RXD	RE	--	Read RX data (32 bits) from external RXD FIFO.
0x20	UART_TRXD	[3:2]	WRITE_BYTES	RW	2'd3	Number of bytes to be written into FIFO via apb 0: write 1 byte 3: write 4 bytes
		[1:0]	READ_BYTES	RW	2'd3	Number of bytes to be read from FIFO 0: read 1 byte 3: read 4 bytes
0x24	UART_TXBUF	[31:0]	TXB_POINTER	RW	32'd0	pointer to memory buffer for sending
0x28	UART_RXBUF	[31:0]	RXB_POINTER	RW	32'd0	pointer to memory buffer for receiving

0x2c	UART_BUF_SIZE	[29:28]	MIN_BYTES	RW	2'b0	Define the number of byte shall be received in fifo before transfer to rx buffer. n: n+1 byte
		[27:16]	RXB_MAX_BYTES	RW	12'd0	Maximum number of bytes can be saved in RXD buffer/FIFO
		[11:0]	TXB_MAX_BYTES	RW	12'd0	Maximum number of bytes to be sent. If TRXBUF is not enabled, it indicates the number of bytes in to be sent in FIFO mode.
0x30	UART_TRX	[31:28]	RXFIFO_NBYTE	RO	-	Number of bytes received in RXFIFO. It is used in rxd FIFO mode (i.e. trxbuf_en = 0)
		[27:16]	RXB_NBYTE	RO	-	number of bytes received
					-	
		[11:0]	TXB_NBYTE	RO	-	Number of bytes has been sent out
0x34	UART_EVENT	[19]	RXB_WR_ERR	RO	-	rx buffer write error
		[18]	TXB_RD_ERR	RO	-	tx buffer read error
		[17]	BREAK_ERR	RO	-	Break condition. The serial data input "0" for longer than the length of a data frame
		[16]	FRAME_ERR	RO	-	framing error detected. A valid stop is not detected
		[15]	PARITY_ERR	RO	-	parity error detected
		[14]	RXFIFO_OVF_ERR	RO	-	overrun error detected. New byte is received but FIFO is full.
		[11]	CTS	RO	-	CTS is activated (set low). clear to send
		[10]	RXB_FULL	RO	-	RX buffer is filled up
		[9]	RXFIFO_FULL	RO	-	rx fifo is full
		[8]	RXFIFO_EMTPY	RO	-	rx fifo is empty
		[7]	RXFIFO_READY	RO	-	One word received in RXD FIFO.
		[6]	RXD_READY	RO	-	One byte received in RXD FIFO, but potentially not yet transferred to RX buffer or bytes received in RXD register
		[5]	TXB_EMPTY	RO	-	Last TX byte has been sent out
		[4]	TXFIFO_FULL	RO	-	txfifo is full
		[3]	TXFIFO_EMPTY	RO	-	tx fifo is empty
		[2]	TXD_READY	RO	-	One byte has been sent out
		[1]	RX_ON	RO	-	RX has started
		[0]	TX_ON	RO	-	TX has started

Please refer to Table 25-1 Explanation of Access in register list for “Access” definition

18.2.1 UART_EN

This register enables uart tx/rx, this should be done as the 1st step in the application flow for uart to enable the module.

18.2.2 UART_TXCTRL & UART_RXCTRL

Uart_txctrl register starts and stops the tx transfer of the uart module. For every start issued the txfifo and tx_nbytes will be cleared meanwhile all configurations for the uart will remain the same. The start control should only be issued once all the configurations for the uart transfer is done. The same applies for uart_rxctrl for rx transfer.

N.B. If the uart_tx is not in idle state the start will not reset the tx_nbytes. To ensure uart_tx will be in idle state, the user can either issue a txctrl stop or check whether the previous transfer is done by checking whether tx_on event is still asserted by reading the event register before issuing the next start. While for rx transfers, the user should issue a rxctrl stop before issuing the next start if the state of the previous rx transfer is unsure.

18.2.3 UART_INT_EN & UART_EVENT

This register enables the interrupt of the uart module to be triggered by certain events defined. Whenever an interrupt is asserted, uart_event can be read out to determine which event triggered the interrupt knowing which interrupts are enabled.

18.2.4 UART_INT_CLR

This register clears the interrupt of the uart module.

18.2.5 UART_CFG

This register configures the transfer settings for the uart transfer. Configurations should be done in according to the settings of current uart transfer. Trxbuf_en selects whether FIFO mode or SDMA mode is to be used while the others are in accordance to standard uart transfer protocol.

18.2.6 UART_TXD & UART_RXD & UART_TRXD

Uart_txd denotes the data to be transferred in FIFO mode. For every data written into the uart_txd register, the number of bytes defined in uart_trxd will be transferred to the txfifo. For example if uart_trxd_write_bytes is set to 0, only 1 byte will be transferred from uart_txd to txfifo everytime a write transfer is done into uart_txd register. In FIFO mode, if txctrl start is issued the uart module will wait until a data is written into uart_txd register and immediately issue a tx transfer after.

Likewise, for uart_rxd, if uart_trxd_read_bytes is set to 1, everytime rxfifo receives a byte of data, the data will be stored in the rxfifo to be read out via uart_rxd to clear up rxfifo data slots, else the next data received will be skipped.

The intentional usage of uart_txd and uart_rxd whether byte by byte or 4xbytes by 4xbytes will be defined in uart_trxd_write_bytes and uart_trxd_read_bytes.

18.2.7 UART_TXBUF & UART_RXBUF & UART_BUF_SIZE

Uart_txbuf and uart_rxbuf denote the start address for the memory buffer for tx and rx transfers.

Uart_buf_size denotes the size of the memory buffers and min_bytes to be received in the fifo before being sent to the rx_buffer. Typically, the min_bytes can be set to 0 so that the SDMA submodule will transfer the data for every byte received in the rxfifo.

18.2.8 UART_TRX

Uart_trx register enables the user to read out the number of bytes currently in the rxfifo, number of bytes received via rx transfer and number of bytes sent via tx transfer. These can be used as pseudo interrupts for tx/rx transfers.

18.3 Application flow

18.3.1 *Uart common configuration flow*

N.B. : The firmware would need to add NVIC_EnableIRQ (UART0_IRQ) to enable the CPU to accept uart0 interrupts

Example format: (Module Name) -> (Register name) . (Register field name) = Value

E.g. **SCR->RST_CTRL.RST_UART0 = 1**

where **SCR** is module, **RST_CTRL** is register name (based on Table 7-1)**. RST_UART0** is register field name **(**based on Table 7-1).

Table 18-2: Usage example of UART

Description	Example

1. Release reset from scr module for uart module	SCR->RST_CTRL.RST_UART0 = 1 (based on Table 7-1)
2. Configure GPIO pins for uart_tx & uart_rx (uart_cts & uart_rts optional)	IOMUX->GPIO00.MODE_SEL = 01 //soc peripherals IOMUX->GPIO00.SRC_SEL = 12 //uart0_tx IOMUX->GPIO01.MODE_SEL = 01 IOMUX->GPIO01.SRC_SEL = 13 //uart0_rx IOMUX->GPIO02.MODE_SEL = 01 IOMUX->GPIO02.SRC_SEL = 18 //uart_cts IOMUX->GPIO03.MODE_SEL = 01 IOMUX->GPIO03.SRC_SEL = 15 //uart0_rts (based on Table 11-4)
3. Configure uart_cfg register according to settings for current uart transfer	UART->CFG.BAUDRATE = 0x01A0A //configure baudrate = 9600 UART->CFG.TRXBUF_EN = 0 //0:FIFO mode 1:SDMA mode UART->CFG.STOP_BITS = 0 //1 stop bit UART->CFG.BIT_ORDER = 0 //lsb first UART->CFG.PARITY_BIT = 0 //no parity bit UART->CFG.HARDWARE_FLOW_CONTROL_EN = 0 //disable hardware flow control (based on Table 18-1)
4. Enable uart tx and rx	UART->EN.TX_EN = 1 UART->EN.RX_EN = 1 (based on Table 18-1)

18.3.2 *FIFO mode tx transfer*

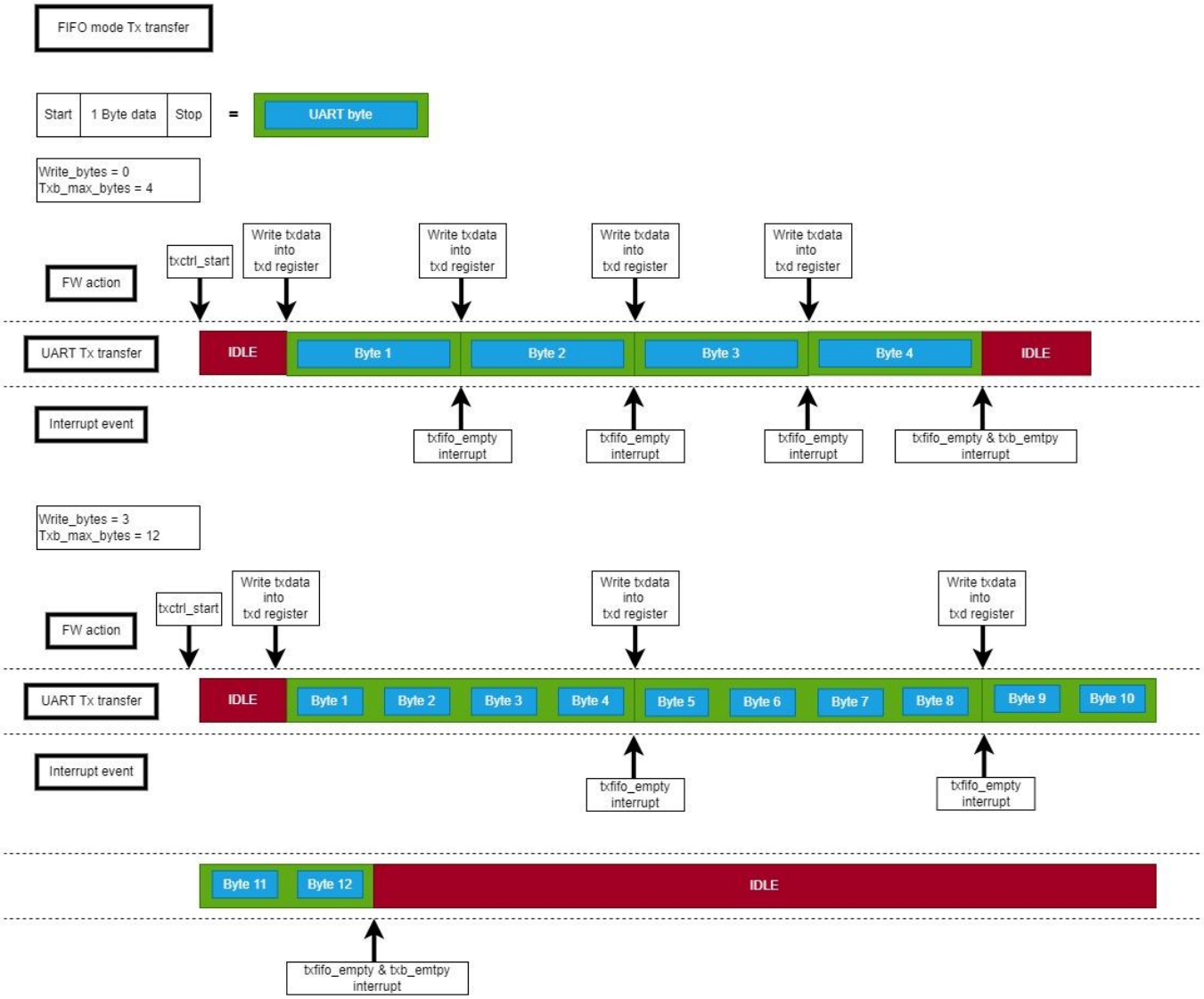


Figure 18-4: Illustration of example tx transfer in fifo mode

N.B.: In FIFO mode, once txfifo is empty without any valid data, the uart module will be in idle state waiting for the next txdata to be set into txd register.

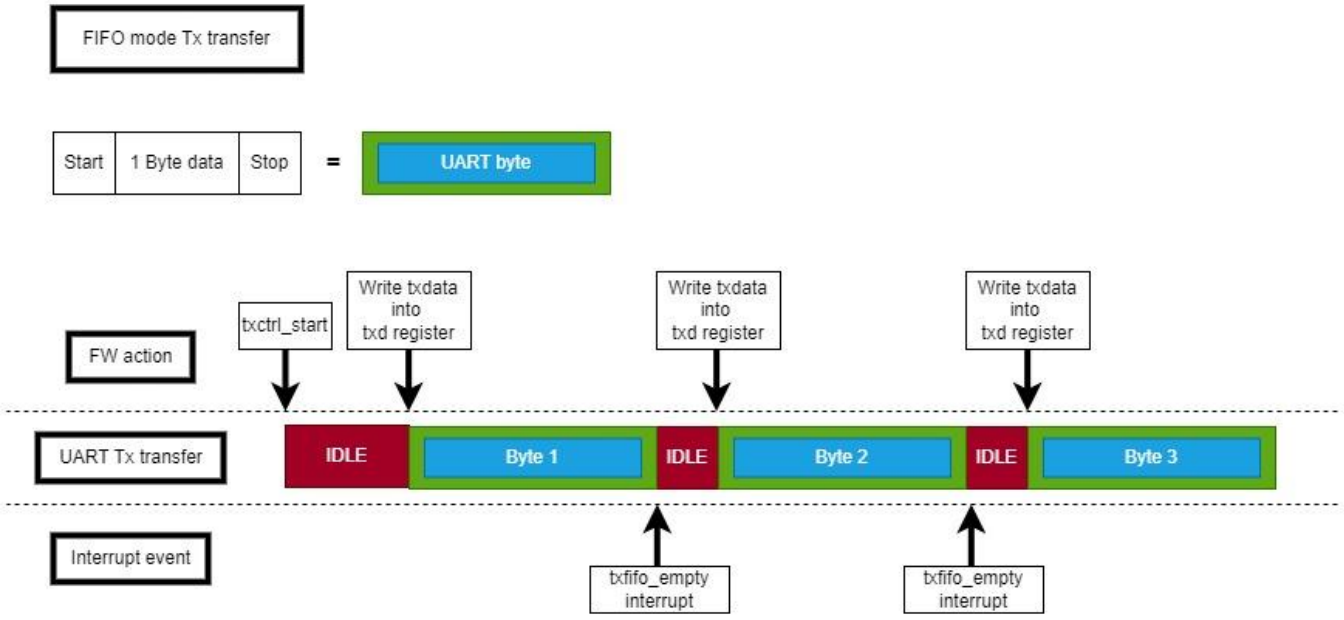


Figure 18-5: Illustration of example idle periods between tx transfer in fifo mode

18.3.3 FIFO *mode* rx transfer

N.B. : Since for rx transfers, it is assumed that the number of bytes for the rx transfer is unfixed thus the flow is set to retrieve the data from rxfifo for every byte received.

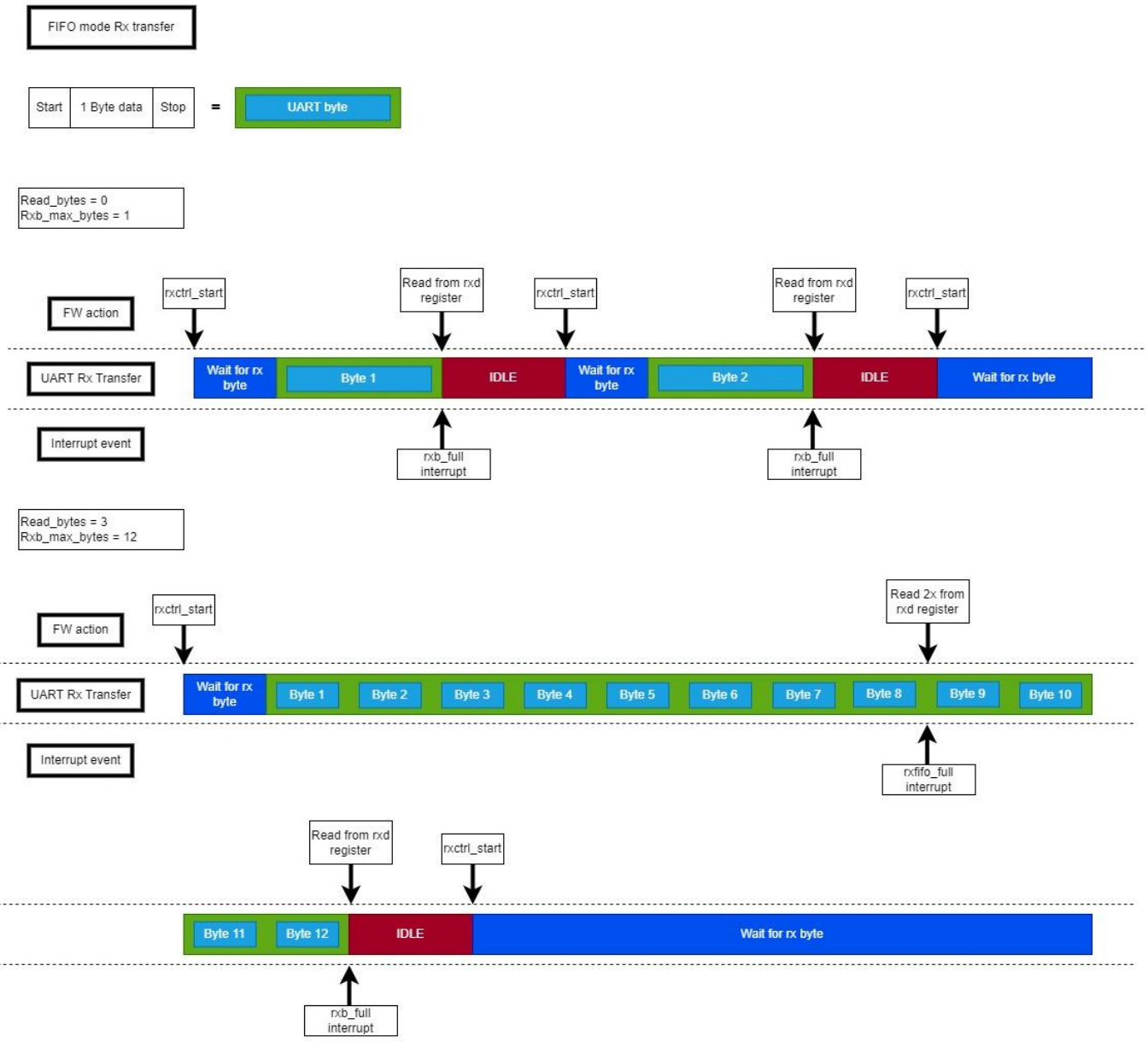


Figure 18-6: Illustration of example rx transfer in fifo mode

18.3.4 SDMA mode tx transfer

Figure 18-7: Illustration of example tx transfer in SDMA mode

N.B. If the uart_tx is not in idle state the start will not reset the tx_nbytes. Txctrl_stop will need to be issued before issuing a txctrl_start to restart the sdma transfer.

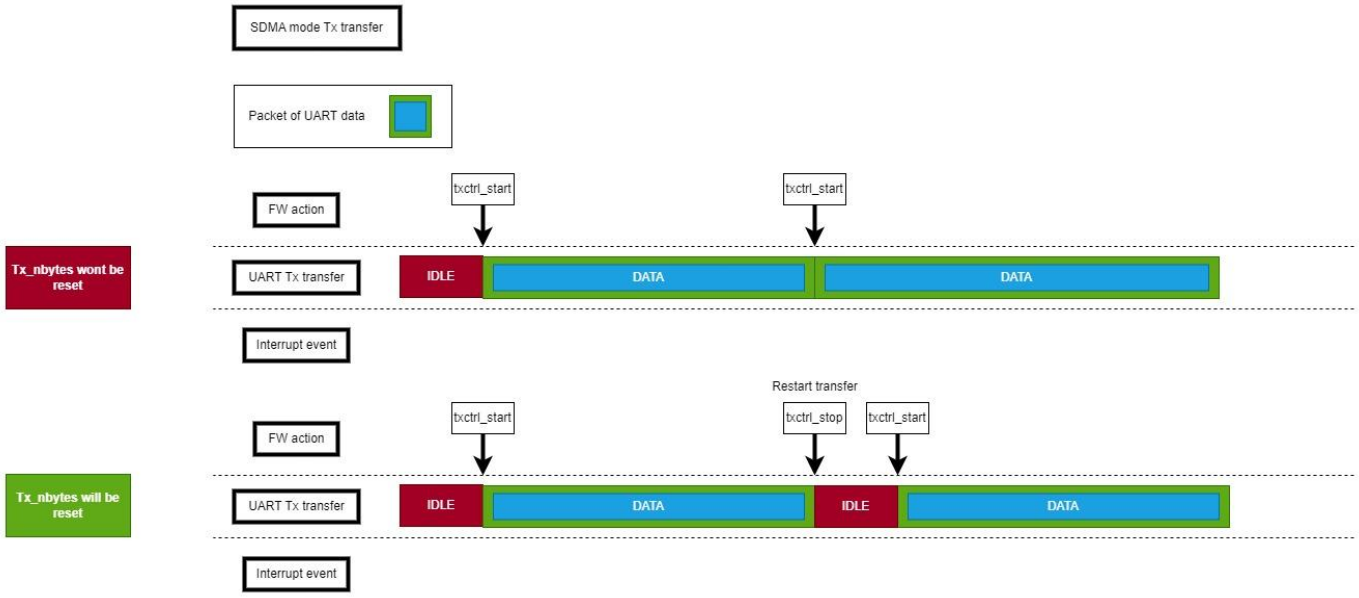


Figure 18- 8: Illustration of example of restart of tx transfer in SDMA mode

18.3.5 SDMA mode rx transfer

N.B. : Since for SDMA mode rx transfers, it is assumed the number of bytes to be received is fixed thus having a fixed rx buffer size. However there are other methods to accommodate unfixed rx transfer sizes

Figure 18- 9: Illustration of example rx transfer in SDMA mode

N.B. Same principles for rx_nbytes applies here. If the uart_rx is not in idle state the start will not reset the rx_nbytes. Rxctrl_stop will need to be issued before issuing a rxctrl_start to restart the sdma transfer.

19 CRC with AHBM

19.1 Function description

The CRC module supports fully customizable parameterized configuration, including INIT, POLY, XOROUT, CRC_ORDER (8,16,32), REFIN, REFOUT.

CRC stands for Cyclic Redundancy Check. It helps to detect errors or data corruption that may occur during data transmission or storage. The CRC algorithm generates a fixed-size checksum or hash value from a block of data, typically a message or a packet. This checksum is appended to the data before transmission or storage. When the data is received or read back, the recipient recalculates the CRC checksum and compares it to the received checksum. If the two checksums match, it indicates that the data was received or read back correctly without errors. If the checksums do not match, it suggests that errors have occurred during the data transfer or storage, and the data may be corrupted. This reduces the risk of undetected errors and improving the overall reliability of the system.

1. Efficient error detection – It is capable of detecting different types of errors including single bit errors, multiple-bit errors. It is particularly effective at detecting burst errors where consecutive bits or bytes are corrupted.
2. Fixed-size checksum – The size of CRC check sum is fixed and determined by the chosen CRC algorithm. The variants include CRC-8, CRC-16, and CRC-32, each with different sizes.

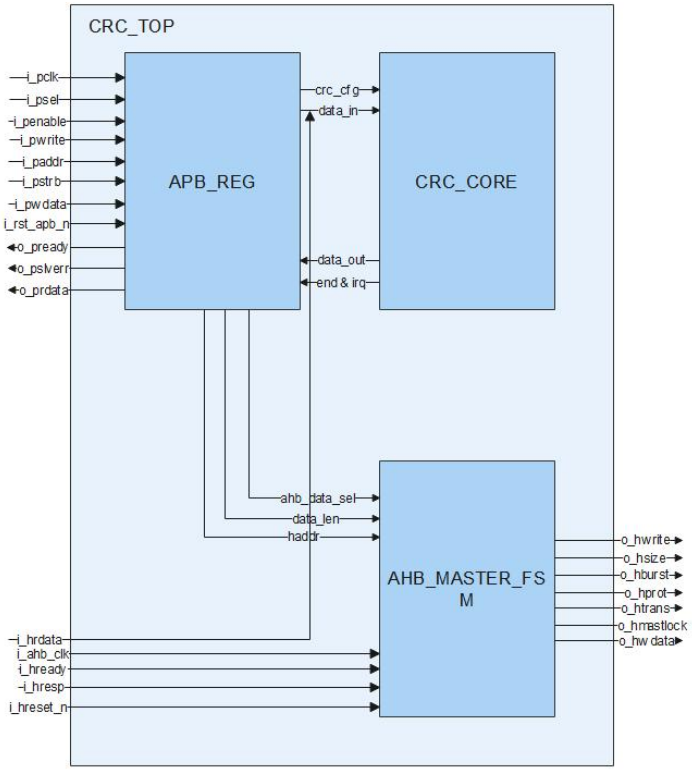


Figure 19-1 : Block diagram of CRC

19.2 Operation

The input data to be calculated can be selected from APB register or from memory on AHB bus, which is defined in register crc_data_sel in Table 19-1. If the data are from APB registers, users need to write data to APB register crc_in and read crc result from crc_result in Table 19-1. If the data are from memory on AHB, users need to specify the data length and start address in the memory, i.e., crc_ahb_addr and crc_data_len.

The state diagram for the AHB master controller is as given in Figure 3-2. There are four states of operation namely

- IDLE
- NSEQ (set AHB start address)
- SEQ (start read operation)
- END (end read operation)

The data flow can be explained as

- When the reset signal is applied all the signals are reset and the state goes into the IDLE state.
- When the s_crc_start signal is applied, the state goes into the NSEQ state, haddr_i will be assigned to the address written in the apb register, htrans_i will be assigned to 2b'10(NSEQ).

- Then the state goes into the SEQ state with no condition, htrans_i will be assigned to 2'b11(SEQ). Incremental reading process starts.
- When r_haddr is equal to end_addr, and the i_hready is high, the state goes into the END state, htrans_i will be assigned to 2'b00(IDLE), waiting for last data with vld sig.
- When the i_hready is high, the state goes into the IDLE state, htrans_i will be assigned to 2'b00(IDLE). Reading process ends

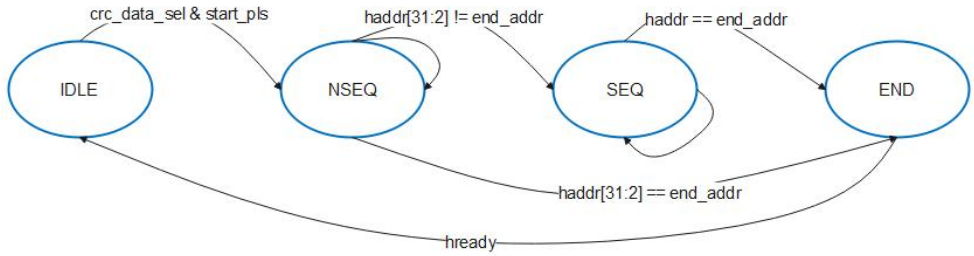


Figure 19-2: AHB Master controller FSM for CRC

19.3 Registers

Table 19-1: Register list of CRC

Offset	Reg	Range	Field	Access	Reset Value	Description
0x0	crc_en	[0]	en	RW	1'b0	enable/disable crc
0x4	crc_cfg	[5]	byte_order	RW	1'b1	byte order: 0: bit [31:24] first in. 1: bit [7:0] first in.
		[4]	init_value	RW	1'b0	crc_initial value 1 or 0
		[3:2]	order_sel	RW	2'd0	crc_order 00:8bit 01:16bit 10:32bit
		[1]	refout	RW	1'b1	output reverse 0: enable 1: disable
		[0]	refin	RW	1'b1	input reverse 0: enable (LSB of byte first) 1: disable (MSB first in)
0x8	crc_poly	[31:0]	poly	RW	32'd0	crc_poly
0xc	crc_xor	[31:0]	xor	RW	32'd0	crc_output_xor_value
0x10	crc_in	[31:0]	data	WE	32'd0	crc_input_data.
0x14	crc_result	[31:0]	data	RO	32'd0	crc_output_data
0x18	crc_data_sel	[0]	sel	RW	1'b1	crc_data_source_0 apb 1 ahb =0: apb =1: ahb
0x1c	crc_ahb_addr	[31:0]	addr	RW	32'd0	crc_ahb_addr, shall start from word boundary. i.e., addr[1:0]=00.
0x20	crc_data_len	[15:0]	data_len	RW	16'd0	number of bytes input to crc.
0x24	crc_start	[1]	Init_en	RW	1'b0	enable initialize crc reg
		[0]	start	WP	1'b0	crc_start_pls
0x28	crc_irq	[1]	en	RW	1'b0	enable irq
		[0]	clear	WP	1'b0	crc_irq_clear
0x2c	crc_end	[0]	end	WS	1'b0	CRC result ready event to indicate crc end. Write 0 to clear it.

Please refer to Table 25-1 Explanation of Access in register list for "Access" definition

19.4 CRC flow with separate data frame

If the data frame is more than 4KB, it can be implemented by dividing the frame into multiple blocks. Only the first cycle of the data frame requires the initialization of registers and the start of CRC calculation; the subsequent cycles only need to start the CRC calculation.

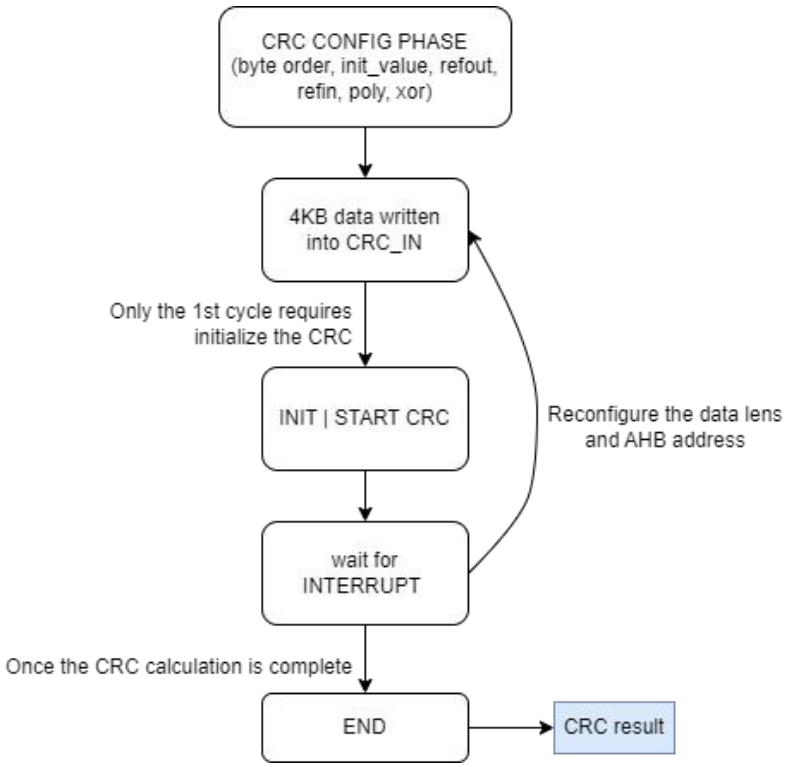


Figure 19-3: Flow of AHB CRC

20 I2C Master

The Inter-Integrated Circuit master is used for the purpose of data transfer between different peripherals. It allows transfer of information using only two wires: a serial data line (SDA) for bidirectional data transfer and a serial clock line (SCL) for synchronization. The I2C interface of this module is connected to bi-directional pads. With an external pull-up resistor on the SDA and SCL lines, the standard I2C protocol can be used to communicate to off-chip I2C peripherals.

Figure 20-1: I2C data frame example

20.1 Module description

The I2C module acts as a Master I2C device initiating read or write I2C data transactions. The I2C module also works in conjunction with DMA module to support multiple bytes read or write transfer.

I2C standard master:

- Start, stop conditions, acknowledge.

- Seven bits addressing.
- I2C single master systems, no arbitration (push/pull on SCL pin).
- No clock stretching

Full synchronous design:

- I2C clock period is programmable and derived from the main clock.
- The I2C SDA drive timing is programmable.

Dataflow control:

- Interrupt based, upon data TX/RX data buffers.
- Polling based, using status register.
- Using flow control together with DMA, using o_dma_ctrl output.

20.1.1 I/Os setup

The I/O of the I2C module will be configured to available GPIOs via the IOMUX to be connected to external I2C bus. The I/O that will need to be configured are:

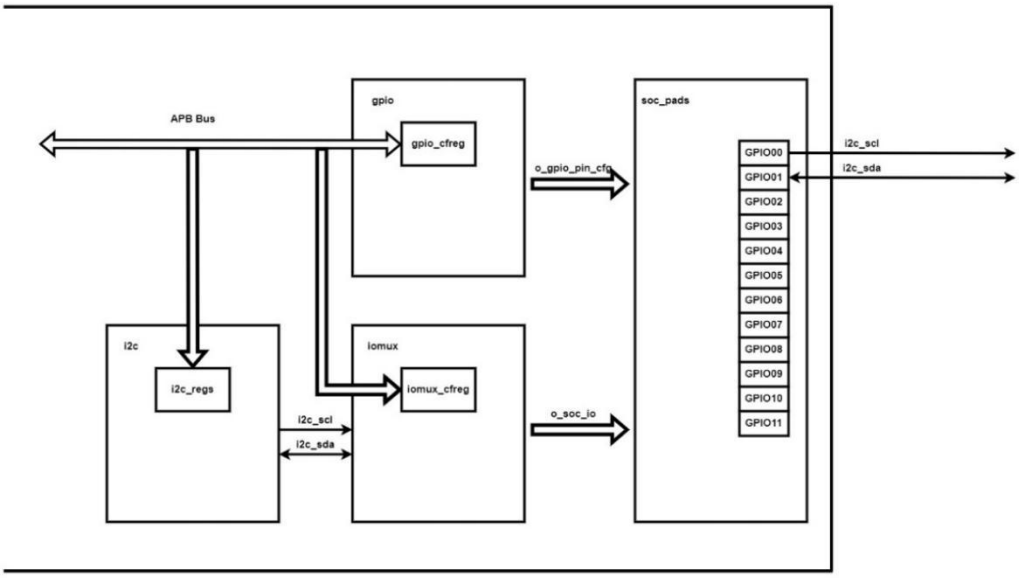


Figure 20- 2: High-Level block diagram of I2C GPIO pin configuration

1. i2c_scl (serial clock line)
2. i2c_sda (serial data line)

The user can also configure the GPIOs to be internally pulled-up to prevent High Z states if the connected line is not pulled-up

N.B.: The I2C line is conventionally setup as a pull-up line. If there is no pull up for the external I2C line. The GPIO can be configured to have an internal pull-up. Please refer to GPIO application notes for the GPIO pull-up setup.

20.1.2 Basic transfer setup

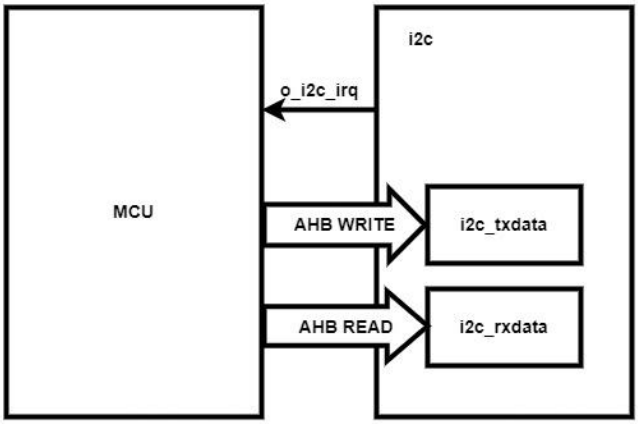


Figure 20-3: MCU interaction with I2C data registers

In basic transfer setup, txdata and rxdata registers will need to be operated alongside with the interrupt. The user will need to enable txdata_empty interrupt for I2C write transfers and rxdata_full interrupt for I2C read transfers. Therefore, whenever an I2C interrupt occurs, the firmware will need to operate the txdata and rxdata registers by writing or reading from it. Alternatively, the status register can be read to determine when to operate the txdata or rxdata.

20.1.3 DMA transfer setup

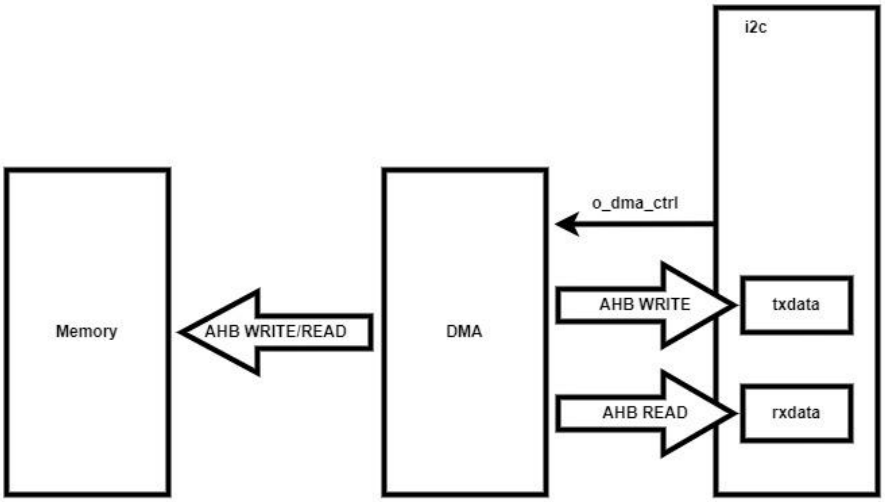


Figure 20-4: DMA data transfer between I2C data registers and memory

In DMA transfer setup, the data transfer of i2c module works in conjunction with the dma module. In the i2c write case, whenever txdata is empty, the i2c module will assert o_dma_ctrl which is hardwired to the flow control channel 0 of the dma module to signal that it is ready for the dma module to transfer the next txdata to be sent via i2c. On the other hand, for i2c read whenever rxdata is full o_dma_ctrl will be asserted and data will be transferred away from the i2c rxdata buffer. Therefore, this eliminates the need for the CPU to operate the txdata and rxdata registers. The DMA will need to be configured accordingly for this setup to function.

Refer section *DMA — Direct Memory Access.*

20.2 Registers

Table 20-1: Register list of I2C

Offset	Reg	Range	Field	Access	Reset Value	Description
0x00	cr (control)	[31:20]	byte_count	RW	12'h0	Total no of bytes
		[18]	no_stop	RW	1'b0	1: No stop bit at the end of transaction

		[17]	NACK_last_byte	RW	1'b0	1: Return NACK after last byte of transfer
		[16]	incl_ra	RW	1'b0	1: include register address
		[15:8]	reg_addr	RW	8'h00	Register Address
		[7]	rw	RW	1'b0	RW bit 0: Write 1: Read
		[6:0]	dev_addr	RW	7'b0	Device address
0x04	fr (config)	[28]	dma_sel	RW	1'b0	DMA Select (for o_dma_ctrl output) 0: rxdata_full controlled 1: txdata_empty controlled
		[27:16]	twdrive	RW	12'h0ff	drive period
		[13]	cr_i2c_enable	RW	1'b0	1: enable
		[12]	prop_mode	RW	1'b0	Reserved
		[11:0]	twscl	RW	12'hfff	clock period
0x08	i2c_rxdata	[31:0]		RO	32'h0	Rx Read Data Register
0x0C	i2c_txdata	[31:0]		RW	32'h0	Tx Write Data Register
0x10	Status	[2]	i2c_ready	RO	1'b0	I2C transaction done status signal
		[1]	txdata_empty	RO	1'b0	I2C txdata empty status signal
		[0]	rxdata_full	RO	1'b0	I2C rxdata full status signal
0x14	i2c_irq	[3]	i2c_nack_irq_en	RW	1'b0	Enable i2c nack irq
		[2]	i2c_ready_irq_en	RW	1'b0	Enable i2c ready irq
		[1]	i2c_txdata_irq_en	RW	1'b0	Enable i2c txdata empty irq
		[0]	i2c_rxdata_irq_en	RW	1'b0	Enable i2c rxdata full irq
0x18	irq_reg	[3]	i2c_nack_irq	RO	1'b0	i2c nack received irq status register
		[2]	i2c_ready_irq	RO	1'b0	i2c transaction done irq status register
		[1]	i2c_txdata_irq	RO	1'b0	i2c txdata empty irq status register
		[0]	i2c_rxdata_irq	RO	1'b0	i2c rxdata full irq status register

Please refer to Table 25-1 Explanation of Access in register list for “Access” definition

20.2.1 Cr Register (control)

This register is used to control the I2C transfer. This register is also used as a start signal for the I2C module, once a new config is written into this register, the I2C transaction will be initiated.

- Byte_count
  This bit space denotes the total no of bytes for configured for the current transfer. The I2C master module will continue reading or writing up until the byte_count is reached.

- No_stop
  This bit space controls whether a stop bit should be included in the I2C transfer for a repeated start condition. This option allows for the I2C master to maintain control of the I2C bus without releasing it.

- NACK_last_byte
  This bit space controls whether a NACK should be sent after sending/receiving the last byte of the I2C transfer. Typically, a NACK should be sent by the I2C master if the master is done receiving each data from the slave during a I2C read.

- Incl_ra & reg_addr
  This bit space controls whether a register address should be included in the transfer. This is an optional configuration and if incl_ra is set to 1 register address should be written into reg_addr bit space else 0x00 would be sent via the I2C transfer.

- Rw
  This bit space configures the RW bit sent via I2C transaction 1: I2C Read transaction 0: I2C Write transaction.

- Dev_addr
  This bit space denotes the device address for the I2C transfer. If left empty, the I2C transfer would send out 0x00 for the device address.

20.2.2 Fr Register (config)

This register is used to configure the I2C module.

- Dma_sel
  This bitspace configures I2C to use rxdata_full or txdata_empty event to control o_dma_ctrl output. This output is connected to dma channel 0 of all flow control channels and used as a dma request signal to signal the dma to write to i2c_txdata register of read from i2c_rxdata register.
- Twdrive
  This bitspace configures the drive period of the SDA line during which SCL is pulled low. Typically, this can be set as ¼ of Twscl.
- Cr_i2c_enable
  This bitspace enables the I2C module.
- Twscl
  This bitspace configures the SCL period for the I2C transfer. Below shows the drive periods of Twdrive and Twscl.

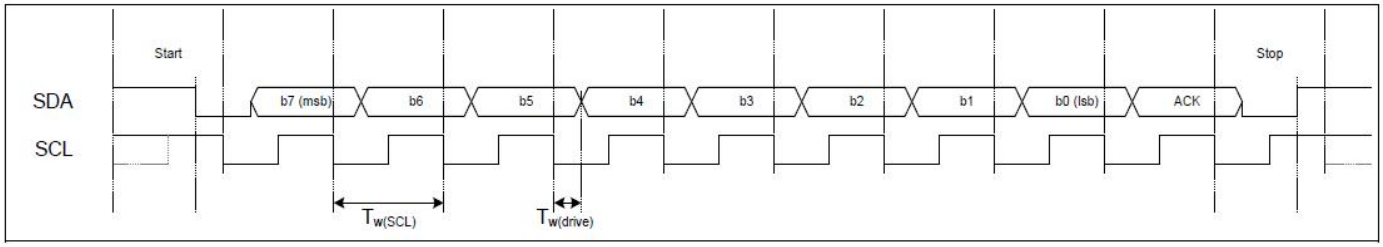


Figure 20-5 : I2C transfer timing diagram

20.2.3 I2C Rxdata Register

This register stores the rxdata received during a I2C Read transfer. If data bytes more than 4 are to be read, this register needs to be operated appropriately by either reading manually or setting up a data channel with the dma module. Else if data is not read out, the incoming data will overwrite the previous data resulting in an error.

20.2.4 I2C Txdata Register

This register is used to configure the txdata transferred during a I2C Write transfer. Similarly, this register will need to be operated appropriately by writing in data manually or setting up a data channel with the dma. Else the I2C write transfer will repeat the same data stored.

20.2.5 Status Register

This register is used to check the status of the I2C register.

- I2c_ready
  This event signal will be asserted once the I2C transaction is complete.
- Txdata_empty
  This event signal will indicate that the current txdata is fully sent and i2c_txdata register should be updated with the next txdata.
- Rxdata_full
  This event signal will indicate that the i2c_rxdata register is currently full and should be read out.

20.2.6 I2C IRQ Register

This register is used to configure interrupts for the I2C module.

- I2c_nack_irq_en
  Enable interrupt to trigger whenever I2C NACK is received.
- I2c_ready_irq_en

Enable interrupt to trigger whenever I2C transaction is done.

- I2c_txdata_irq_en
  Enable interrupt to trigger whenever I2C txdata register is empty.

- I2c_rxdata_irq_en
  Enable interrupt to trigger whenever I2C rxdata register is full.

20.2.7 IRQ Register

This register enables the user to check which interrupt triggered the register whenever an interrupt occurs. It will only be cleared when this register is read out.

20.3 Application flow

20.3.1 *I2C common configuration flow*

N.B.: The firmware would need to add NVIC_EnableIRQ (I2C_IRQ) to enable the CPU to accept I2C interrupts

Example format: (Module Name) -> (Register name) . (Register field name) = Value

E.g. **SCR->RST_CTRL.RST_I2C = 1**

SCR: module, **RST_CTRL:**register name(based on Table 20-1)**, RST_I2C:** register field name(based on Table 20-1)**.**

Table 20-2: Usage example of I2C

Description	Example
5. Release reset from scr module for I2C module	SCR->RST_CTRL.RST_I2C = 1 (based on Table 7-1)
6. Configure GPIO pins for i2c_scl and i2c_sda	IOMUX->GPIO00.MODE_SEL = 01 //soc peripherals IOMUX->GPIO00.SRC_SEL = 20 //i2c_scl IOMUX->GPIO01.MODE_SEL = 01 IOMUX->GPIO01.SRC_SEL = 21 //i2c_sda (based on Table 11-4)
7. Configure I2C config register	I2C->FR.TWSCL = 0x280 //64Mhz/640 = 100kHz I2C->FR.CR_I2C_ENABLE = 1 //enable I2C module I2C->FR.TWDRIVE = 0xA0 //1/4 of TWSCL (based on Table 20-1)

20.3.2 *Basic Write Flow*

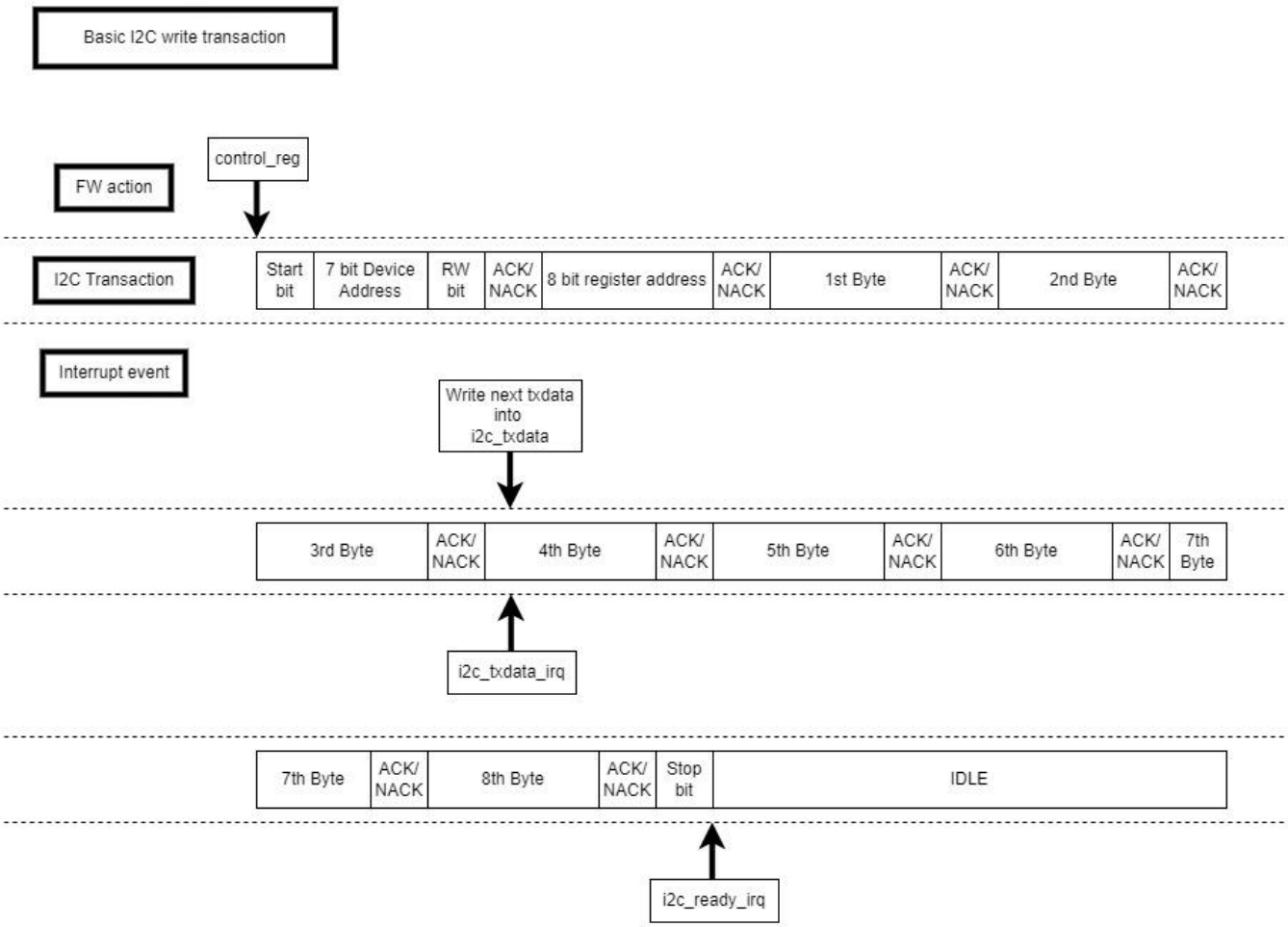


Figure 20-6 : Example of basic I2C write transaction

20.3.1 *Basic Read flow*

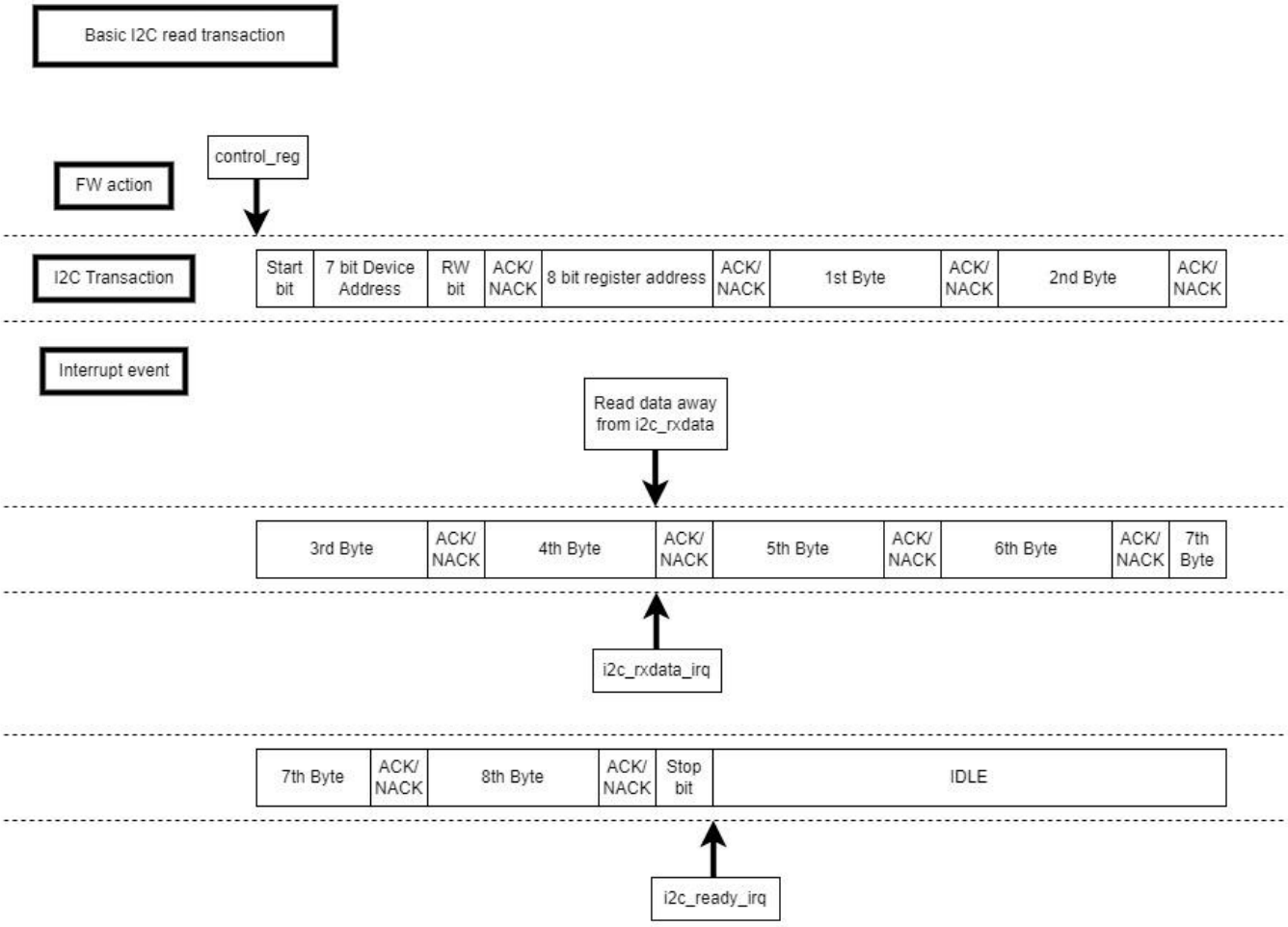


Figure 20- 7 : Example of basic I2C read transaction

20.3.1 *Write with DMA flow*

DMA data channel will need to be configured before starting I2C transaction. Please refer to DMA application notes for more details on DMA module. The DMA will be setup with a I2C txbuffer as the source address and I2C->I2C_TXDATA register as the destination address.

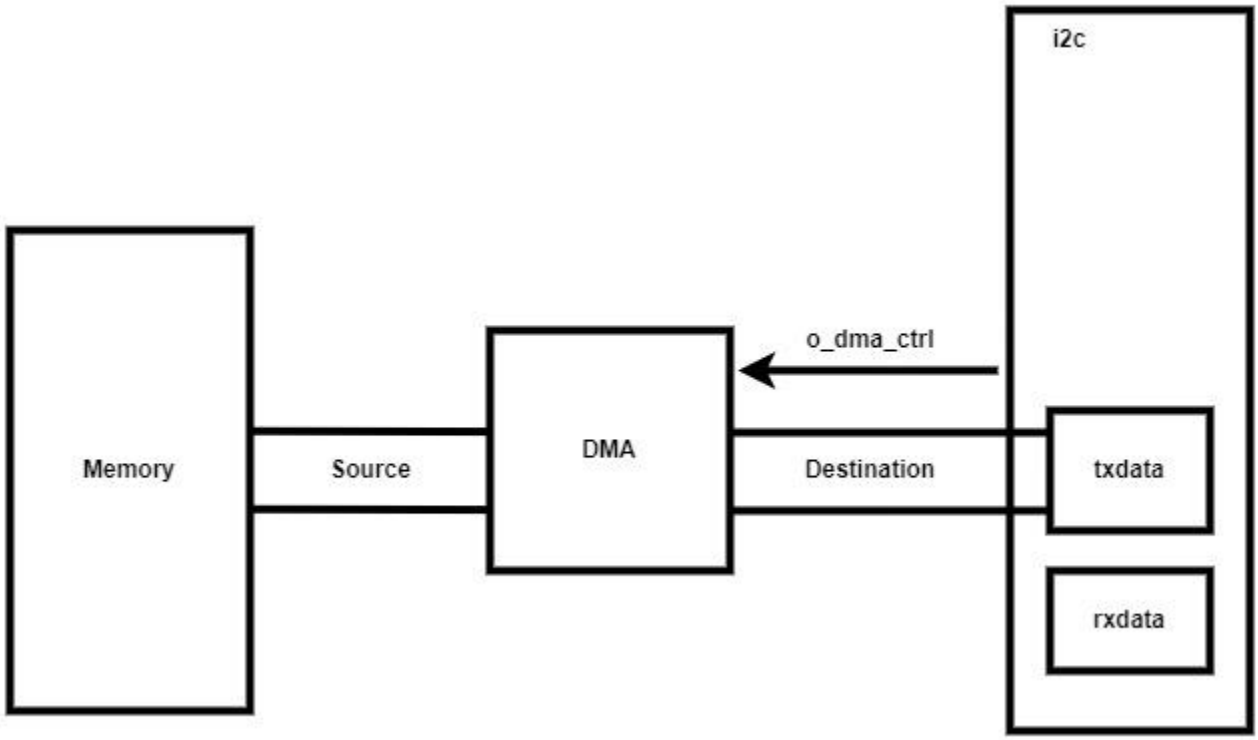


Figure 20-8： I2C write transaction with DMA setup

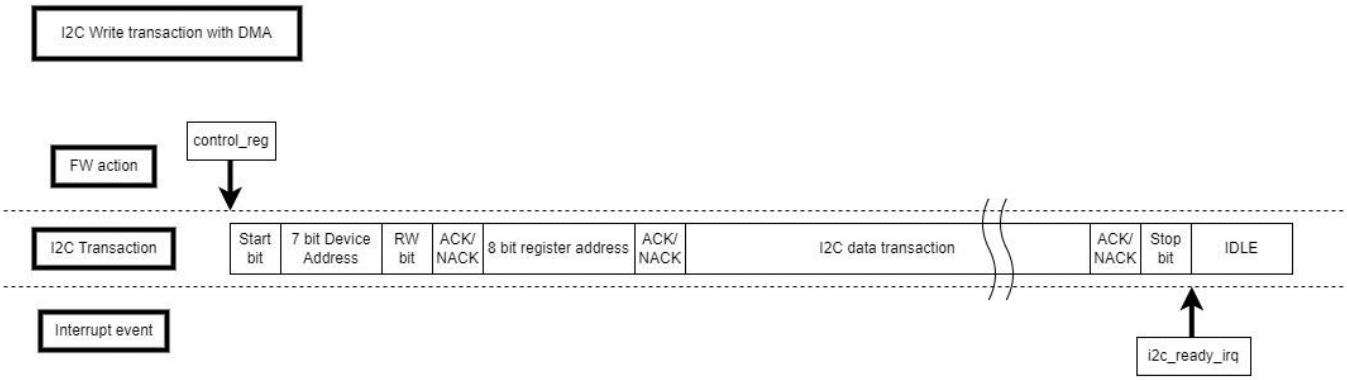


Figure 20-9： Example of I2C transaction with DMA

20.3.1 *Read with DMA flow*

Similarly, a DMA data channel will need to be pre-configured before starting I2C transaction. The DMA will be setup with a I2C->I2C_RXDATA as the source address and the I2C rxbuffer as the destination address.

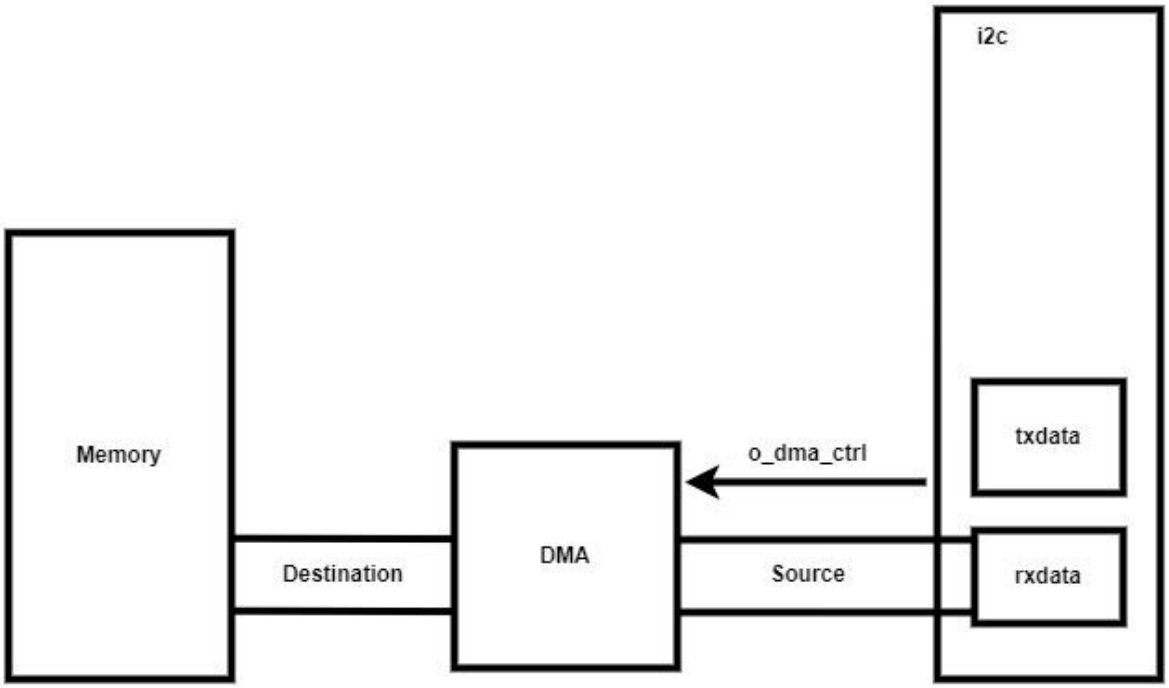


Figure 20- 10 : I2C read transaction with DMA setup

21 TIMER — Timer/counter

21.1 Functional description

- 4 identical timeout values (timer 0/1/2/3) with level-triggered interrupt to CPU
- Timer 0/1/2/3 can be configured in either one-shot or free-running mode
- Timer has a prescaler to generate counting ticks in configurable frequency
- It has start, clear and pause control signals configured through the register
- Timer also uses clock select to use 1Mhz clock signal to reduce power consumption
- The input clock and the pclk for the APB bus are synchronized hence simplifying the design

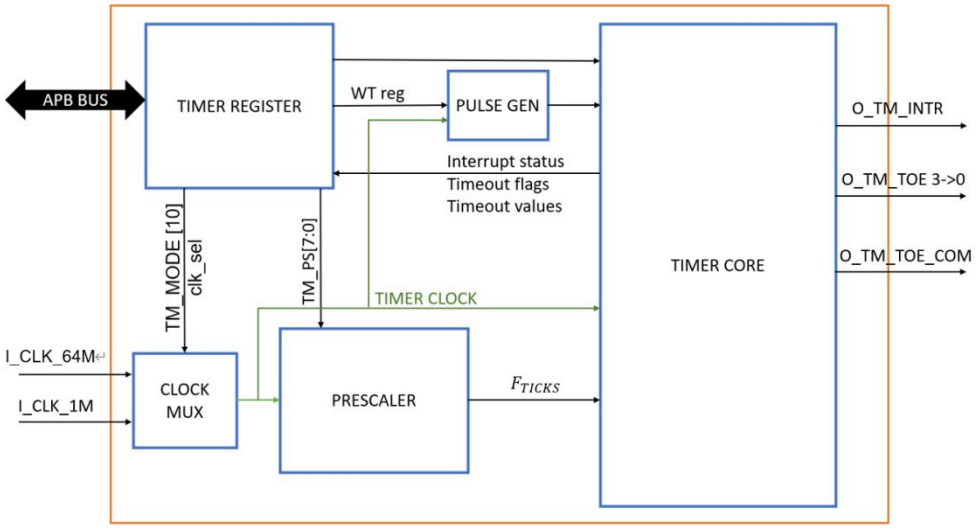


Figure 21-1: Block diagram of timer module

21.1.1 Timer prescaler

The prescaler generates the ticks based on which the counter counts. It counts at input clock frequency and generates a one cycle tick signal based on the prescaler value for the counter core to count. The frequency of the tick is configurable through the register TM_PS. It is implemented through a down-counter logic. When the counter reaches 0 it generates a tick.

If f_{tick} <= 1Mhz then we use the 1Mhz clock signal instead of the 64 Mhz. This is implemented to reduce power consumption. The structure of the prescaler is as shown below in *Figure 21-2*. The tick is calculated as

$$f_{tick} = \frac{64Mhz}{(n+1)}$$

where, n is the prescaler value [7:0] written in the TM_PS register.

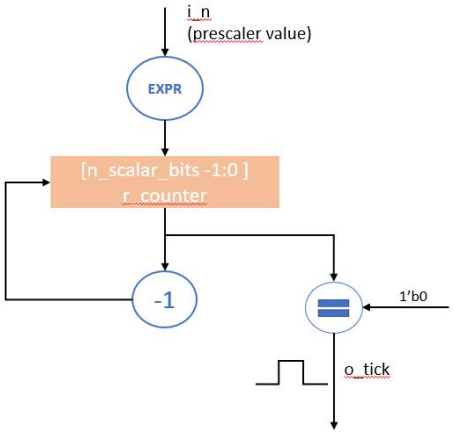


Figure 21-2: Structure of prescaler

21.1.2 Timer core

The timer core contains the main counter module. It receives the tick from the prescaler and the control signals from the register. Once the start signal is asserted along with the enable signal the counter starts to count based on the ticks generated by the prescaler. If the pause signal is asserted, the timer stops and starts from the previous value when the pause signal is disserted again. The counter is cleared if the timer clear register is written or the start signal is asserted again.

The timer counts against 4 time-out values. It compares to find the maximum timeout value and raises the timeout flag signals for each value if the timeout enable signals are asserted. If the interrupt signal is asserted and the count is equal to any of the 4 timeout values, then an interrupt flag is raised. This interrupt signal & timeout signals are cleared using the toe_int_clr and toe [0->3] clr fields of TM_INT_CLR register which generates an interrupt clear and timeout clear signal respectively.

Another feature of the timer is the time value latch register signals are used to write the current counter value as the timeout values in the timeout value registers using the timeout latch pulse generated by TM_TVL register for each timeout value.

The timer operates in two modes.

1. One-shot mode
2. Free-running mode

In one-shot mode, when the counter reaches the highest value of the 4 timeout values, it stops the counting. Whereas in the free-running mode, the counter overflows when it reaches the highest value among the four timeout values and continues counting till the enable signal is disserted.

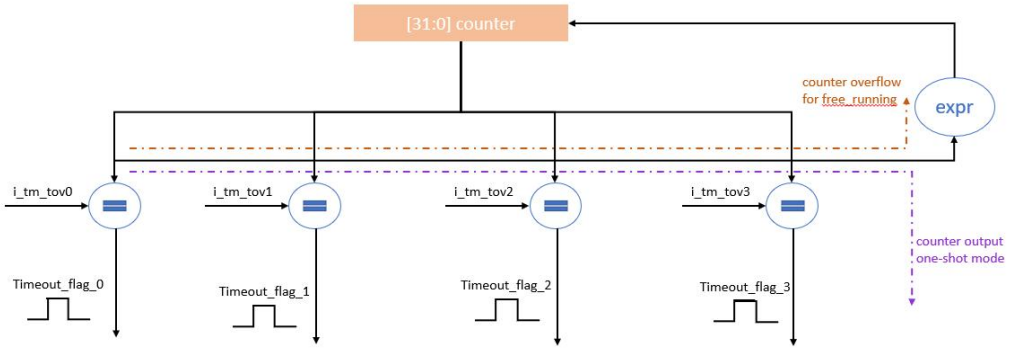


Figure 21-3: Timer core structure

21.1.3 Max value of 4

This module is used to compare the 4 timeout values and determine the maximum value for timeout. In one shot mode, when this maximum value is reached, then the counter stops counting. In free-running mode, when the counter reaches the maximum timeout value it overflows and counts again till the counter is cleared/ enable is disserted.

The implementation involves comparing data 1 & 2 and computing the maximum value between the two. The same is repeated for data 3 & 4. Then the maximum value from both data sets is then compared to calculate the maximum which is sent to the timer_core module.

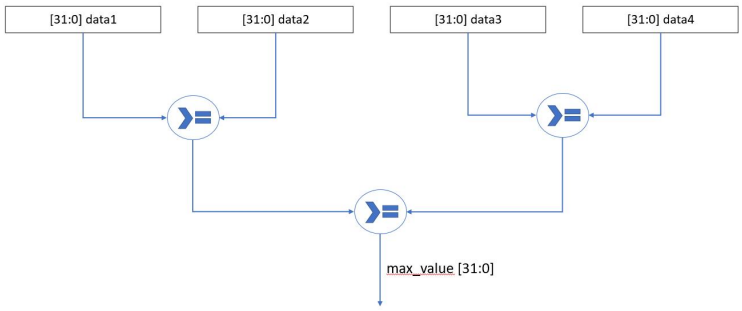


Figure 21-4: Structure of max_value4 module

21.2 Registers

Table 21-1: Register list of Timer

Offset	Reg	Range	Field	Access	Reset Value	Description
0x0	TM_EN	[0]	en	RW	1'b0	enable/disable timer
0x4	TM_START	[0]	tm_start	WT	1'b1	Timer start/restart register =0: no action =1: write 1 to start timer

0x8	TM_CTRL	[0]	tm_pause	RW	1'b0	pause timer. =1: pause timer =0: resume timer
		[1]	tm_clear	WT	1'b1	Timer clear and stop. =1: write 1 to set register tm_clear. The high pulse of tm_clear is used to clear the timer value. =0: no action if writes 0
0xc	TM_TVL	[0]	tm_tvl0	WP	1'b1	current time value latching =1: write 1 to latch the current time value to TM_TOV0 registers =0: no action
		[1]	tm_tvl1	WP	1'b1	current time value latching =1: write 1 to latch the current time value to TM_TOV1 registers =0: no action
		[2]	tm_tvl2	WP	1'b1	current time value latching =1: write 1 to latch the current time value to TM_TOV2 registers =0: no action
		[3]	tm_tvl3	WP	1'b1	current time value latching =1: write 1 to latch the current time value to TM_TOV3 registers =0: no action
0x10	TM_INT_EN	[0]	toe0_int_en	RW	1'b1	timeout 0 interrupt enable =1: enable interrupt for timeout event 0 =0: disable interrupt for timeout event 0
		[1]	toe1_int_en	RW	1'b1	timeout 1 interrupt enable =1: enable interrupt for timeout event 1 =0: disable interrupt for timeout event 1
		[2]	toe2_int_en	RW	1'b1	timeout 2 interrupt enable =1: enable interrupt for timeout event 2 =0: disable interrupt for timeout event 2
		[3]	toe3_int_en	RW	1'b1	timeout 3 interrupt enable =1: enable interrupt for timeout event 3 =0: disable interrupt for timeout event 3
0x14	TM_INT_CLR	[0]	toe_int_clr	WT	1'b1	Clear timeout interrupt =1: write 1 to clear timeout interrupt
		[1]	toe0_clr	WT	1'b1	clear timeout event 0
		[2]	toe1_clr	WT	1'b1	clear timeout event 1
		[3]	toe2_clr	WT	1'b1	clear timeout event 2
		[4]	toe3_clr	WT	1'b1	clear timeout event 3
0x18	TM_INT_ST	[0]	int	RO	-	Interrupt status Each timeout event can assert the interrupt signal if it's interrupt is enabled
0x1c	TM_PS	[7:0]	ps_value	WL	8'd0	Prescaler register, =n: it will generate ticks in frequency F_{timer} = 32M/(n+1)
0x20	TM_MODE	[0]	run_type	WL	1'b0	=0: one shot =1: free running
		[1]	evt_type	WL	1'b0	Timeout event clear method in free running mode =0: timeout event is a pulse signal =1: timeout event will be auto clear if another timeout event occurs
		[7:2]	evt_width	WL	6'd1	pulse width =n: number cycles of the pulse width

		[9:8]	evt_com	WL	2'd0	Timeout event common type =0: tm_toe_com = tm_toe0 | tm_toe1 =1: tm_toe_com = tm_toe0 | tm_toe1 | tm_toe2 =2: tm_toe_com = tm_toe0 | tm_toe1 | tm_to2 | tm_to3
		[10]	clk_sel	RW	1'b0	time clock source selection =0: apb clock =1: apb clock frequency / 32. It can use for low power purpose
		[11]	auto_clear	RW	1'b0	=0: counter will not be clear after timeout =1: counter will be clear once it reaches the maximum timeout value in TM_TOV0,1,2,3.
0x24	TM_TO_EN	[0]	tov0_en	RW	1'b0	Timeout event 0 enable =0: disable timeout 0 =1: enable timeout 0 If timeout n is enabled, the timeout event will be generated when the counter value is equal to the timeout value n. Otherwise, this register can be used to latch the current counter value.
		[1]	tov1_en	RW	1'b0	Timeout event 1 enable
		[2]	tov2_en	RW	1'b0	Timeout event 2 enable
		[3]	tov3_en	RW	1'b0	Timeout event 3 enable
		[4]	tovcom_en	RW	1'b0	Timeout common enable =1: enable timeout common =0: disable timeout common
0x28	TM_TOV0	[31:0]	tm_tov0	WS	32'd0	Timeout value 0
0x2c	TM_TOV1	[31:0]	tm_tov1	WS	32'd0	Timeout value 1
0x30	TM_TOV2	[31:0]	tm_tov2	WS	32'd0	Timeout value 2
0x34	TM_TOV3	[31:0]	tm_tov3	WS	32'd0	Timeout value 3
0x38	TM_TOE	[0]	tm_toe0	RO	-	timeout event 0
		[1]	tm_toe1	RO	-	timeout event 1
		[2]	tm_toe2	RO	-	timeout event 2
		[3]	tm_toe3	RO	-	timeout event 3
		[4]	tm_toe_com	RO	-	timeout common event tm_toe_com = tm_toe0 | tm_toe1 | tm_to2 | tm_to3 or tm_toe0 | tm_toe1 or tm_toe0 | tm_toe1 | tm_toe2

Please refer to Table 25-1 Explanation of Access in register list for “Access” definition

21.2.1 TM_CTRL

Write bit 1 (tm_clear) to set register tm_clear to clear the timer value. When the maximum timeout value is reached, the counter will be automatically cleared. The auto-clear is used to clear the counter when it reaches the maximum timeout value in TM_TOV*

21.2.2 TM_TVL

Write 1 to latch the current time value into the TM_TOV register. This register is designed as 4 bits, each bit latches the current value into the corresponding TM_TOV0, TM_TOV1, TM_TOV2, TM_TOV3 register.

21.2.3 TM_INT_EN

This register is designed as 4 bits, corresponding to timeout 0, 1, 2 and 3 to enable interrupts.

21.2.4 TM_INT_CLR

This register is designed as 5 bits, of which only bit 0 is used to clear the timeout interrupt, and the rest correspond to clear timeout events 0, 1, 2, and 3 respectively.

21.3 Application flow

21.3.1 Timeout event

Noted: Each timeout event can assert the interrupt signal if it's interrupt is enabled. Thus, if there is more than one interrupt enable, then the 1st interrupt needs to be clear before next another interrupt check is coming.

The diagram illustrates the behavior of the timer interrupt system involving two timeout events, Timeout 0 and Timeout 1. Timeout 0 and Timeout 1 are enabled. The counter increments and once it reaches the predefined Timeout 0 value, the interrupt is triggered. The CPU waits for the interrupt and sends a clear signal which resets the interrupt state. The counter continues counting and when it reaches Timeout 1, the interrupt is triggered again. The process of counting, interrupt triggering, and interrupt clearing continues for each timeout event and the system transitions between the timeout states as the counter reaches the corresponding timeout value.

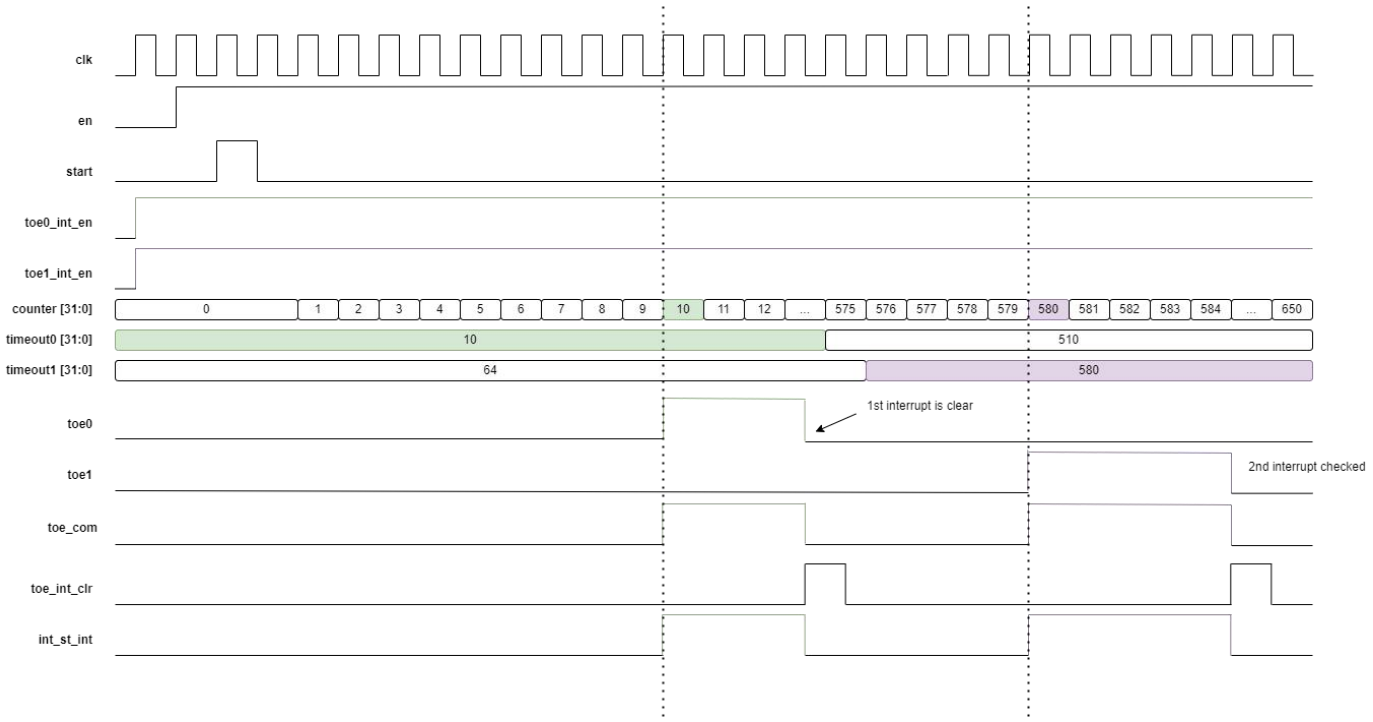


Figure 21-5: Behaviour of interrupt system with two timeout events

21.3.2 PWM signal generation

Noted: The timer generates PWM by configuring two timeout values to generate timeout events. The relationship between the two events is that when timeout event 1 is high, timeout event 0 will be cleared, and vice versa. The pulse width can be adjusted by controlling the timeout value of event 1, which must occur after timeout event 0. The width of the next timeout event 0 will be generated shorter or longer, depending on the behavior of timeout event 1.

The I/O of the timer (timeout 0/1/2/3 and common event) is configurable to available GPIOs via the IOMUX.

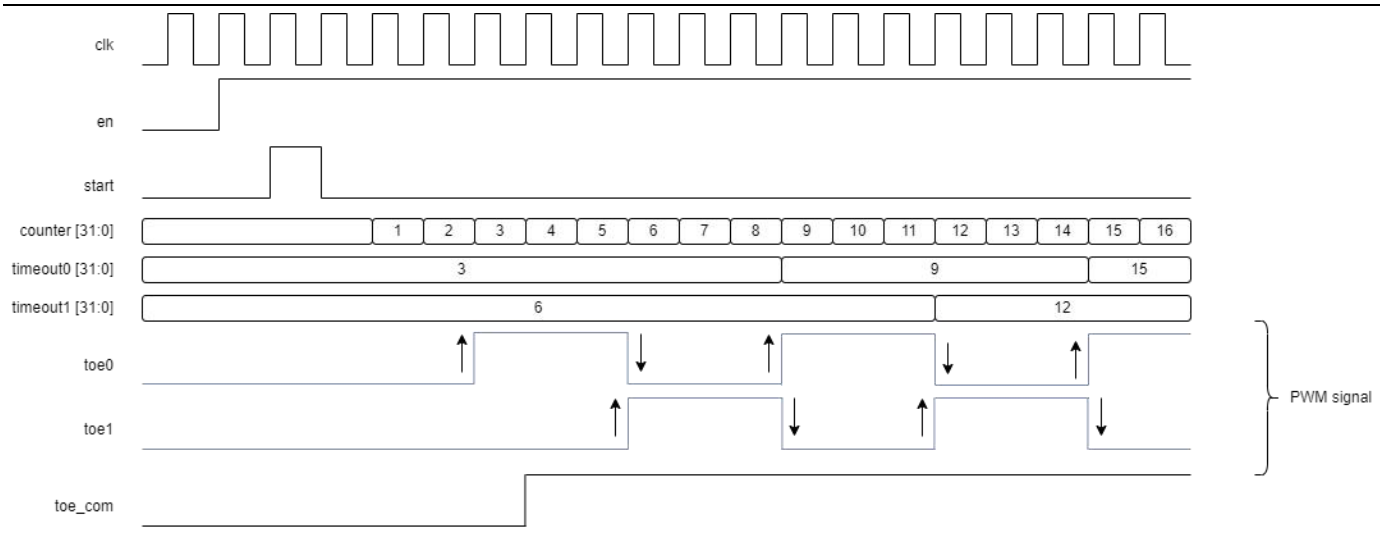


Figure 21- 6: PWM signal generation

22 WDT — Watchdog timer

22.1 Function description

22.1.1 Watchdog Timer (WDT) Protocol

The Watchdog Timer (WDT) is designed to protect against system lock-ups by utilizing a down counter. The WDT starts when a value of 0xAC624A73 is written to CSR reg, and bit WDT_CTRL[0] is set to 1'b1 (enable). The counter value (already predefined in TOV reg) starts down counting to 0. Once the counter reaches 0, an interrupt signal is issued. Additionally, system reset signal is generated after the interrupt depending on the configuration (CFG) register. The WDT operates on a 32 kHz clock sourced from an RC oscillator, and can be disabled by writing 1'b0 to the corresponding bit in the control register.

The WDT can be paused when the CPU enters sleep mode or is halted by a debugger. However, the timer can be configured to continue counting in these conditions through settings in the configuration (CFG) register.

22.1.2 State Machine (wdt_fsm)

The function of the counter is defined as state machine.

- **IDLE** – The timer enters this state only if the i_wdt_en = 1 i.e., only when the WDT_CTRL [0] reg is written bit 1. In this state, counter values are reset to 0.
- **LD_CNT** – This state is achieved only if i_wdt_cnt_start = 1 i.e., only when the value 0xAC624A73 is written to the WDT_CSR reg. In this state, the counter value which is defined in WDT_TOV reg is loaded to the counter (r_counter).
- **DCOUNT** – In this state, the counter (r_counter) is decremented till 0. If the CPU sleep mode or debug halt signals are issued then, the counter is either active or inactive depending on the configuration of the timer which is set by the WDT_CFG register.
- **RST_GEN** – This state is achieved when the counter (r_counter) reaches 0 and i_cfg_rst_en = 1 (set by WDT_CFG reg). In this state, the reset counter value defined by the WDT_CFG is decremented. When the counter reaches 0, then system reset signal is generated and goes to IDLE state.

When the timer counter (r_counter) reaches 0, an interrupt signal is generated. When int_clear signal is issued then the interrupt is cleared. The interrupt signal is only given to the output when int_en = 1 (set by WDT_IER reg).

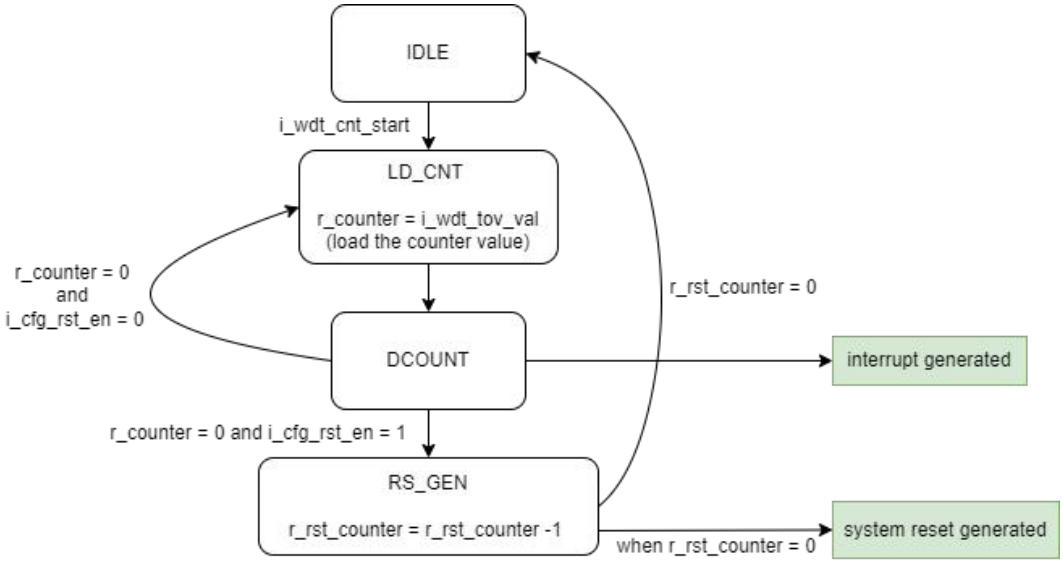


Figure 22-1 : Structure of the FSM

22.1.3 Configuration of WDT

There are few configurations for the WDT module being:

- ***WDT in sleep status***
  WDT will not decrement its counter while the CPU is in sleep mode. It can either continue to operate or be paused, depending on the configuration setting WDT_CFG.RUN_IN_SLEEP.
- ***WDT in debugger mode***
  WDT is typically paused during debugging to prevent unwanted resets while the developer analyzes the system. However, it can be configured using WDT_CFG.RUN_IN_HALT to remain active if necessary.
- ***Interrupt generated***
  WDT is designed to generate an interrupt signal when the counter reaches zero while the state machine is in the P_DCOUNT state. The design includes a mechanism for interrupt clearing, interrupt can be reset based on the signal WDT_ICR.INT_CLEAR.
- ***System reset generated***
  WDT can trigger a system reset if the WDT_CFG.RST_EN configuration bit is set. Initially, the WDT generates an interrupt, and if this interrupt is not cleared before a second timeout occurs, the WDT will initiate a system reset. When WDT_CFG.RST_EN is asserted, the FSM transitions to the P_RST_GEN state, where the system reset counter is loaded with a predefined timeout value and begins counting down. During the countdown, the system reset signal remains high. Once the reset counter reaches zero, the system reset signal is driven low, triggering a system reset.

22.2 Registers

Table 22-1: Register list for WDT module

Offset	Reg	Range	Field	Access	Reset Value	Note
0x0	WDT_CTRL	[0]	enable	RW	1'b1	enable the watch dog timer =0: disable wdt. It will stop wdt counter =1: enable wdt. Still need write WDT_CRR to start counter
		[1]	lock	RW	1'b0	config register is locked =0: can write WDT_TOV and WDT_CFG =1: WDT_TOV and WDT_CFG are locked and cannot overwrite
0x4	WDT_CSR	[0]	start	WT	1'b0	counter start/restart register. =0x1： Write the value 1 to toggle the start register. The edge of start register is used to restart counter when WDT_CTRL[0]=1, also clear interrupt within 2 clk_wdt cycles =0: no action if writes 0

						This address is not readable
0x8	WDT_IER	[0]	int_en	RW	1'b1	interrupt enable/disable register =0: disable interrupt =1: enable interrupt when interrupt is disabled, the interrupt signal is inactive.
0x0c	WDT_ICR	[0]	int_clear	WT	1'b0	Interrupt clear register =1: clear interrupt. Write 1 to generate a pulse signal to reset interrupt. =0: no action This register is not readable
0x10	WDT_SR	[0]	running	RO	-	running status register =1: counter is running
		[1]	interrupt	RO	-	interrupt status =1: interrupt signal issued. =0: no interrupt
0x14	WDT_CVR	[31:0]	cnt_value	RO	-	counter value register
0x18	WDT_TOV	[31:0]	to_val	WL	32'hFFFF	timeout value. i.e., to_val+2 is the number of clk_wdt cycles
0x1c	WDT_CFG	[0]	rst_en	WL	1'b0	Enable to generate reset signal 0: generate an interrupt only 1: generate an interrupt first, if not cleared by the time a second timeout occurs, then generate a system reset
		[13:1]	rst_to_val	RW	13'h0	reset timeout period in clk_wdt cycles. =n: n clk_wdt cycles after interrupt, reset is generated in mode=1.
		[14]	run_in_sleep	RW	1'b0	wdt in sleep status. =0: puase wdt while cpu is sleeping =1: keep running while cpu is sleeping
		[15]	run_in_halt	RW	1'b0	wdt in debugger mode =0: pause wdt while cpu is halted by the debugger =1: keep running while cpu is halted by the debugger

Please refer to Table 25-1 Explanation of Access in register list for “Access” definition

23 BLE subsystem

The BLE sub system consists of BLE PHY and BLE controller.

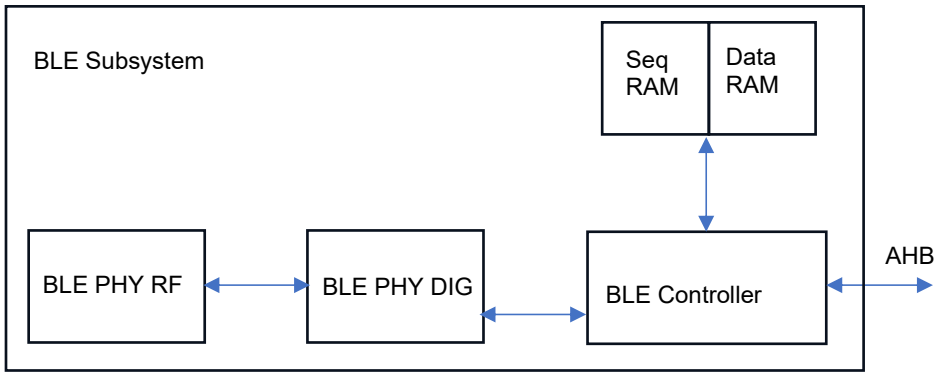


Figure 23-1: BLE COMBO interface block diagram

23.1 BLE PHY

The BLE PHY RF front-end includes a RF transceiver composed of a frequency synthesizer, PA and receiver. The frequency synthesizer shares reference clock with UWB. TRSW and matching for TX and RX of the BLE are also implemented on chip.

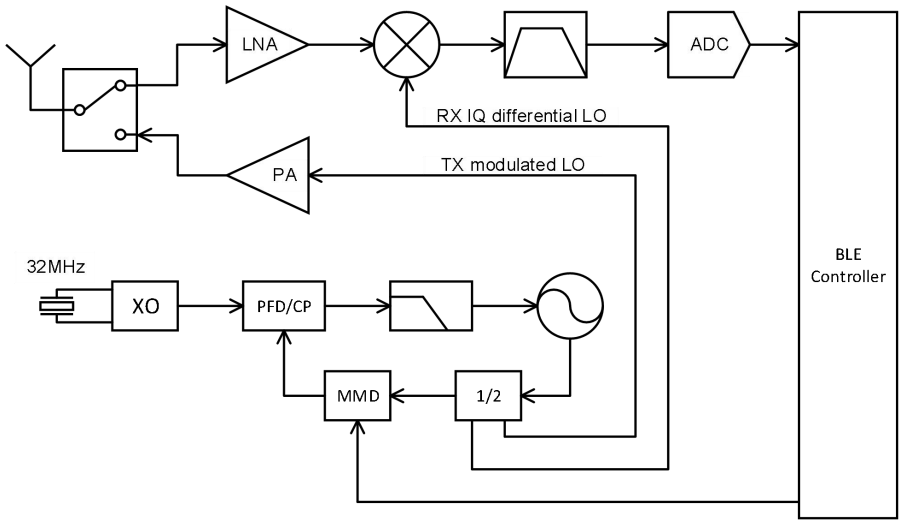


Figure 23- 2: BLE PHY Functional Block Diagram

The BLE50 PHY implements the radio transceiver that supports Bluetooth Low-Energy and IEEE 802.15.4 standards.

All blocks within the PHY hard IP use the power supply provided by the integrated Low Drop-Out (LDO) regulators, and are optimized for ultra-low power consumption and small area. Performance is always optimized over the operational temperature range and supply voltage, using automatic calibrations.

1. Low Noise Amplifier (LNA) - The input frequency range for the LNA is 2.40 - 2.484 GHz with two programmable gains. The Automatic Gain Control (AGC) algorithm controls the gain of the LNA, the mixer and the complex filter. The receiver can handle signal levels from -96 dBm below Bluetooth LE requirements, up to 0 dBm with optimal performance.
2. I-Q Mixers - The receive path uses a low intermediate frequency (IF) architecture with a mixer, converting the input signal from 2.4 GHz, to an IF at 1 MHz The output low IF frequency can be programmed to 2 MHz to support 2-Mbps operation.
3. Baseband Filter - After the exiting the mixer the signal enters the baseband filter. The filter is programmable to support 1 MHz and 2 MHz IF frequencies.
4. Analog-to-Digital Converter (ADC) - The ADC quantizes the IF signal into 10 bits. The output of the ADC is sent to the soft IP for signal filtration, demodulation, and evaluation of the received signal strength indicator (RSSI) value.
5. Frequency Synthesizer - The frequency synthesizer is a Sigma-Delta (\u03a3\u0394) Fractional-N PLL with a differential LC VCO. The transmit modulation is applied directly to the loop, and the VCO output is sent to the PA for transmission. In receive mode, the VCO output is divided by two to generate I and Q signals at 2.4 GHz. The I and Q signals are sent to the RX mixers.
6. Power Amplifier (PA) - The class-E power amplifier (PA) can be programmed to transmit output levels from -20 dBm through 0 dBm. The PA includes a mode to generate a 6-dBm transmit output level, using an external 1.5-V power supply. The PA efficiency is optimized for the nominal output power of 0 dBm. At start and end of packets, the PA output is ramped-up and down to prevent unwanted spectral splatter.
7. Calibration Machines - The IP includes fast foreground and background automatic self-calibration algorithms to ensure optimal performance and minimum testing time. The automatic calibration engine includes:
    a. Automatic Frequency Correction (AFC)
    b. VCO auto-tuning
    c. Automatic RX filter bandwidth calibration and tuning
    d. PLL bandwidth calibration

e. RX offset calibration algorithms

8. Crystal Oscillator (XO) - The 32-MHz crystal oscillator (XO) reduces the SoC bill of materials. It uses a single crystal for the whole system. The start-up time of the crystal oscillator is less than 100μs. This clock is available for upper layers (Bluetooth Low Energy Link Layer or 802.15.4 MAC) and for system clocking, to support synchronous system operation, and to minimize possible spurs generation (due to clocks coupling through the substrate).
9. Low Drop-Out (LDO) Bank - Six LDOs provide the circuit islands with optimum isolation. They accept an input supply range from 1.10 V through 3.6 V. The BLE50 Combo PHY includes the following LDOs:
    a. RFFE
    b. VCO
    c. FSYN
    d. ANA
    e. ADC

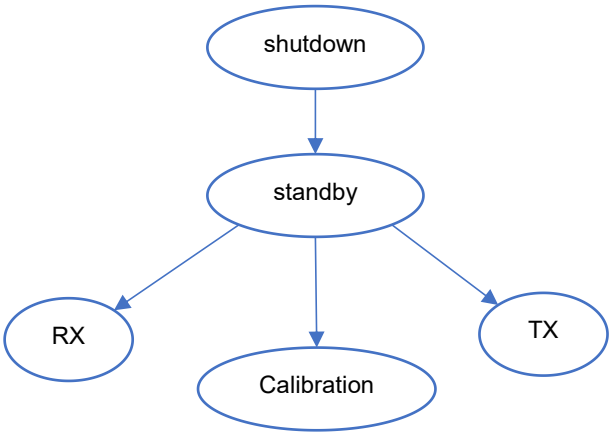


Figure 23- 3: BLE PHY COMBO operation state

Table 23- 1: Block status for different Operating mode

Model	Operating mode		
	Shutdown	Standby	TX/RX
Digital Core Supply System (at SoC)	Disabled	ON	ON
System LDOs	OFF	Disabled	ON
SPI Interface	OFF	ON	ON
Register Retention	No	Full	Full
Digital Core	OFF	ON	ON
32-Mhz Crystal Oscillator	OFF	ON	ON
Radio & Baseband Cores	OFF	Disabled	ON

23.2 BLE Controller

The BLE controller provides the Link Layer (LL) function that complies with the Bluetooth Core specification, version 5.0. It is designed to interface with the BLE PHY to provide a complete solution for a BLE subsystem. The BLE packet format in *Figure 23-5* is supported.

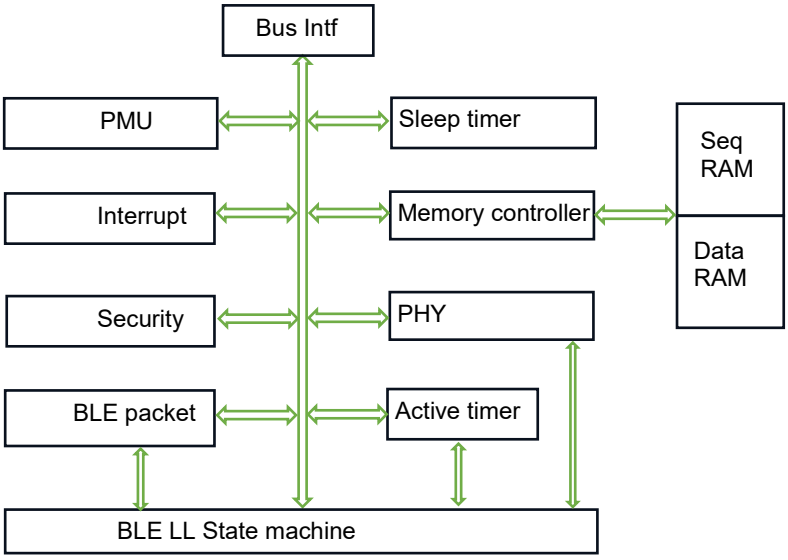


Figure 23-4: Block Diagram of BLE controller

The BLE controller supports the following features:

- Complies with Bluetooth Core Specification, Version 5
  - Device privacy and network privacy modes
  - Advertising extension PDUs
  - Anonymous device address type
  - Up to 2 Mbps data rate
  - Long range
  - High-duty cycle, non-connectable advertising
  - Channel selection algorithm #2
  - High output power
- Complies with Bluetooth Core Specification, Version 5.1
  - Angle of arrival (AoA) and Angle of Departure (AoD)
  - Advertising Channel Index
  - Periodic Advertising Sync Transfer (PAST)
  - All roles specified by Bluetooth specifications
  - Broadcaster
  - Observer
  - Peripheral
  - Central
- Up to 20 connections in any role in addition to Advertiser/Scanner roles
- Compile/configuration options to reduce size based on intended application
  - Master and Slave role
  - Advertiser role
  - Scanner role
- Device filtering through programmable size white lists

- Direct test mode
- Complies with IEEE 802.15.4-2015 in the following
  - Data rate of 250 kbps
  - Chip rate of 2 Mbps
  - CSMA-CA channel access
  - Low power consumption
  - 16 channels in the 2.4GHz ISM band
  - Beacon management
  - Two addressing modes, 16-bit short and 64-bit IEEE addressing
  - PAN formation along with association and disassociation
  - Fully handshake protocol for transfer reliability; frame validation and acknowledged frame delivery
- Highly optimized power, area, and memory footprint
- Link Layer functions distributed selectively across RTL and firmware
- Protocol timing is handled in a way that allows the SoC system processor to work at relatively low clock frequencies and to reduce processor average required run time.
- Product upgrade capability through firmware updates
- Operating system abstraction layer for support of multiple operating systems, including bare metal
- AMBA3 AHB-Lite interface to SoC system processor
  - Data and register programming interface
  - Asynchronous interface supports system processor, operating at a different frequency than Link Layer

Figure 23- 5: Uncoded phy packet format

24 UWB subsystem

The UWB module is integrated in the chip to realize an IEEE 820.15.4a/z standard compliant transceiver, which supports the following features:

- Compliant with IEEE 802.15.4z, IEEE 802.15.4-2015, and FiRa standards.
- Support band group two from 6 to 10 GHz, i.e., channel 5, 6, 8, 9, 10, 12, 13, 14.
- Support packet length up to 4095 bytes
- Support various data rates from 850Kbps, 6.8Mbps, 7.8Mbps, 27.2Mbps, and 31.2Mbps
- Support STS packet configuration 0, 1 & 3, with integrated AES-128 for secure ranging.
- Support two-way ranging (TWR), and time difference of arrival (TDOA)
- Support 3D angle of arrivals (AOA) with 1 Tx and 3 Rx.

The structure of UWB module is shown in *Figure 24- 1*.

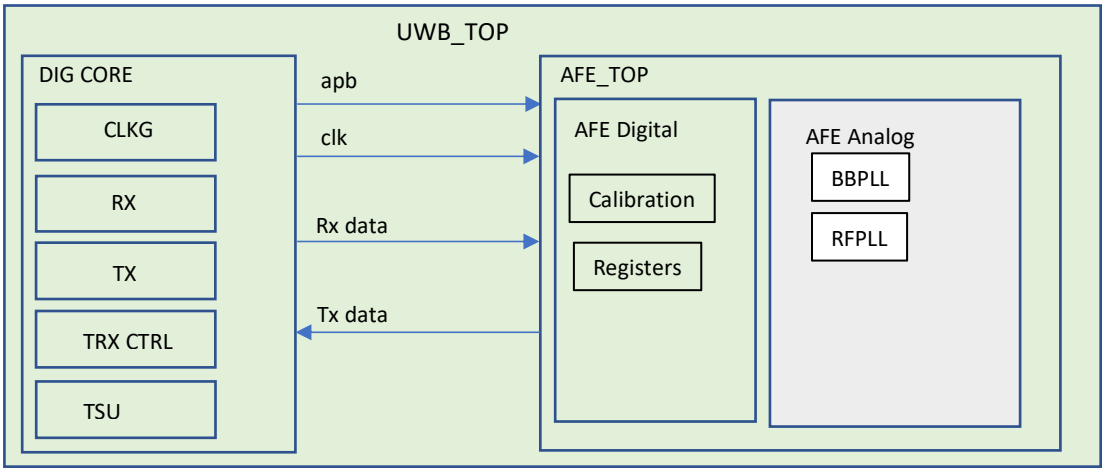


Figure 24-1: Block diagram of UWB module

The clock generation module CLKG is used to generate clocks used by UWB, which includes 125 MHz for TX and RX, 250 MHz for RX and timestamp sampling.

The TX module is used to fetch the data from the system memory, encode the data and then transmit to UWB analog front end (AFE) module.

The RX module is used to process the received data from AFE, which include digital frond end (DFE), timing and frequency synchronization, channel correction, and packet decoding. The final bitstream is saved in system memory via AHB interface.

The TRX control unit is used as an intermediate control stage (responsible for inter-packet timing, most of the packet Rx/Tx time-outs and Rx/Tx dataflow management). It contains a set of timers to generate hardware related IRQs on one side and enable/reset for Tx/Rx hardware on the other side.

The timestamp sampling unit TSU is used to record the timestamp of TX and RX events, which is further used for ranging and position purpose.

The module AFE DIG is used to interface the DIG CORE and AFE. It also contains registers for the configuration of TX and RX channels in AFE. The ADC calibration circuits are included for high performance ADC calibration.

25 Appendix

25.1 Explanation of Aceess type in Register table

Table 25-1 Explanation of Access in register list

Access	ResetValue	Note
RW		Read and write
WP		Write a pulse. Register will automatically reset to 0.
CONST		constant
WT	value_to_write	Writing a value to toggle the register to indicate the write.
WE		write external register. No register implemented in the module. Will generate two outputs: o_regname_wr and o_regname_wdata
WL		write lock. Write if register is not locked. Will generate an input: _lock that is used to lock the write operation
WS		Write and set. Register can be written via APB and also be set directly from external. Will generate 2 inputs: i_regname_set and i_regname_sdata

RO		Read only. Value to be read directly input from external. Will generate a input signal i_regname
RE		Read external register. A read signal is output to indicate the read operation: o_regname_rd

26 Reference

[1] ARM (no date) *TrustZone technology for Armv8-M Architecture Version 2.1*, *Documentation – arm developer*. Available at: https://developer.arm.com/documentation/100690/0201/Switching-between-Secure-and-Non-secure-state (Accessed: 30 June 2023).

27 Version history

Data	Version	Description
2025	1.0	Initial version

28 Disclaimers

CHIPSBANK INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. CHIPSBANK MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. CHIPSBANK AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". CHIPSBANK AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

CHIPSBANK DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF CHIPSBANK COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM CHIPSBANK UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF CHIPSBANK.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE CHIPSBANK PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. CHIPSBANK RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. CHIPSBANK MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES CHIPSBANK ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. “TYPICAL” PARAMETERS WHICH MAY BE PROVIDED IN CHIPSBANK DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING “TYPICALS” MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER’S TECHNICAL EXPERTS. CHIPSBANK DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. CHIPSBANK PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE CHIPSBANK PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE CHIPSBANK PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD CHIPSBANK AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT CHIPSBANK WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.