T32CM11

Zigbee 2.4-GHz RF SoC

Introduction

The T32CM11 is an ultra-low power, high-performance ARM® Cortex®-M3 based Zigbee 3.0 RF SoC with multi-protocol Bluetooth Low Energy 5.3, Thread/Matter, and proprietary 2.4G networking stack compatibility to facilitate home & building automation, smart lighting, smart locks, sensor network applications, etc. A comprehensive mix of analog and digital peripherals are integrated with the 2.4GHz RF transceiver which is compliant with Zigbee 3.0 and IEEE® 802.15.4 requirements. Ultra-low current consumption is achieved in the RF receive & transmit modes and power-down modes to support the latest IPv6-based IoT applications.

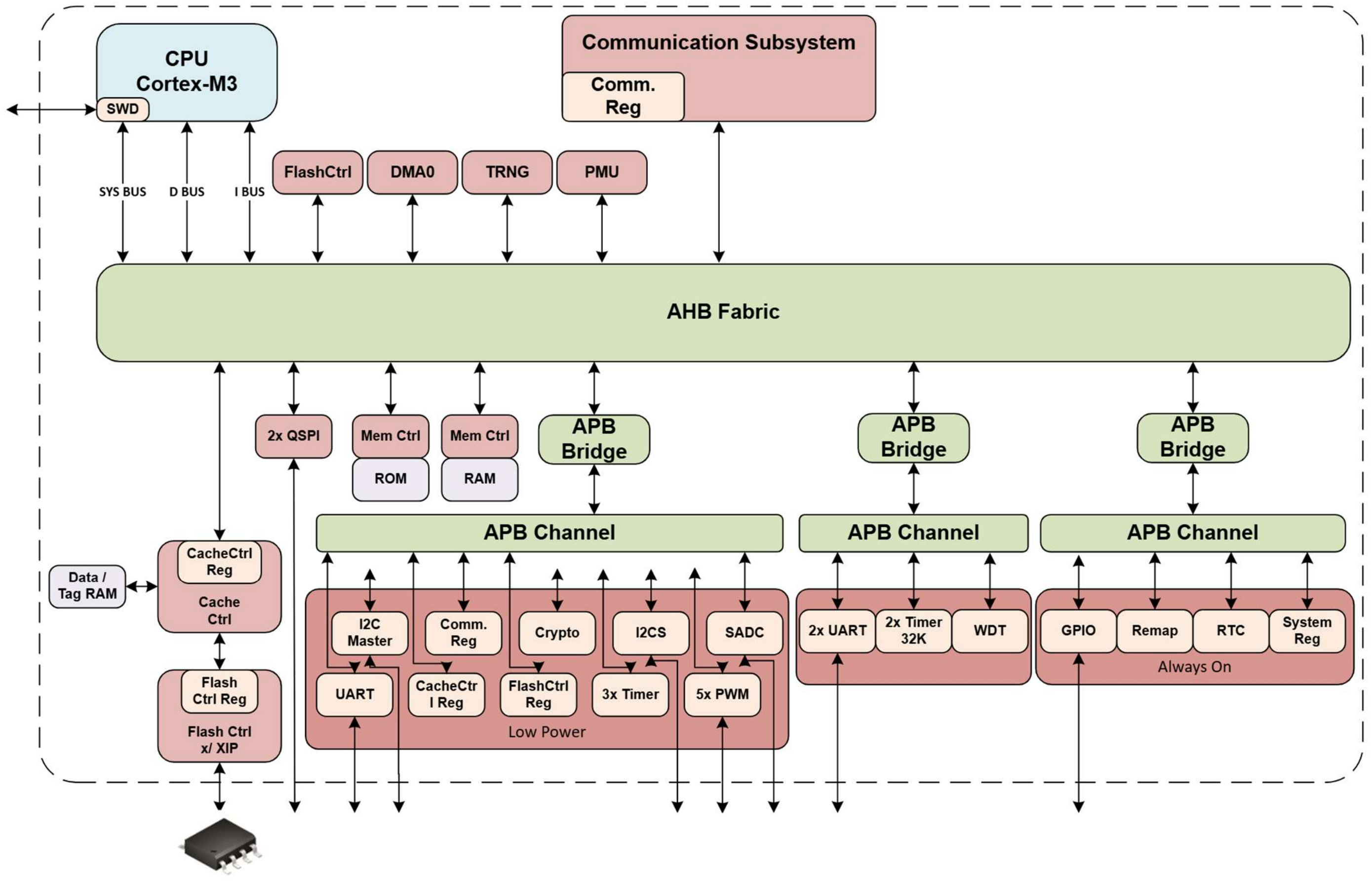


Figure 1 T32CM11 Block Diagram

Table of Contents

1 Feature List

- Transmit Power
  - Zigbee/Thread: + 10dBm with <2% EVM
  - Bluetooth Low Energy: + 10dBm
- Receive Sensitivity
  - Zigbee/Thread: -100dBm
  - Bluetooth Low Energy: -104dBm @125kbps
- Supported Modulations
  - multi-rate FSK, GFSK
  - DSSS-OQPSK
- Data Rates
  - Zigbee/Thread: 250Kbps
  - Bluetooth Low Energy: 125K/500K/1M/2Mbps
- Multi-protocol support: Bluetooth® Low Energy 5.3 & IEEE 802.15.4, Zigbee 3.0
- Supports Thread/Matter applications with Bluetooth Low Energy commissioning
- RF offset cancellation loop
- Frequency hopping capability
- Programmable channel filter bandwidth
- External 32MHz high frequency crystal & internal low frequency RC Oscillator
- + 1.8 ~ + 3.6VDC Operating Voltage
  - POR (power-on reset) & UVLO (under voltage-lockout)
  - Power Management Unit for power state switching
  - Integrated DC-DC converter and LDO regulator
- Various Power Saving Modes including
  - Sleep1: 5.2 µA
- Embedded ARM® Cortex-M3 CPU, internal MCU for MAC
  - ARM® CPU speed: 32M/48M/64MHz
  - SWD (2-pin Serial Wire Debug)
  - 32KB Boot ROM
  - 208KB SRAM (CPU+MCU)
  - 2048 KB Flash
- Multiple integrated peripherals
  - Integrated security: SHA/AES/ECC
  - DMA controller
  - UART interface x3 (1 with CTS/RTS, 2 without CTS/RTS)
  - I^2S interface
  - I^2C master interface
  - SPI interface x2
  - PWM x 5
  - 32-bit Timer x5
  - 32-bit Real Time Clock Timer x2
  - Temperature Sensor
  - 4CH 12-bit 350Ksps ADC
  - GPIO (w/interrupt) x22
  - WiFi Co-existence interface
  - Watchdog x1
- Lead-Free 5x5 40-pin Quad Flat No-lead (QFN) package
- Halogen-free / RoHS 2.0 / Reach Annex 14 & 17
- Operating Temperature: - 40 ºC ~ + 85/105 ºC
- ESD: HBM 2KV / MM 200V, Latch-up: 150M

2 CPU

The Cortex™-M3 processor is a high performance 32-bit processor designed for the microcontroller market. It offers significant benefits to developers, including:

- outstanding processing performance combined with fast interrupt handling
- enhanced system debug with extensive breakpoint and trace capabilities
- efficient processor core, system and memories
- ultra-low power consumption with integrated sleep modes
- platform security robustness with integrated memory protection unit (MPU).

The Cortex-M3 processor is built on a high-performance processor core with a 3-stage pipeline Harvard architecture making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and extensively optimized design. It provides high-end processing hardware including a range of single-cycle & SIMD multiplication and multiply-with-accumulate capabilities, saturating arithmetic and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M3 processor implements tightly-coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M3 processor implements a version of the Thumb® instruction set based on Thumb-2 technology ensuring high code density and reduced program memory requirements. The Cortex-M3 instruction set provides the exceptional performance expected of a modern 32-bit architecture with the high code density of 8-bit & 16-bit microcontrollers.

The Cortex-M3 processor closely integrates a configurable nested vector interrupt controller (NVIC), to deliver industry-leading interrupt performance. The NVIC includes a non-maskable interrupt (NMI) with interrupt priority levels. The tight integration of the processor core and NVIC provides fast execution of interrupt service routines (ISRs) dramatically reducing the interrupt latency. This is achieved through the hardware stacking of registers and the ability to suspend load-multiple & store-multiple operations. Interrupt handlers do not require wrapping in assembly code thus removing code overhead from the

ISRs. Tail-chain optimization also significantly reduces the overhead when switching from one ISR to another.

3 CPU Module Configuration Supported

Core Option	Description	Implementation
NVIC	Nested Vector Interrupt Controller	33 vectors
PRIORITIES	Priority bits	3
WIC	Wakeup Interrupt Controller	YES
Endianness	Memory system endianness	Little endian
Bit Banding	Bit banded memory	YES
DWT	Data Watchpoint and Trace	YES
SysTick	System tick timer	YES

Module	Description	Implementation
MPU	Memory protection unit	YES
DAP	Debug Access Port	YES
ETM	Embedded Trace Macrocell	NO
ITM	Instrumentation Trace Macrocell	YES
TPIU	Trace Port Interface Unit	YES
HTM	AHB Trace Macrocell	NO

3.1 Interrupts

IRQ Index	Source	Active State

IRQ[0]	GPIO	programmable
IRQ[1]	Timer0	level high
IRQ[2]	Timer1	level high
IRQ[3]	WDT	level high
IRQ[4]	UART0	level high
IRQ[5]	I2C_M	level high
IRQ[6]	FlashCtrl	level high
IRQ[7]	UART1	level high
IRQ[8]	RTC	level high
IRQ[9]	DMA0[0]	level high
IRQ[10]	DMA0[1]	level high
IRQ[11]	DMA0[2]	level high
IRQ[12]	DMA0[3]	level high
IRQ[13]	QSPI0	level high
IRQ[14]	I2S0	level high
IRQ[15]	Timer3	level high
IRQ[16]	Timer4	level high
IRQ[17]	*reserved*	level high
IRQ[18]	BOD	level high
IRQ[19]	UART2	level high
IRQ[20]	COMM	level high
IRQ[21]	PWM0	level high

IRQ[22]	PWM1	level high
IRQ[23]	PWM2	level high
IRQ[24]	PWM3	level high
IRQ[25]	PWM4	level high
IRQ[26]	SADC	level high
IRQ[27]	Crypto	level high
IRQ[28]	Timer2	level high
IRQ[29]	QSPI1	level high
IRQ[30]	Software	level high
IRQ[31]	TRNG	level high
IRQ[32]	AUX_COMP	level high

4 Memory

T32CM11C10GQ40-AR contains ROM, Flash and RAM for code & data storage.

4.1 Memory Map

Start Address	End Address	Size	Module
0x00000000	0x0FFFFFFF		Remap for Boot-Up
0x10000000	0x10007FFF	32KB	ROM (REMAP0)
0x20000000	0x2001FFFF	128KB	RAM (REMAP3)
0x22000000	0x23FFFFFF		*RAM Bit-band alias*
0x40000000	0x4FFFFFFF		ApbBridge_AO
0x40000000	0x400FFFFF		GPIO
0x40200000	0x402FFFFF		PMU
0x40600000	0x406FFFFF		RTC
0x40800000	0x408FFFFF		SYS_CTRL
0x42000000	0x43FFFFFF		*GPIO Bit-band alias*
0x80000000	0x8FFFFFFF		QSPI1
0x90000000	0x9FFFFFFF		XIP (REMAP1)
0xA0000000	0xAFFFFFFF		ApbBridge_LP
0xA0000000	0xA00FFFFF	1MB	UART0
0xA0100000	0xA01FFFFF	1MB	I2C_M
0xA0200000	0xA02FFFFF	1MB	CacheCtrl
0xA0300000	0xA03FFFFF	1MB	FlashCtrl
0xA0400000	0xA04FFFFF	1MB	Communication Subsystem
0xA0500000	0xA05FFFFF	1MB	UART1
0xA0600000	0xA06FFFFF	1MB	UART2
0xA0700000	0xA07FFFFF	1MB	Timer0
0xA0800000	0xA08FFFFF	1MB	Timer1
0xA0900000	0xA09FFFFF	1MB	WDT
0xA0A00000	0xA0AFFFFF	1MB	I2S0
0xA0B00000	0xA0BFFFFF	1MB	XDMA
0xA0C00000	0xA0CFFFFF	1MB	PWM
0xA0D00000	0xA0DFFFFF	1MB	SADC

0xA0E00000	0xA0EFFFFF	1MB	Crypto
0xA0F00000	0xA0F3FFFF	0.25MB	Timer2
0xA0F40000	0xA0F7FFFF	0.25MB	Timer3
0xA0F80000	0xA0FFFFFF	0.5MB	Timer4
0xB0000000			QSPI0
0xC0000000			DMA0

4.2 Memory Remap

Strap_0 || !Strap_1	intRemap[1]	intRemap[0]	Memory
1	0	0	REMAP0 – ROM
1	0	1	REMAP1 – XIP
1	1	0	*reserved*
1	1	1	REMAP3 – RAM
0	0	X	REMAP1 – XIP
0	1	X	REMAP3 – RAM

Notes:

Strap_0: Boot strapped of GPIO[17]

Strap_1: Boot strapped of GPIO[20]

5 DMA

The DMA is a 4-channel direct memory access controller. It is intended to be used as a controller to transfer data directly from system memory to memory or system memory to peripheral device.

Once configured and enabled, the DMA controller is primarily an AHB Master which initiates data transfers across the AHB bus to/from a peripheral device through the DMA Buffer. The DMA Buffer is an 8x32bit FIFO which is useful for peripheral devices requiring a steady stream of data.

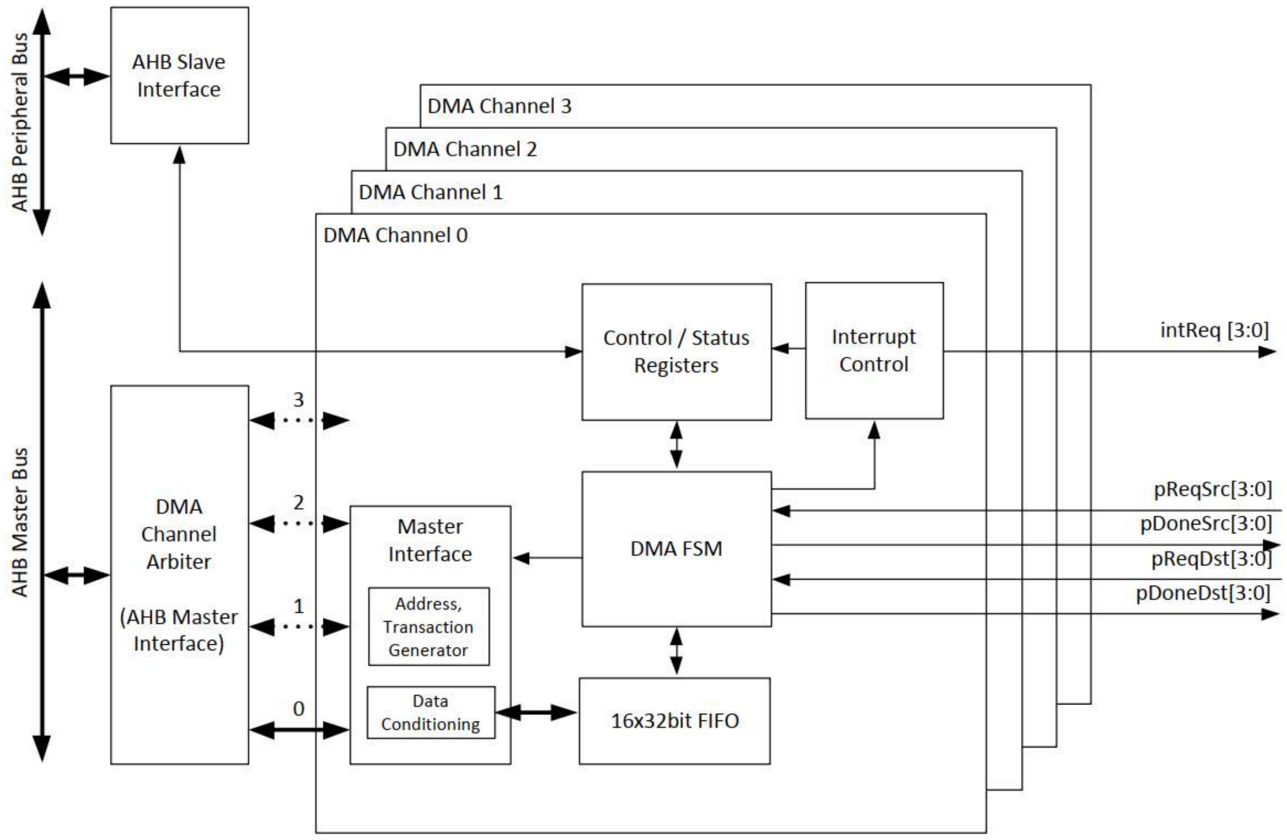
The DMA controller contains useful features such incrementing & non-incrementing addressing and linked list operation. Linked list support is useful for noncontiguous memory transfer operations.

The DMA Channel Arbiter determines which DMA Channel has access to the external AHB Master Bus. A round-robin algorithm is implemented giving equal priority to each active channel.

5.1 Features

- Four DMA Channels
- Internal Arbitration for Single AHB Master Interface
- Memory to Memory, Memory to Peripheral and Peripheral to Memory modes
- Source & destination address descriptors
- Single word & burst transfer requests
- Programmable burst size
- Current address status
- Incrementing & non-incrementing addressing
- Linked list support
- Transfer complete interrupt

5.2 Block Diagram



5.3 Functional Description

The DMA has two AHB interfaces: an AHB Slave interface and an AHB Master interface. The AHB Slave interface provides access to the control/status registers of the DMA and is used to program the specifics of the desired DMA transfer. Some of these specifics include source base address, destination base address, transfer length in bytes and AHB bus control information. The DMA Controller executes the DMA transfer via its AHB Master interface.

The DMA Controller serves as the flow controller for all transfers from the source (memory or peripheral) to the destination (memory or peripheral). DMA transfer to/from a peripheral is identical to DMA transfer to/from memory, except for the handshaking signals. Memories are defined as AHB devices that do not use a handshaking mechanism.

A DMA transfer begins with the DMA Controller executing AHB reads from the source starting from the source base address. Data retrieved from the source is transferred to an internal FIFO. After the series of AHB reads, the DMA Controller reads the internal FIFO and executes a series of AHB writes to the destination peripheral starting with the destination base address. Reads and writes alternate in this fashion until the number of requested bytes has been transferred.

The DMA Controller supports Linked List operation mode. The DMA Controller uses its AHB Master interface to retrieve control information from a linked list element AHB memory. The DMA Controller uses the control information from the linked list element to execute the DMA transfer.

The DMA Controller is designed to interface with a microprocessor. The DMA Controller issues an interrupt upon linked list element completion and/or DMA block transfer completion by asserting the interrupt signal. Block completion is determined by the address equaling the descriptor address plus the transfer length.

5.4 Memory to Memory Transfer

On a memory to memory transfer, data is read from the source memory using the source descriptor address. The data is read and loaded into the DMA 8x32 bit FIFO. If the FIFO is full or if the necessary data has been read from the source, the Finite State Machine (FSM) will write the data to the destination memory using the destination descriptor address. A maximum AHB burst value may be specified for the source/destination with the DMACtrl *srcMaxBurst*, *dstMaxBurst* bits. The DMActrl *srcAutoIncrN* and *dstAutoIncrN* bits should be cleared to automatically increment the source and destination addresses by 0x01, 0x02, or 0x04, depending on the source/destination transaction widths.

5.5 Memory to Peripheral Transfer

For a memory to peripheral transfer, data is read from the source memory using the source descriptor address. The data is read and loaded into the DMA 8x32 bit FIFO. If the FIFO is full or if the necessary data has been read from the source, the FSM will then write the data to the destination memory using the destination descriptor address.

The peripheral controls the transfer initiation by asserting the *pReqDst* signal. A maximum AHB burst value may be specified for the source/destination with the DMACtrl *srcMaxBurst*, *dstMaxBurst* bits. The DMActrl *srcAutoIncrN* bit should be cleared to have the DMA Controller increment the source addresses by 0x01, 0x02, or 0x04, depending on the source transaction width. Depending on the register map of the peripheral, the DMActrl *dstAutoIncrN* bit may need to be set to prevent the destination address from incrementing.

5.6 Peripheral to Memory Transfer

For a peripheral to memory transfer, data is read from the source peripheral using the source descriptor address. The data is loaded into the DMA 8x32 bit FIFO. If the FIFO is full or if the necessary data has been read from the source, the FSM will write the data to the destination memory using the destination descriptor address.

The peripheral controls the transfer initiation by asserting the *pReqSrc* signal. A maximum AHB burst value may be specified for the source/destination with the DMACtrl *srcMaxBurst*, *dstMaxBurst* bits. Depending on the register map of the peripheral, the DMACtrl *srcAutoIncrN* bit may need to be set to prevent the source address from incrementing. DMACtrl *dstAutoIncrN* should be cleared to automatically increment the destination addresses by 0x01, 0x02, or 0x04, depending on the destination transaction width.

5.7 Control and Status Registers

The Control and Status Registers are connected directly to a bus through an AHB Slave interface. The registers are selected from the processor. Individual registers are offset from the base address.

The control registers are used by the programmer to set up the specifics of the DMA transfer(s) including AHB bus control information such as HSIZE and HPROT, address incrementing, transfer type, source base address, destination base address, transfer length and the desired interrupting conditions. After the DMA control registers have been programmed, writing 0x00000001 to the Enable Register starts the DMA transfer.

Regarding the DMACtrl fields *srcMaxBurst* & *dstMaxBurst*, note these fields do not control the number of AHB transactions executed by the DMA Controller to fill/empty its internal FIFO. Neither does it guarantee that the specified burst value will be used on the AHB bus to execute the read/write. These fields simply indicate the DMA Controller's maximum AHB Burst size that will be used to/from the device. If the burst size specified in these fields is less than the optimal burst value which the DMA Controller would otherwise use, the DMA Controller simply will implement the necessary transfers as a series of non-sequential (single) transfers. For this reason, unless there is a compelling reason to limit the burst size it is recommended that these fields be programmed with the default value indicating an 8-beat incrementing transfer.

The *srcMaxBurst* & *dstMaxBurst* fields are also used to specify a wrapping burst. Since there is no guarantee a 4- or 8-beat wrapping burst will actually be implemented on the AHB Bus, the DMA Controller simply uses these fields for wrapping the generated addresses at the appropriate boundary and implements the desired burst transfer as a series of non-sequential (single) transfers. Note that a DMA transfer involving a wrapping burst may be better specified as (at most) two separate DMA transfers.

Regarding the Source/Destination descriptors and the Transfer Length, the programmer must be aware of two things:

1. It is the programmer’s responsibility to ensure the Source Base Address is aligned with the value specified in the *srcWidth* subfield of the Control Register and the Destination Base Address is aligned with the value specified in the *dstWidth* subfield of the Control Register. For example, if *srcWidth* specifies 32-bit transfers, the Source Base Address must be on a 32-bit boundary, e.g. 32’h08000424. A Source Base Address of 32’h08000425 is not valid but would be valid if srcWidth specifies 8-bit transfers.
2. It is the programmer’s responsibility to ensure the Transfer Length is aligned with the value specified in the *dstWidth* subfield of the Control Register. For example, if *dstWidth* specifies 16-bit transfers, the Transfer Length (Bytes) field must specify a multiple of 16-bits, e.g. 0x22. A 0x21 Transfer Length is illegal in this example but would be legal if *dstWidth* specified 8-bit transfers.

In cases of unaligned transfers from one 32-bit memory to another, 8-bit DMA transfers to and from the 32-bit devices is recommended. In some cases, it may also be possible to implement the bulk of the transfers as 32-bit transactions and the remainder as 8-bit transactions in two separate DMA transfers.

5.8 Interrupt Controller

Interrupts are generated from the following conditions, which are enabled under program control:

- Block Transfer = Completed
- Source Peripheral HRESP_DMA = ERROR
- Destination Peripheral HRESP_DMA = ERROR
- LLI Element Transfer = Completed

Interrupts are enabled by setting appropriate bits in the IntrCtrl register. The processor can determine the status of each interrupt by reading the DMAStatus register. The respective interrupt conditions may be cleared by writing a "1" to the related bit(s) in the IntrClear register.

5.9 Peripheral Handshaking

There are two peripheral request signals to the DMA controller: *pReqSrc* and *pReqDst* for the source and destination peripherals respectively. Once the DMA is enabled, the source peripheral read transfer is enabled when *pReqSrc* is asserted. Likewise the destination peripheral write transfer is enabled when *pReqDst* is asserted. *pReqSrc* & *pReqDst* must remain asserted until the designated peripheral samples a logic '1' from the *pDoneSrc*/*pDoneDst* handshake signal of the DMA controller. After sampling *pDoneSrc*/*pDoneDst* as logic '1', the peripheral must de-assert *pReqSrc*/*pReqDst* before another transfer can commence.

Because there is no mechanism to pause a transfer once it has begun, it is the responsibility of the programmer to determine the maximum atomic transfer size (in bytes) supported by each peripheral, and program the *XferLength* register of the DMA Controller accordingly. An additional mechanism exists to minimize system involvement in instances where DMA transfers occur to/from slow, low-volume peripherals. This mechanism is accessed by setting the *repeatEnable* bit in the DMACtrl register along with the *RepeatVal* register. Setting the *repeatEnable* bit to a logic '1' and programming the *RepeatVal* register enables the DMA Controller to execute the desired transfer *RepeatVal*+1 times. Each time the DMA transfer is repeated, the *pReqSrc*/*pDoneSrc* or *pReqDst*/*pDoneDst* handshake is executed but the Block Transfer Complete interrupt or the LLI Element Transfer Complete interrupt is not generated until after iteration number *repeatVal*+1. Note that this feature is available for DMA transfers of any length and is not limited to peripherals. However it is recommended the repeat mechanism is used only in instances where hardware handshaking to/from a peripheral needs to pace the transfer of a large number of bytes. In such case, the repeat mechanism accomplishes the desired behavior by essentially executing a series of small (e.g. *XferLength* < 16) handshaked transfers without involving the system. Note: when *repeatEnable* is logic ‘1’, the effective block transfer length in bytes is ((*repeatVal*+1) * *XferLength*).

5.10 Linked List Support

If the linkListEnable bit is set in the DMACtrl register, then the srcDescriptor register contains the base address of the initial element of a linked list in system memory. After the DMA Controller has been enabled in linked list mode, the DMA Controller reads system memory at the address specified in the srcDescriptor register for the first list element. Elements of the linked list are also referred to as Linked List Items (LLI).

Upon reading the Linked List memory, the DMA controller automatically updates the DMACtrl, SrcDescriptor, DestDescriptor, XferLength, and RepeatVal registers with the contents of the linked list element before executing the desired transfer. Other writes to these five registers while the DMA Controller is enabled in linked list mode are strongly discouraged. Unpredictable results may occur.

Linked List Elements are always read as 8-beat incrementing bursts from a 32-bit memory.

The Linked List Elements are organized as follows:

Field	Width	Offset	R/W	Description
SrcAddress	31:0	0x00	R	Source Address Descriptor
DestAddress	31:0	0x04	R	Destination Address Descriptor
XferLengthBytes	31:0	0x08	R	Transfer Length in bytes

Control	31:0	0x0C	R	Control Information mirroring DMACtrl register: it is the programmer's responsibility to ensure the validity of each of the subfields and to ensure the *linkedListEnable* bit is set.
NextPtr	31:0	0x10	R	Pointer to Memory location containing the next Linked List Element. 0xFFFFFFFF in this field indicates this element is the final element in the linked list.
SharedStatus	31:0	0x14	R/W	Status element should be initialized to 32'd0 by the programmer. This field is updated with 32'd1 when the element transfer is complete.
RepeatVal	31:0	0x18	R	This field mirrors the *RepeatVal* register. When the *repeatEnable* bit of the Control Field is 1, the *RepeatVal* field determines how many times this linked list element is executed before the element transfer is complete and how the source/destination address is incremented after each repeated transfer. When the *repeatEnable* bit of the Control Field is 0, this field is reserved but present.
reserved	31:0	0x1C	R	Reserved but present.

5.11 DMA Channels vs. Peripheral Connections

DMA Controller	DMA Channel	Source	Destination
DMA0	Channel #0	QSPI0_RX	
	Channel #1		QSPI0_TX
	Channel #2	QSPI1_RX	
	Channel #3		QSPI1_TX
DMA1	Channel #0	UART1_RX	
	Channel #1		UART1_TX
	Channel #2	UART2_RX	

	Channel #3		UART2_TX

5.12 Register Descriptions

Each DMA Channel has an independently configurable register set. Registers are accessed via the AHB Slave interface of the DMA Controller. The base address of DMA Channel 0 is equal to the base address of the DMA Controller. The following table shows the complete register address map for all DMA channels.

Channel	Base Address	Offset	Description
0	DMA_BASE_ADDR	0x000	Base Address of Channel 0 Registers = Base Address + Offset
1	DMA_BASE_ADDR	0x100	Base Address of Channel 1 Registers = Base Address + Offset
2	DMA_BASE_ADDR	0x200	Base Address of Channel 2 Registers = Base Address + Offset
3	DMA_BASE_ADDR	0x300	Base Address of Channel 3 Registers = Base Address + Offset

The registers described below are present in each channel and each channel has its own independent register set controlling ONLY that channel.

5.12.1 DMACtrl (offset: 0x00)

Signal	Bits	Default	R/W	Description
srcAutoIncrN	31	b0	R/W	1 = Do not increment the source address 0 = Increment the source address
srcWidth	30:28	b000	R/W	Width of Source device in terms of the AHB signal HSIZE: 000 = 8-bit device 010 = 32-bit device

				All other values are reserved.
srcMaxBurst	27:25	b101	R/W	Maximum Source Burst according to AHB signal HBURST: 000 = No bursting, single transfers only 001 = Reserved, not supported 010 = 4-beat Wrapping Burst 011 = 4-beat Incrementing Burst 100 = 8-beat Wrapping Burst 101 = 8-beat Incrementing Burst All other values are reserved.
srcProt	24:21	b0010	R/W	Source Protection coding according to AHB signal HPROT: HPROT[3]: 1 = cacheable, 0 = non-cacheable HPROT[2]: 1 = bufferable, 0 = non-bufferable HPROT[1]: 1 = Priveleged Access, 0 = User Access HPROT[0]: 1 = Opcode Fetch, 0 = Data Access
srcReqSel	20	b0	R/W	Select peripheral source request of RX: 1 = single request 0 = burst request
reserved	19:16			
dstAutoIncN	15	b0	R/W	1 = Do not increment the destination address 0 = Increment the destination address
dstWidth	14:12	b000	R/W	Width of Destination device in terms of the AHB parameter HSIZE: 000 = 8-bit device 010 = 32-bit device

				All others reserved.
dstMaxBurst	11:9	b101	R/W	Maximum Destination Burst according to AHB signal HBURST: 000 = No bursting, single transfers 001 = Reserved, not supported 010 = 4-beat Wrapping Burst 011 = 4-beat Incrementing Burst 100 = 8-beat Wrapping Burst 101 = 8-beat Incrementing Burst All other values are reserved.
dstProt	8:5	b0001	R/W	Source Protection coding according to AHB signal HPROT: HPROT[3]: 1 = cacheable, 0 = non-cacheable HPROT[2]: 1 = bufferable, 0 = non-bufferable HPROT[1]: 1 = Priveleged Access, 0 = User Access HPROT[0]: 1 = Opcode Fetch, 0 = Data Access
reserved	4			
repeatEnable	3	b0	R/W	1 = transfer of XferLength bytes is repeated according to the value in the RepeatVal register 0 = transfer is not repeated
mode	2:1	b00	R/W	Modes: 00 = Source Memory to Destination Memory 01 = Source Memory to Destination Peripheral 10 = Source Peripheral to Destination Memory 11 = reserved

linkListEnable	0	b0	R/W	1 = enable the linked list pointers 0 = use registers; no linked lists

5.12.2 DMAStatus (offset: 0x04)

Signal	Bits	Default	R/W	Description
reserved	31:16			
lliElementCount	15:8	x00	RO	8-bit rollover counter that tracks the number of completed linked list transfers. This counter is reset by a write to the Enable register.
enable	7	b0	RO	1 = enabled; 0 = not enabled
reserved	6:4			
lliElementDone	3	b0	RO	1 = the DMA controller has processed at least one list element in LLI mode since the last DMAStatus Register read 0 = the DMA controller has not processed any list elements in LLI mode since the last DMAStatus Register read This bit is automatically cleared by reading the DMAStatus Register.
dstPerErr	2	b0	RO	Destination peripheral HRESP = ERROR; this bit is cleared by writing 0 to the Enable Register to disable the DMA controller operation.
srcPerErr	1	b0	RO	Source peripheral HRESP = ERROR; this bit is cleared by writing 0 to the Enable Register to disable the DMA controller operation.
xferDone	0	b0	RO	Transfer complete; this bit is cleared by writing 0 to the Enable Register to disable the DMA controller operation.

5.12.3 SrcDescriptor (offset: 0x08)

Signal	Bits	Default	R/W	Description
SrcDescriptor	31:0	x0	R/W	Source Address Descriptor Note: If linkListEnable = 1, this register is interpreted as the initial linked list pointer.

5.12.4 DstDescriptor (offset: 0x0C)

Signal	Bits	Default	R/W	Description
DstDescriptor	31:0	x0	R/W	Destination Address Descriptor

5.12.5 XferLength (offset: 0x10)

Signal	Bits	Default	R/W	Description
XferLength	31:0	x0	R/W	Transfer length in bytes

5.12.6 CurSrcAddr (offset: 0x14)

Signal	Bits	Default	R/W	Description
CurSrcAddr	31:0	x0	RO	Current source address

5.12.7 CurDstAddr (offset: 0x18)

Signal	Bits	Default	R/W	Description
CurDstAddr	31:0	x0	RO	Current destination address

5.12.8 IntrCtrl (offset: 0x1C)

Signal	Bits	Default	R/W	Description
reserved	31:4			
enaLliElementDone	3	b0	R/W	Enable linked list element complete interrupt
enaDstPerErr	2	b0	R/W	Enable Destination peripheral HRESP=ERROR interrupt
enaSrcPerErr	1	b0	R/W	Enable Source peripheral HRESP=ERROR interrupt
enaXferDone	0	b0	R/W	Enable Transfer complete interrupt

5.12.9 IntrClear (offset: 0x20)

Signal	Bits	Default	R/W	Description
reserved	31:4			
clrLliElementDone	3	b0	W1C	Clear linked list element complete interrupt
clrDstPerErr	2	b0	W1C	Clear Destination peripheral HRESP=ERROR interrupt
clrSrcPerErr	1	b0	W1C	Clear Source peripheral HRESP=ERROR interrupt
clrXferDone	0	b0	W1C	Clear Transfer complete interrupt

5.12.10 Enable (offset: 0x24)

Signal	Bits	Default	R/W	Description
reserved	31:2			
dmaClkGate	1	b0	R/W	1 = DMA master clock gating Enable 0 = DMA master clock gating Disable
Enable	0	b0	R/W	1 = DMA Enable 0 = DMA Disable

5.12.11 LliNextPtr (offset: 0x28)

Signal	Bits	Default	R/W	Description
LliNextPtr	31:0	x0	RO	Displays the current LLI Next Pointer when DMA transfer is under link list control (linkListEnable = 1). When link list control is disabled, this register is read as 0.

5.12.12 RepeatVal (offset: 0x2C)

Signal	Bits	Default	R/W	Description
reserved	31:29			
rptSrcAddrIncrMult	28:24	x0	R/W	Allowed values: 0, 1, 2, 4, 8 If the repeatEnable bit (in the DMACtrl register) is 1, this subfield determines how the source address is incremented after each iteration of the repeated transfer. rptSrcAddrIncrMult is right-shifted by srcWidth to get the increment value. For example, if srcWidth = 3’b001 and rptSrcAddrIncrMult = 8, the source address will be incremented by 16 after each iteration. A value of 0 in this

				subfield has the effect of not incrementing the base source address between iterations.
reserved	23:21			
rptDstAddrIncrMult	20:16	x0	R/W	Allowed values: 0, 1, 2, 4, 8 If the repeatEnable bit (in the DMACtrl register) is 1, this subfield determines how the destination address is incremented after each iteration of the repeated transfer. rptDstAddrIncrMult is right-shifted by dstWidth to get the increment value. For example, if dstWidth = 3'b010 and rptDstAddrIncrMult = 4, the destination address will be incremented by 16 after each iteration. A value of 0 in this subfield has the effect of not incrementing the base destination address between iterations.
reserved	15:12			
repeatVal	11:0	x0	R/W	If the repeatEnable bit (in the DMACtrl register) is 1, this subfield determines the number of times a given DMA transfer of XferLength bytes is repeated. repeatVal=0 means that the requested transfer will execute once and will not be repeated. A maximum value of 4095 means that the requested transfer will be executed 4096 times.

5.12.13 RepeatStatus (offset: 0x30)

Signal	Bits	Default	R/W	Description
reserved	31:12			

RepeatStatus	11:0	x0	RO	If the repeatEnable bit (in the DMACtrl register) is 1, this field represents a current status of the down-counter that tracks the number of times a DMA transfer is repeated. RepeatStatus will start with the value in RepeatVal and will be decremented by 1 upon completion of the transfer of XferLength bytes until RepeatStatus is 0. If the repeatEnable bit is 0, this field has no meaning and it can be ignored.

6 Flash Controller

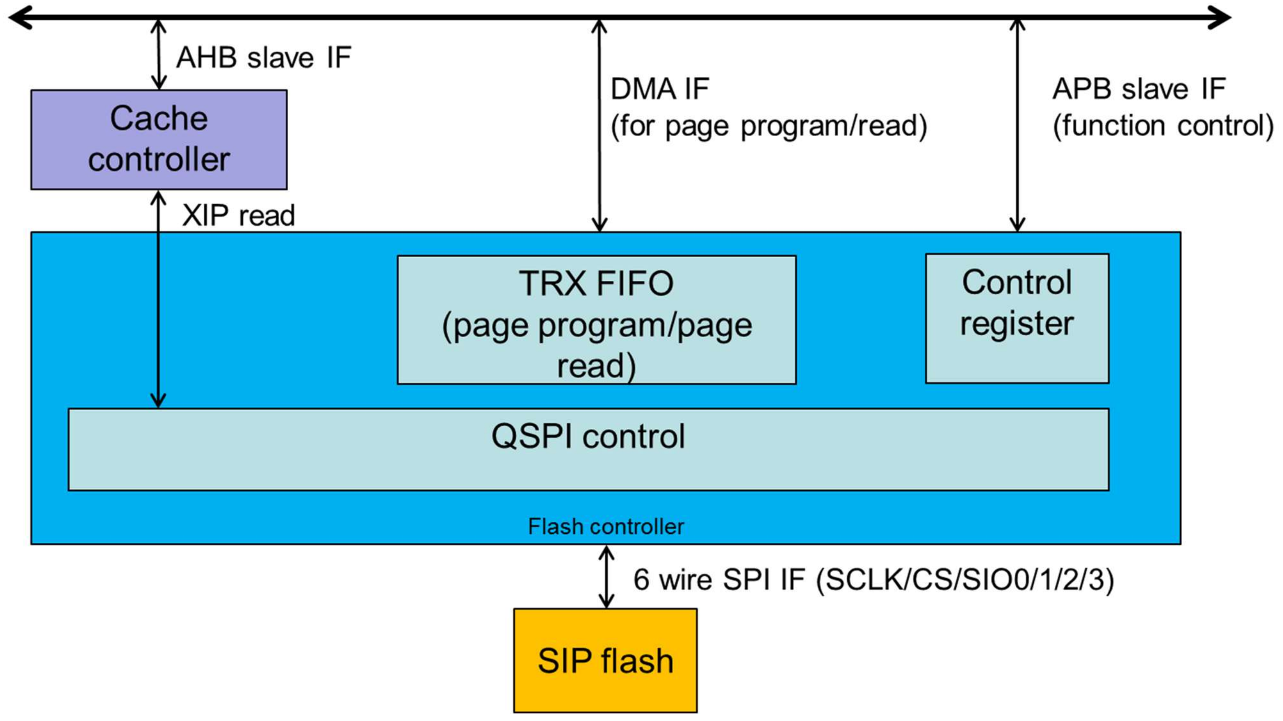
The flash controller controls the internal flash.

6.1 Features

- Supports standard SPI/Dual-bit mode/Qual-bit mode Flash
- Same clock as the system clock
- Interrupt control

6.2 Block Diagram

The Flash Controller includes the QSPI control block, TRX FIFO block and control register block.



6.3 Functional Description

The Flash Controller provides system AHB bus access to the internal flash through the cache controller. It also supports the SW driver changing flash content by erasing & programming the content, moving data to memory, changing the registers in the internal flash, etc.

6.4 Register Description

6.4.1 Controller Command (offset: 0x00)

Signal	Bits	Default	R/W	Description
Reserved	31:7			
SecurityRegisterOperation	6		R/W	Security register control 1'b0: memory array 1'b1: security register
Command	5:0		R/W	6'b00_XXXX: Read function XXXX definitions: 4'b0000: read byte (data @ reg: FlashReadData) 4'b0010: read SFDP (data @ reg: FlashReadData) 4'b0100: read verify (data @ reg: PageCRC) 4'b1000: read page (data @ mem: MemoryAddress) 4'b1001: read unique ID (data @ mem: MemoryAddress) 4'b1010: read SFDP (data @ mem: MemoryAddress) 4'b1101: read status register 1 (data @ reg: FlashReadData) 4'b1110: read status register 2 (data @ reg: FlashReadData)

				4'b1111: read status register 3 (data @ reg: FlashReadData) 6'b01_XXXX: Write function XXXX: 4'b0000: program byte (data@ reg: FlashWriteData) 4'b1000: program page (data @ reg: FlashWriteData) 4'b1100: write status register 1 (&2, by 01h instruction setting by Sr12, data @ reg: StatusRegister1&2) 4'b1101: write status register 1 (data @ reg: StatusRegister1) 4'b1110: write status register 2 (data @ reg: StatusRegister2) 6'b10_XXXX: Erase function XXXX: 4'b0001: sector erase 4'b0010: block 32K erase 4'b0100: block 64K erase 4'b1000: chip erase

6.4.2 Flash Address (offset: 0x04)

Signal	Bits	Default	R/W	Description
FlashAddress	31:0		R/W	Flash address for operation (value can map to 0x90000000)

6.4.3 Controller Start (offset: 0x08)

Signal	Bits	Default	R/W	Description

Reserved	31:9			
ControllerBusy	8		R	Controller status
Reserved	7:1			
ControllerStart	0		W1C	Control start, needs match with the set pattern

6.4.4 Flash Status Register (offset: 0x0C)

Signal	Bits	Default	R/W	Description
Reserved	31:16		R/W	
StatusRegister2	15:8		R/W	Write flash status register 2
StatusRegister1	7:0		R/W	Write flash status register 1

6.4.5 Flash Data (offset: 0x10)

Signal	Bits	Default	R/W	Description
Reserved	31:16			
FlashReadData	15:8		R	Read data from flash: byte read data, byte read SFDP, read status register
FlashWriteData	7:0		R/W	Write data into flash: byte program data

6.4.6 Memory Address Setting (offset: 0x14)

Signal	Bits	Default	R/W	Description

MemoryAddress	31:0		R/W	Page program/read data from/to memory address (need 4 byte aligned, [1:0] need to be 2'b00)

6.4.7 Controller Setting (offset: 0x18)

Signal	Bits	Default	R/W	Description
reserved	31:10			
SuspendEnable	9		R/W	When flash program/erase, cache can access flash 1'b0: disable 1'b1: enable
XIPReadEnable	8		R/W	Cache can access flash 1'b0: disable 1'b1: enable
SrQe	7		R/W	QE @ status register 1'b0: status register 1[6] 1'b1: status register 2[1]
Sr12	6		R/W	Status register 1 & 2 can use instruction 01h write 1'b0: no 1'b1: yes
ProgramMode	5:4		R/W	2'b00: 1-1-1 program (instruction = 02h) 2'b10: 1-1-4 program (instruction = 32h) 2'b11: 1-4-4 program (instruction = 38h)
ReadMode	3:1		R/W	3'b000: 1-1-1 read (instruction = 0Bh)

				3'b001: 1-1-2 read (instruction = 3Bh) 3'b01x: 1-1-4 read (instruction = 6Bh) 3'b100: 1-1-1 read (instruction = 0Bh) 3'b101: 1-2-2 read (instruction = BBh, no support continuous mode) 3'b11x: 1-4-4 read (instruction = EBh, no support continuous mode)
EnterDpdEnable	0		R/W	Enter deep power-down when system is sleep

6.4.8 CRC (offset: 0x1C)

Signal	Bits	Default	R/W	Description
reserved	31:8			
PageCRC	7:0		R	Page program/verify for page CRC result

6.4.9 DPD Time (offset: 0x20)

Signal	Bits	Default	R/W	Description
Reserved	31:14			
DPDTime	13:0		R/W	Deep power-down time

6.4.10 RDPD Time (offset: 0x24)

Signal	Bits	Default	R/W	Description
Reserved	31:14			
RDPDTime	13:0		R/W	Release deep power-down time

6.4.11 Suspend Time (offset: 0x28)

Signal	Bits	Default	R/W	Description
Reserved	31:14			
SuspendTime	13:0		R/W	Suspend time

6.4.12 Resume Time (offset: 0x2C)

Signal	Bits	Default	R/W	Description
Reserved	31:14			
ResumeTime	13:0		R/W	Resume time

6.4.13 Page Read byte Number (offset: 0x34)

Signal	Bits	Default	R/W	Description
reserved	31:8			

PageReadByte	7:0		R/W	Page read byte number, need 4 byte (double word, [1:0] must = 2'b11)

6.4.14 Flash Information (offset: 0x38)

Signal	Bits	Default	R/W	Description
reserved	31:24			
FlashCapacityID	23:16		R	Flash capacity ID
FlashTypeID	15:8		R	Flash type ID
ManufacturerID	7:0		R	Flash manufacturer ID

6.4.15 Flash Control Interrupt (offset: 0x40)

Signal	Bits	Default	R/W	Description
reserved	31:17			
InterruptClear	16		W1C	Interrupt clear
reserved	15:9			
InterruptEnable	8		R/W	Interrupt enable
reserved	7:1			
InterruptStatus	0		R	Interrupt status

6.4.16 Operation Pattern Match (offset: 0x44)

Signal	Bits	Default	R/W	Description
PatternMatch	31:0		R/W	Need write RABT ASCII (0x52_41_42_54) for start operation

7 Cache Controller

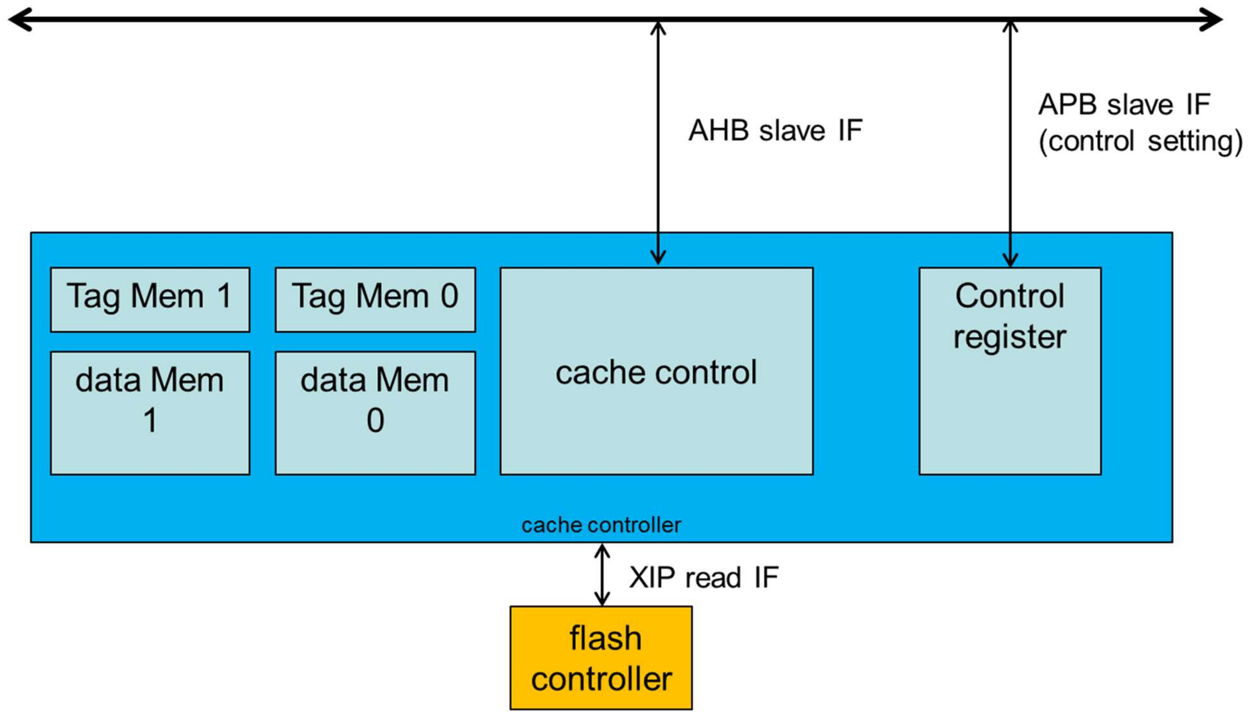
The cache controller is designed as a flash cache for instruction and data to be stored in the internal flash. It is also an interface between the AHB bus and the flash controller.

7.1 Features

Supports 2-way set-associative cache and can close one way for power saving.

7.2 Block Diagram

The cache controller includes the cache control block, the control register block and two-way cache memory block for tag and data stored separately.



7.3 Functional Description

The cache controller provides system AHB bus access to the flash controller. It can automatically be a buffer when the CPU Cortex-M3 or any peripheral needs data in the internal flash through the system AHB bus. When the flash content is changed, the cache must first be cleared by SW to avoid corrupting the new content.

7.4 Register Description

7.4.1 Controller register (offset: 0x00)

Signal	Bits	Default	R/W	Description
Reserved	31:10			
CacheWay1Clr	9		W1C	Write one to clear cache way 1 content
CacheWay0Clr	8		W1C	Write one to clear cache way 0 content
Reserved	7:2			
CacheWay1Enable	1		R/W	Cache Way 1 enable 1'b1: Cache 2 way (way 1 enable) 1'b0: Cache 1 way (way 1 disable)
CacheEnable	0		R/W	Cache enable 1'b1: Cache Enable 1'b0: Cache Disable

8 QSPI

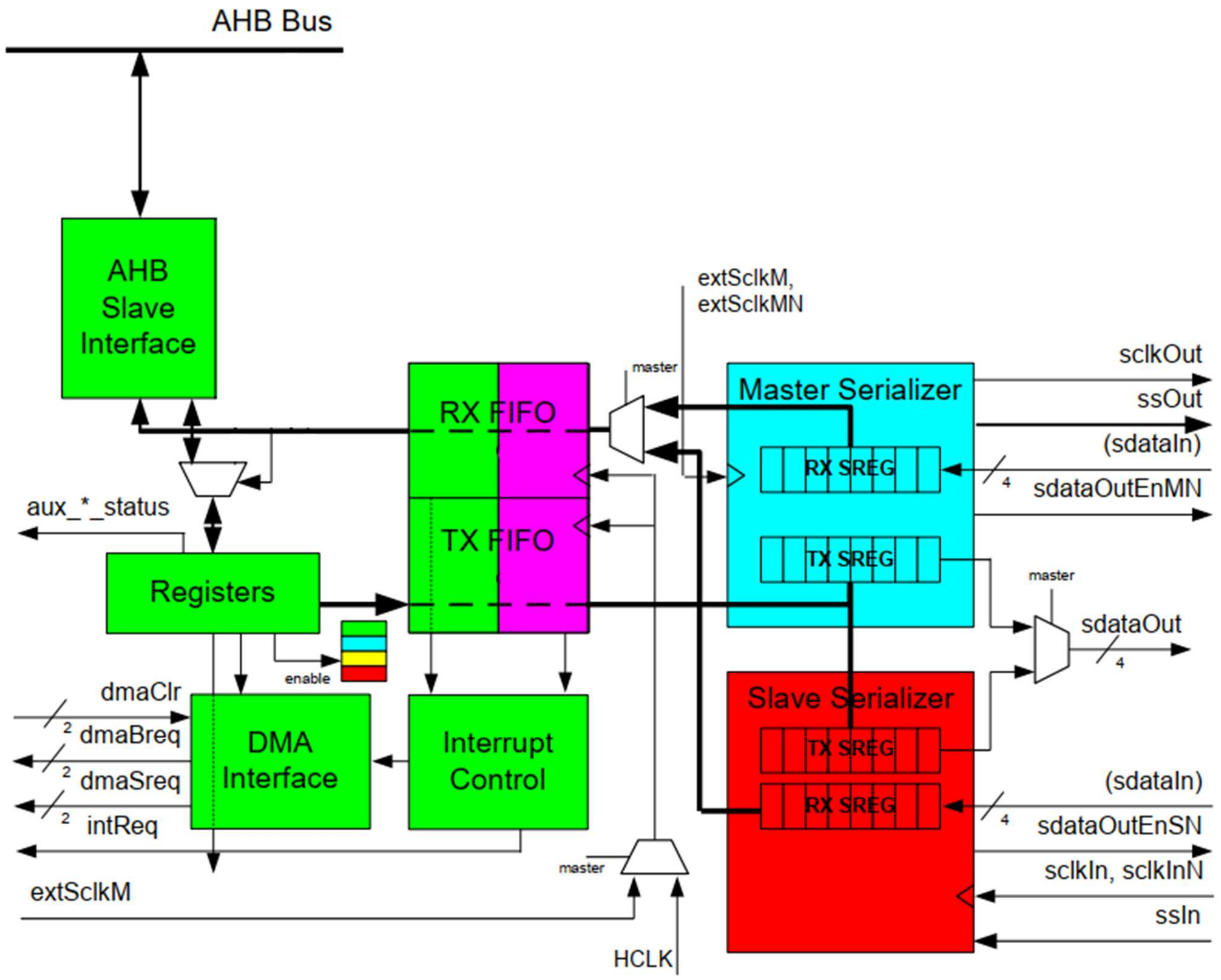
The Quad Serial Peripheral Interface module either controls a serial data link as a master or reacts to a serial data link as a slave.

The AHB bus controller can be configured under software control to be a master or slave device. Reading & writing the core is done on the AMBA AHB bus interface. The core operates in various data modes (8-bit or 32-bit). The data is serialized and then transmitted, either LSB or MSB first, using the standard 4-wire SPI bus interface or the extended Quad mode bus.

8.1 Features

- 8- or 32-bit serial transmit & receive
- Half duplex operation
- Software programmable Master or Slave mode
- Quad-bit mode operation
- Dual-bit mode operation
- separate SCLK input for Master Mode
- 32-word Transmit FIFO
- 32-word Receive FIFO
- Interrupt control
- LSB or MSB mode
- Tristate Slave MISO signaling for multiple slaves
- DMA Interface
- Compatible with many industry-standard FLASH devices

8.2 Block Diagram



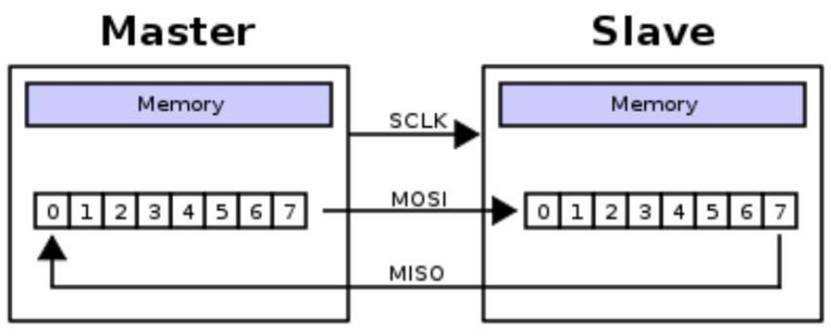
In the diagram, there are several clock domains represented: AHB clock (green), External Master SCLK (light blue) and Slave SCLK (red). The FIFOs serve as the clock boundary between the AHB clock and the SPI bus clock. Production of the SPI bus clock varies, depending on Master/Slave mode. In Master mode, the QSPI core produces the SPI bus clock directly from the External Master SCLK. In Slave mode, the QSPI core receives the SPI bus clock from another SPI Master. In Master mode, the Master Serializer and the right side of the FIFOs (purple) use the same clock. In Slave mode, the Slave Serializer (red) is clocked directly by the gapped SPI bus clock *sclkIn*/*sclkInN*, and the right side of the FIFOs (purple) is clocked by HCLK. In Slave mode, either *ssIn* or an internal signal that compares the receive data count against a terminal value is synchronized to the HCLK domain. The FIFO pointers are advanced on the rising edge of this active-high synchronized signal.

The master bit is responsible for directing much of the internal multiplexing. One other signal is also very important: enable. The enable bit is synchronized to the different clock domains (when appropriate). These synchronized enable signals, along with proper programming order of the AHB registers, allows logic in other domains to safely sample other AHB register values and control bits since they are guaranteed to be stable after enable goes high. In short, the enable bit serves as a master synchronizer for the entire module.

8.3 Functional Description

8.3.1 SPI Data Link

Data is transmitted synchronously with MOSI (Master Out, Slave In) relative to the SCLK generated by the master device. The master also receives data on the MISO (Master In, Slave Out) signal.



8.3.2 DMA Operation

QSPI has its native DMA engine. DMA registers (offset 0x60 ~ 0x84) can be set to control TX and RX DMA transactions.

8.4 Register Description

8.4.1 Serial Data Transmit (to Tx FIFO) (offset: 0x00)

Signal	Bits	Default	R/W	Description
TX_Data_Reg	31:0	x0	WO	Serial Transmit Data Register No packing of transmit data is attempted. The number of bits transmitted per FIFO element is determined by the “bitsize” value in the Control Register (e.g. bitsize = 010 means only 12 bits are used). Also, the “msb1st” value in the Control Register determines whether the least-significant bits or the most-significant bits of the FIFO element are sent first.

8.4.2 Serial Data Receive (from Rx FIFO) (offset: 0x04)

Signal	Bits	Default	R/W	Description
RX_Data_Reg	31:0	x0	RO	Serial Receive Data Register No packing of receive data is attempted. The number of bits stored per FIFO element is determined by the “bitsize” value in the Control Register (e.g. bitsize = 011 means only 16 bits are used).

8.4.3 Control Register (offset: 0x0C)

Signal	Bits	Default	R/W	Description
reserved	31:16			
txWmarkSet	15:14	b00	R/W	Sets the Watermark Level for TX FIFO operation

				00: reserved 01: 8 entries 10: 16 entries 11: 24 entries
rxWmarkSet	13:12	b00	R/W	Sets the Watermark Level for RX FIFO operation 00: reserved 01: 8 entries 10: 16 entries 11: 24 entries
mWaitEn	11	b0	R/W	Inter-transfer delay for Master operation 0: disable delay (full speed), 1: enable delay.
reserved	10:7			
sdata0or1	6	b0	R/W	In normal SPI mode, set 0 for master and set 1 for slave 0: SDATA[0] = sdata_out[0] sdata_in[0] = SDATA[1] 1: SDATA[1] = sdata_out[0] sdata_in[0] = SDATA[0].
master	5	b1	R/W	SPI port bus master operating mode 0: slave mode 1: master mode
cpol	4	b0	R/W	0: SPI clock rests low 1: SPI clock rests high

cpha	3	b0	R/W	0: first bit(s) of sdataOut occur as SS is asserted 1: sdataOut occurs on first SPI clock edge after SS
msb1st	2	b0	R/W	MSB/LSB 1: MSB first 0: LSB first Note: In Quad or Dual mode, this setting only affects the order of (4-bit or 2-bit) chunk transmission, not the bit-order within the chunk. sdataOut[3]/sdataIn[3] is always the MSB of the chunk for Quad mode. sdataOut[1]/sdataIn[1] is always the MSB of the chunk for Dual mode.
byteSwap	1	b0	R/W	Byte swap in bitsize = 16/32 bit for TX/RX FIFO.
contXfer	0	b1	R/W	Continuous Transfer bit. This is relevant for Master operation only. Note this setting also affects the timing of the master output enable signal sdataOutEnMN. 0: ssOut goes inactive between successive transfers 1: ssOut stays active until the TX FIFO is empty AND the contXferExtend bit (in the Auxiliary Control Register) is low.

8.4.4 Auxiliary Control Register (offset: 0x10)

Note: For Master operation, the Auxiliary Control Register may be safely reconfigured while the QSPI Controller is enabled under the following circumstance: the TX FIFO is empty AND there is no transfer in progress.

Signal	Bits	Default	R/W	Description
reserved	31:8			
contXferExtend	7	b0	R/W	Continuous Transfer Extend Bit. This is relevant for Master operation only. This bit has no effect if the contXfer bit is 0. This register bit is synchronized to the SCLK domain for use there. Note this setting also affects the timing of the master output enable signal, sdataOutEnMN. Set to 1 to have ssOut[x] remain active even when the TX FIFO is empty. Set to 0 to have ssOut[x] become inactive when the TX FIFO is empty.
bitsize	6:4	b111	R/W	bitsize is set for access format of Tx/Rx FIFO: 001: 8-bit mode 111: 32-bit mode Other: reserved Note for DMA TX, if DMA_TX_Len is not a multiple of 4-bytes, bitsize must be set to 3’b001. In other cases, bitsize could be set to 3’b111.
inhibitDin	3	b0	R/W	Set to 1 to inhibit the SPI serializers from writing to the Read Data FIFO.
inhibitDout	2	b0	R/W	Set to 1 to inhibit the SPI serializers from reading the Transmit FIFO. Note: this is intended to be used as a Slave mode configuration. In many cases, a Slave simply receives data from a Master. In such cases, setting this bit will allow the Slave to receive data without causing a TX FIFO underflow condition.
qmode	1:0	b00	R/W	SPI Modes

				11: Quad SPI 10: Dual SPI 01: Reserved 00: Normal SPI

8.4.5 Status Register (offset: 0x14)

Signal	Bits	Default	R/W	Description
reserved	31:8			
rxFull	7	b0	RO	Receive FIFO full flag
rxWmark	6	b0	RO	Receive FIFO watermark flag
rxEmpty	5	b1	RO	Receive FIFO empty flag
txFull	4	b0	RO	Transmit FIFO full flag
txWmark	3	b1	RO	Transmit FIFO watermark flag
txEmpty	2	b1	RO	Transmit FIFO empty flag
reserved	1			
xferIP	0	b0	RO	Transfer is in progress (ssOut/ssIn is active)

8.4.6 Slave Select Register (offset: 0x18)

Note: This register is relevant only when the SPI Controller is configured as SPI Master.

Signal	Bits	Default	R/W	Description

reserved	31:4			
ssout[3]	3	b0	R/W	0: do not select Slave 3 for TX/RX 1: select Slave 3 for TX/RX
ssout[2]	2			0: do not select Slave 2 for TX/RX 1: select Slave 2 for TX/RX
ssout[1]	1			0: do not select Slave 1 for TX/RX 1: select Slave 1 for TX/RX
ssout[0]	0			0: do not select Slave 0 for TX/RX 1: select Slave 0 for TX/RX

Note: the corresponding output signal ssOut[3:0] will transition to the active state when the module is enabled and its TX FIFO is not empty. See the contXfer bit in the Control Register and the contXferExtend bit in the Auxiliary Control Register for a description of when ssOut[3:0] returns to the inactive state. Important: the Slave Select Polarity Register determines whether “active” means 1 or 0.

8.4.7 Slave Select Polarity Register (offset: 0x1C)

Signal	Bits	Default	R/W	Description
reserved	31:4			
sspol3	3	b0	R/W	0: ssOut[3] (Master) is active low 1: ssOut[3] (Master) is active high
sspol2	2	b0	R/W	0: ssOut[2] (Master) is active low 1: ssOut[2] (Master) is active high
sspol1	1	b0	R/W	0: ssOut[1] (Master) is active low 1: ssOut[1] (Master) is active high

sspol0	0	b0	R/W	0: ssOut[0] (Master) or ssIn (Slave) is active low 1: ssOut[0] (Master) or ssIn (Slave) is active high

8.4.8 Interrupt Bit Mapping

Bit	Condition	Triggered
[5]	rxNotEmptyPulse	Edge: RX FIFO Not Empty flag transitions from low to high
[4]	xferDonePulse	Edge: ssOut(Master)/ssIn(Slave) transitions from active to inactive
[3]	rxFullPulse	Edge: RX FIFO Full flag transitions from low to high
[2]	rxWmarkPulse	Edge: RX FIFO Wmark flag transitions from low to high
[1]	txWmarkPulse	Edge: TX FIFO Wmark flag transitions from high to low
[0]	txEmptyPulse	Edge: TX FIFO Empty flag transitions from high to low

8.4.9 Interrupt Enable Register (offset: 0x20)

Signal	Bits	Default	R/W	Description
reserved	31:6			
intrEnable	5:0	x00	R/W	Write 1 to enable individual interrupts (see Interrupt Mapping). Read will return status of the enabled interrupts.

8.4.10 Interrupt Status Register (offset: 0x24)

Signal	Bits	Default	R/W	Description
reserved	31:6			
intrStatus	5:0	x00	RO	Interrupt Status 1: interrupt source status 0: no interrupt

8.4.11 Interrupt Clear Register (offset: 0x28)

Signal	Bits	Default	R/W	Description
reserved	31:6			
intrClear	5:0	x00	W1C	Clear Interrupt Write 1 to clear the interrupt bit Write 0 has no effect

8.4.12 TX FIFO Level Register (offset: 0x2C)

Signal	Bits	Default	R/W	Description
Reserved	31:7			
tx_FIFO_level	6:0	x00	RO	Read the current transmit FIFO level (0 ~ 32)

8.4.13 RX FIFO Level Register (offset: 0x30)

Signal	Bits	Default	R/W	Description

reserved	31:7			
rx_FIFO_level	6:0	x00	RO	Read the receive FIFO current level (0 ~ 32).

8.4.14 Master Inter-transfer Delay (offset: 0x38)

Signal	Bits	Default	R/W	Description
reserved	31:8			
mWait	7:0	x00	R/W	If master (Control Register) is 1, this register sets the inter-transfer delay for the master serializer. The delay is (mwait+1) SCLK cycles.

8.4.15 Enable Register (offset: 0x3C)

Signal	Bits	Default	R/W	Description
reserved	31:1			
enable	0	b0	R/W	SPI enable 0: Disable SPI TX & RX, also reset RX and TX FIFOs on the transition from 1 to 0 of this bit. Although SPI can be disabled at any time, it is recommended that it be done when FIFOs are empty. After writing 0 to the enable bit, read back the enable bit and any further register configurations can be performed only after the enable bit is verified = 0.

				1: Enable SPI TX & RX.

8.4.16 Mater Clock Divider Register (offset: 0x50)

Signal	Bits	Default	R/W	Description
reserved	31:9			
MstClkDivEn	8	b0	R/W	Master clock divider enable 0: Master clock = 32 MHz (only use when HCLK frequency is higher than 32 MHz) 1: Master clock = clock selected by MstClkDivSel.
MstClkDivSel	7:0	b0	R/W	Master clock divider select 0x00: 32/2 for 16 MHz 0x01: 32/4 for 8 MHz 0x02 = 32/6 for 5.333 MHz 0x03 = 32/8 for 4 MHz etc. … 0xFF = 32/512 for 62.5 KHz

8.4.17 DMA_RX_Start (offset: 0x60)

Signal	Bits	Default	R/W	Description
DMA_RX_Start	31:0	x0	R/W	Start address of RX buffer in SRAM for DMA

8.4.18 DMA_RX_Len (offset: 0x64)

Signal	Bits	Default	R/W	Description
reserved	31:16			
DMA_RX_Len	15:0	x0	R/W	DMA transfer byte length for RX

8.4.19 DMA_TX_Start (offset: 0x68)

Signal	Bits	Default	R/W	Description
DMA_TX_Start	31:0	x0	R/W	Start address of TX buffer in SRAM for DMA

8.4.20 DMA_TX_Len (offset: 0x6C)

Signal	Bits	Default	R/W	Description
reserved	31:16			
DMA_TX_Len	15:0	x0	R/W	DMA transfer byte length for TX

8.4.21 DMA_RX_RemainLen (offset: 0x70)

Signal	Bits	Default	R/W	Description
reserved	31:16			

DMA_RX_RemainLen	15:0	x0	RO	DMA transfer's byte length remaining for the RX process

8.4.22 DMA_TX_RemainLen (offset: 0x74)

Signal	Bits	Default	R/W	Description
reserved	31:16			
DMA_TX_RemainLen	15:0	x0	RO	DMA transfer's byte length remaining for TX process

8.4.23 DMA_IER (offset: 0x78)

Signal	Bits	Default	R/W	Description
reserved	31:2			
DMA_IER_TX	1	x0	R/W	Interrupt enable for DMA TX
DMA_IER_RX	0	x0	R/W	Interrupt enable for DMA RX

8.4.24 DMA_ISR (offset: 0x7C)

Signal	Bits	Default	R/W	Description
reserved	31:2			
DMA_ISR_TX	1	x0	W1C	Interrupt status for DMA TX
DMA_ISR_RX	0	x0	W1C	Interrupt status for DMA RX

8.4.25 DMA_RX_Enable (offset: 0x80)

Signal	Bits	Default	R/W	Description
reserved	31:16			
DMA_txDummy_Dat	15:8	x0	R/W	Dummy data for “DMA_txDummy_En”
reserved	7:3			
DMA_RX_High_Pri	2	x0	R/W	Enable to force both QSPI0 & QSPI1 RX DMA requests having higher priority than TX DMA requests in the AHB arbiter. Note: this bit is only valid at QSPI0 register space.
DMA_txDummy_En	1	x0	R/W	Enable to fill the TX FIFO with dummy data for master RX. Write 0 for completion or abort.
DMA_RX_Enable	0	x0	R/W	DMA enable for RX start. Write 0 for completion or abort.

8.4.26 DMA_TX_Enable (offset: 0x84)

Signal	Bits	Default	R/W	Description
reserved	31:1			
DMA_TX_Enable	0	x0	R/W	DMA enable for TX start. Write 0 for completion or abort.

9 GPIO

The APB GPIO is a configurable module allowing use of 32 scalable I/O lines.

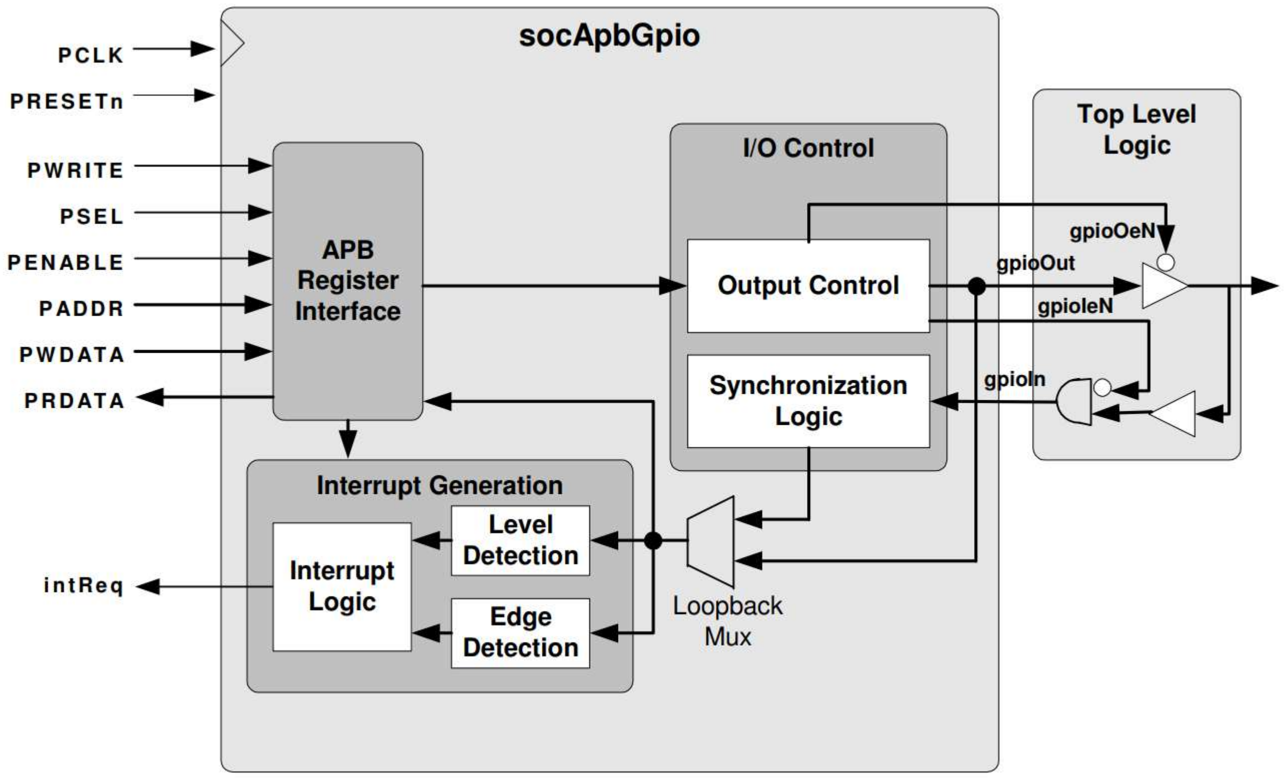
Each line can be configured independently resulting in very useful I/O applications. The GPIO module supports a wide variety of options for synchronization logic and interrupt signals which can be triggered by either a low level, high level, positive edge or negative edge. These GPIO are also fully compatible with the industry-standard APB bus interface.

The GPIO module is a standard APB Slave peripheral and has a standard APB Slave Port. The APB Slave Port is typically connected to one of the various APB Mirrored-Slave Ports of an APB Channel module. The several GPIO registers are accessed through its APB interface.

9.1 Features

- General Purpose Input Output
- Software configurable
- AMBA® APB Compatible
- Each I/O is independent
- Interrupt control for each I/O
- Input Enables can be used to quiet unused inputs

9.2 Block Diagram



9.3 Functional Description

Each GPIO line can be independently programmed as an input or output. Separate Set and Clear registers are provided because it is likely different software tasks may be servicing different I/O signals. The separate Set and Clear registers make read-modify-write operations with interrupts disabled unnecessary.

Inputs are synchronized to the system clock through a pair of flip-flops. Each input can be programmed to generate an interrupt. The interrupt can be programmed as level-sensitive or edge-sensitive and the level (high/low) or edge (rising/falling/either) causing the interrupt can be selected. Interrupts can be individually enabled or disabled. Level-sensitive interrupts stay asserted until the interrupting condition is cleared. Edge-triggered interrupts are cleared by writing to the GPIO interrupt clear register.

For power saving, inputs may be inhibited when not in use by setting gpioIeN[n] high.

A loopback function optionally routes the output signals to the input of the interrupt level & edge detection logic. In loopback mode, the state of the output signals is also reflected in the state of the input registers.

9.3.1 I/O Control

The I/O Control sub-module drives the gpioOut outputs from the GPIO module and handles the synchronization of gpioIn by standard double-registering of these signals to the system clock. The output of the synchronization block drives the interrupt generation logic in the case where loopback mode is not selected.

9.4 Register Descriptions

Note that many registers are one-shot. That means they have a different set & reset vectors. This is done so software doesn't need to keep a local copy of the current state in order to decide whether to set or reset bits. It also allows for different processes or threads to control different I/O's without interference or concern about what might happen if an interrupt occurs.

9.4.1 State Register (offset: 0x00)

Signal	Bits	Default	R/W	Description
State	31:0	x0	RO	Read the current state of I/O signals. For each I/O, the synchronized input signal is read regardless of whether the I/O is configured as input or output.

9.4.2 Set Register (offset: 0x00)

Signal	Bits	Default	R/W	Description
Set	31:0	x0	WO	Each bit that is set = 1 sets the corresponding output signal.

9.4.3 Clear Register (offset: 0x04)

Signal	Bits	Default	R/W	Description
Clear	31:0	x0	WO	Each bit that is set = 1 clears the corresponding output signal.

9.4.4 Interrupts Register (offset: 0x04)

Signal	Bits	Default	R/W	Description
Interrupts	31:0	x0	RO	Read to see the current state of interrupts.

9.4.5 Out Register (offset: 0x08)

Signal	Bits	Default	R/W	Description
Out	31:0	xFFFF_ FFFF	R/W	Each bit that is set = 1 configures the corresponding signal as an output. Read for current state of all active-low output enable signals. All signals default to inputs.

9.4.6 In Register (offset: 0x0C)

Signal	Bits	Default	R/W	Description
In	31:0	xFFFF_ FFFF	R/W	Each bit that is set t= 1 configures the corresponding signal as an input. Read for current state of all active low output enable signals. All signals default to inputs.

9.4.7 Enable Set Register (offset: 0x10)

Signal	Bits	Default	R/W	Description
Enable_Set	31:0	x0	WO	Each bit that is set = 1 enables an interrupt for the corresponding signal.

9.4.8 Enable Clear Register (offset: 0x14)

Signal	Bits	Default	R/W	Description
Enable_Clr	31:0	x0	WO	Each bit that is set = 1 disables an interrupt for the corresponding signal.

9.4.9 Edge Register (offset: 0x18)

Signal	Bits	Default	R/W	Description
Edge	31:0	x0	WO	Each bit that is set = 1 indicates that the interrupt has been set to edge-sensitive.

9.4.10 Level Register (offset: 0x1C)

Signal	Bits	Default	R/W	Description
Level	31:0	x0	WO	Each bit that is set = 1 indicates that the interrupt has been set to level-sensitive.

9.4.11 Set Level Register (offset: 0x20)

Signal	Bits	Default	R/W	Description
Set_Level	31:0	x0	WO	Write 1's to set interrupts for active-high or rising edge.

9.4.12 Clear Level Register (offset: 0x24)

Signal	Bits	Default	R/W	Description
Clr_Level	31:0	x0	WO	Write 1's to set interrupts for active-low or falling edge.

9.4.13 Any Edge Set Register (offset: 0x28)

Signal	Bits	Default	R/W	Description
AnyEdge_Set	31:0	x0	WO	Write 1's to override interrupt edge selection & instead interrupt on ANY edge.

9.4.14 Any Edge Clear Register (offset: 0x2C)

Signal	Bits	Default	R/W	Description
AnyEdge_Clr	31:0	x0	WO	Write 1's to clear edge-sensitive override.

9.4.15 Interrupt Clear Register (offset: 0x30)

Signal	Bits	Default	R/W	Description
Interrupt_Clr	31:0	x0	WO	Write 1's to clear edge-sensitive interrupts.

9.4.16 Control Register (offset: 0x34)

Signal	Bits	Default	R/W	Description
reserved	31:1			
Loopback	0	b0	R/W	Loopback bit. Write 1 to place GPIO in loopback mode. This internally assigns the gpioOut to the synchronized version of gpioIn. Read for the state of loopback. 1: loopback mode 0: normal operation.

9.4.17 Inhibit In Register (offset: 0x38)

Signal	Bits	Default	R/W	Description
Inhibit_In	31:0	x0	R/W	Each bit set to 1 inhibits the corresponding input signal. Read for current state of all active-low output enable signals. All inputs default to enabled. Note that the enable signal is just another output and must be used outside this module to be effective.

9.4.18 Allow In Register (offset: 0x3C)

Signal	Bits	Default	R/W	Description
Allow_In	31:0	x0	R/W	Each bit set to 1 allows/enables the corresponding input signal. Read for current state of all active-low output enable signals. All inputs default to enabled.

9.4.19 De-bounce Enable (offset: 0x40)

Signal	Bits	Default	R/W	Description
DEB_EN	31:0	x0	R/W	Each bit set to 1 enables de-bounce for the corresponding signal.

9.4.20 De-bounce Disable (offset: 0x44)

Signal	Bits	Default	R/W	Description
DEB_DIS	31:0	x0	R/W	Each bit set to 1 disables de-bounce for the corresponding signal.

9.4.21 De-bounce Time (offset: 0x48)

Signal	Bits	Default	R/W	Description
reserved	31:3			
DEB_SEL	2:0	b0	R/W	de-bounce time selection 000: 32 slow clocks 001: 64 slow clocks 010: 128 slow clocks 011: 256 slow clocks 100: 512 slow clocks 101: 1024 slow clocks 110: 2048 slow clocks 111: 4096 slow clocks

10 RTC

The Real Time Clock (RTC) is a clock-calendar IP core that keeps track of real Time of Day.

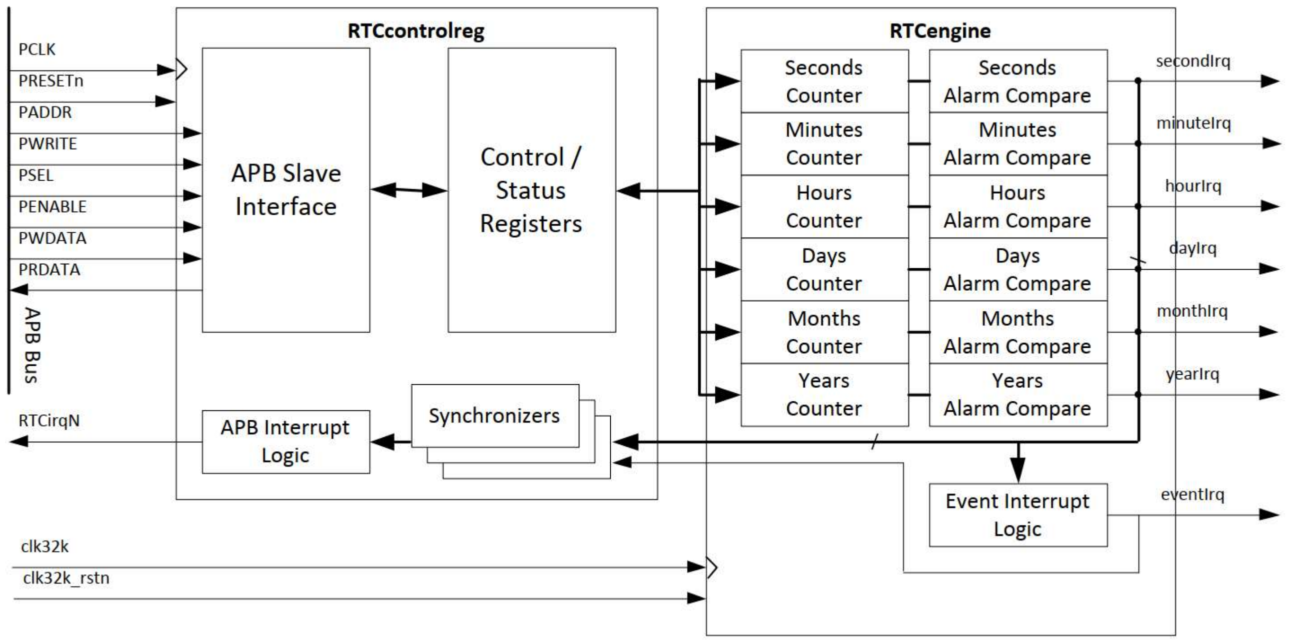
10.1 Features

- Clock / Calendar (BCD Format)
  - Seconds
  - Minutes
  - Hours
  - Days

- Months
- Years

- Alarm Mode for each "Time Unit"
- Repeat Alarm Mode for each "Time Unit"
- AMBA APB Interface
- Interrupt control for Alarms

10.2 Block Diagram



10.3 Functional Description

The RTC is a clock-calendar IP core tracking of the "Time of Day". This core is organized as a series of BCD (Binary Coded Decimal) counters for Seconds, Minutes, Hours, Days, Months and Years (ie. "Time Units").

The RTC Seconds counter’s time base is generated from a clock input that is separate from the system clock. The time base for the seconds counter is 32 kHz. To aid precision, there is programmable terminal count (divisor) for this clock. The terminal count should be set to generate a 1Hz clock enable for a signal to the seconds counter. This terminal count can also be used to adjust the clock accuracy dynamically by periodically adding or subtracting values from the nominal value. Calibration accuracy is controlled by software.

Each of the Time Units can be loaded by writing BCD information to the proper Time Unit register. The RTC will immediately load the new time and continue to count.

The IP core can also be loaded with an Alarm Compare value for each Time Unit that will generate an interrupt when that Time Unit reaches the compare value. A Repeat Alarm function is also available. When enabled, this function will cause an interrupt at the terminal count of each Time Unit counter. For example, if the Repeat Alarm bit (bit 7 or Seconds Alarm Register) for the Seconds counter is set, the RTC engine will generate an interrupt every second when the counter rolls over from 59 to 00.

A "Load" Register aids synchronization between the two clock domains (system clock and 32kHz clock). Writing to this register will produce 1-clock-wide pulse(s) on the 32kHz clock so that all values for the current time registers, alarm value registers, and/or the clock divisor register are updated safely from the APB interface registers.

testMode bits for each Time Unit counter will enable the counter to be clocked by the secondsClk in order to speed testability. This can also be used to incrementally update the RTC.

The *rtcClrPls* bit enables resetting the RTC counters (synchronous clear in the clk32k domain) under software control. Otherwise the RTC counters are free running and do not reset upon system (APB) reset.

The RTC operates as a slave device on the AMBA APB bus. Each register is accessed by simple APB bus transactions.

10.4 Register Descriptions

Note: In general, the seconds, minutes, etc. values and alarm registers are BCD (Binary Coded Decimal).

- Upper bits [N+4:4] represent a digit in the 10's place: (0-9) * 10
- Lower Bits [3:0] represent a digit in the 1's place: (0-9)

10.4.1 Seconds Value (offset: 0x00)

Signal	Bits	Default	R/W	Description
reserved	31:7			
SEC_00_50	6:4	b0	R/W	Tens place of seconds (BCD)
SEC_0_9	3:0	b0	R/W	Units place of seconds (BCD)

10.4.2 Minutes Value (offset: 0x04)

Signal	Bits	Default	R/W	Description
Reserved	31:7			
MIN_00_50	6:4	b0	R/W	Tens place of minutes (BCD)
MIN_0_9	3:0	b0	R/W	Units place of minutes (BCD)

10.4.3 Hours Value (offset: 0x08)

Signal	Bits	Default	R/W	Description
Reserved	31:6			
HR_00_20	5:4	b0	R/W	Tens place of hours (BCD)

HR_0_9	3:0	b0	R/W	Units place of hours (BCD)

10.4.4 Days Value (offset: 0x0C)

Signal	Bits	Default	R/W	Description
Reserved	31:6			
DAY_00_30	5:4	b0	R/W	Units place of days (BCD)
DAY_0_9	3:0	b0	R/W	Units place of days (BCD)

10.4.5 Months Value (offset: 0x10)

Signal	Bits	Default	R/W	Description
Reserved	31:5			
MONTH_00_10	4	b0	R/W	Tens place of months (BCD)
MONTH_0_9	3:0	b0	R/W	Units place of months (BCD)

10.4.6 Years Value (offset: 0x14)

Signal	Bits	Default	R/W	Description
Reserved	31:8			
YEAR_00_90	7:4	b0	R/W	Tens place of years (BCD)

YEAR_0_9	3:0	b0	R/W	Units place of years (BCD)

10.4.7 RTC Control (offset: 0x18)

Signal	Bits	Default	R/W	Description
Reserved	31:8			
rtcClrPls	7	b0	R/W	When 1 is written, a pulse is created on the clk32k domain to clear all counters & values to their initial state. Writing 0 to this bit has no effect. Read values are 1: clear in progress 0: clear completed
Reserved	6			
testMode	5:0	b0	R/W	This can be used to speed up Time Unit counters for test. If a bit is set to 1, then the corresponding time unit counter will change on the same timing as Seconds. [0]=Unused, [1]=Minutes, [2]=Hours, [3]=Days, [4]=Months, [5]=Years.

10.4.8 Clock Divisor (offset: 0x1C)

Signal	Bits	Default	R/W	Description

Reserved	31:16			
clkDiv	15:0	d31999	R/W	Clock Divisor for generating a 1Hz enable pulse to the Seconds Counter. The Clock Divisor value (D) and the input clock frequency (F) are related by the following equation: D = F - 1

10.4.9 Seconds Alarm (offset: 0x20)

Alarm Compare value is used to generate interrupt(s) on a seconds counter match.

Signal	Bits	Default	R/W	Description
Reserved	31:10			
SECA_IRQ_REPEAT	9	b0	R/W	When set, configures the event interrupt to trigger every second. When clear, seconds = seconds alarm for the event interrupt to be triggered.
SECA_IRQ_EVENT	8	b0	R/W	When set, configures the seconds interrupt to trigger every second. When clear, the seconds interrupt will be triggered once per minute when seconds count = seconds alarm.
reserved	7			
SECA_00_50	6:4	b0	R/W	Tens place of seconds alarm (BCD)
SECA_0_9	3:0	b0	R/W	Units place of seconds alarm (BCD)

10.4.10 Minutes Alarm (offset: 0x24)

Alarm Compare value used to generate interrupt(s) based on minutes counter match.

Signal	Bits	Default	R/W	Description

reserved	31:10			
MINA_IRQ_REPEAT	9	b0	R/W	Note: This field is relevant only when SECA_IRQ_REPEAT is NOT set. When set, the event interrupt is configured to trigger every minute when seconds = seconds alarm. When clear, seconds = seconds alarm AND minutes = minutes alarm is required for the event interrupt to be triggered.
MINA_IRQ_EVENT	8	b0	R/W	When set, configures the minutes interrupt to trigger on the first second in each new minute. When clear, the minutes interrupt will be triggered once per hour on the first second of the minute in which minutes = minutes alarm.
reserved	7			
MINA_00_50	6:4	b0	R/W	Tens place of minutes alarm (BCD)
MINA_0_9	3:0	b0	R/W	Units place of minutes alarm (BCD)

10.4.11 Hours Alarm (offset: 0x28)

Alarm Compare value used to generate interrupt(s) on an hours counter match.

Signal	Bits	Default	R/W	Description
reserved	31:10			
HRA_IRQ_REPEAT	9	b0	R/W	Note: This field is relevant only when MINA_IRQ_REPEAT and SECA_IRQ_PEPEAT are NOT set. When set, the event interrupt is configured to trigger every hour when seconds = seconds alarm AND minutes = minutes alarm. When clear, seconds = seconds alarm AND

				minutes = minutes alarm AND hours = hours alarm is required for the event interrupt to be triggered.
HRA_IRQ_EVENT	8	b0	R/W	When set, the hours interrupt is configured to trigger on the first second in each new hour. When clear, the hours interrupt will be triggered once per day on the first second of the hour when hours = hours alarm.
reserved	7:6			
HRA_00_20	5:4	b0	R/W	Tens place of hours alarm (BCD)
HRA_0_9	3:0	b0	R/W	Units place of hours alarm (BCD)

10.4.12 Days Alarm (offset: 0x2C)

Alarm Compare value used to generate interrupt(s) on a days counter match.

Signal	Bits	Default	R/W	Description
reserved	31:10			
DAYA_IRQ_REPEAT	9	b0	R/W	Note: This field is relevant only when HRA_IRQ_REPEAT, MINA_IRQ_REPEAT and SECA_IRQ_REPEAT are NOT set. When set, the event interrupt is configured to trigger every day when seconds = seconds alarm AND minutes = minutes alarm AND hours = hours alarm. When clear, seconds = seconds alarm AND minutes = minutes alarm AND hours = hours alarm AND days = days alarm is required for the event interrupt to be triggered.
DAYA_IRQ_EVENT	8	b0	R/W	When set, the days interrupt is configured to trigger on the first second in each new day. When clear, the days interrupt will be triggered once per month on the first second of the day in which days = days alarm.

reserved	7:6			
DAYA_00_30	5:4	b0	R/W	Tens place of days alarm (BCD)
DAYA_0_9	3:0	b0	R/W	Units place of days alarm (BCD)

10.4.13 Months Alarm (offset: 0x30)

Alarm Compare value used to generate interrupt(s) on a months counter match.

Signal	Bits	Default	R/W	Description
reserved	31:10			
MONTHA_IRQ_REPEAT	9	b0	R/W	Note: This field is relevant only when DAYA_IRQ_REPEAT, HRA_IRQ_REPEAT, MINA_IRQ_REPEAT and SECA_IRQ_REPEAT are NOT set. When set, the event interrupt is configured to trigger every month when seconds = seconds alarm AND minutes = minutes alarm AND hours = hours alarm AND days = days alarm. When clear, seconds = seconds alarm AND minutes = minutes alarm AND hours = hours alarm AND days = days alarm AND months = months alarm is required for the event interrupt to be triggered.
MONTHA_IRQ_EVENT	8	b0	R/W	When set, the months interrupt is configured to trigger on the first second in each new month. When clear, the months interrupt will be triggered once per year on the first second of the month when months = months alarm.
reserved	7:5			
MONTHA_00_10	4	b0	R/W	Tens place of months alarm (BCD)
MONTHA_0_9	3:0	b0	R/W	Units place of months alarm (BCD)

10.4.14 Years Alarm (offset: 0x34)

Alarm Compare value used to generate interrupt(s) on a years counter match.

Signal	Bits	Default	R/W	Description
reserved	31:10			
YEARA_IRQ_REPEAT	9	b0	R/W	Note: This field is relevant only when MONTHA_IRQ_REPEAT, DAYA_IRQ_REPEAT, HRA_IRQ_REPEAT, MINA_IRQ_REPEAT and SECA_IRQ_REPEAT are NOT set. When set, the event interrupt is configured to trigger every year when seconds = seconds alarm AND minutes = minutes alarm AND hours = hours alarm AND days = days alarm AND months = months alarm. When clear, seconds = seconds alarm AND minutes = minutes alarm AND hours = hours alarm AND days = days alarm AND months = months alarm AND years = years alarm is required for the event interrupt to be triggered.
YEARA_IRQ_EVENT	8	b0	R/W	When set, configures the years interrupt to trigger on the first second in each new year. When clear, the years interrupt will be triggered once per century on the first second of the year in which years = years alarm.
YEARA_00_90	7:4	b0	R/W	Tens place of years alarm (BCD)
YEARA_0_9	3:0	b0	R/W	Units place of years alarm (BCD)

10.4.15 Interrupt Control (offset: 0x38)

Enable or disable various interrupt conditions. Write 1 to enable, write 0 to disable.

Signal	Bits	Default	R/W	Description
reserved	31:7			
INT_EVENT_EN	6	b0	R/W	Enable Event Alarm interrupt.
INT_YEAR_EN	5	b0	R/W	Enable Year Alarm interrupt.
INT_MONTH_EN	4	b0	R/W	Enable Month Alarm interrupt.
INT_DAY_EN	3	b0	R/W	Enable Day Alarm interrupt.
INT_HOUR_EN	2	b0	R/W	Enable Hour Alarm interrupt.
INT_MIN_EN	1	b0	R/W	Enable Minute Alarm interrupt.
INT_SEC_EN	0	b0	R/W	Enable Second Alarm interrupt.

10.4.16 Interrupt Status (offset: 0x3C)

Read the current interrupt status related to the various interrupting conditions.

Signal	Bits	Default	R/W	Description
reserved	31:7			
INT_EVENT_ST	6	b0	RO	Event Alarm interrupt status.
INT_YEAR_ST	5	b0	RO	Year Alarm interrupt status.
INT_MONTH_ST	4	b0	RO	Month Alarm interrupt status.
INT_DAY_ST	3	b0	RO	Day Alarm interrupt status.

INT_HOUR_ST	2	b0	RO	Hour Alarm interrupt status.
INT_MIN_ST	1	b0	RO	Minute Alarm interrupt status.
INT_SEC_ST	0	b0	RO	Second Alarm interrupt status.

10.4.17 Interrupt Clear (offset: 0x40)

Clear the current interrupt status related to the various interrupting conditions. Write 1 to clear.

Signal	Bits	Default	R/W	Description
reserved	31:7			
INT_EVENT_CLR	6	b0	WO	Clear Event Alarm interrupt.
INT_YEAR_CLR	5	b0	WO	Clear Year Alarm interrupt.
INT_MONTH_CLR	4	b0	WO	Clear Month Alarm interrupt.
INT_DAY_CLR	3	b0	WO	Clear Day Alarm interrupt.
INT_HOUR_CLR	2	b0	WO	Clear Hour Alarm interrupt.
INT_MIN_CLR	1	b0	WO	Clear Minute Alarm interrupt.
INT_SEC_CLR	0	b0	WO	Clear Second Alarm interrupt.

10.4.18 Load (offset: 0x44)

After configuring the desired Value, Alarm and/or Divisor registers, write 1's to these bits to update (or load) these values onto the clk32k domain (writing 0 to these bits has no effect). Do not power-down the APB domain while any bits in this register are read as 1.

Signal	Bits	Default	R/W	Description
reserved	31:3			

IdDivisor	2	b0	R/W	Write 1 to update the clock divisor used on the 32kHz domain. For read, it is 1: divisor update in progress 0: divisor update done
IdAlarm	1	b0	R/W	Write 1 to update all alarm values on the 32kHz domain. For read, it is 1: alarm value update in progress 0: alarm value update done
IdTime	0	b0	R/W	Write 1 to update all time values on the 32kHz domain. For read, it is 1: time value update in progress 0: time value update done

11 UART

The UART contains the following main sections:

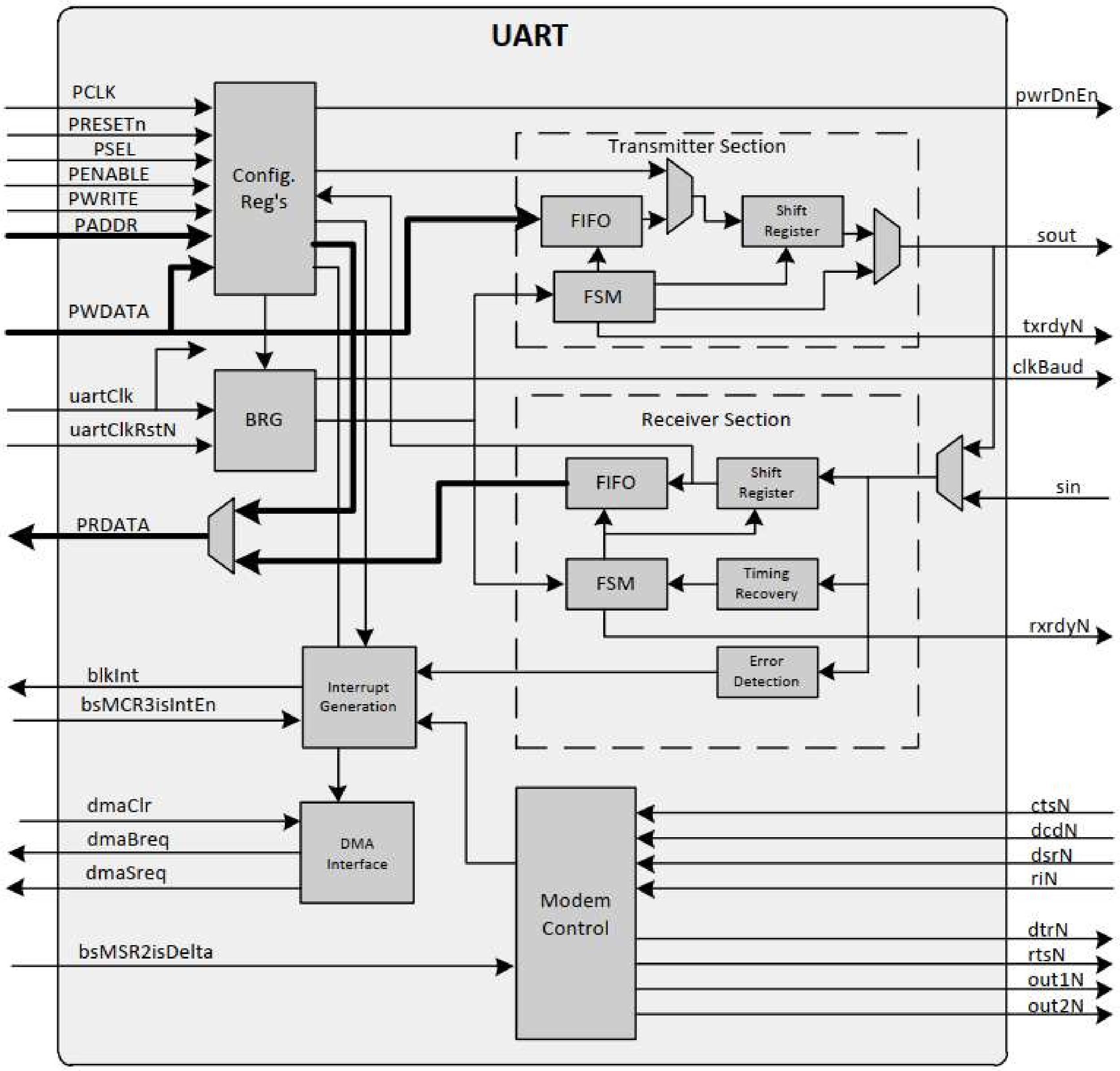
- Configuration Registers
- Baud Rate Generator
- Transmitter
- Receiver
- Interrupt Generation Logic
- Modem Control Logic

Configuration registers are written & read by the processor. The baud rate generator produces timing strobes at the baud rate (for the transmitter) and at 8 times the selected baud rate (for the receiver). The UART clock can be asynchronous to the APB clock. The receiver examines the incoming data and uses the first edge of the start bit to determine the bit timing. The transmit and receive paths can be configured to use a single register for data or to use FIFOs. Finite State Machines (FSMs) control the transmit and receive sections. Various error conditions can cause an interrupt to be generated.

11.1 Features

- 16450 and 16550 Compatible Modes
- Modem control
- Programmable baud rates
- FIFO and non-FIFO modes
- Transmit and Receive FIFOs
- Interrupt control

11.2 Block Diagram


UART
PCLK
PRESETn
PSEL
PENABLE
PWRITE
PADDR
PWDATA
Config. Reg's
pwrDnEn
Transmitter Section
FIFO
Shift Register
FSM
sout
txrdyN
clkBaud
uartClk
uartClkRstN
BRG
Receiver Section
FIFO
Shift Register
sin
PRDATA
FSM
Timing Recovery
rxrdyN
Error Detection
blkInt
bsMCR3isIntEn
Interrupt Generation
dmaClr
dmaBreq
dmaSreq
DMA Interface
Modem Control
ctsN
dcdN
dsrN
riN
dtrN
rtsN
out1N
out2N
bsMSR2isDelta

11.3 Functional Description

11.3.1 Transmitter Operation

The UART Transmitter section is relatively simple. It consists of the transmit holding register (THR), the transmit FIFO, a shift register, a state machine and a multiplexor. In FIFO mode, writing to the THR causes the THR value to be written into the FIFO. If there is no transmission in progress, transmission is started. The state machine controls the shift register and mux to ensure the bits get transmitted in the proper sequence. The baud rate generator controls the pace of the transmission. Parity is calculated if required and inserted into the bit stream.

11.3.2 Receiver Operation

The receiver is more complicated than the transmitter. Since an independent UART is transmitting the signal, timing has to be recovered from the incoming serial bit stream. The falling edge of the start bit triggers a counter that counts into the middle of the bit time programmed into the baud rate generator. Thereafter, *midBit* strobes are generated which occur in the middle of the bit time and control the timing for the duration of the received word. The timing of the far end transmitter can be substantially different from that of the receiver and the word will be successfully received as long as the cumulative error across the word (including start and stop bits) does not exceed one half of the bit time.

A state machine sequences the receive operation. When all bits are received, parity is calculated and compared to the expected parity. Parity errors, framing errors and break conditions are associated with a received data word and are written into the FIFO along with the data. A framing error is defined as the absence of an expected stop bit. A break condition is declared when too many space symbols occur in a row. That is when the serial input stays in the logic 0 state for a number of bit times greater than the sum of the number of start, stop, parity and data bits.

11.3.3 Baud Rate Generator

The baud rate generator can be configured to generate a wide range of baud rates depending on the system clock frequency and the divisor latch. The divisor latch and the system clock frequency are related to the baud rate by the following expression:

$$BaudRate = \frac{SystemClockFrequency}{8 \times DivisorLatch}$$

For example, to configure the UART for a baud rate of 2400 with a system clock frequency of 32 MHz, the Divisor Latch would be decimal 1,667 (0x0683). The most significant byte of the divisor latch would be programmed 0x06 and the least significant byte of the divisor latch would be programmed 0x83. To access the divisor latch, the Divisor Latch Access Bit (*DLAB*) must be set in the Line Control Register (LCR). The divisor latch can be programmed with values ranging 0x0001 to 0xFFFF.

11.3.4 Interrupts

The UART can generate five types of interrupts: receiver line status, received data available, character timeout, transmitter holding register empty and modem status. Any of these interrupts can activate the UART interrupt request if they are enabled.

Enabling these interrupts is controlled by the setting & clearing bits in the Interrupt Enable Register (IER). A set bit in the IER corresponds to an enabled interrupt and a clear bit in the IER corresponds to a disabled interrupt.

The UART will activate its interrupt signal any time one of the five interrupt sources is active and enabled by the appropriate bit in the IER.

IER Bit Number	Corresponding Interrupt Enabled
0	Received Data Available
0	Character Timeout (only when FIFOs are enabled)
1	Transmitter Holding Register Empty
2	Receiver Line Status
3	Modem Status

11.3.4.1 Receiver Line Status

This interrupt occurs when any one of the following conditions exist: framing error, parity error, overrun error or break interrupt. This interrupt is cleared by reading the line status register.

10.3.4.2 Received Data Available

When FIFOs are disabled, this interrupt occurs when the Receiver Buffer Register (RBR) contains received data. Reading the RBR clears this interrupt.

When FIFOs are enabled, this interrupt occurs when the amount of data in receiver FIFO exceeds the trigger level (controlled by FCR[7:6]). This interrupt is cleared when the amount of data in the receiver FIFO drops below the trigger level. This may require multiple reads of the RBR content.

11.3.4.2 Character Timeout

This interrupt only occurs when FIFOs are enabled and no characters have been written to or read from the FIFO in the last 4 character times. Each character time consists of the start bit, data bits, parity bit (if enabled) and stop bit(s). This interrupt is cleared by reading the RBR.

11.3.4.3 Transmitter Holding Register (THR) Empty

When FIFOs are disabled, this interrupt occurs when a character from the THR is loaded into the transmitter shift register indicating that the THR is ready to accept a new character.

When FIFOs are enabled, this interrupt occurs when the transmitter FIFO becomes empty.

In either case, this interrupt is cleared when the THR is read or when the Interrupt Identification Register IIR was read when *THRE* (Transmitter Holding Register Empty) was the source of the interrupt.

Note that reading the IIR clears the *THRE* interrupt, however the THRE bit in the LSR will remain set until a character has been loaded into the THR or transmitter FIFO.

11.3.4.4 Modem Status

The modem status interrupt occurs when any of the following bits in the LSR are set: delta clear to send (*DCTS*), delta data set ready (*DDSR*), trailing edge ring indicator (*TERI*) or delta data carrier detect (*DDCD*). This interrupt is cleared by reading the Modem Status Register (MSR).

Be aware that in loopback mode these “delta” bits are not controlled by the modem inputs but rather by the lower four bits of the Modem Control Register (MCR).

11.3.5 Interrupt Identification

Interrupts generated by the UART are distinguished by the value in the least significant nibble of the Interrupt Identification Register (IIR). Bit 0 of the IIR can be tested to determine if there is an interrupt pending.

Priority	IIR [3:0]	Interrupt Type
n/a	0001	No Interrupt Pending
first	0110	Receiver Line Status

second	0100	Received Data Available
second	1100	Character Timeout
third	0010	Transmitter Holding Register Empty
fourth	0000	Modem Status

11.3.6 FIFO Control

FIFOs are enabled by setting bit 0 in the FIFO Control Register (FCR). When FIFOs are enabled, bits 7 and 6 of the IIR are set, otherwise these bits are clear. The FIFOs are asynchronous to accommodate differences in clock rates between *uartClk* & *PCLK*.

11.3.6.1 Receiver FIFO

The receiver FIFO stores received data until it is read by software. The depth of the receiver FIFO is 16 bytes. Once the programmed trigger level has been reached within the FIFO, the data ready bit (and received data available interrupt, if enabled) will be set. Bits 7 and 6 of the FCR register control the receiver trigger level.

FCR[7:6]	Receiver FIFO Trigger Level (Bytes)
00	1
01	4
10	8
11	12

The receiver FIFO can be reset by setting bit 1 in the FCR. This bit is self-clearing.

11.3.6.2 Transmitter FIFO

The transmitter FIFO is used to store data that is intended to be transmitted. The depth of the transmitter FIFO is 16 bytes. Each byte written to the THR location is loaded into the transmitter FIFO. The transmitter FIFO can be reset by setting bit 2 in the FCR. This bit is self-clearing.

11.3.7 Line Control

11.3.7.1 Asynchronous Serial Data Format

The line control register (LCR) specifies the format of the asynchronous data that is transmitted and received by the UART. The number of data bits in each serial character is specified by bits 1 & 0 of the LCR.

LCR[1:0]	Character Length
00	5 Bits
01	6 Bits
10	7 Bits
11	8 Bits

The number of stop bits in each serial character is specified by bit 2 of the LCR along with the current character length selected by bits 1 & 0 of the LCR.

LCR[2]	Number of Stop Bits
0	1
1	2 *

* Note that this is different from a standard 16550 implementation. When the character length is 5 bits, the number of stop bits should be 1.5. For simplicity, this implementation generates 2 stop bits.

Parity bit generation is controlled by bits 4 & 3 of the LCR. Bit 3 enables parity and bit 4 selects even or odd parity.

LCR[4:3]	Type of Parity
X0	No Parity
01	Odd Parity
11	Even Parity

Even parity sets the parity bit to 1 when there are an even number of ‘1’s in the serial character. The parity bit is set to 0 when there are an odd number of ‘1’s in the serial character. Odd parity sets the parity bit to 1 when there are an odd number of ‘1’s in the serial character and sets the parity bit to 0 when there are an even number of ‘1’s in the serial character.

11.3.7.2 Diagnostic Mechanisms

The line control register (LCR) also provides mechanisms to force break conditions, as well as parity and framing error conditions. Utilizing these mechanisms while in loopback mode allows diagnostic checking of receiver line logic.

Bit 5 of the LCR is the Stick Parity bit. Stick Parity causes the transmitter to force the parity bit of a transmitted serial character to 1 or 0 regardless of the calculated parity of the transmitted serial character. Stick Parity also causes the receiver to check the parity bit of a received serial character against either 1 or 0 regardless of the calculated parity of the received serial character. It is the Stick Parity bit in conjunction with bits 4 and 3 of the LCR which determine how the parity bit is affected.

The following table summarizes all possible combinations of parity functionality:

LCR[5]	LCR[4:3]	Type of Parity
0	00	parity disabled, no parity bit generated/checked

0	01	odd parity generated/checked
0	10	parity disabled, no parity bit generated/checked
0	11	even parity generated/checked
1	00	parity disabled, no parity bit generated/checked
1	01	stick parity, generated/checked as logic ‘1’
1	10	parity disabled, no parity bit generated/checked
1	11	stick parity, generated/checked as logic ‘0’

Bit 6 of the LCR is the Set Break bit. Setting this bit forces the *sout* low (spacing) condition. The *sout* line will remain in the spacing condition until this bit is cleared.

Bit 7 of the LSR is the Divisor Latch Access Bit (*DLAB*). Please see the section on the Baud Rate Generator for a description of the use of this bit.

11.3.8 Line Status

Bit 0 of the LSR is the Data Ready (*DR*) bit. This bit is set when an incoming character is completely received into the RBR or receiver FIFO and it is cleared when the character from the RBR is read or the receiver FIFO has been emptied.

Bit 1 of the LSR is the overrun error indicator. Overrun errors occur in 16450 mode when both the RBR & receiver shift register contain data and another incoming character is received, destroying the character in the receiver shift register. The character in the RBR is not corrupted.

In 16550 mode, overrun errors occur when the receiver FIFO is completely full, the receiver shift register contains a character and another incoming character is received destroying the character in the receiver shift register. The data in the receiver FIFO is not corrupted.

In both modes, the overrun error indicator bit is set as soon as the start bit of the character causing the overrun is detected.

Bit 2 of the LSR is the parity error indicator. This bit is set when a received character’s parity (as calculated by the receiver) does not match the value of its received parity bit. In 16550 mode, this bit is set only when the offending character is at the top of the FIFO. Reading the LSR clears this bit.

Bit 3 of the LSR is the framing error indicator. This bit is set when the receiver does not detect a valid stop bit for an incoming character, i.e. the serial input signal was low during the baud time in which a stop bit (high) was expected. In 16550 mode, this bit is set only when the offending character is at the top of the FIFO. Reading the LSR clears this bit.

Bit 4 of the LSR is the break interrupt indicator. This bit is set whenever the serial input is held in the space (logic 0) state for more time than a complete character transfer (i.e. the time it takes to transmit a start bit, the data bits, the parity bit if enabled and any stop bits). If a break is detected in 16550 mode, only one break character (0x00) is loaded into the receiver FIFO. Reading the LSR clears this bit.

Bit 5 of the LSR is the Transmitter Holding Register Empty bit. When FIFOs are disabled, this bit is set when a character from the THR is loaded into the transmitter shift register, indicating that the THR is ready to accept a new character. It is cleared when a new character is written to the THR.

When FIFOs are enabled, this bit is set when the transmitter FIFO becomes empty and is cleared after at least one character is loaded into the transmitter FIFO.

Bit 6 of the LSR is the Transmitter Empty Bit. This bit is set when both the THR and the transmitter shift register are empty. This bit is clear when either the THR contains a character or the transmitter shift register has not completed shifting-out a character being transmitted.

Bit 7 of the LSR indicates that there is at least one error in the receiver FIFO. In 16450 mode, this bit is always clear. In 16550 mode, this bit is set when there is at least one framing error, parity error, or break indication in the receiver FIFO. Reading the LSR clears this bit.

11.3.9 Modem Control / Status

The Modem Control and Modem Status Registers (MCR and MSR) are used to control and monitor the modem related I/O: Clear to Send, Data Set Ready, Ring Indicator, Data Carrier Detect, Data Terminal Ready and Request to Send.

11.3.9.1 Modem Control

Bits 0 and 1 of the MCR directly affect the state of the Data Terminal Ready and Request to Send outputs (*dtrN* and *rtsN*). Setting bit 0 of the MCR causes the *dtrN* output to be asserted (low) while clearing bit 0 of the MCR causes *dtrN* to be de-asserted (high). In a similar manner, setting bit 1 of the MCR causes the *rtsN* output to be asserted (low) and clearing bit 1 of the MCR causes *rtsN* to be de-asserted (high).

11.3.9.2 Modem Status

Bits 4-7 of the MSR indicate the current state of the Clear to Send, Data Set Ready, Ring Indicator, and Data Carrier Detect input signals, respectively (*ctsN*, *dsrN*, *riN*, *dcdN*). On the condition that any of these input signals are active (low), then the corresponding bits in the MSR will be set. When any of these inputs are in their inactive (high) state, then the corresponding bits in the MSR will be clear.

Bits 0-3 of the MSR are known as the “delta” bits which indicate whether the state of the modem input signals have changed since a previous read of the MSR. The Delta Clear to Send (*DCTS*), Delta Data Set Ready (*DDSR*), and Delta Data Carrier Detect (*DDCD*) bits in the MSR are asserted any time one of the corresponding modem inputs (*ctsN*, *dsrN*, *dcdN*) changes state.

The “delta” bit for the Ring Indicator is configured as a Delta Ring Indicator (*DRI*). Bit 2 of the MSR will be asserted any time the state of the *riN* input changes.

11.3.10 DMA modes

For DMA operation, only Mode 1 is supported and is selected by bit 3 in the FCR.

11.3.10.1 Mode 1 (16550 only)

The UART contains a DMA interface so that an external DMA controller may load and drain the FIFOs without involving a microprocessor. The timing of the signaling is a simple handshake. The UART asserts *dmaBreq[0]* when its TX FIFO falls to its watermark level (or below), and asserts *dmaBreq[1]* when its RX FIFO level rises to its watermark level or above. These signals stay asserted until the DMA controller has performed the appropriate number of reads or writes (indicated by the RX/TX watermark level) and asserts the *dmaClr[x]* signal. When the DMA Controller has finished the writes to the TX FIFO, it asserts *dmaClr[0]*; when the reads from the RX FIFO are complete, the DMA Controller asserts *dmaClr[1]*. When the UART detects *dmaClr*, it de-asserts the corresponding *dmaBreq* signal. The handshake is complete when the DMA controller samples *dmaBreq* low and de-asserts *dmaClr*.

Here are some additional notes about DMA signaling. The UART will only assert *dmaBreq[x]* if *dmaClr[x]* is low, and will only de-assert *dmaBreq[x]* if *dmaClr[x]* is high.

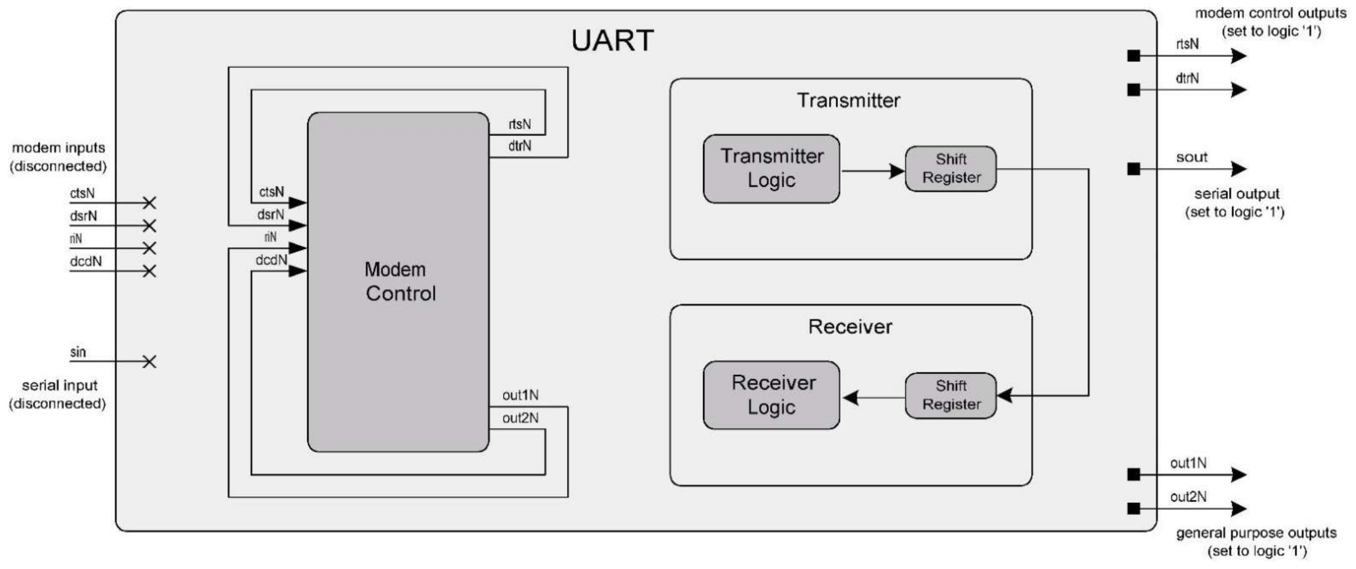
Lastly, the watermark used for both the RX and TX FIFOs is the FIFO trigger level that is set in the FCR register.

11.3.11 Loopback Operation

The UART provides loopback operation for transmitter, receiver, and modem status diagnostics. The UART operates in loopback mode when bit 4 of the MCR is set.

In loopback mode, the serial data input (*sin*) is disconnected, and the serial data output (*sout*) is driven high (marking state). The transmitter is internally connected to the receiver so that any data transmitted is also received within the UART.

The *ctsN*, *dsrN*, *riN*, and *dcdN* modem inputs are disconnected, and the *dtrN*, *rtsN*, *out1N*, and *out2N* outputs are driven high (inactive state). The modem control and general purpose outputs (*dtrN*, *rtsN*, *out1N*, *out2N*) are internally connected to the modem status inputs (*dsrN*, *ctsN*, *riN*, *dcdN*).


UART
modem inputs (disconnected)
ctsN
dsrN
riN
dcdN
sin
serial input (disconnected)
Modem Control
rtsN
dtrN
out1N
out2N
Transmitter
Transmitter Logic
Shift Register
Receiver
Receiver Logic
Shift Register
modem control outputs (set to logic '1')
sout
serial output (set to logic '1')
general purpose outputs (set to logic '1')

The modem status interrupt can still be generated, although in loopback mode it is controlled through the least four significant bits in the MCR.

For example, writing 0x11 to the MCR (asserting *DTR* bit and maintaining loopback bit) will cause the *DSR* bit to go active in the MSR (since *DTR* is connected to *DSR* in loopback mode). The “delta” edge detection logic then detects the change from low to high of the *DSR* bit in the MSR, therefore causing the modem status interrupt (any asserted “delta” bit in the MSR causes the modem status interrupt). Reading the MSR will clear this interrupt.

Be aware that the option to have trailing edge or delta edge detection on the ring indicator also applies in loopback mode. However, in this mode the “delta” or trailing edge will be detected on the *out1* bit in the MCR because this is connected to the Ring Indicator bit in the MSR when operating in loopback mode. This configuration allows diagnostic testing of the Trailing Edge Ring Indicator (*TERI*) and Delta Ring Indicator (*DRI*) in loopback mode by using the out1 bit of the MCR.

11.3.12 General Purpose Outputs

The *out1N* and *out2N* outputs are driven from bits 2 and 3 of the MCR, respectively. These outputs are active low, therefore setting bit 2 in the MCR will cause the *out1N* output to be driven low while clearing bit 2 in the MCR causes the *out1N* output to be driven high. The *out2N* output is controlled in a similar fashion by bit 3 of the MCR. A system reset will set these signals to their inactive state (logic '1') since bits 2 and 3 of the MCR are cleared on a system reset.

11.4 Register Descriptions

11.4.1 Summary

Register Name	Abbre.	Offset	Description
Receiver Buffer	RBR	0x00	Received Data (Read Only)
Transmitter Holding	THR	0x00	Data to be transmitted (Write Only)
Interrupt Enable	IER	0x04	Enables the five types of UART interrupts
Interrupt Identification	IIR	0x08	Four interrupts levels in order of priority: Receiver Line Status, Received Data Ready, Transmitter Holding Register Empty and MODEM Status (Read Only)
FIFO Control	FCR	0x08	Enable FIFOs, clear FIFOs, set RCVR FIFO trigger level (Write Only)
Line Control	LCR	0x0C	Specifies asynchronous data communications exchange format and sets the Divisor Latch Access Bit
MODEM Control	MCR	0x10	Controls interface with MODEM (or peripheral device emulating a MODEM)
Line Status	LSR	0x14	Data Transfer Status information

MODEM Status	MSR	0x18	Current state of control lines from MODEM (or peripheral device) to CPU
Scratch	SCR	0x1C	Register can be used to hold temporary data
Divisor Latch (LS)	DLL	0x00	DLAB = 1, Least significant byte for input to baud rate generator
Divisor Latch (MS)	DLM	0x04	DLAB = 1, Most significant byte for input to baud rate generator
xDMA_RX_Start		0x20	Start address of RX buffer in SRAM for xDMA
xDMA_RX_Len		0x24	xDMA transfer length for RX
xDMA_TX_Start		0x28	Start address of TX buffer in SRAM for xDMA
xDMA_TX_Len		0x2C	xDMA transfer length for TX
xDMA_RX_RemainLen		0x30	xDMA transfer length remain for RX
xDMA_TX_RemainLen		0x34	xDMA transfer length remain for TX
xDMA_IER		0x38	Interrupt enable for xDMA
xDMA_ISR		0x3C	Interrupt status for xDMA
xDMA_RX_Enable		0x40	xDMA enable for RX start
xDMA_TX_Enable		0x44	xDMA enable for TX start

11.4.2 Receiver Buffer Register (RBR) (offset: 0x00)

Signal	Bits	Default	R/W	Description
reserved	31:8			
RBR	7:0	x0	RO	Received Data

11.4.3 Transmitter Holding Register (THR) (offset: 0x00)

Signal	Bits	Default	R/W	Description
reserved	31:8			
THR	7:0	x0	WO	Data To be transmitted

11.4.4 Interrupt Enable Register (IER) (offset: 0x04)

Signal	Bits	Default	R/W	Description
reserved	31:4			
IER_B3	3	b0	R/W	Enable MODEM Status Interrupt
IER_B2	2	b0	R/W	Enable Receiver Line Status Interrupt
IER_B1	1	b0	R/W	Enable Transmitter Holding Register Interrupt
IER_B0	0	b0	R/W	Enable Received Data Available Interrupt

11.4.5 Interrupt Identification Register (IIR) (offset: 0x08)

Signal	Bits	Default	R/W	Description
reserved	31:8			
IIR_B7_B6	7:6	b0	RO	FIFOs Enabled
reserved	5:4			

IIR_B3_B1	3:1	b0	RO	Interrupt Identification Bits
IIR_B0	0	b0	RO	Interrupt Pending – ‘0’ if interrupt pending

11.4.6 FIFO Control Register (FCR) (offset: 0x08)

Signal	Bits	Default	R/W	Description
reserved	31:8			
FCR_B7_B6	7:6	b0	WO	Receiver Trigger Level
reserved	5:4			
FCR_B3	3	b0	WO	DMA Mode Select. Only support 16550 Mode 1.
FCR_B2	2	b0	WO	Transmit FIFO reset (self-clearing)
FCR_B1	1	b0	WO	Receive FIFO reset (self-clearing)
FCR_B0	0	b0	WO	FIFO Enable

11.4.7 Line Control Register (LCR) (offset: 0x0C)

Signal	Bits	Default	R/W	Description
reserved	31:8			

LCR_B7	7	b0	R/W	Divisor Latch Access (DLAB)
LCR_B6	6	b0	R/W	Set Break
LCR_B5	5	b0	R/W	Stick Parity
LCR_B4	4	b0	R/W	Even Parity Select
LCR_B3	3	b0	R/W	Parity Enable
LCR_B2	2	b0	R/W	Number of Stop Bits
LCR_B1_B0	1:0	b0	R/W	Word Length Select

11.4.8 Modem Control Register (MCR) (offset: 0x10)

Signal	Bits	Default	R/W	Description
reserved	31:6			
MCR_B5	5	b0	R/W	Enable CTS to block transmit data
MCR_B4	4	b0	R/W	Loopback mode select
MCR_B3_B2	3:2	b0	R/W	Auxiliary user designated output
MCR_B1	1	b0	R/W	Request to Send (RTS)
MCR_B0	0	b0	R/W	Data Terminal Ready (DTR)

11.4.9 Line Status Register (LSR) (offset: 0x14)

Signal	Bits	Default	R/W	Description
reserved	31:8			
LSR_B7	7	b0	RO	Error in Receiver FIFO

LSR_B6	6	b0	RO	Transmitter Empty
LSR_B5	5	b0	RO	Transmitter Holding Register Empty
LSR_B4	4	b0	RO	Break Interrupt
LSR_B3	3	b0	RO	Framing Error
LSR_B2	2	b0	RO	Parity Error
LSR_B1	1	b0	RO	Overrun Error
LSR_B0	0	b0	RO	Data Ready (DR)

11.4.10 Modem Status Register (MSR) (offset: 0x18)

Signal	Bits	Default	R/W	Description
reserved	31:8			
MSR_B7	7	b0	RO	Data Carrier Detect (DCD)
MSR_B6	6	b0	RO	Ring Indicator (RI)
MSR_B5	5	b0	RO	Data Set Ready (DSR)
MSR_B4	4	b0	RO	Clear To Send (CTS)
MSR_B3	3	b0	RO	Delta Data Carrier Detect (DDCD)
MSR_B2	2	b0	RO	Trailing Edge Ring Indicator (TERI)
MSR_B1	1	b0	RO	Delta Data Set Ready (DDSR)
MSR_B0	0	b0	RO	Delta Clear To Send (DCTS)

11.4.11 Scratch Register (SCR) (offset: 0x1C)

Signal	Bits	Default	R/W	Description
reserved	31:8			
SCR	7:0	x0	R/W	Scratchpad register

11.4.12 Divisor Latch (DLL) (offset: 0x00), LCR_B7(DLAB)=1

Signal	Bits	Default	R/W	Description
reserved	31:8			
DLL	7:0	x01	R/W	Divisor Latch [7:0]

11.4.13 Divisor Latch (DLM) (offset: 0x04), LCR_B7(DLAB)=1

Signal	Bits	Default	R/W	Description
reserved	31:8			
DLM	7:0	x0	R/W	Divisor Latch [15:8]

11.4.14 xDMA_RX_Start (offset: 0x20)

Signal	Bits	Default	R/W	Description
xDMA_RX_Start	31:0	x0	R/W	Start address of RX buffer in SRAM for xDMA

11.4.15 xDMA_RX_Len (offset: 0x24)

Signal	Bits	Default	R/W	Description
Reserved	31:16			
xDMA_RX_Len	15:0	x0	R/W	xDMA transfer byte length for RX

11.4.16 xDMA_TX_Start (offset: 0x28)

Signal	Bits	Default	R/W	Description
xDMA_TX_Start	31:0	x0	R/W	Start address of TX buffer in SRAM for xDMA

11.4.17 xDMA_TX_Len (offset: 0x2C)

Signal	Bits	Default	R/W	Description
Reserved	31:16			
xDMA_TX_Len	15:0	x0	R/W	xDMA transfer byte length for TX

11.4.18 xDMA_RX_RemainLen (offset: 0x30)

Signal	Bits	Default	R/W	Description
Reserved	31:16			
xDMA_RX_RemainLen	15:0	x0	RO	xDMA transfer byte length remaining for RX

11.4.19 xDMA_TX_RemainLen (offset: 0x34)

Signal	Bits	Default	R/W	Description
Reserved	31:16			
xDMA_TX_RemainLen	15:0	x0	RO	xDMA transfer byte length remaining for TX

11.4.20 xDMA_IER (offset: 0x38)

Signal	Bits	Default	R/W	Description
reserved	31:2			
xDMA_IER_TX	1	x0	R/W	Interrupt enable for xDMA TX
xDMA_IER_RX	0	x0	R/W	Interrupt enable for xDMA RX

11.4.21 xDMA_ISR (offset: 0x3C)

Signal	Bits	Default	R/W	Description
reserved	31:2			
xDMA_ISR_TX	1	x0	W1C	Interrupt status for xDMA TX
xDMA_ISR_RX	0	x0	W1C	Interrupt status for xDMA RX

11.4.22 xDMA_RX_Enable (offset: 0x40)

Signal	Bits	Default	R/W	Description
reserved	31:1			
xDMA_RX_Enable	0	x0	R/W	xDMA enable for RX start write 0 for completion or abort

11.4.23 xDMA_TX_Enable (offset: 0x44)

Signal	Bits	Default	R/W	Description
reserved	31:1			
xDMA_TX_Enable	0	x0	R/W	xDMA enable for TX start write 0 for completion or abort

12 I2C Master

This is an implementation of a standard I2C Master. The I2C peripheral contains the following main sections:

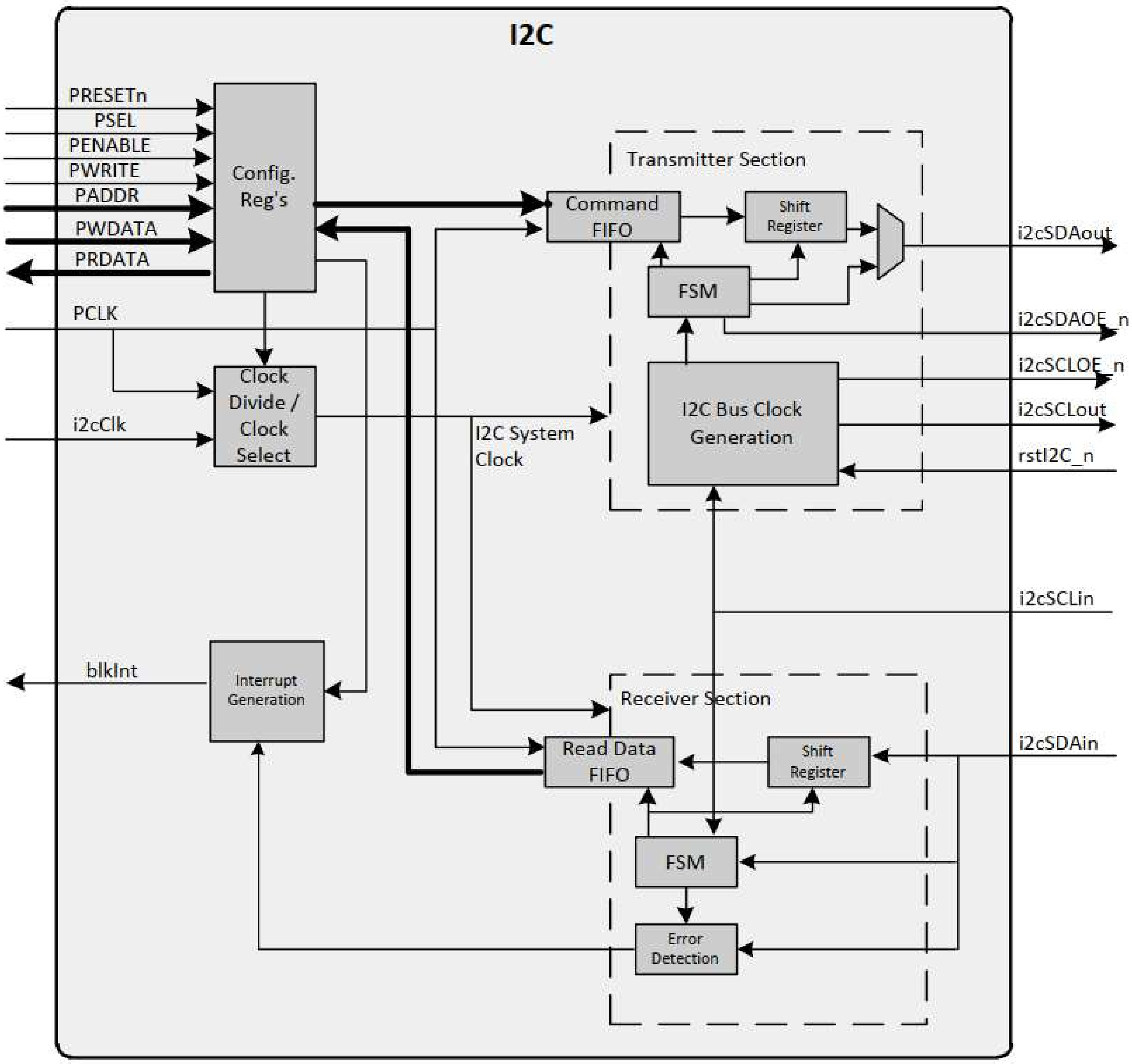
- Configuration Registers
- Clock Divider/Clock Select
- Command FIFO and Read Data FIFO
- I2C Transmit and Receive Engine
- Interrupt Generation Logic

Configuration registers are written and read by the processor via an APB Interface. The clock divider/clock select module is used to customize the frequency of the I2C portion of the module. Two separate FIFOs are used – one for storing up to 32 commands from the APB Interface, the other for storing up to 16 bytes of read data from the I2C Bus. The transmit engine reads commands from the command FIFO and implements these as I2C instructions. The receive engine monitors the I2C bus for slave responses and stores data in a Read Data FIFO, the contents of which are available to the processor on the APB Interface. Various conditions can cause an interrupt to be generated.

12.1 Features

- Standard I2C - Inter-Integrated Circuit Bus Interface
- I2C Master mode
- I2C Transmit and Receive Engine
- AMBA® APB Configuration Registers
- Clock Divider/Clock Select
- Command FIFO and Read Data FIFO
- Interrupt Generation Logic

12.2 Block Diagram



12.3 Functional Description

12.3.1 Clock Divider

The I2C Clock Select/Clock Divider module can be configured to generate a wide range of frequencies. Either the APB clock or an external clock may be chosen and divided down. The clock divider module outputs the selected clock with a clock enable to the I2C transmit & receive logic.

The formula below gives the resulting I2C System Clock Frequency as a function of Selected Clock Frequency and Clock Divide Value (from the Control and Pre-scale Registers):

$$I2CSystemClockFrequency = \frac{SelClockFrequency}{(ClockDivideValue + 1)}$$

Note that the I2C System Clock Frequency is a factor of 4 faster than the nominal I2C Bus Clock (SCL) frequency.

$$I2CBusClockFrequency = \frac{I2CSystemClockFrequency}{4} = \frac{SelClockFrequency}{(ClockDivideValue + 1) \times 4}$$

The additional factor of 4 is necessary for the I2C Engine to generate all phases of the I2C Bus Clock and to generate start & stop conditions on the I2C Bus that are synchronous to the I2C System Clock.

12.3.2 Command Interface

The Command register is the programming interface to the I2C Engine. The commands arrive at the I2C Engine via the Command FIFO, so the first valid command that is written to the Command register is the first I2C instruction implemented on the I2C bus. Because the command interface provides the basic building blocks for any I2C transaction, access to a wide range of I2C slave devices is supported.

If the Command FIFO is empty, up to 32 commands can be written to the Command Interface. It is the programmer’s responsibility to keep track of how many commands have been written to the FIFO in order to prevent a Command FIFO Overflow Error. Alternatively, the programmer may choose to poll the Status Register and check for Command FIFO Full before writing to the Command Register.

The following table shows the commands that can be used to program the I2C Engine to generate I2C events:

Command Name	Value	Tx/Rx	I2C Bus Function

I2C_CMD_NULL	0x00	N/A	This command is used at the end of an I2C command sequence. The I2C_CMD_NULL command has no effect on the I2C Bus.
I2C_CMD_WDAT0	0x10	Tx	This command is used to transmit a one-bit logic “low” condition on the I2C bus. Typically, this command is used to generate a Master Acknowledge (MACK) condition on the I2C bus.
I2C_CMD_WDAT1	0x11	Tx	This command is used to transmit a one-bit logic “high” condition on the I2C bus. Typically, this command is used to generate a Master Not Acknowledge (NACK) condition on the I2C bus.
I2C_CMD_WDAT8 **<DATA>**	0x12	Tx	The I2C_CMD_WDAT8 command sequence is used to transmit an 8-bit data value on the I2C bus. The I2C_CMD_WDAT8 command is the first command in a two-command sequence. The command directly following the I2C_CMD_WDAT8 is the 8-bit data value to be transmitted.
I2C_CMD_RDAT8	0x13	Rx	The I2C_CMD_RDAT8 command is used to alert the receiver to capture an 8-bit data value on the I2C Bus. It is expected that an I2C Slave will drive this value. Once the 8-bit data value has been captured, the I2C Engine places the value in the Read Data register and sets the Read Data FIFO Not Empty flag in the Status register.
I2C_CMD_STOP	0x14	Tx	This command is used to generate an I2C “Stop” condition on the I2C bus.
I2C_CMD_STRT	0x15	Tx	This command is used to generate an I2C “Start” condition on the I2C bus.
I2C_CMD_VACK	0x16	Rx	This command is used to alert the receiver to verify a one-bit logic “low” condition, typically a Slave Acknowledge (ACK). If a logic “high” is detected on the I2C bus, the I2C Engine sets an error flag in the status register.

I2C Protocol requires the receiving device (usually the Slave) to issue an Acknowledge after every 8-bit data transfer. The receiving device accomplishes this by driving the data line low for one I2C Clock cycle after receiving the 8-bit data value. The transmitting device will check for Acknowledge by checking for a logic “low” on the I2C Bus for the clock cycle directly following the 8-bit data transfer. The I2C Engine can check for this Acknowledge with the I2C_CMD_VACK command or produce the Acknowledge with the I2C_CMD_WDAT0 command.

12.3.3 Interrupts

The I2C can generate five types of interrupts: Command FIFO Done, Loss of Arbitration, Error, Read Data FIFO Not Empty and Command FIFO Empty.

12.3.3.1 Enabling Interrupts

Enabling and disabling interrupts is controlled by the setting & clearing bits in the Interrupt Enable Register. A logic “high” in the Interrupt Enable Register corresponds to an enabled interrupt while a logic “low” in the Interrupt Enable Register corresponds to a disabled interrupt. Note the I2C Enable bit in the Control Register must be set also in order to enable the interrupt signal.

IER Bit Number	Corresponding Interrupt Enabled
0	Command FIFO Empty
1	Read Data FIFO Not Empty
2	Error/NACK
3	Loss of Arbitration
4	Command FIFO Done

The Command FIFO Empty interrupt can be used by the processor to commence writing up to 32 commands into the command FIFO.

The Read Data FIFO Not Empty interrupt can be used to alert the processor that an I2C read has occurred and the data is available.

The Error Interrupt signifies that an error condition has occurred. The status register contains more detailed information about the cause of the error.

When a collision occurs on the I2C bus, the first master detects it has lost arbitration of the bus it was trying to drive. This is communicated to the processor with an interrupt so the processor can act accordingly.

The Command FIFO Done interrupt is used by the processor to confirm all commands in the command FIFO are issued to the bus.

12.3.3.2 Setting the Interrupt

The individual interrupt conditions are set by the Interrupt Logic in the I2C Module when it determines an interrupt condition change has been detected.

The Command FIFO Empty interrupt is set for a change from not empty to empty. By design, this interrupt condition is also set when the I2C module emerges from a reset condition.

The Read Data FIFO Not Empty Interrupt is set only on a change from empty to not empty.

The Error interrupt is set for any of the following changes:

Command FIFO Normal Operation → Command FIFO Overflow (or Underflow)

Read Data FIFO Normal Operation → Read Data FIFO Underflow (or Overflow)

I2C Protocol Normal Operation → I2C Protocol Error

The Loss of Arbitration Interrupt is set for the detection of the Loss of Arbitration.

The Command FIFO Done interrupt is set when all commands are issued to the bus.

12.3.3.3 Clearing the Interrupt

The interrupt signal to the processor is cleared by writing 1's to the Interrupt Status Register in the position of the interrupt source or sources. Note that writing to the Interrupt Status Register may only clear the interrupt condition – further action from the processor may be necessary in order to manage the cause of the interrupt.

If the Command FIFO Empty condition is the source of the interrupt, the processor may place up to 32 commands in the Command register. Or it may simply choose to set a software flag indicating that the Command FIFO is empty, if maybe there is no data ready to be written to the I2C Module at the time of the interrupt. Until the Command FIFO is written to, the status register will indicate that the Command FIFO is empty and no further interrupts due to a Command FIFO Empty condition will occur.

If the Read Data FIFO Not Empty condition is the source of the interrupt, then the processor should read the Read Data register until the Status register indicates that the Read Data FIFO is empty. Until the Read Data FIFO has been read enough times so that it is again empty, the Status register will indicate that there is Read Data present in the Read Data FIFO and no further interrupts due to a Read Data FIFO Not Empty condition will occur.

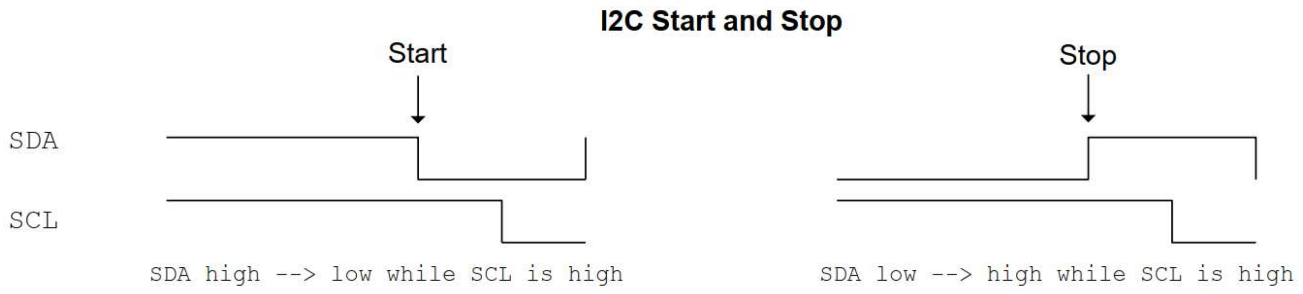
If a NACK Error condition is the source of the interrupt, after clearing the interrupt the processor should also clear the 1 in the appropriate bit of the Status register. A NACK response from a slave could indicate any number of problems, including an incorrect command sequence, a malfunctioning or incompatible Slave, an I2C Bus connection problem, or even an I2C Bus Clock frequency that is too fast for the I2C Slave. Or it could be an expected condition during polling to see what slaves are connected to the I2C bus.

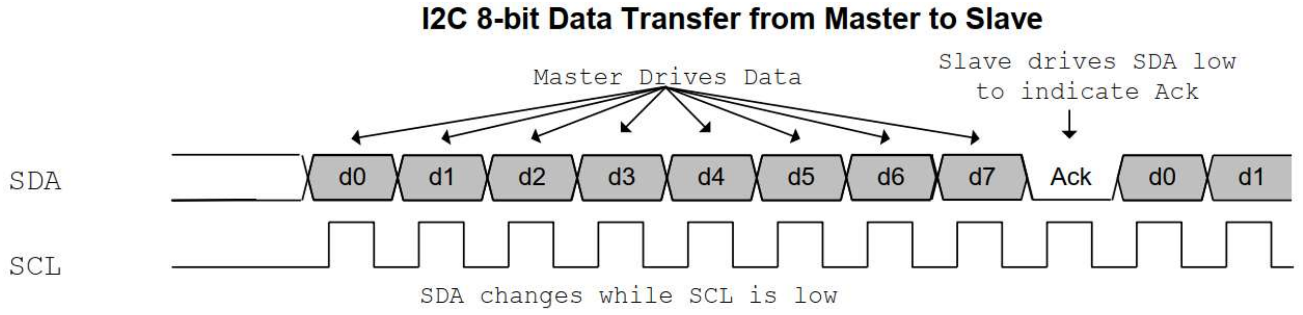
Other Error conditions, such as the Command FIFO Overflow and Read Data FIFO Underflow interrupt conditions are self-clearing. However, these bits will remain set in the Status register until the processor performs a FIFO Clear by writing 1'b1 to the FIFO Clearing bit in the Status register. The processor should wait until the FIFO Clearing bit is 1'b0 indicating recovery from this serious error condition is complete.

The loss of arbitration interrupt should be cleared by writing to the corresponding bit in the Interrupt Status register. This write will clear the interrupt. An additional write to the appropriate bit in the Status register is necessary to clear the detected condition in the I2C clock domain. It may also be necessary for the processor to perform a FIFO Clear if this condition is detected. There may be additional commands in the command FIFO that should be discarded before a new transaction is initiated.

If the Command FIFO Done condition is the source of the interrupt, the processor should make sure the bus is idle before placing new commands into the Command FIFO.

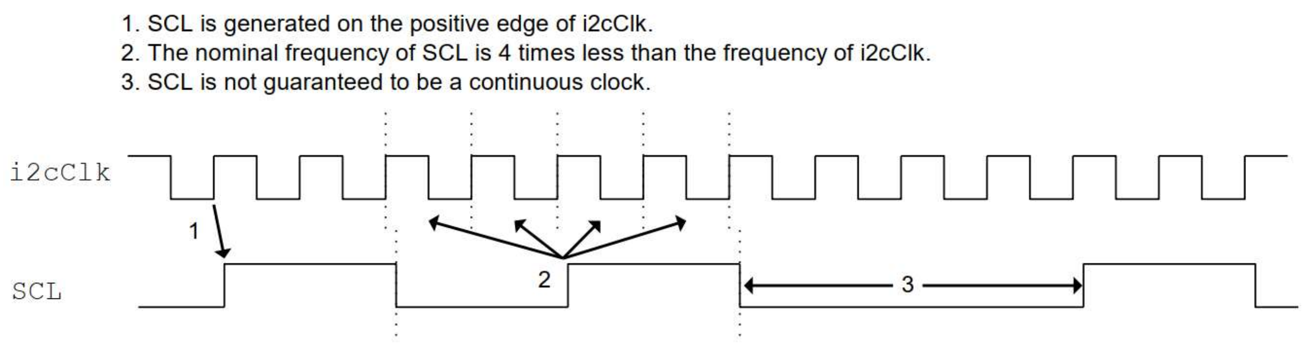
12.3.4 Example I2C Bus Events


I2C Start and Stop
Start
Stop
SDA
SCL
SDA high --> low while SCL is high
SDA low --> high while SCL is high


I2C 8-bit Data Transfer from Master to Slave
Master Drives Data
Slave drives SDA low to indicate Ack
SDA
SCL
d0
d1
d2
d3
d4
d5
d6
d7
Ack
d0
d1
SDA changes while SCL is low

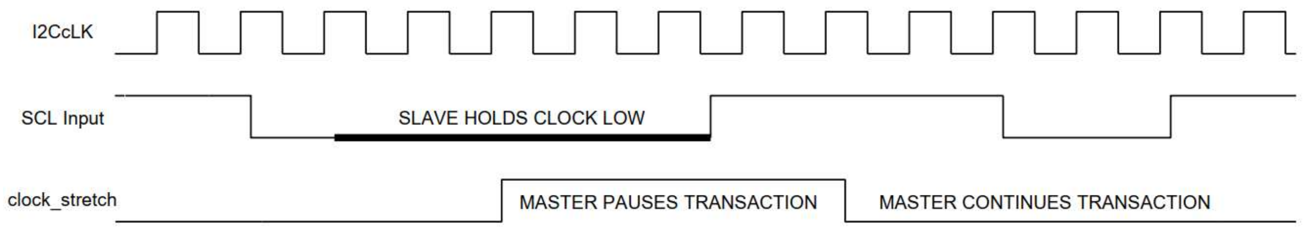
12.3.5 Example Clock Relationships

I2C Engine Clocks


1. SCL is generated on the positive edge of i2cClk.
2. The nominal frequency of SCL is 4 times less than the frequency of i2cClk.
3. SCL is not guaranteed to be a continuous clock.
i2cClk
SCL
1
2
3

12.3.6 Clock Stretching

A slave device may hold the clock low to let the master know that it requires time to complete the transaction (clock stretching). Thus the timing of the input *SCL* clock may controlled by the master or the slave. Normally a possibly asynchronous input would need to be synchronized before being used. The master needs to know when the clock is different from what it is outputting and also when the clock returns to being the same. The slave must hold the clock low within a half period of *SCL* when the master is still holding it low. Since there is no change in state at this time, it is safe for the master to sample the unsynchronized input. There would be no time for the synchronization and transmission would fail if the master did otherwise. However during clock stretching mode and looking for the input *SCL* to return high, the master can afford to wait and use the synchronized version of the input before continuing the transaction.



12.3.7 Arbitration

More than one master is allowed on the I2C bus. A simple arbitration detection mechanism is implemented to allow modest compatibility with other I2C masters. The master detects loss of arbitration when *SCL* is low, *SDAout* is high and *SDAin* is low. When loss of arbitration occurs, an interrupt can signal the processor. Until the processor acknowledges and clears this condition, the master tri-states the *SDA* & *SCL* outputs. It is the responsibility of the processor to attempt any retries or other recovery from loss of arbitration. It is recommended that the loss of arbitration event should be followed with a write to the Status register to initiate FIFO clear.

12.4 Register Descriptions

12.4.1 Summary

Register Name	Offset	Access	Description
Status	0x00	R/W	Operational Status of the I2C Module and its sub-modules
Read Data	0x04	RO	This register provides access to the Read Data FIFO
Command	0x08	WO	Programming interface to the I2C engine
Interrupt Enable	0x0C	R/W	This register enables (with 1's) or disables (with 0's) any of four individually configurable interrupt conditions. An interrupt to the processor can occur if any of these interrupt conditions are enabled AND if the I2C enable bit in the Control register is set. This register returns the state of various interrupt enable bits.
Interrupt Status (R)/ **Interrupt Clear (W)**	0x10	R/W	Reading this register (Interrupt Status) returns the current status of the active interrupts

			Writing 1’s to this register (Interrupt Clear) clears the Interrupt(s)
Control	0x14	R/W	Operational Control of the I2C Module including Enable, I2C Clock Select and I2C Clock Divider
Prescale	0x18	R/W	Extension of the I2C Clock Divider value

12.4.2 Status Register (offset: 0x00)

Signal	Bits	Default	R/W	Description
reserved	31:11			
CLR_FIFO	10	b0	R/W	Read = 1 indicates a FIFO Clear is in progress and is not yet complete, 0 indicates no FIFO Clear is in progress Write = 1 initiates clear both FIFOs (Command FIFO & Read Data FIFO). This self-clearing bit will be read as 0 when the FIFO clear is complete.
CMD_U_FLOW	9	b0	RO	= 1 indicates Error Condition: Command FIFO Underflow
RD_O_FLOW	8	b0	RO	= 1 indicates Error Condition: Read Data FIFO Overflow
ST_TRANS	7	b0	RO	= 1 indicates Transfer In Progress
CMD_FIFO_FU	6	b0	RO	= 1 indicates Command FIFO Full
CMD_O_FLOW	5	b0	RO	= 1 indicates Error Condition: Command FIFO Overflow
RD_U_FLOW	4	b0	RO	= 1 indicates Error Condition: Read Data FIFO Underflow
LOST_ARB	3	b0	R/W	Read (Status) = 1 indicates the Master lost arbitration Write (Clear) = 1 clears this condition on the I2C clock domain

ERR_NO_ACK	2	b0	R/W	Read (Status) = 1 indicates an Error Condition: I2C Protocol Error and no I2C “ACK” was detected from slave when expected Write (Clear) = 1 clears this condition on the I2C clock domain
RD_N_EMP	1	b0	RO	1 = Read Data FIFO Not Empty
CMD_EMP	0	b1	RO	1 = Command FIFO Empty

12.4.3 Read Data Register (offset: 0x04)

Signal	Bits	Default	R/W	Description
reserved	31:8			
READ_DATA	7:0	x0	RO	Data from I2C Slave to be read by processor

12.4.4 Command Register (offset: 0x08)

Signal	Bits	Default	R/W	Description
reserved	31:8			
CMD	7:0	x0	WO	Commands from processor to be implemented as I2C events. Writes to this register go directly into the Command FIFO. If a FIFO clear is in progress, writes to the Command FIFO are inhibited.

12.4.5 Interrupt Enable Register (offset: 0x0C)

Signal	Bits	Default	R/W	Description

reserved	31:5			
CMD_DONE_EN	4	b0	R/W	Enable Command FIFO Done Interrupt (1 = Enable, 0 =Disable)
LOST_ARB_EN	3	b0	R/W	Enable Loss of Arbitration Interrupt (1 = Enable, 0 = Disable)
ERR_EN	2	b0	R/W	Enable Error Interrupt (1 = Enable, 0 = Disable)
RD_N_EMP_EN	1	b0	R/W	Enable Read Data FIFO Not Empty Interrupt (1 = Enable, 0 = Disable)
CMD_EMP_EN	0	b0	R/W	Enable Command FIFO Empty Interrupt (1 = Enable, 0 = Disable)

12.4.6 Interrupt Status Register / Interrupt Clear Register (offset: 0x10)

Signal	Bits	Default	R/W	Description
reserved	31:5			
INT_CMD_DONE	4	b0	R/W	Read (Status) = 1 indicates Command FIFO Done Interrupt is active Write (Clear) = 1 clears the Command FIFO Done Interrupt
INT_LOST_ARB	3	b0	R/W	Read (Status) = 1 indicates Loss of Arbitration Interrupt is active Write (Clear) = 1 clears the Loss of Arbitration Interrupt
INT_ERR	2	b0	R/W	Read (Status) = 1 indicates Error Interrupt is active Write (Clear) = 1 clears the Error Interrupt

INT_RD_N_EMP	1	b0	R/W	Read (Status) = 1 indicates Read Data FIFO Not Empty Interrupt is active Write (Clear) = 1 clears the Read Data FIFO Not Empty Interrupt
INT_CMD_EMP	0	b0	R/W	Read (Status) = 1 indicates Command FIFO Empty Interrupt is active Write (Clear) = 1 clears the Command FIFO Empty Interrupt

12.4.7 Control Register (offset: 0x14)

Signal	Bits	Default	R/W	Description
reserved	31:8			
ENABLE	7	b0	R/W	I2C Enable (1 = Enable, 0 = Disable)
CLK_SOURCE	6	b0	R/W	I2C Clock Source (1 = PCLK, 0 = clkI2C)
CLK_DIV_LOW	5:0	x0	R/W	I2C Clock Divider (Lower 6 bits)

12.4.8 Prescale Register (offset: 0x18)

Signal	Bits	Default	R/W	Description
reserved	31:8			
CLK_DIV_HIGH	7:0	x0	R/W	I2C Clock Divider (Upper 8 bits) Note the entire clock divider is 14 bits

13 APB Timer

The APB Timer module is a 32-bit down counter with a selectable pre-scaler. Prescale values between 1 and 1024 can be selected. The pre-scaler extends the timer's range at the expense of precision.

The Timer provides two modes of operation to provide a free running value and periodic interrupts.

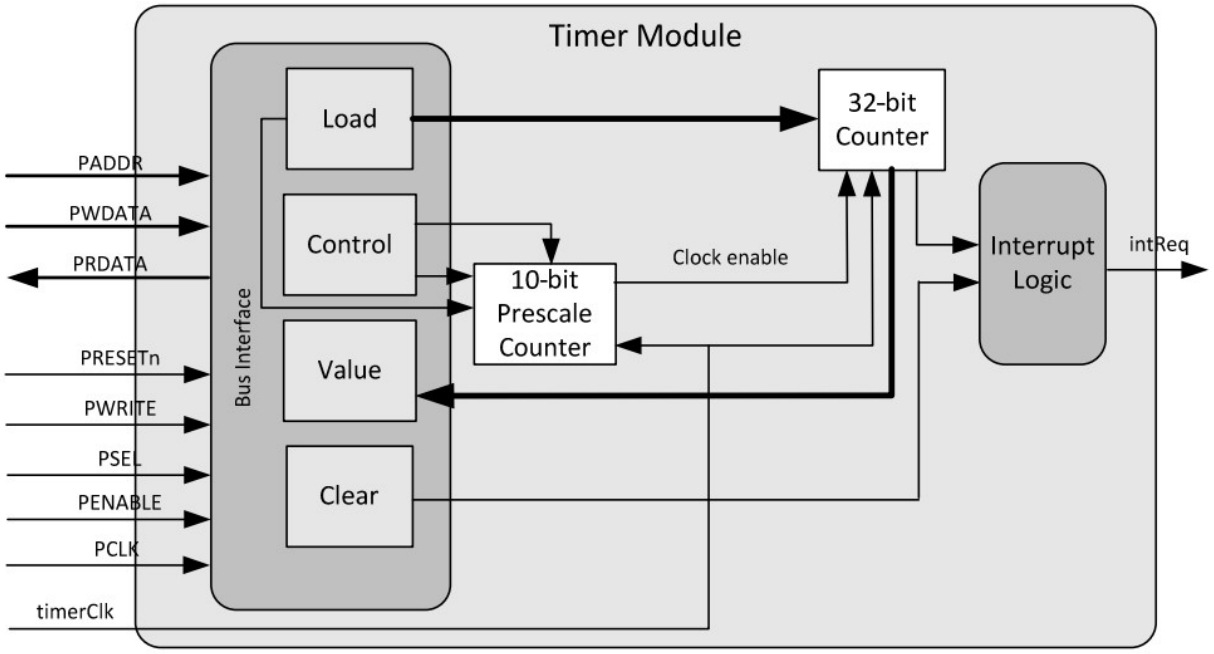
The Timer contains several configuration registers that can be written and read by the processor. A 10-bit pre-scaler precedes a 32-bit counter. The counter can be clocked at either the input clock rate or a choice of 7 pre-scaled rates. The counter can be loaded with a value from a preload register. The counter can optionally generate an interrupt.

The Timer module is a standard APB Slave peripheral. Timer registers are accessed through this interface.

13.1 Features

- 32-bit counter/timer
- 10-bit selectable pre-scale
- Periodic and free-running event timer modes

13.2 Block Diagram



13.3 Functional Description

The timer is a basic 32-bit down counter. When the timer is enabled by setting bit 7 in the Control Register or a new value is set in the Load Register, the Load Register is loaded into the Value Register. Bits 2 ~ 4 of the Control Register set the scaling of *timerClk* creating a timer clock. The Value Register is decremented on the leading edge of each timer clock until it reaches zero. The *intReq* output is then activated until it is cleared by writing to the Clear Register.

The timer has two modes of operation determined by bit 6 in the Control Register: free-running and periodic. When the Value Register reaches zero and the free-running mode is selected, Value is decremented to 0xFFFFFFFF on the next timer clock edge. When the Value Register reaches zero and periodic running mode is selected, the Load Register is reloaded into the Value Register on the next timer clock edge. In either case, the Value Register will continue to be decremented on each timer clock until the Value register reaches zero and the whole process starts over again.

Because zero is a separate count, periodic counts are actually one count longer than the contents of the Load Register. For example, if a value of 99 is written to the Load Register, no pre-scale is selected and the periodic mode is selected so an interrupt will occur every 100 *timerClk* cycles.

The pre-scaler is a free-running counter. It is held in the clear state when the Timer is disabled (bit 7 of the Control Register is zero). It is also cleared when the Load Register is loaded. Bits 2 through 4 of the Control Register determine which pre-scale output is selected. This produces a clock enable for the timer counter register. A bitwise OR of the pre-scale counter and a Mask value (based on three bits in the Control Register) is performed; if all bits of the OR output are high and the timer is enabled, then the timer counter is allowed to decrement.

So in periodic mode the number of clocks between interrupts will be:

$$\# \text{clocks} = (val + 1) * pDiv,$$

where *val* is the contents of the Load register and *pDiv* is the pre-scale divisor value corresponding to the value programmed into the Control register.

Note during periodic interrupt mode with small Load and prescale values, interrupts can be triggered faster than the software can service them. This should be avoided.

The clock for the timer logic is separate from the PCLK used for the APB interface. This allows PCLK to be varied as needed but the timing (and software) will not change. There are a few signals that are synchronized between these clock domains.

13.4 Register Descriptions

13.4.1 Load (offset: 0x00)

Signal	Bits	Default	R/W	Description
LOAD	31:0	x0	R/W	The initial value of the timer, also the reload value when the timer is in periodic mode

13.4.2 Control (offset: 0x08)

Signal	Bits	Default	R/W	Description

reserved	31:9			
INT_STATUS	8	b0	RO	Interrupt Status 0 = Timer interrupt is inactive 1 = Timer interrupt active
ENABLE	7	b0	R/W	0 = Timer Disabled, 1 = Timer Enabled
MODE	6	b0	R/W	0 = Free running mode, 1 = Periodic mode
INT_ENABLE	5	b0	R/W	Interrupt Enable 0=Disable Timer interrupt 1=Enable Timer interrupt
PRESCALE	4:2	b0	R/W	0=clock is divided by one 1=clock is divided by 16 2=clock is divided by 256 3=clock is divided by 2 4=clock is divided by 8 5=clock is divided by 32 6=clock is divided by 128 7=clock is divided by 1024
reserved	1:0			

13.4.3 Clear (offset: 0x0C)

Signal	Bits	Default	R/W	Description
CLEAR	31:0		WO	Clear Interrupt – write any value to clear the timer interrupt

13.4.4 Repeat_Delay (offset: 0x10)

Signal	Bits	Default	R/W	Description
reserved	31:16			
INT_REPEAT_ DELAY	15:0	b0	R/W	Interrupt is active after multiple times. It is only valid for Timer3 and Timer4 when sleep mode is enabled. 16'd0 = interrupt is active when the 1st event occurs 16'd1 = interrupt is active when the 2nd event occurs etc…

14 Watchdog Timer

The Watchdog Timer is useful in detecting the condition where the CPU and software get into an unrecoverable unknown or “stuck” state. Software periodically “kicks” the watchdog by writing to it for resetting its timeout counter.

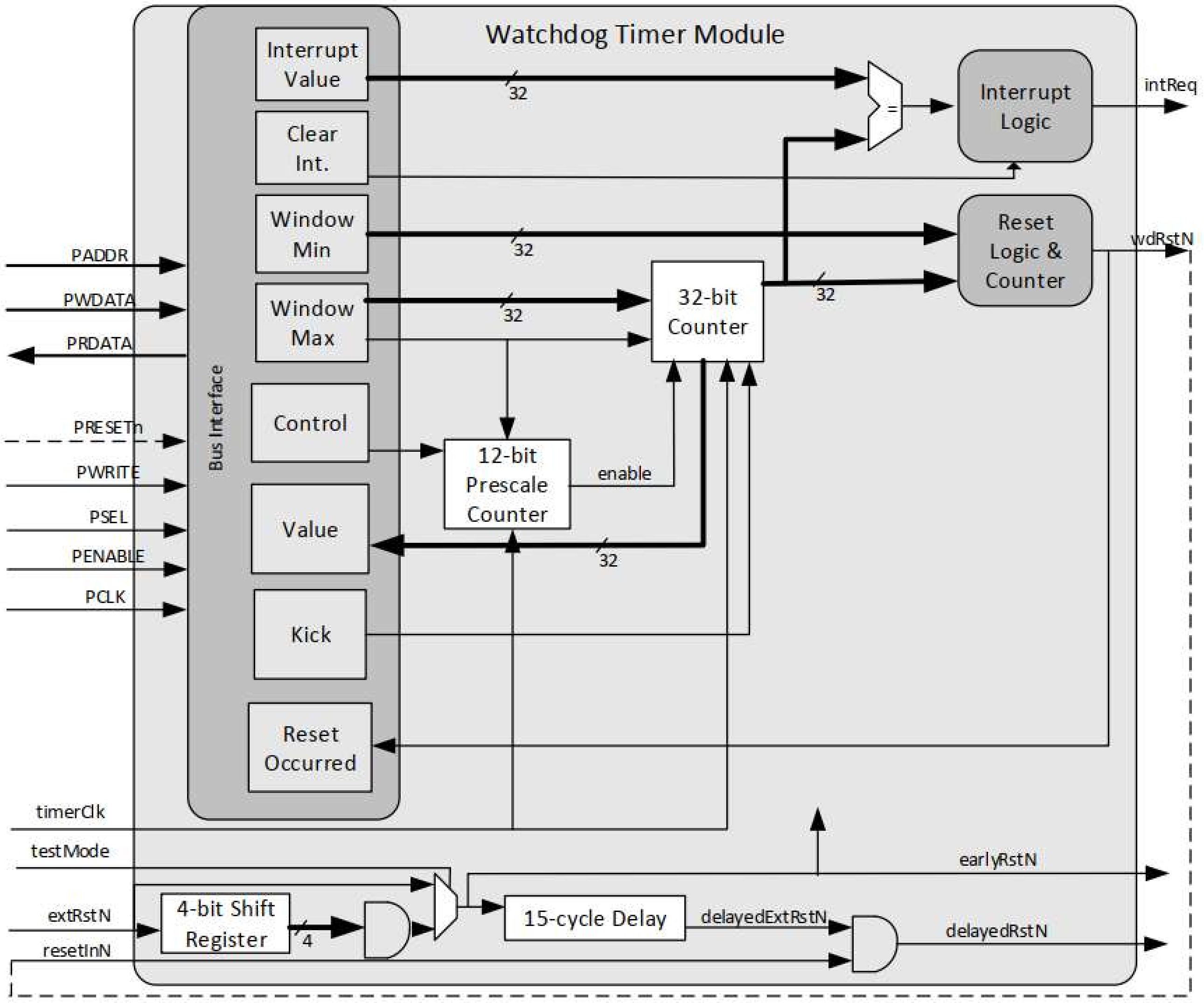
The Watchdog Timer module is a programmable reset controller module. The module is a standard APB Slave peripheral; the Watchdog Timer registers are accessed through this interface. The module implements a 32-bit down counter with a selectable pre-scaler in order to produce a delayed reset signal and a watchdog reset signal, as well as a warning interrupt. The reset and interrupt may be separately enabled. The interrupt could be used, for instance, to save some setup information to non-volatile memory.

The Watchdog Timer has a Windowing feature. This insures that not only does the watchdog need to be kicked at a certain minimum rate, but also that it is not kicked too frequently. This feature protects against instances such as the code being stuck in a loop which includes a timer kick.

14.1 Features

- 32-bit loadable down counter
- 12-bit pre-scaler
- Internal and external reset inputs
- Windowing feature for timing min. and max. “kick” times
- Warning interrupt

14.2 Block Diagram



14.3 Functional Description

The watchdog timer is a basic 32-bit down counter which generates a reset when it decrements to zero. The watchdog timer is only active when enabled and can be enabled by setting bit 7 in the Control Register. Also, the reset output and interrupt output features may be separately enabled or disabled.

Bits 2 thru 4 of the Control Register control prescaling of timerClk creating a timer clock. For example, when all 3 bits are zero, timerClk directly generates the timing, or when bit 2 is asserted and bit 3 is unasserted, timerClk is divided by 16 to generate the timing.

The prescaler is a free running counter. Its output is a 1-clock-wide pulse that feeds the watchdog timer's main 32-bit counter as a clock enable. The prescaler is held in the clear state when the Timer is disabled (bit 6 of the Control Register is zero). It is also cleared when the *WindowMax* Register is loaded. Bits 2 thru 4 of the Control Register determine which prescale divide ratio is selected. A bitwise OR of the prescale counter and a Mask value (based on three bits in the Control Register) is performed; if all bits of the OR output are high and the timer is enabled, then the 32-bit watchdog timer counter is allowed to decrement.

When the watchdog timer is active, the internal timerValue Register counts down from the value of the *WindowMax* Register. The *timerValue* Register is decremented on each enabled clock edge until it reaches zero. The module then asserts *wdRstN* which can be used to restart a system that appears to have timed out. The system should regularly "kick" the Watchdog to prevent this system reset. When kicked, the watchdog reloads the timerValue register with the contents of the *WindowMax* Register and resumes decrementing.

The number of clocks until reset is given by the following formula:

$$resetClks = (WindowMax + 1) \times prescaleDiv$$

where *WindowMax* is the contents of the *WindowMax* register and *prescaleDiv* is the prescale divisor value corresponding to the prescale code programmed into the Control register.

The watchdog can be kicked by a single write to the Kick Register. A timer kick, timer enable, or a write to the *WindowMax* Register all cause the internal timerValue register to be loaded with the contents of the *WindowMax* Register.

If windowing is desired, the *WindowMin* value can be set non-zero, but less than *WindowMax*. When these conditions are met, the timer being kicked between the maximum and minimum levels will also trigger a *wdRstN* output.

The watchdog timer will issue a warning interrupt (assert *intReq* = 1’b1) intValue number of timer counts before it asserts *wdRstN*. The register intValue can be programmed with a 32-bit value. The purpose of the interrupt is to give an early warning the system that a reset will soon occur. *intReq* will remain active until the watchdog has been kicked or until the Clear Register has been written to. This feature also allows the watchdog timer to be used as an additional timer to generate and interrupt if the watchdog reset is not needed.

$$intClks = (WindowMax - intValue + 1) \times prescaleDiv$$

The clock for the timer logic is separate from the PCLK used for the APB interface. This is so that PCLK can be varied as needed but the timing (and software) will not change. There are a few signals that are synchronized back and forth between these clock domains.

14.4 Register Descriptions

14.4.1 WindowMax (offset: 0x00)

The Load Register is R/W unless the lockOut bit in the Control Register has been set to 1’b1. If lockOut = 1’b1, then the register is read-only.

Signal	Bits	Default	R/W	Description
WindowMax	31:0	x0	R/W	This 32-bit value gets loaded into the timer’s counter register and reloaded upon a specific write to the watchdog Kick Register.

14.4.2 TimerVal (offset: 0x04)

Signal	Bits	Default	R/W	Description

TimerVal	31:0	xFFFF	RO	The current value of the watchdog timer synchronized to the PCLK domain. Note that in cases where PCLK is slow, this value may significantly differ from the actual internal count.

14.4.3 Control (offset: 0x08)

The Control Register is a R/W register unless the lockOut bit has been set to 1’b1 via a previous write to the Control Register. If lockOut = 1’b1, then the Control Register is read-only.

Signal	Bits	Default	R/W	Description
reserved	31:8			
ENABLE	7	b0	R/W	0=Timer Disabled, 1=Timer Enabled
INT_EN	6	b0	R/W	0=Interrupt Disabled, 1=Interrupt Enabled.
RESET_EN	5	b0	R/W	0=Reset Disabled, 1=Reset Enabled.
PRESCALE	4:2	b0	R/W	0=clock is divided by one 1=clock is divided by 16 2=clock is divided by 256 3=clock is divided by 32 4=clock is divided by 128

				5=clock is divided by 1024 6=clock is divided by 4096 7=clock is divided by 4096
reserved	1			
LOCKOUT	0	b0	R/W	This bit is the lockOut bit. Write 1'b1 to prevent further configuration changes to the WindowMax, WindowMin, Control, or Reset Occurred Registers

14.4.4 Kick (offset: 0x0C)

The Kick Register is a write-only register. A specific value must be written to the Kick Register in order to reload the watchdog timer with the WindowMax time.

Signal	Bits	Default	R/W	Description
KICK	31:0		WO	Kick the watchdog: reset the internal timer value to WindowMax and clear the interrupt. Write 0xA5A5 to reload the watchdog timer.

14.4.5 Reset Occurred (offset: 0x10)

The Reset Occurred Register is a R/W register unless the lockOut bit in the Control Register has been set to 1'b1. If lockOut = 1'b1, then the Reset Occurred Register is read-only and writes have no effect.

Signal	Bits	Default	R/W	Description
reserved	31:8			

RESET_OCCUR	7:0	b0	R/W	resetOccurred counter Writes have no effect if the lockOut bit in the Control Register is 1’b1. If the lockOut bit is 1’b0, then write any value to the Reset Occurred register to clear the resetOccurred counter. Read the resetOccurred counter to see how many watchdog reset events have occurred. Note: Counter stops incrementing at 255. Write any value to clear.

14.4.6 Clear (offset: 0x14)

The Clear Register is a write-only register that clears the active interrupt signal (*intReq*). This register clears the interrupt regardless of the state of the lockOut bit.

Signal	Bits	Default	R/W	Description
CLEAR	31:0		WO	Write any value to clear intReq interrupt.

14.4.7 Interrupt Value (offset: 0x18)

The Interrupt Value Register is a R/W register unless the lockOut bit in the Control Register has been set to 1’b1. If lockOut = 1’b1, then the register is read-only.

Signal	Bits	Default	R/W	Description
intValue	31:0		R/W	This 32-bit value is compared with the timer’s counter register. If they are equal an interrupt is generated.

14.4.8 WindowMin (offset: 0x1C)

The WindowMin Register is R/W unless the lockOut bit in the Control Register has been set to 1'b1. If lockOut = 1'b1, then the register is read-only.

Signal	Bits	Default	R/W	Description
WindowMin	31:0		R/W	This 32-bit value feeds a comparator for determining if a write to the watchdog Kick Register has occurred too quickly. 0 means this feature is unused. The feature will also be unused if a value higher than WindowMax is programmed.

15 xDMA

The xDMA is a simple data memory access module that embedded into various peripheral. The xDMA will start to access the memory data from the start address pointer whenever a peripheral needs to read or write memory.

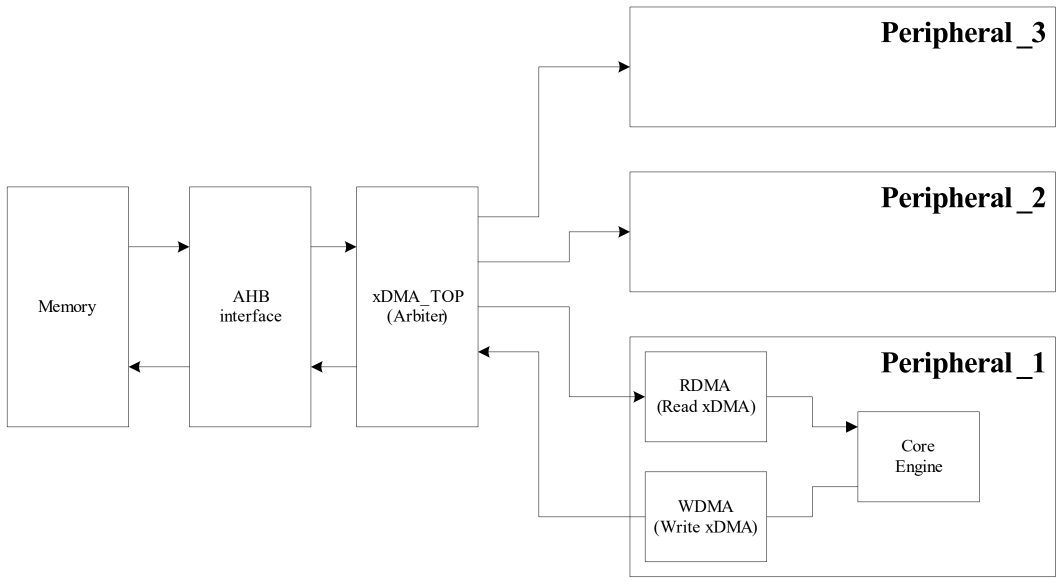
The xDMA has two modes, one is single-shot mode that access memory a predefined number of times. Another mode is continuous mode that access memory infinitely. Each xDMA of peripheral will connect to xDMA_TOP that is a simple round-robin arbiter deciding which request from each peripheral is served.

15.1 Features

- User-defined starting point address
- Read-type xDMA
- Write-type xDMA
- Issue interrupt

- Single-shot mode and continuous mode

15.2 Block diagram



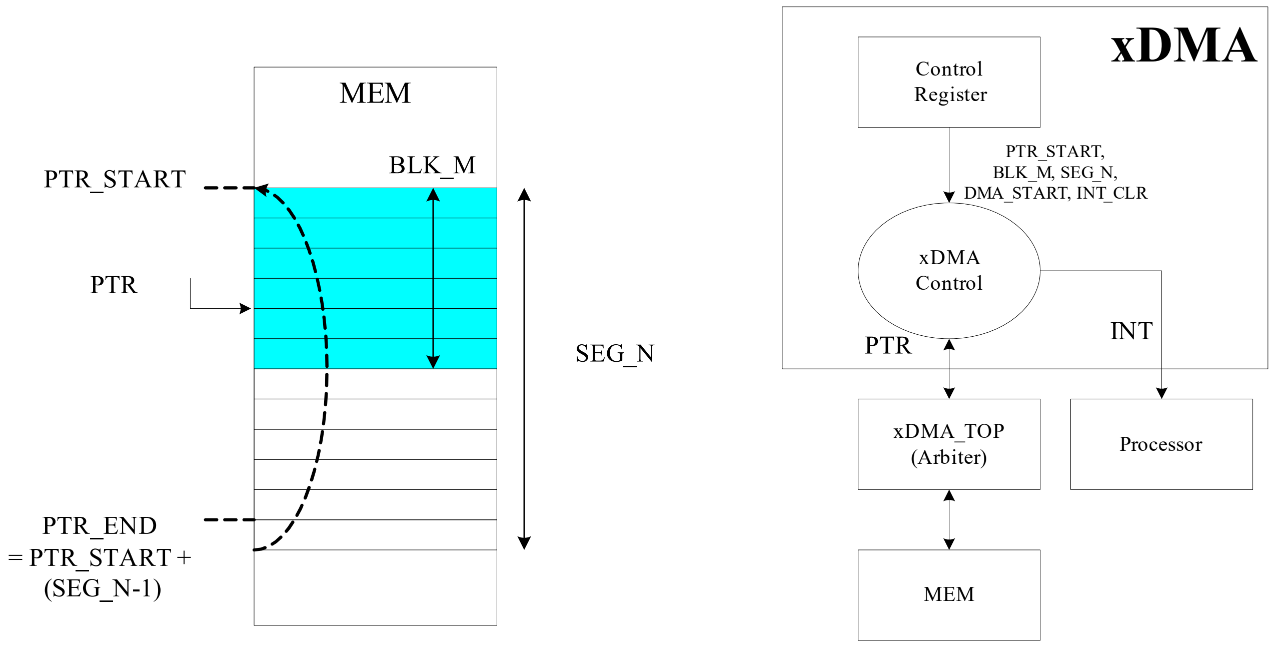
15.3 Function description

xDMA main parameters:

- PTR_START is start address pointer
- SEG_N is segment size
- BLK_M is block size
- INT is interrupt of xDMA
- INT_CLR is interrupt clear
- DMA_START is start trigger to xDMA

The start address pointer is a double buffer design and it is latched into xDMA control whenever DMA_START triggers xDMA to start. The xDMA control will increase the pointer from PTR_START to

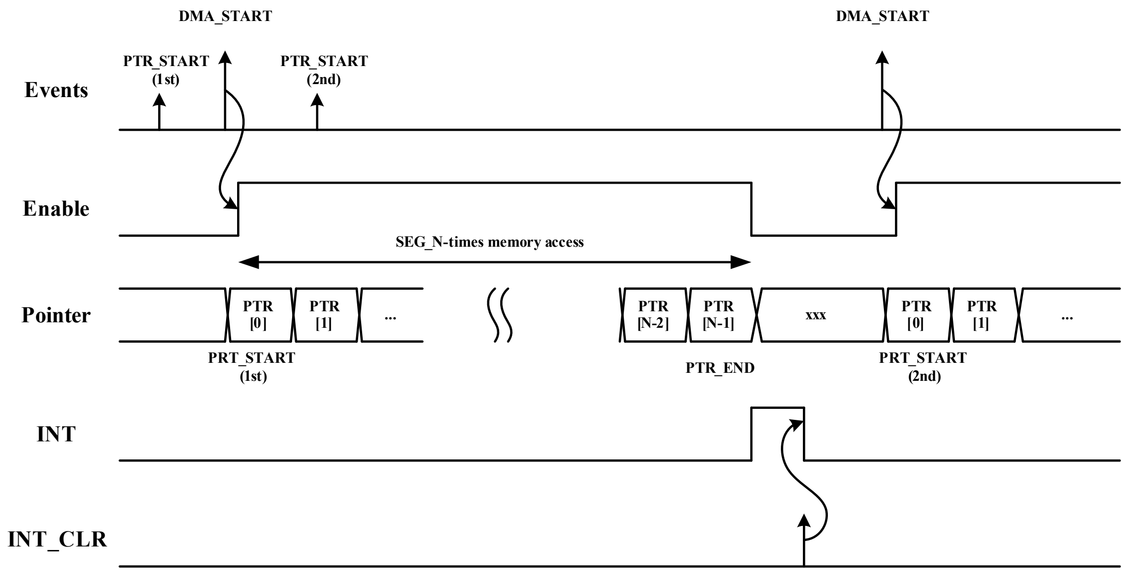
PRT_END. The addressing mode for increasing pointer steps should be word (4-bytes) or half-word (2-bytes) alignment depending on the memory access type of each peripheral.



15.3.1 Single-shot mode

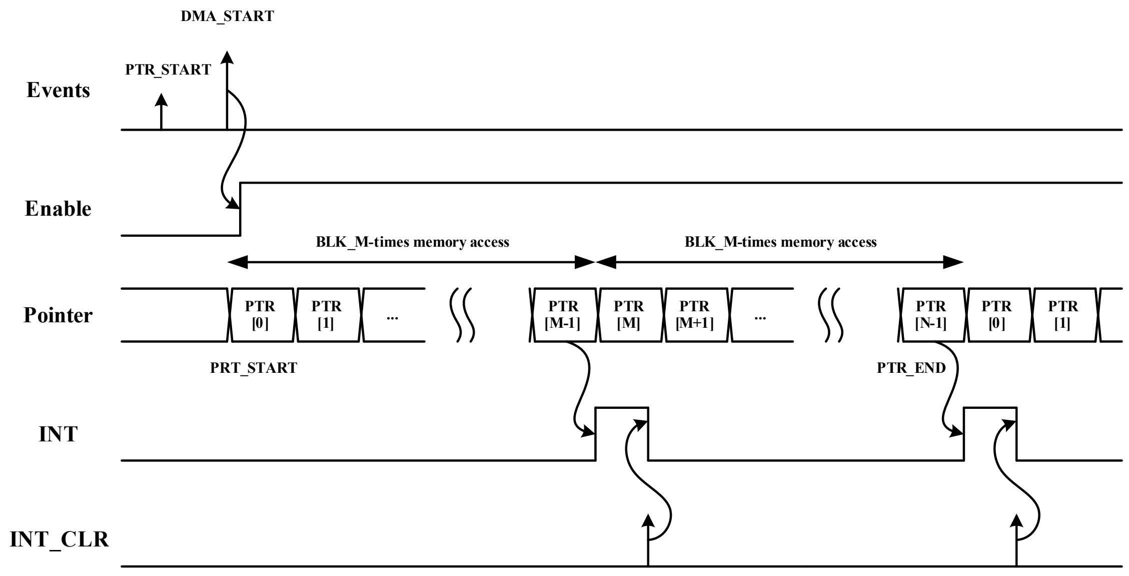
Single-shot mode configures parameters PRT_START and SEG_N as desired values and sets BLK_M to zero to enter single-shot mode. It sets the control register to start xDMA accessing memory from address PTR_START to PTR_END. Once the memory access SEG_N-times is done, the xDMA will issue an interrupt and stop xDMA to access memory. When the processor receives an interrupt it will prepare a next single-shot access (or not) and clear teh interrupt before exiting the interrupt service routine.

The below timing diagram is a simple behavior of single-shot xDMA with BLK_M equal to zero.



15.3.2 Continuous mode

Continuous mode configures the parameters PRT_START and SEG_N as their desired values and sets BLK_M to non-zero and smaller than SEG_N to enter continuous mode. Sets control register to start xDMA accessing memory from address PTR_START to the PTR_END. Once the memory access BLK_M-times is done, the xDMA will issue an interrupt and continuously access memory. If the pointer reaches PRT_END, it will wrap around to the start address PTR_START for the next memory access (ring buffer function). When the processor receives an interrupt, it can write/read the memory content defined by xDMA for further processing and clear the interrupt before the exit interrupt service routine. The below timing diagram is a simple behavior of continuous xDMA with SEG_N = 2*BLK_M as a ping-pong buffer configuration.



15.4 Register Descriptions

The xDMA register is embedded into various peripherals and the usage is all the same. For simplicity, the table uses XXX to represent different peripherals and lists a general form to the xDMA register.

- WDMA represents write type xDMA
- RDMA represents read type xDMA

Register Name	Offset	Access	Description
xxx_RDMA_CTL0	0xX0	W1C	Start trigger xDMA
xxx_RDMA_CTL1	0xX4	W1C	Reset xDMA
xxx_RDMA_SET0	0xX8	R/W	Segment size & Block size of xDMA
xxx_RDMA_SET1	0xXC	R/W	Start address of xDMA
xxx_RDMA_R0	0xX8	R	Next pointer address of xDMA
xxx_RDMA_R1	0xXC	R	Debug

xxx_INT_CLEAR	0xXX	W1C	Clear XXX_RDMA interrupt
xxx_INT_MASK	0xXX	R/W	Mask XXX_RDMA interrupt
xxx_INT_STATUS	0xXX	R	Status of XXX_RDMA interrupt
xxx_WDMA_CTL0	0xX0	W1C	Start trigger xDMA
xxx_WDMA_CTL1	0xX4	W1C	Reset xDMA
xxx_WDMA_SET0	0xX8	R/W	Segment size & Block size of xDMA
xxx_WDMA_SET1	0xXC	R/W	Start address of xDMA
xxx_WDMA_R0	0xX8	R	Next pointer address of xDMA
xxx_WDMA_R1	0xXC	R	Debug
xxx_INT_CLEAR	0xXX	W1C	Clear XXX_WDMA interrupt
xxx_INT_MASK	0xXX	R/W	Mask XXX_WDMA interrupt
xxx_INT_STATUS	0xXX	R	Status of XXX_WDMA interrupt

16 I2S

The I2S module transmits or receives audio samples with predefined protocol through GPIO. It supports two channels with I2S, LJ (Left Justified) and RJ (Right Justified) format. Two xDMA are also included in the I2S module to transfer audio samples between memory and the I2S module without CPU intervention.

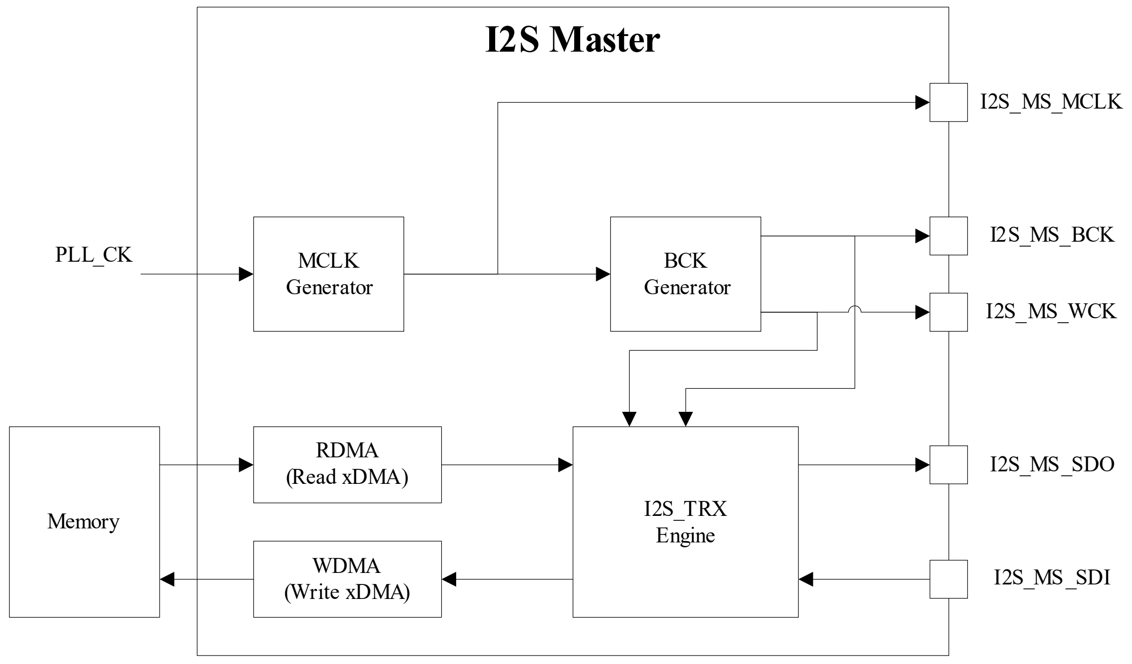
16.1 Features

The I2S features:

- Master mode

- Master clock generator for MCLK
- Tx, Rx and Tx+Rx
- I2S, LJ and RJ format with 16/32-BCK
- 16/24/32 sample width
- Audio sample rates: 8K, 16K, 32K and 48K

16.2 Block Diagram



16.3 Function description

16.3.1 MCLK and BCK configurations

The main parameters of I2S rate configurations are:

- I2S_BCK = 2 * I2S_WCK * (Bit-Length of I2S_BCK per-channel)

- I2S_MCLK = I2S_BCK * Ratio = iMCLK / iMCLK_Divisor
  - iMCLK is the internal MCLK clock rate in the MCLK generator
  - I2S_MCLK is generated from iMCLK divided by iMCLK_Divisor
  - Ratio is selected from 2, 4 and 8
- I2S_MS_SET0 and I2S_MCLK_SET0 are configuration registers

I2S_MCLK_SET0. cfg_mck_isel	iMCLK	Comment
0	12.288M	PLL_CK = 32M, I2S_WCK = 48K
1	8.192M	PLL_CK = 32M, I2S_WCK = 8/16/32K
2	12.288M	PLL_CK = 48M, I2S_WCK = 48K
3	8.192M	PLL_CK = 48M, I2S_WCK = 8/16/32K
4	24.576M	PLL_CK = 64M, I2S_WCK = 48K
5	16.384M	PLL_CK = 64M, I2S_WCK = 8/16/32K

I2S_MCLK_SET1. cfg_mck_div	iMCLK_Divisor	Comment
0	1	I2S_MCLK = iMCLK / 1
1	2	I2S_MCLK = iMCLK / 2
2	4	I2S_MCLK = iMCLK / 4
3	8	I2S_MCLK = iMCLK / 8
4	16	I2S_MCLK = iMCLK / 16
5	32	I2S_MCLK = iMCLK / 32

I2S_MS_SET0. cfg_bck_osr	Ratio	Comment
0	2	I2S_MCLK = I2S_BCK * Ratio
1	4	
2	8	

I2S_MS_SET0. cfg_bck_len	Bit-Length of BCK per channel	Comment
0	16	L/R with 16-BCLK
2	32	L/R with 32-BCLK

For PLL = 32M, the I2S rate configuration is listed in the table below.

I2S_WCK (KHz)	Bit Length of BCK	I2S_BCK (KHz)	Ratio (MCLK/BCK)	I2S_MCLK (KHz)	iMCLK (KHz)	iMCLK_Divisor	I2S_MS_SET0. cfg_bck_len	I2S_MS_SET0. cfg_bck_osr	I2S_MCLK_SET0. cfg_mck_isel	I2S_MCLK_SET1. cfg_mck_div
8	16	256	2	512	8192	16	0	0	1	4
8	16	256	4	1024	8192	8	0	1	1	3
8	16	256	8	2048	8192	4	0	2	1	2
8	32	512	2	1024	8192	8	2	0	1	3
8	32	512	4	2048	8192	4	2	1	1	2
8	32	512	8	4096	8192	2	2	2	1	1
16	16	512	2	1024	8192	8	0	0	1	3
16	16	512	4	2048	8192	4	0	1	1	2
16	16	512	8	4096	8192	2	0	2	1	1
16	32	1024	2	2048	8192	4	2	0	1	2
16	32	1024	4	4096	8192	2	2	1	1	1
16	32	1024	8	8192	8192	1	2	2	1	0
32	16	1024	2	2048	8192	4	0	0	1	2
32	16	1024	4	4096	8192	2	0	1	1	1
32	16	1024	8	8192	8192	1	0	2	1	0
32	32	2048	2	4096	8192	2	2	0	1	1
32	32	2048	4	8192	8192	1	2	1	1	0
32	32	N/A	8	N/A	N/A	N/A	N/A	N/A	N/A	N/A
48	16	1536	2	3072	12288	4	0	0	0	2
48	16	1536	4	6144	12288	2	0	1	0	1
48	16	N/A	8	N/A	N/A	N/A	N/A	N/A	N/A	N/A
48	32	3072	2	6144	12288	2	2	0	0	1
48	32	N/A	4	N/A	N/A	N/A	N/A	N/A	N/A	N/A
48	32	N/A	8	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Per channel								PLL = 32M	

For PLL = 48M, the I2S rate configuration is list in the table below.

I2S_WCK (KHz)	Bit Length of BCK	I2S_BCK (KHz)	Ratio (MCLK/BCK)	I2S_MCLK (KHz)	iMCLK (KHz)	iMCLK_Divisor	I2S_MS_SET0. cfg_bck_len	I2S_MS_SET0. cfg_bck_osr	I2S_MCLK_SET0. cfg_mck_isel	I2S_MCLK_SET1. cfg_mck_div
8	16	256	2	512	8192	16	0	0	3	4
8	16	256	4	1024	8192	8	0	1	3	3
8	16	256	8	2048	8192	4	0	2	3	2
8	32	512	2	1024	8192	8	2	0	3	3
8	32	512	4	2048	8192	4	2	1	3	2
8	32	512	8	4096	8192	2	2	2	3	1
16	16	512	2	1024	8192	8	0	0	3	3
16	16	512	4	2048	8192	4	0	1	3	2
16	16	512	8	4096	8192	2	0	2	3	1
16	32	1024	2	2048	8192	4	2	0	3	2
16	32	1024	4	4096	8192	2	2	1	3	1
16	32	1024	8	8192	8192	1	2	2	3	0
32	16	1024	2	2048	8192	4	0	0	3	2
32	16	1024	4	4096	8192	2	0	1	3	1
32	16	1024	8	8192	8192	1	0	2	3	0
32	32	2048	2	4096	8192	2	2	0	3	1
32	32	2048	4	8192	8192	1	2	1	3	0
32	32	N/A	8	N/A	N/A	N/A	N/A	N/A	N/A	N/A
48	16	1536	2	3072	12288	4	0	0	2	2
48	16	1536	4	6144	12288	2	0	1	2	1
48	16	1536	8	12288	12288	1	N/A	N/A	N/A	N/A
48	32	3072	2	6144	12288	2	2	0	2	1
48	32	3072	4	12288	12288	1	N/A	N/A	N/A	N/A
48	32	N/A	8	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Per channel								PLL = 48M	

The table of various I2S rate configurations is based on PLL = 64M.

I2S_WCK (KHz)	Bit Length of BCK	I2S_BCK (KHz)	Ratio (MCLK/BCK)	I2S_MCLK (KHz)	iMCLK (KHz)	iMCLK_Divisor	I2S_MS_SET0. cfg_bck_len	I2S_MS_SET0. cfg_bck_osr	I2S_MCLK_SET0. cfg_mck_isel	I2S_MCLK_SET1. cfg_mck_div
8	16	256	2	512	16384	32	0	0	5	4
8	16	256	4	1024	16384	16	0	1	5	3
8	16	256	8	2048	16384	8	0	2	5	2
8	32	512	2	1024	16384	16	2	0	5	3
8	32	512	4	2048	16384	8	2	1	5	2
8	32	512	8	4096	16384	4	2	2	5	1
16	16	512	2	1024	16384	16	0	0	5	3
16	16	512	4	2048	16384	8	0	1	5	2
16	16	512	8	4096	16384	4	0	2	5	1
16	32	1024	2	2048	16384	8	2	0	5	2
16	32	1024	4	4096	16384	4	2	1	5	1
16	32	1024	8	8192	16384	2	2	2	5	0
32	16	1024	2	2048	16384	8	0	0	5	2
32	16	1024	4	4096	16384	4	0	1	5	1
32	16	1024	8	8192	16384	2	0	2	5	0
32	32	2048	2	4096	16384	4	2	0	5	1
32	32	2048	4	8192	16384	2	2	1	5	0
32	32	2048	8	16384	16384	1	N/A	N/A	N/A	N/A
48	16	1536	2	3072	24576	8	0	0	4	2
48	16	1536	4	6144	24576	4	0	1	4	1
48	16	1536	8	12288	24576	2	N/A	N/A	N/A	N/A
48	32	3072	2	6144	24576	4	2	0	4	1
48	32	3072	4	12288	24576	2	N/A	N/A	N/A	N/A
48	32	N/A	8	N/A	N/A	N/A	N/A	N/A	N/A	N/A
0	Per channel								PLL = 64M	

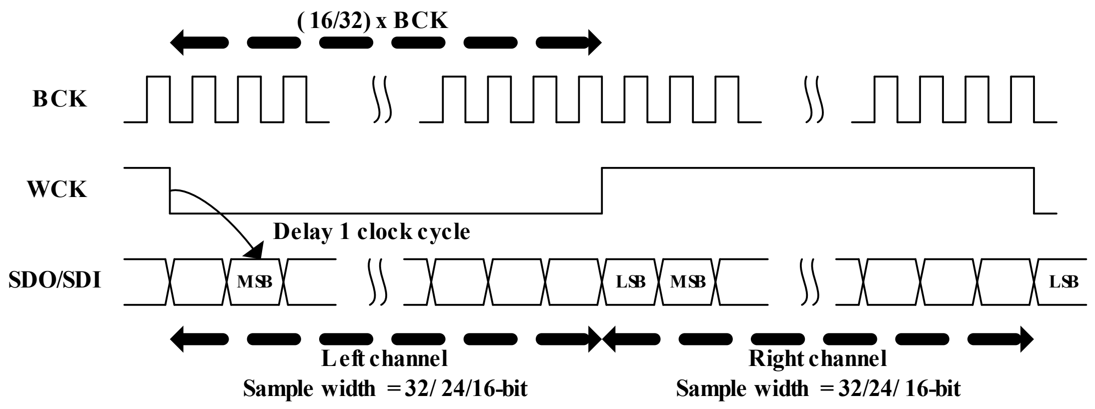
16.3.2 I2S, LJ and RJ format

The I2S supports three formats: I2S, LJ (Left-Justified) and RJ (Right-Justified). Each I2S format can be configured to 16 or 32 BCK cycles per channel. The audio sample width of 16, 24, or 32 bits can be transmitted or received via I2S protocol.

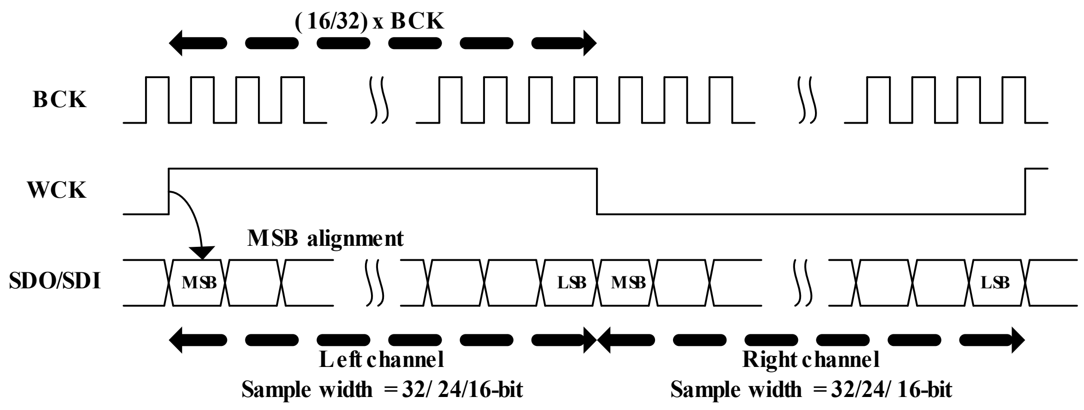
If the audio sample width is less than the number of BCK cycles, the audio sample will insert zeros to adapt the number of BCK cycles.

If the audio sample width is larger than the number of BCK cycles, the audio sample width will truncate the LSB to adapt the number of BCK cycles

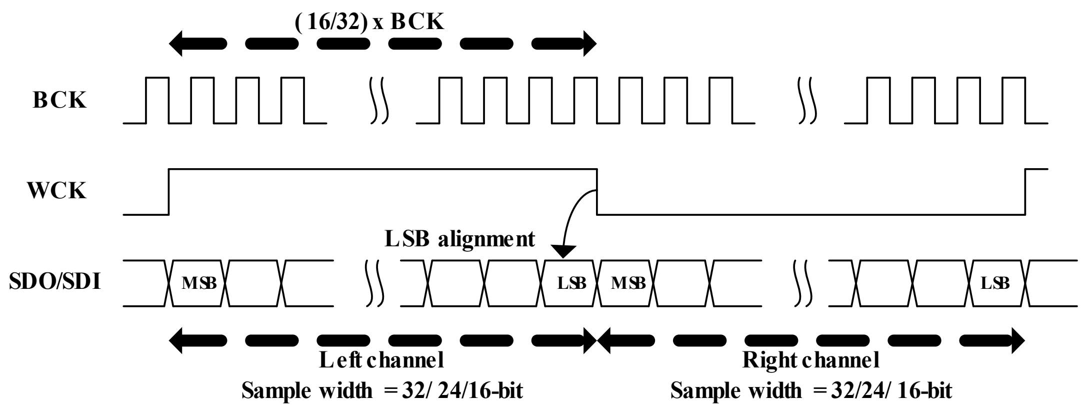
- I2S mode



- LJ format



- RJ format



16.3.3 xDMA and sample format

The I2S module has two xDMA with read type RDMA and two with write type WDMA. Each xDMA addressing mode has word (4-bytes) alignment.

- I2S_RDMA addressing mode with word alignment

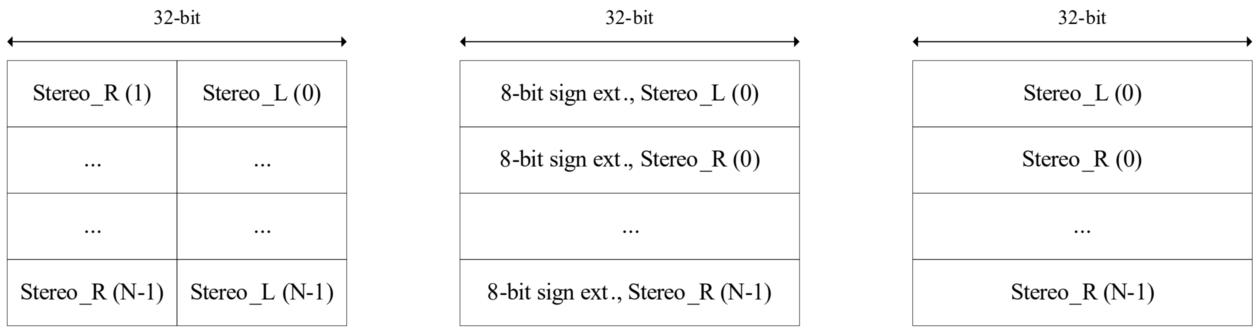
For the RDMA, the segment size and block size are configured from I2S_RDMA_SET0. The start address pointer for memory reading is configured from I2S_RDMA_SET1. The I2S_RDMA_CTL0 is used to trigger the start of RDMA for memory reading. The audio sample width and the channel definition stored in memory are configured from I2S_MS_SET0.

- I2S_WDMA addressing mode with word alignment

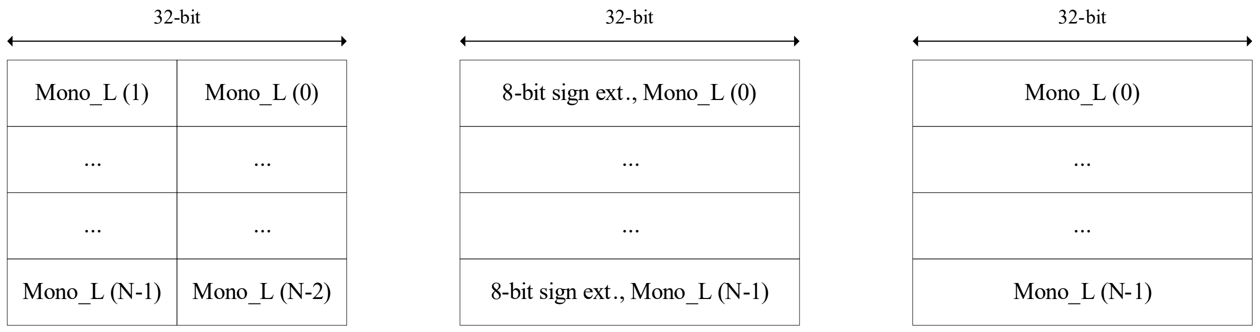
For the WDMA, the segment size and block size are configured from I2S_WDMA_SET0. The start address pointer for memory reading is configured from I2S_WDMA_SET1. The I2S_WDMA_CTL0 is used to trigger WDMA to start memory writing. The audio sample width and the channel definition stored in memory are configured from I2S_MS_SET0.

The audio sample format can have 16, 24 or 32-bits in the memory content. Each audio sample can also be defined as Stereo, Mono-L (Left) and Mono-R (Right) for I2S Left and Right channel respectively.

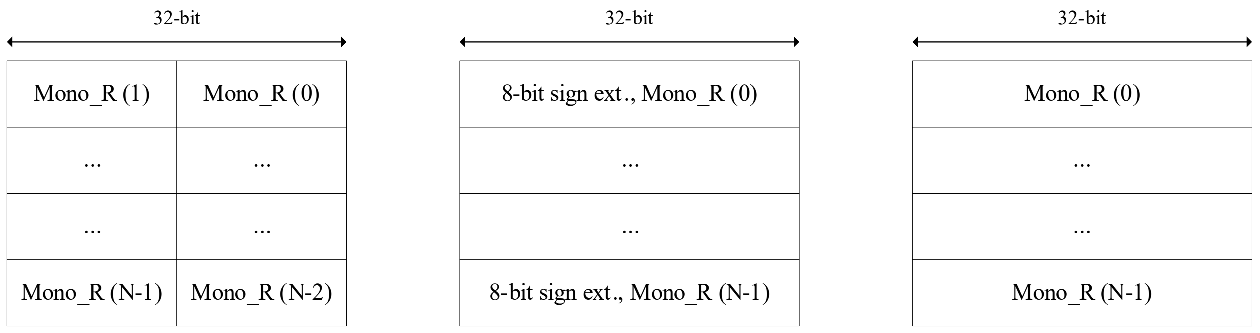
The below diagrams shows different memory content and corresponding register configurations.



- I2S_MS_SET0.cfg_tx/rx_wid = 0, 1, and 2 (16, 24 and 32-bit)
- I2S_MS_SET0.cfg_tx/rx_chn = 0 (Stereo audio sample)



- I2S_MS_SET0.cfg_tx/rx_wid = 0, 1, and 2 (16, 24 and 32-bit)
- I2S_MS_SET0.cfg_tx/rx_chn = 1 (Stereo audio sample)



- I2S_MS_SET0.cfg_tx/rx_wid = 0, 1, and 2 (16, 24 and 32-bit)
- I2S_MS_SET0.cfg_tx/rx_chn = 2 (Stereo audio sample)

16.4 Register Descriptions

16.4.1 Summary

Register Name	Offset	Access	Description
I2S_MS_CTL0	0x00	R/W	Enable MCLK generator and I2S
I2S_MS_CTL1	0x04	W1C	Reset MCLK generator and I2S
I2S_MCLK_SET0	0x08	R/W	Internal iMCLK rate selection

I2S_MCLK_SET1	0x0C	R/W	Output MCLK (to GPIO) divider
I2S_MS_SET0	0x14	R/W	I2S rate, format and Tx/Rx configurations
I2S_RDMA_CTL0	0x40	R/W	Start trigger xDMA
I2S_RDMA_CTL1	0x44	W1C	Reset xDMA
I2S_RDMA_SET0	0x48	R/W	Segment size & Block size of xDMA
I2S_RDMA_SET1	0x4C	R/W	Start address of xDMA
I2S_RDMA_R0	0x58	R	Next pointer address of xDMA
I2S_RDMA_R1	0x5C	R	Debug
I2S_WDMA_CTL0	0x60	R/W	Start trigger xDMA
I2S_WDMA_CTL1	0x64	W1C	Reset xDMA
I2S_WDMA_SET0	0x68	R/W	Segment size & Block size of xDMA
I2S_WDMA_SET1	0x6C	R/W	Start address of xDMA
I2S_WDMA_R0	0x78	R	Next pointer address of xDMA
I2S_WDMA_R1	0x7C	R	Debug
I2S_INT_CLEAR	0xA0	W1C	Clear interrupt
I2S_INT_MASK	0xA4	R/W	Mask interrupt
I2S_INT_STATUS	0xA8	R	Status of interrupt

16.4.2 I2S_MS_CTL0 (offset: 0x00)

Signal	Bits	Default	R/W	Description
cfg_mck_ena	1	b0	R/W	Enable MCLK 0: disable

				1: enable
cfg_i2s_ena	0	b0	R/W	Enable I2S 0: disable 1: enable

16.4.3 I2S_MS_CTL1 (offset: 0x04)

Signal	Bits	Default	R/W	Description
cfg_i2s_rst	0	b0	R/W	Reset I2S 0: disable 1: reset

16.4.4 I2S_MCLK_SET0 (offset: 0x08)

Signal	Bits	Default	R/W	Description
cfg_mck_isel	2:0	d0	R/W	Internal MCLK rate selection 0: average 12.288M when PLL is 32M 1: average 8.192M when PLL is 32M 2: average 12.288M when PLL is 48M 3: average 8.192M when PLL is 48M 4: average 24.576M when PLL is 64M 5: average 16.384M when PLL is 64M

16.4.5 I2S_MCLK_SET1 (offset: 0x0C)

Signal	Bits	Default	R/W	Description
cfg_mck_div	3:0	d0	R/W	Output MCLK divider 0: internal MCLK divided by 1 1: internal MCLK divided by 2 2: internal MCLK divided by 4 3: internal MCLK divided by 8 4: internal MCLK divided by 16 5: internal MCLK divided by 32 others: internal MCLK divided by 32

16.4.6 I2S_MS_SET0 (offset: 0x14)

Signal	Bits	Default	R/W	Description
cfg_bck_osr	1:0	d0	R/W	Ratio of MCLK and BCLK 0: 2 1: 4 2: 8 3: 8
cfg_i2s_mod	[3:2]		R/W	I2S transceiver mode 0: TxRx 1: Tx only

				2: Rx only 3: TxRx
cfg_i2s_fmt			R/W	I2S format 0: LJ mode 1: RJ mode 2: I2S mode 3: I2S mode
cfg_bck_len			R/W	Bit Length of BCLK per channel 0: 16 1: 24 (proprietary I2S) 2: 32 3: 32
cfg_txd_wid			R/W	Sample width for I2S transmitted sample 0: 16-bit 1: 24-bit 2: 32-bit 3: 32-bit
cfg_rxd_wid			R/W	Sample width for I2S received sample 0: 16-bit 1: 24-bit 2: 32-bit 3: 32-bit
cfg_txd_chn			R/W	Tx sample format in xdma

				0: stereo, L/R 1: mono, L 2: mono, R 3: reserved
cfg_rxd_chn			R/W	Rx sample format in xdma 0: stereo, L/R 1: mono, L 2: mono, R 3: reserved

16.4.7 I2S_RDMA_CTL0 (offset: 0x40)

Signal	Bits	Default	R/W	Description
cfg_i2s_rdma_ctl0	0	d0	W1C	[0] Start xdma for memory access 0: disable 1: enable

16.4.8 I2S_RDMA_CTL1 (offset: 0x44)

Signal	Bits	Default	R/W	Description
cfg_i2s_rdma_ctl1	0	d0	W1C	Software reset xdma

				0: disable 1: Reset

16.4.9 I2S_RDMA_SET0 (offset: 0x48)

Signal	Bits	Default	R/W	Description
cfg_i2s_rdma_set0	31:0	d0	R/W	[15:0] Segment size, 4-byte aligned 0: illegal value 1...32768: normal [31:16] Block size, 4-byte aligned 0: single mode 1..32768: normal

16.4.10 I2S_RDMA_SET1 (offset: 0x4C)

Signal	Bits	Default	R/W	Description
cfg_i2s_rdma_set1	31:0	d0	R/W	Start address point, word (4-byte) aligned The 2-LSB bit will be ignored

16.4.11 I2S_RDMA_R0 (offset: 0x58)

Signal	Bits	Default	R/W	Description
cfg_i2s_rdma_r0	31:0	d0	R	Status of xdma next pointer address

16.4.12 I2S_RDMA_R1 (offset: 0x5C)

Signal	Bits	Default	R/W	Description
cfg_i2s_rdma_r1	31:0	d0	R	[0] xdma interrupt status [1] xdma error interrupt status [2] OR (xdma interrupt status, xdma error interrupt status) [9:8] Debug, xdma counter status, used for ping-pong buffer to check buffer status

16.4.13 I2S_WDMA_CTL0 (offset: 0x60)

Signal	Bits	Default	R/W	Description
cfg_i2s_wdma_ctl0	0	d0	W1C	[0] Start xdma for memory access 0: disable 1: enable

16.4.14 I2S_WDMA_CTL1 (offset: 0x64)

Signal	Bits	Default	R/W	Description
cfg_i2s_wdma_ctl1	0	d0	W1C	Software reset xdma 0: disable 1: Reset

16.4.15 I2S_WDMA_SET0 (offset: 0x68)

Signal	Bits	Default	R/W	Description
cfg_i2s_wdma_set0	31:0	d0	R/W	[15:0] Segment size, 4-byte aligned 0: illegal value 1...32768: normal [31:16] Block size, 4-byte aligned 0: single mode 1..32768: normal

16.4.16 I2S_WDMA_SET1 (offset: 0x6C)

Signal	Bits	Default	R/W	Description
cfg_i2s_wdma_set1	31:0	d0	R/W	Start address point, word (4-byte) aligned The 2-LSB bit will be ignored

16.4.17 I2S_WDMA_R0 (offset: 0x78)

Signal	Bits	Default	R/W	Description
cfg_i2s_wdma_r0	31:0	d0	R	Status of xdma next pointer address

16.4.18 I2S_WDMA_R1 (offset: 0x7C)

Signal	Bits	Default	R/W	Description
cfg_i2s_wdma_r1	31:0	d0	R	[0] xdma interrupt status [1] xdma error interrupt status [2] OR (xdma interrupt status, xdma error interrupt status) [9:8] Debug, xdma counter status, ping-pong buffer to check buffer status

16.4.19 I2S_INT_CLEAR (offset: 0xA0)

Signal	Bits	Default	R/W	Description
cfg_i2s_int_clear	3:0	d0	W1C	[0] Clear I2S_RDMA interrupt 0: disable, 1: clear [1] Reserved [2] Clear I2S_WDMA interrupt

				0: disable, 1: clear [3] Reserved

16.4.20 I2S_INT_MASK (offset: 0xA4)

Signal	Bits	Default	R/W	Description
cfg_i2s_int_mask	3:0	d0	R/W	[0] Mask I2S_RDMA interrupt 0: No mask, 1: Mask [1] Reserved [2] Mask I2S_WDMA interrupt 0: No mask, 1: Mask [3] Reserved

16.4.21 I2S_INT_STATUS (offset: 0xA8)

Signal	Bits	Default	R/W	Description
cfg_i2s_int_status	3:0	d0	R/W	[0] Status of I2S_RDMA interrupt [1] Reserved [2] Status of I2S_WDMA interrupt [3] Reserved

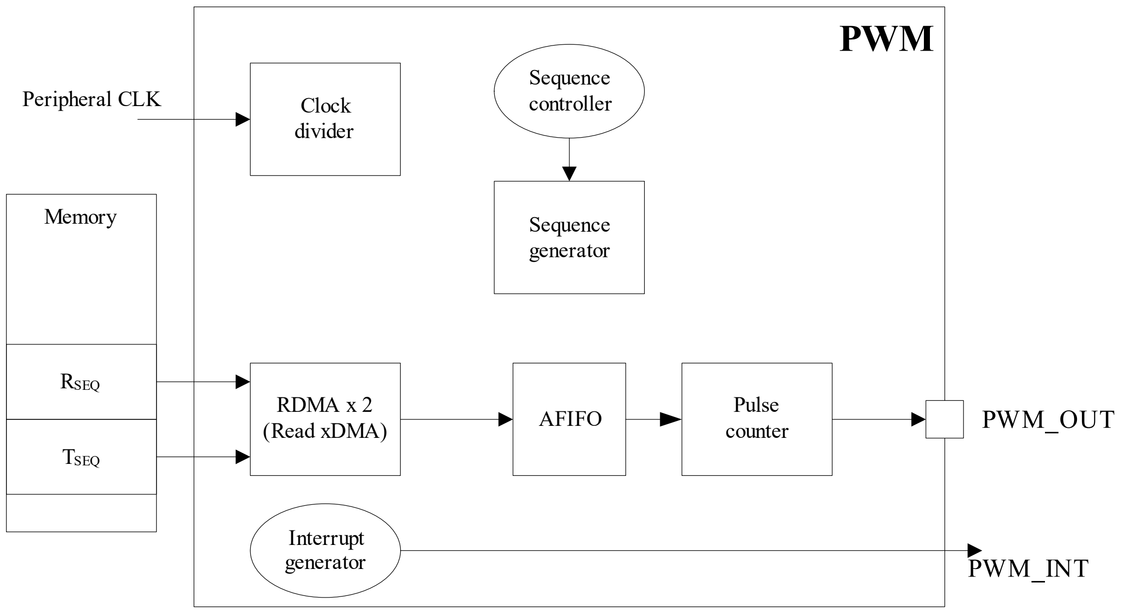
17 PWM

The PWM module generates pulse width modulated signals and drives the assigned GPIOs. It supports a pulse generator with up & up-and-down counting modes. Five PWM modules can provide up to 5 PWM channels and each channel have its own individual frequency control. A read-type xDMA is also included in the PWM module to transfer frequency control between memory and the PWM module without CPU intervention.

17.1 Features

- Programmable clock divider for PWM module
- 5 sets of PWM module and up to five channels for PWM
- PWM pulse generator with up & up-and-down counting modes
- Programmable duty-cycle sequence defined in the memory
- Duty-cycle sequence can be repeated or loop controlled.

17.2 Block Diagram



17.3 Function description

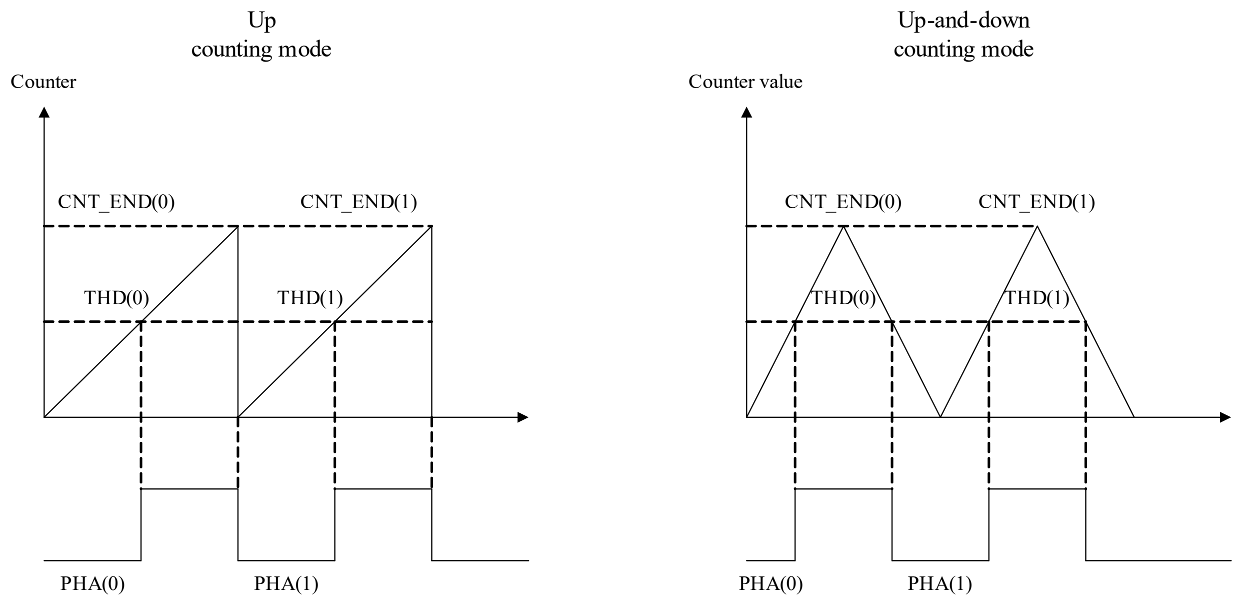
For simplicity, PWMx means PWM0, PWM1, PWM2, PWM3 or PWM4. Some examples use PWM0 as an introduction since all five PWM hardware are the same.

17.3.1 Pulse generator

The pulse generator consists a counter with up & up-and-down counting mode. The internal logic compares the threshold (THD) with the counter value and generates a defined pulse duration to the PWM output. Phase (PHA) indicates the polarity of pulse generator output when the counter starts to count from zero. Once the counter exceeds the THD it will toggle the PWM output.

For the Up counting mode, the counter increases automatically one by one. The counter will wrap around to zero whenever the counter reaches counter end value (CNT_END).

For Up-and-down counting mode, the counter increases automatically one by one. The counter will stop increasing and decrease one by one whenever the counter reaches counter end value (CNT_END).



17.3.2 xDMA and sample format

Each PWM module has two xDMA with read types RDMA0 and RDMA1. Each xDMA addressing mode is word (4-bytes) alignment.

- PWMx_RDMA0 addressing mode with word alignment

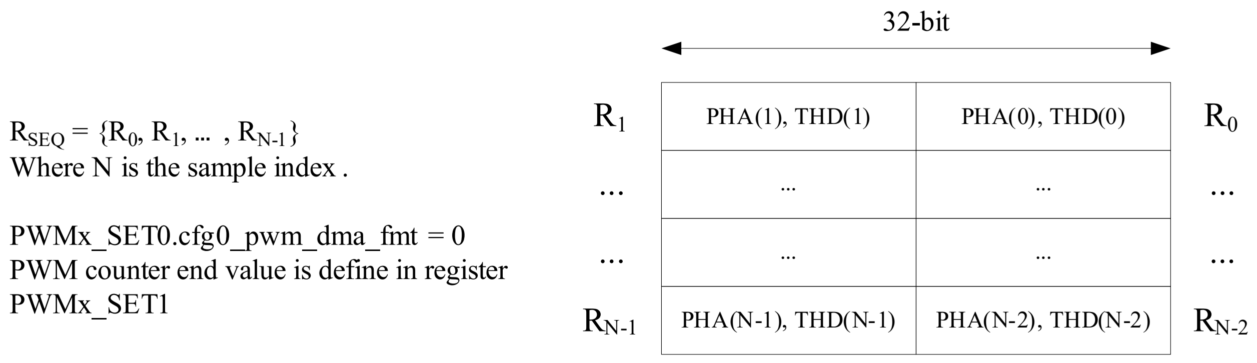
For RDMA0, the segment size and block size are configured from PWMx_RDMA0_SET0. The start address for memory reading is configured from PWMx_RDMA0_SET1. The PWMx_RDMA0_CTL0 is used to trigger RDMA for memory reading. The sample sequence stored in memory is configured from PWMx_SET0.

- PWMx_RDMA1 addressing mode with word alignment

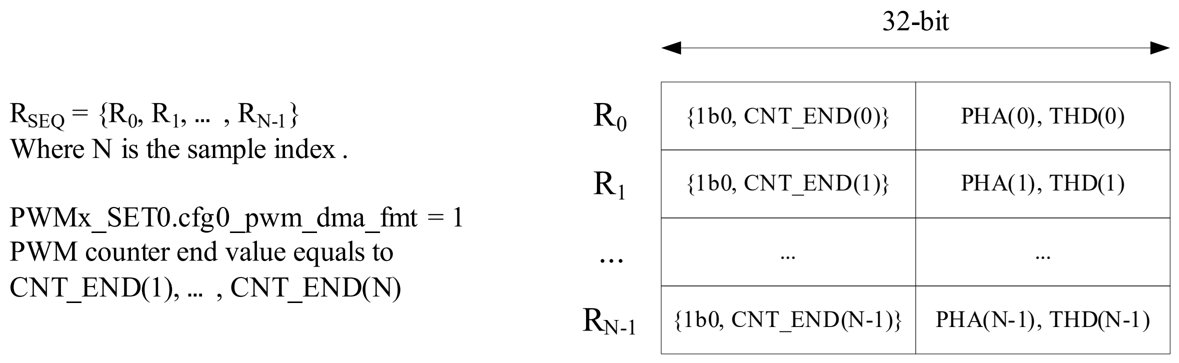
For RDMA1, the segment size and block size are configured from PWMx_RDMA1_SET0. The start address point for memory reading is configured from PWMx_RDMA1_SET1. The PWMx_RDMA0_CTL0 triggers RDMA to start memory reading. The sample sequence stored in memory is configured from PWMx_SET0.

The RDMA0 reads sequence R_{SEQ} from a memory range that is defined by PWMx_RDMA0_SET0~1. The RDMA1 reads sequence T_{SEQ} from a memory range that is defined by PWMx_RDMA1_SET0~1. The samples of both sequences stored in the memory have two formats.

- Format-0 is each 32-bit memory data representing two samples with phase and threshold information. The 32-bits consist of two of 1-bit PHA and two 15-bit THD values. The counter end value depends on the PWMx_SET1 register.


$R_{SEQ} = \{R_0, R_1, \dots, R_{N-1}\}$
Where N is the sample index .
PWMx_SET0.cfg0_pwm_dma_fmt = 0
PWM counter end value is define in register PWMx_SET1
32-bit
R_1
PHA(1), THD(1)
PHA(0), THD(0)
R_0
R_{N-1}
PHA(N-1), THD(N-1)
PHA(N-2), THD(N-2)
R_{N-2}

- Format-1 is each 32-bit of memory data representing one sample with counter end value, phase and threshold information. The 32-bits consist of a 16-bit CNT_END, 1-bit PHA and 15-bit THD value.


$R_{SEQ} = \{R_0, R_1, \dots, R_{N-1}\}$
Where N is the sample index .
PWMx_SET0.cfg0_pwm_dma_fmt = 1
PWM counter end value equals to CNT_END(1), … , CNT_END(N)
32-bit
R_0
{1b0, CNT_END(0)}
PHA(0), THD(0)
R_1
{1b0, CNT_END(1)}
PHA(1), THD(1)
R_{N-1}
{1b0, CNT_END(N-1)}
PHA(N-1), THD(N-1)

When R_{SEQ} sample format is defined, the T_{SEQ} sample format of memory content is as same as R_{SEQ}.

17.3.3 Sequence controller

The sequence controller for R_{SEQ} and T_{SEQ} has two modes: single sequence mode and two sequence mode. Each sample of the sequence can also be manipulated by the sequence controller. It provides repeat and delay of each sample of the sequence.

Below is an example of single sequence mode for R_{SEQ} that has 5 samples stored in the memory, repeat 1-time and end sequence with 2-sample delay by repeating last sample.

Singe sequence mode, R_{SEQ} is 5-samples, repeat 1-time and end with delay 2-sample.	

**Sequence controller
configurations**
	PWM0_SET0. cfg0_seq_order = 0 (play R_{SEQ} 1st)
	PWM0_SET1.cfg0_pwm_cnt_end = 100 (counter ends at 99)
	PWM0_SET2.cfg0_seqx_pcnt = 0 (single sequence)

**R_{SEQ}
register configurations**
	PWM0_SET3.cfg0_seq0_num=5 (number of sample)
	PWM0_SET4.cfg0_seq0_rpt=1 (repeat number of sample)
	PWM0_SET5.cfg0_seq0_dly=2 (delay 2-sample at last)

R_{SEQ}	$\{R_0, R_1, \ldots, R_4\}$, stored in the memory

PWM output	{R_0, R_0, R_1, R_1, ..., R_4, R_4, $R_{4(D)}$, $R_{4(D)}$} $R_{4(D)}$ are sample delay by repeating last sample

Another example of a two sequence mode for R_{SEQ} that has 5 samples stored in the memory, repeat 1-time and end sequence with 2-sample delay by repeating last sample. T_{SEQ} that has 5 samples stored in the memory, repeat 0-time and end sequence with 1-sample delay by repeating last sample. The two sequence can play in a loop several times according to PWM0_SET2.cfg0_seqx_pcnt.

Two sequence mode, R_{SEQ} is 5-samples, repeat 1-time and end with delay 2-sample. T_{SEQ} is 5-samples, repeat 0-time and end with delay 1-sample.	
Sequence controller configurations	PWM0_SET0. cfg0_seq_order = 0 (play R_{SEQ} 1st, then play T_{SEQ}) PWM0_SET1.cfg0_pwm_cnt_end = 100 (counter ends at 99) PWM0_SET2.cfg0_seqx_pcnt = 4 (two sequences with loop play by 4)

**R_{SEQ}
 	
register configurations**
	PWM0_SET3.cfg0_seq0_num=5 (number of sample)
	
	PWM0_SET4.cfg0_seq0_rpt=1 (repeat number of sample)
	
	PWM0_SET5.cfg0_seq0_dly=2 (delay 2-sample at last)

**T_{SEQ}
 	
register configurations**
	PWM0_SET3.cfg0_seq1_num=5 (number of sample)
	
	PWM0_SET4.cfg0_seq1_rpt=0 (repeat number of sample)
	
	PWM0_SET5.cfg0_seq1_dly=2 (delay 2-sample at last)

R_{SEQ}	{R_0, R_1, …, R_4}, stored in the memory
T_{SEQ}	{T_0, T_1, …, T_4}, stored in the memory
PWM output	{R_0, R_0, R_1, R_1, ..., R_4, R_4, $R_{4(D)}$, $R_{4(D)}$, T_0, T_1, …, T_4, $T_{4(D)}$}, {R_0, R_0, R_1, R_1, ..., R_4, R_4, $R_{4(D)}$, $R_{4(D)}$, T_0, T_1, …, T_4, $T_{4(D)}$},

	{R_0, R_0, R_1, R_1, ..., R_4, R_4, $R_{4(D)}$, $R_{4(D)}$, T_0, T_1, …, T_4, $T_{4(D)}$}, {R_0, R_0, R_1, R_1, ..., R_4, R_4, $R_{4(D)}$, $R_{4(D)}$, T_0, T_1, …, T_4, $T_{4(D)}$}. $R_{4(D)}$ represent sample delay by repeating last sample $T_{4(D)}$ represent sample delay by repeating last sample

17.4 Register Descriptions

17.4.1 Summary

The PWM0 ~ PWM4 are the same hardware design except the control register base addresses have a different value.

Start Address	End Address	Size	Module
0xA0C00000	0xA0C000FF	256B	PWM0
0xA0C00100	0xA0C001FF	256B	PWM1
0xA0C00200	0xA0C002FF	256B	PWM2
0xA0C00300	0xA0C003FF	256B	PWM3
0xA0C00400	0xA0C004FF	256B	PWM4
0xA0C00500	0xA0CFFFFF		*Reserved*

The register name and offset from start address to the end address of PWM0 to PWM4 are also the same. For simplicity, below describes the register for PWM0.

Register Name	Offset	Access	Description
PWM0_CTL0	0x000	R/W	Enable PWM

PWM0_CTL1	0x004	W1C	Reset PWM
PWM0_SET0	0x008	R/W	PWM sequence configurations
PWM0_SET1	0x00C	R/W	PWM counter end value
PWM0_SET2	0x010	R/W	Amount of playback
PWM0_SET3	0x014	R/W	Number of element, RSEQ
PWM0_SET4	0x018	R/W	Number of repeat, RSEQ
PWM0_SET5	0x01C	R/W	Number of delay, RSEQ
PWM0_SET6	0x020	R/W	Number of element, TSEQ
PWM0_SET7	0x024	R/W	Number of repeat, RSEQ
PWM0_SET8	0x028	R/W	Number of delay, TSEQ
PWM0_RDMA0_CTL0	0x040	R/W	Start trigger xDMA
PWM0_RDMA0_CTL1	0x044	W1C	Reset xDMA
PWM0_RDMA0_SET0	0x048	R/W	Segment size & Block size of xDMA
PWM0_RDMA0_SET1	0x04C	R/W	Start address of xDMA
PWM0_RDMA0_R0	0x058	R	Next pointer address of xDMA
PWM0_RDMA0_R1	0x05C	R	Debug
PWM0_RDMA1_CTL0	0x060	R/W	Start trigger xDMA
PWM0_RDMA1_CTL1	0x064	W1C	Reset xDMA
PWM0_RDMA1_SET0	0x068	R/W	Segment size & Block size of xDMA
PWM0_RDMA1_SET1	0x06C	R/W	Start address of xDMA
PWM0_RDMA1_R0	0x078	R	Next pointer address of xDMA
PWM0_RDMA1_R1	0x07C	R	Debug

PWM0_INT_CLEAR	0x0A0	W1C	Clear interrupt
PWM0_INT_MASK	0x0A4	R/W	Mask interrupt
PWM0_INT_STATUS	0x0A8	R	Status of interrupt

17.4.2 PWM0_CTL0 (offset: 0x000)

Signal	Bits	Default	R/W	Description
cfg0_ck_ena	1	b0	R/W	Enable pwm gated clock 0: disable 1: enable
cfg0_pwm_ena	0	b0	R/W	Enable pwm 0: disable 1: enable

17.4.3 PWM0_CTL1 (offset: 0x004)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rst	0	b0	W1C	Reset pwm 0: disable 1: reset

17.4.4 PWM0_SET0 (offset: 0x008)

Signal	Bits	Default	R/W	Description

cfg0_pwm_ena_trig	14:12	d7	R/W	PWM counter enable trigger by other PWM 0: trigger whenever PWM0 sequence finish 1: trigger whenever PWM1 sequence finish 2: trigger whenever PWM2 sequence finish 3: trigger whenever PWM3 sequence finish 4: trigger whenever PWM4 sequence finish others: self-trigger Note: if cfgX_seqx_cnt is 65535, trigger whenever PWMX 1st sequence finishes where X is 0~4.
cfg0_ck_div	11:8	d0	R/W	PWM gated clock divider value 0: clock divided by 1 1: clock divided by 2 2: clock divided by 4 3: clock divided by 8 4: clock divided by 16 5: clock divided by 32 6: clock divided by 64 7: clock divided by 128 8: clock divided by 256 otherwise: clock divided by 256
cfg0_dma_auto	6	d1	R/W	Auto Trigger start DMA when the amount of playback 2-sequence play does not reach to cfgx_seqx_pcnt

				0: No auto trigger 1: Auto trigger
cfg0_pwm_cnt_trig	5	d0	R/W	PWM counter trigger 0: trigger by cfg_pwm_ena 1: trigger by cfg_pwm_ena & FIFO not empty
cfg0_pwm_cnt_mode	4	d0	R/W	PWM counter mode 0: Up counter 1: Up and down counter
cfg0_pwm_dma_fmt	3	d0	R/W	PWM dma sample format 0: Format-0, 32-bit is [PHA(N), THD(N), PHA(N-1), THD(N-1)] 1: Format-1, 32-bit is [1’b0, CNT_END(N), PHA(N), THD(N)]
cfg0_seq_mode	2	d0	R/W	PWM sequence play mode 0: non-continuous play 1: continuous play
cfg0_seq_two_sel	1	d0	R/W	Two Sequence selection 0: One sequence 1: Two sequence
cfg0_seq_order	0	d0	R/W	PWM sequence play order 0: RSEQ 1st 1: TSEQ 1st

17.4.5 PWM0_SET1 (offset: 0x00C)

Signal	Bits	Default	R/W	Description
cfg0_pwm_cnt_end	14:0	d0	R/W	PWM counter end value, ignored when cfg0_pwm_dma_fmt is at Format-1 value: 3…32767

17.4.6 PWM0_SET2 (offset: 0x010)

Signal	Bits	Default	R/W	Description
cfg0_seqx_pcnt	15:0	d0	R/W	Amount of sequence play 0: Play one/two sequence 1 time N: Play one/two sequence N-times 65535: Play one/two sequence infinitely

17.4.7 PWM0_SET3 (offset: 0x014)

Signal	Bits	Default	R/W	Description
cfg0_seq0_num	15:0	d0	R/W	Number of element in RSEQ

17.4.8 PWM0_SET4 (offset: 0x018)

Signal	Bits	Default	R/W	Description
cfg0_seq0_rpt	15:0	d0	R/W	Number of repeat for each element in RSEQ

17.4.9 PWM0_SET5 (offset: 0x01C)

Signal	Bits	Default	R/W	Description
cfg0_seq0_dly	15:0	d0	R/W	Number of delay after RSEQ is play finish

17.4.10 PWM0_SET6 (offset: 0x020)

Signal	Bits	Default	R/W	Description
cfg0_seq1_num	15:0	d0	R/W	Number of element in TSEQ

17.4.11 PWM0_SET7 (offset: 0x024)

Signal	Bits	Default	R/W	Description
cfg0_seq1_rpt	15:0	d0	R/W	Number of repeat for each element in TSEQ

17.4.12 PWM0_SET8 (offset: 0x028)

Signal	Bits	Default	R/W	Description
cfg0_seq1_dly	15:0	d0	R/W	Number of delay after TSEQ is play finish

17.4.13 PWM0_RDMA0_CTL0 (offset: 0x040)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma0_ctl0	0	d0	W1C	[0] Start xdma for memory access 0: disable 1: enable

17.4.14 PWM0_RDMA0_CTL1 (offset: 0x044)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma0_ctl1	0	d0	W1C	Software reset xdma 0: disable 1: Reset

17.4.15 PWM0_RDMA0_SET0 (offset: 0x048)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma0_set0	31:0	d0	R/W	[15:0] Segment size, 4-byte aligned 0: Illegal value Others: 1…65535 (Block size < Segment size) [31:16] Block size, 4-byte aligned 0: Single-shot mode

				Others: 1…65535 (Block size < Segment size)

17.4.16 PWM0_RDMA0_SET1 (offset: 0x04C)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma0_set1	31:0	d0	R/W	Start address point, 4-byte aligned The 2-LSB bit will be ignored

17.4.17 PWM0_RDMA0_R0 (offset: 0x058)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma0_r0	31:0	d0	R	Status of xdma next pointer address

17.4.18 PWM_RDMA0_R1 (offset: 0x05C)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma0_r1	31:0	d0	R	[0] xdma interrupt status [1] xdma error interrupt status [2] OR(xdma interrupt status, xdma error interrupt status) [9:8] Debug, xdma counter status, used for ping-pong buffer to check buffer status

17.4.19 PWM0_RDMA1_CTL0 (offset: 0x060)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma1_ctl0	0	d0	W1C	[0] Start xdma for memory access 0: disable 1: enable

17.4.20 PWM0_RDMA1_CTL1 (offset: 0x064)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma1_ctl1	0	d0	W1C	Software reset xdma 0: disable 1: Reset

17.4.21 PWM0_RDMA1_SET0 (offset: 0x068)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma1_set0	31:0	d0	R/W	[15:0] Segment size, 4-byte aligned 0: Illegal value Others: 1…65535 (Block size < Segment size) [31:16] Block size, 4-byte aligned 0: Single-shot mode

				Others: 1…65535 (Block size < Segment size)

17.4.22 PWM0_RDMA1_SET1 (offset: 0x06C)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma1_set1	31:0	d0	R/W	Start address point, 4-byte aligned The 2-LSB bit will be ignored

17.4.23 PWM0_RDMA1_R0 (offset: 0x078)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma1_r0	31:0	d0	R	Status of xdma next pointer address

17.4.24 PWM0_RDMA1_R1 (offset: 0x07C)

Signal	Bits	Default	R/W	Description
cfg0_pwm_rdma1_r1	31:0	d0	R	[0] xdma interrupt status [1] xdma error interrupt status [2] OR(xdma interrupt status, xdma error interrupt status) [9:8] Debug, xdma counter status, used for ping-pong buffer to check buffer status

17.4.25 PWM0_INT_CLEAR (offset: 0x0A0)

Signal	Bits	Default	R/W	Description
cfg0_pwm_int_clear	6:0	d0	W1C	[0] Clear RDMA0 interrupt; 0: disable, 1: clear [1] Reserved [2] Clear RDMA1 interrupt; 0: disable, 1: clear [3] Reserved [4] Clear RSEQ done interrupt; 0: disable, 1: clear [5] Clear TSEQ done interrupt; 0: disable, 1: clear [6] Clear one/two sequence*cfgx_seqx_pcnt done interrupt; 0: disable, 1: clear

17.4.26 PWM0_INT_MASK (offset: 0xA4)

Signal	Bits	Default	R/W	Description
cfg0_pwm_int_mask	6:0	7'b1110000	R/W	[0] Mask RDMA0 interrupt; 0: No mask 1: Mask

				[1] Reserved [2] Mask RDMA1 interrupt; 0: No mask 1: Mask [3] Reserved [4] Mask RSEQ done interrupt; 0: No mask 1: Mask [5] Mask TSEQ done interrupt; 0: No mask 1: Mask [6] Mask one/two sequence*cfgx_seqx_pcnt done interrupt; 0: No mask 1: Mask

17.4.27 PWM0_INT_STATUS (offset: 0xA8)

Signal	Bits	Default	R/W	Description
cfg0_pwm_int_status	6:0	d0	R/W	[0] Status of RDMA0 interrupt [1] Reserved [2] Status of RDMA1 interrupt [3] Reserved [4] Status of RSEQ done interrupt [5] Status of TSEQ done interrupt [6] Status of one/two sequence*cfgx_seqx_pcnt done interrupt

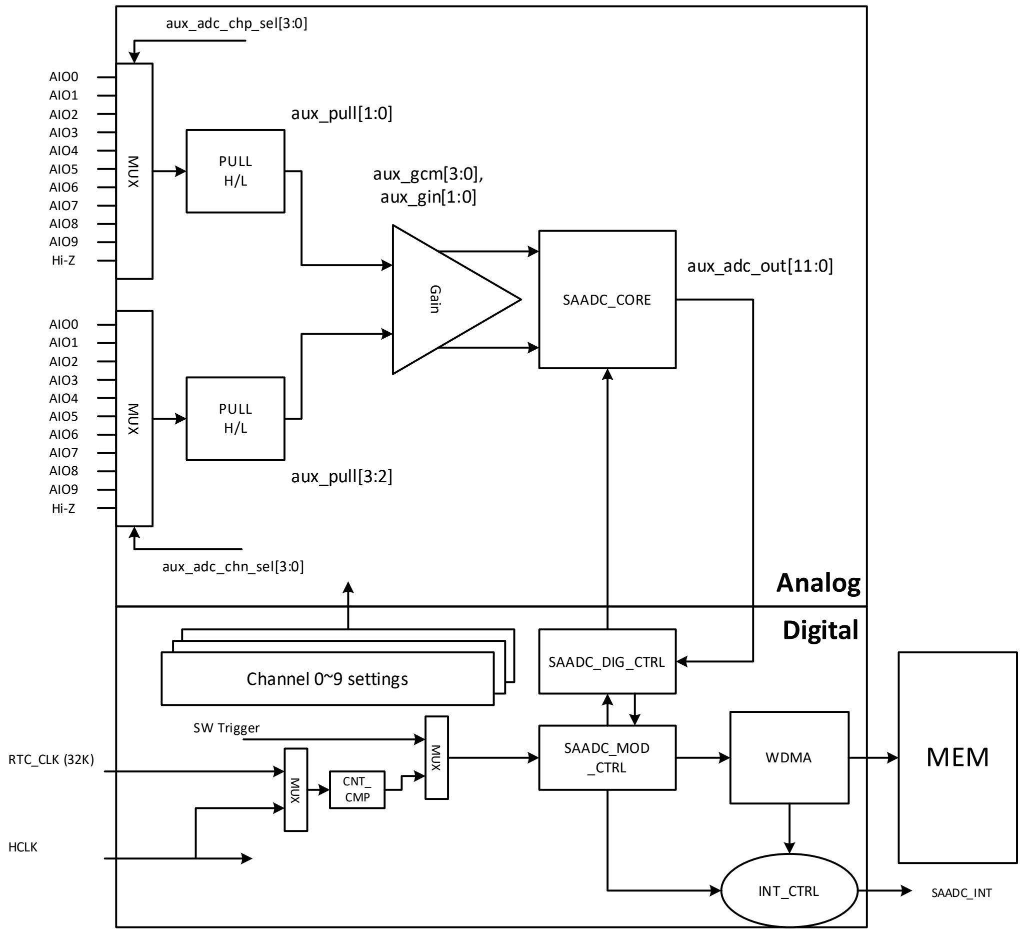
18 SADC

The SADC or AUX_ADC is a differential Successive approximation analog to digital converter. The SADC module is used for measuring a voltage from IO. It has maximum 10-channels for single-ended or 5-channels for differential. Various sampling modes are provided and one WDMA is included for MCU accessing the memory written by the SADC.

18.1 Features

- 8/10/12-bit resolution, and 14-bit resolution at oversampling rate condition
- Maximum 10 input channels
  - One channel for singled ended and two channels for differential
- Hardware sampling trigger from RTC 32K and HCLK
- One-shot, timer and scan mode for channel measuring
- One WDMA can transfer SADC samples to memory with unsigned 16-bit format
- Monitoring SADC sample results of each channel

18.2 Block Diagram



18.3 Function description

The SADC supports up to 10 external analog input channels. Each channel has its own gain, acquisition and end of delay time configuration. The SADC consists two parts: analog to digital converter circuit and a digital controller. The mode controller (sadc_mod_ctrl) defines the operation mode for different usage scenarios. The mode controller can operate with one-shot, timer or scan mode. The digital controller (sadc_dig_ctrl) is used for controlling the SADC analog circuit, sampling the SADC result and sending result to memory via WDMA. The SADC result can be configured to 8, 10, 12-bit and 14-bit resolution with oversampling. The result is unsigned format and it appends zero bits at the MSB to 16-bit format then writes into memory.

18.3.1 One shot mode

One shot mode enables only one channel of SADC_PNSEL_CH0~CH9. Configure register SADC_SET0.[0] sets the trigger source is from the control register. The MCU enables the controller via SADC_CTL0 and triggers the mode controller via SADC_CTRL2 by single shot. It sends a start request to the digital controller. Whenever the digital controller receives start request, it begins sampling the SADC result according to the channel configurations.

18.3.2 Timer mode

Timer mode enables only one channel of SADC_PNSEL_CH0~CH9. Configure the register SADC_SET0.[0] to set the trigger source is from timer. The MCU enables the controller via SADC_CTL0 and triggers periodically by timer. It sends a start request to the digital controller periodically. Whenever the digital controller receives a start request, it begins sampling the SADC results according to the channel configurations.

18.3.3 Scan mode

Scan mode enables two channels or more of SADC_PNSEL_CH0~CH9. Configure the register SADC_SET0.[0] to set the trigger source from the control register. The MCU enables the controller via SADC_CTL0 and triggers the mode controller SADC_CTRL2 by one-shot. It sends a start request to the digital controller for enabling the channels one by one according to the highest priority from CH0 to CH9. Whenever the digital controller receives a start request, it begins to sample the SADC result according to the channel gain, acquisition time and end of delay time.

18.3.4 Analog input channels

The channel registers for SADC_PNSEL_CH0 through SADC_PNSEL_CH9 are all the same. Below is an introduction to CH0.

Channel-0 register	Function	Description
SADC_PNSEL_CH0[3:0]	0~9: AIN0~AIN9 for ADC differential P	Connect to AIO;
	>9: High impendence	No Connection
SADC_PNSEL_CH0[7:4]	0~9: AIN0~AIN9 for ADC differential N	Connect to AIO;
	>9: High impendence	No Connection
SADC_PNSEL_CH0[11:8]	Gain settings, +3dB/Step	
SADC_PNSEL_CH0[15:14]	Gain settings, +6dB/Step	
SADC_PNSEL_CH0[17:16]	2b01, pull high 2b10, pull low 2b11, vcm mode 2b11, floating	Pull high or low for differential P
SADC_PNSEL_CH0[19:18]	2b01, pull high 2b10, pull low 2b11, vcm mode 2b11, floating	Pull high or low for differential N
SADC_PNSEL_CH0[26:24]	1µs to 16µs at 32M system clock	SADC Acquisition time
SADC_PNSEL_CH0[30:28]	1µs to 16µs at 32M system clock	SADC end of delay time

18.3.5 Oversampling mode

The SADC has 12-bit resolution and it can achieve 14-bit resolution by oversampling. Set SADC_SET1[11:8] to a non-zero value, it will oversample SADC result by OSR times and average by the OSR value. The OSR value ranges from 2 to 256. The SADC_SET1[3:0] controls the bit resolution sending to WDMA.

18.3.6 Result monitor

Each channel has its own monitor for value detection and is defined in SADC_THD_CH0~9. The lower part of SADC_THD_CH0[13:0] defines the low threshold value and if the SADC result is lower than this value, the monitor will send an interrupt. The higher bits of SADC_THD_CH0[29:16] define the high threshold value and if the SADC result is higher than this value, the monitor will send an interrupt. The two interrupts are different.

18.3.7 WDMA of SADC

The WDMA is a 2-byte data access and each SADC sample written to memory is 16-bit. The block size, segment size and address of WDMA is 2-byte alignment. The control method of WDMA is the same as described for xDMA.

18.3.8 Interrupt list

The interrupt mask, clear and status are listed in SADC_INT_MASK, SADC_INT_CLEAR and SADC_INT_STATUS. Each bit field has different a meaning as listed below.

Interrupt status bit fields	Description

SADC_INT_STATUS[0]	Status of WDMA interrupt, issued whenever the sample written to memory exceeds the block size
SADC_INT_STATUS[1]	Used for WDMA error
SADC_INT_STATUS[2]	When sadc_dig_ctrl finishes one SADC sampling, it will issue a one-time interrupt. If oversampling is enabled, a number of interrupts are asserted according to the oversampling ratio.
SADC_INT_STATUS[3]	When sadc_dig_ctrl finishes one SADC sampling or SADC sampling with oversampling, it will issue one-time interrupt.
SADC_INT_STATUS[4]	For one-shot mode, it will issue a one-time interrupt whenever the mode operation for one channel is finished. For timer mode, it will issue interrupts periodically whenever the mode operation for one channel is finished and triggered by the timer for the next time. For scan mode, it will issue a one-time interrupt whenever the mode operation for two channels or more are finished.
SADC_INT_STATUS[17:8]	Status of the SADC monitor low threshold value interrupt for CH0 to CH9.
SADC_INT_STATUS[27:18]	Status of the SADC monitor high threshold value interrupt for CH0 to CH9.

18.4 Register Descriptions

Register Name	Offset	Access	Description
SADC_CTL0	0x000	R/W	Enable SADC

SADC_CTL1	0x004	W1C	Reset SADC
SADC_CTL2	0x008	W1C	Start trigger to SADC sampling
SADC_SET0	0x00C	R/W	Timer settings
SADC_SET1	0x010	R/W	Bit oversampling and adjust value
SADC_PNSEL_CH0	0x020	R/W	Configurations to CH0
SADC_SET0_CH0	0x024	R/W	Burst mode for CH0
SADC_THD_CH0	0x028	R/W	Monitor threshold for CH0
SADC_PNSEL_CH1~ **SADC_THD_CH9**	0x030~ 0x0B8	R/W	Configurations to CH1~9 Burst mode for CH1~9 Monitor threshold for CH1~9
SADC_ANA_SET0	0x0BC	R/W	Analog AUX ADC settings
SADC_ANA_SET1	0x0C0	R/W	Analog AUX ADC settings
SADC_WDMA_CTL0	0x100	R/W	Start trigger xDMA
SADC_WDMA_CTL1	0x104	W1C	Reset xDMA
SADC_WDMA_SET0	0x108	R/W	Segment size & Block size of xDMA
SADC_WDMA_SET1	0x10C	R/W	Start address of xDMA
SADC_WDMA_R0	0x114	R	Next pointer address of xDMA
SADC_WDMA_R1	0x118	R	Debug
SADC_INT_CLEAR	0x120	W1C	Clear interrupt
SADC_INT_MASK	0x124	R/W	Mask interrupt
SADC_INT_STATUS	0x128	R	Status of interrupt
SADC_R0	0x12C	R	SADC result from digital controller with oversampling

SDAC_R1	0x130	R	SADC result from digital controller
SDAC_R2	0x130	R	SADC result from analog directly

18.4.1 SADC_CTL0 (offset: 0x000)

Signal	Bits	Default	R/W	Description
cfg_sadc_ena	0	b0	R/W	Enable SADC 0: disable 1: enable
cfg_sadc_ck_free	9:8	b0	R/W	[8]: Enable SADC register clock free-run 0: Auto gated 1: Free-run [9]: Enable SADC engine clock free-run 0: Auto gated 1: Free-run

18.4.2 SADC_CTL1 (offset: 0x004)

Signal	Bits	Default	R/W	Description

cfg_sadc_rst	0	b0	W1C	Reset SADC 0: disable 1: reset
cfg_sadc_afifo_rst	8	b0	W1C	Reset SADC AFIFO (Default is no reset) 0: Disable 1: Reset

18.4.3 SADC_CTL2 (offset: 0x008)

Signal	Bits	Default	R/W	Description
cfg_sadc_start	0	b0	W1C	Software start SADC 0: disable 1: start

18.4.4 SADC_SET0 (offset: 0x00C)

Signal	Bits	Default	R/W	Description
cfg_sadc_smp_mode	0	b0	R/W	Sample rate mode 0: Sample rate depends on software start SADC, cfg_sadc_start 1: Sample rate depends on timer rate1: start
cfg_sadc_tmr_cksel	1	b0	R/W	Timer clock source selection

				0: System clock 1: Slow clock
cfg_sadc_afifo_ckpsel	2	b0	R/W	AFIFO Clock phase selection 0: Write AFIFO at rising edge of clock 1: Write AFIFO at falling edge of clock
cfg_sadc_dbg_sel	6:3	d0	R/W	Debug monitor selection
cfg_sadc_tmr_ckdiv	31:16	d0	R/W	Timer rate configuration Timer rate = timer clock / (cfg_sadc_ckdiv+1) where cfg_sadc_ckdiv = 2…65535

18.4.5 SADC_SET1 (offset: 0x010)

Signal	Bits	Default	R/W	Description
cfg_sadc_bit	3:0	d2	R/W	SAC output Resolution 0: 8-bit 1: 10-bit 2: 12-bit 3: 14-bit (oversample)
cfg_sadc_chx_sel	7:4	d0	R/W	At test mode, select from CH0 to CH9
cfg_sadc_osr	11:8	d0	R/W	Oversample rate selection 0: No oversample 1: Oversample by 2 2: Oversample by 4

				3: Oversample by 8 4: Oversample by 16 5: Oversample by 32 6: Oversample by 64 7: Oversample by 128 8: Oversample by 256 Others: reserved
cfg_sadc_tst	15:12	d0	R/W	[0] SADC value bypass 0: sadc_i = sadc_i + cfg_sadc_val_tst, used for calibration 1: sadc_i = cfg_sadc_val_tst, used for bypass [1] SADC MSB bit inversion 0: disable 1: bit reverse [2] SADC control mode 1 0: normal mode, control by digital 1: manual mode, control by cfg_sadc_chx_sel [3] SADC control mode 2 0: normal mode, control by digital 1: manual mode, control by cfg_sadc_ena, cfg_sadc_vga_ena, cfg_sadc_ldo_ena respectively.
cfg_sadc_val_tst	27:16	d0	R/W	Set SADC adjust value, used for calibration

				Data format is s1.11.0

18.4.6 SADC_PNSEL_CH0 (offset: 0x020)

Signal	Bits	Default	R/W	Description
cfg_sadc_psel_ch0	3:0	d15	R/W	SADC P channel selection 0: AIN0 1: AIN1 2: AIN2 3: AIN3 4: AIN4 5: AIN5 6: AIN6 7: AIN7 8: AIN8 9: AIN9, bypass VGA Others: NC, no connection
cfg_sadc_nsel_ch0	7:4	d0	R/W	SADC N channel selection 0: AIN0 1: AIN1 2: AIN2 3: AIN3 4: AIN4

				5: AIN5 6: AIN6 7: AIN7 8: AIN8 9: AIN9, bypass VGA Others: NC, no connection
cfg_sadc_gain_ch0	13:8	d0	R/W	[3:0] VGA gain selection, +3dB/step [5:4] VGA gain selection, +6dB/step
cfg_sadc_pull_ch0	19:16	d0	R/W	[0] P channel pull high; 1:enable, 0: disable [1] P channel pull low ; 1:enable, 0: disable [2] N channel pull high; 1:enable, 0: disable [3] N channel pull low ; 1:enable, 0: disable if [1:0] = 2b11, channel P is at VCM voltage and used for single-ended if [3:2] = 2b11, channel N is at VCM voltage and used for single-ended
cfg_sadc_tacq_ch0	26:24	d0	R/W	SADC result acquisition time for system clock 32M 0: 0.3μs 1: 1μs 2: 2μs 3: 3μs 4: 4μs 5: 8μs

				6: 12μs 7: 16μs
cfg_sadc_edly_ch0	30:28	d0	R/W	SADC end delay time for system clock 32M 0: 0.3μs 1: 1μs 2: 2μs 3: 3μs 4: 4μs 5: 8μs 6: 12μs 7: 16μs

18.4.7 SADC_SET_CH0 (offset: 0x024)

Signal	Bits	Default	R/W	Description
cfg_sadc_burst_ch0	31	d1	R/W	SADC Burst mode selection 0: Disable burst mode 1: Select burst mode, SADC takes 2^Oversample number of samples and sends the average to Memory

18.4.8 SADC_THD_CH0 (offset: 0x028)

Signal	Bits	Default	R/W	Description
cfg_sadc_lthd_ch0	13:0	d0	R/W	SADC low threshold, u0.14.0
cfg_sadc_hthd_ch0	29:16	d0	R/W	SADC high threshold, u0.14.0

18.4.9 SADC_PNSEL_CH1 (offset: 0x030), same as CH0

18.4.10 SADC_SET_CH1 (offset: 0x034), same as CH0

18.4.11 SADC_THD_CH1 (offset: 0x038), same as CH0

18.4.12 SADC_PNSEL_CH2 (offset: 0x040), same as CH0

18.4.13 SADC_SET_CH2 (offset: 0x044), same as CH0

18.4.14 SADC_THD_CH2 (offset: 0x048), same as CH0

18.4.15 SADC_PNSEL_CH3 (offset: 0x050), same as CH0

18.4.16 SADC_SET_CH3 (offset: 0x054), same as CH0

18.4.17 SADC_THD_CH3 (offset: 0x058), same as CH0

18.4.18 SADC_PNSEL_CH4 (offset: 0x060), same as CH0

18.4.19 SADC_SET_CH4 (offset: 0x064), same as CH0

18.4.20 SADC_THD_CH4 (offset: 0x068), same as CH0

18.4.21 SADC_PNSEL_CH5 (offset: 0x070), same as CH0

18.4.22 SADC_SET_CH5 (offset: 0x074), same as CH0

18.4.23 SADC_THD_CH5 (offset: 0x078), same as CH0

18.4.24 SADC_PNSEL_CH6 (offset: 0x080), same as CH0

18.4.25 SADC_SET_CH6 (offset: 0x084), same as CH0

18.4.26 SADC_THD_CH6 (offset: 0x088), same as CH0

18.4.27 SADC_PNSEL_CH7 (offset: 0x090), same as CH0

18.4.28 SADC_SET_CH7 (offset: 0x094), same as CH0

18.4.29 SADC_THD_CH7 (offset: 0x098), same as CH0

18.4.30 SADC_PNSEL_CH8 (offset: 0x0A0), same as CH0

18.4.31 SADC_SET_CH8 (offset: 0x0A4), same as CH0

18.4.32 SADC_THD_CH8 (offset: 0x0A8), same as CH0

18.4.33 SADC_PNSEL_CH9 (offset: 0x0B0), same as CH0

18.4.34 SADC_SET_CH9 (offset: 0x0B4), same as CH0

18.4.35 SADC_THD_CH9 (offset: 0x0B8), same as CH0

18.4.36 SADC_ANA_SET0 (offset: 0x0BC)

Signal	Bits	Default	R/W	Description
cfg_aux_ana_set0	18:0	d0	R/W	SADC analog settings [0] aux_debug [1] aux_mode [2] aux_sw_vdd [3] aux_lout [5:4] aux_sin [7:6] aux_map [9:8] aux_mdly [11:10] aux_sel_dly [15:12] aux_br [18:16] aux_br_adc

18.4.37 SADC_ANA_SET1 (offset: 0x0C0)

Signal	Bits	Default	R/W	Description
cfg_aux_pdc	4:0	d0	R/W	SADC analog settings
cfg_aux_ndc	12:8	d0	R/W	SADC analog settings
cfg_aux_pw	21:16	d0	R/W	SADC analog settings
cfg_en_clkaux	25	d0	R/W	Enable SADC clock

				0: disable 1: enable

18.4.38 SADC_WDMA_CTL0 (offset: 0x100)

Signal	Bits	Default	R/W	Description
cfg_sadc_wdma_ctl0	0	d0	W1C	[0] Start xdma for memory access 0: disable 1: enable

18.4.39 SADC_WDMA_CTL1 (offset: 0x104)

Signal	Bits	Default	R/W	Description
cfg_sadc_wdma_ctl1	0	d0	W1C	Software reset xdma 0: disable 1: Reset

18.4.40 SADC_WDMA_SET0 (offset: 0x108)

Signal	Bits	Default	R/W	Description
cfg_sadc_wdma_set0	31:0	d0	R/W	[15:0] Segment size, 2-byte aligned 0: illegal value 1...32768: normal [31:16] Block size, 2-byte aligned

				0: single mode 1..32768: normal

18.4.41 SADC_WDMA_SET1 (offset: 0x10C)

Signal	Bits	Default	R/W	Description
cfg_sadc_wdma_set1	31:0	d0	R/W	Start address point, word (2-byte) aligned The 1-LSB bit will be ignored

18.4.42 SADC_WDMA_R0 (offset: 0x114)

Signal	Bits	Default	R/W	Description
cfg_sadc_wdma_r0	31:0	d0	R	Status of xdma next pointer address

18.4.43 SADC_WDMA_R1 (offset: 0x118)

Signal	Bits	Default	R/W	Description
cfg_sadc_wdma_r1	31:0	d0	R	[0] xdma interrupt status [1] xdma error interrupt status [2] OR (xdma interrupt status, xdma error interrupt status) [9:8] Debug, xdma counter status, used for ping-pong buffer to check buffer status

18.4.44 SADC_INT_CLEAR (offset: 0x120)

Signal	Bits	Default	R/W	Description
cfg_sadc_int_clear	27:0	d0	R/W	[0] Clear SADC_WDMA interrupt 0: disable 1: clear [1] Reserved [2] Clear SADC Done interrupt 0: disable 1: clear [3] Clear SADC Valid interrupt 0: disable 1: clear [4] Clear SADC Mode Done interrupt 0: disable 1: clear [7:5] NoUse [17:8] Clear SADC monitor Low interrupt for CH9:0 0: disable

				1: clear [27:18] Clear SADC monitor High interrupt for CH9:0 0: disable 1: clear

18.4.45 SADC_INT_MASK (offset: 0x124)

Signal	Bits	Default	R/W	Description
cfg_sadc_int_mask	27:0	d0	R	[0] Mask SADC_WDMA interrupt [1] Reserved [2] Mask SADC Done interrupt [3] Mask SADC Valid interrupt [4] Mask SADC Mode Done interrupt [7:5] NoUse [17:8] Mask SADC monitor Low interrupt for CH9:0 [27:18] Mask SADC monitor High interrupt for CH9:0

18.4.46 SADC_INT_STATUS (offset: 0x128)

Signal	Bits	Default	R/W	Description
cfg_sadc_int_status	27:0	d0	R	[0] Status of SADC_WDMA interrupt [1] Reserved [2] Status of SADC Done interrupt

				[3] Status of SADC Valid interrupt [4] Status os SADC Mode Done interrupt [7:5] NoUse [17:8] Status of SADC monitor Low interrupt for CH9:0 [27:18] Status of SADC monitor High interrupt for CH9:0

18.4.47 SADC_R0 (offset: 0x12C)

Signal	Bits	Default	R/W	Description
sadc_o	13:0	d0	R	SADC output value after digital control and oversampling, data format is u0.14.0;
sadc_o_chx	19:16	b0	R	SADC channel index

18.4.48 SADC_R1 (offset: 0x130)

Signal	Bits	Default	R/W	Description
sadc_i_12b	11:0	d0	R	SADC result from analog, data format is u0.12.0
sadc_num_res	31:16	d0	R	Number of SADC result write into WDMA since last sadc_start trigger

18.4.49 SADC_R2 (offset: 0x134)

Signal	Bits	Default	R/W	Description

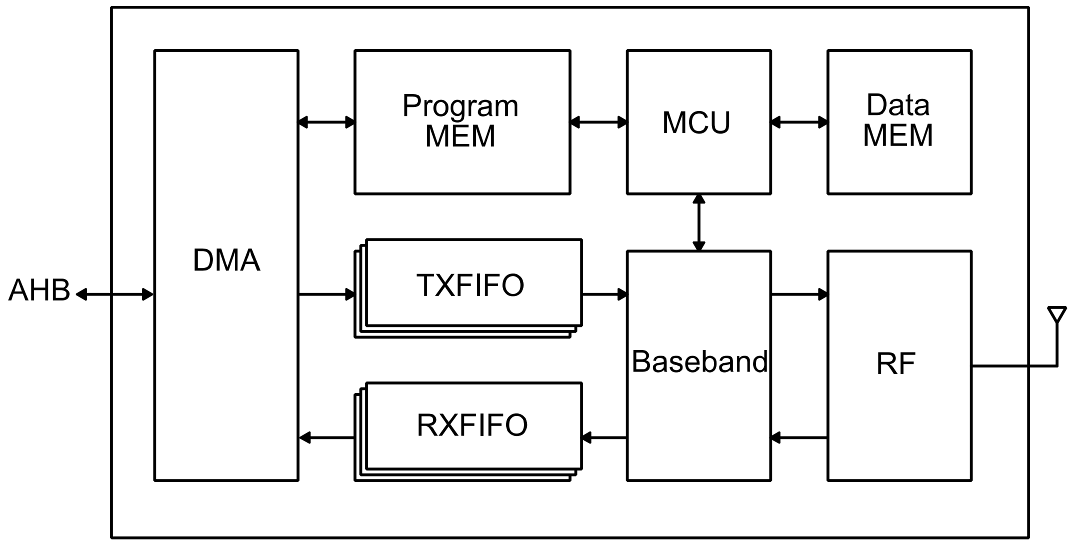
sadc_i_syn	11:0	d0	R	SADC result via AFIFO data format is u0.12.0
sadc_busy	16	d0	R	SADC mode control busy 0: idle 1: busy
sadc_ana_ena	17	d0	R	SADC analog enable

19 Communication Subsystem

The Communication Subsystem transmits and receives data through the wireless channels. To meet real-time requirements of various communication protocols, an embedded MCU processes low-level tasks of the communication protocol. The MCU has its own program and data memories.

In addition, 8 TX FIFOs and 2 RX FIFOs are provided to temporarily store transmitted and received data. The #0 RX FIFO is used to store received data while the #1 RX FIFO is used to store commands from the MCU of the Communication Subsystem.

A DMA with the AHB interface is designed to move data between the host CPU and the Communication Subsystem.



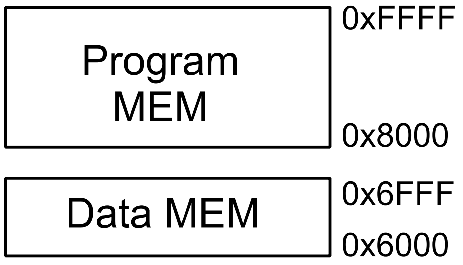
19.1 Register Descriptions

19.1.1 DMA0 (offset: 0x00)

Signal	Bits	Default	R/W	Description
AHB_MEM_ADDR	31:0	b0	R/W	Set the memory address of the host CPU for DMA will write and read

19.1.2 DMA1 (offset: 0x04)

Signal	Bits	Default	R/W	Description
DMA_LENGTH	31:16	b0	R/W	Data length to be transferred. It must be greater than 0. Zero is undefined.
reserved	15:14			
COMM_MEM_ADDR	13:0	b0	R/W	Set the memory address or FIFO IDs of the Communication Subsystem for DMA write & read.

				If dma_type = 2'b00, this indicates RXFIFO ID. Valid values are 0~1. If dma_type = 2'b01, this indicates TXFIFO ID. Valid values are 0~7. If dma_type = 2'b1x, this indicates the MCU memory address. It must be the double-word aligned, i.e. addr[16:2]. The MCU memory map is:  Program MEM 0xFFFF 0x8000 Data MEM 0x6FFF 0x6000

19.1.3 DMA2 (offset: 0x08)

Signal	Bits	Default	R/W	Description
reserved	31:2			
DMA_TYPE	1:0	b0	R/W	Set the DMA transfer type. 2'b00: Move data from the RX FIFO to the memory of the host CPU

				2'b01: Move data from the memory of the host CPU to the TX FIFO 2'b10: Move data from the memory of the communication subsystem to the memory of the host CPU 2'b11: Move data from the memory of the host CPU to the memory of the communication subsystem

19.1.4 HOST_CTRL (offset: 0x0C)

This register defines several controls of the host CPU. All bits of this register are write only. Their values are cleared automatically by the hardware after writing.

Signal	Bits	Default	R/W	Description
reserved	31:26			
DIS_HOST_MODE	25	b0	WO	Disable the host mode of the Communication Subsystem.
EN_HOST_MODE	24	b0	WO	Enable the host mode of the Communication Subsystem.
reserved	23:17			
DMA_EN	16	b0	WO	Enable the DMA.
reserved	15:12			
SYS_RESET	11	b0	WO	Reset the Communication Subsystem
SLEEP_EN	9	b0	WO	Let the Communication Subsystem enter the Sleep Mode
WAKEUP	8	b0	WO	Wake up the Communication Subsystem from Sleep mode.
HOST_CMD	7:0	b0	WO	Host commands

19.1.5 INTR0 (offset: 0x10)

Signal	Bits	Default	R/W	Description
reserved	31:9			
INTR_EN	8:0	b0	R/W	Enable interrupts of the Communication Subsystem. [8] Enable DMA done interrupt [7] Enable RTC wakeup interrupt [6] Enable BMU data transfer error interrupt [5] Enable RX packet valid interrupt [4] Enable MCU software interrupt 4 [3] Enable MCU software interrupt 3 [2] Enable MCU software interrupt 2 [1] Enable MCU software interrupt 1 [0] Enable MCU software interrupt 0

19.1.6 INTR1 (offset: 0x14)

All bits of this register are write only. Their values are cleared automatically by the hardware after writing.

Signal	Bits	Default	R/W	Description
reserved	31:9			

INTR_CLR	8:0	b0	WO	Clear interrupt statuses of the Communication Subsystem. [8] Write 1 to clear DMA done interrupt [7] Write 1 to clear RTC wakeup interrupt [6] Write 1 to clear BMU data transfer error interrupt [5] Write 1 to clear RX packet valid interrupt [4] Write 1 to clear MCU software interrupt 4 [3] Write 1 to clear MCU software interrupt 3 [2] Write 1 to clear MCU software interrupt 2 [1] Write 1 to clear MCU software interrupt 1 [0] Write 1 to clear MCU software interrupt 0

19.1.7 INTR2 (offset: 0x18)

Signal	Bits	Default	R/W	Description
reserved	31:17			
DMA_BUSY	16	b0	RO	DMA busy flag. 1'b0 : DMA is not busy 1'b1 : DMA is busy
reserved	15:9			
INTR_STA	8:0	b0	RO	Interrupt statuses of the Communication Subsystem. [8] DMA done interrupt is pending [7] RTC wakeup interrupt is pending [6] BMU data transfer error interrupt is pending

				[5] RX packet valid interrupt is pending [4] MCU software interrupt 4 is pending [3] MCU software interrupt 3 is pending [2] MCU software interrupt 2 is pending [1] MCU software interrupt 1 is pending [0] MCU software interrupt 0 is pending

19.1.8 INFO0 (offset: 0x1C)

Signal	Bits	Default	R/W	Description
reserved	31:27			
RX_PKT_LEN	26:16	b0	RO	RX packet length
reserved	15:12			
MAC_RX_INFO	11:8	b0	RO	MAC RX information of the currently received packet [11] 1 indicates PHR ERROR [10] 1 indicates CRC ERROR [9] 1 indicates MIC ERROR [8] 1 indicates the invalid packet
reserved	7:5			
HOST_CMD_BUSY	4	b0	RO	1 indicates the HOST COMMAND is busy
reserved	3:2			
RX_FIFO_RRDY	1:0	b0	RO	[1] 1 indicates the #1 RX FIFO is valid for read. [0] 1 indicates the #0 RX FIFO is valid for read.

19.1.9 INFO1 (offset: 0x20)

Signal	Bits	Default	R/W	Description
reserved	31:27			
COMMAND_LEN	26:16	b0	RO	Command length from the MCU of the Communication Subsystem
MCU_STATE	15:8	b0	RO	The MCU state
TX_FIFO_WRDY	7:0	b0	RO	[7] 1 indicates the #7 TX FIFO is valid for write [6] 1 indicates the #6 TX FIFO is valid for write [5] 1 indicates the #5 TX FIFO is valid for write [4] 1 indicates the #4 TX FIFO is valid for write [3] 1 indicates the #3 TX FIFO is valid for write [2] 1 indicates the #2 TX FIFO is valid for write [1] 1 indicates the #1 TX FIFO is valid for write [0] 1 indicates the #0 TX FIFO is valid for write

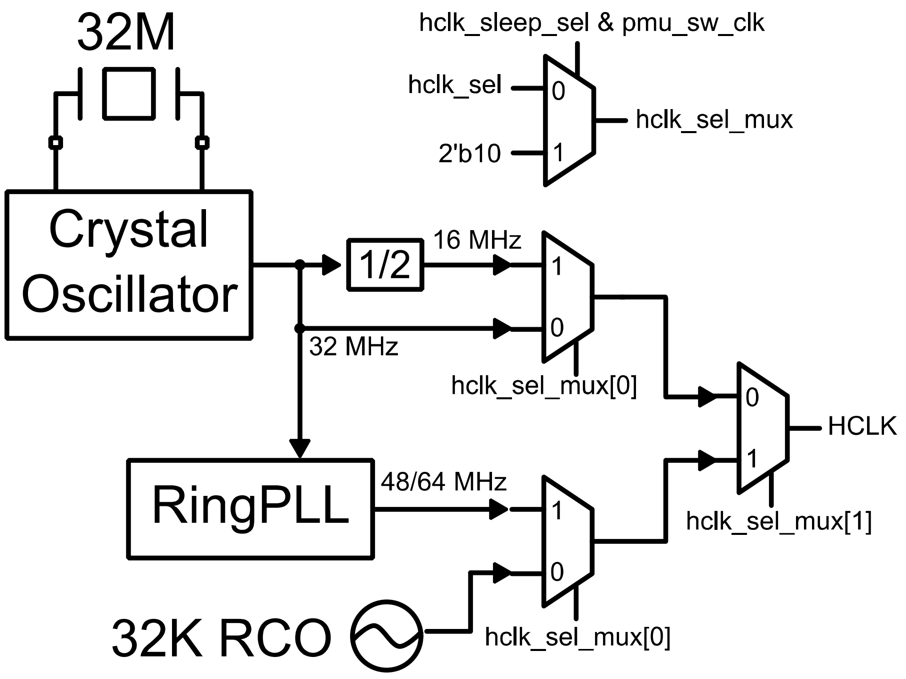
20 System Control

This section describes the system related control such as power states, clocks and GPIO multi-function mappings.

20.1 Clocks

The 32 MHz crystal oscillator provides the basic clock to the system. A baseband PLL can be enabled to provide the higher-frequency clock (48 or 64 MHz) to the ARM MCU. There is also an internal 32 KHz RC oscillator to provide a slow clock source to the system.

The clock scheme is illustrated below.

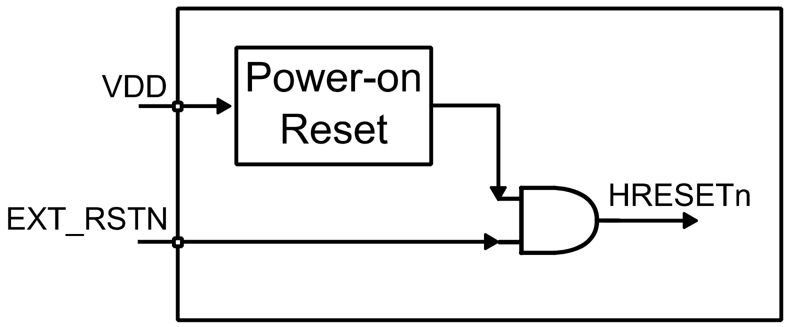


The system clock, HCLK, is controlled by hclk_sel and hclk_sleep_sel. The "hclk_sel" selects the HCLK frequency in the Normal Mode, while the "hclk_sleep_sel" controls the HCLK behavior in the Sleep Mode. The control table of hclk_sel and hclk_sleep_sel is listed below.

hclk_sel	hclk_sleep_sel	HCLK/PCLK	
		Normal Mode	**Sleep Mode**
00	0	32 MHz	No clock
	1		32 KHz
01	0	16 MHz	No clock
	1		32 KHz
10	0	32 KHz	32 KHz
	1		32 KHz
11	0	48/64 MHz	No clock
	1		32 KHz

20.2 Resets

The chip has a power-on reset circuit that will reset the whole system once the power is applied. In addition, the system can also be reset by an external pin.



20.3 Register Descriptions

20.3.1 POWER_STATE_CTRL (offset: 0x00)

Signal	Bits	Default	R/W	Description
reserved	31:2			
SLEEP_EN	0	b0	WO	Write 1 to enter the Sleep power state. It will be automatically cleared after writing.

20.3.2 SYS_CLK_CTRL (offset: 0x04)

Signal	Bits	Default	R/W	Description
reserved	31:10			
CFG_PLL_FREQ	9	b0	R/W	Baseband PLL frequency 1'b0: 48 MHz

				1'b1: 64 MHz
CFG_PLL_EN	8	b0	R/W	Baseband PLL enable. 1'b0 : Disabled 1'b1 : Enabled.
SLOW_CLK_SEL	7:6	b0	R/W	Slow clock selection 2'b00: From internal RCO 2'b01: From GPIO 2'b10: From 32M / 1000 2'b11: From XO 32K
Reserved	5	b0	R/W	
HCLK_SLEEP_SEL	4	b1	R/W	HCLK select in the Sleep power state. 1'b0 : HCLK is disabled in Sleep 1'b1 : HCLK is switched to slow clock in Sleep
Reserved	3:2	b0	R/W	
HCLK_SEL	1:0	b0	R/W	HCLK select in the Normal power state 2'b00 : 32 MHz 2'b01 : 16 MHz 2'b10 : Slow clock 2'b11 : PLL clock

20.3.3 GPIO_MAP0 (offset: 0x10)

Signal	Bits	Default	R/W	Description
reserved	31			
GPIO07_SEL	30:28	b0	R/W	0: GPIO7 1: SPI0_CSN0 2: N/A 3: X51_TDI 4: N/A 5: EXT32K 6: UART2_RX 7: DBG_OUT[7]
reserved	27			
GPIO06_SEL	26:24	b0	R/W	0: GPIO6 1: SPI0_SCLK 2: N/A 3: X51_TCK 4: N/A 5: EXT32K 6: UART2_TX 7: DBG_OUT[6]
reserved	23			
GPIO05_SEL	22:20	b0	R/W	0: GPIO5

				1: SPI0_SDATA3 2: N/A 3: X51_TMS 4: I2S0_MCK 5: EXT32K 6: UART1_RX 7: DBG_OUT[5]
reserved	19			
GPIO04_SEL	18:16	b0	R/W	0: GPIO4 1: SPI0_SDATA2 2: N/A 3: N/A 4: I2S0_MCK 5: EXT32K 6: UART1_TX 7: DBG_OUT[4]
reserved	15			
GPIO03_SEL	14:12	b0	R/W	0: GPIO3 1: N/A 2: SPI0_CSN1 3: SPI0_CSN2 4: I2S0_SDI 5: EXT32K

				6: SPI0_CSN3 7: DBG_OUT[3]
reserved	11			
GPIO02_SEL	10:8	b0	R/W	0: GPIO2 1: N/A 2: SPI0_CSN1 3: SPI0_CSN2 4: I2S0_SDO 5: EXT32K 6: SPI0_CSN3 7: DBG_OUT[2]
reserved	7			
GPIO01_SEL	6:4	b0	R/W	0: GPIO1 1: N/A 2: SPI0_CSN1 3: SPI0_CSN2 4: I2S0_WCK 5: EXT32K 6: SPI0_CSN3 7: DBG_OUT[1]
reserved	3			
GPIO00_SEL	2:0	b0	R/W	0: GPIO0 1: N/A

				2: SPI0_CSN1 3: SPI0_CSN2 4: I2S0_BCK 5: EXT32K 6: SPI0_CSN3 7: DBG_OUT[0]

20.3.4 GPIO_MAP1 (offset: 0x14)

Signal	Bits	Default	R/W	Description
reserved	31			
GPIO15_SEL	30:28	b0	R/W	0: GPIO15 1: SPI0_SDATA3 2: N/A 3: N/A 4: PWM3 5: I2S1_MCK 6: UART1_CTSN 7: DBG_OUT[11]
reserved	27			
GPIO14_SEL	26:24	b0	R/W	0: GPIO14 1: SPI0_SDATA2 2: N/A

				3: N/A 4: PWM2 5: SPI1_SDATA2 6: UART1_RTSN 7: DBG_OUT[10]
reserved	23			
GPIO13_SEL	22:20	b0	R/W	0: GPIO13 1: PWM0 2: PWM1 3: PWM2 4: I2S0_SDI 5: PWM3 6: UART2_RX 7: PWM4
reserved	19			
GPIO12_SEL	18:16	b0	R/W	0: GPIO12 1: PWM0 2: PWM1 3: PWM2 4: I2S0_SDO 5: PWM3 6: UART2_TX 7: PWM4

reserved	15			
GPIO11_SEL	14:12	b0	R/W	0: GPIO11 1: PWM0 2: PWM1 3: PWM2 4: I2S0_WCK 5: PWM3 6: UART1_RX 7: PWM4
reserved	11			
GPIO10_SEL	10:8	b0	R/W	0: GPIO10 1: PWM0 2: PWM1 3: PWM2 4: I2S0_BCK 5: PWM3 6: UART1_TX 7: PWM4
reserved	7			
GPIO09_SEL	6:4	b0	R/W	0: GPIO9 1: SPI0_SDATA1 2: N/A 3: X51_RTCK

				4: PWM1 5: EXT32K 6: N/A 7: DBG_OUT[9]
reserved	3			
GPIO08_SEL	2:0	b0	R/W	0: GPIO8 1: SPI0_SDATA0 2: N/A 3: X51_TDO 4: PWM0 5: EXT32K 6: N/A 7: DBG_OUT[8]

20.3.5 GPIO_MAP2 (offset: 0x18)

Signal	Bits	Default	R/W	Description
reserved	31			
GPIO23_SEL	30:28	b0	R/W	0: GPIO23 1: PWM0 2: PWM1 3: PWM2 4: I2C_SDA

				5: PWM3 6: N/A 7: PWM4
reserved	27			
GPIO22_SEL	26:24	b0	R/W	0: GPIO22 1: PWM0 2: PWM1 3: PWM2 4: I2C_SCL 5: PWM3 6: N/A 7: PWM4
reserved	23			
GPIO21_SEL	22:20	b0	R/W	0: GPIO21 1: PWM0 2: PWM1 3: PWM2 4: I2C_SDA 5: PWM3 6: UART1_CTSN 7: PWM4
reserved	19			
GPIO20_SEL	18:16	b0	R/W	0: GPIO20

				1: PWM0 2: PWM1 3: PWM2 4: I2C_SCL 5: PWM3 6: UART1_RTSN 7: PWM4
reserved	15			
GPIO19_SEL	14:12	b0	R/W	0: GPIO19 1: PWM0 2: PWM1 3: PWM2 4: I2C_SDA 5: PWM3 6: UART1_CTSN 7: PWM4
reserved	11			
GPIO18_SEL	10:8	b0	R/W	0: GPIO18 1: PWM0 2: PWM1 3: PWM2 4: I2C_SCL 5: PWM3

				6: UART1_RTSN 7: PWM4
reserved	7			
GPIO17_SEL	6:4	b0	R/W	0: GPIO17 1: N/A 2: N/A 3: N/A 4: N/A 5: N/A 6: UART0_TX 7: N/A
reserved	3			
GPIO16_SEL	2:0	b0	R/W	0: GPIO16 1: N/A 2: N/A 3: N/A 4: N/A 5: N/A 6: UART0_RX 7: N/A

20.3.6 GPIO_MAP3 (offset: 0x1C)

Signal	Bits	Default	R/W	Description
reserved	31			
GPIO31_SEL	30:28	b0	R/W	0: GPIO31 1: SPI0_SDATA1 2: N/A 3: N/A 4: N/A 5: SPI1_SDATA1 6: UART2_RX 7: DBG_OUT[15]
reserved	27			
GPIO30_SEL	26:24	b0	R/W	0: GPIO30 1: SPI0_SDATA0 2: N/A 3: N/A 4: N/A 5: SPI1_SDATA0 6: UART2_TX 7: DBG_OUT[14]
reserved	23			
GPIO29_SEL	22:20	b0	R/W	0: GPIO29 1: SPI0_CSN0 2: N/A

				3: N/A 4: N/A 5: SPI1_CSN 6: UART1_RX 7: DBG_OUT[13]
reserved	19			
GPIO28_SEL	18:16	b0	R/W	0: GPIO28 1: SPI0_SCLK 2: N/A 3: N/A 4: N/A 5: SPI1_SCLK 6: UART1_TX 7: DBG_OUT[12]
reserved	15			
GPIO27_SEL	14:12	b0	R/W	0: GPIO27 1: SPI0_SDATA1 2: SPI0_CSN1 3: SPI0_CSN2 4: N/A 5: SPI1_SDATA1 6: SPI0_CSN3 7: N/A

reserved	11			
GPIO26_SEL	10:8	b0	R/W	0: GPIO26 1: SPI0_SDATA0 2: SPI0_CSN1 3: SPI0_CSN2 4: N/A 5: SPI1_SDATA0 6: SPI0_CSN3 7: N/A
reserved	7			
GPIO25_SEL	6:4	b0	R/W	0: GPIO25 1: SPI0_CSN0 2: SPI0_CSN1 3: SPI0_CSN2 4: N/A 5: SPI1_CSN 6: SPI0_CSN3 7: N/A
reserved	3			
GPIO24_SEL	2:0	b0	R/W	0: GPIO24 1: SPI0_SCLK 2: SPI0_CSN1 3: SPI0_CSN2

				4: N/A 5: SPI1_SCLK 6: SPI0_CSN3 7: N/A

20.3.7 GPIO_PULL_CTRL0 (offset: 0x20)

Signal	Bits	Default	R/W	Description
reserved	31			
GPIO_PULL_SEL7	30:28	b110	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	27			
GPIO_PULL_SEL6	26:24	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull

				5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	23			
GPIO_PULL_SEL5	22:20	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	19			
GPIO_PULL_SEL4	18:16	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	15			
GPIO_PULL_SEL3	14:12	b0	R/W	0: no pull

				1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	11			
GPIO_PULL_SEL2	10:8	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	7			
GPIO_PULL_SEL1	6:4	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up

				6: 100K pull up 7: 1M pull up
reserved	3			
GPIO_PULL_SEL0	2:0	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up

20.3.8 GPIO_PULL_CTRL1 (offset: 0x24)

Signal	Bits	Default	R/W	Description
reserved	31			
GPIO_PULL_SEL15	30:28	b110	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down

				4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	27			
GPIO_PULL_SEL14	26:24	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	23			
GPIO_PULL_SEL13	22:20	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	19			

GPIO_PULL_SEL12	18:16	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	15			
GPIO_PULL_SEL11	14:12	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	11			
GPIO_PULL_SEL10	10:8	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull

				5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	7			
GPIO_PULL_SEL9	6:4	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	3			
GPIO_PULL_SEL8	2:0	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up

20.3.9 GPIO_PULL_CTRL2 (offset: 0x28)

Signal	Bits	Default	R/W	Description
reserved	31			
GPIO_PULL_SEL23	30:28	b110	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	27			
GPIO_PULL_SEL22	26:24	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	23			

GPIO_PULL_SEL21	22:20	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	19			
GPIO_PULL_SEL20	18:16	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	15			
GPIO_PULL_SEL19	14:12	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull

				5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	11			
GPIO_PULL_SEL18	10:8	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	7			
GPIO_PULL_SEL17	6:4	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	3			
GPIO_PULL_SEL16	2:0	b0	R/W	0: no pull

				1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up

20.3.10 GPIO_PULL_CTRL3 (offset: 0x2C)

Signal	Bits	Default	R/W	Description
reserved	31			
GPIO_PULL_SEL31	30:28	b110	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	27			
GPIO_PULL_SEL30	26:24	b0	R/W	0: no pull 1: 10K pull down

				2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	23			
GPIO_PULL_SEL29	22:20	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	19			
GPIO_PULL_SEL28	18:16	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up

				7: 1M pull up
reserved	15			
GPIO_PULL_SEL27	14:12	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	11			
GPIO_PULL_SEL26	10:8	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	7			
GPIO_PULL_SEL25	6:4	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down

				3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up
reserved	3			
GPIO_PULL_SEL24	2:0	b0	R/W	0: no pull 1: 10K pull down 2: 100K pull down 3: 1M pull down 4: no pull 5: 10K pull up 6: 100K pull up 7: 1M pull up

20.3.11 GPIO_DRV_CTRL0 (offset: 0x30)

Signal	Bits	Default	R/W	Description
GPIO_DRV_SEL15	31:30	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL14	29:28	b11	R/W	0: 4 mA

				1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL13	27:26	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL12	25:24	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL11	23:22	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL10	21:20	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL9	19:18	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA

GPIO_DRV_SEL8	17:16	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL7	15:14	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL6	13:12	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL5	11:10	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL4	9:8	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL3	7:6	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA

				3: 20 mA
GPIO_DRV_SEL2	5:4	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL1	3:2	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL0	1:0	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA

20.3.12 GPIO_DRV_CTRL1 (offset: 0x34)

Signal	Bits	Default	R/W	Description
GPIO_DRV_SEL31	31:30	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL30	29:28	b11	R/W	0: 4 mA 1: 10 mA

				2: 14 mA 3: 20 mA
GPIO_DRV_SEL29	27:26	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL28	25:24	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL27	23:22	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL26	21:20	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL25	19:18	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL24	17:16	b11	R/W	0: 4 mA

				1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL23	15:14	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL22	13:12	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL21	11:10	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL20	9:8	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL19	7:6	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA

GPIO_DRV_SEL18	5:4	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL17	3:2	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA
GPIO_DRV_SEL16	1:0	b11	R/W	0: 4 mA 1: 10 mA 2: 14 mA 3: 20 mA

20.3.13 GPIO_DO_CTRL (offset: 0x38)

Signal	Bits	Default	R/W	Description
GPIO_EN_OD	31:0	b0	R/W	GPIOx open-drain enable 1'b0 : Disable open-drain for GPIOx 1'b1 : Enable open-drain for GPIOx

20.3.14 GPIO_AIO_CTRL (offset: 0x3C)

Signal	Bits	Default	R/W	Description

reserved	31:8	b0		

GPIO_EN_AIO	7:0	b0	R/W	AIOx enable.
				
				The GPIO with AIO capability are:
				
				GPIO24 = AIO0
				
				GPIO25 = AIO1
				
				GPIO26 = AIO2
				
				GPIO27 = AIO3
				
				GPIO28 = AIO4
				
				GPIO29 = AIO5
				
				GPIO30 = AIO6
				
				GPIO31 = AIO7
				
				1'b0 : disable
				
				1'b1 : enable

20.3.15 Scratchpad_0 (offsets: 0x60)

Signal	Bits	Default	R/W	Description
Scratchpad0	31:0	x0	R/W	Scratchpad #0 register value

20.3.16 Scratchpad_1 (offsets: 0x64)

Signal	Bits	Default	R/W	Description
Scratchpad1	31:0	x0	R/W	Scratchpad #1 register value

20.3.17 Scratchpad_2 (offsets: 0x68)

Signal	Bits	Default	R/W	Description
Scratchpad2	31:0	x0	R/W	Scratchpad #2 register value

20.3.18 Scratchpad_3 (offsets: 0x6C)

Signal	Bits	Default	R/W	Description
Scratchpad3	31:0	x0	R/W	Scratchpad #3 register value

20.3.19 Scratchpad_4 (offsets: 0x70)

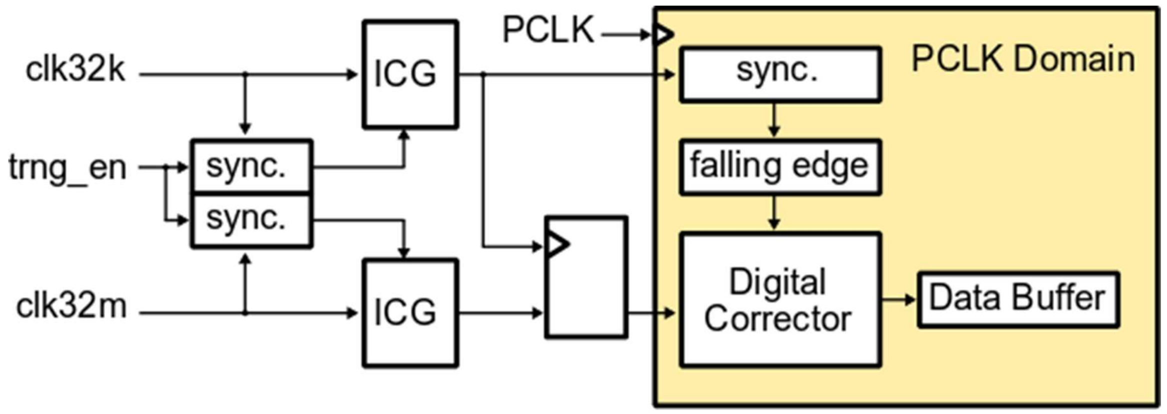
Signal	Bits	Default	R/W	Description
Scratchpad4	31:0	x0	R/W	Scratchpad #4 register value

20.3.20 Scratchpad_5 (offsets: 0x74)

Signal	Bits	Default	R/W	Description
Scratchpad5	31:0	x0	R/W	Scratchpad #5 register value

21 Random Number Generator

The T32CM11C10GQ40-AR incorporates a Random Number Generator. The random source is from the jitter of the internal RCO32K. The output of the RCO32K is then used to sample a high-speed clock to produce a random sequence. The sequence is then passed to a digital corrector to reduce bias. The block diagram of the Random Number Generator is drawn below:



Two types of digital correctors can be selected: Von Neumann and XOR.

21.1 Register Descriptions

The registers of the True Random Number Generator (TRNG) are located within SYS_CTRL with base address mapped to 0x40800000.

21.1.1 TRNG0 (offsets: 0x40)

Signal	Bits	Default	R/W	Description
reserved	31:2	b0		

TRNG_INTR_CLR	1	b0	W	Write 1 to clear TRNG interrupt status.
TRNG_EN	0	b0	W	Write 1 to trigger TRNG

21.1.2 TRNG1 (offsets: 0x44)

Signal	Bits	Default	R/W	Description
reserved	31:2	b0		
TRNG_INTR_EN	1	b0	R/W	TRNG interrupt enable 1'b0: disabled 1'b1: enabled
TRNG_SEL	0	b0	R/W	TRNG corrector method 1'b0: von Neumann corrector 1'b1: XOR corrector

21.1.3 TRNG2 (offsets: 0x48)

Signal	Bits	Default	R/W	Description
reserved	31:2	b0		
TRNG_INTR_STA	1	b0	R	TRNG interrupt status 1'b0: no interrupt pending 1'b1: interrupt pending
TRNG_BUSY	0	b0	R	TRNG busy 1'b0: not busy 1'b1: busy

21.1.4 TRNG3 (offsets: 0x4C)

Signal	Bits	Default	R/W	Description
TRNG_OUT	31:0	b0	R	TRNG output.

22 Crypto accelerator

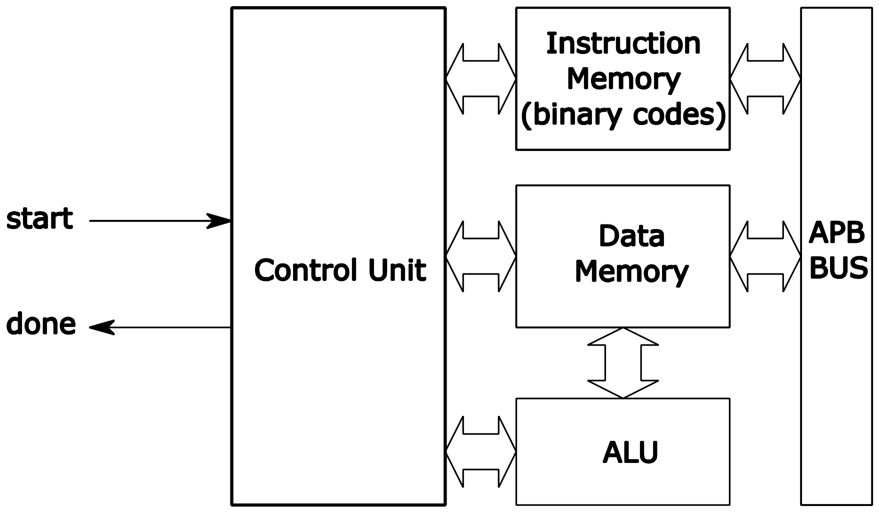
To execute security applications faster, a crypto accelerator is supplied for various algorithms. This module is essentially a processor which accepts binary codes to execute.

22.1 Features

- AES
  - 128-bit, 192-bit and 256-bit key length encryption/decryption
  - ECB, CBC, CTR, CMAC, CCM cipher modes
- SHA256
  - Can be extended to HMAC, HMAC DRBG or other
- ECC
  - 192-bit and 256-bit key length
  - Both prime field GF(p) and binary field GF(2m)
  - NIST P-192 curve (also knowns as SECP192R1), P-256 curve (also knowns as SECP256R1 or Prime256v1)

- NIST B-163 curve (also knowns as SECT163R2)
- Curve 25519
- ECDH and ECDSA protocol

22.2 Block Diagram



22.3 Function description

There are internal memories in the crypto accelerator: instruction memory and data memory. The instruction memory needs binary code pre-loaded before starting. Due to the requirements of various binary codes according to different crypto algorithms, refer to the SDK document for the usage of this module.

23 Revision History & Contact Information

Revision History

Revision	Date	Description
2	2025-10-28	Removed Deep sleep and sleep 2
1	2025-04-21	Beta Release

Contact Information

Contact Information	
Contact	www.TridentIoT.com
Sales	sales@tridentiot.com
Technical Support	support@tridentiot.com

© 2025 by Trident iOt, LLC All Rights Reserved.