MAX32665/MAX32666 USER GUIDE

UG6971; Rev 3; 2/2022

Abstract**:** This user guide provides application developers information on how to use the memory and peripherals of the **MAX32665/MAX32666** microcontroller. Detailed information for all registers and fields in the device are covered. Guidance is given for managing all the peripherals, clocks, power and startup for the device family.

MAX32665/MAX32666 User Guide

Table of Contents

List of Figures

List of Tables

1. Overview

The MAX32665/MAX32666 are Arm® Cortex®-M4 with FPU-based microcontrollers with 1MB flash and up to 560KB SRAM that can be configured as 448KB SRAM with error correction coding (ECC). They are ideal for wearable medical fitness applications. Optionally available is a second Arm Cortex-M4 with FPU for audio signal processing in a wireless headset/earbud application. The architecture includes a Bluetooth® 5 Low Energy radio.

The MAX32665—MAX32666 feature a second Arm Cortex-M4 with FPU with supporting ROM and cache. This feature supports extended data processing capabilities such as audio processing for wireless Bluetooth applications. Refer to the Ordering Information in the device data sheet for device feature detail.

The devices feature five powerful and flexible power modes. Built-in dynamic clock gating and firmware-controlled power gating allows the user to optimize power for the specific application. A built-in single inductor multiple output (SIMO) switch mode power supply allows the device to be optionally self-powered by a primary lithium cell.

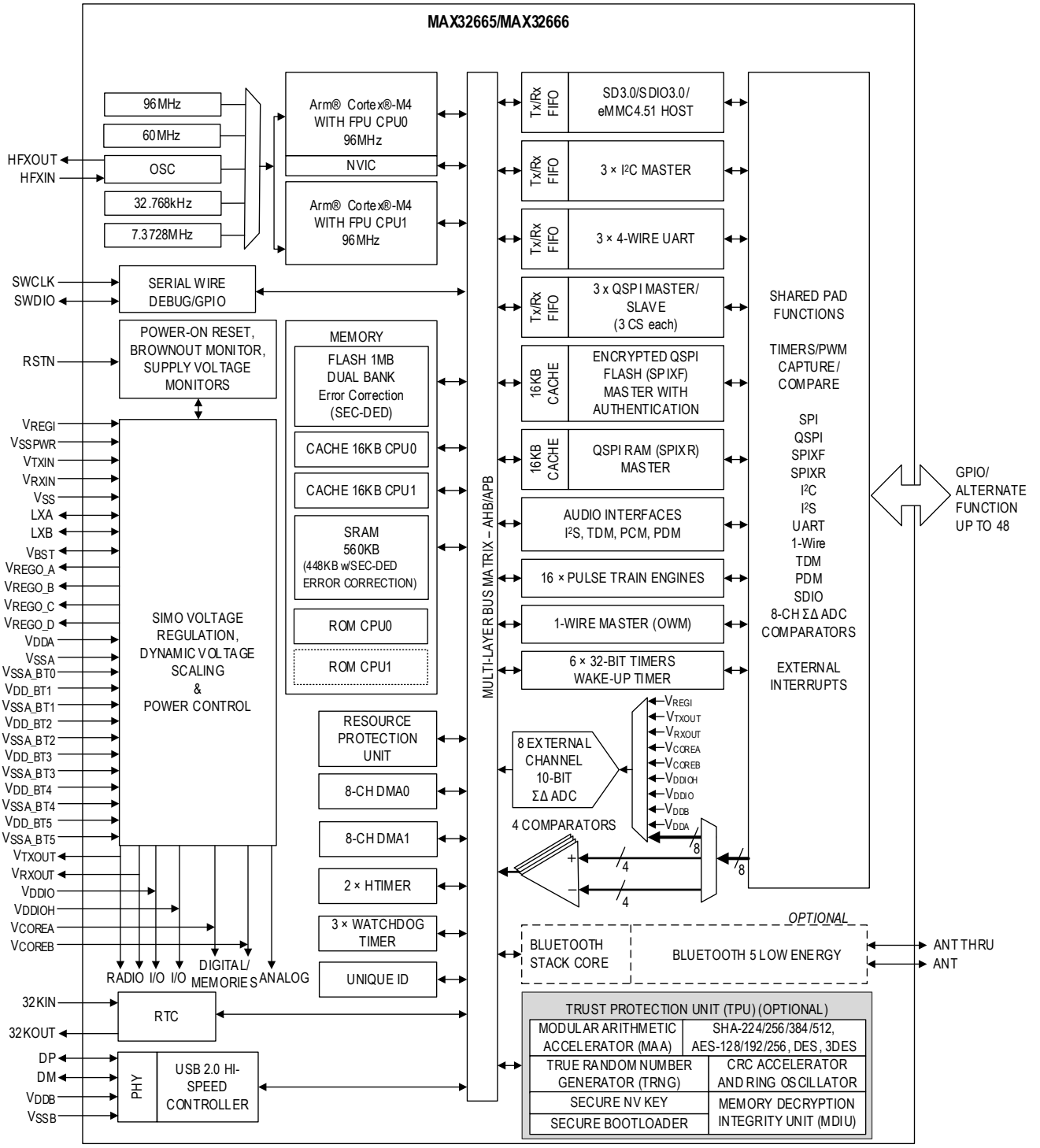
The flash memory is split into two banks of 512KB to provide flexibility when programming over-the-air. The devices have an ECC with single error correction double error detection (SEC-DED) for flash, and SRAM providing extremely reliable code execution. Dedicated hardware runs the Bluetooth 5 Low Energy stack freeing the CPUs for data processing tasks. Multiple SPI, UART, and I2C serial interfaces, 1-Wire® Master, and USB 2.0 High-Speed Device interface allow for interconnection to a wide variety of external sensors. An audio subsystem supporting PDM, PCM, I^2S, and TDM. An 8-input, 10-bit ADC is available to monitor analog input from external sensors and meters.

The MAX32665/MAX32666 incorporate a trust protection unit (TPU) with encryption and advanced security features. These features include a modular arithmetic accelerator (MAA) for fast ECDSA, a hardware AES engine, a hardware TRNG entropy generator, a SHA-2 accelerator and a secure bootloader.

The high-level block diagram for the MAX32665/MAX32666 is shown in *Figure 1-1*.

1.1 Block Diagram

Figure 1-1: MAX32665/MAX32666 Block Diagram



2. Resource Protection Unit (RPU)

The resource protection unit (RPU) is a dedicated module that augments the Arm Cortex-M4 core memory protection unit (MPU). The RPU allows the system to provide resource protection for AHB bus masters accessing system memory or AHB/APB bus slaves.

The MPU enforces privilege and access controls that restrict a CPU from accessing user-defined segments of system memory. It does not, however, provide access controls for other AHB bus masters accessing system memory or AHB/APB bus slaves.

The RPU is separate to and maintains software compatibility with Arm's MPU privilege levels on the CPU side and uses the standard AMBA bus.

The RPU features include the following:

- Software compatibility with Arm Memory Protection Unit (MPU)
- Provides access control for DMA and other AHB masters
- Access controls for each bus slave independently configurable
- Dedicated access-protection register for each bus slave

2.1 Instances

The IC has three buses:

- AHB
- APB Bus 0
- APB Bus 1

The peripherals in *Table 2-1* can be mapped to either APB Bus 0 or the fixed frequency APB Bus 1. Each peripheral has a different register set depending on the which bus the peripheral is connected to.

Table 2-1: Dual Mapped APB Peripherals

Peripheral	APB Bus 0	ABP Bus 1
I2C0	*I2C0_BUS0_*	*I2C0_BUS1_*
I2C1	*I2C1_BUS0_*	*I2C1_BUS1_*
I2C2	*I2C2_BUS0_*	*I2C2_BUS1_*

The AHB bus masters are listed in *Table 2-2*. Each bus slave has a dedicated RPU access control register. Each bit position in a slave's RPU register allows or denies access by a specific AHB bus master as shown in in *Table 2-2*. Register bits corresponding to unimplemented bus masters should not be changed from their reset default value.

Because of the structure of the APB bus, there is only one access control bit. This means that the read and write access permissions for a particular master must always be the same. Access permissions for read and write can be configured separately for AHB slaves.

Table 2-2: MAX32665/MAX32666 Master Permission Bits

AHB Master	Bit Position in SLAVEAPB Register			Bit Position in SLAVEAHB Register	Description
	Write	Read	Write	Read	
DMAC0	access[0]		access[1]	access[0]	Standard DMA Controller 0
DMAC1	access[1]		access[3]	access[2]	Standard DMA Controller 1
USBHS	access[2]		access[5]	access[4]	USB Endpoint Buffer Manager
SYS0	access[3]		access[7]	access[6]	System Control, CPU0

AHB Master	Bit Position in SLAVEAPB Register			Bit Position in SLAVEAHB Register	Description
	Write	Read	Write	Read	
SYS1	access[4]		access[9]	access[8]	System Control, CPU1
SDMAD	access[6]		access[11]	access[10]	Smart DMA D
SDMAI	access[7]		access[13]	access[12]	Smart DMA I
CRYPTO	access[8]		access[15]	access[14]	Dedicated Cryptographic DMA
SDIO	access[9]		access[17]	access[16]	SDIO Memory

Table 2-3 lists the AHB slaves addressable by AHB bus masters. Each AHB slave has a dedicated RPU access control register similar to the APB RPU access control registers, but each AHB slave has two control bits.

Table 2-3: MAX32665/MAX32666 AHB Slaves

AHB Slave	Address	Description
USBHS	USB FIFO	USB Endpoint Data
SDIO	SDIO/SDHC Target Memory	*SDIO*
SPIXFM_FIFO	SPI FIFO Memory	SPI Bus Master FIFO
SPI0	SPI0 FIFO	SPI0 Data Buffer
SYSRAM0	Configurable by Arm MPU	System RAM, Memory Instance 0
SYSRAM1	Configurable by Arm MPU	System RAM, Memory Instance 1
SYSRAM2	Configurable by Arm MPU	System RAM, Memory Instance 2
SYSRAM3	Configurable by Arm MPU	System RAM, Memory Instance 3
SYSRAM4	Configurable by Arm MPU	System RAM, Memory Instance 4
SYSRAM5	Configurable by Arm MPU	System RAM, Memory Instance 5
SYSRAM6-11	Configurable by Arm MPU	System RAM, Memory Instances 6-11

The AHB bus prohibits some AHB master and slave interactions as shown in *Table 2-4.* The AHB slave ignores the state of prohibited combinations.

Table 2-4: MAX32665/MAX32666 AHB Master/Slave Interconnect Matrix

AHB Slave	AHB Master								
	DMAC0	DMAC1	USB	SYS0	SYS1	SDMAD	SDMAI	CRYPTO	SDIO/SDHC MASTER
SYSRAM0	Y	Y	Y	Y	Y	Y	Y	Y	Y
SYSRAM1	Y	Y	Y	Y	Y	Y	Y	Y	Y
SYSRAM2	Y	Y	Y	Y	Y	Y	Y	Y	Y
SYSRAM3	Y	Y	Y	Y	Y	Y	Y	Y	Y

SYSRAM4	Y	Y	Y	Y	Y	Y	Y	Y	Y
SYSRAM5	Y	Y	Y	Y	Y	Y	Y	Y	Y
SYSRAM6-11	Y	Y	Y	Y	Y	Y	Y	Y	Y
SPIXFM_FIFO	Y	Y		Y	Y	Y			
USBHS	Y	Y		Y	Y				
SRCC	Y	Y		Y	Y	Y	Y	Y	
SDIO	Y	Y		Y	Y	Y	Y	Y	
SPI0	Y	Y		Y	Y	Y			

2.2 Usage

2.2.1 Reset State

During a power-on-reset event, RPU registers are reset to their reset value. If RPU protection is desired, the registers must be reprogrammed during the boot sequence.

The RPU registers can also be reset by writing 1 to the Global Control register bit *GCR_RSTR1.rpu*.

2.2.2 MPU Implementation

Accesses to system memory still involve interaction with both the RPU first and then MPU. The RPU grants access to MPU functionality including:

- Protection regions
- Overlapping protection regions, with ascending region priority
- Access permissions
- Exporting memory attributes to the system

The MPU can:

- Enforce privilege rules
- Separate processes
- Enforce access rules

2.2.3 MPU Protection Fault

The MPU can generate three types of faults:

- Background fault
- Permission fault
- Alignment fault

When a fault occurs, the memory access or instruction fetch is synchronously aborted, and a prefetch abort or data abort exception is taken as appropriate. No memory accesses are performed on the AXI bus master interface or peripheral.

2.2.4 RPU Protection Fault

An RPU protection fault occurs when an AHB master attempts to access a slave that does not have the corresponding bits in its RPU register cleared. This will be expressed as an AHB bus fault.

2.2.5 RPU Fault Handler

Sample code demonstrating the implementation of an RPU Fault Handler is provided in the SDK.

2.3 Registers

See *Table 3-3* for the RPU Peripheral Base Address.

Table 2-5: RPU APB Register Offsets, Names, Access, and Descriptions

Offset	Register	Access	Description
[0x0000]	*GCR*	R/W	GCR RPU Register
[0x0004]	*SIR*	R/W	SIR RPU Register
[0x0008]	*FCR*	R/W	FCR RPU Register
[0x0010]	*TPU*	R/W	TPU RPU Register
[0x0020]	*RPU*	R/W	RPU Register
[0x0030]	*WDT0*	R/W	WDT0 RPU Register
[0x0034]	*WDT1*	R/W	WDT1 RPU Register
[0x0038]	*WDT2*	R/W	WDT2 RPU Register
[0x0040]	*SMON*	R/W	SMON RPU Register
[0x0044]	*SIMO*	R/W	SIMO RPU Register
[0x0048]	*DVS*	R/W	DVS RPU Register
[0x0050]	*AES*	R/W	AES RPU Register
[0x0060]	*RTC*	R/W	RTC RPU Register
[0x0064]	*WUT*	R/W	WUT RPU Register
[0x0068]	*PWRSEQ*	R/W	PWRSEQ RPU Register
[0x006C]	*MCR*	R/W	MCR RPU Register
[0x0080]	*GPIO0*	R/W	GPIO0 RPU Register
[0x0090]	*GPIO1*	R/W	GPIO1 RPU Register
[0x0100]	*TMR0*	R/W	TMR0 RPU Register
[0x0110]	*TMR1*	R/W	TMR1 RPU Register
[0x0120]	*TMR2*	R/W	TMR2 RPU Register
[0x0130]	*TMR3*	R/W	TMR3 RPU Register
[0x0140]	*TMR4*	R/W	TMR4 RPU Register
[0x0150]	*TMR5*	R/W	TMR5 RPU Register
[0x01B0]	*HTIMER0*	R/W	HTIMER0 RPU Register
[0x01C0]	*HTIMER1*	R/W	HTTIMER1 RPU Register

Offset	Register	Access	Description
[0x01D0]	*I2C0_BUS0*	R/W	I2C0_BUS0 RPU Register
[0x01E0]	*I2C1_BUS0*	R/W	I2C1_BUS0 RPU Register
[0x01F0]	*I2C2_BUS0*	R/W	I2C2_BUS0 RPU Register
[0x0260]	*SPIXM*	R/W	SPIXM RPU Register
[0x0270]	*SPIXFC*	R/W	SPIXFC RPU Register
[0x0280]	*DMA0*	R/W	DMA0 RPU Register
[0x0290]	*FLC0*	R/W	FLC0 RPU Register
[0x0294]	*FLC1*	R/W	FLC1 RPU Register
[0x02A0]	*ICC0*	R/W	ICC0 RPU Register
[0x02A4]	*ICC1*	R/W	ICC1 RPU Register
[0x02F0]	*SFCC*	R/W	SFCC RPU Register
[0x0330]	*SRCC*	R/W	SRCC RPU Register
[0x0340]	*ADC*	R/W	ADC RPU Register
[0x0350]	*DMA1*	R/W	DMA1 RPU Register
[0x0360]	*SDMA*	R/W	SDMA RPU Register
[0x0370]	*SDHCCTRL*	R/W	SD Host Controller (APB)
[0x03A0]	*SPIXR*	R/W	SPIXR RPU Register
[0x03C0]	*PTG_BUS0*	R/W	PTG_BUS0 RPU Register
[0x03D0]	*OWM*	R/W	OWM RPU Register
[0x03E0]	*SEMA*	R/W	SEMA RPU Register
[0x0420]	*UART0*	R/W	UART0 RPU Register
[0x0430]	*UART1*	R/W	UART1 RPU Register
[0x0440]	*UART2*	R/W	UART2 RPU Register
[0x0460]	*SPI1*	R/W	SPI1 RPU Register
[0x0470]	*SPI2*	R/W	SPI2 RPU Register
[0x04C0]	*AUDIO*	R/W	AUDIO RPU Register
[0x04D0]	*TRNG*	R/W	TRNG RPU Register
[0x0500]	*BTLE*	R/W	BTLE RPU Register
[0x11D0]	*I2C0_BUS1*	R/W	I2C0_BUS1 RPU Register

Offset	Register	Access	Description
[0x11E0]	*I2C1_BUS1*	R/W	I2C1_BUS1 RPU Register
[0x11F0]	*I2C2_BUS1*	R/W	I2C2_BUS1 RPU Register
[0x13C0]	*PTG_BUS1*	R/W	PTG_BUS1 RPU Register

Table 2-6: RPU AHB Slave Register Addresses, Names, Access, and Descriptions

APB Address	Register	Access	Description
[0x0B10]	*USBHS*	R/W	USBHS RPU Register
[0x0B60]	*SDIO*	R/W	SDIO/SDHC Target RPU Register
[0x0BC0]	*SPIXFM_FIFO*	R/W	SPIXFM_FIFO RPU Register
[0x0BE0]	*SPI0*	R/W	SPI0 RPU Register
[0x0F00]	*SYSRAM0*	R/W	SYSRAM0 RPU Register
[0x0F10]	*SYSRAM1*	R/W	SYSRAM1 RPU Register
[0x0F20]	*SYSRAM2*	R/W	SYSRAM2 RPU Register
[0x0F30]	*SYSRAM3*	R/W	SYSRAM3 RPU Register
[0x0F40]	*SYSRAM4*	R/W	SYSRAM4 RPU Register
[0x0F50]	*SYSRAM5*	R/W	SYSRAM5 RPU Register
[0x0F60]	*SYSRAM6-11*	R/W	SYSRAM6-11 RPU Register
[0x0F20]	*SYSRAM0*	R/W	SYS_RAM (MI6) RPU Register

2.4 Register Details

Table 2-7: RPU APB Slave Permission Registers

Register Name	Register Mnemonic	Reference
Global Control RPU Register	**GCR**	See *Table 2-5*
System Interface RPU Register	**SIR**	See *Table 2-5*
Function Control RPU Register	**FCR**	See *Table 2-5*
Trust Protection Unit RPU Register	**TPU**	See *Table 2-5*
Resource Protection Unit RPU Register	**RPU**	See *Table 2-5*
Watchdog Timer 0 RPU Register	**WDT0**	See *Table 2-5*
Watchdog Timer 1 RPU Register	**WDT1**	See *Table 2-5*
Watchdog Timer 2 RPU Register	**WDT2**	See *Table 2-5*
Security Monitor RPU Register	**SMON**	See *Table 2-5*

Register Name	Register Mnemonic	Reference
SIMO Controller RPU Register	**SIMO**	See *Table 2-5*
DVS Controller RPU Register	**DVS**	See *Table 2-5*
AES Keys RPU Register	**AES**	See *Table 2-5*
Real-Time Clock RPU Register	**RTC**	See *Table 2-5*
Wake-Up Timer RPU Register	**WUT**	See *Table 2-5*
Power Sequencer RPU Register	**PWRSEQ**	See *Table 2-5*
Miscellaneous Control Register RPU Register	**MCR**	See *Table 2-5*
GPIO Port 0 RPU Register	**GPIO0**	See *Table 2-5*
GPIO Port 1 RPU Register	**GPIO1**	See *Table 2-5*
Timer 0 RPU Register	**TMR0**	See *Table 2-5*
Timer 1 RPU Register	**TMR1**	See *Table 2-5*
Timer 2 RPU Register	**TMR2**	See *Table 2-5*
Timer 3 RPU Register	**TMR3**	See *Table 2-5*
Timer 4 RPU Register	**TMR4**	See *Table 2-5*
Timer 5 RPU Register	**TMR5**	See *Table 2-5*
HTimer 0 RPU Register	**HTIMER0**	See *Table 2-5*
HTimer 1 RPU Register	**HTIMER1**	See *Table 2-5*
I2C Bus 0 (Bus 0) RPU Register	**I2C0_BUS0**	See *Table 2-5*
I2C Bus 1 (Bus 0) RPU Register	**I2C1_BUS0**	See *Table 2-5*
I2C Bus 2 (Bus 0) RPU Register	**I2C2_BUS0**	See *Table 2-5*
SPIXF Master RPU Register	**SPIXM**	See *Table 2-5*
SPIXF Master Controller RPU Register	**SPIXFC**	See *Table 2-5*
Standard DMA 0 RPU Register	**DMA0**	See *Table 2-5*
Flash Controller 0 RPU Register	**FLC0**	See *Table 2-5*
Flash Controller 1 RPU Register	**FLC1**	See *Table 2-5*
Instruction Cache Controller 0 RPU Register	**ICC0**	See *Table 2-5*
Instruction Cache Controller 1 RPU Register	**ICC1**	See *Table 2-5*
Instruction Cache Controller XIP RPU Register	**SFCC**	See *Table 2-5*

Register Name	Register Mnemonic	Reference
SPI XIP RAM Cache Controller RPU Register	**SRCC**	See *Table 2-5*
Analog-to-Digital Converter RPU Register	**ADC**	See *Table 2-5*
Standard DMA 1 RPU Register	**DMA1**	See *Table 2-5*
Smart DMA RPU Register	**SDMA**	See *Table 2-5*
SD Host Controller (APB) RPU Register	**SDHCCTRL**	See *Table 2-5*
SPIXR Master Controller RPU Register	**SPIXR**	See *Table 2-5*
Pulse Train Engine (Bus 0) RPU Register	**PTG_BUS0**	See *Table 2-5*
1-Wire Module RPU Register	**OWM**	See *Table 2-5*
Semaphores RPU Register	**SEMA**	See *Table 2-5*
UART 0 RPU Register	**UART0**	See *Table 2-5*
UART 1 RPU Register	**UART1**	See *Table 2-5*
UART 2 RPU Register	**UART2**	See *Table 2-5*
SPI 1 RPU Register	**SPI1**	See *Table 2-5*
SPI 2 RPU Register	**SPI2**	See *Table 2-5*
Audio Subsystem RPU Register	**AUDIO**	See *Table 2-5*
True Random Number Generator RPU Register	**TRNG**	See *Table 2-5*
BTLE Registers and IQ RAMs RPU Register	**BTLE**	See *Table 2-5*
I2C 0 (Bus 1) RPU Register	**I2C0_BUS1**	See *Table 2-5*
I2C 1 (Bus 1) RPU Register	**I2C1_BUS1**	See *Table 2-5*
I2C 2 (Bus 1) RPU Register	**I2C2_BUS1**	See *Table 2-5*
Pulse Train Engine (Bus 1) RPU Register	**PTG_BUS1**	See *Table 2-5*

Bits	Name	Access	Reset	Description
31:0	access	R/W	0	**APB Slave Peripheral Access Disable** 0: AHB master read/write access allowed 1: AHB master read/write access denied *This field allows or denies access to the peripheral by one or more AHB masters as shown in Table 2-2. Unused bits should not be changed from their reset default values*

Table 2-8: RPU AHB Slave Permission Register

Register Name	Register Mnemonic	Reference
USB Endpoint Data RPU Register	**USBHS**	**See *Table 2-6***
SDIO/SDHC Target RPU Register	**SDIO**	**See *Table 2-6***
SPI Bus Master FIFO RPU Register	**SPIXFM_FIFO**	**See *Table 2-6***
SPI0 Data Buffer RPU Register	**SPI0**	**See *Table 2-6***
System RAM, Memory Instance 0	**SYSRAM0**	**See *Table 2-6***
System RAM, Memory Instance 1	**SYSRAM1**	**See *Table 2-6***
System RAM, Memory Instance 2	**SYSRAM2**	**See *Table 2-6***
System RAM, Memory Instance 3	**SYSRAM3**	**See *Table 2-6***
System RAM, Memory Instance 4	**SYSRAM4**	**See *Table 2-6***
System RAM, Memory Instance 5	**SYSRAM5**	**See *Table 2-6***
System RAM, Memory Instance 6 - 11	**SYSRAM6-11**	**See *Table 2-6***

Bits	Name	Access	Reset	Description
31:0	access	R/W	0	**AHB Slave Peripheral Access Disable** 0b00: AHB master write/read access allowed 0b01: AHB master write access allowed / read access denied 0b10: AHB master write access denied / read access allowed 0b11: AHB master write/read access denied Each bit pair of this field allows or denies access to the peripheral by one or more AHB masters as shown in *Table 2-2*. Unused bits should not be changed from their reset default values.

3. Memory, Register Mapping, and Access

3.1 Memory, Register Mapping, and Access Overview

The Arm Cortex-M4 architecture defines a standard memory space for unified code and data access. This memory space is addressed in units of single bytes but is most typically accessed in 32-bit (4 byte) units. It may also be accessed, depending on the implementation, in 8-bit (1 byte) or 16-bit (2 byte) widths. The total range of the memory space is 32 bits wide (4GB addressable total), from addresses 0x0000 0000 to 0xFFFF FFFF.

It is important to note, however, that the architectural definition does not require the entire 4GB memory range to be populated with addressable memory instances.

Figure 3-1: Code Memory Mapping

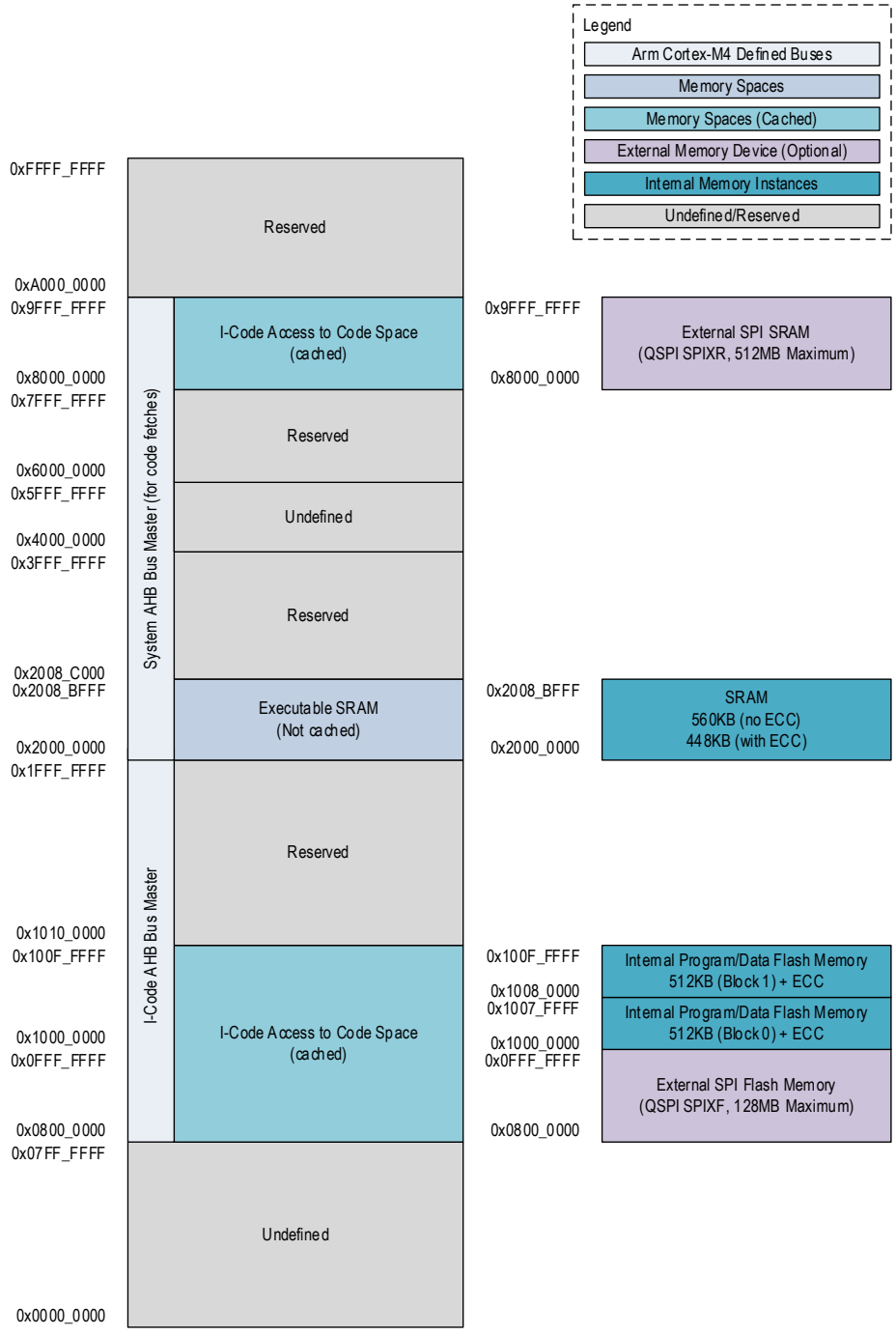
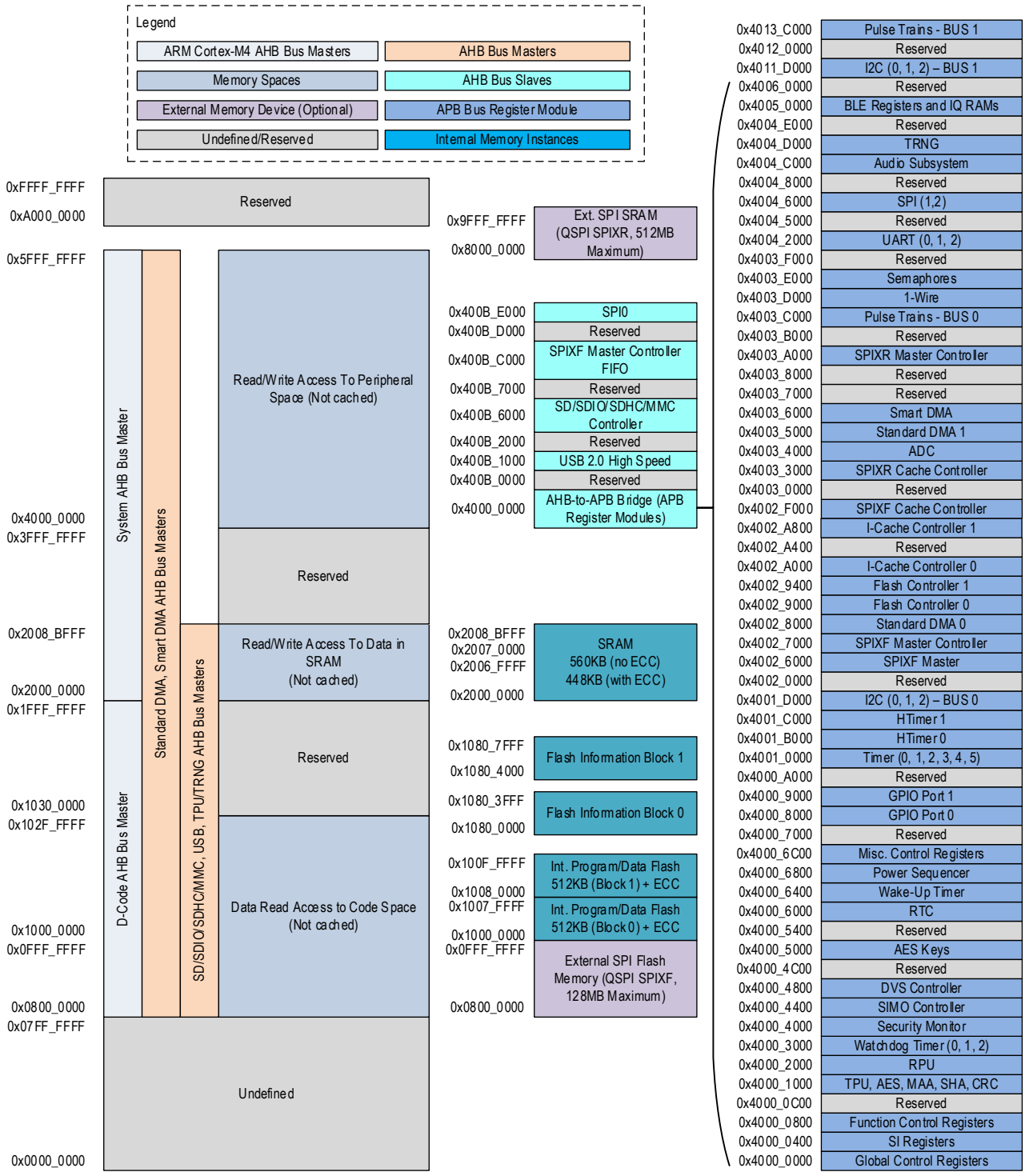


Figure 3-2: Data Memory Mapping



3.2 Field Access Definitions

All the fields that are accessible by user software have distinct access capabilities. Each register table contained in this user guide has an access type defined for each field. The definition of each field access type is presented in *Table 3-1*.

Table 3-1: Field Access Definitions

Access Type	Definition
RO	**Reserved** This access type is reserved for static fields. Reads of this field will return the reset value. Writes are ignored.
DNM	**Reserved. Do Not Modify** Reads of this field will return indeterminate values. Software must first read this field and write the same value whenever writing to this register.
R	**Read Only** Reads of this field will return a value. Writes to the field have no effect on device operation.
W	**Write Only** Reads of this field will return indeterminate values. Writes to the field may change the field's state and may affect device operation.
R/W	**Unrestricted Read/Write** Reads of this field will return a value. Writes to the field may change the field's state and may affect device operation.
RC	**Read-to-Clear** Reads of this field may return a value. Any read of this register will clear the field to 0. Writes to the field have no effect on device operation.
RS	**Read-to-Set** Reads of this field may return a value. Any read of this register will set the field to 1. Writes to the field have no effect on device operation.
R/W0	**Read-Write-0-Only** Reads of this field may return a value. Writing 0 to this field may change the field's state and may affect device operation. Writing 1 to the field has no effect on device operation.
R/W1	**Read-Write-1-Only** Reads of this field may return a value. Writing 1 to this field may change the field's state and may affect device operation. Writing 0 to the field has no effect on device operation.
R/W1C	**Read-Write-1-to-Clear** Reads of this field may return a value. Writing 1 to this field will clear this field to 0. Writing 0 to the field has no effect on device operation.
W1C	**Write-1-to-Clear** Reads of this field will return indeterminate values. Writing 1 to this field will clear this field to 0. Writing 0 to the field has no effect on device operation.
R/W0S	**Read-Write-0-to-Set** Reads of this field may return a value. Writing 0 to this field will set this field to 1. Writing 1 to the field has no effect on device operation.

3.3 Standard Memory Regions

Several standard memory regions are defined for the Arm Cortex-M4 architecture; the use of many of these is optional for the system integrator. At a minimum, the MAX32665/MAX32666, Cortex-M4-based devices, must contain some code and data memory for application code and variable/stack use, as well as certain components which are part of the instantiated core.

3.3.1 Code Space

The code space area of memory is designed to contain the primary memory used for code execution by the device. This memory area is defined from byte address range 0x0000 0000 to 0x1FFF FFFF (0.5GB maximum). Two different standard core bus masters are used by the Cortex-M4 core and Arm debugger to access this memory area. The I-Code AHB bus master is used for instruction decode fetching from code memory, while the D-Code AHB bus master is used for data fetches from code memory. This is arranged so that data fetches avoid interfering with instruction execution.

The MAX32665/MAX32666 code memory mapping is illustrated in *Figure 3-1*. The code space memory area contains the main internal flash memory, which holds most of the instruction code that will be executed on the device. The internal flash memory is mapped into both code and data space from 0x1000 0000 to 0x100F FFFF. It is partitioned as two 512KB blocks of usable flash plus extra flash storage for Error Correction Coding (ECC) check bits, if ECC is enabled. This additional storage is not user accessible, even when ECC is disabled.

This program memory area must also contain the default system vector table and the initial settings for all system exception handlers and interrupt handlers. The reset vector for the device is 0x0000 0000 where a vector to re-direct to 0x1000 0000 is located.

The code space memory on the MAX32665/MAX32666 also contains the mapping for the flash information block, from 0x1080 0000 to 0x1080 7FFF. However, this mapping is generally only present during Analog Devices production test; it is disabled once the information block has been loaded with valid data and the info block lockout option has been set. This memory is accessible for data reads only and cannot be used for code execution. The flash information block is user read only accessible and contains the Unique Serial Number (USN).

Optionally, the SPIXF (SPI Execute In Place Flash) and the SPIXR (SPI Execute In Place RAM) modules can be used to expand the available code and data memory space. This expansion consists of mapping the contents of an external SPI flash memory device (up to 128MB) or an external SPI RAM memory device (up to 512MB) into a read-only area of the code memory map. If enabled, the external SPIXF memory is mapped starting at byte address 0x0800 0000 up to a maximum of 0x0FFF FFFF (for a 128MB device). Also, if enabled, the external SPIXR memory is mapped starting at byte 0x8000 000 up to a maximum of 0x9FFF FFFF (for a 512MB device). This external memory can be used for code execution as well as static data storage.

3.3.2 SRAM Space

The SRAM area of memory is intended to contain the primary SRAM data memory of the device and is defined from byte address range 0x2000 0000 to 0x3FFF FFFF (0.5GB maximum). This memory can be used for general purpose variable and data storage, code execution, and the Arm Cortex-M4 stack.

The MAX32665/MAX32666 data memory mapping is illustrated in *Figure 3-2* and the SRAM configuration is defined in *Table 3-2*. This memory area contains the main system SRAM. The size of the internal SRAM is 560KB when not using ECC. Its address range is 0x2000 0000 to 0x2008 BFFF. If ECC is enabled, the SRAM size decreases to 448KB. The address range with ECC enabled is 0x2000 0000 to 0x2006 FFFF.

The entirety of the SRAM memory space on the MAX32665/MAX32666 is contained within the dedicated Arm Cortex-M4 SRAM bit-banding region from 0x2000 0000 to 0x200F FFFF (1MB maximum for bit-banding). This means that the CPU can access the entire SRAM either using standard byte/word/doubleword access or using bit-banding operations. The bit-banding mechanism allows any single bit of any given SRAM byte address location to be set, cleared, or read individually by reading from or writing to a corresponding doubleword (32-bit wide) location in the bit-banding alias area.

The alias area for the SRAM bit-banding is located beginning at 0x2200 0000 and is a total of 32MB maximum, which allows the entire 1MB bit banding area to be accessed. Each 32-bit (4 byte aligned) address location in the bit-banding alias area

translates into a single bit access (read or write) in the bit-banding primary area. Reading from the location performs a single bit read, while writing either a 1 or 0 to the location performs a single bit set or clear.

Note: The Arm Cortex-M4 core translates the access in the bit-banding alias area into the appropriate read cycle (for a single bit read) or a read-modify-write cycle (for a single bit set or clear) of the bit-banding primary area. This means that bit-banding is a core function (i.e., not a function of the SRAM memory interface layer or the AHB bus layer), and thus is only applicable to accesses generated by the core itself. Reads/writes to the bit-banding alias area by other (non-Arm-core) bus masters will not trigger a bit-banding operation and will instead result in an AHB bus error.

The SRAM area on the MAX32665/MAX32666 can be used to contain executable code. Code stored in the SRAM is accessed directly for execution (using the system bus) and is not cached. The SRAM is also where the Arm Cortex-M4 stack must be located, as it is the only general-purpose SRAM memory on the device. A valid stack location inside the SRAM must be set by the system exception table (which is, by default, stored at the beginning of the internal flash memory).

The MAX32665/MAX32666 specific AHB Bus Masters can access the SRAM to use as general storage or working space. Specifically, in the case of the USB interface, SRAM memory area can be used to store the descriptor table for the endpoint buffers as well as the endpoint buffers themselves.

Table 3-2: SRAM Configuration

System RAM Block #	Size (Words)	Start Address	End Address	ECC SRAM Complement
sysram0	8K	2000-0000	2000-7FFF	sysram6
sysram1	8K	2000-8000	2000-FFFF	sysram7
sysram2	16K	2001-0000	2001-FFFF	sysram8
sysram3	16K	2002-0000	2002-FFFF	sysram9
sysram4	32K	2003-0000	2004-FFFF	sysram10
sysram5	32K	2005-0000	2006-FFFF	sysram11
sysram6	2K	2007-0000	2007-1FFF	-
sysram7	2K	2007-2000	2007-3FFF	-
sysram8	4K	2007-4000	2007-7FFF	-
sysram9	4K	2007-8000	2007-BFFF	-
sysram10	8K	2007-C000	2008-3FFF	-
sysram11	8K	2008-4000	2008-BFFF	-

3.3.3 Peripheral Space

The peripheral space area of memory is intended for mapping of control registers, internal buffers/working space, and other features needed for the firmware control of non-core peripherals. It is defined from byte address range 0x4000 0000 to 0x5FFF FFFF (0.5GB maximum). On the MAX32665/MAX32666, all device-specific module registers are mapped to this memory area, as well as any local memory buffers or FIFOs which are required by modules.

As with the SRAM region, there is a dedicated 1MB area at the bottom of this memory region (from 0x4000 0000 to 0x400F FFFF) that is used for bit-banding operations by the Arm core. Four-byte-aligned read/write operations in the peripheral bit-banding alias area (32MB in length, from 0x4200 0000 to 0x43FF FFFF) are translated by the core into read/mask/shift or read/modify/write operation sequences to the appropriate byte location in the bit-banding area.

Note: The bit-banding operation within peripheral memory space is, like bit-banding function in SRAM space, a core remapping function. As such, it is only applicable to operations performed directly by the Arm core. If another memory bus master accesses the peripheral bit-banding alias region, the bit-banding remapping operation will not take place. In this case, the bit-banding alias region will appear to be a non-implemented memory area (causing an AHB bus error).

On the MAX32665/MAX32666, access to the region that contains most peripheral registers (0x4000 0000 to 0x400F FFFF) goes from the AHB bus through an AHB-to-APB bridge. This allows the peripheral modules to operate on the lower power APB bus matrix. This also ensures that peripherals with slower response times do not tie up bandwidth on the AHB bus, which must necessarily have a faster response time since it handles main application instruction and data fetching.

A secondary region within the peripheral memory space (0x0400B 0000 to 0x400F FFFF) allows peripherals that require more rapid data transfer to handle this data transfer using their own local AHB slave instances (instead of going indirectly through the AHB-to-APB bridge). This allows peripherals which have FIFOs or other functions requiring large amounts of data to be transferred quickly (such as the SD/SDIO/SDHC/MMC or communications peripherals like SPI) to benefit from the more rapid data transfer rate of the AHB bus.

3.3.4 External RAM Space

The external RAM space area of memory is intended for use in mapping off-chip external memory and is defined from byte address range 0x6000 0000 to 0x9FFF FFFF (1GB maximum). The MAX32665/MAX32666 implements support for external SPI SRAM. The external SPI SRAM SPIXR interface is mapped to byte address 0x8000 000 to 0x9FFF FFFF (up to 512MB).

3.3.5 External Device Space

The external device space area of memory is intended for use in mapping off-chip device control functions onto the AHB bus. This memory space is defined from byte address range 0xA000 0000 to 0xDFFF FFFF (1GB maximum). The MAX32665/MAX32666 does not implement this memory area.

3.3.6 System Area (Private Peripheral Bus)

The system area (private peripheral bus) memory space contains register areas for functions that are only accessible by the Arm core itself (and the Arm debugger, in certain instances). It is defined from byte address range 0xE000 0000 to 0xE00F FFFF. This APB bus is restricted and can only be accessed by the Arm core and core-internal functions. It cannot be accessed by other modules which implement AHB memory masters, such as the SD/SDIO/SDHC/MMC interface.

In addition to being restricted to the core, application code is only allowed to access this area when running in the privileged execution mode (as opposed to the standard user thread execution mode). This helps ensure that critical system settings controlled in this area are not altered inadvertently or by errant code that should not have access to this area.

Core functions controlled by registers mapped to this area include the SysTick timer, debug and tracing functions, the NVIC (interrupt handler) controller, and the Flash Breakpoint controller.

3.3.7 System Area (Vendor Defined)

The system area (vendor defined) memory space is reserved for vendor (system integrator) specific functions that are not handled by another memory area. It is defined from byte address range 0xE010 0000 to 0xFFFF FFFF. The MAX32665/MAX32666 does not implement this memory region.

3.4 Device Memory Instances

This section details physical memory instances on the MAX32665/MAX32666 (including internal flash memory and SRAM instances) that are accessible as standalone memory regions using either the AHB or APB bus matrix. Memory areas which are only accessible via FIFO interfaces, or memory areas consisting of only a few registers for a specific peripheral, are not covered here.

3.4.1 Main Program Flash Memory

The main program flash memory is 1MB in size (two banks of 512KB) and consists of 128 logical pages of 8,192 Bytes per page.

3.4.2 Cache Memories

3.4.2.1 Instruction Cache Controller 0 (ICC0)

This internal flash memory instruction cache is 16,384 bytes in size and is used to cache instructions fetched using the I-Code bus, including instructions fetched from the internal flash memory. It is dedicated to CPU0. This instruction cache controller is referred to as ICC0 throughout this document. See *Figure 4-7* for cache controller control details.

3.4.2.2 Instruction Cache Controller 1 (ICC1)

This internal flash memory instruction cache is 16,384 Bytes in size and is used to cache instructions fetched using the I-Code bus, including instructions fetched from the internal flash memory. It is dedicated to CPU1. This instruction cache controller is referred to as ICC0 throughout this document. See *Figure 4-7* for cache controller control details.

3.4.2.3 SPI Execute-In-Place Flash Cache Controller (SFCC)

The SFCC instruction cache, is also 16,384 Bytes and is used to cache instructions fetched from an external SPI memory device. SFCC is only available if the SPIXF controller is enabled. It is supported by the SPIXF interface. See *Figure 4-7* for cache controller control details.

Note: The instruction caches, ICC0, ICC1, and SFCC are used for instruction fetches only. Data fetches (including code literal values) from the internal flash memory or external SPIXF memory do not use the instruction cache.

3.4.2.4 SPI Execute-In-Place RAM Cache Controller (SRCC)

The SRCC cache is a dedicated 16,384 Byte 2-way set-associative Least Recently Used (LRU) write-through cache. It is supported by the SPIXR interface. See *Figure 4-7* for cache controller control details.

3.4.3 Information Block Flash Memory

The information block 0 is a separate flash instance of 16kB that is used to store trim settings (option configuration and analog trim) as well as other nonvolatile device-specific information. The information block 0 also contains the Unique Serial Number (USN). The USN is a 104 bit field. USN bits 0 thru 7 contain the die revision.

Figure 3-3: Unique Serial Number Format

		Bit Position																													
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Address	0x10800000	USN bits 16 - 0																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	0x10800004	x	USN bits 47-17																														
	0x10800008	USN bits 64 - 48																	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
	0x1080000C	x	USN bits 95 - 65																														
	0x10800010	x	x	x	x	x	x	x	x	x	USN bits 103 - 96								x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

To access the Unique Serial Number, the information block must be unlocked. To unlock this block, use the following steps:

1. Write 0x3A7F5CA3 to *FLC_ACTNL*.
2. Write 0xA1E34F20 to *FLC_ACTNL*.
3. Write 0x0x9608B2C1 to *FLC_ACTNL*.
4. Information block is now accessible.

To re-lock the information block to prevent access, simply write any 32-bit word (with a value other than one of the three values required for the unlock sequence above) to *FLC_ACTNL* which is at 0x4002_9040.

3.4.4 System SRAM

The system SRAM is 560KB in size and can be used for general purpose data storage, the Arm system stack, USB data transfers (endpoints), SD Host Controller (SDIO) interface, TPU and code execution if desired.

3.4.5 AES Key and Working Space Memory

The AES key memory and working space for AES operations (including input and output parameters) are in a dedicated register file memory tied to the AES engine block. This AES memory is mapped into AHB space for rapid firmware access.

3.4.6 MAA Key and Working Space Memory

The MAA contains a dedicated memory for key storage, input and output parameters for operations, and working space. It is mapped into the AHB memory space for ease of loading and unloading.

3.4.7 TPU Memory

The MAX32665/MAX32666 contains a specialized 128-bit memory that is designed to preserve critical data (such as a 128-bit AES key) even when the device is in the lowest power-saving state. As long as the RTC power supply is still available, the contents of this memory will be retained, even if the AES block and the main SRAM are shut down completely.

The Secure Key Storage Area consists of four V_{COREA} supply backed 32-bit registers: TPU_TSR_SKS0, TPU_TSR_SKS1, TPU_TSR_SKS2, and TPU_TSR_SKS3.

3.5 AHB Interfaces

This section details memory accessibility on the AHB and the organization of AHB master and slave instances.

3.5.1 Core AHB Interfaces

3.5.1.1 I-Code

This AHB master is used by the Arm core for instruction fetching from memory instances located in code space from byte addresses 0x0000 0000 to 0x1FFF FFFF. This bus master is used to fetch instructions from the internal flash memory and the external SPIF flash memory (if SPIXF is enabled). Instructions fetched by this bus master are returned by the instruction cache, which in turn triggers a cache line fill cycle to fetch instructions from the internal flash memory or the external SPIXF flash memory when a cache miss occurs.

3.5.1.2 D-Code

This AHB master is used by the Arm core for data fetches from memory instances located in code space from byte addresses 0x0000 0000 to 0x1FFF FFFF. This bus master has access to the internal flash memory, the external SPIXF flash memory (if SPIXF is enabled), and the information block.

3.5.1.3 System

This AHB master is used by the Arm core for all instruction fetches and data read and write operations involving the SRAM data cache. The APB mapped peripherals (through the AHB-to-APB bridge) and AHB mapped peripheral and memory areas are also accessed using this bus master.

3.5.2 AHB Masters

3.5.2.1 Standard DMA

The Standard DMA bus master has access to all off-core memory areas accessible by the System bus. It does not have access to the Arm Private Peripheral Bus area.

3.5.2.2 SDIO

The SDIO bus master has access to all off-core memory areas accessible by the System bus. It does not have access to the Arm Private Peripheral Bus area.

3.5.2.3 Trust Protection Unit (TPU)

The TPU bus master has access to all off-core memory areas accessible by the System bus. It does not have access to the Arm Private Peripheral Bus area.

3.6 Peripheral Register Map

3.6.1 APB Peripheral Base Address Map

Table 3-3 contains the base address for each of the APB mapped peripherals. The base address for a given peripheral is the start of the register map for the peripheral. For a given peripheral, the address for a register within the peripheral is defined as the APB peripheral base address plus the registers offset.

Table 3-3: APB Peripheral Base Address Map

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
Global Control	GCR_	0x4000 0000	0x4000 03FF
System Interface	SIR_	0x4000 0400	0x4000 07FF
Function Control	FCR_	0x4000 0800	0x4000 0BFF
Trust Protection Unit	TPU_	0x4000 1000	0x4000 1FFF
Resource Protection Unit	RPU_	0x4000 2000	0x4000 2FFF
Watchdog Timer 0	WDT0_	0x4000 3000	0x4000 33FF
Watchdog Timer 1	WDT1_	0x4000 3400	0x4000 37FF
Watchdog Timer 2	WDT2_	0x4000 3800	0x4000 3BFF
Security Monitor	SMON_	0x4000 4000	0x4000 43FF
SIMO Controller	SIMO_	0x4000 4400	0x4000 47FF
DVS Controller	DVS_	0x4000 4800	0x4000 4BFF
AES Keys	AES_	0x4000 5000	0x4000 53FF
Real-Time Clock	RTC_	0x4000 6000	0x4000 63FF
Wake Up Timer	WUT_	0x4000 6400	0x4000 67FF
Power Sequencer	PWRSEQ_	0x4000 6800	0x4000 6BFF
Misc. Control Registers	MCR_	0x4000 6C00	0x4000 6FFF
GPIO Port 0	GPIO0_	0x4000 8000	0x4000 8FFF
GPIO Port 1	GPIO1_	0x4000 9000	0x4000 9FFF
Timer 0	TMR0_	0x4001 0000	0x4001 0FFF
Timer 1	TMR1_	0x4001 1000	0x4001 1FFF
Timer 2	TMR2_	0x4001 2000	0x4001 2FFF
Timer 3	TMR3_	0x4001 3000	0x4001 3FFF
Timer 4	TMR4_	0x4001 4000	0x4001 4FFF
Timer 5	TMR5_	0x4001 5000	0x4001 5FFF
HTimer 0	HTMR0_	0x4001 B000	0x4001 BFFF
HTimer 1	HTMR1_	0x4001 C000	0x4001 CFFF
I2C 0 (bus 0)	I2C0_BUS0_	0x4001 D000	0x4001 DFFF

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
I2C 1 (bus 0)	I2C1_BUS0_	0x4001 E000	0x4001 EFFF
I2C 2 (bus 0)	I2C2_BUS0_	0x4001 F000	0x4001 FFFF
SPIXF Master	SPIXF_	0x4002 6000	0x4002 6FFF
SPIXF Master Controller	SPIXFC_	0x4002 7000	0x4002 7FFF
Standard DMA 0	DMA0_	0x4002 8000	0x4002 8FFF
Flash Controller 0	FLC0_	0x4002 9000	0x4002 93FF
Flash Controller 1	FLC1_	0x4002 9400	0x4002 97FF
Instruction-Cache Controller 0	ICC0_	0x4002 A000	0x4002 A3FF
Instruction Cache Controller 1	ICC1_	0x4002 A800	0x4002 ABFF
Instruction Cache Controller XIP	SFCC_	0x4002 F000	0x4002 FFFF
External Memory Cache Controller	SRCC_	0x4003 3000	0x4003 3FFF
Analog to Digital Converter	ADC_	0x4003 4000	0x4003 4FFF
Standard DMA 1	DMA1_	0x4003 5000	0x4003 5FFF
Reserved	-	0x4003 6000	0x4003 6FFF
Reserved	-	0x4003 7000	0x4003 7FFF
SPIXR Master Controller	SPIXR_	0x4003 A000	0x4003 AFFF
Pulse Train Engine (bus 0)	PTG_BUS0_	0x4003 C000	0x4003 CFFF
1-Wire	OWM_	0x4003 D000	0x4003 DFFF
Semaphores	SEMA_	0x4003 E000	0x4003 EFFF
UART 0	UART0_	0x4004 2000	0x4004 2FFF
UART 1	UART1_	0x4004 3000	0x4004 3FFF
UART 2	UART2_	0x4004 4000	0x4004 4FFF
SPI1	SPI1_	0x4004 6000	0x4004 6FFF
SPI2	SPI2_	0x4004 7000	0x4004 7FFF
Audio Subsystem	AUDIO_	0x4004 C000	0x4004 CFFF
TRNG	TRNG_	0x4004 D000	0x4004 DFFF
BTLE Registers and IQ RAMs	BTLE_	0x4005 0000	0x4005 FFFF
I2C 0 (bus 1)	I2C0_BUS1_	0x4011 D000	0x4011 DFFF
I2C 1 (bus 1)	I2C1_BUS1_	0x4011 E000	0x4011 EFFF
I2C 2 (bus 1)	I2C2_BUS1_	0x4011 F000	0x4011 FFFF
Pulse Train Engine (bus 1)	PTG_BUS1_	0x4013 C000	0x4013 CFFF

3.6.2 AHB Peripheral Base Address Map

Table 3-4 contains the base address for each of the AHB mapped peripherals. The base address for a given peripheral is the start of the register map for the peripheral. For a given peripheral, the address for a register within the peripheral is defined as the AHB peripheral base address plus the registers offset.

Table 3-4: AHB Peripheral Base Address Map

AHB Peripheral Register Name	Register Prefix	AHB Base Address	AHB End Address
USB Hi-Speed Host	USBHS_	0x400B 1000	0x400B 1FFF
SDIO/SDHC Controller (AHB)	SDIO_	0x400B 6000	0x400B 6FFF
SPIXF Master Controller FIFO	SPIXFM_FIFO_	0x400B C000	0x400B CFFF
SPI0	SPI0_	0x400B E000	0x400B E3FF

3.7 Error Correction Coding (ECC) Module

This device features an Error Correction Coding (ECC) module which helps ensures data integrity by detecting and correcting bit corruption of memory arrays. More specific, this feature is Single Error Correcting, Double Error Detecting (SEC-DED). It corrects any single bit flip, detects 2-bit errors, and features a transparent zero wait state operation for reads.

The ECC works by creating check bits for all data written to memory. These check bits are then stored along with the data. During a read, both the data and check bits are used to determine if one or more bits have become corrupt. If a single bit has been corrupted this can be corrected. If two bits have been corrupted, it will be detected, but not corrected.

If only one bit is determined to be corrupt, reads will contain the “corrected” value. Reading memory does not correct the errored value stored at the read memory location. It is up to the application firmware to determine the appropriate time and method to write the correct data to memory. It is strongly recommended that the application firmware correct the memory as soon as possible to minimize the chance of a second bit from becoming corrupt, resulting in data loss. Since ECC error checking only occurs during a “read” operation, it is recommended that the application periodically “reads” critical memory so that errors can be identified and corrected.

3.7.1 SRAM

To integrate the ECC SEC-DED module into a RAM, there must be a secondary RAM instance to store the check bits. In the case of a 32-bit wide RAM, 7 check bits are needed. The secondary check bit RAM can hold the 7 check bits in each byte, therefore needs ¼ the number of words as the RAM itself. Also, the address sent to the check bit RAM is divided by 4 to map the 32-bit data words to 8-bit check bit addresses.

For example, a 32-bit by 8192 word RAM would need a 32-bit by 2048 word sized secondary RAM instance. When ECC is enabled, each system RAM module requires an appropriately sized secondary RAM.

3.7.1.1 Limitations

Any read from non-initialized RAM could trigger an ECC error since the random check bits will most likely not match the random data bits. Writing the memory to all zeroes at bootup can prevent this at the expense of the time required.

3.7.2 FLASH

The flash implements the SEC-DED ECC by including an additional 9 check bits for every 128 data bits. These additional bits do not appear in the device's memory map making the additional bits inaccessible by the user. Reads from and writes to the flash memory behave the same whether ECC is enabled or not. However, it is recommended to always write the flash in 128-bit blocks when ECC is enabled. With ECC enabled, writing 32 bits to the flash will set the check bits for the full 128-bit word. Since the check bits are also stored in flash, another 32-bit write into the same 128-bit word will fail because the device will be unable to update the check bits.

3.7.2.1 Limitations

Any read from a fully erased section flash memory will trigger an ECC error. A 128-bit read of an erased flash location will read 0xFDFFFFFF_FFFFFFFF_FFFFFFFF_FFFFFFFF. The 9 check bits will be 0x1FF. The flash controller sees the ECC as 1FF and changes bit 121 to a 0 to fix what it thinks is a single-bit error. In order to read all F's you would need to disable ECC. The onus is on the software application to keep track of which flash locations are unprogrammed and avoid reading from them when ECC is enabled.

3.7.3 Cache

Any type of ECC error (single or double) is treated as a cache miss. There are separate ECC check bits for both the data RAM and tag RAM inside the cache.

4. System, Power, Clocks, Reset

Different peripherals and subsystems use several clocks. These clocks are highly configurable by software, allowing developers to select the combination of application performance and power savings required for the target systems.

The selected System Oscillator (SYS_OSC) is the clock source for most internal blocks. Select SYS_OSC from the following clock sources:

- 96MHz Internal High-Frequency Oscillator
- 60MHz Low-Power Internal Oscillator
- 7.3728MHz Internal Oscillator
  - Selectable for UART baud rate generation
- 8kHz Internal Ultra-Low Power Nano-Ring Oscillator
- 32.768kHz External Crystal Oscillator
  - The clock source for the Real-Time Clock (RTC)
- 32MHz External Crystal Oscillator
  - The clock source for the Bluetooth 5 radio

The selected SYS_OSC is the input to the system oscillator prescaler to generate the System Clock (SYS_CLK). The system oscillator prescaler divides SYS_OSC by a prescaler using the *GCR_CLKCN.psc* field as shown in *Equation 4-1*.

Equation 4-1: System Clock Scaling

$$\text{SYS\_CLK} = \frac{\text{SYS\_OSC}}{2^{\text{psc}}}$$

GCR_CLKCN.psc is selectable from 0 to 7, resulting in divisors of 1, 2, 4, 8, 16, 32, 64 or 128.

SYS_CLK drives the Arm Cortex-M4 with FPU cores and is used to generate the following internal clocks as shown below:

- Advanced High-Performance Bus (AHB) Clock
  - HCLK = SYS_CLK
- Advanced Peripheral Bus (APB) Clock,
  - $\text{PCLK} = {}^{\text{SYS\_CLK}}/_{2}$
- Always On Domain (AOD) Clock,
  - $\text{AODCLK} = {}^{\text{PCLK}}/_{2^{\text{GCR\_PCKDIV.aoncd}}}$
- *GCR_PCKDIV.aoncd* is selectable from 0 to 3 for divisors of 1, 2, 4 or 8

There are additional internal clocks that are generated. These clocks are independent of SYS_OSC and SYS_CLK as follows:

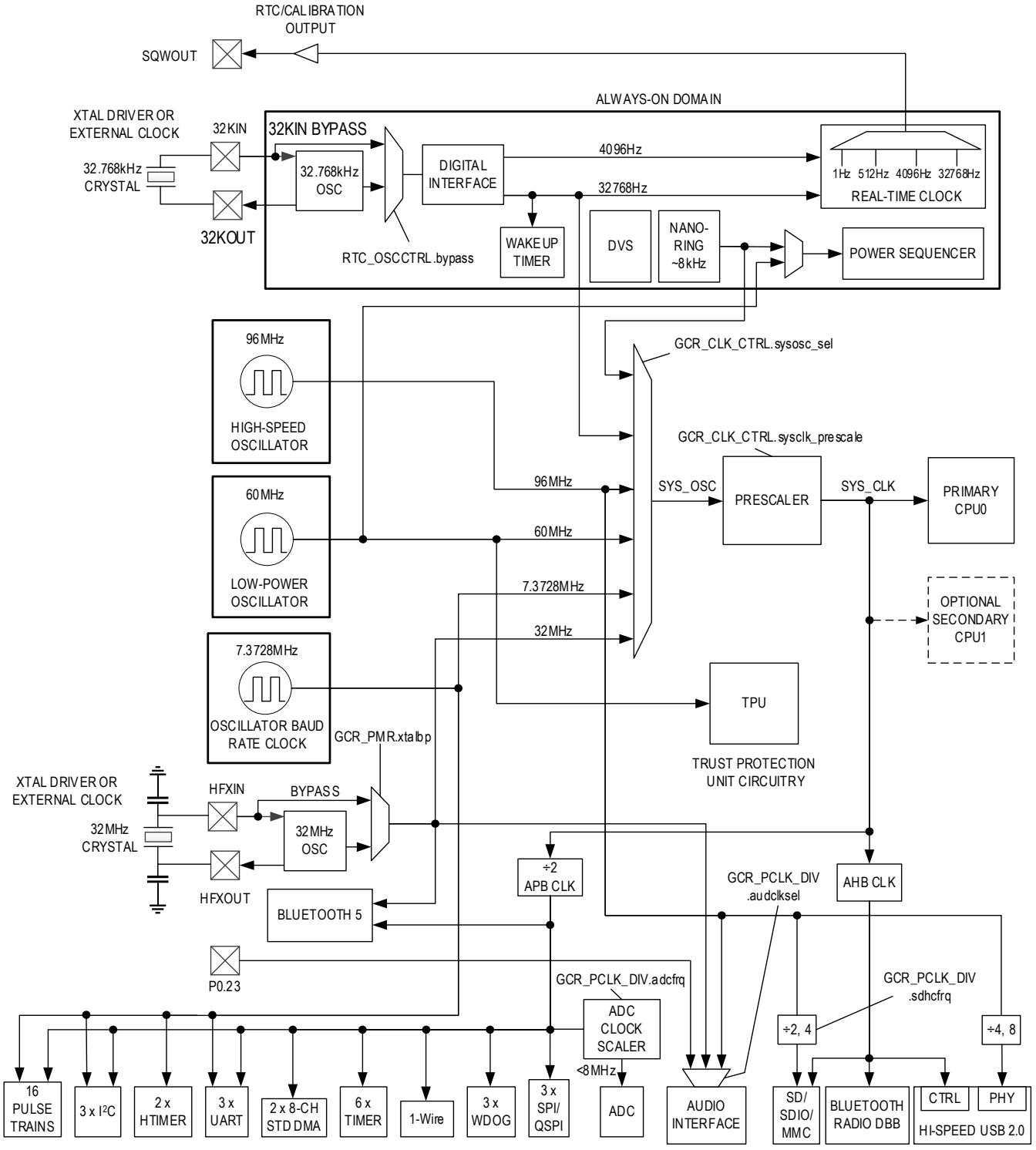
- The USB PHY uses the 96MHz oscillator
- The SDHC/SDIO controller uses the 96MHz oscillator divided by 2
- The RTC uses the 32.768kHz oscillator
- The Trust Protection Unit (TPU) uses the 60MHz Low-Power Internal Oscillator

All oscillators are reset to default at Power-On Reset (POR) and System Reset. Oscillator status is not reset by a Soft Reset or Peripheral Reset.

4.1 Oscillator Sources and Clock Switching

On Power-On Reset (POR) and System Reset, all oscillator states are reset to the default: The 60MHz and 8kHz oscillators are enabled, while the 96MHz, 32MHz 32.768kHz and 7.3728MHz oscillators are disabled. Oscillators are not reset on Soft Reset or Peripheral Reset.

Figure 4-1: Clock Block Diagram



4.1.1 Oscillator Implementation

Before using any oscillator, the desired oscillator must first be enabled by setting the oscillator's enable bit in the *GCR_CLKCN* register. Once an oscillator's enable bit is set, the oscillator's ready bit must read 1 before attempting to use the oscillator as a system oscillator source. The oscillator-ready status flags are contained in the *GCR_CLKCN* register.

Once the corresponding oscillator-ready bit is set, the oscillator can be selected as SYS_OSC by configuring the Clock Source Select field (*GCR_CLKCN.clksel*).

Any time software changes SYS_OSC by changing *GCR_CLKCN.clksel,* the Clock Ready bit *GCR_CLKCN.ckrdy* is automatically cleared to indicate that a SYS_OSC switchover is in progress. When the switchover is complete, *GCR_CLKCN.ckrdy* is set to 1 by hardware indicating the oscillator selected is ready for use.

CAUTION: When switching the SYS_OSC or touching the SYS_OSC prescaler (GCR_CLK_CTRL.sysclk_prescale), any device peripherals using SYS_CLK, APB clock, or AHB clock become unstable. The software should understand that all peripherals should be disabled before switching SYS_OSC or touching the SYS_OSC prescaler.

4.1.2 96MHz Internal Main High-Speed Oscillator

The devices are available with a 96MHz internal high-speed oscillator. This is the fastest oscillator and draws the most power.

This oscillator is also used by the USB PHY and the SDHC. If the USB or SDHC is enabled, the 96MHz oscillator must be enabled, independent of the selection of SYS_OSC.

Optionally, this oscillator can be powered down automatically when in DEEPSLEEP mode by setting register bit *GCR_PM.hirc96mpd*.

4.1.3 60MHz Low Power Internal Oscillator

This is a low-power internal oscillator that can be selected as SYS_OSC. This oscillator is automatically selected as SYS_OSC after a System Reset or POR. This oscillator is also the dedicated clock for the TPU. If the TPU is enabled, the 60MHz internal oscillator must be enabled, independent of the selection of SYS_OSC. When used as the TPU clock it can be divided by 2.

4.1.4 32MHz Bluetooth Radio Oscillator

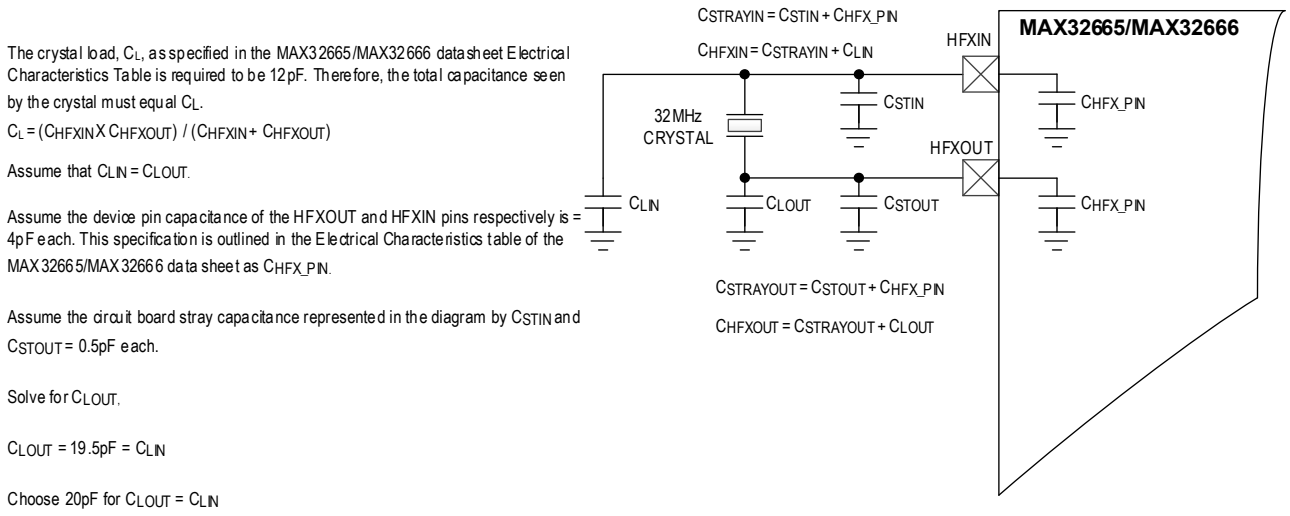
This is the oscillator that directly drives the Bluetooth radio. It can also be selected as SYS_OSC. It is important to use the correct capacitor values on the PCB when connecting the crystal. *Figure 4-2* depicts the method to determine the capacitor values C_{LIN} and C_{LOUT}. In order to enable this oscillator, the Bluetooth LDOs must be enabled by setting the *GCR_BTLELDOCN.ldowen, GCR_BTLELDOCN.ldowoen*, *GCR_BTLELDOCN.ldowvsel, GCR_BTLELDOCN.ldowovsel* register fields.

In order to use this oscillator as SYS_OSC, the following steps must be followed:

1. Enable the 60MHz Low Power Internal Oscillator by setting *GCR_CLKCN.hirc_en*.
2. Wait until *GCR_CLKCN.hirc_rdy* is set. The 60MHz Low Power Internal Oscillator is now operating.
3. Enable the 32MHz Bluetooth Radio Oscillator by setting *GCR_CLKCN.x32M_en*.
4. Wait until *GCR_CLKCN.x32M_r*dy is set. The 32MHz Bluetooth Radio Oscillator is now operating.
5. Set *GCR_CLKCN.clksel* = 2. This will select the 32MHz Bluetooth Radio Oscillator as the SYS_OSC.
6. Wait until *GCR_CLKCN.ckrdy* is set. The 32MHz Bluetooth Radio Oscillator is now operating as the SYS_OSC.

Note: The 60MHz Low Power Internal Oscillator must remain enabled while the 32MHz Bluetooth Radio Oscillator is operating as SYS_OSC.

Figure 4-2: Example 32MHz Crystal Capacitor Determination



4.1.5 7.3728MHz Internal Oscillator

The 7.3728MHz internal oscillator is a very low-power internal oscillator that can be selected as SYS_OSC.

This oscillator can optionally be selected as a dedicated baud rate clock for the UARTs. This selection becomes valid if the SYS_OSC selected does not allow the targeted UART baud rate.

Firmware selection of the voltage that controls this oscillator is controlled by the register bit *GCR_CLKCN.hirc8m_vs*. The internal CPU core supply voltage (V_{CORE}) is the default option. The external pin V_{DDA} can also be selected. The V_{DDA} pin goes to an internal 1V regulator that also provides the analog supply voltage for this device.

This oscillator can optionally be automatically powered down when in DEEPSLEEP mode by setting register bit *GCR_PM.hirc8mpd*.

This oscillator is disabled by default at power-up.

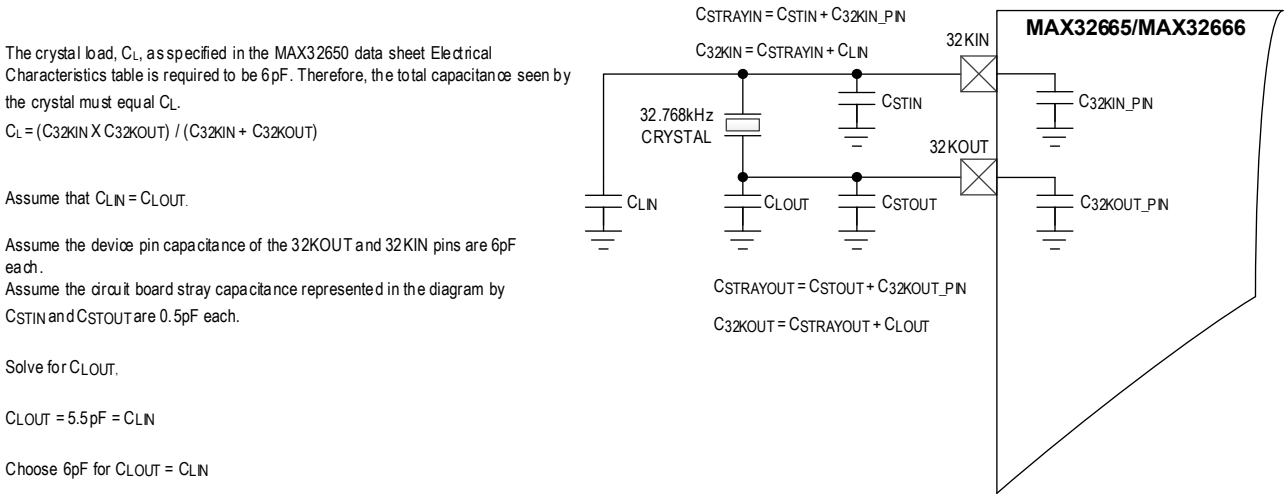
4.1.6 32.768kHz External Crystal Oscillator

The 32.768kHz external crystal oscillator is a very low-power internal oscillator that can be selected as SYS_OSC. This oscillator can optionally use a 32.768kHz input clock instead of an external crystal. The internal 32.768kHz clock is available as an output on GPIO as an alternate function (SQWOUT).

This oscillator is the dedicated clock for the real-time Clock (RTC). If the RTC is enabled, the 32.768kHz external oscillator must be enabled independent of the selection of SYS_OSC. This oscillator is disabled at power-up.

It is essential to use the correct capacitor values on the PCB when connecting the crystal. *Figure 4-3* depicts the method to determine the capacitor values C_{LIN} and C_{LOUT}.

Figure 4-3: Example 32.768kHz Crystal Capacitor Determination



With reference to *Figure 4-3*, calculate the values of C_{LOUT} and C_{LIN} using the following steps:

1. The crystal load, C_L, as specified in the device data sheet electrical characteristics table, is required to be 6pF. Therefore, the total capacitance seen by the crystal must equal C_L.

   Equation 4-2: Determining Load Capacitance for ERTCO

   $$C_L = C_{32KIN} \times C_{32KOUT}/(C_{32KIN} + C_{32KOUT}).$$

2. $C_{LIN} = C_{LOUT}$
3. The device pin capacitance of the 32KOUT and 32KIN pins are 6pF each.
4. Assume the circuit board stray capacitance represented in the diagram by C_{STIN} and C_{STOUT} are 0.5pf each.
5. Solve for C_{LOUT}

   a. $C_{LOUT} = 5.5pF = C_{LIN}$
6. Choose 6pF as the closest standard value for $C_{LOUT} = C_{LIN}$

4.1.7 8kHz Ultra-Low Power Nano-Ring Internal Oscillator

This is an ultra-low power internal oscillator that can be selected as SYS_OSC.

This oscillator is enabled at power-up and cannot be disabled by firmware.

4.2 Operating Modes

The MAX32665/MAX32666 provides four operating modes:

- ACTIVE
- SLEEP
- DEEPSLEEP
- BACKUP

ACTIVE is the highest performance operating mode. Any low power state can wake up to ACTIVE by a wakeup event shown in *Table 4-1*.

Table 4-1: Wakeup Sources

Operating Mode	Wakeup Source
SLEEP	Interrupts (RTC, GPIO, USB, Comparators), RSTN assertion, Wakeup Timer.

DEEPSLEEP	Interrupts (RTC, GPIO, USB, Comparators), RSTN assertion, Wakeup Timer.
BACKUP	Interrupts (RTC, GPIO, USB, Comparators), RSTN assertion, Wakeup Timer.

The Arm Cortex-M family of CPUs have two built-in low power modes, designated SLEEP and DEEPSLEEP. Implementation of these low-power modes are specific to the microcontroller's design. These modes are enabled using the System Control Register (SCR), an Arm Cortex System Control Block register. Write register bit SCR.*deepsleep* to select the low power mode as shown in the pseudocode below.

```
SCR.sleepdeep = 0; // SLEEP mode enabled
SCR.sleepdeep = 1; // DEEPSLEEP mode enabled
```

Once enabled, the device enters the enabled low power mode when either a WFI (Wait For Interrupt) or WFE (Wait For Event) instruction is executed.

Refer to the Arm Cortex-M4 core reference for more information on SCR.

4.2.1 ACTIVE Mode

This is the highest performance mode. All internal clocks, registers, memory, and peripherals are enabled. The CPU is running and executing application code. All oscillators are available.

Dynamic clocking allows firmware to selectively enable or disable clocks and power to individual peripherals, providing the optimal mix of high-performance and power conservation. Internal RAM that can be enabled, disabled, or placed in low-power RAM Retention Mode include data SRAM memory blocks, on-chip caches, and on-chip FIFOs.

4.2.2 SLEEP Low Power Mode

This is a low power mode that suspends the CPU with a fast wakeup time to ACTIVE mode. It is like ACTIVE mode except the CPU clock is disabled, which temporarily prevents the CPU from executing code. All oscillators remain active if enabled and the Always On Domain (AOD) and RAM retention is enabled.

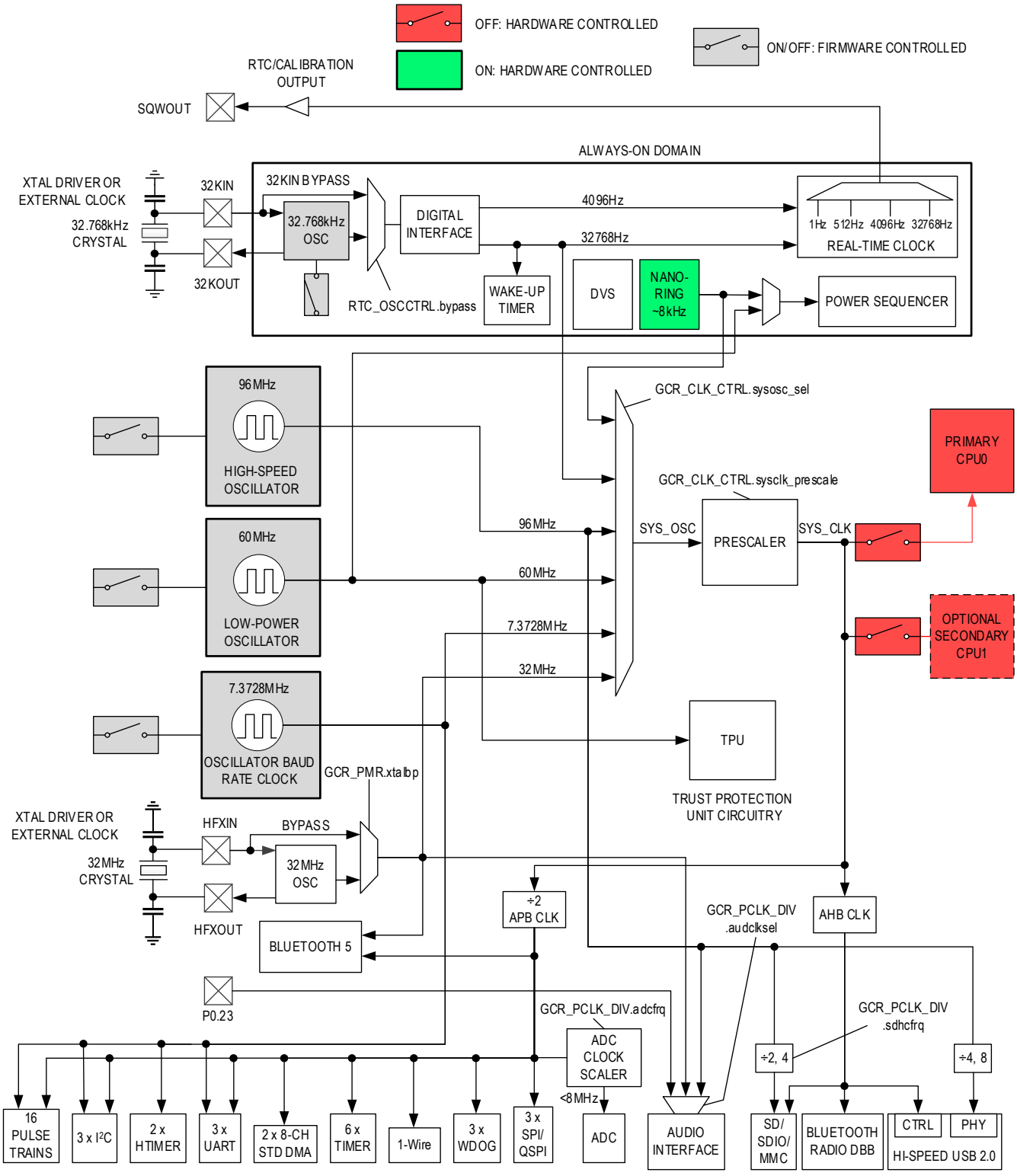
The device returns to ACTIVE mode from any internal or external interrupt.

The following pseudocode places the device in SLEEP mode:

```
SCR.sleepdeep = 0; // SLEEP mode enabled
WFI (or WFE);      // Enter the low power mode enabled by SCR.sleepdeep
```

Figure 4-4, below, shows the clocks available and blocks disabled during SLEEP mode.

Figure 4-4: SLEEP Mode Clock Control



4.2.3 DEEPSLEEP Low Power Mode

All internal clocks, except the 8kHz, are gated off. SYS_OSC is gated off, so the two main bus clocks PCLK and HCLK are inactive. The CPU state is retained. The 32kHz oscillator can be enabled via firmware.

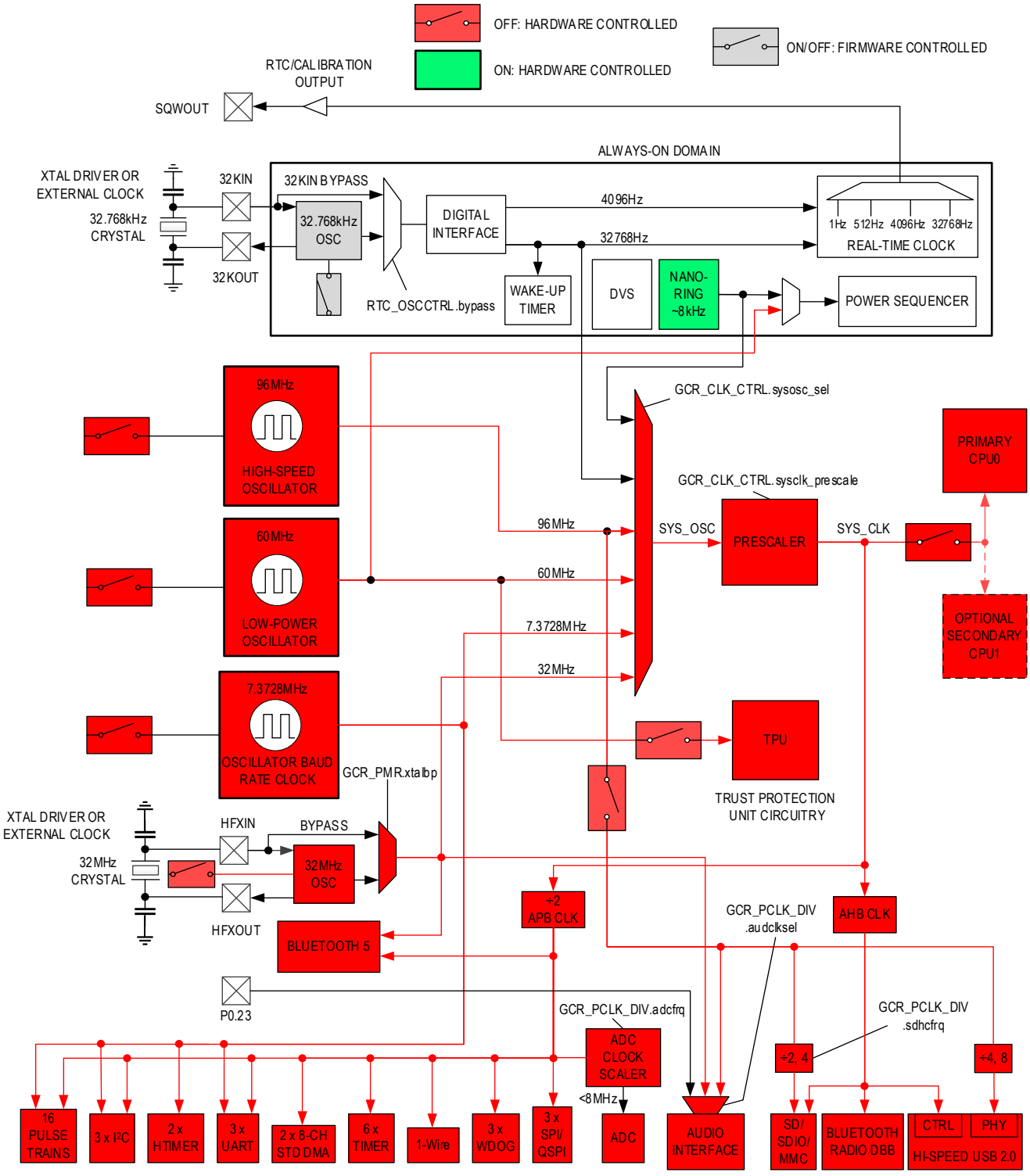
Because the main bus clocks are disabled, all peripherals are inactive except for the RTC which has its own independent oscillator. Only the RTC, USB wakeup or external interrupt can return the device to ACTIVE. The Watchdog Timers are inactive in this mode.

All internal register contents and all RAM contents are preserved. The GPIO pins retain their state in this mode.

To enter DEEPSLEEP mode,

```
SCR.sleepdeep = 1; // DEEPSLEEP mode enabled
WFI (or WFE);      // Enter DEEPSLEEP mode
```

Figure 4-5: DEEPSLEEP Clock Control



4.2.4 BACKUP Low Power Mode

This is the lowest power operating mode. All oscillators are disabled except for the 8kHz and the 32kHz oscillator. The 32kHz oscillator is firmware controlled. SYS_OSC is gated off, so PCLK and HCLK are inactive. The CPU state is not maintained.

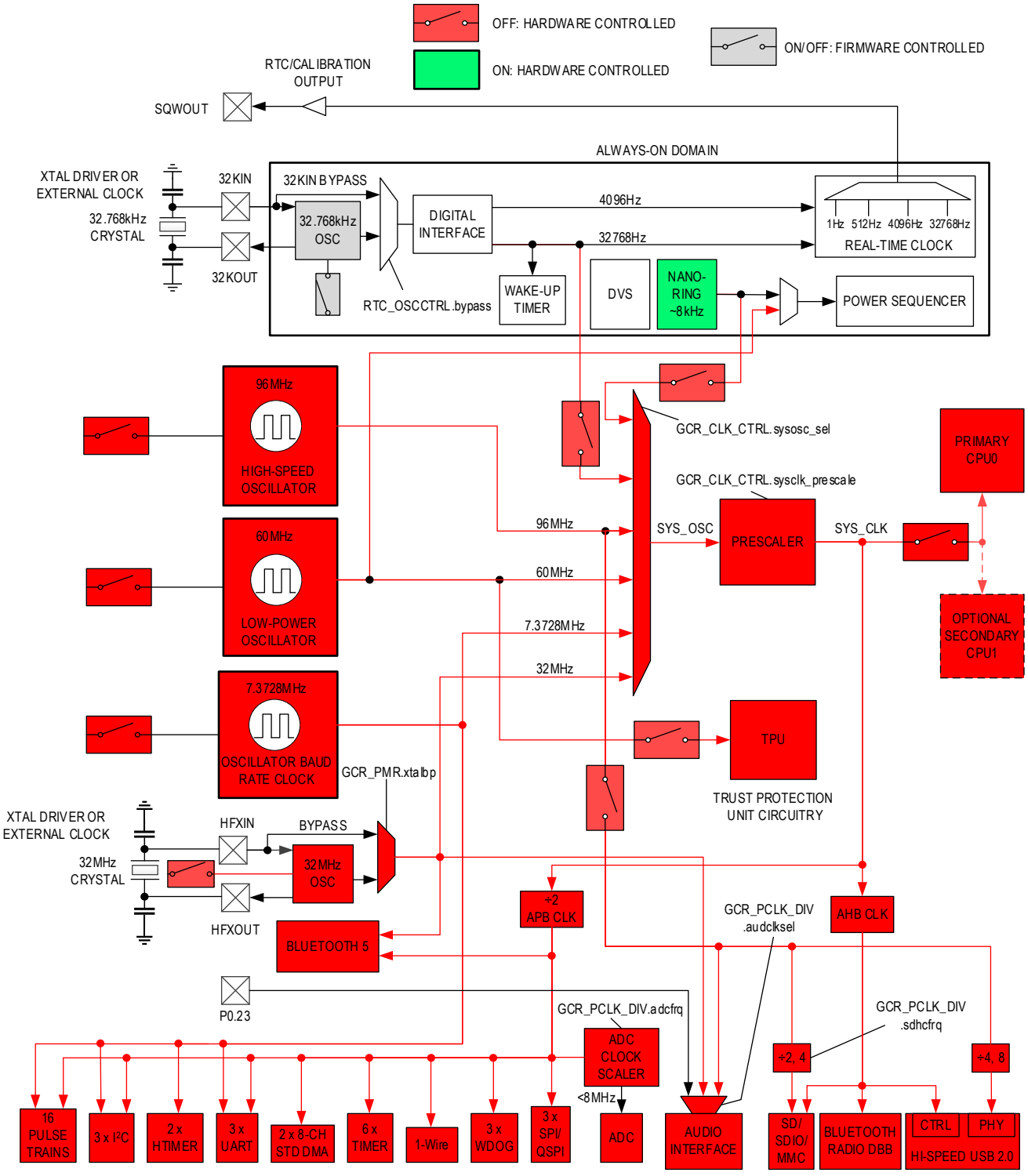
Only the RTC can operate in BACKUP mode. The AoD and RAM retention can optionally be set to automatically disable (and clear) themselves when entering this mode. RAM may be optionally retained. The amount of RAM retained is controlled by appropriately setting the *PWRSEQ_LPCN*.*ramret* register bits.

BACKUP mode supports the same wakeup sources as DEEPSLEEP mode.

To immediately enter BACKUP mode, write *GCR_PM*.*mode* = 0b100.

Figure 4-6, shows the clock control during BACKUP mode.

Figure 4-6: BACKUP Mode Clock Control



4.3 Device Resets

Multiple device resets are available – External Reset, Peripheral Reset, Soft Reset, System Reset, and Power-On Reset. On completion of any of the reset cycles, all peripherals are reset. On completion of any reset cycle HCLK and PCLK are

operational, the CPU core receives clocks and power, and the device is in ACTIVE mode. Program execution begins at the reset vector address.

Contents of the Always-On Domain (AoD) are reset only on power-cycling V_{DDA} and V_{COREA}.

Each of the on-chip peripherals can also be reset to their POR default state using the two reset registers *GCR_RSTR0* and *GCR_RSTR1*.

Table 4-2 shows the effects of the reset types and power modes.

Table 4-2: Reset and Low Power Mode Effects

	Peripheral Reset[4]	Soft Reset[4]	External/System Reset[4]	POR	ACTIVE Mode	SLEEP Mode	DEEPSLEEP Mode	BACKUP[3] Mode
GCR	-	-	Reset	Reset	R	-	-	-
8kHz Osc	On	On	On	On	On	On	On	On
32kHz Osc	-	-	-	Off	-	-	-	-
7.3728 MHz Osc	-	-	Off	Off	R	-	Off	Off
60MHz Osc	-	-	On[2]	On[2]	R	-	Off	Off
32MHz Osc	-	-	Off	Off	R	-	Off	Off
96MHz Osc	-	-	Off	Off	R	-	Off	Off
SYS_CLK	On	On	On[2]	On[2]	On	On	Off	Off
CPU0, CPU1 Clock	On	On	On	On	On	Off	Off	Off
RTC				Reset	FW	FW	FW	FW
V_{REGO_A}	On	On	On	On	On	On	On	On
V_{REGO_B}	On	On	On	On	On	On	On	FW
V_{REGO_C}	On	On	On	On	On	On	On	On
V_{REGO_D}	-	-	-	On	-	-	-	-
Bluetooth LDOs	-	-	-	Off	-	-	-	-
CPU	-	-	Reset	Reset	R	Off	Off	Off
RPU	-	-	Reset	Reset	R	-	-	-
WDT0/1/2	-	-	Reset	Reset	R	-	Off	Off
GPIO	-	Reset	Reset	Reset	R	-	-	-
Other Peripherals	Reset	Reset	Reset	Reset	R	-	Off	Off
Always-On Domain[1]	-	-	-	Reset	-	-	-	-
RAM Retention	-	-	-	Reset	-	-	On	FW

Table key:
FW = Controlled by firmware
On = Enabled by hardware (cannot be disabled)
Off = Disabled by hardware (cannot be enabled)
- = No Effect
R = Restored to previous ACTIVE mode setting when exiting DEEPSLEEP, restored to system reset state when exiting BACKUP

1: The always-on domain (AOD) is only reset on power-cycling V_{DDA} and V_{COREA}.

2: On a system reset or POR, the 60MHz oscillator will automatically be selected as SYS_OSC.

3: A system reset occurs when returning from BACKUP low-power mode.

4: Peripheral, soft, and system resets are initiated by firmware though the *GCR_RSTR0* register. System reset can also be triggered by the RSTN device (external reset) pin or watchdog reset.

4.3.1 Peripheral Reset

This resets all peripheral registers to their POR state unless otherwise noted. The CPU retains its state. The GPIO, watchdog timers, AoD, RAM retention, and general control registers (GCR), including the clock configuration, are unaffected.

To start a peripheral reset, set *GCR_RSTR0.prst* = 1. The reset will be completed immediately upon setting *GCR_RSTR0.prst* = 1.

4.3.2 Soft Reset

This is the same as a peripheral reset except that it also resets the GPIO to their POR state.

To start a soft reset, set *GCR_RSTR0.srst* = 1. The reset will be initiated immediately upon setting *GCR_RSTR0.srst* = 1.

4.3.3 System Reset, External Reset

This is the same as soft reset except it also resets all GCR, resetting the clocks to their default state. The CPU state is reset as well as the watchdog timers. The AoD and RAM are unaffected.

A watchdog timer reset event initiates a system reset. To start a system reset from firmware, set *GCR_RSTR0.system* = 1. Assertion of the RSTN device pin, which is an external reset, also initiates a system reset.

4.3.4 Power-On Reset

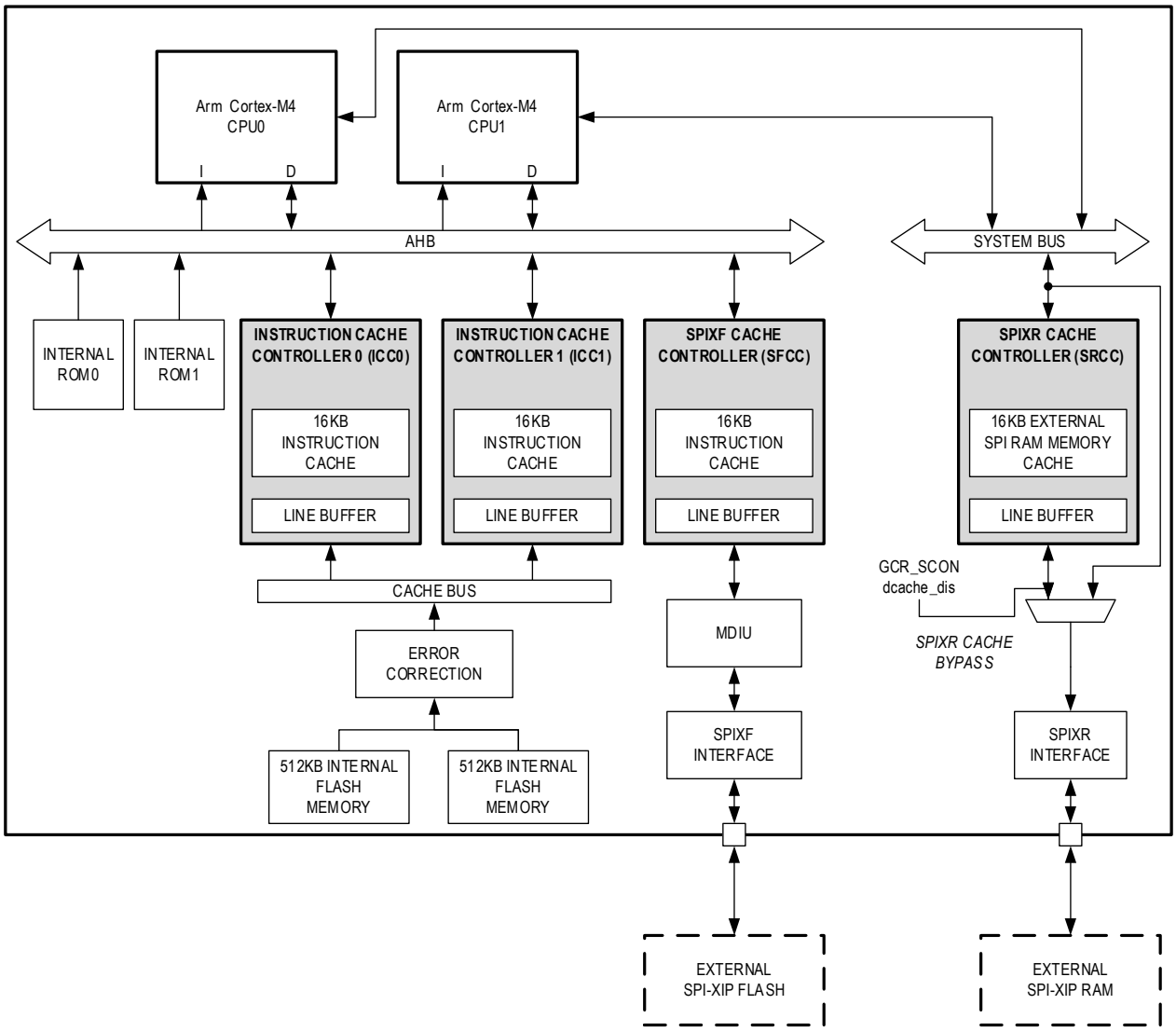
A POR resets everything in the device to its default state.

4.4 Cache

Each of the four cache controllers are independently managed. *Figure 4-7* shows the four cache controllers and their memory interfaces. Instruction Cache Controller 0 (ICC0 dedicated to CPU0) and Instruction Cache Controller 1 (ICC1 dedicated to CPU1) and the SPIXF Cache Controller (SFCC) are used for instruction caching only. ICC0 and ICC1 interfaces to the internal 1MB flash and SFCC interfaces to an external SPI flash device for external code execution. The SPIXR Data Cache Controller (SRCC) is used for data and instruction caching for external SPI SRAM memories. The SRCC is implemented as a write-through cache.

All four caches are managed separately using their specific cache controller, ICC0, ICC1, SFCC, SRCC. Each controller can be enabled, disabled, and invalidated. Each cache clock can be disabled by placing it in LIGHTSLEEP.

Figure 4-7: MAX32665/MAX32666 Cache Controllers Control



4.5 Instruction Cache Controller

ICC0, ICC1, and SFCC are independent cache controllers and each is controlled directly using their respective register set.

4.5.1 Enabling ICC0/ICC1/SFCC

Perform the following steps to enable ICC0:

1. Set *PWRSEQ_LPMEMSD.icachesd* to 0 to ensure the cache power is on.
2. Touch *ICCn_INVALIDATE.invalid*. Any write to this field invalidates the cache.
3. Read *ICCn_CACHE_CTRL.rdy* until it returns 1.
4. Set *ICCn_CACHE_CTRL.en* to 1.
5. Read *ICCn_CACHE_CTRL.rdy* until it returns 1.

Perform the following steps to enable ICC1:

1. Set *PWRSEQ_LPMEMSD.ic1sd* to 0 to ensure the cache power is on.
2. Touch *ICCn_INVALIDATE.invalid*. Any write to this field invalidates the cache.
3. Read *ICCn_CACHE_CTRL.rdy* until it returns 1.
4. Set *ICCn_CACHE_CTRL.en* to 1.
5. Read *ICCn_CACHE_CTRL.rdy* until it returns 1.

Perform the following steps to enable SFCC:

1. Set *PWRSEQ_LPMEMSD.icachexipsd* to 0 to ensure the cache power is on.
2. Touch *SFCC_INVALIDATE.invalid*. Any write to this field invalidates the cache.
3. Read *SFCC_CACHE_CTRL.rdy* until it returns 1.
4. Set *SFCC_CACHE_CTRL.enable* to 1.
5. Read *SFCC_CACHE_CTRL.rdy* until it returns 1.

4.5.2 Flushing the ICC0/ICC1/SFCC Cache

The System Configuration Register (*GCR_SCON*) includes a field for flushing these caches simultaneously. Setting *GCR_SCON.ccache_flush* to 1 performs a flush of all three caches. Flush only one of the caches by invalidating the cache contents. Setting the *ICCn_INVALIDATE* register to 1 invalidates the respective cache and forces a cache flush. Read the *ICCn_CACHE_CTRL.rdy* field until it returns 1 to determine when the flush is completed.

4.5.3 Flushing SRCC Cache

The System Configuration Register (*GCR_SCON*) includes a field for flushing these caches. Setting *GCR_SCON*.dcache_flush to 1 performs a flush of the cache.

4.6 Instruction Cache Controller Registers

See *Table 3-3* for the ICC0, ICC1, and SFCC, Peripheral Base Addresses.

Table 4-3: Instruction Cache Controller Register Summary

Offset	Register	Description
[0x0000]	*ICCn_CACHE_ID*	Cache ID Register
[0x0004]	*ICCn_MEMCFG*	Cache Memory Size Register
[0x0100]	*ICCn_CACHE_CTRL*	Instruction Cache Control Register
[0x0700]	*ICCn_INVALIDATE*	Instruction Cache Controller Invalidate Register

Table 4-4: SPIXF Cache Controller Register Summary

Offset	Register	Description
[0x0000]	*SFCC_CACHE_ID*	SPIXF Cache ID Register
[0x0004]	*SFCC_MEMCFG*	SPIXF Cache Memory Size Register
[0x0100]	*SFCC_CACHE_CTRL*	SPIXF Cache Control Register
[0x0700]	*SFCC_INVALIDATE*	SPIXF Cache Controller Invalidate Register

Table 4-5: ICCn Cache ID Register

ICCn Cache ID			ICCn_CACHE_ID	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	-	**Reserved** Do not modify this field.
15:10	cchid	R	-	**Cache ID** Returns the Cache ID for this Cache instance.
9:6	partnum	R	-	**Cache Part Number** Returns the part number indicator for this Cache instance.
5:0	relnum	R	-	**Cache Release Number** Returns the release number for this Cache instance.

Table 4-6: ICCn Memory Size Register

ICCn Memory Size			ICCn_MEMCFG	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	memsz	R	-	**Addressable Memory Size** Indicates the size of addressable memory by this cache controller instance in 128KB units.
15:0	cchsz	R	-	**Cache Size** Returns the size of the cache RAM memory in 1KB units. 16: 16KB Cache RAM

Table 4-7: ICCn Cache Control Register

ICCn Cache Control			ICCn_CACHE_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	R/W	-	**Reserved** Do not modify this field.
16	rdy	R	-	**Ready** This field is cleared by hardware anytime the cache as a whole is invalidated (including a Power On Reset event). Hardware automatically sets this field to 1 when the invalidate operation is complete and the cache is ready. 0: Cache Invalidate in process. 1: Cache is ready. *Note: While this field reads 0, the cache is bypassed and reads come directly from the line fill buffer.*
15:1	-	R/W	-	**Reserved** Do not modify this field.
0	en	R/W	0	**Enable** Set this field to 1 to enable the cache. Setting this field to 0 automatically invalidates the cache contents. When this cache is disabled, reads are handled by the line fill buffer. 0: Disable cache 1: Enable cache

Table 4-8: ICCn Invalidate Register

ICCn Invalidate			ICCn_INVALIDATE	[0x0700]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	invalid	WO	-	**Invalidate** Any write to this register of any value invalidates the cache.

Table 4-9: SFCC Cache ID Register

SFCC Cache ID			SFCC_CACHE_ID	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	-	**Reserved** Do not modify this field.
15:10	cchid	R	-	**Cache ID** Returns the Cache ID for this Cache instance.
9:6	partnum	R	-	**Cache Part Number** Returns the part number indicator for this Cache instance.
5:0	relnum	R	-	**Cache Release Number** Returns the release number for this Cache instance.

Table 4-10: SFCC Memory Size Register

SFCC Memory Size			SFCC_MEMCFG	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	memsz	R	-	**Addressable Memory Size** Indicates the size of addressable memory by this cache controller instance in 128KB units.
15:0	cchsz	R	-	**Cache Size** Returns the size of the cache RAM memory in 1KB units. 16: 16KB Cache RAM

Table 4-11: SFCC Cache Control Register

SFCC Cache Control			SFCC_CACHE_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	R/W	-	**Reserved** Do not modify this field.
16	rdy	R	-	**Ready** This field is cleared by hardware anytime the cache as a whole is invalidated (including a Power On Reset event). Hardware automatically sets this field to 1 when the invalidate operation is complete and the cache is ready. 0: Cache Invalidate in process. 1: Cache is ready. *Note: While this field reads 0, the cache is bypassed and reads come directly from the line fill buffer.*
15:1	-	R/W	-	**Reserved** Do not modify this field.

SFCC Cache Control			SFCC_CACHE_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
0	en	R/W	0	**Enable** Set this field to 1 to enable the cache. Setting this field to 0 automatically invalidates the cache contents. When this cache is disabled, reads are handled by the line fill buffer. 0: Disable cache 1: Enable cache

Table 4-12: SFCC Invalidate Register

SFCC Invalidate			SFCC_INVALIDATE	[0x0700]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	invalid	WO	-	**Invalidate** Any write to this register of any value invalidates the cache.

4.7 External RAM SPIXR Cache Controller (SRCC)

See for detailed usage information for the SRCC and the SRCC register interface.

4.8 RAM Memory Management

This device has many features for managing the on-chip RAM. The on-chip RAM includes data RAM, instruction and data caches, and peripheral FIFOs.

4.8.1 RAM Zeroization

The GCR Memory Zeroize Register, *GCR_MEMZCN*, allows clearing memory for firmware or security reasons. Zeroization writes all zeros to the specified memory.

The following RAM memories can be zeroized:

- The Internal Data RAMs 0 through 6.
  - Each of the internal data RAM segments can be zeroized independently by setting the *GCR_MEMZCN.sram0z* through *GCR_MEMZCN.sram6z* fields to 1.
- The USB FIFO
  - Write 1 to *GCR_MEMZCN.usbfifoz*
- ICC0 16KB Cache
  - Write 1 to *GCR_MEMZCN.icachez*
- ICC1 16KB Cache
  - Write 1 to *GCR_MEMZCN.icache1z*
- SRCC Cache Tags
  - Write 1 to *GCR_MEMZCN.scachetagz*
  - This clears the Cache tags, ultimately invalidating the SRCC cache memory as well.
- SRCC 16KB Cache Data
  - Write 1 to *GCR_MEMZCN.scachedataz*
- SFCC 16KB Cache
  - Write 1 to *GCR_MEMZCN.icachexipz*
- Crypto MAA RAM (MAX32666)
  - Write 1 to *GCR_MEMZCN.cryptoz*

4.8.2 RAM Low Power Modes

RAM low power modes and shutdown are controlled on a bank basis. The System RAM banks are shown with corresponding bank sizes and base addresses in below:

Table 4-13: RAM Block Size and Base Address

System RAM Block #	Size (Words)	Base Address
sysram0	8K	2000-0000
sysram1	8K	2000-8000
sysram2	16K	2001-0000
sysram3	16K	2002-0000
sysram4	32K	2003-0000
sysram5	32K	2005-0000
sysram6	2K	2007-0000
sysram7	2K	2007-2000
sysram8	4K	2007-4000
sysram9	4K	2007-8000
sysram10	8K	2007-C000
sysram11	8K	2008-4000

4.8.2.1 RAM LIGHTSLEEP

RAM can be placed in a low power mode, referred to as LIGHTSLEEP, using the Memory Clock Control Register, *GCR_MEMCKCN*. LIGHTSLEEP gates off the clock to the RAM and makes the RAM unavailable for read/write operations, while memory contents are retained, reducing power consumption. LIGHTSLEEP is available for the 6 Data RAM blocks and the ECC RAM block, the USB FIFO, Crypto RAM, ICC0 RAM, ICC1 RAM, SFCC RAM and the SRCC RAM. RAM contents are available when exiting LIGHTSLEEP mode.

4.8.2.2 RAM Shut Down

RAM memories can individually be shut down further reducing the power consumption for the device. Shutting down a memory gates off the clock and removes power to the memory. Shutting down a memory invalidates (destroys) the contents of the memory and results in a POR of the memory when it is enabled. RAM memory shut down is configured using the *PWRSEQ_LPMEMSD* register.

4.9 Miscellaneous Control Registers

This set of control registers provides control for system related aspects such as ECC enable, Comparator enable, Square Wave Out enable, Power Down signaling enable, Power Control, Rest Pullup control, and SIMO Clock enable.

See *Table 3-3* for the the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 4-14: Miscellaneous Control Register Summary

Offset	Register	Description
[0x0000]	*MCR_ECCEN*	*Error Correction Coding Enable Register*
[0x0004]	*MCR_HIRC96M*	*96MHz High Frequency Clock Adjustment Register*
[0x0008]	*MCR_OUTEN*	*SQWOUT and PDOWN Enable Register*
[0x000C]	*MCR_AINCOMP*	*Comparator Enable Register*
[0x0010]	*MCR_CTRL*	*Control Register*

4.10 Miscellaneous Control Registers Details

Table 4-15: Error Correction Coding (ECC) Enable Register

Error Correction Coding Enable			MCR_ECCEN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:13	-	RO	0	**Reserved** Do not modify this field.
12	fl1eccen	R/W	0	**Flash1 ECC Enable** 0: Disable 1: Enable *Note: See Table 7-1: MAX32665/MAX32666 Internal Flash Memory Organization.*
11	fl0eccen	R/W	0	**Flash0 ECC Enable** 0: Disable 1: Enable *Note: See Table 7-1: MAX32665/MAX32666 Internal Flash Memory Organization.*
10	icxipeccen	R/W	0	**ICacheXIP ECC Enable** 0: Disable 1: Enable
9	ic1eccen	R/W	0	**ICache1 ECC Enable** 0: Disable 1: Enable
8	ic0eccen	R/W	0	**ICache0 ECC Enable** 0: Disable 1: Enable
7:6	-	RO	0	**Reserved** Do not modify this field.
5	sysram5eccen	R/W	0	**Sysram5 ECC Enable** 0: Disable 1: Enable
4	sysram4eccen	R/W	0	**Sysram4 ECC Enable** 0: Disable 1: Enable
3	sysram3eccen	R/W	0	**Sysram3 ECC Enable** 0: Disable 1: Enable
2	sysram2eccen	R/W	0	**Sysram2 ECC Enable** 0: Disable 1: Enable
1	sysram1eccen	R/W	0	**Sysram1 ECC Enable** 0: Disable 1: Enable
0	sysram0eccen	R/W	0	**Sysram0 ECC Enable** 0: Disable 1: Enable

Table 4-16: SQWOUT and PDOWN Output Enable Register

SQWOUT/PDOWN Output Enable			MCR_OUTEN	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved** Do not modify this field.
3	pdownout1en	R/W	0	**PDOWN Enable P0.26** 0: GPIO function on pin. 1: PDOWN signal on pin if *PWRSEQ_LPCN*.*pdowndslen* = 1.
2	pdownout0en	R/W	0	**PDOWN Enable P0.18** 0: GPIO function on pin. 1: PDOWN signal on pin if *PWRSEQ_LPCN*.*pdowndslen* = 1.
1	sqwout1en	R/W	0	**SQWOUT Enable P0.27** 0: GPIO function on pin. 1: SQWOUT signal on pin.
0	sqwout0en	R/W	0	**SQWOUT Enable P0.19** 0: GPIO function on pin. 1: SQWOUT signal on pin.

Table 4-17: Comparator Enable Register

Comparator Enable			MCR_AINCOMP	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved** Do not modify this field.
5:4	aincomphyst	R/W	0	**Comparator Hysteresis Control** Refer to the V_{HYST} specification in the data sheet electrical characteristics for the values corresponding to this field value.
3	aincomp3pd	R/W	1	**Comparator 3 Disable** 0:Comparator is powered and enabled 1:Comparator is powered down and disabled
2	aincomp2pd	R/W	1	**Comparator 2 Disable** 0:Comparator is powered and enabled 1:Comparator is powered down and disabled
1	aincomp1pd	R/W	1	**Comparator 1 Disable** 0:Comparator is powered and enabled 1:Comparator is powered down and disabled
0	aincomp0pd	R/W	1	**Comparator 0 Disable** 0:Comparator is powered and enabled 1:Comparator is powered down and disabled

Table 4-18: Control Register

Control			MCR_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved** Do not modify this field.

Control			MCR_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
10	rstn_voltage_sel	R/W	0	**RSTN Voltage Supply Select** This bit setting determines which supply voltage drives the RSTN device pin. 0: V_{DDIO} 1: V_{DDIOH} *Note: This bit is only affected by a Power On Reset event. RSTN assertion has no affect on this bit setting.*
9	p1m	R/W	0	**RSTN Device Pin Internal Pullup** This pin controls the internal pullup value connected to the RSTN device pin. 0: 25kΩ 1: 1MΩ
8	buckclkscalen	R/W	0	**SIMO Dynamic Clock Scaling Enable** Allows the dynamic scaling of the SIMO clock as part of the Dynamic Voltage Scaling operation. 0: Disabled 1: Enabled
7:4	-	RO	0	**Reserved** Do not modify this field.
3	usbswen_n	R/W	0	**USB PHY Power Gate Control** This bit is sampled when entering DEEPSLEEP or BACKUP mode. 0: USB Switch On 1: USB Switch Off
2:1	vddcsw	R/W	01	**V_{COREB} Switch** VCOREB can be operated at a lower voltage to minimize leakage in any of the low power modes SLEEP, DEEPSLEEP, and BACKUP. Allows the CPU cores to operate from VCOREA during these low power modes. 0b00: Reserved for Future Use 0b01: CPU0, CPU1 operate from VCOREA in low power mode. When VCOREB reaches operating voltage during exit from low power mode, firmware sets this and the device switches to operate the cores from VCOREB. 0b10: CPU0, CPU1 operate from VCOREA in low power mode. When VCOREB is less than VCOREA, firmware sets this and the device switches back to a low power mode 0b11: Automatically set by hardware to swicth the CPU0, CPU1 cores to VCOREB.
0	vddcswen	R/W	0	**V_{COREB} Switch Enable** 0: Disabled 1: Allows exit from low power mode according to the setting of *MCR_CTRL.vddcsw*

4.11 Single Inductor Multiple Output (SIMO) Power Supply

The Single Inductor Multiple Output (SIMO) switch mode power supply allows the device to operate autonomously from a single lithium cell. The SIMO provides four buck switching regulators (V_{REGO_A} thru V_{REGO_D}). Each of the four regulator voltages can be controlled by the CPU individually. For the SIMO top operate properly, the four buck regulator outputs must drive the power supply pins of the device as follows in *Table 4-19.*

4.11.1 Power Supply Monitor

The system also provides a power monitor that monitors the external power supplies relative to the on-chip bandgap voltage. The following power supplies are monitored:

- VCOREA (V_{COREA}) Digital Core Supply Voltage A for the Always-On Domain
- VCOREB (V_{COREB}) Digital Core Supply Voltage B
- VDDIO (V_{DDIO}) GPIO Supply Voltage
- VDDIOH (V_{DDIOH}) GPIO High Supply Voltage
- VDDA (V_{DDA}) AOD Analog Supply Voltage
- VREGI (V_{REGI}) Input Supply Voltage, Battery
- VDDB (V_{DDB}) USB Supply Voltage
- VTXOUT (V_{TXOUT}) Bluetooth Transmitter Supply Voltage Output
- VRXOUT (V_{RXOOUT}) Bluetooth Receiver Supply Voltage Output

Each of the power supply monitors' settings is found in the Low Power Control register *PWRSEQ_LPCN*. When the corresponding power monitor is enabled, the input voltage pin is constantly monitored. If the voltage drops below the trigger threshold, all registers and peripherals in that power domain are reset. This improves reliability and safety by guarding against a low voltage condition corrupting the contents of the registers and the device state.

Refer to the data sheet Electrical Characteristics table for the trigger threshold values and power-fail reset voltages.

Table 4-19: SIMO Power Supply Device Pin Connectivity

SIMO Supply Output Pin	Connection	Device Power Supply Input Pin	Supply Monitor Reset Action
V_{REGO_A}	→	V_{DDA}	Domain reset
V_{REGO_B}	→	V_{COREB}	Domain Reset
V_{REGO_C}	→	V_{COREA}	Domain Reset
V_{REGO_D}	→	V_{RXIN}, V_{TXIN}	-
-	-	V_{REGI}	Domain Reset
-	-	V_{TXOUT}	Bluetooth transmitter POR
-	-	V_{RXOUT}	Bluetooth receiver POR
-	-	V_{DDB}	USB peripheral reset
-	-	V_{DDIO} Power On	GPIO pad held in reset until the voltage rises above V_{RST}
-	-	V_{DDIOH} Power On	GPIO pad held in reset until the voltage rises above V_{RST}.
-	-	V_{DDIO}	GPIO pad logic enters POR.
-	-	V_{DDIOH}	GPIO pad logic enters POR.

4.12 Single Inductor Multiple Output (SIMO) Registers

See *Table 3-3* for the the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 4-20: SIMO Controller Register Summary

Offset	Register	Description
[0x0004]	*VREGO_A*	*Buck Voltage Regulator A Control Register*
[0x0008]	*VREGO_B*	*Buck Voltage Regulator B Control Register*
[0x000C]	*VREGO_C*	*Buck Voltage Regulator C Control Register*
[0x0010]	*VREGO_D*	*Buck Voltage Regulator D Control Register*
[0x0014]	*IPKA*	***Reserved*** *Reserved. Do not modify this field.*
[0x0018]	*IPKB*	***Reserved*** *Reserved. Do not modify this field.*
[0x001C]	*MAXTON*	***Reserved*** *Reserved. Do not modify this field.*
[0x0020]	*ILOAD_A*	***Reserved*** *Reserved. Do not modify this field.*
[0x0024]	*ILOAD_B*	***Reserved*** *Reserved. Do not modify this field.*
[0x0028]	*ILOAD_C*	***Reserved*** *Reserved. Do not modify this field.*
[0x002C]	*ILOAD_D*	***Reserved*** *Reserved. Do not modify this field.*
[0x0030]	*BUCK_ALERT_THR_A*	***Reserved*** *Reserved. Do not modify this field.*
[0x0034]	*BUCK_ALERT_THR_B*	***Reserved*** *Reserved. Do not modify this field.*
[0x0038]	*BUCK_ALERT_THR_C*	***Reserved*** *Reserved. Do not modify this field.*
[0x003C]	*BUCK_ALERT_THR_D*	***Reserved*** *Reserved. Do not modify this field.*
[0x0040]	*BUCK_OUT_READY*	*Buck Regulator Output Ready Register*
[0x0044]	*ZERO_CROSS_CAL_A*	***Reserved*** *Reserved. Do not modify this field.*
[0x0048]	*ZERO_CROSS_CAL_B*	***Reserved*** *Reserved. Do not modify this field.*
[0x004C]	*ZERO_CROSS_CAL_C*	***Reserved*** *Reserved. Do not modify this field.*
[0x0050]	*ZERO_CROSS_CAL_D*	***Reserved*** *Reserved. Do not modify this field.*

4.13 Single Inductor Multiple Output (SIMO) Registers Details

Table 4-21: Buck Voltage Regulator A Control Register

Buck Voltage Regulator A Control			VREGO_A	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	-	**Reserved** Do not modify this field.
7	rangea	R/W	1	**Regulator Output Range** 0: 0.5V to 1.77 1: 0.6V to 1.87V
6:0	vseta	R/W	0x78h	**Regulator Output Voltage** Each increment in the register represents 10mV. rangea = 1: $Output\ Voltage = 0.6V + (10mV \times vseta)$ rangea = 0: $Output\ Voltage = 0.5V + (10mV \times vseta)$ Default: 0x78h = 1.7V when rangea = 0; 1.8V when rangea = 1 *Warning: When this regulator is connected as shown in SIMO Power Supply Device Pin Connectivity:* *A: The maximum setting for this regulator must be followed for V_{DDA} as indicated in the device data sheet.* *B: Setting the regulator to a voltage below the Power-Fail Reset Voltage for V_{DDA} will initiate the Power Monitor Reset Action.*

Table 4-22: Buck Voltage Regulator B Control Register

Buck Voltage Regulator A Control			VREGO_B	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	-	**Reserved** Do not modify this field.
7	rangeb	R/W	0	**Regulator Output Range** 0: 0.5V to 1.77 1: 0.6V to 1.87V
6:0	vsetb	R/W	0x32h	**Regulator Output Voltage** Each increment in the register represents 10mV. rangeb = 1; $Output\ Voltage = 0.6V + (10mV \times vsetb)$ rangeb = 0; $Output\ Voltage = 0.5V + (10mV \times vsetb)$ 0x7Fh: 1.77V when rangeb = 0; 1.87V when rangeb = 1 Default: 0x32h = 1.0V when rangeb = 0; 1.1V when rangeb = 1 *Warning: When this regulator is connected as shown in SIMO Power Supply Device Pin Connectivity:* *A: The maximum setting for this regulator must be followed for V_{COREB} as indicated in the device data sheet.* *B: Setting the regulator to a voltage below the Power-Fail Reset Voltage for V_{COREB} will initiate the Power Monitor Reset Action.*

Table 4-23: Buck Voltage Regulator C Control Register

Buck Voltage Regulator A Control			VREGO_C	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	-	**Reserved** Do not modify this field.
7	rangec	R/W	0	**Regulator Output Range** The voltage regulator output range setting. 0: 0.5V to 1.77 1: 0.6V to 1.87V
6:0	vsetc	R/W	0x32h	**Regulator Output Voltage** Each increment in the register represents 10mV. rangec = 1; $Output\ Voltage = 0.6V + (10mV \times vsetc)$ rangec = 0; $Output\ Voltage = 0.5V + (10mV \times vsetc)$ 0x7Fh: 1.77V when rangec = 0; 1.87V when rangec = 1 Default: 0x32h = 1.0V when rangec = 0; 1.1V when rangec = 1 *Warning: When this regulator is connected as shown in SIMO Power Supply Device Pin Connectivity:* *A: The maximum setting for this regulator must be followed for V_{COREA} as indicated in the device data sheet.* *B: Setting the regulator to a voltage below the Power-Fail Reset Voltage for V_{COREA} will initiate the Power Monitor Reset Action.*

Table 4-24: Buck Voltage Regulator D Control Register

Buck Voltage Regulator D Control			VREGO_D	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	-	**Reserved** Do not modify this field.
7	ranged	R/W	0	**Regulator Output Range** 0: 0.5V to 1.77 1: 0.6V to 1.87V
6:0	vsetd	R/W	0x32h	**Regulator Output Voltage** Each increment in the register represents 10mV. ranged = 1; $Output\ Voltage = 0.6V + (10mV \times vsetd)$ ranged = 0; $Output\ Voltage = 0.5V + (10mV \times vsetd)$ 0x7Fh: 1.77V when ranged = 0; 1.87V when ranged = 1 Default: 0x32h = 1.0V when ranged = 0; 1.1V when ranged = 1 *Warning: When this regulator is connected as shown in SIMO Power Supply Device Pin Connectivity:* *A: The maximum setting for this regulator must be followed for V_{RXIN} and V_{TXIN} as indicated in the device data sheet.* *B: Setting the regulator to a voltage below the Power-Fail Reset Voltage for V_{RXIN} and/or V_{TXIN} will initiate the Power Monitor Reset Action.*

Table 4-25: High Side FET Peak Current VREGO_A VREGO_B Register

High Side FET Peak Current VREGO_A VREGO_B			IPKA	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	-	**Reserved** Reserved. Do not modify this field.
7:4	ipksetb	R/W	0x8h	**Reserved** Reserved. Do not modify this field.
3:0	ipkseta	R/W	0x8h	**Reserved** Reserved. Do not modify this field.

Table 4-26: High Side FET Peak Current VREGO_C VREGO_D Register

High Side FET Peak Current VREGO_C VREGO_D			IPKB	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	-	**Reserved** Do not modify this field.
7:4	ipksetd	R/W	0x8h	**Reserved** Reserved. Do not modify this field.
3:0	ipksetc	R/W	0x8h	**Reserved** Reserved. Do not modify this field.

Table 4-27: Maximum High Side FET Time On Register

Maximum High Side FET Time On			MAXTON	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	R/W	-	**Reserved** Do not modify this field.
3:0	tonset	R/W	0x8h	**Reserved** Reserved. Do not modify this field.

Table 4-28: Buck Cycle Count VREGO_A Register

Buck Cycle Count VREGO_A			ILOAD_A	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved** Do not modify this field.
7:0	iloada	RO	0	**Reserved** Reserved. Do not modify this field.

Table 4-29: Buck Cycle Count VREGO_B Register

Buck Cycle Count VREGO_B			ILOAD_B	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved** Do not modify this field.
7:0	iloadb	RO	0	**Reserved** Reserved. Do not modify this field.

Table 4-30: Buck Cycle Count VREGO_C Register

Buck Cycle Count VREGO_C			ILOAD_C	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved** Do not modify this field.
7:0	iloadc	RO	0	**Reserved** Reserved. Do not modify this field.

Table 4-31: Buck Cycle Count VREGO_D Register

Buck Cycle Count VREGO_D			ILOAD_D	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved** Do not modify this field.
7:0	iloadd	RO	0	**Reserved** Reserved. Do not modify this field.

Table 4-32: Buck Cycle Count Alert VREGO_A Register

Buck Cycle Count Alert VREGO_A			BUCK_ALERT_THR_A	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved** Do not modify this field.
7:0	buckthra	R/W	0	**Reserved** Reserved. Do not modify this field.

Table 4-33: Buck Cycle Count Alert VREGO_B Register

Buck Cycle Count Alert VREGO_A			BUCK_ALERT_THR_B	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved** Do not modify this field.
7:0	buckthrb	R/W	0	**Reserved** Reserved. Do not modify this field.

Table 4-34: Buck Cycle Count Alert VREGO_C Register

Buck Cycle Count Alert VREGO_A			BUCK_ALERT_THR_C	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved** Do not modify this field.
7:0	buckthrc	R/W	0	**Reserved** Reserved. Do not modify this field.

Table 4-35: Buck Cycle Count Alert VREGO_D Register

Buck Cycle Count Alert VREGO_D			BUCK_ALERT_THR_D	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved** Do not modify this field.
7:0	buckthrd	R/W	0	**Reserved** Reserved. Do not modify this field.

Table 4-36: Buck Regulator Output Ready Register

Buck Regulator Output Ready			BUCK_OUT_READY	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	-	**Reserved** Do not modify this field.
3	buckoutrdyd	R	0	**VREGO_D Output Ready** When *VREGO_D.vsetd* changes, this bit will be set when the output voltage has reached its regulated value. It will not be cleared if the output voltage drops below its set value. 0: Not ready 1: Ready
2	buckoutrdyc	R	0	**VREGO_C Output Ready** When *VREGO_C.vsetc* changes, this bit will be set when the output voltage has reached its regulated value. It will not be cleared if the output voltage drops below its set value. 0: Not ready 1: Ready
1	buckoutrdyb	R	0	**VREGO_B Output Ready** When *VREGO_B.vsetb* changes, this bit will be set when the output voltage has reached its regulated value. It will not be cleared if the output voltage drops below its set value. 0: Not ready 1: Ready
0	buckoutrdya	R	0	**VREGO_A Output Ready** When *VREGO_A.vseta* changes, this bit will be set when the output voltage has reached its regulated value. It will not be cleared if the output voltage drops below its set value. 0: Not ready 1: Ready

Table 4-37: Zero Cross Calibration VREGO_A Register

Zero Cross Calibration VREGO_A			ZERO_CROSS_CAL_A	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	-	**Reserved** Do not modify this field.
4:0	zxcala	RO	0	**Reserved** Reserved. Do not modify this field.

Table 4-38: Zero Cross Calibration VREGO_B Register

Zero Cross Calibration VREGO_B			ZERO_CROSS_CAL_B	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	-	**Reserved** Do not modify this field.
4:0	zxcalb	RO	0	**Reserved** Reserved. Do not modify this field.

Table 4-39: Zero Cross Calibration VREGO_C Register

Zero Cross Calibration VREGO_C			ZERO_CROSS_CAL_C	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	-	**Reserved** Do not modify this field.
4:0	zxcalc	RO	0	**Reserved** Reserved. Do not modify this field.

Table 4-40: Zero Cross Calibration VREGO_D Register

Zero Cross Calibration VREGO_D			ZERO_CROSS_CAL_D	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	-	**Reserved** Do not modify this field.
4:0	zxcald	RO	0	**Reserved** Reserved. Do not modify this field.

4.14 Power Sequencer and Always-On Domain Registers

See *Table 3-3* for the the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 4-41: Power Sequencer and Always-On Domain Register Summary

Offset	Register	Description
[0x0000]	*PWRSEQ_LPCN*	*Low Power Control Register*
[0x0004]	*PWRSEQ_LPWKST0*	*GPIO0 Low Power Wakeup Status Flags*

Offset	Register	Description
[0x0008]	*PWRSEQ_LPWKEN0*	*GPIO0 Low Power Wakeup Enable Register*
[0x000C]	*PWRSEQ_LPWKST1*	*GPIO1 Low Power Wakeup Status Flags*
[0x0010]	*PWRSEQ_LPWKEN1*	*GPIO1 Low Power Wakeup Enable Register*
[0x0030]	*PWRSEQ_LPPWST*	*Peripheral Low Power Wakeup Status Flags*
[0x0034]	*PWRSEQ_LPPWEN*	*Peripheral Low Power Wakeup Enable Register*
[0x0040]	*PWRSEQ_LPMEMSD*	*RAM Shutdown Control Register*
[0x0044]	*PWRSEQ_LPVDDPD*	*VDD Low Power Domain Control Register*
[0x0048]	*PWRSEQ_BURET*	*BACKUP Return Vector Register*
[0x004C]	*PWRSEQ_BUAOD*	*BACKUP AoD Register*

4.15 Power Sequencer and Always-On Domain Register Details

Table 4-42: Low Power Control Register

Low Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31	-	RO	0	**Reserved** Do not modify this field.
30	pdowndslen	R/W	0	**PDOWN DEEPSLEEP Output Enable** 0: Disabled 1: PDOWN signal is asserted when device enters DEEPSLEEP
29	vtxoutmd	R/W	0	**VTXOUT (V_{TXOUT}) Bluetooth Transmitter Supply Power Monitor Disable** Reserved. Do not modify this field.
28	vrxoutmd	R/W	0	**VRXOUT (V_{RXOUT}) Bluetooth Receiver Supply Power Monitor Disable** Reserved. Do not modify this field.
27	vddbmd	R/W	0	**VDDB (V_{DDB}) USB Supply Power Monitor Disable** Reserved. Do not modify this field.
26	porvddiohmd	R/W	0	**VDDIOH (V_{DDIOH}) GPIO Supply Power On Reset Monitor Disable** Reserved. Do not modify this field.
25	porvddiomd	R/W	0	**VDDIO (V_{DDIO}) GPIO Supply Power On Reset Monitor Disable** Reserved. Do not modify this field.
24	vddiohmd	R/W	0	**VDDIOH (V_{DDIOH}) GPIO Supply Power Fail Monitor Disable** Reserved. Do not modify this field.
23	vddiomd	R/W	0	**VDDIO (V_{DDIO}) GPIO Supply Power Fail Monitor Disable** Reserved. Do not modify this field.
22	vddamd	R/W	0	**VDDA (V_{DDA}) Analog Supply Power Monitor Disable** Reserved. Do not modify this field.
21	vregimd	R/W	0	**VREGI (V_{REGI})Power Monitor Disable** Reserved. Do not modify this field.

Low Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
20	vcoremd	R/W	0	**VCOREA and VCOREB Supply Power Monitor Disable** Do not modify this field.
19:12	-	RO	0	**Reserved** Do not modify this field.
11	bgoff	R/W	1	***DEEPSLEEP* Low Power Mode and *BACKUP* Low Power Mode Modes Bandgap Off** 0: System Bandgap is always on 1: System Bandgap is off in DEEPSLEEP and BACKUP modes
10	fwkm	R	0	***DEEPSLEEP* Low Power Mode Mode Fast Wakeup Enable** When enabled, the system exits DEEPSLEEP mode faster by: • Bypassing the INRO warmup • Reducing the warmup time for the IPO • Reducing the warmup time for the LDO Code execution resumes at the next instruction after the entry to *DEEPSLEEP* Low Power Mode. When disabled, the system wakes up from *DEEPSLEEP* Low Power Mode as if a system reset occurred (code execution starts at the reset vector). 0: Disabled 1: Enabled
9	bkgrnd	R/W	0	***BACKGROUND* Mode Enable** Reserved. Do not modify this field.
8:2	-	RO	0	**Reserved** Do not modify this field.
1:0	ramret	R/W	0b00	***BACKUP* Low Power Mode Mode Data RAM Retention** 0b00: RAM retention in BACKUP mode disable *0b01: Reserved* *0b10: Reserved* 0b11: Sysram2 thru Sysram 11 retained

Table 4-43: GPIO0 Low Power Wakeup Status Flags

GPIO0 Low Power Wakeup Status Flags			PWRSEQ_LPWKST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	wakest	R/W1C	0	**GPIO0 Pin Wakeup Status Flag** Whenever a GPIO0 pin, in any power mode, transitions from low-to-high or high-to-low, the corresponding bit in this register is set. The device will transition from a low-power to ACTIVE mode if the corresponding interrupt enable bit is set in *PWRSEQ_LPWKEN0*. This register should be cleared before entering any low power mode.

Table 4-44: GPIO0 Low Power Wakeup Enable Registers

GPIO0 Low Power Wakeup Enable			PWRSEQ_LPWKEN0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**

31:0	wakeen	R/W	0	**GPIO0 Pin Wakeup Interrupt Enable** Setting a bit in this register will cause an interrupt be generated and will wakeup the device from any low power mode to ACTIVE mode if the corresponding bit in the PWRSEQ_LPWKST0 register is set. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the device to wakeup from a low power mode on a GPIO pin transition, first set the "GPIO Wakeup enable" register bit GCR_PM.gpiowken = 1.*

Table 4-45: GPIO1 Low Power Wakeup Status Flags

GPIO1 Low Power Wakeup Status Flags			**PWRSEQ_LPWKST1**	**[0x000C]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:18		RO	0	**Reserved** Do not modify this field.
17:0	wakest	R/W1C	0	**GPIO1 Pin Wakeup Status Flag** Whenever a GPIO1 pin, in any power mode, transitions from low-to-high or high-to-low, the corresponding bit in this register is set. Bits corresponding to unimplemented GPIO are ignored. *Note: The device will transition from a low-power to ACTIVE mode if the corresponding interrupt enable bit is set in PWRSEQ_LPWKEN1. This register should be cleared before entering any low power mode. Bit 16 reflects the status of the SWDIO pin when it is configured as GPIO. Bit 17 reflects the status of the SWCLK pin when it is configured as GPIO.*

Table 4-46: GPIO1 Low Power Wakeup Enable Registers

GPIO1 Low Power Wakeup Enable			**PWRSEQ_LPWKEN1**	**[0x0010]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:18		RO	0	**Reserved** Do not modify this field.
17:0	wakeen	R/W	0	**GPIO1 Pin Wakeup Interrupt Enable** Write 1 to any bit to enable the corresponding pin on the 32-bit GPIO port to generate an interrupt to wakeup the device from any low power mode to ACTIVE mode. A wakeup occurs on any low-to-high or high-to-low transition on the corresponding pin. *Note: To enable the device to wakeup from a low power mode on a GPIO pin transition, first set the "GPIO Wakeup enable" register bit GCR_PM.gpiowken = 1. Bit 16 reflects the enable of the SWDIO pin when it is configured as GPIO. Bit 17 reflects the enable of the SWCLK pin when it is configured as GPIO.*

Table 4-47: Peripheral Low Power Wakeup Status Flags

Peripheral Low Power Wakeup Status Flags			**PWRSEQ_LPPWST**	**[0x0030]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:18	-	RO	0	**Reserved** Do not modify this field.

Peripheral Low Power Wakeup Status Flags			PWRSEQ_LPPWST	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
17	rstwkst	R/W1C	0	**Reset Detect Wakeup Status Flag** Set when an external reset has caused a wakeup.
16	bbmodest	R/W1C	0	***BACKUP* Low Power Mode Mode Wakeup Status Flag** Set when the device wakes from BACKUP mode
15:8	-	RO	0	**Reserved** Do not modify this field.
11	aincomp3st	R	0	**Analog Input Comparator 3 Output Status Flag** The state of this bit reflects output of Analog Input Comparator 3 0: Comparator output is low. 1: Comparator output is high.
10	aincomp2st	R	0	**Analog Input Comparator 2 Output Status Flag** The state of this bit reflects output of Analog Input Comparator 2 0: Comparator output is low. 1: Comparator output is high.
9	aincomp1st	R	0	**Analog Input Comparator 1 Output Status Flag** The state of this bit reflects output of Analog Input Comparator 1 0: Comparator output is low. 1: Comparator output is high.
8	aincomp0st	R	0	**Analog Input Comparator 0 Output Status Flag** The state of this bit reflects output of Analog Input Comparator 0 0: Comparator output is low. 1: Comparator output is high.
7	aincomp3wkst	R/W1C	0	**Analog Input Comparator 3 Wakeup Status Flag** This bit is set when the comparator inputs detect an event. Write 1 to clear. *Note: If the corresponding bit in PWRSEQ_LPPWEN register is set, the event generates an interrupt to wakeup the device from a low power mode when PWRSEQ_LPCN.bgoff is cleared to 0.*
6	aincomp2wkst	R/W1C	0	**Analog Input Comparator 2 Wakeup Status Flag** This bit is set when the comparator inputs detect an event. Write 1 to clear. *Note: If the corresponding bit in PWRSEQ_LPPWEN register is set, the event generates an interrupt to wakeup the device from a low power mode when PWRSEQ_LPCN.bgoff is cleared to 0.*
5	aincomp1wkst	R/W1C	0	**Analog Input Comparator 1 Wakeup Status Flag** This bit is set when the comparator inputs detect an event. Write 1 to clear. *Note: If the corresponding bit in PWRSEQ_LPPWEN register is set, the event generates an interrupt to wakeup the device from a low power mode when PWRSEQ_LPCN.bgoff is cleared to 0.*
4	aincomp0wkst	R/W1C	0	**Analog Input Comparator 0 Wakeup Status Flag** This bit is set when the comparator inputs detect an event. Write 1 to clear. *Note: If the corresponding bit in PWRSEQ_LPPWEN register is set, the event generates an interrupt to wakeup the device from a low power mode when PWRSEQ_LPCN.bgoff is cleared to 0.*

Peripheral Low Power Wakeup Status Flags			PWRSEQ_LPPWST	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
3	-	R/W	0	**Reserved.** Do not modify this bit from its reset value.
2	usbvbuswkst	R/W1C	0	**USB VBUS State Change Detect Flag** 0: Normal operation 1: The USB has been powered on or off by plugging or unplugging an external USB Host. *Note: If the corresponding bit in PWRSEQ_LPPWEN register is set, the event generates an interrupt to wakeup the device from a low power mode when PWRSEQ_LPCN.bgoff is cleared to 0.*
1:0	usblswkst	R/W1C	0	**USB Line State Change Detect Status Flag** If one or both USB differential pair D+/D- pins change state, one or both of this field's bits are correspondingly set. 0b00: No state change 0b01: D+ changed state 0b10: D- changed state 0b11: Both D+ and D- changed state. *Note: If the corresponding bit in PWRSEQ_LPPWEN register is set, the event generates an interrupt to wakeup the device from a low power mode when PWRSEQ_LPCN.bgoff is cleared to 0.*

Table 4-48: Peripheral Low Power Wakeup Enable Register

Peripheral Low Power Wakeup Enable			PWRSEQ_LPPWEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved** Do not modify this field.
7	aincomp3wken	R/W	0	**Analog Input Comparator 3 Wakeup Enable** Write to 1 to enable an interrupt to be generated when *PWRSEQ_LPPWST.aincomp3wkst* is set.
6	aincomp2wken	R/W	0	**Analog Input Comparator 2 Wakeup Enable** Write to 1 to enable an interrupt to be generated when *PWRSEQ_LPPWST.aincomp2wkst* is set.
5	aincomp1wken	R	0	**Analog Input Comparator 1 Wakeup Enable** Write to 1 to enable an interrupt to be generated when *PWRSEQ_LPPWST.aincomp1wkst* is set.
4	aincomp0wken	R/W	0	**Analog Input Comparator 0 Wakeup Enable** Write to 1 to enable an interrupt to be generated when *PWRSEQ_LPPWST.aincomp0wkst* is set.
3	-	R/W	0	**Reserved** Do not modify this bit from its reset value.
2	usbvbuswken	R/W	0	**USB VBUS State Change Wakeup Enable** Write 1 to enable an interrupt and wakeup the device from any low power mode when *PWRSEQ_LPPWST.usbvbuswkst* = 1.

Peripheral Low Power Wakeup Enable			PWRSEQ_LPPWEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
1:0	usblswkst	R/W	0	**USB Line State Change Wakeup Enable** Write 0b11 to enable an interrupt and wakeup the device from any low power mode when *PWRSEQ_LPPWST.usblswkst* does not equal 0.

Table 4-49: RAM Shutdown Control Register

RAM Shutdown Control			PWRSEQ_LPMEMSD	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:15	-	RO	0	**Reserved** Do not modify this field.
14	ic1sd	R/W	0	**Internal Flash ICC1 Shut Down** Write 1 to shut off power to the Internal Flash Memory ICC1. *Note: When this field is set, the contents of the Internal Flash Memory ICache1 RAM are destroyed. See GCR_MEMCKCN register for retention mode power settings*
13	rom1sd	R/W	0	**ROM1 Shut Down** 0: Power enabled. 1: Power shut down.
12	romsd	R/W	0	**ROM0 Shut Down** 0: Power enabled. 1: Power shut down.
11	usbfifosd	R/W	0	**USB FIFO Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *Note: See GCR_MEMCKCN register for retention mode power settings.*
10	cryptosd	R/W	0	**Crypto MAA RAM Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *Note: See GCR_MEMCKCN register for retention mode power settings.*
9	srccsd	R/W	0	**SRCC Cache RAM Shut Down** Write 1 to shut off power to the SPIXR Cache RAM. *Note: When this field is set, the contents of the SPIXR Cache RAM, are destroyed. See GCR_MEMCKCN register for retention mode power settings.*
8	icachexipsd	R/W	0	**SFCC Cache RAM Shut Down** Write 1 to shut off power to the SPI-XIPF Cache RAM. *Note: When this field is set, the contents of the SPI-XIPF Cache RAM, are destroyed. See GCR_MEMCKCN register for retention mode power settings.*
7	icachesd	R/W	0	**Internal Flash ICC0 Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *Note: When this field is set, the contents of the Internal Flash Memory ICache0 RAM are destroyed. Note: See GCR_MEMCKCN register for retention mode power settings.*

RAM Shutdown Control			PWRSEQ_LPMEMSD	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
6	-	RO	-	**Reserved** Do not modify this field.
5	sram5sd	R/W	0	**Sysram5 and Sysram11 Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *See Table 3-2 for base address and size information. Note: See GCR_MEMCKCN register for retention mode power settings.*
4	sram4sd	R/W	0	**Sysram4 and Sysram10 Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *See Table 3-2 for base address and size information. Note: See GCR_MEMCKCN register for retention mode power settings.*
3	sram3sd	R/W	0	**Sysram3 and Sysram9 Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *See Table 3-2 for base address and size information. Note: See GCR_MEMCKCN register for retention mode power settings.*
2	sram2sd	R/W	0	**Sysram2 and Sysram8 Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *See Table 3-2: SRAM for base address and size information. Note: See GCR_MEMCKCN register for retention mode power settings.*
1	sram1sd	R/W	0	**Sysram1 and Sysram7 Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *See Table 3-2: SRAM for base address and size information. Note: See GCR_MEMCKCN register for retention mode power settings.*
0	sram0sd	R/W	0	**Sysram0 and Sysram6 Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *See Table 3-2: SRAM for base address and size information. Note: See GCR_MEMCKCN register for retention mode power settings.*

Table 4-50: Low Power VDD Power Down Register

Low Power VDD Power Down			PWRSEQ_LPVDDPD	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	R/W	-	**Reserved** Reserved. Do not modify this field.

Low Power VDD Power Down			PWRSEQ_LPVDDPD	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
11	vdd5pd	R/W	0	**VDD5 Power Domain Control** This register field controls the power to the Bluetooth digital baseband domain. 0: Power enabled 1: Power disabled *Caution: If this bit is set and subsequently cleared, the device enters the POR state.*
10	vdd4pd	R/W	0	**VDD4 Power Domain Control** This register field controls the power to the USB, SDHC/SDIO, and TPU peripherals. 0: Power enabled 1: Power disabled *Caution: If this bit is set and subsequently cleared, the device enters the POR state.*
9	vdd3pd	R/W	0	**VDD3 Power Domain Control** This register field controls the power to the CPU1 domain. 0: Power enabled 1: Power disabled *Caution: If this bit is set and subsequently cleared, the device enters the POR state.*
8	vdd2pd	R/W	0	**Reserved** Reserved. Do not modify this field.
7:2	-	R/W	-	**Reserved** Reserved. Do not modify this field.
1	vregodpd	R/W	0	**VREGO_D Regulator ACTIVE Mode Power Down** 0: Regulator enabled in any power mode 1: Regulator disabled in any power mode
0	vregobpd	R/W	0	**VREGO_B Regulator *BACKUP* Low Power Mode Mode Power Down** 0: Regulator enabled in BACKUP mode 1: Regulator automatically disabled when entering BACKUP mode

Table 4-51: BACKUP Return Vector Register

BACKUP Return Vector			PWRSEQ_BURETVEC	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	gpr0	R/W	0	**BACKUP Return Vector** This register is used as a jump address when waking from BACKUP mode.

Table 4-52: BACKUP AoD Register

BACKUP AoD			PWRSEQ_BUAOD	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	gpr1	R/W	0	**General Purpose Register 0** This register can be used for storage as it is preserved in the AoD domain from BACKUP mode.

4.16 Global Control Registers (GCR)

See *Table 3-3* for the the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field.

Note: The General Control Registers are only reset on a System Reset or Power-On Reset. A Soft Reset or Peripheral Reset does not affect these registers.

Table 4-53: Global Control Register Summary

Offset	Register	Description
[0x0000]	*GCR_SCON*	*System Control Register*
[0x0004]	*GCR_RSTR0*	*Reset Register 0*
[0x0008]	*GCR_CLKCN*	*Clock Control Register*
[0x000C]	*GCR_PM*	*Power Management Register*
[0x0018]	*GCR_PCKDIV*	*Peripheral Clocks Divisor*
[0x0024]	*GCR_PERCKCN0*	*Peripheral Clocks Disable 0*
[0x0028]	*GCR_MEMCKCN*	*Memory Clock Control*
[0x002C]	*GCR_MEMZCN*	*Memory Zeroize Register*
[0x0040]	*GCR_SYSST*	*System Status Flags*
[0x0044]	*GCR_RSTR1*	*Reset Register 1*
[0x0048]	*GCR_PERCKCN1*	*Peripheral Clocks Disable 1*
[0x004C]	*GCR_EVENT_EN*	*Event Enable Register*
[0x0050]	*GCR_REVISION*	*Revision Register*
[0x0054]	*GCR_SYSSIE*	*System Status Interrupt Enable*
[0x0064]	*GCR_ECCER*	*Error Correction Coding Error Register*
[0x0068]	*GCR_ECC_CED*	*Error Correction Coding Correctable Error Detected*
[0x006C]	*GCR_ECCIRQEN*	*Error Correction Coding Interrupt Enable Register*
[0x0070]	*GCR_ECCERRAD*	*Error Correction Coding Error Address Register*
[0x0074]	*GCR_BTLELDOCN*	*Bluetooth LDO Control Register*
[0x0078]	*GCR_BTLELDODLY*	*Bluetooth LDO Delay Count Register*
[0x0080]	*GCR_GP0*	*BACKUP Return Vector*
[0x0084]	*GCR_APBASYNC*	*Arm Peripheral Bus Asynchronous Bridge Select Register*

4.17 Global Control Register Details (GCR)

Table 4-54: System Control Register

System Control			GCR_SCON	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:18	-	RO	0	**Reserved** Do not modify this field.

System Control			GCR_SCON	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
17:16	ovr	R/W	0	**Operating Voltage Range** To allow on-chip volatile memory to operate at the optimal timing range, set this to be the same as V_{COREB}. 0b00: 0.9V ±10% 0b01: 1.0V ±10% 0b10: 1.1V ±10% 0b11: Reserved.
15	chkres	R	0	**ROM Checksum Calculation Pass/Fail** This is the result after setting bit *GCR_SCON.cchk*. This bit is only valid after the ROM checksum is complete and *GCR_SCON.cchk* is cleared. 0: Pass 1: Fail
14	-	RO	0	**Reserved** Do not modify this field.
13	cchk	R/W	0	**Calculate ROM Checksum** This bit is self-clearing when the ROM checksum calculation is complete, and the result is available at bit *GCR_SCON.chkres*. Writing a 0 has no effect. 0: No operation 1: Start ROM checksum calculation.
12:10	-	RO	0	**Reserved** Do not modify this field.
9	srcc_dis	R/W	0	**SPIXR Cache Controller (SRCC) Disable** This disables the SRCC used for SPIXR code and data cache. Setting this field disables the cache and bypasses the cache line buffer. 0: Cache enabled 1: Cache disabled, and line buffer bypassed
8	-	RO	0	**Reserved** Do not modify this field.
7	dcache_flush	R/W	0	**SPIXR Cache (SRCC) Flush** Write 1 to flush the SPIXR 16KB cache. This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect. 0: Memory flush not in progress. 1: Memory flush in progress.
6	ccache_flush	R/W	0	**ICC0/ICC1/SFCC Code Cache Flush** Write 1 to flush all three caches. This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect. 0: Memory flush not in progress. 1: Memory flush in progress.
5	-	RO	0	**Reserved** Do not modify this field.

System Control			GCR_SCON	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
4	flash_page_flip	R/*	0	**Flash Page Flip Flag** Flips the bottom and top halves of Flash memory. This bit is controlled by hardware. Firmware should not change the state of this bit during normal operation. Any change to this bit also flushes both code and data caches. 0: Physical layout matches logical layout 1: Top and Bottom halves flipped
3	-	RO	0	**Reserved** Do not modify this field.
2:1	sbusarb	R/W	1	**System Bus Arbitration Scheme** 00: Fixed Burst 01: Round-Robin 10: Reserved 11: Reserved
0	bstapen	R/W	*	**Boundary Scan Tap Enable** **Reserved** Do not modify this field. Reset value matches *GCR_SYSST.icelock.*

Table 4-55: Reset Register 0

Reset 0			GCR_RSTR0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31	system	R/W	0	**System Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
30	prst	R/W	0	**Peripheral Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress. *Note: Watchdog Timers, GPIO Ports, the AoD, RAM Retention and the General Control Registers (GCR) are unaffected.*
29	srst	R/W	0	**Soft Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
28	uart2	R/W	0	**UART2 Reset** Write 1 to reset. 0: Not in reset. 1: Reset in progress.
27	dma1	R/W	0	**DMA1 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.

Reset 0			GCR_RSTR0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
26	adc	R/W	0	**ADC Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
25:24	-	R/W	0	**Reserved** Do not modify this field.
23	usb	R/W	0	**USB Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
22	smphr	R/W	0	**Semaphore Block Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
21:19	-	R/W	-	**Reserved** Do not modify this field.
18	crypto	R/W	0	**Cryptographic Reset** Write 1 to reset. This resets the AES block, SHA block, and DES block. 0: Not in reset 1: Reset in progress.
17	rtc	R/W	0	**RTC Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
16	i2c0	R/W	0	**I2C0 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
15	-	R/W	-	**Reserved** Do not modify this field.
14	spi2	R/W	0	**SPI2 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
13	spi1	R/W	0	**SPI1 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
12	uart1	R/W	0	**UART1 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.

Reset 0			GCR_RSTR0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
11	uart0	R/W	0	**UART0 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
10	timer5	R/W	0	**TMR5 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
9	timer4	R/W	0	**TMR4 Reset** Write 1 to reset the peripheral. 1: Reset in progress. 0: Not in reset
8	timer3	R/W	0	**TMR3 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
7	timer2	R/W	0	**TMR2 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
6	timer1	R/W	0	**TMR1 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
5	timer0	R/W	0	**TMR0 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
4	-	RO	-	**Reserved** Do not modify this field.
3	gpio1	R/W	0	**GPIO1 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
2	gpio0	R/W	0	**GPIO0 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.
1	wdt0	R/W	0	**Watchdog Timer 0 Reset** Write 1 to reset. 0: Not in reset 1: Reset in progress.

Reset 0			GCR_RSTR0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
0	dma	R/W	0	**DMA Access Block** Write 1 to reset. 0: Not in reset 1: Reset in progress.

Table 4-56: System Clock Control Register

System Clock Control			GCR_CLKCN	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	0b011	**Reserved** Do not modify this field.
28	hirc8m_rdy	R	0	**7.3728MHz Internal Oscillator Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
27	hirc96m_rdy	R	0	**96MHz Internal Oscillator Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
26	hirc_rdy	R	1	**60MHz Internal Oscillator Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
25	x32k_rdy	R	0	**32.768kHz External Oscillator Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
24	x32M_rdy	R	0	**32MHz Bluetooth Oscillator Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
23:22	-	RO	0	**Reserved** Do not modify this field.
21	hirc8m_vs	R/W	0	**7.3728MHz Internal Oscillator Voltage Source Select** In DEEPSLEEP the 7.3728MHz oscillator voltage is sourced by pin V_{DDA}. When exiting DEEPSLEEP the voltage is automatically switched back to this bit setting. 0: V_{COREA} 1: Internal 1V regulator sourced from pin V_{DDA}
20	hirc8m_en	R/W	0	**7.3728MHz Internal Oscillator Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCN.hirc8m_rdy* = 1.
19	hirc96m_en	R/W	0	**96MHz Internal Oscillator Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCN.hirc96m_rdy* = 1.
18	hirc_en	R/W	1	**60MHz Internal Oscillator Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCN. hirc_rdy* = 1.
17	x32k_en	R/W	0	**32.768kHz External Oscillator Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCN.x32k_rdy* = 1.

System Clock Control			GCR_CLKCN	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
16	x32M_en	R/W	0	**32MHz Bluetooth Oscillator Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCN.x32m_rdy* = 1.
15	ccd	R/W	0	**Crypto Accelerator Clock Divider Status** 0: Crypto clock divide by 1 1: Crypto clock is divide by 2
14	-	RO	0	**Reserved** Do not modify this field.
13	ckrdy	R	0	**SYS_OSC Select Ready** When SYS_OSC is changed by modifying *clksel*, there is a delay until the switchover is complete. This bit is cleared until the switchover is complete. 0: Switch to new clock source not yet complete. 1: SYS_OSC is clock source selected in *clksel*.
12	-	RO	-	**Reserved** Do not modify this field.
11:9	clksel	R/W	0	**System Oscillator Source Select** Selects the system oscillator (SYS_OSC) source used to generate the system clock (SYS_CLK). Modifying this field immediately clears *GCR_CLKCN.ckrdy*. 0: 60MHz LP Internal Oscillator 1: Reserved 2: 32MHz Bluetooth Oscillator 3: 8kHz Internal Oscillator 4: 96MHz Internal Oscillator 5: 7.3728MHz Internal Oscillator 6: 32.768kHz External Oscillator 7: Reserved
8:6	psc	R/W	0	**System Oscillator Prescaler** Sets the divider for generating SYS_CLK from the selected SYS_OSC as shown in the following equation: $SYS\_CLK = \frac{SYS\_OSC}{2^{psc}}$
5:0	-	R/W	0b001000	**Reserved** Do not modify this field.

Table 4-57: Power Management Register

Power Management			GCR_PM	0x000C
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0	**Reserved** Do not modify this field.
20	xtalbp	R/W	0	**32MHz Bluetooth Oscillator Bypass** This bit is set to 0 on a POR and is not affected by other resets. 0: Clock source is crystal oscillator, driving 32MHz crystal connected between HFXIN and HFXOUT pins. 1: Clock source is 32MHz square wave driven into HFXIN pin.

Power Management			GCR_PM	0x000C
Bits	**Field**	**Access**	**Reset**	**Description**
19:18	-	RO	0	**Reserved** Do not modify this field.
17	hirc8mpd	R/W	1	**7.3728MHz Internal Oscillator Power Down** Not Used.
16	hirc96mpd	R/W	1	**96MHz Internal Oscillator Power Down** Not Used.
15	hircpd	R/W	1	**60MHz Internal Oscillator Power Down** Not Used.
14:10	-	RO	0	**Reserved** Do not modify this field.
9	compwken	R/W	0	**Comparator Input Wake Up Enable** The device will exit SLEEP, DEEPSLEEP, and BACKUP modes when hardware generates the corresponding interrupt. 0: Disabled. 1: Enabled.
8	-	RO	0	**Reserved** Do not modify this field.
7	wutwken	R/W	0	**Wakeup Timer Wake Up Enable** A wakeup event exits: SLEEP DEEPSLEEP BACKUP 0: Disabled. 1: Enabled.
6	usbwken	R/W	0	**USB Wakeup Enable** A wakeup event causes an exit from all low power modes and transitions directly to ACTIVE mode. 0: Disabled. 1: Enabled. Any USB bus activity or USB power on/off wakes up the device, except in BACKUP mode when only a USB power on/off wakes the device.
5	rtcwken	R/W	0	**RTC Alarm Wakeup Enable** Set this field to 1 to enable an RTC alarm to wake the device from any low power mode to ACTIVE mode. 0: Disabled. Wakeup from RTC alarm disabled, regardless if the RTC is configured to generate a wakeup alarm. 1: Enabled.
4	gpiowken	R/W	0	**GPIO Wakeup Enable** Activity on any GPIO pin configured for wakeup causes an exit from SLEEP, DEEPSLEEP, and BACKUP modes. 0: Disabled. 1: Enabled.

Power Management			GCR_PM	0x000C
Bits	**Field**	**Access**	**Reset**	**Description**
3	-	RO	0	**Reserved** Do not modify this field.
2:0	mode	R/W	0	**Operating Mode** 0b000: ACTIVE 0b010: DEEPSLEEP 0b011: Reserved. 0b100: BACKUP *Note: All other values are Reserved.*

Table 4-58: Peripheral Clock Divisor Register

Peripheral Clocks Divisor			GCR_PCKDIV	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	-	**Reserved** Do not modify this field.
17:16	audclksel	R/W	0	**Audio Subsytem Clock Source Select** 0: 96MHz high-frequency internal oscillator 1: 32MHz external crystal oscillator 2: External clock source provided at device pin P0.23 3-7: Reserved
15:14	aoncd	R/W	0	**Always-on Domain (AoD) Clock Divider** Configures the frequency of the Always On Domain clock as shown in the following equation. $f_{aod_{clk}} = \frac{f_{PCLK}}{(4 \times 2^{aoncd})}$ *Note: aoncd valid values are 0, 1, 2, and 3.*
13:10	adcfrq	R/W	0	**ADC Clock Divider** Configures the frequency of the ADC peripheral from the PCLK. 0x0: Reserved 0x1: Reserved 0x2 – 0xF: $f_{adc_{clk}} = {f_{PCLK}}/{adcfrq}$
9:8	-	RO	0	**Reserved** Do not modify this field.
7	sdhcfrq	R/W	0	**SDHC Clock Frequency** Configures the frequency of the SDHC as a divisor of the 96MHz high-speed oscillator. 0: $f_{SDHC\_CLK} = {96MHz}/{2}$ 1: $f_{SDHC\_CLK} = {96MHz}/{4}$
6:0	-	RO	0	**Reserved**

Table 4-59: Peripheral Clock Disable Register 0

Peripheral Clocks Disable 0			GCR_PERCKCN0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31	spimd	R/W	1	**SPIXF Master Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
30	spixipd	R/W	1	**SPIXF Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled
29	ptd	R/W	1	**Pulse Train Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled
28	i2c1d	R/W	1	**I2C1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled
27:24	-	R/W	0b1111	**Reserved** Do not modify this field.
23	adcd	R/W	1	**ADC Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled
22:21	-	R/W	0b11	**Reserved** Do not modify this field.
20	timer5d	R/W	1	**TMR5 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
19	timer4d	R/W	1	**TMR4 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
18	timer3d	R/W	1	**TMR3 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.

Peripheral Clocks Disable 0			GCR_PERCKCN0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
17	timer2d	R/W	1	**TMR2 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
16	timer1d	R/W	1	**TMR1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
15	timer0d	R/W	1	**TMR0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
14	cryptod	R/W	1	**Crypto Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
13	i2c0d	R/W	1	**I2C0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
12:11	-	RO	0	**Reserved**
10	uart1d	R/W	1	**UART1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
9	uart0d	R/W	1	**UART0 Clock Disable** Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 1: Clock disabled to peripheral. 0: Clock enabled to peripheral.
8	-	RO	0	Reserved
7	spi2d	R/W	1	**SPI2 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.

Peripheral Clocks Disable 0			GCR_PERCKCN0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
6	spi1d	R/W	1	**SPI1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
5	dmad	R/W	1	**DMA Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
4	-	RO	-	**Reserved** Do not modify this field.
3	usbd	R/W	1	**USB Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
2	-	RO	-	**Reserved** Do not modify this field.
1	gpio1d	R/W	1	**GPIO1 Port and Pad Logic Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.
0	gpio0d	R/W	1	**GPIO0 Port and Pad Logic Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled. 1: Clock disabled.

Table 4-60: Memory Clock Control Register

Memory Clock Control			GCR_MEMCKCN	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31	icache1ls	R/W	0	**Internal Flash ICC1 LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled.
30	rom1ls	R/W	0	**ROM1 LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled.

Memory Clock Control			GCR_MEMCKCN	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
29	romls	R/W	0	**ROM0 LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled.
28	usbls	R/W	0	**USB FIFO LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled.
27	cryptols	R/W	0	**Crypto RAM LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled.
26	scachels	R/W	0	**SRCC Cache LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled.
25	icachexipls	R/W	0	**SFCC Cache RAM LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled.
24	icachels	R/W	0	**Internal Flash ICC0 LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled.
23	-	RO	-	**Reserved** Do not modify this field.
22	sysram6ls	R/W	0	**Sysram6 to Sysram11 LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. See *Table 3-2* for base address and size information. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register.*
21	sysram5ls	R/W	0	**Sysram5 LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. See *Table 3-2:* SRAM for base address and size information. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register.*

Memory Clock Control			GCR_MEMCKCN	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
20	sysram4ls	R/W	0	**Sysram4 LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. See *Table 3-2:* SRAM for base address and size information. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register.*
19	sysram3ls	R/W	0	**Sysram3 LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. See *Table 3-2:* SRAM for base address and size information. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register.*
18	sysram2ls	R/W	0	**Sysram2 LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. See *Table 3-2:* SRAM for base address and size information. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register.*
17	sysram1ls	R/W	0	**Sysram1 LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. See *Table 3-2:* SRAM for base address and size information. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register.*
16	sysram0ls	R/W	0	**Sysram0 LIGHTSLEEP Enable** Data is unavailable for read/write operations in LIGHTSLEEP mode but is retained. See *Table 3-2:* SRAM for base address and size information. 0: ACTIVE mode. 1: LIGHTSLEEP mode enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register.*
15:3	-	RO	-	**Reserved** Do not modify this field.
2:0	fws	R/W	0b101	**Program Flash Wait States** Number of wait-state cycles per flash code read access. 0: Invalid 1 – 7: Number of flash code access wait states *Note: For the 60MHz clock and slower, minimum wait state is 1.* *Note: For the 96MHz clock, the minimum wait states should be 2.*

Table 4-61: Memory Zeroization Control Register

Memory Zeroization Control			GCR_MEMZCN	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:15	-	RO	-	**Reserved** Do not modify this field.
14	icache1z	R/W	0	**CPU1 ICC1 Cache Data and Tag Zeroization** Write 1 to initiate the operation. Only valid on devices with optional CPU1. 0: Operation complete. 1: Operation in progress.
13	usbfifoz	R/W	0	**USB FIFO Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
12	cryptoz	R/W	0	**Crypto MAA Memory Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
11	scachetagz	R/W	0	**SRCC Cache Tag Zeroization** Write 1 to clear the SRCC tag RAM. 0: Operation complete. 1: Operation in progress.
10	scachedataz	R/W	0	**SRCC Cache Data Zeroization** Write 1 to initiate the operation to clear the SFCC Data RAM to 0. 0: Operation complete. 1: Operation in progress.
9	icachexipz	R/W	0	**SFCC Cache Data and Tag Zeroization** Write 1 to clear the ICC1 16KB cache RAM to 0. The bit is set to 0 when the operation is complete. 0: Operation complete. 1: Operation in progress.
8	icachez	R/W	0	**CPU0 ICC0 Cache Data and Tag Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
7	-	R/W	0	**Reserved** Do not modify this field.
6	sram6z	R/W	0	**Sysram6 to Sysram11 Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
5	sram5z	R/W	0	**Sysram5 Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Memory Zeroization Control			GCR_MEMZCN	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
4	sram4z	R/W	0	**Sysram4 Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
3	sram3z	R/W	0	**Sysram3 Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
2	sram2z	R/W1	0	**Sysram2 Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
1	sram1z	R/W1	0	**Sysram1 Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
0	sram0z	R/W1	0	**Sysram0 Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Table 4-62: System Status Flag Register

System Status Flag			GCR_SYSST	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved** Do not modify this field.
5	scmemf	R	0	**SRCC Cache Memory Error Status Flag** Indicates a memory fault has occurred in the cache while receiving data from the SPIXR interface. 0: Normal operation 1: SPIXR cache memory fault
4:2	-	RO	0	**Reserved** Do not modify this field.
1	codeinterr	R/W1C	0	**Flash SPIXF Code Integrity Error Status Flag** Write 1 to clear this field. This indicates a code integrity error has occurred in the Flash SPI-XIPF interface. 0: Error detected 1: No error.
0	icelock	R	0	**Arm ICE Lock Status Flag** 0: Arm ICE is unlocked (enabled) 1: Arm ICE is locked (disabled)

Table 4-63: Reset Register 1

Reset 1			GCR_RSTR1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved** Do not modify this field.
25	simo	R/W	0	**Single Inductor Multiple Output Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
24	dvs	R/W	0	**Dynamic Voltage Scaling Controller Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
23	htmr1	R/W	0	**HTIMER1 Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
22	htmr0	R/W	0	**HTIMER0 Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
21	rpu	R/W	0	**Resource Protection Unit Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
20	i2c2	R/W	0	**I2C2 Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
19	audio	R/W	0	**Audio Interface Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
18	btle	R/W	0	**Bluetooth Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
17	wdt2	R/W	0	**Watchdog Timer 2 Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
16	smphr	R/W	0	**Semaphore Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Reset 1			GCR_RSTR1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
15	spixmem	R/W	0	**SPIXR Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
14:10	-	R/W	-	**Reserved** Do not modify this field.
9	spi0	R/W	0	**SPI0 Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
8	wdt1	R/W	0	**Watchdog Timer 1 Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
7	owire	R/W	0	**1-Wire Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
6	sdhc	R/W	0	**SDHC Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
5	-	RO	0	**Reserved.**
4	xspim	R/W	0	**XSPI Master Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
3	spixip	R/W	0	**SPI-XIPF Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
2	-	RO	0	**Reserved**
1	pt	R/W	0	**Pulse Train Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
0	i2c1	R/W	0	**I2C1 Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Table 4-64: Peripheral Clock Disable Register 1

Peripheral Clock Disable 1			GCR_PERCKCN1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31	cpu1	R/W	1	**CPU1 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
30	-	RO	-	**Reserved** Do not modify this field.
29	wdt2	R/W	1	**WDT2 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
28	wdt1	R/W	1	**Watchdog Timer 1 Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
27	wdt0	R/W	1	**Watchdog Timer 0 Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
26	htmr1	R/W	1	**HTIMER1 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
25	htmr0	R/W	1	**HTIMER0 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
24	i2c2	R/W	1	**I²C2 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
23	audio	R/W	1	**Audio Interface Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
22	-	RO	1	**Reserved** Do not modify this field.

Peripheral Clock Disable 1			GCR_PERCKCN1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
21	dma1	R/W	1	**DMA1 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
20	spixipdd	R/W	1	**SPIXR Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
19:15	-	RO	1	**Reserved** Do not modify this field.
14	spi0d	R/W	1	**SPI0 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
13	owired	R/W	1	**1-Wire Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
12	icachexipd	R/W	1	**SFCC Flash Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
11	-	R/W	1	**Reserved** Do not modify this field.
10	sdhcd	R/W	1	**SDHC Controller Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
9	smphrd	R/W	1	**Semaphore Block Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
8	sdmad	R/W	1	**Bluetooth Hardware Accelerator Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.

Peripheral Clock Disable 1			GCR_PERCKCN1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
7	scached	R/W	1	**SRCC Cache Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
6:3	-	R/W	1	**Reserved** Do not modify this field.
2	trngd	R/W	1	**TRNG Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
1	uart2d	R/W	1	**UART2 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
0	btled	R/W	1	**Bluetooth Digital Baseband Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.

Table 4-65: Event Enable Register

Event Enable			GCR_EVENT_EN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved** Do not modify this field.
5	cpu1txevent	R/W	0	**CPU1 TXEV On SEV Enable** When enabled, any SEV (Send Event) instruction will set CPU1's event register. WFE (Wait for Event) instructions executed on CPU1 will then complete. 0: Disabled. 1: Enabled.
4	cpu1dma1event	R/W	0	**CPU1 DMA1 CTZ Wake-Up Enable** When enabled, interrupts generated by DMA Controller 1 (DMAC1) will also set CPU1's event register. WFE (Wait for Event) instructions executed on CPU1 will then complete. 0: Disabled. 1: Enabled.
3	cpu1dmaevent	R/W	0	**CPU1 DMA0 CTZ Wake-Up Enable** When enabled, interrupts generated by DMA Controller 0 (DMAC0) will also set CPU1's event register. WFE (Wait for Event) instructions executed on CPU1 will then complete. 0: Disabled. 1: Enabled..

Event Enable			GCR_EVENT_EN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
2	cpu0txevent	R/W	0	**CPU0 TXEV On SEV Enable** When enabled, any SEV (Send Event) instruction will set CPU0's event register. WFE (Wait for Event) instructions executed on CPU0 will then complete. 0: Disabled. 1: Enabled.
1	cpu0dma1event	R/W	0	**CPU0 DMA1 CTZ Wake-Up Enable** When enabled, interrupts generated by DMA1 will also set CPU0's event register. WFE (Wait for Event) instructions executed on CPU0 will then complete. 0: Disabled. 1: Enabled.
0	cpu0dmaevent	R/W	0	**CPU0 DMA0 CTZ Wake-Up Enable** When enabled, interrupts generated by DMA0 will also set CPU0's event register. WFE (Wait for Event) instructions executed on CPU0 will then complete. 0: Disabled. 1: Enabled.

Table 4-66: Revision Register

Revision			GCR_REVISION	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	revision	R	*	**Device Revision** Returns the chip revision ID as a packed BCD. For example, 0xA1 would indicate the device is revision A1.

Table 4-67: System Status Interrupt Enable Register

System Status Interrupt Enable			GCR_SYSSIE	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	-	**Reserved** Do not modify this field.
5	scmfie	R/W	0	**SRCC Cache Memory Fault Interrupt Enable** Generates an interrupt if hardware detects an error in the SPIXR code. 0: Disabled. 1: Enabled.
4:2	-	RO	-	**Reserved** Do not modify this field.
1	cieie	R/W	0	**SPIXF Code Integrity Error Interrupt Enable** Generates an interrupt if hardware detects an error in the SPIXF. 0: Disabled. 1: Enabled.

System Status Interrupt Enable			GCR_SYSSIE	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
0	iceulie	R/W	0	**Arm ICE Unlocked Interrupt Enable** Generates an interrupt if the Arm ICE is unlocked. 0: Disabled. 1: Enabled.

Table 4-68: Error Correction Coding Error Register

Error Correction Coding Error			GCR_ECCER	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:13	-	RO	0	**Reserved**
12	fl1eccerr	R/W1C	0	**Flash1 ECC Error** Write to 1 to clear the flag. 0: No error 1: Error
11	fl0eccerr	R/W1C	0	**Flash0 ECC Error** Write to 1 to clear the flag. 0: No error 1: Error
10	icxipfeccerr	R/W1C	0	**SPIXF Instruction Cache ECC Error** Write to 1 to clear the flag. 0: No error 1: Error
9	ic1eccerr	R/W1C	0	**Instruction Cache 1 ECC Error** Write to 1 to clear the flag. 0: No error 1: Error
8	ic0eccerr	R/W1C	0	**Instruction Cache 0 ECC Error** Write to 1 to clear the flag. 0: No error 1: Error
7:6	-	RO	0	**Reserved**
5	sysram5eccerr	R/W1C	0	**System RAM5 ECC Error** Write to 1 to clear the flag. 0: No error 1: Error
4	sysram4eccerr	R/W1C	0	**System RAM4 ECC Error** Write to 1 to clear the flag. 0: No error 1: Error
3	sysram3eccerr	R/W1C	0	**System RAM3 ECC Error** Write to 1 to clear the flag. 0: No error 1: Error

Error Correction Coding Error			GCR_ECCER	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
2	sysram2eccerr	R/W1C	0	**System RAM2 ECC Error** Write to 1 to clear the flag. 0: No error 1: Error
1	sysram1eccerr	R/W1C	0	**System RAM1 ECC Error** Write to 1 to clear the flag. 0: No error 1: Error
0	sysram0eccerr	R/W1C	0	**System RAM0 ECC Error** Write to 1 to clear the flag. 0: No error 1: Error

Table 4-69: Error Correction Coding Correctable Error Detected

Error Correction Coding Correctable Error Detected			GCR_ECC_CED	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:13	-	RO	0	**Reserved**
12	fl1eccnded	R/W1C	1	**Flash1 Not Double ECC Error Detected** When cleared, indicates that there is a single correctable error in the Flash1 bank. Write to 1 to clear the flag. 0: No error 1: Error
11	fl0eccnded	R/W1C	1	**Flash0 Correctable ECC Error Detected** When cleared, indicates that there is a single correctable error in the Flash0 bank. Write to 1 to clear the flag. 0: No error 1: Error
10	icxipeccnded	R/W1C	1	**SPIXF Instruction Cache Correctable ECC Error Detected** When cleared, indicates that there is a single correctable error in SPIXF Instruction Cache. Write to 1 to clear the flag. 0: No error 1: Error
9	ic1eccnded	R/W1C	1	**Instruction Cache 1 Correctable ECC Error Detected** When cleared, indicates that there is a single correctable error in Instruction Cache 1. Write to 1 to clear the flag. 0: No error 1: Error
8	ic0eccnded	R/W1C	1	**Instruction Cache 0 Correctable ECC Error Detected** When cleared, indicates that there is a single correctable error in Instruction Cache 0. Write to 1 to clear the flag. 0: No error 1: Error
7:6	-	RO	0	**Reserved**

Error Correction Coding Correctable Error Detected			GCR_ECC_CED	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
5	sysram5eccnded	R/W1C	1	**System RAM5 Correctable ECC Error Detected** When cleared, indicates that there is a single correctable error in the RAM5 block. Write to 1 to clear the flag. 0: No error 1: Error
4	sysram4eccnded	R/W1C	1	**System RAM4 Correctable ECC Error Detected** When cleared, indicates that there is a single correctable error in the RAM4 block. Write to 1 to clear the flag. 0: Error 1: No Error
3	sysram3eccnded	R/W1C	1	**System RAM3 Correctable ECC Error Detected** When cleared, indicates that there is a single correctable error in the RAM3 block. Write to 1 to clear the flag. 0: Error 1: No Error
2	sysram2eccnded	R/W1C	1	**System RAM2 Correctable ECC Error Detected** When cleared, indicates that there is a single correctable error in the RAM2 block. Write to 1 to clear the flag. 0: Error 1: No Error
1	sysram1eccnded	R/W1C	1	**System RAM1 Correctable ECC Error Detected** When cleared, indicates that there is a single correctable error in the RAM1 block. Write to 1 to clear the flag. 0: Error 1: No Error
0	sysram0eccnded	R/W1C	1	**System RAM0 Correctable ECC Error Detected** When cleared, indicates that there is a single correctable error in the RAM0 block. Write to 1 to clear the flag. 0: Error 1: No Error

Table 4-70: Error Correction Coding Interrupt Enable Register

Error Correction Coding Interrupt Enable			GCR_ECCIRQEN	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:13	-	RO	0	**Reserved.**
12	fl1eccen	R/W	0	**Flash1 ECC Error Interrupt Enable** 0: Disabled 1: Enabled
11	fl0eccen	R/W	0	**Flash0 ECC Error Interrupt Enable** 0: Disabled 1: Enabled
10	icxipeccen	R/W	0	**SPIXF Instruction Cache ECC Error Interrupt Enable** 0: Disabled 1: Enabled

Error Correction Coding Interrupt Enable				GCR_ECCIRQEN	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**	
9	ic1eccen	R/W	0	**Instruction Cache 1 ECC Error Interrupt Enable** 0: Disabled 1: Enabled	
8	ic0eccen	R/W	0	**Instruction Cache 0 ECC Error Interrupt Enable** 0: Disabled 1: Enabled	
7:6	-	RO	0	**Reserved.**	
5	sysram5eccen	R/W	0	**Sysram5 ECC Error Interrupt Enable** 0: Disabled 1: Enabled	
4	sysram4eccen	R/W	0	**Sysram4 ECC Error Interrupt Enable** 0: Disabled 1: Enabled	
3	sysram3eccen	R/W	0	**Sysram3 ECC Error Interrupt Enable** 0: Disabled 1: Enabled	
2	sysram2eccen	R/W	0	**Sysram2 ECC Error Interrupt Enable** 0: Disabled 1: Enabled	
1	sysram1eccen	R/W	0	**Sysram1 ECC Error Interrupt Enable** 0: Disabled 1: Enabled	
0	sysram0eccen	R/W	0	**Sysram0 ECC Error Interrupt Enable** 0: Disabled 1: Enabled	

Table 4-71: Error Correction Coding Address Register

Error Correction Coding Address				GCR_ECCERRAD	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**	
31	tagramerr	R	0	**ECC Error Address/TAG RAM Error** Data depends on which block has reported the error. If sysram, fl0, or fl1, then this bit(s) represents the bit(s) of the AMBA address of read which produced the error. If the error is from one of the caches, then this bit is set as shown below: 0: No error 1: Tag_Error. The error is in the TAG RAM.	
30	tagrambank	R	0	**ECC Error Address/TAG RAM Error Bank** Data depends on which block has reported the error. If sysram, fl0, or fl1, then this bit(s) represents the bit(s) of the AMBA address of read which produced the error. If the error is from one of the caches, then this bit is set as shown below: 0: Error is in TAG RAM Bank 0 1: Error is in TAG RAM Bank 1	

Error Correction Coding Address				GCR_ECCERRAD	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**	
29:16	tagramaddr	R	0	**ECC Error Address/TAG RAM Error Address** Data depends on which block has reported the error. If sysram, fl0, or fl1, then this bit(s) represents the bit(s) of the AMBA address of read which produced the error. If the error is from one of the caches, then this bit is set as shown below: [TAG ADDRESS]: Represents the TAG RAM Address	
15	dataramerr	R	0	**ECC Error Address/DATA RAM Error Address** Data depends on which block has reported the error. If sysram, fl0, or fl1, then this bit(s) represents the bit(s) of the AMBA address of read which produced the error. If the error is from one of the caches, then this bit is set as shown below: 0: No error 1: DATA RAM Error. The error is in the Data RAM	
14	datarambank	R	0	**ECC Error Address/DATA RAM Error Bank** Data depends on which block has reported the error. If sysram, fl0, or fl1, then this bit(s) represents the bit(s) of the AMBA address of read which produced the error. If the error is from one of the caches, then this bit is set as shown below: 0: Error is in DATA RAM Bank 0 1: Error is in DATA RAM Bank 1	
13:0	dataramaddr	R	0	**ECC Error Address/TAG RAM Error Address** Data depends on which block has reported the error. If sysram, fl0, or fl1, then this bit(s) represents the bit(s) of the AMBA address of read which produced the error. If the error is from one of the caches, then this bit is set as shown below: [DATA ADDRESS]: Represents the DATA RAM Error Address	

Table 4-72: Bluetooth LDO Control Register

Bluetooth LDO Control				GCR_BTLELDOCN	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:16	-	RO	0	**Reserved** Do not modify this field.	
15	ldotxbypenendly	R	0	**LDOTX Bypass Enable Delay** Not used	
14	ldorxbypenendly	R	0	**LDORX Bypass Enable Delay** Not used.	
13	ldowendly	R	0	**LDORX Enable Delay** Not used.	
12	ldowoendly	R	0	**LDOTX Enable Delay** Not used.	
11	ldowodisch	R/W	0	**LDOTX Discharge** This bit is used to discharge the LDOTX output using a strong pulldown. For use when switching between Bypass Mode and Regulation Mode. 0: No discharge 1: Discharge	

Bluetooth LDO Control				GCR_BTLELDOCN	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**	
10	ldowobyp	R/W	0	**LDOTX Bypass Enable** This bit enables the LDOTX Bypass Mode and charge the LDOTX output voltage to its input voltage level.	
9	ldowdisch	R/W	0	**LDORX Discharge** This bit is used to discharge the LDORX output using a strong pulldown. For use when switching between Bypass Mode and Regulation Mode. 0: No discharge 1: Discharge	
8	ldobyp	R/W	0	**LDORX Bypass Enable** This bit enables the LDORX Bypass Mode and charges the LDORX output voltage to its input voltage level.	
7:6	ldowvsel	R/W	b'01	**LDORX Output Voltage Setting** 0b00: 0.7V 0b01: 0.85V 0b10: 0.9V 0b11: 1.1V	
5	ldowpulld	R/W	1	**LDORX Pulldown** This bit is used to provide a weak pulldown to the LDORX output. 0: Pulldown disabled 1: Pulldown enabled	
4	ldowen	R/W	0	**LDORX Enable** Enable the LDO and charge its output voltage to the setting in *GCR_BTLELDOCN.ldowvsel*. 0: Disabled 1: Enabled	
3:2	ldowovsel	R/W	b'01	**LDOTX Output Voltage Setting** 0b00: 0.7V 0b01: 0.85V 0b10: 0.9V 0b11: 1.1V	
1	ldowopulld	R/W	1	**LDOTX Pull Down** Setting this bit enables a weak pulldown on the output of the TX LDO. 0: Pulldown disabled 1: Pulldown enabled	
0	ldowoen	R/W	0	**LDOTX Enable** Enable the LDO and charge its output voltage to the setting in *GCR_BTLELDOCN.ldowovsel*. 0: Disabled 1: Enabled	

Table 4-73: Bluetooth LDO Delay Count Register

Bluetooth LDO Delay Count				GCR_BTLELDODLY	[0x0078]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:29	-	RO	0	**Reserved**	

Bluetooth LDO Delay Count				GCR_BTLELDODLY	[0x0078]
Bits	**Field**	**Access**	**Reset**	**Description**	
28:20	ldotxdlycnt	R/W	0x01B	**Bluetooth LDOTX Delay Count** Not used.	
19:17	-	RO	0	**Reserved**	
16:8	ldorxdlycnt	R/W	0x01B	**Bluetooth LDORX Delay Count** Not used.	
7:0	bypdlycnt	R/W	0x28	**Bluetooth LDO Bypass Delay Count** Not used.	

Table 4-74: General Purpose 0 Register

General Purpose 0				GCR_GP0	[0x0080]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	gpr0	R/W	0	User-defined register RAM	

Table 4-75: Arm Peripheral Bus Asynchronous Bridge Select Register

Arm Peripheral Bus Asynchronous Bridge Select				GCR_APBASYNC	[0x0084]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:4	-	RO	0	**Reserved** Do not modify this field.	
3	apbasyncpt	R/W	0	**Pulse Trains Peripheral Bus Select** This peripheral can be connected to the APB PCLK domain or a 7.3728MHz bus can be used. It takes 3 cycles of the 7.3728MHz clock to switch PCLK or 3 cycles of the PCLK clock to switch to 7.37MHz clock. 0: Peripheral is accessed on the PCLK bus. 1: Peripheral is accessed on the 7.3728MHz bus.	
2	apbasyncI2C2	R/W	0	**I2C2 Peripheral Bus Select** The access for this peripheral can be performed via one of two different peripheral bus configurations. The system PCLK can be used as any of the other system peripherals that are connected to the APB PCLK domain or a 7.3728MHz bus can be used. It takes 3 cycles of the 7.3728MHz clock to switch PCLK or 3 cycles of the PCLK clock to switch to 7.37MHz clock. After switching, ensure enough time before accessing the peripheral registers. 0: PCLK bus selected 1: 7.3728MHz bus selected	
1	apbasyncI2C1	R/W	0	**I2C1 Peripheral Bus Select** The access for this peripheral can be performed via one of two different peripheral bus configurations. The system PCLK can be used as any of the other system peripherals that are connected to the APB PCLK domain or a 7.3728MHz bus can be used. It takes 3 cycles of the 7.3728MHz clock to switch PCLK or 3 cycles of the PCLK clock to switch to 7.37MHz clock. After switching, ensure enough time before accessing the peripheral registers. 0: PCLK bus selected 1: 7.3728MHz bus selected	

Arm Peripheral Bus Asynchronous Bridge Select			GCR_APBASYNC	[0x0084]
Bits	**Field**	**Access**	**Reset**	**Description**
0	apbasyncI2C0	R/W	0	**I2C0 Peripheral Bus Select** The access for this peripheral can be performed via one of two different peripheral bus configurations. The system PCLK can be used as any of the other system peripherals that are connected to the APB PCLK domain or a 7.3728MHz bus can be used. It takes 3 cycles of the 7.3728MHz clock to switch PCLK or 3 cycles of the PCLK clock to switch to 7.37MHz clock. After switching, ensure enough time before accessing the peripheral registers. 0: PCLK bus selected 1: 7.3728MHz bus selected

4.18 Function Control Registers

See *Table 3-3* for the the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset only on a system reset or POR, but not a soft reset.

Table 4-76: Function Control Register Summary

Offset	Register	Description
[0x0000]	*GCR_FCR*	Function Control Register

4.19 Function Control Register Details

Table 4-77: Function Control Register 0

Function Control 0			GCR_FCR	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved** Do not modify this field.
25	i2c2_scl_filter_en	R/W	0	**I2C2 SCL Glitch Filter Enable** 0: Disabled 1: Enabled
24	i2c2_sda_filter_en	R/W	0	**I2C2 SDA Glitch Filter Enable** 0: Disabled 1: Enabled
23	i2c1_scl_filter_en	R/W	0	**I2C1 SCL Glitch Filter Enable** 0: Disabled 1: Enabled
22	i2c1_sda_filter_en	R/W	0	**I2C1 SDA Glitch Filter Enable** 0: Disabled 1: Enabled
21	i2c0_scl_filter_en	R/W	0	**I2C0 SCL Glitch Filter Enable** 0: Disabled 1: Enabled
20	i2c0_sda_filter_en	R/W	0	**I2C0 SDA Glitch Filter Enable** 0: Disabled 1: Enabled

Function Control 0				GCR_FCR	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
19:18	-	RO	0	**Reserved** Do not modify this field.	
17	qspi0_fnc_sel	R/W	0	**SPI0 Function Select** 0: High speed 96MHz oscillator 1: External clock input *Note: See the GPIO chapter for the external clock input pin*	
16	usb_clk_sel	R/W	0	**USB Reference Clock Source Select** This selects the clock source for the USB Hi-Speed Interface. 0: High speed 96MHz oscillator 1: External clock input *Note: See the GPIO chapter for the external clock input pin.*	
15:0	-	RO	0	**Reserved** Do not modify this field.	

4.20 AES Key Registers

See *Table 3-3*: APB Peripheral Base Address Map for the AES Key Registers' Peripheral Base Address.

Table 4-78: AES Key Register Summary

Offset	Register	Description
[0x0000]	*AES_KEY0*	128-bit AES Key Register 0
[0x0080]	*AES_KEY1*	128-bit AES Key Register 1
[0x0100]	*AES_KEY2*	128-bit AES Key Register 2
[0x0180]	*AES_KEY3*	128-bit AES Key Register 3

4.21 AES Key Register Details

Table 4-79: AES Key 0 and 1 Registers

AES Key 0			AES_KEY0	[0x0000]
AES Key 1			**AES_KEY1**	**[0x0080]**
Bits	**Field**	**Access**	**Reset**	**Description**
127:0	aes_key	R/W	0	**AES 128-bit Key Registers** These two registers make up the 256-bit AES key, with the most significant bits in *AES_KEY1* and the least significant bits in *AES_KEY0*. This register is reset only on AoD Reset.

Table 4-80: AES Key 2 and 3 Registers

AES Key 2	AES_KEY2	[0x0100]
AES Key 3	AES_KEY3	[0x0180]

Bits	Field	Access	Reset	Description
127:0	aes_key	R/W	-	**AES 128-bit Key Registers** Each of these registers are loaded at system initialization with user-defined 128-bit keys. See the secure bootloader section in the TPU supplement for more information.

5. Interrupts and Exceptions

Interrupts and exceptions are managed by the Arm Cortex-M4 with FPU Nested Vector Interrupt Controller (NVIC). The NVIC handles the interrupts, exceptions, priorities and masking. *Table 5-1* details the MAX32665/MAX32666 interrupt vector table and describes each exception and interrupt.

5.1 Features

- 59 maskable interrupts not including the 15 system exceptions of the Arm Cortex-M4 with FPU
- 8 programmable priority levels
- Nested exception and interrupt support
- Interrupt masking

5.2 Interrupt Vector Table

Table 5-1 lists the interrupt and exception table for the MAX32665/MAX32666. There are 95 interrupt entries for the MAX32665/MAX32666, including reserved for future use interrupt place holders. Including the 15 system exceptions for the Arm Cortex-M4 with FPU, the total number of entries is 110.

Table 5-1: MAX32665/MAX32666 Interrupt Vector Table

Exception (Interrupt) Number	Offset	Name	Description
1	[0x0004]	Reset_Handler	Reset
2	[0x0008]	NMI_Handler	Non-Maskable Interrupt
3	[0x000C]	HardFault_Handler	Hard Fault
4	[0x0010]	MemManage_Handler	Memory Management Fault
5	[0x0014]	BusFault_Handler	Bus Fault
6	[0x0018]	UsageFault_Handler	Usage Fault
7:10	[0x001C]-[0x0028]	-	Reserved
11	[0x002C]	SVC_Handler	Supervisor Call Exception
12	[0x0030]	DebugMon_Handler	Debug Monitor Exception
13	[0x0034]	-	Reserved
14	[0x0038]	PendSV_Handler	Request Pending for System Service
15	[0x003C]	SysTick_Handler	System Tick Timer
16	[0x0040]	SysFault_IRQHandler	System Fault interrupt
17	[0x0044]	WDT0_IRQHandler	Watchdog Timer 0 Interrupt
18	[0x0048]	USB_IRQHandler	USB Interrupt
19	[0x004C]	RTC_IRQHandler	Real-Time Clock Interrupt
20	[0x0050]	TRNG_IRQHandler	True Random Number Generator Intterrupt

Exception (Interrupt) Number	Offset	Name	Description
21	[0x0054]	TMR0_IRQHandler	Timer 0 Interrupt
22	[0x0058]	TMR1_IRQHandler	Timer 1 Interrupt
23	[0x005C]	TMR2_IRQHandler	Timer 2 Interrupt
24	[0x0060]	TMR3_IRQHandler	Timer 3 Interrupt
25	[0x0064]	TMR4_IRQHandler	Timer 4 Interrupt
26	[0x0068]	TMR5_IRQHandler	Timer 5 Interrupt
27	[0x006C]	-	Reserved
28	[0x0070]	-	Reserved
29	[0x0074]	I2C0_IRQHandler	I2C Port 0 Interrupt
30	[0x0078]	UART0_IRQHandler	UART Port 0 Interrupt
31	[0x007C]	UART1_IRQHandler	UART Port 1 Interrupt
32	[0x0080]	SPI1_IRQHandler	SPI Port 1 Interrupt
33	[0x0084]	SPI2_IRQHandler	SPI Port 2 Interrupt
34	[0x0088]	-	Reserved
35	[0x008C]	-	Reserved
36	[0x0090]	ADC_IRQHandler	ADC Interrupt
37:38	[0x0094]:[0x098]	-	Reserved
39	[0x009C]	FLC0_IRQHandler	Flash Controller 0 Interrupt
40	[0x00A0]	GPIO0_IRQHandler	GPIO Port 0 Interrupt
41	[0x00A4]	GPIO1_IRQHandler	GPIO Port 1 Interrupt
42	[0x00A8]	-	Reserved
43	[0x00AC]	TPU_IRQHandler	Trust Protection Unit Interrupt
44	[0x00B0]	DMA0_IRQHandler	DMA0 Interrupt
45	[0x00B4]	DMA1_IRQHandler	DMA1 Interrupt
46	[0x00B8]	DMA2_IRQHandler	DMA2 Interrupt
47	[0x00BC]	DMA3_IRQHandler	DMA3 Interrupt
48:49	[0x00C0 : 0x00C4]	-	Reserved
50	[0x00C8]	UART2_IRQHandler	UART Port 2 Interrupt
51	[0x00CC]	-	Reserved

Exception (Interrupt) Number	Offset	Name	Description
52	[0x00D0]	I2C1_IRQHandler	I2C Port 1 Interrupt
53	[0x00D4]	-	Reserved
54	[0x00D8]	SPIXFC_IRQHandler	SPI XIP Flash Controller Interrupt
55	[0x00DC]	BTLE_TX_DONE_IRQHandler	Bluetooth Transmitter Done Interrupt
56	[0x00E0]	BTLE_RX_RCVD_IRQHandler	Bluetooth Receive Data Interrupt
57	[0x00E4]	BTLE_RX_ENG_DET_IRQHandler	Bluetooth Receive Energy Detected Interrupt
58	[0X00E8]	BTLE_SFD_DET_IRQHandler	BTLE SFD Detected
59	[0x00EC]	BTLE_SFD_TO_IRQHandler	BTLE SFD Timeout
60	[0x00F0]	BTLE_GP_EVENT_IRQHandler	BTLE Timestamp
61	[0x00F4]	BTLE_CFO_IRQHandler	BTLE CFO Done
62	[0x00F8]	BTLE_SIG_DET_IRQHandler	BTLE Signal Detected
63	[0x00FC]	BTLE_AGC_EVENT_IRQHandler	BTLE AGC Event
64	[0x0100]	BTLE_RFFE_SPIM_IRQHandler	BTLE RFFE SPIM Done
65	[0x0104]	BTLE_TX_AES_IRQHandler	BTLE TX AES Done
66	[0x0108]	BTLE_RX_AES_IRQHandler	BTLE RX AES Done
67	[0x010C]	BTLE_INV_APB_ADDR_IRQHandler	BTLE Invalid APB Address
68	[0x0110]	BTLE_IQ_DATA_VALID_IRQHandler	BTLE IQ Data Valid
69	[0x0114]	WUT_IRQHandler	Wakeup Timer Interrupt
70	[0x0118]	GPIOWAKE_IRQHandler	GPIO or USB Wakeup Interrupt
71	[0x011C]	-	Reserved
72	[0x0120]	SPI0_IRQHandler	SPI Port 0 AHB Interrupt
73	[0x0124]	WDT1_IRQHandler	Watchdog Timer 1 Interrupt
74	[0x0128]	-	Reserved
75	[0x012C]	PT_IRQHandler	Pulse Train Interrupt
76	[0x0130]	SDMA0_IRQHandler	Smart DMA Interrupt
77	[0x0134]	-	Reserved
78	[0x0138]	I2C2_IRQHandler	I2C Port 2 Interrupt
79:81	[0x0138 : 0x0144]	-	Reserved

Exception (Interrupt) Number	Offset	Name	Description
82	[0x0148]	SDHC_IRQHandler	SDHC Interrupt
83	[0x014C]	OWM_IRQHandler	1-Wire Master Interrupt
84	[0x0150]	DMA4_IRQHandler	DMA4 Interrupt
85	[0x0154]	DMA5_IRQHandler	DMA5 Interrupt
86	[0x0158]	DMA6_IRQHandler	DMA6 Interrupt
87	[0x015C]	DMA7_IRQHandler	DMA7 Interrupt
88	[0x0160]	DMA8_IRQHandler	DMA8 Interrupt
89	[0x0164]	DMA9_IRQHandler	DMA9 Interrupt
90	[0x0168]	DMA10_IRQHandler	DMA10 Interrupt
91	[0x016C]	DMA11_IRQHandler	DMA11 Interrupt
92	[0x0170]	DMA12_IRQHandler	DMA12 Interrupt
93	[0x0174]	DMA13_IRQHandler	DMA13 Interrupt
94	[0x0178]	DMA14_IRQHandler	DMA14 Interrupt
95	[0x017C]	DMA15_IRQHandler	DMA15 Interrupt
97	[0x0184]	WDT2_IRQHandler	Watchdog Timer 2 Interrupt
98	[0x0188]	ECC_IRQHandler	Error Correction Coding Block Interrupt
99	[0x018C]	DVS_IRQHandler	DVS Controller Interrupt
100	[0x0190]	SIMO_IRQHandler	SIMO Controller Interrupt
101	[0x0194]	-	Reserved
102	[0x0198]	AUDIO_IRQHandler	Audio Interface Interrupt
103	[0x019C]	FLC1_IRQHandler	FLASH Controller 1 Interrupt
104:108	[0x01A0 : 0x01B0]	-	Reserved
109	[0x01B4]	HTMR0_IRQHandler	HTimer 0 Interrupt
110	[0x01B8]	HTMR1_IRQHandler	HTimer 1 Interrupt

6. General-Purpose I/O and Alternate Function Pins (GPIO)

General-purpose I/O (GPIO) pins can be individually configured to operate in a digital I/O mode or in an alternate function (AF) mode, which maps a signal associated with an enabled peripheral to that GPIO. The number of GPIO pins and the assignment of alternate functions are shown in the GPIO and Alternate Function Matrices. The GPIO supports dynamic switching between I/O mode and its alternate function modes. Configuring a pin for an alternate function supersedes its use as a digital I/O; however, the state of the GPIO can still be read through the GPIO input register.

The electrical characteristics of a GPIO pin are identical whether the pin is configured as an I/O or alternate function, except where explicitly noted in the data sheet electrical characteristics tables.

GPIO is logically divided into ports of 32 pins. Some devices and package variants may not implement all pins of a specific 32-bit GPIO port.

Each port pin has an interrupt function that can be independently enabled and configured as a level- or edge-sensitive interrupt. All GPIOs of a given port share the same interrupt vector as detailed in *GPIO Interrupt Handling*.

The features for each GPIO pin include:

- Full CMOS outputs with configurable drive strength settings.
- Input modes options:
  - High-impedance
  - Weak pullup/pulldown
  - Strong pullup/pulldown
- Output data can be from the GPIOn_OUT register or an enabled AF peripheral
- Input data can be read from the GPIOn_IN input register or the enabled peripheral
- Bit set and clear registers for efficient bit-wise write access to the pins and configuration registers
- Wake from low-power modes using edge-triggered inputs
- Selectable GPIO voltage supply
  - V_{DDIO}
  - V_{DDIOH}
- Selectable interrupt events:
  - Level triggered low
  - Level triggered high
  - Edge triggered rising edge
  - Edge triggered falling edge
  - Edge triggered rising or falling edge
- All GPIO pins default to high-impedance input mode during power-on-reset events.

6.1 Instances

Table 6-1 shows the number of GPIO available on each IC package. Some packages and part numbers do not implement all bits of a 32-bit GPIO port. Register fields corresponding to unimplemented GPIO contain indeterminate values and should not be modified.

Table 6-1: MAX32665/MAX32666 GPIO Pin Count

PACKAGE	GPIO	PINS	ALTERNATE FUNCTION MATRIX
109 WLP	GPIO0[31:0]	32	*Table 6-2*
	GPIO1[15:0]	16	*Table 6-2*
121 CTBGA	GPIO0[31:0]	32	*Table 6-2*
	GPIO1[15:0]	16	*Table 6-2*

Table 6-2 shows the alternate functions mapped to each GPIO pin.

Table 6-2: MAX32665/MAX32666 GPIO and Alternate Function Matrix, 140 WLP

GPIO Port*[pin]*	GPIO	ALTERNATE FUNCTION 1	ALTERNATE FUNCTION 2	ALTERNATE FUNCTION 3	ALTERNATE FUNCTION 4
GPIO0[0]	P0.0	SPIXF_SS0		UART2_CTS	TMR0
GPIO0[1]	P0.1	SPIXF_MOSI/SDIO0		UART2_TX	TMR1
GPIO0[2]	P0.2	SPIXF_MISO/SDIO1		UART2_RX	TMR2
GPIO0[3]	P0.3	SPIXF_SCK		UART2_RTS	TMR3
GPIO0[4]	P0.4	SPIXF_SDIO2		OWM_IO	TMR4
GPIO0[5]	P0.5	SPIXF_SDIO3		OWM_PE	TMR5
GPIO0[6]	P0.6	I2C0_SCL		SWDIO2	TMR0
GPIO0[7]	P0.7	I2C0_SDA		SWCLK2	TMR1
GPIO0[8]	P0.8	SPIXR_SS0	QSPI0_SS0	UART0_CTS	TMR2
GPIO0[9]	P0.9	SPIXR_MOSI/SDIO0	QSPI0_MOSI/SDIO0	UART0_TX	TMR3
GPIO0[10]	P0.10	SPIXR_MISO/SDIO1	QSPI0_MISO/SDIO1	UART0_RX	TMR4
GPIO0[11]	P0.11	SPIXR_SCK	QSPI0_SCK	UART0_RTS	TMR5
GPIO0[12]	P0.12	SPIXR_SDIO2	QSPI0_SDIO2	OWM_IO	TMR0
GPIO0[13]	P0.13	SPIXR_SDIO3	QSPI0_SDIO3	OWM_PE	TMR1
GPIO0[14]	P0.14	I2C1_SCL	QSPI0_SS1		TMR2
GPIO0[15]	P0.15	I2C1_SDA	QSPI0_SS2		TMR3
GPIO0[16]	P0.16	AIN0/AIN0N	QSPI1_SS0	OWM_IO	TMR4
GPIO0[17]	P0.17	AIN1/AIN0P	QSPI1_MOSI/SDIO0	OWM_PE	TMR5
GPIO0[18]	P0.18	AIN2/AIN1N	QSPI1_MISO/SDIO1		TMR0
GPIO0[19]	P0.19	AIN3/AIN1P	QSPI1_SCK		TMR1
GPIO0[20]	P0.20	AIN4/AIN2N	QSPI1_SDIO2	UART1_RX	TMR2
GPIO0[21]	P0.21	AIN5/AIN2P	QSPI1_SDIO3	UART1_TX	TMR3
GPIO0[22]	P0.22	AIN6/AIN3N	QSPI1_SS1	UART1_CTS	TMR4
GPIO0[23]	P0.23	AIN7/AIN3P	QSPI1_SS2	UART1_RTS	TMR5
GPIO0[24]	P0.24	PCM_LRCLK	QSPI2_SS0	OWM_IO	TMR0
GPIO0[25]	P0.25	PCM_DOUT	QSPI2_MOSI/SDIO0	OWM_PE	TMR1
GPIO0[26]	P0.26	PCM_DIN	QSPI2_MISO/SDIO1		TMR2
GPIO0[27]	P0.27	PCM_BCLK	QSPI2_SCK		TMR3
GPIO0[28]	P0.28	PDM_DATA2	QSPI2_SDIO2	UART2_RX	TMR4
GPIO0[29]	P0.29	PDM_DATA3	QSPI2_SDIO3	UART2_TX	TMR5
GPIO0[30]	P0.30	PDM_RX_CLK	QSPI2_SS1	UART2_CTS	TMR0
GPIO0[31]	P0.31	PDM_MCLK	QSPI2_SS2	UART2_RTS	TMR1
GPIO1[0]	P1.0	SDHC_DAT3		SDMA_TMS	PT0

GPIO Port*[pin]*	GPIO	ALTERNATE FUNCTION 1	ALTERNATE FUNCTION 2	ALTERNATE FUNCTION 3	ALTERNATE FUNCTION 4
GPIO1[1]	P1.1	SDHC_CMD		SDMA_TDO	PT1
GPIO1[2]	P1.2	SDHC_DAT0		SDMA_TDI	PT2
GPIO1[3]	P1.3	SDHC_CLK		SDMA_TCK	PT3
GPIO1[4]	P1.4	SDHC_DAT1		UART0_RX	PT4
GPIO1[5]	P1.5	SDHC_DAT2		UART0_TX	PT5
GPIO1[6]	P1.6	SDHC_WP		UART0_CTS	PT6
GPIO1[7]	P1.7	SDHC_CDN		UART0_RTS	PT7
GPIO1[8]	P1.8	QSPI0_SS0			PT8
GPIO1[9]	P1.9	QSPI0_MOSI/SDIO0			PT9
GPIO1[10]	P1.10	QSPI0_MISO/SDIO1			PT10
GPIO1[11]	P1.11	QSPI0_SCK			PT11
GPIO1[12]	P1.12	QSPI0_SDIO2		UART1_RX	PT12
GPIO1[13]	P1.13	QSPI0_SDIO3		UART1_TX	PT13
GPIO1[14]	P1.14	I2C2_SCL		UART1_CTS	PT14
GPIO1[15]	P1.15	I2C2_SDA		UART1_RTS	PT15

Each device pin can be individually configured as a GPIO or an alternate function, as shown in *Table 6-3.* The correct alternate function setting must be selected for each pin of a given multi-pin peripheral for proper operation.

Table 6-3: MAX32665/MAX32666 GPIO Pin Configuration

MODE	*GPIOn_EN*	*GPIOn_EN1*	*GPIOn_EN2*
AF1	0	0	0
AF2	0	1	0
AF3	0	0	1
AF4	0	1	1
I/O (transition to AF1)	1	0	0
I/O (transition to AF2)	1	1	0
I/O (transition to AF3)	1	0	1
I/O (transition to AF4)	1	1	1

Table 6-4 shows the configuration options for digital I/O in input mode. Refer to the data sheet for details of specific electrical characteristics.

Table 6-4: MAX32665/MAX32666 Input Mode Configuration

Input Mode	Mode Select		Pullup/Pulldown Strength	Power Supply
	GPIOn_PAD_CFG2[pin]	*GPIOn_PAD_CFG1[pin]*	*GPIOn_PS[pin]*	*GPIOn_VSSEL[pin]*
High-impedance	0	0	N/A	N/A

Input Mode	Mode Select		Pullup/Pulldown Strength	Power Supply
	GPIOn_PAD_CFG2[pin]	*GPIOn_PAD_CFG1[pin]*	*GPIOn_PS[pin]*	*GPIOn_VSSEL[pin]*
Weak Pullup to V_{DDIO} (1MΩ)	0	1	0	1
Strong Pullup to V_{DDIO} (25kΩ)	0	1	1	1
Weak Pulldown to V_{DDIO} (1MΩ)	1	0	0	0
Strong Pulldown to V_{DDIO} (25kΩ)	1	0	1	0
Reserved	1	1	N/A	N/A

Table 6-5 shows the configuration options for digital I/O in output mode. Refer to the device data sheet for details of specific electrical characteristics.

Table 6-5: MAX32665/MAX32666 Output Mode Configuration

Input Mode	Drive Strength		Power Supply
	GPIOn_DS1[pin]	*GPIOn_DS[pin]*	*GPIOn_VSSEL[pin]*
Output Drive Strength 0, V_{DDIO} Supply	0	0	0
Output Drive Strength 1, V_{DDIO} Supply	0	1	0
Output Drive Strength 2, V_{DDIO} Supply	1	0	0
Output Drive Strength 3, V_{DDIO} Supply	1	1	0
Output Drive Strength 0, V_{DDIOH} Supply	0	0	1
Output Drive Strength 1, V_{DDIOH} Supply	0	1	1
Output Drive Strength 2, V_{DDIOH} Supply	1	0	1
Output Drive Strength 3, V_{DDIOH} Supply	1	1	1

Each GPIO port is assigned a dedicated interrupt vector, as shown in the following table.

Table 6-6: MAX32665/MAX32666 GPIO Port Interrupt Vector Mapping

GPIO Interrupt Source	GPIO Interrupt Status Register	Device Specific Interrupt Vector Number	GPIO Interrupt Vector
GPIO0[31:0]	GPIO0_INT_STAT	40	GPIO0_IRQHandler
GPIO1[15:0]	GPIO1_INT_STAT	41	GPIO1_IRQHandler

6.2 Usage

6.2.1 Reset State

During a power-on-reset event, each GPIO is reset to the default input mode enabled as follows:

- The GPIO Configuration Enable bits shown in *Table 6-3* are set to I/O (transition to AF1) mode.
- Input mode enabled (*GPIOn_IN_EN[pin]* = 1).
- High impedance mode enabled (*GPIOn_PAD_CFG1[pin]* = 0, *GPIOn_PAD_CFG2[pin]* = 0).
- Output mode disabled (*GPIOn_OUT_EN[pin]* = 0)
- Interrupt disabled (*GPIOn_INT_EN[pin]* = 0)

6.2.2 Input Mode Configuration

Perform the following steps to configure one or more pins for input mode:

1. Set the GPIO Configuration Enable bits shown in *Table 6-3* to any one of the I/O mode settings.
2. Configure the electrical characteristics of the pin as desired, as shown in *Table 6-4*.
3. Enable the input buffer connected to the GPIO pin by setting *GPIOn_IN_EN[pin]* to 1.
4. Read the input state of the pin using the *GPIOn_IN[pin]* field.

6.2.3 Output Mode Configuration

Perform the following steps to configure a pin for output mode:

1. Set the GPIO Configuration Enable bits shown in *Table 6-3* to any one of the I/O mode settings.
2. Configure the electrical characteristics of the pin as desired, as shown in *Table 6-5.*
3. Set the output high or low using the *GPIOn_OUT[pin]* bit.
4. Enable the output buffer for the pin by setting *GPIOn_OUT_EN[pin]* to 1.

6.2.4 Alternate Function Configuration

Most GPIO support one or more alternate functions which are selected with the GPIO Configuration Enable bits shown in *Table 6-3*. The bits that select the AF must only be changed while the pin is in one of the I/O modes (*GPIOn_EN* = 1). The specific I/O mode must match the desired AF. For example, if a transition to AF1 is desired, first select the setting corresponding to I/O (transition to AF1). Then enable the desired mode by selecting the AF1 mode.

1. Set the GPIO Configuration Enable bits shown in *Table 6-3* to the I/O mode corresponding with the desired new AF setting. For example, select “I/O (transition to AF1)” if switching to AF1. Switching between different I/O mode settings does not affect the state or electrical characteristics of the pin.
2. Configure the electrical characteristics of the pin. See *Table 6-4* if the assigned alternate function will use the pin as an input. See *Table 6-5* if the assigned alternate function uses the pin as an output.
3. Set the GPIO Configuration Enable bits shown in *Table 6-3* to the desired alternate function.

6.3 Configuring GPIO (External) Interrupts

Each GPIO pin supports external interrupt events when the GPIO is configured for I/O mode, and the input mode is enabled. If the GPIO is configured as an alternate peripheral function, the interrupts are peripheral-controlled.

GPIO interrupts can be individually enabled and configured as an edge or level triggered independently on a pin-by-pin basis. The edge trigger can be a rising, falling, or both transitions.

Each GPIO pin has a dedicated status bit in its corresponding *GPIOn_INT_STAT* register. A GPIO interrupt will occur when the status bit transitions from 0 to 1 if the corresponding bit is set in the corresponding *GPIOn_INT_EN* register. Note that the interrupt status bit is set when the current interrupt configuration event occurs, but an interrupt is generated if explicitly enabled.

The following procedure details the steps for enabling ACTIVE mode interrupt events for a GPIO pin:

1. Disable interrupts by setting the *GPIOn_INT_EN[pin]* field to 0. This prevents any new interrupts on the pin from triggering but does not clear previously triggered (pending) interrupts. The application can disable all interrupts for a GPIO port by writing 0 to the *GPIOn_IN_EN* register. To maintain previously enabled interrupts, read the *GPIOn_IN_EN* register and save the state before setting the register to 0.
2. Clear pending interrupts by writing 1 to the *GPIOn_INT_CLR[pin]* bit.
3. Configure the pin for the desired interrupt event.
4. Set *GPIOn_INT_MOD[pin]* to select the desired interrupt.
    a. For level-triggered interrupts, the interrupt triggers on an input high (*GPIOn_INT_POL[pin]* = 0) or low level.
    b. For edge-triggered interrupts, the interrupt triggers on a transition from low to high (*GPIOn_INT_POL[pin]* = 0) or high to low (*GPIOn_INT_POL[pin]* = 1).
5. Optionally set *GPIOn_INT_DUAL_EDGE [pin]* to 1 to trigger on both the rising and falling edges of the input signal.
6. Set *GPIOn_INT_EN[pin]* to 1 to enable the interrupt for the pin.

6.3.1 GPIO Interrupt Handling

Each GPIO port is assigned its own dedicated interrupt vector, GPIOn_IRQHandler, as shown in *Table 6-6*.

To handle GPIO interrupts in your interrupt vector handler, complete the following steps:

1. Read the *GPIOn_INT_STAT* register to determine the GPIO pin that triggered the interrupt.
2. Complete interrupt tasks associated with the interrupt source pin (application-defined).
3. Clear the interrupt flag in the *GPIOn_INT_STAT* register by writing a 1 to the *GPIOn_INT_CLR* bit position that triggered the interrupt. This also clears and re-arms the edge detectors for edge-triggered interrupts.
4. Signal an end-of-interrupt to the interrupt controller by writing to the End-of-Interrupt register.
5. Return from the interrupt vector handler.

See *Table 6-8* for a summary of the available register field settings to enable GPIO interrupts.

Note: The SLEEP and DEEPSLEEP power modes can also use the GPIOn_IRQHandler interrupt vector by using the GPIOn_WAKE_EN

6.3.2 Using GPIO for Wake-up from Low Power Modes

Low-power modes support an asynchronous wake-up from edge-triggered interrupts on the GPIO ports. Level-triggered interrupts are not supported for wake-up because the system clock must be active to detect levels.

A single wake-up interrupt vector, GPIOWAKE_IRQHandler, is assigned for all pins of all GPIO ports. When the GPIO wake-up event occurs, the application software must interrogate each *GPIOn_INT_STAT* register to determine which external port pin caused the wake-up event.

Table 6-7: MAX32665/MAX32666 GPIO Wakeup Interrupt Vector

GPIO Wake Interrupt Source	GPIO Wake Interrupt Status Register	Device-Specific Interrupt Vector Number	GPIO Wake-up Interrupt Vector
GPIO0	GPIO0_INT_STAT	70	GPIOWAKE_IRQHandler
GPIO1	GPIO1_INT_STAT	70	GPIOWAKE_IRQHandler

To enable low power mode wake-up (SLEEP, DEEPSLEEP, and BACKUP) using an external GPIO interrupt, complete the following steps:

1. Clear pending interrupt flags by writing to *GPIOn_INT_CLR[pin].*
2. Activate the GPIO wake-up function by writing 1 to *PWRSEQ_LPWKEN0n[pin].*

3. Configure the power manager to use the GPIO as a wake-up source by setting *GCR_PM.gpiowken* field to 1.

See *Table 6-8* for a summary of the available register field settings to enable GPIO interrupts.

Note: The ACTIVE power mode can also use the GPIOWAKE_IRQHandler interrupt vector.

Table 6-8: MAX32665/MAX32666 Power Mode GPIO Interrupt Enable Settings

Operating Power Mode	*GPIOn_WAKE_EN*	*PWRSEQ_LPWKEN0 n*	*GCR_PM.gpiowken*	*GPIOn_INT_EN*	*Table 5-1* GPIOn_IRQ Handler	*Table 5-1* GIPOWAKE_IRQ handler
ACTIVE				X	X	
ACTIVE		X	X			X
SLEEP	X			X	X	
SLEEP*		X	X			X
DEEPSLEEP	X			X	X	
DEEPSLEEP		X	X			X
BACKUP		X	X			X
* In order to use the GPIOWAKE_IRQHandler in SLEEP mode, the software must first enter DEEPSLEEP mode, return to ACTIVE mode, and then switch to SLEEP mode.						

6.4 Registers

See *Table 3-3* for the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 6-9: GPIO Register Summary

Offset	Register	Description
[0x0000]	*GPIOn_EN*	GPIO Port n Configuration Enable Bit 0 Register
[0x0004]	*GPIOn_EN_SET*	Atomic set for *GPIOn_EN* register
[0x0008]	*GPIOn_EN_CLR*	Atomic clear for for *GPIOn_EN* register
[0x000C]	*GPIOn_OUT_EN*	GPIO Port n Output enable register
[0x0010]	*GPIOn_OUT_EN_SET*	Atomic set for *GPIOn_OUT_EN* register
[0x0014]	*GPIOn_OUT_EN_CLR*	Atomic clear for *GPIOn_OUT_EN* register
[0x0018]	*GPIOn_OUT*	GPIO Port n Output register
[0x001C]	*GPIOn_OUT_SET*	Atomic set for *GPIOn_OUT* register
[0x0020]	*GPIOn_OUT_CLR*	Atomic clear for *GPIOn_OUT* register
[0x0024]	*GPIOn_IN*	GPIO Port n Input register
[0x0028]	*GPIOn_INT_MOD*	GPIO Port n Interrupt mode register
[0x002C]	*GPIOn_INT_POL*	GPIO Port n Interrupt polarity register
[0x0030]	*GPIOn_IN_EN*	GPIO Port n Input enable register
[0x0034]	*GPIOn_INT_EN*	GPIO Port n Interrupt enable register
[0x0038]	*GPIOn_INT_EN_SET*	Atomic set for *GPIOn_INT_EN* register

Offset	Register	Description
[0x003C]	*GPIOn_INT_EN_CLR*	Atomic clear for *GPIOn_INT_EN* register
[0x0040]	*GPIOn_INT_STAT*	GPIO Port n Interrupt status register
[0x0048]	*GPIOn_INT_CLR*	Atomic clear for *GPIOn_INT_STAT* register
[0x004C]	*GPIOn_WAKE_EN*	GPIO Port n Wake from Low Power modes enable register
[0x0050]	*GPIOn_WAKE_EN_SET*	Atomic set for *GPIOn_WAKE_EN* register
[0x0054]	*GPIOn_WAKE_EN_CLR*	Atomic clear for *GPIOn_WAKE_EN* register
[0x005C]	*GPIOn_INT_DUAL_EDGE*	GPIO Port n Interrupt dual edge register
[0x0060]	*GPIOn_PAD_CFG1*	GPIO Port n Input mode selection register 1
[0x0064]	*GPIOn_PAD_CFG2*	GPIO Port n Input mode selection register 2
[0x0068]	*GPIOn_EN1*	GPIO Port n Configuration Enable Bit 1 Register
[0x006C]	*GPIOn_EN1_SET*	Atomic Set for *GPIOn_EN1* register
[0x0070]	*GPIOn_EN1_CLR*	Atomic Clear for *GPIOn_EN1* register
[0x0074]	*GPIOn_EN2*	GPIO Port n Configuration Enable Bit 2 Register
[0x0078]	*GPIOn_EN2_SET*	Atomic Set for *GPIOn_EN2* register
[0x007C]	*GPIOn_EN2_CLR*	Atomic Clear for *GPIOn_EN2* register
[0x00B0]	*GPIOn_DS*	GPIO Port n Output Drive strength selection register
[0x00B4]	*GPIOn_DS1*	GPIO Port n Output Drive strength selection register 1
[0x00B8]	*GPIOn_PS*	GPIO Port n Pulldown/Pullup strength select register
[0x00C0]	*GPIOn_VSSEL*	GPIO Port n Voltage select register

6.5 Register Details

Table 6-10: GPIO Port n Configuration Enable Bit 0 Register

GPIO Port n Configuration Enable Bit 0				GPIOn_EN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W	1	**GPIO Configuration Enable, Bit 0** This bit, in conjunction with bits in *Table 6-3* configures the corresponding device pin for digital I/O or an alternate function mode. This field can be modified directly by writing to this register or indirectly through *GPIOn_EN_SET* or *GPIOn_EN_CLR*. Some GPIO is not implemented on all devices. The bits associated with unimplemented GPIO should not be changed from their default value. This bit's setting does not affect the input and interrupt functionality of the associated pin.	

Table 6-11: GPIO Port n Configuration Enable Atomic Set Bit 0 Register

GPIO Port n Configuration Enable Atomic Set Bit 0				GPIOn_EN_SET	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W	0	**GPIO Configuration Enable Atomic Set, Bit 0** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN* register set to 1.	

Table 6-12: GPIO Port n Configuration Enable Atomic Set Bit 0 Register

GPIO Port n Configuration Enable Atomic Clear Bit 0			GPIOn_EN_CLR	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	WO	0	**GPIO Configuration Enable Atomic Clear, Bit 0** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN* register cleared to 0.

Table 6-13: GPIO Port n Output Enable Register

GPIO Port n Output Enable			GPIOn_OUT_EN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Output Enable** Set bit to 1 to enable the output driver for the corresponding GPIO pin. A bit can be enabled directly by writing to this register or indirectly through *GPIOn_OUT_EN_SET* or *GPIOn_OUT_EN_CLR*. 0: Pin is set to input mode; output driver disabled. 1: Pin is set to output mode.

Table 6-14: GPIO Port n Output Enable Atomic Set Register

GPIO Port n Output Enable Atomic Set			GPIOn_OUT_EN_SET	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	WO	0	**GPIO Output Enable Atomic Set** Writing 1 to one or more bits sets the corresponding bits in *GPIOn_OUT_EN*. 0: No effect. 1: Corresponding bits in *GPIOn_OUT_EN* set to 1.

Table 6-15: GPIO Port n Output Enable Atomic Clear Register

GPIO Port n Output Enable Atomic Clear			GPIOn_OUT_EN_CLR	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	WO	0	**GPIO Output Enable Atomic Clear** Writing 1 to one or more bits sets the corresponding bits in *GPIOn_OUT_EN*. 0: No effect. 1: Corresponding bits in *GPIOn_OUT_EN* cleared to 0.

Table 6-16: GPIO Port n Output Register

GPIO Port n Output			GPIOn_OUT	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Output** Set the corresponding output pin high or low. 0: Drive the corresponding output pin low (logic 0). 1: Drive the corresponding output pin high (logic 1).

Table 6-17: GPIO Port n Output Atomic Set Register

GPIO Port n Output Atomic Set			GPIOn_OUT_SET	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	WO	0	**GPIO Output Atomic Set** Writing 1 to one or more bits in this register sets the corresponding bits in the *GPIOn_OUT* register. Writing 1 to one or more bits sets the corresponding bits in *GPIOn_OUT*. 0: No effect. 1: Corresponding bits in *GPIOn_OUT_EN* set to 1.

Table 6-18: GPIO Port n Output Atomic Clear Register

GPIO Port n Output Atomic Clear			GPIOn_OUT_CLR	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	WO	0	**GPIO Output Atomic Clear** Writing 1 to one or more bits in this register clears the corresponding bits in the *GPIOn_OUT* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUT_EN* cleared to 0.

Table 6-19: GPIO Port n Input Register

GPIO Port n Input			GPIOn_IN	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	-	**GPIO Input** Returns the state of the input pin only if the corresponding bit in the *GPIOn_IN_EN* register is set. The state is not affected by the pin's configuration as an output or alternate function. 0: Input pin low 1: Input pin high. Note: This bit is ignored if the corresponding bit position in the GPIOn_OUT_EN register and GPIOn_OUT_EN register is not set..

Table 6-20: GPIO Port n Interrupt Mode Register

GPIO Port n Interrupt Mode			GPIOn_INT_MOD	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Interrupt Mode** Selects interrupt mode for the corresponding GPIO pin. 0: Level triggered interrupt. 1: Edge triggered interrupt. Note: This bit has no effect unless the corresponding bit in the GPIOn_INT_EN register is set.

Table 6-21: GPIO Port n Interrupt Polarity Register

GPIO Port n Interrupt Polarity			GPIOn_INT_POL	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Interrupt Polarity** Interrupt polarity selection bit for the corresponding GPIO pin. Level triggered mode (*GPIOn_INT_MOD* = 0): 0: Input low (logic 0) triggers interrupt. 1: Input high (logic 1) triggers interrupt. Edge triggered mode (*GPIOn_INT_MOD* = 1): 0: Falling edge triggers interrupt 1: Rising edge triggers interrupt. *Note: This bit has no effect unless the corresponding bit in the GPIOn_INT_EN register is set.*

Table 6-22: GPIO Port n Input Enable Register

GPIO Port n Input Enable			GPIOn_IN_EN	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	1	**GPIO Input Enable** Connects the corresponding input pad to the specified input pin for reading the pin state using the *GPIOn_IN* register. 0: Input not connected. 1: Input pin connected to the pad for reading via *GPIOn_IN* register.

Table 6-23: GPIO Port n Interrupt Enable Register

GPIO Port n Interrupt Enable			GPIOn_INT_EN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Interrupt Enable** Enable or disable the interrupt for the corresponding GPIO pin. 0: GPIO interrupts disabled. 1: GPIO interrupts enabled. *Note: Disabling a GPIO interrupt does not clear pending interrupts for the associated pin. Use the GPIOn_INT_CLR register to clear pending interrupts.*

Table 6-24: GPIO Port n Interrupt Enable Atomic Set Register

GPIO Port Interrupt Enable Atomic Set			GPIOn_INT_EN_SET	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	WO	0	**GPIO Interrupt Enable Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_INT_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_INT_EN* register set to 1.

Table 6-25: GPIO Port n Interrupt Enable Atomic Clear Register

GPIO Port Interrupt Enable Atomic Clear			GPIOn_INT_EN_CLR	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	WO	0	**GPIO Interrupt Enable Atomic Clear** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_INT_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_INT_EN* register cleared to 0.

Table 6-26: GPIO Port n Interrupt Status Register

GPIO Port Interrupt Status			GPIOn_INT_STAT	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**GPIO Interrupt Status** An interrupt is pending for the associated GPIO pin when this bit reads 1. 0: No interrupt pending for associated GPIO pin. 1: GPIO interrupts pending for associated GPIO pin. *Note: Write a 1 to the corresponding bit in the GPIOn_INT_CLR register to clear the interrupt pending status flag.*

Table 6-27: GPIO Port n Interrupt Clear Register

GPIO Port Interrupt Clear			GPIOn_INT_CLR	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1C	0	**GPIO Interrupt Clear** Write 1 to clear the associated interrupt status (GPIOn_INT_STAT). 0: No effect on the associated GPIOn_INT_STAT flag. 1: Clear the associated interrupt pending flag in the GPIOn_INT_STAT register.

Table 6-28: GPIO Port n Wakeup Enable Register

GPIO Port Wakeup Enable			GPIOn_WAKE_EN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Wakeup Enable** Enable the I/O as a wake-up from low power modes (SLEEP, DEEPSLEEP, BACKUP). 0: GPIO is not enabled as a wake-up source from low power modes. 1: GPIO is enabled as a wake-up source from low power modes.

Table 6-29: GPIO Port n Wakeup Enable Atomic Set Register

GPIO Port Wakeup Enable Atomic Set			GPIOn_WAKE_EN_SET	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Wakeup Enable Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the GPIOn_WAKE_EN register. 0: No effect. 1: Corresponding bits in GPIOn_WAKE_EN register set to 1.

Table 6-30: GPIO Port n Wakeup Enable Clear Register

GPIO Port Wakeup Enable Atomic Clear			GPIOn_WAKE_EN_CLR	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Wakeup Enable Atomic Clear** Writing 1 to one or more bits clears the corresponding bits in the GPIOn_WAKE_EN register. 0: No effect. 1: Corresponding bits in GPIOn_WAKE_EN register cleared to 0.

Table 6-31: GPIO Port n Interrupt Dual Edge Mode Select Register

GPIO Port n Interrupt Dual Edge Mode Select			GPIOn_INT_DUAL_EDGE	[0x005C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Port n Interrupt Dual-Edge Mode Select** Setting this bit triggers interrupts on both the rising and falling edges of the corresponding GPIO if the associated *GPIOn_INT_MOD* bit is set to edge-triggered. The associated polarity (*GPIOn_INT_POL*) setting has no effect when this bit is set. 0: No effect on interrupt generation. 1: Enable dual-edge mode interrupts.

Table 6-32: GPIO Port n Pullup Pulldown Selection 1 Register

GPIO Port n Pullup Pulldown Selection 1			GPIOn_PAD_CFG1	[0x0060]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Pullup Pulldown Selection 1** Input mode configuration for the associated GPIO pin. Input mode selection and selecting a weak or strong pullup or weak or strong pulldown resistor are described in *Table 6-4*.

Table 6-33: GPIO Port n Pullup Pulldown Selection 2 Register

GPIO Port Pullup Pulldown Selection 2			GPIOn_PAD_CFG2	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Pullup Pulldown Selection 2** Input mode configuration for the associated GPIO pin. Input mode selection and selecting a weak or strong pullup or weak or strong pulldown resistor are described in *Table 6-4*.

Table 6-34: GPIO Port n Configuration Enable 1 Register

GPIO Port n Configuration Enable 1			GPIOn_EN1	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Configuration Enable 1** In conjunction with bits in *Table 6-3*, this bit configures the corresponding device pin as a GPIO or an alternate function mode. Some GPIO is not implemented on all devices. The bits associated with unimplemented GPIO should not be changed from their default value. See *Table 6-1* concerning which pins are available. This bit's setting does not affect the input and interrupt functionality of the associated pin.

Table 6-35: GPIO Port n Configuration Enable 1 Atomic Set Register

GPIO Port n Configuration Enable 1 Atomic Set			GPIOn_EN1_SET	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Configuration Enable 1 Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register set to 1.

Table 6-36: GPIO Port n Configuration Enable 1 Atomic Clear Register

GPIO Port n Configuration Enable 1 Atomic Clear			GPIOn_EN1_CLR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Configuration Enable 1 Atomic Clear** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register cleared to 0.

Table 6-37: GPIO Port n Configuration Enable 2 Register

GPIO Port n Configuration Enable 2			GPIOn_EN2	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Configuration Enable 2** In conjunction with bits in *Table 6-3*, this bit configures the corresponding device pin as a GPIO or an alternate function mode. Some GPIO is not implemented on all devices. The bits associated with unimplemented GPIO should not be changed from their default value. See *Table 6-1* concerning which pins are available. This bit's setting does not affect the input and interrupt functionality of the associated pin.

Table 6-38: GPIO Port n Configuration Enable 2 Atomic Set Register

GPIO Port n Configuration Enable 2 Atomic Set			GPIOn_EN2_SET	[0x0078]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Configuration Enable 2 Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register set to 1.

Table 6-39: GPIO Port n Configuration Enable Atomic Clear Register

GPIO Port n Configuration Enable Atomic Clear			GPIOn_EN2_CLR	[0x007C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Alternate Function Select Atomic Clear** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register cleared to 0.

Table 6-40: GPIO Port n Output Drive Strength Selection Register

GPIO Port n Output Drive Strength Selection			GPIOn_DS	[0x00B0]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Output Drive Strength Selection** See *Table 6-5* for details on setting the GPIO output drive strength and other electrical characteristics.

Table 6-41: GPIO Port n Output Drive Strength Selection 1 Register

GPIO Port n Output Drive Strength Selection 1			GPIOn_DS1	[0x00B4]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Output Drive Strength Selection 1** See *Table 6-5* for details on setting the GPIO output drive strength and other electrical characteristics.

Table 6-42: GPIOn Pulldown/Pullup Strength Select Register

GPIO Port Pulldown/Pullup Strength Select			GPIOn_PS	[0x00B8]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Pulldown/Pullup Strength Select** Selects the strength of the pullup or pulldown resistor for a pin configured for input mode. 0: Weak pulldown/pullup resistor for input pin. 1: Strong pulldown/pullup resistor for input pin. *Note: Refer to the device data sheet for specific electrical characteristics of the Pulldown/Pullup resistances.*

Table 6-43: GPIOn Supply Voltage Select Register

GPIO Port n Voltage Select			GPIOn_VSSEL	[0x00C0]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Supply Voltage Select** 0: V_{DDIO} set as the pin's supply. 1: V_{DDIOH} set as the pin's supply.

7. Flash Controller (FLC)

The Flash Controller manages read, write, and erase accesses to the internal flash. It provides the following features:

- Up to 1 MB total internal flash memory
- 128 pages
- 8,192 bytes per page
- 2,048 words by 128 bits per page
- 128-bit data reads and writes
- Page erase and mass erase support
- Write protection

7.1 Instances

The device provides two instances of the FLC.

The 1MB of internal flash memory is organized as two distinct instances of 512KB, each with its own dedicated controller, for storing user application and data. These internal flash memory instances are programmable via serial wire debug interface (in-system) or directly with user application code (in-application).

The flash instances are organized as an array of pages. Each page is 2,048 words by 128 bits, or 8,192 bytes per page. *Table 7-1* shows the start address and end address for each flash instance. The internal flash memory is mapped with a start address of 0x1000 0000 and an end address of 0x100F FFFF for a total of 1MB.

Table 7-1: MAX32665/MAX32666 Internal Flash Memory Organization

Instance Number	Page Number	Size	Start Address	End Address
FLC0	1	8,192 Bytes	0x1000 0000	0x1000 1FFF
	2	8,192 Bytes	0x1000 2000	0x1000 3FFF
	3	8,192 Bytes	0x1000 4000	0x1000 5FFF
	4	8,192 Bytes	0x1000 6000	0x1000 7FFF
	.	.	.	.
	63	·	0x1007 C000·	0x1007 DFFF
	64	·	0x1007 E000	0x1007 FFFF
FLC1	1	.	0x1008 0000	0x1008 1FFF
	2	.	0x1008 2000	0x1008 3FFF
	3	.	0x1008 4000	0x1008 5FFF
	4	.	0x1008 6000	0x1008 7FFF
	.	.	.	.
	63	8,192 Bytes	0x100F C000	0x100F DFFF
	64	8,192 Bytes	0x100F E000	0x100F FFFF

7.2 Usage

Each Flash Controller manages write and erase operations for internal flash memory and provides a lock mechanism to prevent unintentional writes to the internal flash. In-application and in-system programming, page erase and mass erase operations are supported.

7.2.1 Clock Configuration

The Flash Controller requires a 1MHz peripheral clock for operation. The input clock to the Flash Controller block is the system clock, f_{SYSCLK}. Use the Flash Controller clock divisor to generate $f_{FLCnCLK} = 1\text{MHz}$, as shown in *Equation 7-1*. For the 96MHz Oscillator as the system clock, the *FLCn_CLKDIV.clkdiv* should be set to 96 (0x60).

Equation 7-1: Flash Controller Clock Frequency

$$f_{FLCnCLK} = \frac{f_{SYSCLK}}{FLCn\_CLKDIV.clkdiv} = 1MHz$$

7.2.2 Lock Protection

A locking mechanism prevents accidental memory writes and erases. All writes and erase operations require the *FLCn_CN.unlock* field be set to 0x2 prior to starting the operation. Writing any other value to this field, *FLCn_CN.unlock*, results in:

1) The flash instance remaining locked,

or,

2) The flash instance becoming locked from the unlocked state.

Note: If a write, page erase or mass erase operation is started and the unlock code was not set to 0x2, the flash controller hardware sets the access fail flag, FLCn_CN.af, to indicate an access violation occurred.

7.2.3 Flash Write Width

Each Flash Controller supports write widths of 128-bits only. The target address bits *FLCn_ADDR*[3:0] are ignored resulting in 128-bit alignment.

Table 7-2: Valid Addresses Flash Writes

	FLCn_ADDR[31:0]																															
Bit Number	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
128-bit Write	0	0	0	1	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	0	0	0

7.2.4 Flash Write

Writes to a flash location are only successful if the targeted location is already in its erased state. Perform the following steps to write to a flash memory instance:

1. If desired, enable flash controller interrupts by setting the *FLCn_INTR.access_error_ie* and *FLCn_INTR.doneie* bits.
2. Read the *FLCn_CN.pend* bit until it returns 0.
3. Configure *FLCn_CN.clkdiv* to match the SYS_CLK frequency.
4. Set the *FLCn_ADDR* register to a valid target address. Reference *Table 7-2*.
5. Set *FLCn_DATA3*, *FLCn_DATA2*, *FLCn_DATA1*, and *FLCn_DATA* to the data to write. *FLCn_DATA3* is the most significant word and *FLCn_DATA* is the least significant word. Each word of the data to write follows the little-endian format where the least significant byte of the word is stored at the lowest-numbered byte and the most significant byte is stored at the highest-numbered byte.
6. Set *FLCn_CN.unlock* to 0x2 to unlock the flash instance.
7. Set *FLCn_CN.wr* to 1. This field is automatically cleared by the Flash Controller when the write operation is finished.
8. *FLCn_INTR.done* is set by hardware when the write completes and if an error occurred, the *FLCn_INTR.af* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.
9. Set *FLCn_CN.unlock* to any value other than 0x2 to re-lock the flash instance.

Note: Code execution can occur within the same flash instance as targeted programming. If the ICC is enabled, it should either be disabled before writing to flash or flushed after writing to flash.

7.2.5 Page Erase

CAUTION: Care must be taken to not erase the page from which application code is currently executing.

Perform the following to erase a page of a flash memory instance:

1. If desired, enable flash controller interrupts by setting the *FLCn_INTR.access_error_ie* and *FLCn_INTR.doneie* bits.
2. Read the *FLCn_CN.pend* bit until it returns 0.
3. Configure *FLCn_CLKDIV.clkdiv* to match the SYS_CLK frequency.
4. Set the *FLCn_ADDR* register to an address within the target page to be erased. *FLCn_ADDR*[12:0] are ignored by the Flash Controller to ensure the address is page aligned.
5. Set *FLCn_CN.unlock* to 0x2 to unlock the flash instance.
6. Set *FLCn_CN.erase_code* to 0x55 for page erase.
7. Set *FLCn_CN.page_erase* to 1 to start the page erase operation.
8. The *FLCn_CN.pend* bit is set by the flash controller while the page erase is in progress and the *FLCn_CN.page_erase* and *FLCn_CN.pend* are cleared by the flash controller when the page erase is complete.
9. *FLCn_INTR.done* is set by hardware when the page erase completes and if an error occurred, the *FLCn_INTR.afl* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.
10. Set *FLCn_CN.unlock* to any value other than 0x2 to re-lock the flash instance.

7.2.6 Mass Erase

CAUTION: Care must be taken to not erase the flash from which application code is currently executing.

Mass erase clears the internal flash memory on an instance basis. Perform the following steps to mass erase a single flash memory instance:

1. Read the *FLCn_CN.pend* bit until it returns 0.
2. Configure *FLCn_CLKDIV.clkdiv* to match the SYS_CLK frequency.
3. Set *FLCn_CN.unlock* to 0x2 to unlock the internal flash.
4. Set *FLCn_CN.erase_code* to 0xAA for mass erase.
5. Set *FLCn_CN.me* to 1 to start the mass erase operation.
6. The *FLCn_CN.pend* bit is set by the flash controller while the mass erase is in progress and the *FLCn_CN.me* and *FLCn_CN.pend* are cleared by the flash controller when the mass erase is complete.
7. *FLCn_INTR.done* is set by the flash controller when the mass erase completes and if an error occurred, the *FLCn_INTR.af* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.
8. Set *FLCn_CN.unlock* to any value other than 0x2 to re-lock the flash instance.

7.3 Flash Error Correction Coding

The Flash Controller ECC data register *FLCn_ECC_DATA* stores the ECC bits from the last flash instance read memory location. The register contains 9 bits of ECC data of the even 128-bit flash memory location *FLCn_ECC_DATA.ecc_even* and 9 bits of ECC data of the 128-bit odd flash memory location *FLCn_ECC_DATA.ecc_odd*. These 9-bit ECC data fields are dynamic and are valid only immediately after each location read and represent the ECC for 256 bits of flash. The 128-bit even location of this even/odd pair is matched with the 128-bit odd location of the lower-valued memory address. In case of ECC error from internal flash memory read cycles, the *FLCn_ECC_DATA* can be used in conjunction with the *Table 4-70* to debug the ECC failure.

7.4 Flash Controller Registers

See *Table 3-3* for the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 7-3: Flash Controller Registers

Offset	Register Name	Access	Description
[0x0000]	*FLCn_ADDR*	R/W	Flash Controller Address Pointer Register
[0x0004]	*FLCn_CLKDIV*	R/W	Flash Controller Clock Divisor Register
[0x0008]	*FLCn_CN*	R/W	Flash Controller Control Register
[0x0024]	*FLCn_INTR*	R/W	Flash Controller Interrupt Register
[0x0028	*FLCn_ECC_DATA*	R/W	Flash Controller Error Correction Code Data
[0x0030]	*FLCn_DATA*	R/W	Flash Controller Data Register 0
[0x0034]	*FLCn_DATA1*	R/W	Flash Controller Data Register 1
[0x0038]	*FLCn_DATA2*	R/W	Flash Controller Data Register 2
[0x003C]	*FLCn_DATA3*	R/W	Flash Controller Data Register 3
[0x0040]	*FLC_ACTNL*	R/W	Flash Controller Access Control

7.5 Flash Controller Register Details

Table 7-4: Flash Controller Address Pointer Register

Flash Address Register			FLCn_ADDR	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	See Description	**Flash Address** This field contains the target address for a write operation. A valid internal flash memory address is required for all write operations. The reset value for this field is always 0x0010 0000.

Table 7-5: Flash Controller Clock Divisor Register

Flash Controller Clock Divisor Register			FLCn_CLKDIV	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved for Future Use** Do not modify this field.
7:0	clkdiv	R/W	0x60	**Flash Controller Clock Divisor** The system clock, SYS_CLK, is divided by the value in this field to generate the FLCn peripheral clock, $f_{FLCnCLK}$. The FLCn peripheral clock must equal 1MHz. The default on all forms of reset is 96 (0x60), resulting in $f_{FLCnCLK}$ = 1MHz. The FLCn peripheral clock is only used during erase and program functions and not during read functions.

Table 7-6: Flash Controller Control Register

Flash Controller Control Register			FLCn_CN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:28	unlock	R/W	0	**Flash Unlock** Write the unlock code, 0x2, prior to any flash write or erase operation to unlock the Flash. Writing any other value to this field locks the internal flash. 0x2: Flash unlock code
27:25	-	RO	-	**Reserved for Future Use** Do not modify this field.
24	pend	RO	0	**Flash Busy Flag** When this field is set, writes to all flash registers except the *FLCn_INTR* register are ignored by the Flash Controller. *Note: If the Flash Controller is busy (FLCn_CN.*pend *= 1), reads, writes and erase operations are not allowed and result in an access failure (FLCn_CN.*af *= 1).* 0: Flash idle 1: Flash busy
23:16	-	RO	-	**Reserved for Future Use** Do not modify this field.
15:8	erase_code	R/W	0	**Erase Code** Prior to an erase operation this field must be set to 0x55 for a page erase or 0xAA for a mass erase. The flash must be unlocked prior to setting the erase code. This field is automatically cleared after the erase operation is complete. 0x00: Erase disabled. 0x55: Page erase code. 0xAA: Enable mass erase.
7:3	-	RO	-	**Reserved for Future Use** Do not modify this field.
2	pge	R/W1O	0	**Page Erase** Write a 1 to this field to initiate a page erase at the address in *FLCn_ADDR.addr*. The flash must be unlocked prior to attempting a page erase, see *FLCn_CN.unlock* for details. The Flash Controller hardware clears this bit when a page erase operation is complete. 0: No page erase operation in process or page erase is complete. 1: Write a 1 to initiate a page erase. If this field reads 1, a page erase operation is in progress. *Note: This field is protected and cannot be set to 0 by application code.*

Flash Controller Control Register		FLCn_CN		[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
1	me	R/W1O	0	**Mass Erase** Write a 1 to this field to initiate a mass erase of the internal flash memory. The flash must be unlocked prior to attempting a mass erase, see *FLCn_CN.unlock* for details. The Flash Controller hardware clears this bit when the mass erase operation completes. 0: No operation 1: Initiate mass erase *Note: This field is protected and cannot be set to 0 by application code.*
0	wr	R/W1O	0	**Write** If this field reads 0, no write operation is pending for the flash. To initiate a write operation, set this bit to 1 and the Flash Controller will write to the address set in the *FLCn_ADDR* register. 0: No write operation in process or write operation complete. 1: Write 1 to initiate a write operation. If this field reads 1, a write operation is in progress. *Note: This field is protected and cannot be set to 0 by application code.*

Table 7-7: Flash Controller Interrupt Register

Flash Controller Interrupt Register		FLCn_INTR		[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	-	**Reserved for Future Use** Do not modify this field.
9	afie	R/W	0	**Flash Access Fail Interrupt Enable** Set this bit to 1 to enable interrupts on flash access failures. 0: Disabled 1: Enabled
8	doneie	R/W	0	**Flash Operation Complete Interrupt Enable** Set this bit to 1 to enable interrupts on flash operations complete. 0: Disabled 1: Enabled
7:2	-	RO	-	**Reserved for Future Use** Do not modify this field.
1	af	R/W0C	0	**Flash Access Fail Interrupt Flag** This bit is set when an attempt is made to write or erase the flash while the flash is busy or locked. Only hardware can set this bit to 1. Writing a 1 to this bit has no effect. This bit is cleared by writing a 0. 0: No access failure has occurred. 1: Access failure occurred.
0	done	R/W0C	0	**Flash Operation Complete Interrupt Flag** This flag is automatically set by hardware after a flash write or erase operation completes. 0: Operation not complete or not in process. 1: Flash operation complete.

Table 7-8: Flash Controller ECC Data Register

Flash Controller ECC Data Register			FLCn_ECC_DATA	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:25	-	RO	-	**Reserved for Future Use** Do not modify this field.
24:16	ecc_odd	RO	0	**Error Correction Code Odd Data** 9-bit ECC data recorded from the last flash read memory location of odd address of the even/odd pair of 128-bit flash memory content.
15:9	-	RO	-	**Reserved for Future Use** Do not modify this field.
8:0	ecc_even	RO	0	**Error Correction Code Even Data** 9-bit ECC data recorded from the last flash read memory location of even address of the even/odd pair of 128-bit flash memory content.

Table 7-9: Flash Controller Data Register 0

Flash Controller Data Register 0			FLCn_DATA	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data0	R/W	0	**Flash Data 0** Flash data for bits 31:0.

Table 7-10: Flash Controller Data Register 1

Flash Controller Data Register 1			FLCn_DATA1	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data1	R/W	0	**Flash Data 1** Flash data for bits 63:32

Table 7-11: Flash Controller Data Register 2

Flash Controller Data Register 2			FLCn_DATA2	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data2	R/W	0	**Flash Data 2** Flash data for bits 95:64

Table 7-12: Flash Controller Data Register 3

Flash Controller Data Register 3			FLCn_DATA3	[0x003C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data3	R/W	0	**Flash Data 3** Flash data for bits 127:96.

Table 7-13. Flash Controller Access Control Register

Flash Controller Access Control Register			FLC_ACTNL	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	acntl	R/W	0	**Access Control** When this register is written with the access control sequence, the information block can be accessed. *Information Block Flash Memory* for details.

8. External Memory

8.1 Overview

External memory can be accessed through multiple interfaces. There are three external memory interfaces, two of which are backed by 16KB of cache:

- SPI Execute-in-Place Flash (SPIXF)
  - 16KB dedicated cache
- SPI Execute-in-Place RAM (SPIXR)
  - 16KB dedicated cache
- SD/SDIO/SDHC/MMC

8.2 SPI Execute-in-Place Flash (SPIXF)

The SPIXF provides the following features:

- Up to 48MHz operation in mode 0 and 3
- Single slave select
- Four-wire mode for single-bit slave device communication
- Dual and Quad I/O supported
- Programmable SCK frequency and duty cycle
- SS assertion and de-assertion timing with respect to the leading and trailing SCK edge
- Configurable command, address, dummy, and data fields to support a variety of SPI flashes

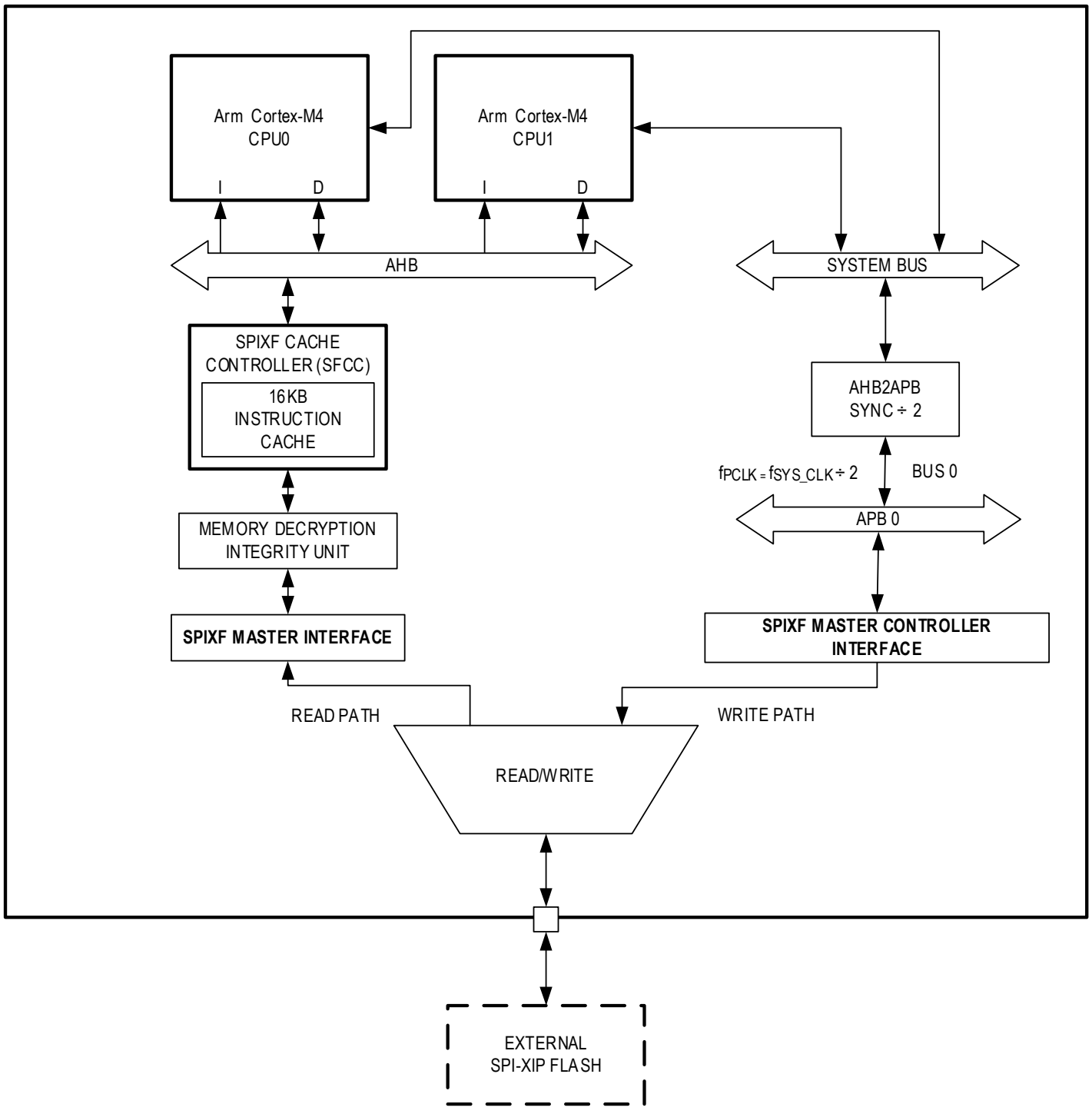
The SPIXF allows the CPU to transparently execute instructions stored in an external SPI flash. Instructions fetched using the SPIXF are cached just like instructions fetched from internal program memory. You can also use the SPIXF to access large amounts of external static data that would otherwise reside in internal data memory. This device supports a wide variety of external SPI flash memory devices.

Before using the SPI flash device, you must configure the SPIXF interface.

To prevent disclosure of intellectual property, code and data can optionally be stored in external flash in an encrypted form using the SPIXF. Generation of the encrypted data can be done through user software or with the cryptographic accelerator. The SPIXF can transparently decrypt this information in real-time using Memory Decryption Integrity Unit (MDOU) with the AES-128 algorithm in ECB mode.

The SPIXF consists of the SPIXF Master and SPIXF Master Controller, as shown below. The SPIXF Master transparently reads the external SPI flash device while the SPIXF Master Controller is used to manually write data to the external SPI flash and to configure the SPI external flash device registers.

Figure 8-1: Simplified SPIXF Block Diagram



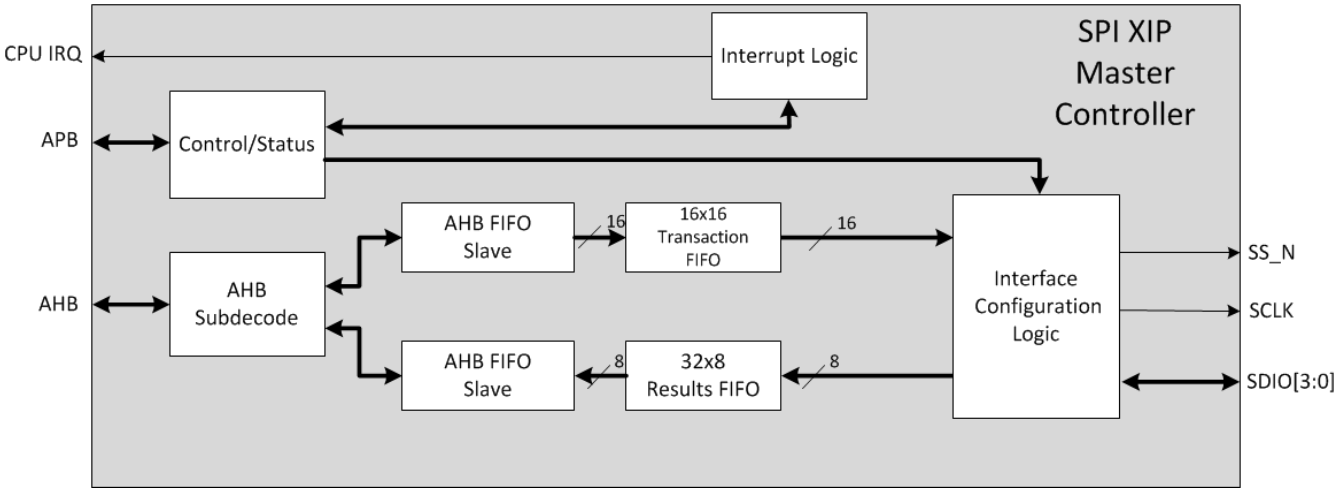
8.2.1 SPIXF Master Controller

The SPIXF Master Controller block (SPIXFC) shown in *Figure 8-2* consists of transmit and receive shift registers (supported by FIFOs) and a control unit. Communication and interface configuration is set up using the APB registers. It contains one 16×16 FIFO (Transmit FIFO) to support the transmit direction and one 32×8 FIFO (Receive FIFO) to support the receive direction. These FIFOs are accessible to software using an AHB interface to support high-speed data transfers. New data is moved automatically from the Transmit FIFO into the shift register at the start of every new SPI transfer as long as there is

data in the Transmit FIFO. At the end of every SPI transfer, data is moved from the shift register into the Receive FIFO. Status flags and interrupts are available to monitor the data levels in these FIFOs.

When an SPI transfer occurs, a multi-byte (selectable from 1 to 1024 bytes) packet is shifted out if the Transmit FIFO has configured the device to transmit using the Transmit FIFO header entry. The most significant bit is sent first. If the Transmit FIFO configures the device to receive, the device receives data most significant bit first and places each byte received into the Receive FIFO.

Figure 8-2: Simplified Block Diagram



8.2.1.1 SPIXFC Configuration

The SPIXF Master Controller shares pins with the SPIXF Master so that the SPI flash is configured for code execution or data transfer. See the SPI Pins Configuration section of the SPIXF Master for more information.

8.2.1.1.1 Configuration Modes Overview

Once the main SPIXF Master Controller clock is set up, the remainder of the configuration and operation for this block is mapped into three categories:

1. Static configuration: Performed during SPI initial setup, when the communication port is disabled, or both.
   a. *SPIXFC_GEN_CTRL* register: SCK Feedback mode, enable Transmit and Receive FIFOs
   b. *SPIXFC_SS_POL* register: Slave Select signal polarity
   c. *SPIXFC_CONFIG* register: Active slave, SPI clock polarity and phase, clock and slave select timing.
2. Dynamic configuration: Configuration required to communicate with a specific slave device, which may occur while the communication port is enabled, but the slave select is de-asserted.
   a. *SPIXFC_CONFIG* register: SPI page transfer size if using pages. See header information in *Table 8-1*.
3. Interrupt servicing: Status and control used by an application to efficiently service the SPI data transfer.
   a. *SPIXFC_FIFO_CTRL* register: Transmit and Receive FIFO monitoring levels
   b. *SPIXFC_INTFL* register: Interrupt flag bits
   c. *SPIXFC_INTEN* register: Interrupt enable bits

8.2.1.1.2 SPI Master Controller Transaction

Once the SPIXFC is configured to communicate to a specific slave, SPI transactions are initiated by writing to the SPI Transmit FIFO *SPIXF_FIFO_TX*. The FIFO is 16-bits wide and expects a 16-bit header followed by an optional payload padded out to a word boundary

The header format is shown in *Table 8-1*. If the transaction generates received data, this data is pushed into the SPI. The Receive FIFO is *SPIXF_FIFO_RX*.

A complete access sequence to an SPI device is made up of one or more transactions. In some cases, the slave select signal remains asserted across several transactions. In other cases, the access sequence defined by the slave device might require de-assertion of the slave selection in the middle of the access sequence. In general, any part of the access sequence that requires a change in direction, width, or timing requires another transaction. Interrupt logic is provided to allow efficient servicing of the SPI Master functionality by software.

Table 8-1: SPI Header Format

Name	Bits	Description	Settings
Header Type	15:14	***Reserved for Future Use.*** This header field should always be set to 0b00.	0b00
De-assert SS	13	Slave Select control.	0: Maintain assertion of slave select after transaction. 1: De-assert slave select at the completion of transaction
RFU	12:11	***Reserved for Future Use.*** This header field should always be set to 0b00.	0b00
Width	10:9	Number of SDIO pins to use for the transaction.	0x00: Single I/O mode 0x01: Dual I/O mode 0x02: Quad I/O mode 0x03: Invalid
Size	8:4	Size of transaction in terms of units.	0x00: 32 0x01: 1 0x02: 2 … 0x0F: 15
Size Units	3:2	Defines units to use when interpreting the size field. Bit transactions are available only for Tx (that is, *Direction* = 1 transactions).	0: Bits 1: Bytes 2: Pages (See the *SPIXFC_CONFIG.page_size* field for page size definition)
Direction	1:0	Defines direction of information transfer. For headers that do not define a transmission (that is, direction = None or Rx), no payload is required. Conversely, headers that do not define a reception (that is,. direction = None or Tx), result in no data pushing into the Receive FIFO.	0: None 1: Tx 2: Rx 3: Both

8.2.1.1.3 Sample SPIXF Master Controller Example

Here is an example of how to set up the Master Controller:

1. Configure the SPIXF Master Controller mode, the number of bytes per page (see *SPIXFC_CONFIG.page_size and Note below*), SCK high and low values, and Slave Select (SS) active timing and inactive timing.
    a. Example:
        i. *SPIXFC_CONFIG.mode* = 0b00 ***SPI Mode 0***
        ii. *SPIXFC_CONFIG.page_size* = 0b00 ***Page size = four bytes***
        iii. *SPIXFC_CONFIG.lo_clk* = 0b01 ***SCK low time = 1 peripheral clock period***
        iv. *SPIXFC_CONFIG.hi_clk* = 0b01 ***SCK high time = 1 peripheral clock period***

v. *SPIXFC_CONFIG*.*ss_act* = 0b10 ***SS Active time = 4 peripheral clock periods***

vi. *SPIXFC_CONFIG*.ss_*inact* = 0b10 ***SS Inactive stretch time = 8 peripheral clock periods***

2. Configure the Almost Empty and Almost Full levels for monitoring the FIFOs.
   a. Example:
      i. *SPIXFC_FIFO_CTRL*.*tx_fifo_ae_lvl* = 0x8 ***Almost Empty = 8***
      ii. *SPIXFC_FIFO_CTRL*.*rx_fifo_af_lvl* = 0xC ***Almost Full = 12***
3. Enable the Transmit and Receive FIFOs and the feedback clock.
   a. *SPIXFC_GEN_CTRL*.*rx_fifo_en* = 1 **Receive *FIFO enabled***
   b. *SPIXFC_GEN_CTRL*.*tx_fifo_en* = 1 **Transmit *FIFO enabled***
4. Write the header, then payload data to the Transmit FIFO (*SPIXF_FIFO_TX*) to send a command to configure the SPI flash for configuration or programming. More than one command may be loaded to the FIFO.
5. Initialize the SPIXF Master Controller interrupt flags by clearing the *SPIXFC_INTFL* register.
6. Set the interrupt for Transmit FIFO stalled and almost empty to ensure that the Transmit FIFO does not stall the AHB bus.
7. Write to the SPIXF Master Controller enable bit to start the transmission (*SPIXFC_GEN_CTRL*.*enable* = 1).
8. Monitor the receive stalled interrupt status flag to know when data is available in the Receive FIFO if this command receives data.
9. Monitor the *SPIXFC_INTFL*.*txrdy* for an indication that the command has been completed.
10. Repeat steps 6 through 9 to monitor multiple commands to the SPI flash if necessary.

Note: Page size is used if enabled by the SPIXF Master Controller header to configure the SPIXF Master Controller for larger transaction packet sizes (not to be confused with the SPI flash page size).

Multiple headers and payloads are written to the Transmit FIFO for consecutive execution. As an example, complete the following steps to set up the external SPI flash bus width using the SPIXF Master Controller:

1. Configure the SPIXF Master Controller to communicate with the default configuration of the external SPI flash chosen using the appropriate register and header settings.
2. Write the header and initial payload to the Transmit FIFO to send configuration of the data width (single/dual/quad) to the external SPI flash. This might require multiple commands to write status registers of the external flash device or to send specific commands.
3. Write header and payload to the Transmit FIFO to complete subsequent commands (read/write external SPI flash registers and program external SPI flash) using the new external SPI flash IO configuration.
4. Enable the SPIXF Master Controller to send commands to the external SPI flash.

8.2.1.1.4 Clock Phase and Polarity Controls

The SPIXF Master Controller and the SPIXF Master support configuration of SCK phase and polarity:

- Clock polarity (CLKPOL) selects an active low/high clock and does not affect the transfer format
- Clock phase (PHASE) selects one of two different transfer formats

The master always places data on the MOSI line a half-cycle before the SCK edge for the slave to latch the data.

Table 8-2 details the SCK phase and polarity combinations supported. See *SPIXFC_CONFIG*.*mode* register.

Table 8-2: Clock Polarity and Phase Combinations

MODE	PHASE	CLKPOL	SCK Transmit Edge	SCK Receive Edge	SCK Idle State
0	0	0	Falling	Rising	Low

MODE	PHASE	CLKPOL	SCK Transmit Edge	SCK Receive Edge	SCK Idle State
3	1	1	Falling	Rising	High

Note: Do not change the clock phase and polarity control while executing or reading from SPIXF space. Ideally, this configuration should be done before SPIXF transactions and remain unchanged while reading or executing from SPIXF space. If the clock phase and polarity need to be changed after the SPIXF slave select is active, the user must not be executing from SPIXF space, and the SPIXF block should be reset by setting GCR_RSTR1.spixip = 1.

8.2.1.1.5 Serial Clock Configuration

The output clock speed and pulse width can be controlled with the *SPIXFC_CONFIG.hi_clk* and *SPIXFC_CONFIG.lo_clk* register fields.

$$Target\ SPI\ Clock\ Hi\ Time = t_{PCLK}\ x\ \text{SPIXFC\_CONFIG}.hi\_clk$$

$$Target\ SPI\ Clock\ Lo\ Time = t_{PCLK}\ x\ \text{SPIXFC\_CONFIG}.lo_{clk}$$

where,

$$t_{PCLK} = \frac{1}{f_{PCLK}} = \frac{2}{f_{SYS\_CLK}}$$

$$Target\ SPI\ Clock\ Period = Target\ SPI\ Clock\ Hi\ Time + Target\ SPI\ Clock\ Lo\ Time$$

$$Target\ SPI\ Frequency = \frac{1}{Target\ SPI\ Clock\ Period}$$

$$Target\ SPI\ Clock\ Duty\ Cycle = \frac{Target\ SPI\ Clock\ Hi\ Time}{Target\ SPI\ Clock\ Lo\ Time + Target\ SPI\ CLock\ Hi\ Time}$$

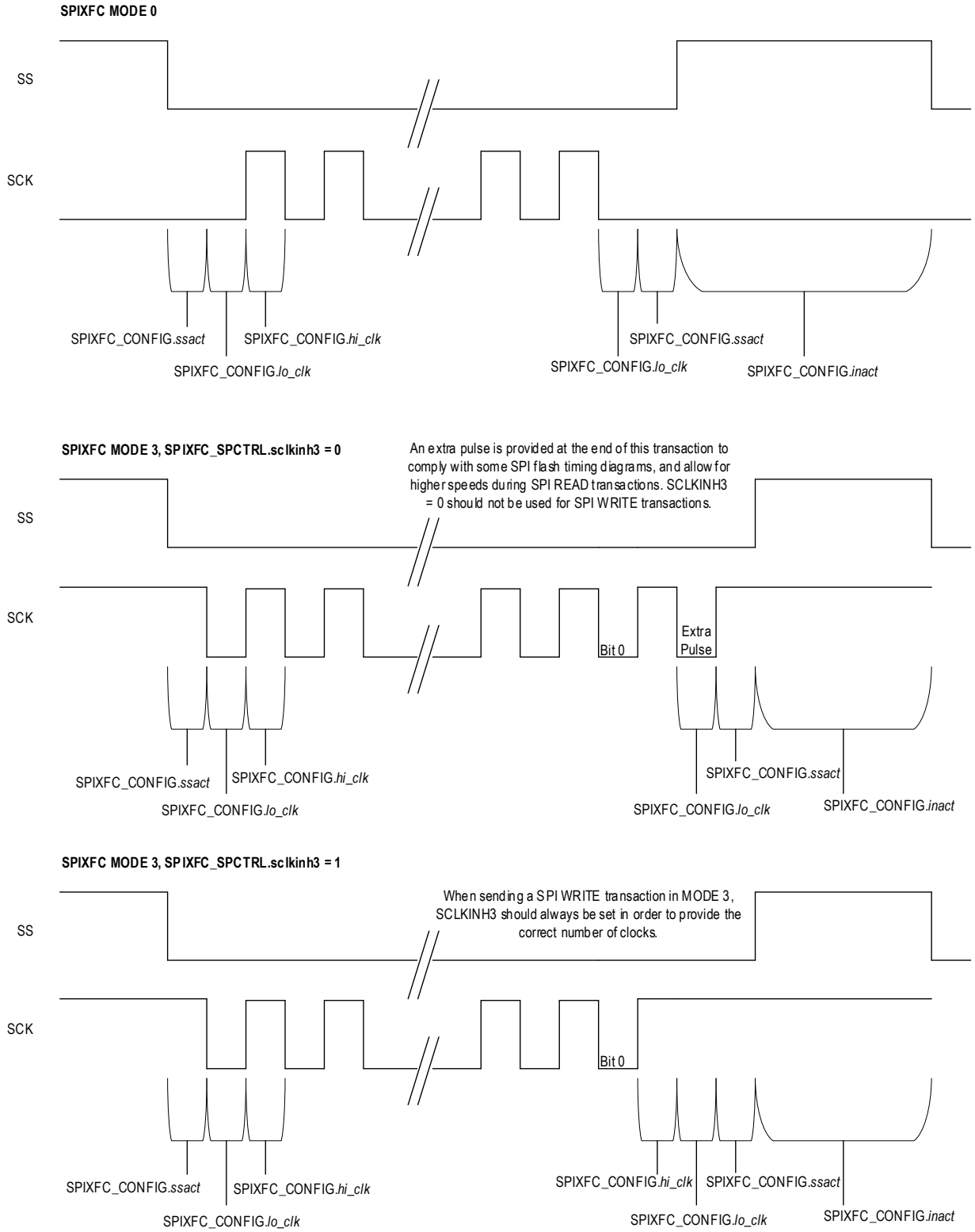
8.2.1.1.6 Slave Select Transaction Delay Configuration

The transaction delay and slave select timing with respect to the active or inactive slave select edge are determined by a combination of the following register fields:

- *SPIXFC_CONFIG.ss_act*
- *SPIXFC_CONFIG.ssiact*
- *SPIXFC_CONFIG.hi_clk*
- *SPIXFC_CONFIG.lo_clk*
- *SPIXFC_CONFIG.mode*
- *SPIXFC_SPCTRL.sclkinh3* (if in mode 3)

Automatic slave selection de-assert for the SPIXF Master Controller occurs when the Transaction Header Deassert Slave Select field is set. The Slave Select is automatically de-asserted if the SPIXF Master Controller is disabled (*SPIXFC_GEN_CTRL.enable* = 0) or *GCR_RSTR1.spixip* = 1, resetting this peripheral.

Figure 8-3: SPIXFC Transaction Delay



8.2.1.1.7 Slave Select

The SPIXF Master Controller operates with one slave device. A dedicated select pin for slave #0 is provided and controlled by hardware. Both execute-in-place and data storage are supported on slave #0.

8.2.1.1.8 Interrupts

Interrupt logic is provided to allow efficient servicing of the SPIXF Master Controller by software. You can group interrupts into the following two categories:

- Keeping the Transmit FIFO full
- Keeping the Receive FIFO empty

Programmable levels in the FIFO_CTRL register allow interrupting events to be issued if the Transmit FIFO falls below a certain level or if the Receive FIFO fills above a certain level. See the FIFO_CTRL register description for more information.

8.2.1.1.9 External SPI Flash Encryption

The user may store encrypted data or code in the external SPI flash. The SPI flash data encryption is achieved using the cryptographic accelerator to encrypt the data and the SPIXF Master Controller to write the data. Data should be encrypted using AES-128, ECB mode.

Also, the following cryptographic accelerator control bits should be set when encrypting the SPIXF address space:

- *TPU_CTRL.bsi*
- *TPU_CTRL.bso*

Setting *TPU_CTRL.src* = 0b11 selects the key stored for MDIU use in memory locations 0x4000 5080 to 0x4000 508F.

The data must be pre-processed with an address mask. 128 bits plain data blocks are XORed (^) with a 128-bit address mask to avoid patterns in encrypted data. The address mask, addr_mask below, results in 128-bit aligned addressing by masking off the lower four bits of the input address (addr_in) as follows:

```
addr_mask = addr_in & 0xFFFF FFF0
```

For encryption, the data stored in the SPI flash, data_out below, is calculated as follows:

```
data_out = AES(data_in ^ ((addr_mask << 96) | ((addr_mask+4) << 64) | \
                          ((addr_mask+8) << 32) | (addr_mask+12)))
```

where:

```
data_in = word0:word1:word2:word3 (big endian format)
```

When using the cryptographic accelerator, the input data should be loaded as follows:

```
crypto_din0 = word0 ^ (addr_mask)
crypto_din1 = word1 ^ (addr_mask+4)
crypto_din2 = word2 ^ (addr_mask+8)
crypto_din3 = word3 ^ (addr_mask+12)
```

Once the encrypted data is available (either through FIFO or through Crypto Data Output Registers [3:0]), this data may be written to SPI flash using the SPIXF Master Controller.

The available output bytes from the cryptographic accelerator should be written to SPIXF flash space as shown in *Table 8-3.*

Table 8-3: Encrypted Data Write Order to SPIX Flash Memory

Least Significant Word			Most Significant Word
crypto_dout0	crypto_dout1	crypto_dout2	crypto_dout3

8.2.1.2 SPIXF Master Controller Registers

See *Table 3-3* for the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 8-4: SPIXF Master Controller Register Offsets, Names, Access, and Description

Offset	Register	Access	Description
[0x0000]	*SPIXFC_CONFIG*	R/W	SPIXF Controller Configuration Register
[0x0004]	*SPIXFC_SS_POL*	R/W	SPIXF Controller Slave Select Polarity Register
[0x0008]	*SPIXFC_GEN_CTRL*	R/W	SPIXF Controller General Controller Register
[0x000C]	*SPIXFC_FIFO_CTRL*	R/W	SPIXF Controller FIFO Control and Status Register
[0x0010]	*SPIXFC_SPCTRL*	R/W	SPIXF Controller Special Control Register
[0x0014]	*SPIXFC_INTFL*	R/W	SPIXF Controller Interrupt Status Register
[0x0018]	*SPIXFC_INTEN*	R/W	SPIXF Controller Interrupt Enable Register

8.2.1.3 SPIXF Master Controller Register Details

Table 8-5: SPIXF Controller Configuration Register

SPIXF Controller Configuration Register			SPIXFC_CONFIG	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	R/W	0	**Reserved for Future Use** Do not modify this field.
23:20	iosmpl	R/W	0	**Sample Delay** Defines additional delay in SPI clock periods to wait before sampling SDIO input. This value must be less than or equal to the value set for *hi_clk* (for SPI modes 0 and 3). This value applies only in non-clock feedback mode (*SPIXFC_GEN_CTRL*.*sclk_fb* = 0). 0b0000: No Delay 0b0001: 1 SPI Clock delay 0b1111: 15 SPI Clock delay
19:18	ss_inact	R/W	0	**Slave Select Inactive Stretch** This field controls the number of peripheral clocks the bus is inactive between the end of a transaction (Slave Select inactive) and the start of the next transaction (Slave Select active). See section *Slave Select Transaction Delay Configuration* for detailed information. 0b00: 4 peripheral clocks 0b01: 6 peripheral clocks 0b10: 8 peripheral clocks 0b11: 12 peripheral clocks

SPIXF Controller Configuration Register			SPIXFC_CONFIG	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
17:16	ss_act	R/W	0	**Slave Select Holdoff** Controls the delay from assertion of slave select to the start of SCK pulse and the delay from the end of SCK pulses to de-assertion of slave select. See section *Slave Select Transaction Delay Configuration* for detailed information. 0b00: 0 peripheral clocks 0b01: 2 peripheral clocks 0b10: 4 peripheral clocks 0b11: 8 peripheral clocks
15:12	lo_clk	R/W	2	**SCK Low Clocks** Number of peripheral clocks that SCK is held low when SCK pulses are generated. 0: 16 peripheral clocks 1: 1 peripheral clock 2: 2 peripheral clocks 3: 3 peripheral clocks All other values: This value defines the number of peripheral clock that SCK is held low.
11:8	hi_clk	R/W	2	**SCK High Clocks** Number of peripheral clocks that SCK is held high when SCK pulses are generated. 00: 16 peripheral clocks. All other values: This value defines the number of peripheral clock that SCK is held high.
7:6	page_size	R/W	0	**Page Size** Defines the number of bytes per page for transactions that define transfers in terms of pages. 00: 4 bytes 01: 8 bytes 10: 16 bytes 11: 32 bytes
5:4	mode	R/W	0	**SPI Mode.** Defines the SPI mode. 00: SPI Mode 0. Clock Polarity = 0, Clock Phase = 0 01: Invalid 10: Invalid 11: SPI Mode 3. Clock Polarity = 1, Clock Phase = 1
3	-	R/W	0	**Reserved for Future Use** Do not modify this field.
2:0	ssel	R/W	0	**Slave Select.** Only Slave 0 is supported. 0b000: Slave 0 is selected 0b001-0b111: Invalid

Table 8-6: SPIXF Controller Slave Select Polarity Register

SPIXF Controller Slave Select Polarity Register			SPIXFC_SS_POL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	R/W	0	**Reserved for Future Use** Do not modify this field.
0	sspol_0	R/W	0	**Slave Select 0 Polarity** 0: Active Low 1: Active High

Table 8-7: SPIXF Controller General Control Register

SPIXF Controller General Control Register			SPIXFC_GEN_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	-	R/W	0	**Reserved for Future Use** Do not modify this field.
25	sckfbinv	R/W	0	**SCK Inversion** 0: Use SCK as feedback clock. 1: Use inverted SCK as feedback clock.
24	sclk_fb	R/W	0	**Enable SCK Feedback mode** 0: Disable SCK feedback mode. 1: Enable SCK feedback mode.
23	-	R/W	0	**Reserved for Future Use** Do not modify this field.
22	smplss	R/W	0	**Simple Mode Slave Select** 0: No action 1: Deassert Slave Select when simple = 1
21	simplerx	R/W	0	**Simple Receive Enable** Setting this bit to a 1 initiates a SPI transaction as defined in the Receive-Only Transaction Header when in Simple mode. 0: No action 1: Initiate SPI transaction
20	simple	R/W	0	**Simple Mode Enable** 0: Simple mode disabled 1: Simple mode enabled
19:16	bb_data_out_en	R/W	0	**Bit Bang SDIO Output Enable** Enable output of SDIO0-3 in Bit-Bang mode. bit3 = SDIO[3] bit2 = SDIO[2] bit1 = SDIO[1] bit0 = SDIO[0]
15:12	bb_data	R/W	0	**SDIO Drive value in Bit-Bang mode** Defines the output state of the SDIO outputs when in Bit-Bang mode (*SPIXFC_GEN_CTRL*.*bbmode* = 1) bit[3]: SDIO[3] bit[2]: SDIO[2] bit[1]: SDIO[1] bit[0]: SDIO[0]

SPIXF Controller General Control Register		SPIXFC_GEN_CTRL		[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
11:8	sdio_data_in	R/W	-	**SDIO Input Data Value** Returns the state of the SDIO Input values. Writes to this field have no effect. bit3: SDIO[3] bit2: SDIO[2] bit 1: SDIO[1] bit 0: SDIO[0]
7	-	R/W	0	**Reserved for Future Use** Do not modify this field.
6	sclk_dr	R/W	0	**SCK Drive and State** This bit reflects the state of the SCK. When in Bit-Bang mode (*SPIXFC_GEN_CTRL*.*bbmode* = 1), this bit is written to control the output state of the SCK. 0: SCK is 0. 1: SCK is 1.
5	-	R/W	1	**Reserved for Future Use** Do not modify this field.
4	ssdr	R/W	0	**Slave Select Drive and State** This bit reflects the state of the slave select. This accounts for the polarity as defined in the *SPIXFC_SS_POL* register. When in Bit-Bang mode, this bit is written to control the output state of the slave select. 0: Selected Slave Select Output is 0 1: Selected Slave Select Output is 1.
3	bbmode	R/W	0	**Bit-Bang Mode** 0: Disable Bit-Bang mode 1: Enable Bit-Bang mode
2	rx_fifo_en	R/W	0	**Receive FIFO Enable** Setting this bit enables the Receive FIFO. Clearing this bit disables the Receive FIFO and places it into a reset state. 0: Disable result FIFO. 1: Enable result FIFO.
1	tx_fifo_en	R/W	0	**Transmit FIFO Enable** Setting this bit to 1 enables the Transmit FIFO. Clearing this bit disables the Transmit FIFO and places it into reset state. 0: Disable Transmit FIFO. 1: Enable Transmit FIFO.
0	enable	R/W	0	**SPI Master Enable** Setting this bit to 1 enables SPI Master for processing transactions. Clearing this bit disables the SPI Master and puts the block into reset state. 0: Disable SPI Master, putting it into a reset state. 1: Enable SPI Master for processing transactions.

Table 8-8: SPIXF Controller FIFO Control and Status Register

SPIXF Controller FIFO Control and Status Register			SPIXFC_FIFO_CTRL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:30	-	R/W	00	**Reserved for Future Use** Do not modify this field.
29:24	rx_fifo_cnt	R/W	0	**Receive FIFO Entry Count** Current number of used entries (bytes) in Receive FIFO. Writes to this field are ignored.
23:21	-	R/W	0	**Reserved for Future Use** Do not modify this field.
20:16	rx_fifo_af_lvl	R/W	0x1F	**Receive FIFO Almost Full Level** The Almost Full flag is asserted when the number of used FIFO entries (bytes) exceed this value. FIFO depth is 32 bytes.
15:13	-	R/W	0	**Reserved for Future Use** Do not modify this field.
12:8	tx_fifo_cnt	R/W	0	**Transmit FIFO Entry Count** Current number of used entries (words) in the Transmit FIFO. Writes to this field are ignored.
7:4	-	R/W	0	**Reserved for Future Use** Do not modify this field.
3:0	tx_fifo_ae_lvl	R/W	0xF	**Transmit FIFO Almost Empty Level** The Almost Empty flag is asserted when the number of unused FIFO entries in words exceeds this value. FIFO depth is 16 words.

Table 8-9: SPIXF Controller Special Control Register

SPIXF Controller Special Control Register			SPIXFC_SPCTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:17	-	R/W	0	**Reserved for Future Use** Do not modify this field.
16	sclkinh3	R/W	0	**SCK Inhibit mode 3** In SPI mode 3, some SPI flash read timing diagrams show the last SCK going low prior to de-assertion. The default is to support this additional falling edge of the clock. When this bit is set, and the device is in SPI mode 3, the SPI clock is held high while slave select is de-asserted. This is to support some SPI flash write timing diagrams. 0: Allow trailing SCK low pulse prior to slave select de-assertion. 1: Inhibit trailing SCK low pulse prior to slave select de-assertion.
15:12	-	R/W	0	**Reserved for Future Use** Do not modify this field.

SPIXF Controller Special Control Register			SPIXFC_SPCTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
11:8	sdiooe	R/W	0	**SDIO Output Enable Sample Mode** Defines whether the output is enabled for each SDIO pin. Bit 11: SDIO[3] Bit 10: SDIO[2] Bit 9: SDIO[1] Bit 8: SDIO[0] 0: SDIO output disabled. 1: SDIO output enabled.
7:4	sdioout	R/W	0	**SDIO Output Value Sample Mode** Defines the values for the SDIO outputs when in Sample Mode (*SPIXFC_SPCTRL.sampl* = 1). Bit 7: SDIO[3] Bit 6: SDIO[2] Bit 5: SDIO[1] Bit 4: SDIO[0]
3:1	-	R/W	0	**Reserved for Future Use** Do not modify this field
0	sampl	R/W	0	**SDIO Sample Mode Enable** Setting this bit to a 1 enables the ability to drive SDIO outputs prior to the assertion of Slave Select. This bit must only be set when the SPIXF bus is idle and the transmit FIFO is empty. This bit is automatically cleared by hardware after the next slave select assertion. 0: Sample Mode disabled 1: Sample mode enabled

Table 8-10: SPIXF Controller Interrupt Status Register

SPIXF Controller Interrupt Status Register			SPIXFC_INTFL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:6	-	R/W1C	0	**Reserved for Future Use** Do not modify this field.
5	rx_fifo_af	R/W1C	0	**Receive FIFO Almost Full Flag.** This flag is set by hardware when the Receive FIFO is almost full as defined by *SPIXFC_FIFO_CTRL.rx_fifo_af_lvl*. 0: Receive FIFO level below the Almost Full level 1: Receive FIFO level at almost full level.
4	tx_fifo_ae	R/W1C	1	**Transmit FIFO Almost Empty Flag.** This flag is set by hardware when the Transmit FIFO is almost empty as defined by *SPIXFC_FIFO_CTRL.tx_fifo_ae_lvl*. This does not depend on block enable or the slave select value. 0: Transmit FIFO not Almost Empty 1: Transmit FIFO Almost Empty.
3	rx_done	R/W1C	0	**Receive Done Interrupt Status.** This flag is set by hardware when the Receive FIFO is not empty, and the slave select is deasserted. 0: Receive FIFO ready 1: Receive FIFO Not ready.

SPIXF Controller Interrupt Status Register			SPIXFC_INTFL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
2	tx_ready	R/W1C	0	**Transmit Ready Interrupt Status.** This flag is set by hardware when the Transmit FIFO is empty, and the slave select is deasserted. 0: Transmit FIFO not ready 1: Transmit FIFO is ready.
1	rx_stalled	R/W1C	0	**Receive Stalled Interrupt Flag.** This flag is set by hardware when the Receive FIFO is full, and the selected slave select is asserted. 0: Normal FIFO operation. 1: Stalled FIFO.
0	tx_stalled	R/W1C	0	**Transmit Stalled Interrupt Flag.** This flag is set by hardware when the Transmit FIFO is empty, and the selected slave select is asserted. 0: Normal FIFO. 1: Stalled FIFO.

Table 8-11: SPIXF Controller Interrupt Enable Register

SPIXF Controller Interrupt Enable Register			SPIXFC_INTEN	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
31:6	-	R/W	0	**Reserved for Future Use** Do not modify this field.
5	rx_fifo_af	R/W	0	**Receive FIFO Almost Full Interrupt Enable.** Setting this bit enables interrupt generation when the *SPIXFC_INTFL.rx_fifo_af* flag is set. Clearing this bit means that no interrupt is generated. 0: Disable Receive FIFO Almost Full Interrupt 1: Enable Receive FIFO Almost Full Interrupt.
4	tx_fifo_ae	R/W	1	**Transmit FIFO Almost Empty Interrupt Enable.** Setting this bit enables interrupt generation when the *SPIXFC_INTFL.tx_fifo_ae* flag is set. Clearing this bit means that no interrupt is generated. 0: Disable Transmit FIFO Almost Empty Interrupt. 1: Enable Transmit FIFO Almost Empty Interrupt.
3	rx_done	R/W	0	**Receive Done Interrupt Enable.** Setting this bit enables interrupt generation when the *SPIXFC_INTFL.rx_done* flag is set. Clearing this bit means that no interrupt is generated. 0: Disable Receive Done Interrupt. 1: Enable Receive Done Interrupt.
2	tx_ready	R/W	0	**Transmit Ready Interrupt Enable.** Setting this bit enables interrupt generation when the *SPIXFC_INTFL.tx_ready* flag is set. Clearing this bit means that no interrupt is generated. 0: Disable Transmit Ready Interrupt. 1: Enable Transmit Ready Interrupt.

SPIXF Controller Interrupt Enable Register			SPIXFC_INTEN	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
1	rx_stalled	R/W	0	**Receive Stalled Interrupt Enable.** Setting this bit enables the Receive Stalled Interrupt. Clearing this bit means that no interrupt is generated. 0: Disable Receive Stalled Interrupt. 1: Enable Receive Stalled Interrupt.
0	tx_stalled	R/W	0	**Transmit Stalled Interrupt Enable.** Setting this bit enables interrupt generation when the *SPIXFC_INTFL.tx_stalled* flag is set. Clearing this bit means that no interrupt is generated. 0: Disable Transmit Stalled Interrupt. 1: Enable Transmit Stalled Interrupt.

8.2.1.4 SPIXF Master Controller FIFO Registers

See *Table 3-4* for the SPIXF Master Controller FIFO Peripheral Base Address.

Table 8-12: SPIXF Master Controller FIFO Register Offsets, Names, Access and Description

Offset	Register	Access	Description
[0x0000]	*SPIXF_FIFO_TX*	WO	SPIXF Master Controller TX FIFO Register
[0x0004]	*SPIXF_FIFO_RX*	R	SPIXF Master Controller RX FIFO Register

8.2.1.5 SPIXF Master Controller FIFO Register Details

Table 8-13: SPIXF Master Controller TX FIFO Register

SPIXF Master Controller TX FIFO Register			SPIXF_FIFO_TX	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	txfifo	WO	0	**TX FIFO** Writes to this register are put into the TX FIFO for the SPIXF Master Controller.

Table 8-14: SPIXF Master Controller TX FIFO Register

SPIXF Master Controller RX FIFO Register			SPIXF_FIFO_RX	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	rx_fifo	R	0	**RX FIFO** Reads from this register return the data from the SPIXF Master Controller RX FIFO.

8.2.2 SPIXF Master

The SPIXF Master (SPIXF) is an AHB slave interface that is driven by a 16KB Unified Instruction and Constant cache to support cache operation. The AHB slave supports either instruction execution or data fetching from external SPI flash. This interface is accessible to software using an AHB interface to support high-speed data transfer. The address for SPI flash access is determined by the AHB access and is mapped from address 0x0800 0000 to 0x0FFF FFFF for a total addressable space of 128MB.

The command used to transfer SPI flash data is configured using software. Then, the access to SPI flash space (either code execution or data) may be performed by software. The AHB transaction initiated by the software provides address and other transaction critical parameters to control the data transfer from the external SPI flash.

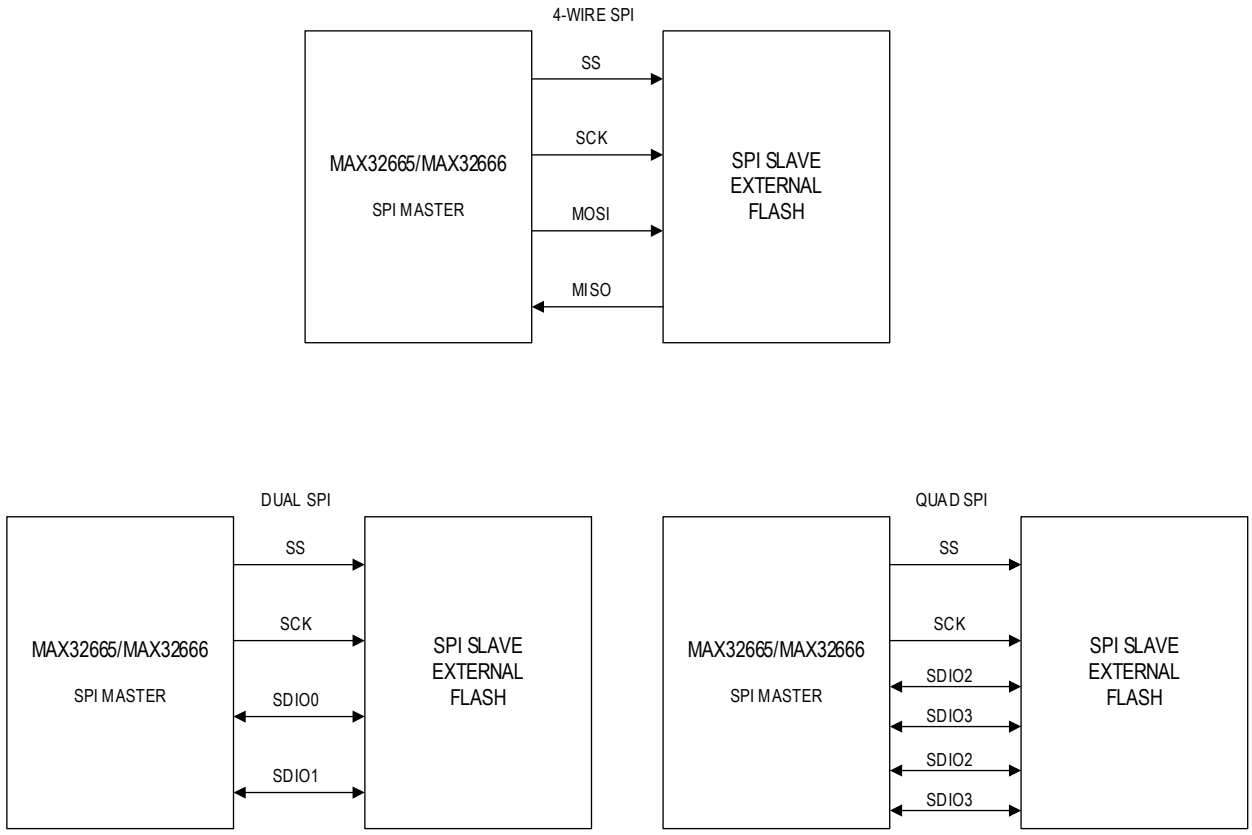
You should exercise care in choosing the correct configuration and command to support data transfer speed. The SPIXF Master provides SCK periods as fast as the AHB clock speed divided by two. The external SPI flash configuration to support data transfer rates must be performed by the SPIXF Master Controller.

8.2.2.1 SPIXF Pin Configuration

The SPIXF Master and SPIXF Master Controller use a highly-configurable, flexible, and efficient interface supporting single, dual, or quad I/O. Dedicated pins are provided to support high-speed communication. The following pin configurations are supported and shown in *Figure 8-4*:

- Four-wire SPI: SS, SCK, MOSI on SDIO0, and MISO on SDIO1
- Dual SPI: SS, SCK, SDIO0, and SDIO1
- Quad SPI: SS, SCK, SDIO0, SDIO1, SDIO2, and SDIO3

Figure 8-4: Supported SPI configuration



8.2.2.2 Slave-Select Transaction Delay Configuration

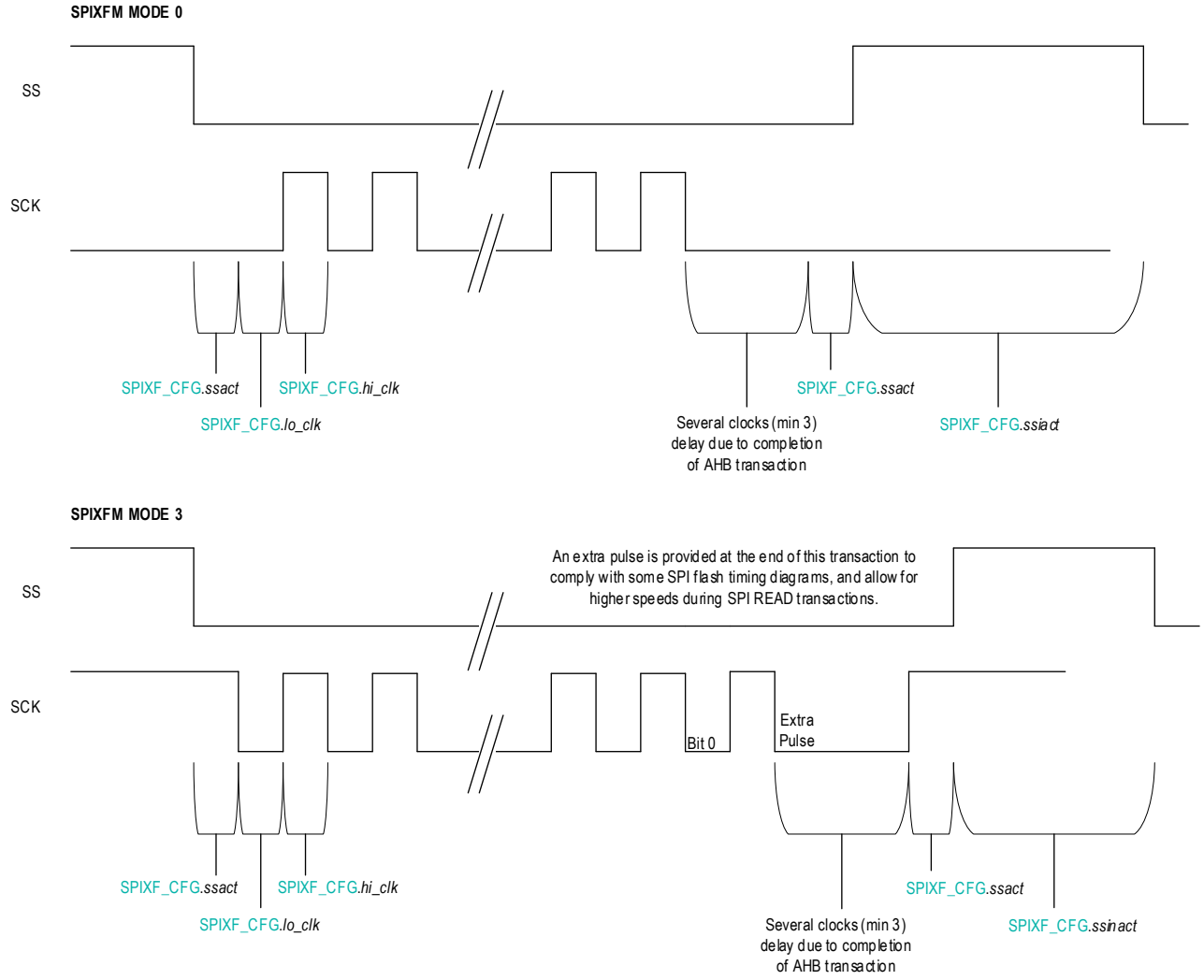
The transaction delay and clock timing with respect to the active or inactive slave-select edge is determined by a combination of the following register fields:

- *SPIXFC_CONFIG.ssact*
- *SPIXFC_CONFIG.ssinact*
- *SPIXFC_CONFIG.hi_clk*
- *SPIXFC_CONFIG.lo_clk*
- *SPIXFC_CONFIG.mode*

Automatic slave-select de-assertion only occurs when the next flash address fetched is not contiguous to the current flash address that is being read or used for execution. The SPIXF does not automatically de-assert slave selection under any other circumstance, including data read or execution of areas outside of the SPIXF space. For these cases, manual control of the slave select is provided. Invoke manual control only when running from internal memory. You can de-assert slave-select

safely by setting *GCR_RSTR1*.*spixip*. This resets the SPIXF block (including turning off decryption if previously enabled) and causes the slave select to de-assert. The SPIXF block requires reconfiguration prior to subsequent access to external SPI flash space either for execution or data reads.

Figure 8-5: SPIXF Delay Configuration



8.2.2.3 SPIXF Read Sequence Configuration and Control

Assertion of SPIXF slave select followed by the read command, then the read address. After the read address is sent 0 or more clocks are generated (called dummy bytes or mode clocks) to allow the flash to access the data being addressed. The remainder of the SPI access is read data. Sequential bytes are read until the de-assertion of SPIXF slave select.

Depending on the read command and the SPI flash configuration, the read command is sent over 1, 2, or 4 bits per clock. The same is true for the address, data, and mode/dummy clocks. Also, configure the device to eliminate the sending of the read command once the command is sent to the SPI flash device. This is enabled and disabled through special data sent during the mode or dummy period between address and read data.

8.2.2.4 Sample SPIXF Master Configuration - Execute Code

Complete the following steps to execute the SPIXF Master Configuration sample:

1. Turn on ICache XIP Clock (*GCR_PERCKCN1*.*icachexipd* = 1).
    a. The cache can be put into different power states. See *GCR_MEMCKCN* for options.

2. Configure the SPIXF Master mode, slave select polarity, slave number, and slave select timing.
    a. Example:
        i. SPI Mode 0
        ii. Slave select high
        iii. Slave #0
        iv. 1 SPI clock per 2 AHB clocks
        v. *SPIXFC_CONFIG* = 0x1104.
3. Configure the command value, the command, address, data width, and whether the address is three- or four-byte mode.
    a. Example Read command:
        i. command value = 0x03
        ii. command width = single data I/O
        iii. address bit = single data I/O,
        iv. data width = single data I/O
        v. 3-byte address mode
        vi. *SPIXFM_FETCH_CTRL* = 0x0003.
4. Configure the SPIXF mode/dummy field and the data for the mode/dummy field
    a. Example:
        i. Mode clocks = 0 (no dummy field)
        ii. *SPIXFM_MODE_CTRL* = 0x0
5. Enable the SPIXF feedback clock control.
    a. Example:
        i. SPIXF feedback clock enabled using non-inverted serial feedback clock
        ii. *SPIXFM_SCLK_FB_CTRL* = 0x0001.
6. Jump to the start of application code in SPI flash space.
    a. Example pseudo code:
        i. jump_to_external_flash = (void) (0x08000001)
        ii. jump_to_external_flash()

8.2.2.5 Clock Phase and Polarity Control

The SPIXF clock phase and polarity should match the configuration set up by the SPIXFC Master Controller. For more information about clock phase and polarity control please see *Clock Phase and Polarity Control*.

8.2.2.6 Serial Clock Feedback Mode

The SPIXF supports high-speed transfer up to 48MHz using the Serial Feedback Clock mode (*SPIXFM_SCLK_FB_CTRL .fb_en* = 1). The master output clock is routed back into the digital logic to sample incoming data from the slave. This allows for automatic alignment of the master clock to the input slave data for faster speeds. The Serial Feedback mode should not be changed while the SPIXF slave select is low. This configuration should be done prior to SPIXF transactions and remain unchanged while reading or executing from SPIXF space. If the Serial Feedback mode needs to be changed after the SPIXF slave select is low, the user must not be executing from SPIXF space, and the SPIXF block should be reset by setting *GCR_RSTR1.spixip* = 1.

8.2.2.7 External SPI Flash Decryption

If data in the SPI flash is encrypted when written, it might be transparently decrypted on read back using either code execution or data reads. Decryption is not enabled by default. Setting *SPIXFM_SEC_CTRL.decen* = 1 enables the Memory Decryption Integrity Unit (MDIU). The MDIU uses an AES-128 algorithm in ECB mode. This key is written by the user to the register file locations 0x4000 5020 to 0x4000 502F, which is automatically used by the MDIU for decryption.

See *SPIXF Master Controller* for information about data encryption for external SPI flash.

8.2.2.8 SPIXF Master Registers

See *Table 3-3* for the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.
Reserved register bits should only be written as 0.

Table 8-15: SPIXF Master Register Offsets, Names, Access and Description

Offset	Register	Access	Description
[0x0000]	*SPIXFM_CFG*	R/W	SPIXF Master Configuration Register
[0x0004]	*SPIXFM_FETCH_CTRL*	R/W	SPIXF Master Fetch Control Register
[0x0008]	*SPIXFM_MODE_CTRL*	R/W	SPIXF Master Mode Control Register
[0x000C]	*SPIXFM_MODE_DATA*	R/W	SPIXF Master Mode Data Register
[0x0010]	*SPIXFM_SCLK_FB_CTRL*	R/W	SPIXF Master SCK Feedback Control Register
[0x001C]	*SPIXFM_IO_CTRL*	R/W	SPIXF Master I/O Control Register
[0x0020]	*SPIXFM_SEC_CTRL*	R/W	SPIXF Master Memory Security Register
[0x0024]	*SPIXFM_BUS_IDLE*	R/W	SPIXF Master Bus Idle Detection

8.2.2.9 SPIXF Master Register Details

Table 8-16: SPIXF Master Configuration Register

SPIXF Master Configuration			SPIXFM_CFG	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved for Future Use** Do not modify this field.
19:18	ssiact	R/W	0	**Slave Select Inactive Timing** Controls delay from de-assertion of slave select to re-assertion of slave select for another SPI transaction. See *8.2.1.1.6*, for details on slave select transaction delay configuration. 0b00: 1 system clocks 0b01: 3 system clocks 0b10: 5 system clocks 0b11: 9 system clocks

SPIXF Master Configuration			SPIXFM_CFG	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
17:16	ssact	R/W	0	**Slave Select Active Timing** Controls delay from assertion of slave select to start of the SCK pulse and delay from the end of SCK pulses to de-assertion of slave select. See *8.2.1.1.6*, for details on slave select transaction delay configuration. 0b00: 0 system clocks 0b01: 2 system clocks 0b10: 4 system clocks 0b11: 8 system clocks
15:12	hi_clk	R/W	0b0010	**SCK High Clocks** Number of system clocks that SCK is held high when SCK pulses are generated. 0: Invalid All other values: The number of system clocks that SCK is held high.
11:8	lo_clk	R/W	0b0010	**SCK Low Clocks** Number of system clocks that SCK is held low when SCK pulses are generated. 0: Invalid All other values: The number of system clocks that SCK is held low.
7	-	R/W	0	**Reserved for Future Use** Do not modify this field.
6:4	ssel	R/W	0	**Slave Select** Only valid value is zero
3	-	R/W	0	**Reserved for Future Use** Do not modify this field.
2	sspol	R/W	1	**Slave Select Polarity** This bit controls the polarity of the slave select. 0: Slave Select active high 1: Slave Select active low
1:0	mode	R/W	0	**SPI mode** Set this field to the required SPI mode. 0b00: SPI mode 0 0b01: Reserved 0b10: Reserved 0b11: SPI mode 3

Table 8-17: SPIXF Master Fetch Control Register

SPIXF Master Fetch Control			SPIXFM_FETCH_CTRL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:17	-	R/W	0	**Reserved for Future Use** Do not modify this field.
16	four_byte_addr	R/W	0	**Four-Byte Address mode** Enables 4-byte Flash Address mode. Defaults to value as defined by parameter in instantiation. User can override. 0: 3-byte address mode 1: 4-byte address mode

SPIXF Master Fetch Control			SPIXFM_FETCH_CTRL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
15:14	-	R/W	0	**Reserved for Future Use** Do not modify this field.
13:12	data_width	R/W	0	**Data Width** Number of data I/O used to receive data. 0b00: Single SDIO 0b01: Dual SDIO 0b10: Quad SDIO 0b11: Reserved
11:10	addr_width	R/W	0	**Address Width** Number of data I/O used to send address and mode/dummy clocks. 0b00: Single SDIO 0b01: Dual SDIO 0b10: Quad SDIO 0b11: Reserved
9:8	cmd_width	R/W	0	**Command Width** Number of data I/O used to send commands. 0b00: Single SDIO 0b01: Dual SDIO 0b10: Quad SDIO 0b11: Reserved
7:0	cmdval	R/W	3	**Command Value** Command value sent to target to initiate fetching from SPI flash.

Table 8-18: SPIXF Master Mode Control Register

SPIXF Master Mode Control			SPIXFM_MODE_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	R/W	0	**Reserved for Future Use** Do not modify this field.
9	mode_send	R/W	0	**Mode Send** Setting this field ensures that the next SPI flash transaction will send the mode byte as defined in the *SPIXFM_MODE_DATA*.*data* field. When this field is set, the next SPI flash read operation exits continuous mode cleanly. This field and the *SPIXFM_MODE_CTRL*.*no_cmd* field is automatically cleared by hardware after the next SPI transaction. 0: No Action. 1: Send Mode Byte on next transaction
8	no_cmd	R/W	0	**No Command Mode** Read command sent only once after this bit is set. 0: Send read command every time SPI transaction is initiated. 1: Send read command on first transaction only and not on subsequent transactions.
7:4	-	R/W	0	**Reserved for Future Use** Do not modify this field.

SPIXF Master Mode Control			SPIXFM_MODE_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
3:0	mdclk	R/W	0	**Mode Clocks** Number of SPI clocks needed during the mode/dummy phase of fetch.

Table 8-19: SPIXF Master Mode Data Register

SPIXF Master Mode Data			SPIXFM_MODE_DATA	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	out_en	R/W	0	**Mode Output Enable** Output enable state for each corresponding data bit in *SPIXFM_MODE_DATA.data*. 0: Output enable off, I/O is tristate stated. 1: Output enable on, I/O is driving *SPIXFM_MODE_DATA.data*.
15:0	data	R/W	0	**Mode Data** Specifies the data to send with the dummy/mode clocks.

Table 8-20: SPIXF Master SCK Feedback Control Register

SPIXF Master SCK Feedback Control			SPIXFM_SCLK_FB_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	R/W	0	**Reserved for Future Use** Do not modify this field.
1	invert_en	R/W	0	**SCLK Feedback Clock Inversion.** The feedback clock can be phase selected to increase the timing margin for input data from the slave external flash device. 0: Non-inverted SCK is used for feedback clock 1: Inverted SCK is used for feedback clock
0	fb_en	R/W	1	**SCK Feedback Mode** Enable. Enable SCK feedback mode 0: Disable SCK feedback mode 1: Enable SCK feedback mode

Table 8-21: SPIXF Master I/O Control Register

SPIXF Master I/O Control			SPIXFM_IO_CTRL	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:5	-	R/W	0	**Reserved for Future Use** Do not modify this field.
4:3	pu_pd_ctrl	R/W	0b01	**IO Pullup/Pulldown Control** These bits control the pullups and pulldowns associated with all SPIXF SDIO pins. 0b00: tristate 0b01: pullup 0b10: pulldown 0b11: pullup

SPIXF Master I/O Control			SPIXFM_IO_CTRL	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
2	sdio_ds	R/W	1	**SDIO Drive Strength** This bit controls the drive strength of all SDIO pins. 0: Low Drive Strength. 1: Hi Drive Strength.
1	ss_ds	R/W	1	**Slave Select Drive Strength** This bit controls the drive strength on the dedicated slave select pin. 0: Low Drive Strength. 1: Hi Drive Strength.
0	sclk_ds	R/W	1	**SCK Drive Strength** This bit controls the drive strength on the SCK pin. 0: Low Drive Strength. 1: Hi Drive Strength.

Table 8-22: SPIXF Master Memory Security Control Register

SPIXF Master Memory Security Control			SPIXFM_SEC_CTRL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	R/W	0	**Reserved for Future Use** Do not modify this field.
1	auth_disable	R/W	0	**Integrity Enable.** 0: Integrity checking enabled. 1: Integrity checking disabled.
0	decen	R/W	0	**Decryption Enable.** 0: Disable decryption of SPIXF data. 1: Enable decryption of SPIXF data.

Table 8-23: SPIXF Master Bus Idle Detection

SPIXF Master Bus Idle Detection			SPIXFM_BUS_IDLE	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field.
15:0	busidle	R/W	0	**Bus Idle Timer Limit** A 16-bit timer will be triggered for each external access. The timer will be restarted if another access is performed before the timer expires. When the timer expires, slave select will be deactivated. This register contains the limit (expiration) value for the timer. A value of 0 will disable the bus idle detection and non-zero values enable bus idle detection. This feature is useful when fetching code out of I-cache, ROM or in SLEEP and DEEPSLEEP modes. When this number is too small, Slave Select will be deactivated on every access, which may reduce current consumption, but decreases performance.

8.3 SPI Execute-in-Place RAM (SPIXR)

The SPI Execute-in-Place RAM Master Controller (SPIXR) is an instantiation of the Quad SPI Interface with the following features:

- Four SPI modes (mode 0, 1, 2, and 3)
- Master mode only support
- Dual SPI Mode with two bidirectional serial data I/O (SDIO) lines
- High Performance Quad SPI Mode with four bidirectional SDIO lines
- Programmable Serial Clock (SCK) frequency and duty cycle with 48MHz maximum
- 32-byte Transmit FIFO, 32-byte Receive FIFO with DMA support backed by a 16KB Data Cache

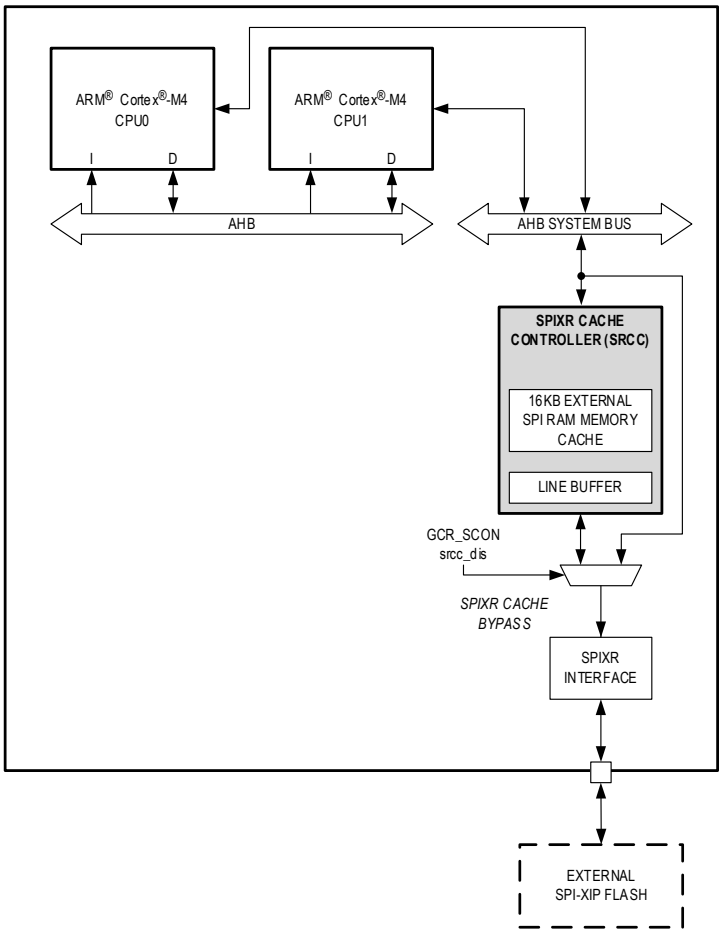
The SPIXR Master Controller allows the CPU to transparently execute instructions stored in an external SPI SRAM device. Instructions fetched using the SPIXR Master Controller are cached just like instructions fetched from internal program memory. You can also use the SPIXR Master Controller to access large amounts of external data that would otherwise reside in internal data memory.

Prior to using the SPI SRAM device, you must configure the SPIXR interface.

The command used to transfer SPI SRAM data is configured using software. Then, the access to SPI SRAM space (either code execution or data) may be performed by software. The AHB transaction initiated by the software provides address and other transaction critical parameters to control the data transfer from the external SPI SRAM.

Care should be exercised when choosing the correct configuration and command to support the speed of data transfer. The SPIXR Master Controller provides SCK periods as fast as the AHB clock speed divided by two. The external SPI SRAM configuration to support data transfer rates must be performed by the SPIXR Master Controller.

Figure 8-6: Simplified SPIXR Block Diagram



8.3.1 SPIXR Master Controller Registers

See *Table 3-3* for the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 8-24: SPIXR Master Controller Register Offsets, Names, Access, and Descriptions

Offset	Register	Access	Description
[0x0000]	*SPIXR_DATA*	R/W	SPIXR FIFO Data Register
[0x0004]	*SPIXR_CTRL1*	R/W	SPIXR Master Signals Control Register
[0x0008]	*SPIXR_CTRL2*	R/W	SPIXR Transmit Packet Size Register
[0x000C]	*SPIXR_CTRL3*	R/W	SPIXR Static Configuration Register
[0x0010]	*SPIXR_*	R/W	SPIXR Slave Select Timing Register
[0x0014]	*SPIXR_BRG_CTRL*	R/W	SPIXR Master Baud Rate Register
[0x001C]	*SPIXR_DMA*	R/W	SPIXR DMA Control Register
[0x0020]	*SPIXR_I*	R/W1C	SPIXR Interrupt Status Flags Register
[0x0024]	*SPIXR_I*	R/W	SPIXR Interrupt Enable Register
[0x0028]	*SPIXR_WAKE*	R/W1C	SPIXR Wakeup Status Flags Register
[0x002C]	*SPIXR_WAKEE*	R/W	SPIXR Wakeup Enable Register
[0x0030]	*SPIXR_STAT*	R	SPIXR Active Status Register
[0x0034]	*SPIXR_XMEM_CTRL*	R/W	SPIXR XMEM Control Register

8.3.2 SPIXR Register Details

Table 8-25: SPIXR FIFO Data Register

SPIXR FIFO Data Register			SPIXR_DATA	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**SPIXR FIFO Data** FIFO data for the SPIXR.

Table 8-26: SPIXR Master Signals Control Register

SPIXR Master Signals Control Register			SPIXR_CTRL1	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:17	-	R/W	0	**Reserved for Future Use** Do not modify this field.
16	ss	R/W	0	**Master Slave Select** This field selects the slave select pin for the SPIXR interface. 0: The slave select pin is not selected for the SPIXR. 1: The SPIXR slave select pin is used for the SPIXR.
15:9	-	R/W	0	**Reserved for Future Use** Do not modify this field.

SPIXR Master Signals Control Register			SPIXR_CTRL1	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
8	ss_ctrl	R/W	0	**Master Slave Select Control** Setting this field to 1 leaves the Slave Select signal asserted at the end of the transmission. This enables multiple transmissions to occur without the Slave Select signal being deasserted. At the completion of all transmissions with the SPIXR device, this field must be set to 0 to deassert the Slave Select line. 0: Slave Select is deasserted at the end of a transmission 1: Slave Select stays asserted at the end of a transmission
7:6	-	R/W	0	**Reserved for Future Use** Do not modify this field.
5	tx_start	R/W1AC	0	**Master Start Data Transmission** Set this field to 1 to start the transaction with the slave device. Hardware automatically clears this field after the transaction is started. 0: No SPIXR data transmission is in process. 1: Master initiates a data transmission. *Note: Ensure that all pending transactions are complete before writing a 1.* **Warning**: If the transmit FIFO is enabled, there must be at least one byte in the TX FIFO before setting this bit.
4	ssio	R/W	0	**Master Slave Select Signal Direction** This field must be set to 0 for SPIXR operation. 0: Slave Select is an output *Note: The SPIXR only operates as a SPI master in single master mode. Writing 1 to this field is invalid.*
3:2	-	R/W	0	**Reserved for Future Use** Do not modify this field.
1	mmen	R/W	0	**SPIXR Master Mode Enable** This field must be set to 1 to use the SPIXR peripheral. 1: Master Mode *Note: The SPIXR peripheral only operates in Master Mode. Writing 0 to this field is invalid.*
0	spien	R/W	0	**SPIXR Enable/Disable** Set this field to 1 to enable the SPIXR peripheral. 0: SPIXR is disabled. 1: SPIXR is enabled. *Note: Setting this field to 0 disables the SPIXR but maintains all registers and the FIFO data.*

Table 8-27: SPIXR Transmit Packet Size Register

SPIXR Transmit Packet Size Register			SPIXR_CTRL2	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	rx_num_char	R/W	0	**Number of characters to receive in RX FIFO** The number of characters in the RX FIFO. *Note: This field is only used if the SPIXR is configured for Three-Wire SPI operation, SPIXR_CTRL3.three_wire = 1.*

SPIXR Transmit Packet Size Register			SPIXR_CTRL2	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	tx_num_char	R/W	0	**Number of characters to transmit from TX FIFO** The number of characters in the TX FIFO. *Note: If the SPIXR is set to Four-wire mode, SPIXR_CTRL3.three_wire = 0, this field represents both the RX and TX FIFO character count.*

Table 8-28: SPIXR Static Configuration Register

SPIXR Static Configuration Register			SPIXR_CTRL3	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:17	-	R/W	0	**Reserved for Future Use** Do not modify this field.
16	sspol	R/W	0	**Slave Select Polarity** 0: SS is active low 1: SS is active high
15	three_wire	R/W	0	**Three-Wire Mode Enable** 0: Four-wire mode enabled (Single Mode only) 1: Three-wire mode enabled
14	-	R/W	0	**Reserved for Future Use** Do not modify this field.
13:12	data_width	R/W	0	**SPIXR Data Width** Sets the number of data lines (SDIO pins) for communication. 0: 1-data pin (Single Mode) 1: 2-data pins (Dual Mode) 2: 4-data pins (Quad Mode) 3: Reserved for Future Use
11:8	numbits	R/W	0	**Number of Bits per Character** Sets the number of bits per character for an SPIXR transaction.
7:5	-	R/W	0	**Reserved for Future Use** Do not modify this field.
4	sclk_fb_inv	R/W	0	**SCK Inverted** This field must always be set to 0 for SPIXR operation. SCK inversion for a specific mode is not supported by the SPIXR peripheral. Use the *SPIXR_CTRL3.cpol* field to set the polarity of the clock for a given mode. 0: Normal SCK output. 1: Invalid, not supported.
3:2	-	R/W	0	**Reserved for Future Use** Do not modify this field.
1	cpol	R/W	0	**Clock Polarity** Sets the SCK clock polarity for the supported modes. 0: Normal clock. Use when in SPI Mode 0 and Mode 1 1: Inverted clock. Use when in SPI Mode 2 and Mode 3 *Note: This field is set depending on the SPI Mode configuration.*

SPIXR Static Configuration Register			SPIXR_CTRL3	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
0	cpha	R/W	0	**Clock Phase** Sets the SPIXR SCK clock phase. 0: Data sampled on clock rising edge. Use when in SPI Mode 0 and Mode 2 1: Data sampled on clock falling edge. Use when in SPI Mode 1 and Mode 3 *Note: This field must be set based on the SPI Mode configuration.*

Table 8-29: SPIXR Slave Select Timing Register

SPIXR Slave Select Timing Register			SPIXR_CTRL4	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	R/W	0	**Reserved for Future Use** Do not modify this field.
23:16	ssinact	R/W	0	**SS Inactive Clock Delay** This is the time SS is inactive, and the bus is inactive between character transmission. It is the number of system clock cycles from the time a character is transmitted, and SS is inactive to the time SS is active and a new character is transmitted.
15:8	ssact2	R/W	0	**Slave Select Active After Last SCK** Number of system clock cycles that SS is active from the last SCK edge to when SS is inactive.
7:0	ssact1	R/W	0	**Slave Select Active Before SCK** Number of system clock cycles between the time SS is asserted until the first SCK edge.

Table 8-30: SPIXR Master Baud Rate Generator

SPIXR Master Baud Rate Generator Register			SPIXR_BRG_CTRL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W		**Reserved for Future Use** Do not modify this field.

SPIXR Master Baud Rate Generator Register	SPIXR_BRG_CTRL	[0x0014]

Bits	Name	Access	Reset	Description
19:16	scale	R/W	0	**System Clock to SPIXR Clock Scale Factor** Scales the system clock by 2scale to generate the internal SPIXR peripheral clock. $f_{SPIXR\_CLK} = \frac{f_{SYS\_CLK}}{2^{scale}}$
15:8	hi	R/W	0	**SCK Hi Clock Cycles Control** Setting this field to 0 disables the high duty cycle control for SCK. Setting this field to any non-zero value sets the high cycle time to: $SCK\_HIGH = hi \times SPIXR\_CLK$ *Note: If SPIXR_BRG_CTRL.scale = 0, SPIXR_BRG_CTRL.hi = 0, and SPIXR_BRG_CTRL.low = 0, character sizes of 2 and 10 bits are not supported.*
7:0	low	R/W	0	**SCK Low Clock Cycles Control** Setting this field to 0 disables the low duty cycle control for SCK. Setting this field to any non-zero value sets the high cycle time to: $SCK\_LOW = lo \times SPIXR\_CLK$ *Note: If SPIXR_BRG_CTRL.scale = 0, SPIXR_BRG_CTRL.hi = 0, and SPIXR_BRG_CTRL.low = 0, character sizes of 2 and 10 bits are not supported.*

scale	f_{SPIXR_CLK}
0	f_{SYS_CLK}
1	$\frac{f_{SYS\_CLK}}{2^1}$
2	$\frac{f_{SYS\_CLK}}{2^2}$
3	$\frac{f_{SYS\_CLK}}{2^3}$
4	$\frac{f_{SYS\_CLK}}{2^4}$
5	$\frac{f_{SYS\_CLK}}{2^5}$
6	$\frac{f_{SYS\_CLK}}{2^6}$
7	$\frac{f_{SYS\_CLK}}{2^7}$
8	$\frac{f_{SYS\_CLK}}{2^8}$
9 - 15	Reserved for Future Use

Table 8-31: SPIXR DMA Control Register

SPIXR DMA Control Register			SPIXR_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31	rx_dma_en	R/W	0	**RX DMA Enable** Enable or disable the RX DMA. 0: RX DMA is disabled. Any pending DMA requests are cleared 1: RX DMA is enabled
30	-	R/W	0	**Reserved for Future Use** Do not modify this field.
29:24	rx_fifo_cnt	R	0	**Number of Bytes in the RX FIFO** Reading this field returns the number of bytes currently in the RX FIFO
23	rx_fifo_clear	R/W1O	0	**Clear the RX FIFO** Set this field to clear the RX FIFO and all related RX FIFO flags in the *SPIXR_I* register. When cleared, the *SPIXR_I.rx_fifo_empty* flag is set by hardware. 1: Clear the RX FIFO and any pending RX FIFO flags in *SPIXR_I*. This should be done when the RX FIFO is inactive. *Note: Writing 0 has no effect.*
22	rx_fifo_en	R/W	0	**RX FIFO Enabled** Set this field to 1 to enable the RX FIFO. 0: RX FIFO disabled 1: RX FIFO enabled
21	-	R/W	0	**Reserved for Future Use** Do not modify this field.
20:16	rx_fifo_level	R/W	0	**RX FIFO Threshold Level** When the RX FIFO has more than this field, a DMA request is triggered, and the *SPIXR_I.rx_thresh* interrupt flag is set. Valid values are 0x00 to 0x1E. 0x1F is not a valid value.
15	tx_dma_en	R/W	0	**TX DMA Enable** 0: TX DMA is disabled. Any pending DMA requests are cleared. 1: TX DMA is enabled
14	-	R/W	0	**Reserved for Future Use** Do not modify this field.
13:8	tx_fifo_cnt	R	0	**Number of Bytes in the TX FIFO** Read returns the number of bytes currently in the TX FIFO
7	tx_fifo_clear	WO	0	**Clear the TX FIFO** Set this field to 1 to clear the TX FIFO and all TX FIFO related flags in the *SPIXR_I* register. When the TX FIFO is cleared, the *SPIXR_I.tx_fifo_empty* flag is set by hardware. 1: Clear the TX FIFO and any pending TX FIFO flags in *SPIXR_I*. This should be done when the TX FIFO is inactive. *Note: Writing a 0 has no effect.*
6	tx_fifo_en	R/W	0	**TX FIFO Enabled** Set to 1 to enable the TX FIFO. 0: TX FIFO disabled 1: TX FIFO enabled

SPIXR DMA Control Register			SPIXR_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
5	-	R/W	0	**Reserved for Future Use** Do not modify this field.
4:0	tx_fifo_level	R/W	0x10	**TX FIFO Threshold Level** When the TX FIFO has fewer than this field, a DMA request is triggered and the *SPIXR_I.tx_thresh* interrupt flag is set.

For all read-only fields, writes have no effect.

Table 8-32: SPIXR Interrupt Status Flag Register

SPIXR Interrupt Status Flag Register			SPIXR_IRQ	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W1C	0	**Reserved for Future Use**
15	rx_und	R/W1C	0	**RX FIFO Underrun Flag** Set when a read is attempted from an empty RX FIFO.
14	rx_ovr	R/W1C	0	**RX FIFO Overrun Flag** Set if SPI is in Slave Mode, and a write to a full RX FIFO is attempted. If the SPI is in Master Mode, this bit is not set as the SPI stalls the clock until data is read from the RX FIFO
13	tx_und	R/W1C	0	**TX FIFO Underrun Flag** Set if SPI is in Slave Mode, and a read from empty TX FIFO is attempted. If SPI is in Master Mode, this bit is not set as the SPI stalls the clock until data is written to the empty TX FIFO.
12	tx_ovr	R/W1C	0	**TX FIFO Overrun Flag** Set when a write is attempted to a full TX FIFO.
11	m_done	R/W1C	0	**Master Data Transmission Done Flag** Set if SPI is in Master Mode, and all transactions have completed.
10	-	R/W1C	0	**Reserved for Future Use** Do not modify this field.
9	abort	R/W1C	0	**Slave Mode Transaction Abort Detected Flag** Set if the SPI is in Slave Mode, and SS is deasserted before a complete character is received.
8	fault	R/W1C	0	**Multi-Master Fault Flag** Set if the SPI is in Master Mode, Multi-Master Mode is enabled, and a Slave Select input is asserted. A collision also sets this flag.
7:6	-	R/W1C	0	**Reserved for Future Use** Do not modify this field.
5	ssd	R/W1C	0	**Slave Select Deasserted Flag**
4	ssa	R/W1C	0	**Slave Select Asserted Flag**
3	rx_full	R/W1C	0	**RX FIFO Full Flag** Set when the RX FIFO is full.

SPIXR Interrupt Status Flag Register			SPIXR_IRQ	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
2	rx_thresh	R/W1C	0	**RX FIFO Threshold Level Crossed Flag** Set when the RX FIFO exceeds the value in *SPIXR_DMA.rx_fifo_level*.
1	tx_empty	R/W1C	1	**TX FIFO Empty Flag** Set when the TX FIFO is empty.
0	tx_thresh	R/W1C	0	**TX FIFO Threshold Level Crossed Flag** Set when the TX FIFO is less than the value in *SPIXR_DMA.tx_fifo_level*.

Table 8-33: SPIXR Interrupt Enable Register

SPIXR Interrupt Enable Register			SPIXR_IRQE	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field.
15	rx_und	R/W	0	**RX FIFO Underrun Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled
14	rx_ovr	R/W	0	**RX FIFO Overrun Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled
13	tx_und	R/W	0	**TX FIFO Underrun Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled
12	tx_ovr	R/W	0	**TX FIFO Overrun Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled
11	m_done	R/W	0	**Master Data Transmission Done Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled
10	-	R/W	0	**Reserved for Future Use** Do not modify this field.
9	abort	R/W	0	**Slave Mode Transaction Abort Detected Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled
8	fault	R/W	0	**Multi-Master Fault Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled
7:6	-	R/W	0	**Reserved for Future Use** 1: Interrupt enabled 0: Interrupt disabled
5	ssd	R/W	0	**Slave Select Deasserted Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled

SPIXR Interrupt Enable Register			SPIXR_IRQE	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
4	ssa	R/W	0	**Slave Select Asserted Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled
3	rx_full	R/W	0	**RX FIFO Full Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled
2	rx_thresh	R/W	0	**RX FIFO Threshold Level Crossed Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled
1	tx_empty	R/W	1	**TX FIFO Empty Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled
0	tx_thresh	R/W	0	**TX FIFO Threshold Level Crossed Interrupt Enable** 1: Interrupt enabled 0: Interrupt disabled

Table 8-34: SPIXR Wakeup Flag Register

SPIXR Wakeup Flag Register			SPIXR_WAKE	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W1C	0	**Reserved for Future Use** Do not modify this field.
3	rx_full	R/W1C	0	**Wake on RX FIFO Full Flag** If set, the RX FIFO Full condition caused the wake event.
2	rx_thresh	R/W1C	0	**Wake on RX FIFO Threshold Level Crossed Flag** If set, the RX FIFO Threshold Level Crossed condition caused the wake event.
1	tx_empty	R/W1C	0	**Wake on TX FIFO Empty Flag** If set, the TX FIFO empty condition caused the wake event.
0	tx_thresh	R/W1C	0	**Wake on TX FIFO Threshold Level Crossed Flag** If set, the TX FIFO threshold level crossed caused the wake event.

Table 8-35: SPIXR Wakeup Enable Register

SPIXR Wakeup Enable Register			SPIXR_WAKEE	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved for Future Use** Do not modify this field.
3	rx_full	R/W	0	**Wake on RX FIFO Full Enable** Set to 1 to wake up the device when this RX FIFO is full. 0: Wakeup Disabled for this condition. 1: Wakeup Enabled for this condition.

SPIXR Wakeup Enable Register			SPIXR_WAKEE	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
2	rx_thresh	R/W	0	**Wake on RX FIFO Threshold Level Crossed Enable** Set to 1 to wake up the device when this RX FIFO is full. 0: Wakeup Disabled for this condition. 1: Wakeup Enabled for this condition.
1	tx_empty	R/W	0	**Wake on TX FIFO Empty Enable** Set to 1 to wake up the device when this RX FIFO is full. 0: Wakeup Disabled for this condition. 1: Wakeup Enabled for this condition.
0	tx_thresh	R/W	0	**Wake on TX FIFO Threshold Level Crossed Enable** Set to 1 to wake up the device when this RX FIFO is full. 0: Wakeup Disabled for this condition. 1: Wakeup Enabled for this condition.

Table 8-36: SPIXR Active Status Register

SPIXR Active Status Register			SPIXR_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	R	0	**Reserved for Future Use** Do not modify this field.
0	busy	R	0	**SPI Active Status** This field returns the status of the SPIXR communications. Hardware sets and clears this field automatically when SPI communications are active or complete. 0: SPI is not active. Cleared when the last character is sent. 1: SPI is active. Set when transmit starts.

Table 8-37: SPIXR External Memory Control Register

SPIXR External Memory Control Register			SPIXR_XMEM_CTRL	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31	xmem_en	R/W	0	**Enable External Memory** 0: XMEM disabled 1: XMEM enabled
30:24	-	R/W	0	**Reserved for Future Use** Do not modify this field.
23:16	dummy_clk	R/W	0	**Number of dummy characters between address phase and read data from the external memory.** 0: No delay between address and read data 1 to 255 delay number of characters
15:8	wr_cmd	R/W	0	**Write command to be received at the external memory** Vendor specific value
7:0	rd_cmd	R/W	0	**Read command to be received at the external memory** Vendor specific value

8.4 SPIXR Cache Controller (SRCC)

The SPIXR Cache Controller is an AHB block that has multiple interfaces. The address and data interface is connected to the AHB and the SRCC register interface is connected via the APB.

The SRCC is a 16KB 2-way set-associative cache. It operates with the LRU replacement policy and has write-through implementation used for caching instructions and data from an external SPI-XiP RAM device. The SRCC includes tag RAM, cache RAM and a line fill buffer as shown in *Figure 4-7.* Write allocate and critical word first are options controlled by the application. Each cache line is 256-bits wide with the lower 5-bits of the address used as the cache line index. The SRCC uses tag cache RAM with 8-bits of the address index and a 5-bit line offset to access the tag cache RAM. 16-bits of the address are stored in tag RAM for hit/miss checking enabling the SRCC to access up to 512MB of external memory. The SRCC interfaces to the address range of 0x8000 0000 to 0x9FFF FFFF for a maximum of 512MB external.

8.4.1 Features

- 2-way set associative, LRU (Least-Recently Used) replacement policy
- Write-no-allocate with option to Write-allocate
- Write-through
- Read critical word first and streaming
- 512MB addressable range
- 16KB size

8.4.2 Enabling the SRCC

Enable the SRCC as follows:

1. Set the *GCR_SCON.dcache_dis* field to 0.
2. Set the *SRCC_CACHE_CTRL.cache_en* field to 1.

Once enabled, the cache is empty and begins filling when a read from or write to (if write allocate is enabled) the external memory is performed.

After a Power-On-Reset event, the cache tag RAM is cleared by hardware ensuring that no corrupted data is accessed from the initial cache read.

8.4.3 Disabling the SRCC

Disabling the SRCC cache automatically invalidates the cache contents. All access to the external memory while the SRCC cache is disabled are performed using the Line Buffer.

Disable the SRCC by setting *SRCC_CACHE_CTRL.cache_en* to 0.

The SRCC cache and Line Buffer can both be bypassed by setting *GCR_SCON.srcc_dis* to 1. Bypassing the SRCC enables direct access to the external memory from the application software.

8.4.4 SRCC Registers

See *Table 3-3* for the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 8-38: External Memory Cache Controller Register Addresses and Descriptions

Offset	Register Name	Access	Description
[0x0000]	*SRCC_CACHE_ID*	R	Cache ID Register
[0x0004]	*SRCC_MEMCFG*	R	Cache Memory Size Register
[0x0100]	*SRCC_CACHE_CTRL*	R/W	Cache Control Register
[0x0700]	*SRCC_INVALIDATE*	WO	Invalidate Register

8.4.5 SRCC Register Details

Table 8-39: SRCC Cache ID Register

SRCC Cache ID Register			SRCC_CACHE_ID	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R	-	**Reserved for Future Use** Do not modify this field.
15:10	cchid	R	-	**Cache ID** Returns the Cache ID for this Cache instance.
9:6	partnum	R	-	**Cache Part Number** Returns the part number indicator for this Cache instance.
5:0	relnum	R	-	**Cache Release Number** Returns the release number for this Cache instance.

Table 8-40: SRCC Memory Size Register

SRCC Memory Size Register			SRCC_MEMCFG	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	memsz	R	-	**Addressable Memory Size** Indicates the size of addressable memory by this cache controller instance in 128KB units.
15:0	cchsz	R	-	**Cache Size** Returns the size of the cache RAM memory in 1KB units. 16: 16KB Cache RAM

Table 8-41: SRCC Cache Control Register

SRCC Cache Control Register			SRCC_CACHE_CTRL	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
31:17	-	R/W	-	**Reserved for Future Use** Do not modify this field.
16	cache_rdy	R	-	**Ready** This field is cleared by hardware anytime the cache as a whole is invalidated (including a Power-On Reset event). Hardware automatically sets this field to 1 when the invalidate operation is complete and the cache is ready. 0: Cache Invalidate in process. 1: Cache is ready. *Note: While this field reads 0, the cache is bypassed and reads come directly from the line fill buffer.*
15:3	-	R/W	-	**Reserved for Future Use** Do not modify this field.

SRCC Cache Control Register			SRCC_CACHE_CTRL	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
2	cwfst_dis	R/W	0	**Critical Word First (CWF) Disable** Setting this field to 1 disables Critical Word First operation. When CWF is disabled, the cache fills the cache line before sending the data to the Arm Cortex core. When CWF is enabled, any data fetch that results in a cache miss immediately sends the data read to the Arm Cortex core prior to filling the cache line. 0: Enable Critical Word First. 1: Critical Word First Disabled. *Note: This field is only writable when the EMCC is disabled (SRCC_CACHE_CTRL.cache_en = 0).*
1	write_alloc	R/W	0	**Write Allocate Enable** Set this field to enable write allocate for the cache. When this is enabled, writes to the memory update the external memory and the cache line associated with the write is filled from the external memory. Disabling write allocate, default mode, performs a write to the external memory on any write operation, but the associated cache line is not refilled. When disabled, writes to successive memory locations are more efficient. 0: Write allocate disabled (default) 1: Write allocate enabled. *Note: The EMCC is a write-through cache resulting in any write to the external memory performing an immediate write to the external device.*
0	cache_en	R/W	0	**Enable** Set this field to 1 to enable the cache. Setting this field to 0 automatically invalidates the cache contents. When this cache is disabled, reads are handled by the line fill buffer. 0: Disable Cache 1: Enable Cache

Table 8-42: SRCC Invalidate Register

SRCC Invalidate Register			SRCC_INVALIDATE	[0x0700]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	WO	-	**Invalidate** Any write to this register of any value invalidates the cache.

8.5 Secure Digital Host Controller

The Secure Digital Host Controller (SDHC) provides an interface between the AHB and Embedded MultiMediaCards (e.MMCs), Secure Digital I/O (SDIO) cards, Standard Capacity SD Memory Cards and High-Capacity SD Memory Cards. The SDHC handles the SDIO/SD protocol at the transmission level, packing data, adding cyclic redundancy check (CRC), Start/End bit, and checking for transaction format correctness. Details of the SD communication and protocol are not part of the scope of this document. The MAX32665/MAX32666 SDHC only supports a single SD card.

SD memory card and SDIO card specifications are available at *https://www.sdcard.org*.

The e.MMC specifications are available from JEDEC at *http://www.jedec.org*.

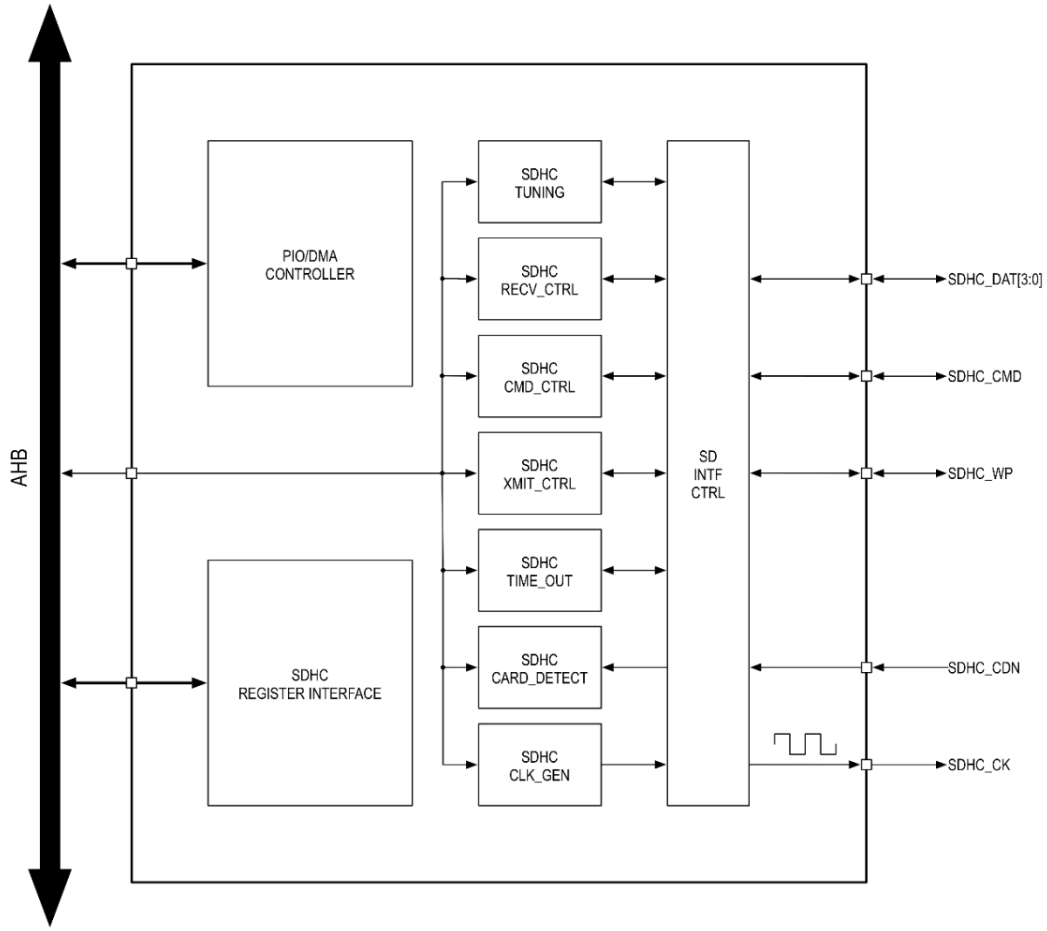
Compliance

- SD Host Controller Standard Specification Version 3.00
- SDIO Card Specification Version 3.0
- SD Memory Card Specification Version 3.01
- SD Memory Card Security Specification version 1.01
- e.MMC Specification version 4.51

SD/SDIO Card Interface

- Supports SDR50 with SDHC clock of up to 48MHz (24MB/sec)
- Supports DDR50 with SDHC clock of up to24MHz (24MB/sec)
- Designed to work with I/O cards, Read-Only cards, and Read/Write cards
- 1-bit and 4-bit data transfers in SD modes and SPI mode
- Double buffer for transfers configurable from 512B to 1KB.
- Auto Command (AutoCMD12 or AutoCMD23) support
- Multi-block transfers
- Variable-length data transfers
- Default and high-speed mode transfers
- Card insertion/removal events
- Read Wait Control, Suspend/Resume operation
- CRC7 for command and CRC16 for data integrity
- Single Operation DMA (SDMA) for data transfer
- Advanced DMA (ADMA) support

Figure 8-7: SDHC Block Diagram



8.5.1 Instances

The SDHC pin mapping for the SD Host Controller Standard Specification Version 3.0 are shown in *Table 8-43*.

Table 8-43: MAX32665/MAX32666 SDHC Alternate Function Mapping to SDHC Specification Pin Names

Alternate Function	Alternate Function Number	Pin Name	SDHC Specification Pin Name	Direction	Signal Description
SDHC_CDN	AF1	P1.7	SDCD#	I	Card present, active low.
SDHC_CLK	AF1	P1.3	SDCLK	O	SD clock signal.
SDHC_WP	AF1	P1.6	SDWP	I	Write protect signal, active high.
SDHC_CMD	AF1	P1.1	CMD	I/O	SD bus command signal.
SDHC_DAT0	AF1	P1.2	DAT[0]	I/O	SD data bus bit 0.
SDHC_DAT1	AF1	P1.4	DAT[1]	I/O	SD data bus bit 1.

Alternate Function	Alternate Function Number	Pin Name	SDHC Specification Pin Name	Direction	Signal Description
SDHC_DAT2	AF1	P1.5	DAT[2]	I/O	SD data bus bit 2.
SDHC_DAT3	AF1	P1.0	DAT[3]	I/O	SD data bus bit 3.

For configuration of the GPIO for SDHC peripheral usage see the General-Purpose I/O and Alternate Function Pins chapter.

8.5.2 SDHC Peripheral Clock Selection

The input clock to the SDHC peripheral is driven by the high speed system oscillator always, 96MHz. This 96MHz input clock is either divided by 2 (default) or by 4 to drive the SDHC peripheral. Set the SDHC peripheral clock divisor using the *GCR_PCLK_DIV.sdhcfrq* bit as shown in *Equation 8-1.*

Equation 8-1: SDHC Peripheral Clock

$$f_{SDHC\_CLK} = \frac{96MHz}{2^{GCR\_PCLK\_DIV.sdhcfrq}}$$

8.5.3 Usage

Communication over the SD bus is based on command and data bit streams/blocks that are initiated by a start bit and terminated by a stop bit.

- **Command**: A command is a token that starts an operation and is sent by the SDHC to the card in the embedded card slot. A command is transferred serially using the *SDHC_CMD* pin.
- **Response**: A response is a token sent from the card to the SDHC in response to a previously received command and is transferred serially using the *SDHC_CMD* pin.
- **Data**: You can transfer data from the card to the SDHC or vice versa using the SDHC_DAT[3:0] pins.

Figure 8-8, *Figure 8-9*, and *Figure 8-10* show the basic types of SD operations as described in the Physical Layer Simplified Specification Version 6.00 from the SD Card Association.

Figure 8-8: SD Bus Protocol - No Response and No Data Operations

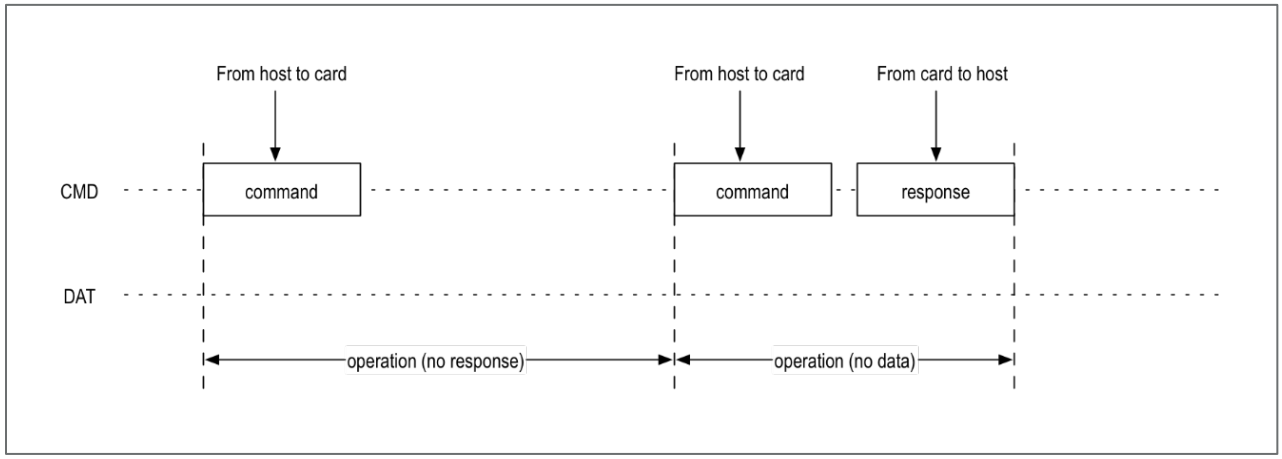


Figure 8-9: SD Bus Protocol - Multi-Block Read Operation

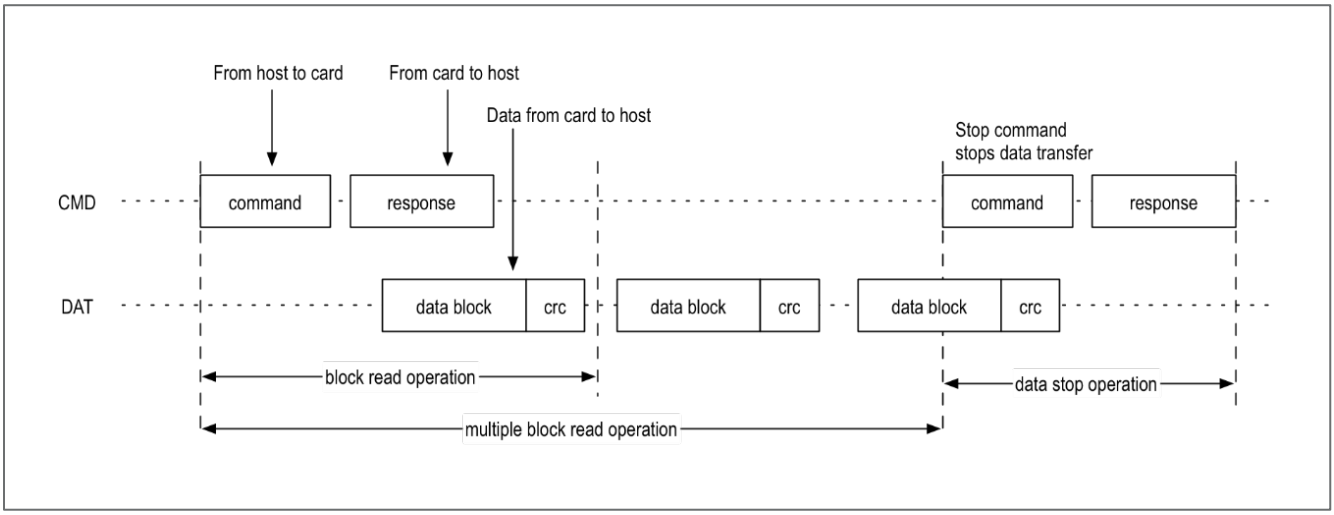
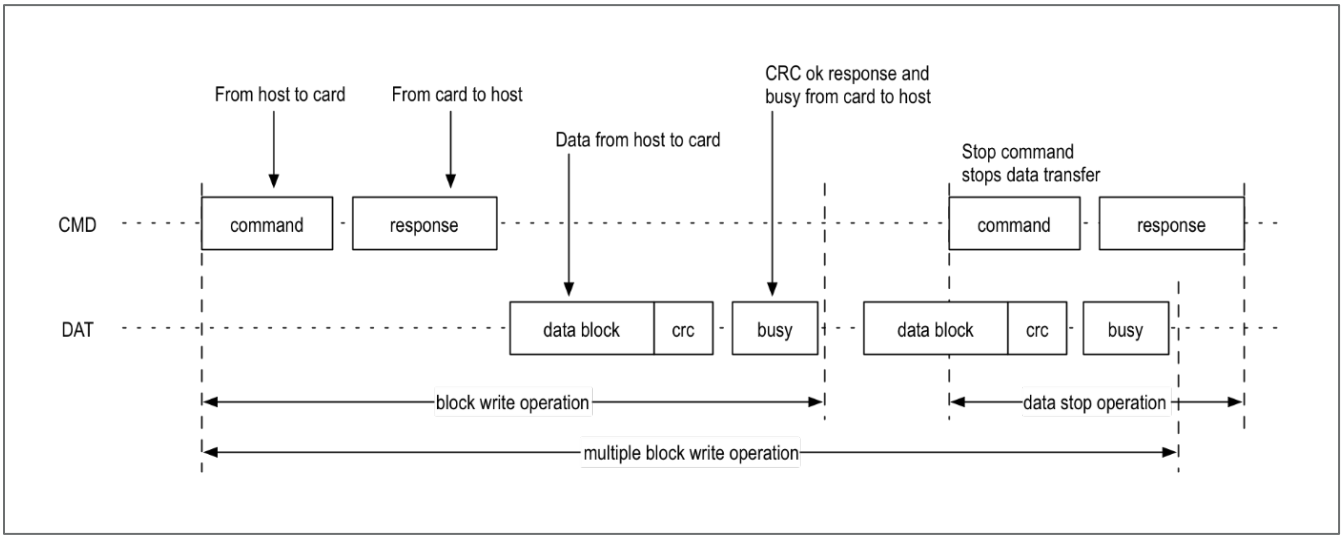


Figure 8-10: SD Bus Protocol - Multi Block Write Operation



8.5.4 SD Command Generation

Table 8-44 shows the registers required for three transaction types: SDMA generated transactions, ADMA generated transactions, and CPU transactions (includes data transfers and Non-DAT transfers). When initiating a transaction, you should program the registers sequentially starting with the *SDHC_SDMA* register and finishing with the *SDHC_CMD* register. When the upper byte of the *SDHC_CMD* register is written, it triggers the SDHC to issue the SD command.

Table 8-44: Registers Used to Generate SD Commands

Register	SDMA Command	ADMA Command	CPU Data Transfer	Non-DAT (No Data) Transfer
SDMA System Address/Argument 2 *SDHC_SDMA*	Yes/No	No/Auto CMD23	No/AutoCMD23	No/No
Block Size *SDHC_BLK_SIZE*	Yes	Yes	Yes	No (Protected)
Block Count *SDHC_BLK_CNT*	Yes	Yes	Yes	No (Protected)

Register	SDMA Command	ADMA Command	CPU Data Transfer	Non-DAT (No Data) Transfer
Argument 2 *SDHC_SDMA*	Yes	Yes	Yes	No (Protected)
Command *SDHC_CMD*	Yes	Yes	Yes	Yes

8.5.5 SDHC Registers

See *Table 3-3* for the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 8-45: SDHC Register Offsets, Names and Descriptions

Offset	Register Name	Description
[0x0000]	*SDHC_SDMA*	SDMA System Address / Argument 2
[0x0004]	*SDHC_BLK_SIZE*	Block Size register
[0x0006]	*SDHC_BLK_CNT*	Block Count register
[0x0008]	*SDHC_ARG_1*	Argument 1 register
[0x000C]	*SDHC_TRANS*	Transfer Mode register
[0x000E]	*SDHC_CMD*	Command register
[0x0010]	*SDHC_RESP_0*	Response register 0
[0x0012]	*SDHC_RESP_1*	Response register 1
[0x0014]	*SDHC_RESP_2*	Response register 2
[0x0016]	*SDHC_RESP_3*	Response register 3
[0x0018]	*SDHC_RESP_4*	Response register 4
[0x001A]	*SDHC_RESP_5*	Response register 5
[0x001C]	*SDHC_RESP_6*	Response register 6
[0x001E]	*SDHC_RESP_7*	Response register 7
[0x0020]	*SDHC_BUFFER*	Buffer Data Port register
[0x0024]	*SDHC_PRESENT*	Present State register
[0x0028]	*SDHC_HOST_CN_1*	Host Control 1 register
[0x0029]	*SDHC_PWR*	Power Control register
[0x002A]	*SDHC_BLK_GAP*	Block Gap Control register
[0x002B]	*SDHC_WAKEUP*	Wakeup Control register
[0x002C]	*SDHC_CLK_CN*	Clock Control register
[0x002E]	*SDHC_TO*	Timeout Control register
[0x002F]	*SDHC_SW_RESET*	Software Reset register
[0x0030]	*SDHC_INT_STAT*	Normal Interrupt Status register
[0x0032]	*SDHC_ER_INT_STAT*	Error Interrupt Status register
[0x0034]	*SDHC_INT_EN*	Normal Interrupt Status Enable register

Offset	Register Name	Description
[0x0036]	*SDHC_ER_INT_EN*	Error Interrupt Status Enable register
[0x0038]	*SDHC_INT_SIGNAL*	Normal Interrupt Signal Enable register
[0x003A]	*SDHC_ER_INT_SIGNAL*	Error Interrupt Signal Enable register
[0x003C]	*SDHC_AUTO_CMD_ER*	Auto CMD Error Status register
[0x003E]	*SDHC_HOST_CN_2*	Host Control 2 register
[0x0040]	*SDHC_CFG_0*	Capabilities register 0
[0x0044]	*SDHC_CFG_1*	Capabilities register 1
[0x0048]	*SDHC_MAX_CURR_CFG*	Maximum Current Capabilities register
[0x0050]	*SDHC_FORCE_CMD*	Force Event Register for Auto CMD Error Status
[0x0052]	*SDHC_FORCE_EVENT_INT_STAT*	Force Event Register for Error Interrupt Status
[0x0054]	*SDHC_ADMA_ER*	ADMA Error Status register
[0x0058]	*SDHC_ADMA_ADDR_0*	ADMA System Address register 0
[0x005C]	*SDHC_ADMA_ADDR_1*	ADMA System Address register 1
[0x0060]	*SDHC_PRESET_0*	Preset Value for Initialization
[0x0062]	*SDHC_PRESET_1*	Preset Value for Default Speed
[0x0064]	*SDHC_PRESET_2*	Preset Value for High Speed
[0x0066]	*SDHC_PRESET_3*	Preset Value for SDR12
[0x0068]	*SDHC_PRESET_4*	Preset Value for SDR25
[0x006A]	*SDHC_PRESET_5*	Preset Value for SDR50
[0x006C]	*SDHC_PRESET_6*	Preset Value for SDR104
[0x006E]	*SDHC_PRESET_7*	Preset Value for DDR50
[0x00FC]	*SDHC_SLOT_INT*	Slot Interrupt Status register
[0x00FE]	*SDHC_HOST_CN_VER*	Host Controller Version register

8.5.6 SDHC Register Details

Table 8-46: SDHC SDMA System Address / Argument Register

SDMA System Address / Argument 2 Register			SDHC_SDMA	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**SDMA System Address** This register is the address of the buffer used for a SDMA transfer. You must set this register to a valid data buffer address prior to starting an SDMA transfer. A SDHC DMA interrupt (*SDHC_INT_STAT*.dma = 1) is generated if the total size of the SDMA transfer exceeds the Host SDMA Buffer Size (*SDHC_BLK_SIZE.host_buf*). The card driver must update the SDMA System Address (*SDHC_SDMA*) with the address of the next data to transfer and clear the SDHC DMA interrupt flag prior to the transfer resuming. When the SDMA transfer is complete, this register contains the address of the next contiguous data address. When resuming a SDMA transfer, using the Resume command or by setting the *SDHC_BLK_GAP.gap_cont* bit to 1, the SDHC resumes using the address in this register for the data to transfer. Reading this register during a SDMA transfer might return an invalid value unless the transfer is paused as the result of a SDHC DMA interrupt. This field is not used for ADMA transfers. **Argument 2** This register is used with Auto CMD23 to set a 32-bit block count value to the argument of CMD23 while executing Auto CMD23. If Auto CMD23 is used with ADMA, then the full 32-bit block count value is used. If Auto CMD23 is used without AMDA, the available block count value is limited by the *SDHC_BLK_GAP* register to 65,535 blocks.

Table 8-47: SDHC SDMA Block Size Register

SDMA Block Size Register			SDHC_BLK_SIZE	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:15	-	R/W	0	**Reserved for Future Use** Do not modify this field.

<table>
<tr><th colspan="3">SDMA Block Size Register</th><th>SDHC_BLK_SIZE</th><th>[0x0004]</th></tr>
<tr><th>Bits</th><th>Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td>14:12</td><td>host_buf</td><td>R/W</td><td>0</td><td>Host SDMA Buffer Size
This field specifies the boundary of the contiguous buffer in the system memory for SDMA transfers. When an SDMA transfer crosses the host_buf address boundary, an SDHC DMA interrupt (SDHC_INT_STAT.dma) occurs. The SDMA transfer pauses until the card driver updates the SDMA System Address (SDHC_SDMA) register with the next buffer address to transfer and clears the SDHC DMA interrupt flag. When the SDMA transfer is complete, a SDHC transfer complete interrupt (SDHC_INT_STAT.trans_comp = 1) is generated. The SDHC DMA interrupt flag is not set when the SDMA transfer completes.
<table><tr><th>host_buf Value</th><th>Host SDMA Buffer Size (KB)</th></tr><tr><td>0b000</td><td>4</td></tr><tr><td>0b001</td><td>8</td></tr><tr><td>0b010</td><td>16</td></tr><tr><td>0b011</td><td>32</td></tr><tr><td>0b100</td><td>64</td></tr><tr><td>0b101</td><td>128</td></tr><tr><td>0b110</td><td>256</td></tr><tr><td>0b111</td><td>512</td></tr></table>Note: This field is used for SDMA transfers only.</td></tr>
<tr><td>11:0</td><td>trans</td><td>R/W</td><td>0x0200</td><td>Data Transfer Block Size
Sets the block size of data transfers for CMD17, CMD18, CMD24, CMD25, and CMD53. You can set values ranging from 1 up to the maximum buffer size. Setting this field to 0 indicates there is no data to transfer.
During a transfer, reading this field might return an invalid value, and writes to this field are ignored.
<table><tr><th>trans Value</th><th>Block Size in Bytes</th></tr><tr><td>0x0800</td><td>2,048</td></tr><tr><td>0x07FF</td><td>2,047</td></tr><tr><td>...</td><td>...</td></tr><tr><td>0x200</td><td>512</td></tr><tr><td>0x01FF</td><td>511</td></tr><tr><td>...</td><td>...</td></tr><tr><td>0x0004</td><td>4</td></tr><tr><td>0x0003</td><td>3</td></tr><tr><td>0x0002</td><td>2</td></tr><tr><td>0x0001</td><td>1</td></tr><tr><td>0x0000</td><td>No data transfer</td></tr></table></td></tr>
</table>

Table 8-48: SDHC SDMA Block Count Register

SDMA Block Count Register			SDHC_BLK_CNT	[0x0006]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field.
15:0	count	R/W	0x0200	**Current Transfer Block Count** Set to the total number of blocks to transfer prior to a block transfer operation. Set the Block Count Enable (*SDHC_TRANS.blk_cnt_en*) bit to 1 for a block transfer. If Block Count Enable is clear, then this field is unused. When set to 1, the value in this register is the total number of blocks to transfer. After each block transfer, this register is decremented by 1, and stops when the count reaches 0. Reads from this register are only valid when no transactions are active. A setting of 0 results in no blocks transferred. When a Suspend command is complete, the number of remaining blocks to transfer is contained in this field. Before issuing a Resume command, the card driver must restore the previously-saved block count to this field.

***trans* Value**	**Block Count**
0xFFFF	65,535
0xFFFE	65,534
….	….
0x0002	2
0x0001	1
0x0000	Stop count or no block transfer

Table 8-49: SDHC SDMA Argument 1 Register

SDMA Argument 1 Register			SDHC_ARG_1	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	cmd	R/W	0	**SD Command Argument 1** The SD Command Argument 1 is specified as bit [39:8] of the Command-Format in the Physical Layer Specification.

Table 8-50: SDHC SDMA Transfer Mode Register

SDMA Transfer Mode Register			SDHC_TRANS	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:6	-	R/W	0	**Reserved for Future Use** Do not modify this field.
5	multi	R/W	0	**Multi/Single Block Select** Used for DAT line transfers and multiple-block commands. For all other commands, set this bit to 0. 1: Multiple-block or DAT line transfer 0: Single Block *Note: The SDHC_BLK_CNT register is ignored if this field is set to 0.*

SDMA Transfer Mode Register			SDHC_TRANS	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
4	read_write	R/W	0	**Data Transfer Direction Select** Sets the direction for DAT line data transfers. Set to 1 to transfer data from the SD card to the SDHC (Read). For all other commands, set this bit to 0 (Write). 1: Read (from card to host) 0: Write (from host to card)
3:2	auto_cmd_en	R/W	0	**Auto CMD Enable / Function Selection** 0b00: Auto Command Disabled 0b01: Auto CMD12 Enable 0b10: Auto CMD23 Enable 0b11: Reserved for Future Use **Auto CMD12 Enable** When auto_cmd_en is set to 1, the SDHC issues CMD12 automatically after completion of the last block transfer. If an error occurs from Auto CMD12, then the error is saved to the *SDHC_AUTO_CMD_ER* register. *Note: Do not set to 1 if an Auto CMD12 is not required.* **Auto CMD23 Enable** When this bit field is set to 0b10, the Host Controller issues a CMD23 automatically before issuing the command specified in the *SDHC_CMD* (Command) register. The following conditions are required to use Auto CMD23: • Auto CMD23 support (Host Controller Version is 3.00 or later) • A memory card that supports CMD23 (SCR[33] = 1) • If using DMA, ADMA mode only • Only when CMD18 or CMD25 is issued You can use Auto CMD23 with or without ADMA. By writing to the Command register, the SDHC issues a CMD23 first, and then issues the command specified by the Command Index (*SDHC_CMD.idx*) in the Command register. If response errors are detected from CMD23, then the second command is not issued. A CMD23 error is indicated in the Auto CMD Error Status register (*SDHC_AUTO_CMD_ER*). The 32-bit block count value for CMD23 is set to the SDMA System Address / Argument 2 register (*SDHC_SDMA*). *Note: The SDHC does not check the command index.*
1	blk_cnt_en	R/W	0	**Block Count Enable** Set to enable the Block Count register (*SDHC_BLK_CNT*) for multiple block transfers. When this bit is 0, the Block Count register (*SDHC_BLK_CNT*) is disabled, which is useful if executing an infinite transfer. 1: Enable *SDHC_BLK_CNT* register 0: Disable *SDHC_BLK_CNT* register
0	dma_en	R/W	0	**DMA Enable** Enables DMA functionality per the Capabilities register. If this bit is set to 1, a DMA operation begins when the card driver writes to the upper byte of the Command register (*SDHC_CMD*). 1: DMA mode is enabled as specified in the *SDHC_HOST_CN_1.dma_select* field. 0: DMA mode disabled.

Table 8-51: Summary of how register settings determine type of data transfer

Multi/Single Block Select *SDHC_TRANS.multi*	Block Count Enable *SDHC_TRANS.blk_cnt_en*	Block Count *SDHC_BLK_CNT.count*	Function
0	N.A.	N.A.	Single transfer
1	0	N.A.	Infinite transfer
1	1	≠0	Multiple transfer
1	1	0	Stop Multiple transfer

Table 8-52: SDHC Command Register

Command Register		SDHC_CMD		[0x000E]
Bits	**Name**	**Access**	**Reset**	**Description**
31:14	-	R/W	NA	**Reserved for Future Use** Do not modify this field.
13:8	idx	R/W	0	**Command Index** Valid command number (CMD0-63, ACMD0-63) per the SD Physical Specification and SDIO Card Specification.
7:6	type	R/W	0	**Command Type** The following table lists the values for this field, the type of command, and provides notes about what the command type is typically used for: *type* Value / Command Type / Notes 0b11 / Abort / CMD12, CMD52 for writing I/O Abort in CCCR. 0b10 / Resume / CMD52 for writing Function Select in CCCR. 0b01 / Suspend / CMD52 for writing Bus Suspend in CCCR. 0b00 / Normal / Other commands
5	data_pres_sel	R/W	0	**Data Present Select** 1: Set to indicate data is present and transferable using the DAT line. 0: Commands that only use the CMD line (for example, CMD52), commands with no data transfer but are using the busy signal on SDHC_DAT[0], or a Resume command.
4	idx_chk_en	R/W	0	**Command Index Check Enable** 1: SDHC checks the index field in the response and sets a Command Index Error if it does not match the value in the *SDHC_CMD.idx* field. 0: Index of response is not checked.
3	crc_chk_en	R/W	0	**Command CRC Check Enable** 1: SDHC verifies the CRC field in the response, and if an error is detected, it is reported as a Command CRC Error. 0: CRC not checked by hardware.
2	-	R/W	0	**Reserved for Future Use** Do not modify this field.
1:0	resp_type	R/W	0	**Response Type Select** 0b00: No Response 0b01: Response Length 136 0b10: Response Length 48 0b11: Response Length 48, and check if busy after response

Table 8-53: Relationship between Parameters and the Name of Response Type

Response Type *SDHC_CMD.resp_type*	Index Check Enable *SDHC_CMD.idx_chk_en*	CRC Check Enable *SDHC_CMD.crc_chk_en*	Name of Response Type
0b00	0	0	No Response
0b01	0	1	R2
0b10	0	0	R3, R4
0b10	1	1	R1, R5, R6, R7
0b11	1	1	R1b, R5b

Table 8-54: SDHC Response 0 Register

Response 0 Register			SDHC_RESP_0	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	cmd_resp	R	0	**Response Register 0** Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 8-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32665/MAX32666. *Table 8-63* shows the SD types of response mapped to the MAX32665/MAX32666 Response registers.

Table 8-55: SDHC Response 1 Register

Response 1 Register			SDHC_RESP_1	[0x0012]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	cmd_resp	R	0	**Response Register 1** Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 8-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32665/MAX32666. *Table 8-63* shows the SD types of response mapped to the MAX32665/MAX32666 Response registers.

Table 8-56: SDHC Response 2 Register

Response 2 Register			SDHC_RESP_2	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	cmd_resp	R	0	**Response Register 2** Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 8-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32665/MAX32666. *Table 8-63* shows the SD types of response mapped to the MAX32665/MAX32666 Response registers.

Table 8-57: SDHC Response 3 Register

Response 3 Register			SDHC_RESP_3	[0x0016]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	cmd_resp	R	0	**Response Register 3** Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 8-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32665/MAX32666. *Table 8-63* shows the SD types of response mapped to the MAX32665/MAX32666 Response registers.

Table 8-58: SDHC Response 4 Register

Response 4 Register			SDHC_RESP_4	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	cmd_resp	R	0	**Response Register 4** Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 8-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32665/MAX32666. *Table 8-63* shows the SD types of response mapped to the MAX32665/MAX32666 Response registers.

Table 8-59: SDHC Response 5 Register

Response 5 Register			SDHC_RESP_5	[0x001A]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	cmd_resp	R	0	**Response Register 5** Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 8-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32665/MAX32666. *Table 8-63* shows the SD types of response mapped to the MAX32665/MAX32666 Response registers.

Table 8-60: SDHC Response 6 Register

Response 6 Register			SDHC_RESP_6	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	cmd_resp	R	0	**Response Register 6** Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 8-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32665/MAX32666. *Table 8-63* shows the SD types of response mapped to the MAX32665/MAX32666 Response registers.

Table 8-61: SDHC Response 7 Register

Response 7 Register			SDHC_RESP_7	[0x001E]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	cmd_resp	R	0	**Response Register 7** Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 8-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32665/MAX32666. *Table 8-63* shows the SD types of response mapped to the MAX32665/MAX32666 Response registers.

Table 8-62: SDHC Response Register Mapping to SD Host Controller Response Register Convention

Register	Register Name	Register Offset	SDHC REP[] Bit Mapping
SDHC_RESP_0	Response 0	0x10	REP[15:0]
SDHC_RESP_1	Response 1	0x12	REP[31:16]
SDHC_RESP_2	Response 2	0x14	REP[47:32]
SDHC_RESP_3	Response 3	0x16	REP[63:48]
SDHC_RESP_4	Response 4	0x18	REP[79:64]
SDHC_RESP_5	Response 5	0x1A	REP[95:80]
SDHC_RESP_6	Response 6	0x1C	REP[111:96]
SDHC_RESP_7	Response 7	0x1E	REP[127:112]

Table 8-63: Kind of SD Card Response Mapping to SDHC Response Registers

Kind of Response	Meaning of Response	REP[] Specification Mapping	SDHC Response Register MSW	SDHC Response Register LSW
R1, R1b (normal response)	Card Status	REP[31:0]	*SDHC_RESP_1*	*SDHC_RESP_0*
R1b (Auto CMD12 response)	Card Status for Auto CMD12	REP[127:96]	*SDHC_RESP_7*	*SDHC_RESP_6*
R1 (Auto CMD23 response)	Card Status for Auto CMD23	REP[127:96]	*SDHC_RESP_7*	*SDHC_RESP_6*
R2 (CID, CSD register)	CID or CSD reg. incl.	REP [119:0]	*SDHC_RESP_7*	*SDHC_RESP_0*
R3 (OCR register)	OCR register for memory	REP [31:0]	*SDHC_RESP_1*	*SDHC_RESP_0*
R4 (OCR register)	OCR register for I/O, etc	REP [31:0]	*SDHC_RESP_1*	*SDHC_RESP_0*
R5, R5b	SDIO response	REP [31:0]	*SDHC_RESP_1*	*SDHC_RESP_0*
R6 (Published RCA response)	Newly published RCA[31:16], etc	REP [31:0]	*SDHC_RESP_1*	*SDHC_RESP_0*

Table 8-64: SDHC Buffer Data Port Register

Buffer Data Port Register			SDHC_BUFFER	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Buffer Data** Pointer to the SDHC internal data buffer.

Table 8-65: SDHC Present State Register

Present State Register			SDHC_PRESENT	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:25	-	R	0	**Reserved for Future Use** Do not modify this field.
24	cmd_signal_level	R	0	**CMD Line Signal Level** Indicates the CMD line level for error recovery and debugging.

Present State Register			SDHC_PRESENT	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
23:20	dat_signal_level	R	-	**SDHC_DAT[3:0] Line Signal Level** Indicates the DAT line level for error recovery and debugging. Use to detect the busy signal level as indicated on SDHC_DAT[0].
19	wp	R	-	**Write Protect Switch Pin Level** The write protect switch is supported for memory and combo cards. This bit reflects the state of the SDHC_WP pin. 1: Write enabled (SDHC_WP = 1) 0: Write protected (SDHC_WP = 0)
18	card_detect	R	-	**Card Detect Pin Level** This bit reflects the inverted state of the SDHC_CDN pin. Debouncing is not performed on this bit. When Card State Stable is set to 1, this bit might be valid, but is not guaranteed. To use this bit, the card driver must debounce the bit. 1: Card present (SDHC_CDN = 0) 0: No card present (SDHC_CDN = 1)
17	card_state	R	-	**Card State Stable** Used for debugging only. If this bit reads 0, the SDHC_CDN pin level is not stable. If this bit reads 1, the SDHC_CDN pin level is stable. 1: No card or card inserted 0: Reset or debouncing *Note: This bit is not valid unless the SDHC_PRESENT.card_inserted bit reads 1.*
16	card_inserted	R	-	**Card Inserted** Indicates if a card is inserted. This signal is debounced by the SDHC hardware. A change in state from 0 to 1 on this bit generates an SDHC_IRQ with the *SDHC_INT_STAT.card_insertion* flag set. Conversely, a transition of this bit from a 1 to a 0 generates an SDHC_IRQ interrupt with the *SDHC_INT_STAT.card_removal* field set. 1: Card Inserted 0: Reset, debouncing, or no card inserted
15:12	-	R	0	**Reserved for Future Use** Do not modify this field.
11	buffer_read	R	0	**Buffer Read Status** If this bit reads 1, then data is available in the buffer for non-DMA transfers. This bit is cleared when all available block data is read from the buffer. This bit transitions from 0 to 1 when block data is ready in the buffer resulting in a SDHC_IRQ interrupt, if enabled, with the *SDHC_INT_STAT.buffer_rd_ready* flag set. 1: Read data available 0: No data to read
10	buffer_write	R	0	**Buffer Write Status** If this bit reads 1, then space is available in the buffer for write data. This bit is cleared when no space is available in the buffer. This bit transitions from a 0 to a 1 when top-of-block data is written to the buffer, resulting in a SDHC_IRQ interrupt, if enabled, with the *SDHC_INT_STAT.buffer_wr_ready* flag set. 1: Space available in the buffer for write data 0: No space available in the buffer for write data

Present State Register			SDHC_PRESENT	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
9	read_transfer	R	0	**Read Transfer Active** Indicates completion of a read transfer. This bit is set to 1 for either of the following conditions: 1) After the end bit of a Read command. 2) When a read operation is restarted by setting the *SDHC_BLK_GAP.cont* bit (Continue Request). This bit is set to 0 for either of the following conditions: 1) The last data block as specified by the block length is transferred to the SDHC. 2) When all valid data blocks are transferred to the system, and no current block transfers are sent because the Stop At Block Gap Request register field is set to 1. A SDHC_IRQ interrupt is generated, if enabled. 1: Transferring data 0: No valid data
8	write_transfer	R	0	**Write Transfer Active** This bit is set to 1 for either of the following conditions: 1) After the end bit of the Write command. 2) When a write operation is restarted by setting the *SDHC_BLK_GAP.cont* bit to 1. This bit is cleared to 0 for either of the following conditions: 1) After getting the CRC status of the last data block transfer as specified by the transfer count, single and multiple block, *SDHC_BLK_CNT* register. 2) After getting the CRC status of any block where data transmission is stopped by a Stop At Block Gap Request (*SDHC_BLK_GAP.stop*). When *SDHC_BLK_GAP.stop* (stop at block gap request) is set, a change in *write_transfer* from 1 to 0 causes an SDHC_IRQ interrupt, if enabled, with the *SDHC_INT_STAT.blk_gap_event* flag set to 1. The *blk_gap_event* field indicates to the card driver that a non-DAT command can be issued during an active write. 1: Transferring data 0: No valid data for transfer
7:4	-	R	0	**Reserved for Future Use** Do not modify this field.
3	retuning	R	0	**Re-Tuning Request** If this field reads 1, a retuning request was received from the external device. 0: Re-tuning request has not been received. 1: Re-tuning request received.
2	dat_line_active	R	0	**DAT Line Active** A value of 1 indicates one or more DAT lines (SDHC_DAT[3:0]) are in use on the SD Bus. 0: No SD Bus DAT lines in use. 1: 1 or more DAT lines are in use.

Present State Register			SDHC_PRESENT	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
1	dat	R	0	**Command Inhibit (DAT)** This bit is set if DAT Line Active or the Read Transfer Active bits are set. A SDHC_IRQ interrupt is generated, if enabled, when this bit transitions from a 1 to a 0 with the *SDHC_INT_STAT.trans_comp* flag set. The card driver can save registers in the range of 0x000 to 0x00D for a suspend transaction after the *SDHC_INT_STAT.trans_comp* interrupt event. 1: Command that uses DAT line cannot be issued. 0: Command that uses DAT line can be issued.
0	cmd	R	0	**Command Inhibit (CMD)** If this bit reads 0, the CMD line is not in use. This bit is set to 1 by the SDHC immediately after the *SDHC_CMD* register is written, and the bit is cleared to 0 when the Command Response is received. Auto CMD12 and Auto CMD23 consist of two responses, and this bit is not cleared until the read/write portion of the sequence is complete. 1: Command cannot be issued. 0: Can issue command using only CMD line.

Table 8-66: SDHC Host Control 1 Register

Host Control 1 Register			SDHC_HOST_CN_1	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
7	card_detect_signal	R/W	0	**Card Detect Signal Selection** 1: The Card Detect Test Level is selected (for test purposes) 0: SDHC_CDN is used for card detection (normal operation) *Note: Disable the Card Detect Interrupt when changing this bit.*
6	card_detect_test	R/W	-	**Card Detect Test Level** This bit is enabled when the Card Detect Signal Selection, *SDHC_HOST_CN_1*. card_detect_signal, field is set to 1. 1: Card Inserted 0: No card inserted
5	ext_data_transfer_width	R/W	0	**Extended Data Transfer Width** Extended data transfer width is not supported on the MAX32665/MAX32666. Always reads 0. 0: Bus width is selected by SHDC_HOST_CN_1.*data_transfer_width* field
4:3	dma_select	R/W	0	**DMA Select** Sets the DMA mode. 0b00: SDMA mode 0b01: Reserved 0b10: 32-bit address ADMA2 mode 0b11: Reserved
2	hs_en	R/W	0	**High Speed Enable** 1: High-speed mode 0: Normal-speed mode
1	data_transfer_width	R/W	0	**Data Transfer Width** Sets the data transfer width of the SDHC. 1: 4-bit mode 0: 1-bit mode

Host Control 1 Register			SDHC_HOST_CN_1	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
0	led_cn	R/W	0	**LED Control** 1: LED on 0: LED off

Table 8-67: SDHC Power Control Register

Power Control Register			SDHC_PWR	[0x0029]
Bits	**Name**	**Access**	**Reset**	**Description**
7:4	-	R/W	0	**Reserved for Future Use** Do not modify this field.
3:1	bus_volt_sel	R/W	6	**SD Bus Voltage Select** Sets the voltage level for the SD card. Validate the setting against the Capabilities Register (*SDHC_CFG_0*). 7: 3.3V typical 6: 3.0V typical 5: 1.8V typical 4: Reserved for Future Use. 3: Reserved for Future Use. 2: Reserved for Future Use. 1: Reserved for Future Use. 0: Reserved for Future Use.
0	bus_power	R/W	0	**SD Bus Power** Before setting this bit, configure the SDHC_PWR.*bus_volt_sel* field. If no card is detected, then this bit is automatically set to 0 by the SDHC. 1: Power Enabled 0: Power Disabled

Table 8-68: SDHC Block Gap Control Register

Block Gap Control Register			SDHC_BLK_GAP	[0x002A]
Bits	**Name**	**Access**	**Reset**	**Description**
7:4	-	R	0	**Reserved for Future Use** Do not modify this field.
3	intr	R/W	0	**Interrupt at Block Gap** Setting this bit to 1 enables interrupt detection at the block gap for a multiple block transfer. 1: Enabled 0: Disabled *Note: This bit is only valid if SDHC_PWR.data_transfer_width = 1 (4-bit mode).*

Block Gap Control Register	SDHC_BLK_GAP	[0x002A]

Bits	Name	Access	Reset	Description
2	read_wait	R/W	0	**Read Wait Control** If the card supports read wait (optional for SDIO cards), setting this bit enables use of the read wait protocol to stop reading data using the SDHC_DAT[2] line. If the card does not support read wait, the SDHC stops the SD Clock to hold read data, preventing command generation. When a card is inserted, the card driver must set this field based on the CCCR of the SDIO card inserted. Suspend/Resume is not supported when this bit is set to 0. 1: Enable Read Wait Control 0: Disable Read Wait Control *Note: If the SDIO card does not support read wait, then you must not set this bit to 1. Setting it to 1 when read wait is not supported might cause a SDHC_DAT line conflict.*
1	cont	R/W	0	**Continue Request** This bit is used to restart a transaction that was stopped using the Stop At Block Gap Request (*SDHC_BLK_GAP.stop*). To cancel a stop at the block gap, set *SDHC_BLK_GAP.stop* to 0, and set this bit, *SDHC_BLK_GAP.cont*, to 1 to restart the transfer. This bit is automatically cleared by hardware for either of the following conditions: • During a read transaction, the DAT Line Active changes from 0 to 1 as the write transaction restarts. • During a write transaction, the Write Transfer Active changes from 0 to 1 as the write transaction restarts. 1: Restart 0: No effect
0	stop	R/W	0	**Stop At Block Gap Request** Setting this bit stops executing read and write transactions at the next block gap for non-DMA, SDMA, and ADMA transfers. This bit must remain set to 1 until the *SDHC_INT_STAT.trans_comp* bit is set to 1. For write transfers where the card driver writes data to the Buffer Data Port Register (*SDHC_BUFFER*), the card driver must set this bit after all block data is written. 1: Stop 0: Transfer This bit affects the following fields: • Read Transfer Active, *SDHC_PRESENT.read_transfer* • Write Transfer Active, *SDHC_PRESENT.write_transfer* • SDHC_DAT Line Active, *SDHC_PRESENT.dat_line_active* • Command Inhibit (DAT), *SDHC_PRESENT.dat* *Note: If this bit is set to 1, the card driver must not write data to the Buffer Data Port Register (SDHC_BUFFER).* *Note: Clearing both the SDHC_BLK_GAP.stop and SDHC_BLK_GAP.cont fields does not cause a transaction to restart.* *Note: You can set this bit to 1 regardless of whether the card inserted supports Read Wait Control. The SDHC stops the card through Read Wait Control or by stopping the SD clock.*

Table 8-69: SDHC Wakeup Control Register

Wakeup Control Register		SDHC_WAKEUP		[0x002B]
Bits	**Name**	**Access**	**Reset**	**Description**
7:3	-	R/W	0	**Reserved for Future Use** Do not modify this field.
2	card_rem	R/W	0	**Wakeup Event Enable on SD Card Removal** Enable wakeup event interrupt when the *SDHC_INT_STAT.card_removal* flag occurs. 1: Enable Interrupt 0: Disable Interrupt
1	card_ins	R/W	0	**Wakeup Event Enable on SD Card Insertion** Enable wakeup event interrupt when the *SDHC_INT_STAT.card_inserted* flag occurs. 1: Enable Interrupt 0: Disable Interrupt
0	card_int	R/W	0	**Wakeup Event Enable On Card Interrupt** Enable wakeup event interrupt when the *SDHC_INT_STAT.card_intr* flag occurs.

Table 8-70: SDHC Clock Control Register

<table>
<tr><th colspan="3">Clock Control Register</th><th>SDHC_CLK_CN</th><th>[0x002C]</th></tr>
<tr><th>Bits</th><th>Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td>15:8</td><td>sdclk_freq_sel</td><td>R/W</td><td>0</td><td>SDCLK Frequency Select
Selects the SD Clock Frequency output on the SDHC_CLK pin.
The SD Clock Frequency Select is a total of 10bits. The divisors shown below consist of the upper_sdclk_freq_sel bits as bits 9:8, and the sdclk_freq_sel bits as bits 7:0 of the divisor.
<table>
<tr><th>upper_sdclk_freq_sel</th><th>sdclk_freq_sel</th><th>SDCLK Divisor (N)</th></tr>
<tr><td>0b11</td><td>0b11111111</td><td>1023</td></tr>
<tr><td>0b11</td><td>0b00000000</td><td>768</td></tr>
<tr><td>…</td><td>…</td><td>…</td></tr>
<tr><td>0b10</td><td>0b01010101</td><td>597</td></tr>
<tr><td>….</td><td>….</td><td>….</td></tr>
<tr><td>….</td><td>….</td><td>N</td></tr>
<tr><td>….</td><td>….</td><td>….</td></tr>
<tr><td>0b00</td><td>0b00000010</td><td>2</td></tr>
<tr><td>0b00</td><td>0b00000001</td><td>1</td></tr>
<tr><td>0b00</td><td>0b00000000</td><td>0 (MAX)</td></tr>
</table>
Setting upper_sdclk_freq_sel and sdclk_freq_sel to 0 results in the maximum SDCLK frequency of $f_{SDHC\_CLK\_FRQ}$. All other settings for upper_sdclk_freq_sel and sdclk_freq_sel follow the equation below:
$SDHC\_CLK = {f_{SDHC\_CLK\_FRQ}}/{(2 \times N)}$
Note: The SD Clock Enable must be disabled (SDHC_CLK_CN.sd_clk_en = 0) prior to modification of this field.</td></tr>
<tr><td>7:6</td><td>upper_sdclk_freq_sel</td><td>R/W</td><td>0</td><td>Upper Bits of SDCLK Frequency Select
Bits 9 and 8 of the 10-bit SDCLK frequency select. See the SDHC_CLK_CN.sdclk_freq_sel field for details about the clock select calculation.
Note: The SD Clock Enable must be disabled (SDHC_CLK_CN.sd_clk_en = 0) prior to modification of this field.</td></tr>
<tr><td>5</td><td>clk_gen_sel</td><td>R</td><td>0</td><td>Clock Generator Select
Reads 0 indicating Divided Clock mode only for SD Clock Frequency generation.
0: Divided clock mode</td></tr>
<tr><td>4:3</td><td>-</td><td>R</td><td>0</td><td>Reserved for Future Use
Do not modify this field.</td></tr>
</table>

Clock Control Register			SDHC_CLK_CN	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
2	sd_clk_en	R/W	0	**SD Clock Enable** Enable/disable SD Clock generation. 1: Enable the SD Clock and output on the SDHC_CLK pin. 0: SD Clock is disabled. *Note: This bit is cleared by the SDHC if the card-inserted field in the Present State register is cleared.* *Note: The internal_clk_en bit must be set to 1, and the internal_clk_stable bit must read 1 prior to setting this bit to 1.*
1	internal_clk_stable	R	0	**Internal Clock Stable** This bit is set to 1 when the internal clock is stable. *Note: The internal clock must be enabled (SDHC_CLK_CN.internal_clk_en = 1) before this field is used.*
0	internal_clk_en	R/W	0	**Internal Clock Enable** Enable the internal clock. *Note: This bit must be set, and the internal_clk_stable bit must read 1 prior to setting the SD Clock Enable (SDHC_CLK_CN.sd_clk_en) bit.* *Note: This bit is set to 0 by the SDHC if waiting for a wakeup interrupt.*

Table 8-71: SDHC Timeout Control Register

Timeout Control Register			SDHC_TO	[0x002E]
Bits	**Name**	**Access**	**Reset**	**Description**
7:4	-	R/W	0	**Reserved for Future Use** Do not modify this field.
3:0	data_count_value	R/W	0	**Data Timeout Counter Value** Determines the interval for DAT line timeout detection. The timeout clock frequency is generated by dividing PCLK by the value calculated using this register. See Capabilities 0 Register (*SDHC_CFG_0*) for the definition of TMCLK. The calculation for Data Timeout is shown in the following equation: $$\text{Data Timeout} = \text{TMCLK} \times 2^{(13+\text{data\_count\_value})}$$ **Setting** / **Data Timeout** 0b1111 — Reserved 0b1110 — $\text{TMCLK} \times 2^{(27)}$ 0b1101 — $\text{TMCLK} \times 2^{(26)}$ … — … 0b0010 — $\text{TMCLK} \times 2^{(15)}$ 0b0001 — $\text{TMCLK} \times 2^{(14)}$ 0b0000 — $\text{TMCLK} \times 2^{(13)}$ *Note: Disable the Data Timeout Error Status Enable in the Error Interrupt Status Enable register (SDHC_ER_INT_EN.data_to).*

Table 8-72: SDHC Software Reset Register

<table>
<tr><th colspan="3">Software Reset Register</th><th>SDHC_SW_RESET</th><th>[0x002F]</th></tr>
<tr><th>Bits</th><th>Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td>7:3</td><td>-</td><td>R/W</td><td>0</td><td>Reserved for Future Use
Do not modify this field.</td></tr>
<tr><td>2</td><td>reset_dat</td><td>R/W1</td><td>0</td><td>Software Reset for DAT Line
1: Reset
0: Ready
The following registers and fields are cleared/initialized when this bit is set:<table><tr><th>Register</th><th>Field</th></tr><tr><td>SDHC_BUFFER</td><td>data</td></tr><tr><td>SDHC_PRESENT</td><td>buffer_read</td></tr><tr><td></td><td>buffer_write</td></tr><tr><td></td><td>read_transfer</td></tr><tr><td></td><td>write_transfer</td></tr><tr><td></td><td>dat_line_active</td></tr><tr><td></td><td>dat</td></tr><tr><td></td><td>cmd</td></tr><tr><td>SDHC_BLK_GAP</td><td>cont</td></tr><tr><td></td><td>stop</td></tr><tr><td>SDHC_INT_STAT</td><td>buff_rd_ready</td></tr><tr><td></td><td>buff_wr_ready</td></tr><tr><td></td><td>dma</td></tr><tr><td></td><td>blk_gap_event</td></tr><tr><td></td><td>trans_comp</td></tr></table>Note: After setting this bit to 1, the Card Driver must poll this bit until it reads 0 to determine reset completion.</td></tr>
<tr><td>1</td><td>reset_cmd</td><td>R/W1</td><td>0</td><td>Software Reset for CMD Line
1: Reset
0: Ready
The following registers and fields are cleared by setting this bit.<table><tr><th>Register</th><th>Field</th></tr><tr><td>SDHC_PRESENT</td><td>cmd</td></tr><tr><td>SDHC_INT_STAT</td><td>cmd_comp</td></tr></table>Note: After setting this bit to 1, the card driver must poll this bit for 0 to determine when the reset is complete.</td></tr>
</table>

Software Reset Register			SDHC_SW_RESET	[0x002F]
Bits	**Name**	**Access**	**Reset**	**Description**
0	reset_all	R/W1	0	**Software Reset for All** Reset the SDHC except for the card detection interface. All registers are reset to their Reset/POR state. 1: Reset 0: Ready *Note: After the Card Driver sets this bit to 1, the Card Driver should poll this bit until it reads 0 to determine when the SDHC completes the reset all request.*

8.5.6.1 Normal Interrupt Status Register

The Normal Interrupt Status Enable affects reads of this register, but Normal Interrupt Signal Enable does not. An interrupt is generated when the Normal Interrupt Signal Enable is enabled, and at least one of the status bits is set to 1. You can clear more than one status with a single register write. The Card Interrupt (*SDHC_INT_STAT.card_intr*) is cleared when the card stops asserting the interrupt after the Card Driver services the interrupt condition.

Table 8-73: SDHC Normal Interrupt Status Register

Normal Interrupt Status Register			SDHC_INT_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
15	err_intr	R/W0	0	**Error Interrupt** If any of the bits in the Error Interrupt Status register are set, then this bit is set. Therefore, the Host Driver can efficiently test for an error by checking this bit first. 0: No Error 1: Error
14:13	-	-	0	**Reserved for Future Use**
12	retuning	R/W0	0	**Re-Tuning Event** This status is set if the Re-Tuning Request bit in the Present State register changes from 0 to 1. The SDHC requests the Host Driver to perform re-tuning for the next data transfer. However, you can complete the current data transfer (not large block count) without re-tuning. 0: Re-tuning is not required 1: Perform re-tuning before the next data transfer
11:9	-	-	0	**Reserved for Future Use**
8	card_intr	R	0	**Card Interrupt** In one-bit mode, the SDHC detects the Card Interrupt without the SD Clock to support wakeup. In four-bit mode, the card interrupt signal is sampled during the interrupt cycle resulting in a delay between the interrupt signal from the memory card and the interrupt signal to the host driver. 1: Generate Card Interrupt 0: No Card Interrupt *Note: Writing a 1 to this bit does not clear this bit. It is cleared by resetting the SDHC_INT_EN.card_int flag.*
7	card_removal	R/W1C	0	**Card Removal** Set if the Card Inserted field in the Present State register (*SDHC_PRESENT.card_inserted*) changes from 1 to 0. 1: Card removed 0: Card state stable or hardware debouncing

Normal Interrupt Status Register			SDHC_INT_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
6	card_insertion	R/W1C	0	**Card Inserted** Set if the Card Inserted field in the Present State register (*SDHC_PRESENT.card_inserted*) changes from 0 to 1. 1: Card inserted 0: Card state stable or hardware debouncing
5	buff_rd_ready	R/W1C	0	**Buffer Read Ready** Set if the Buffer Read Enable field in the Present State register (*SDHC_PRESENT.buff_rd_ready*) changes from 0 to 1. 1: Ready to read buffer 0: Not ready to read buffer *Note: This field is set to 1 for every CMD19 execution while performing a tuning procedure (SDHC_HOST_CN_2.execute = 1).*
4	buff_wr_ready	R/W1C	0	**Buffer Write Ready** Set if the Buffer Write Enable field in the Present State register (*SDHC_PRESENT.buff_wr_ready*) changes from 0 to 1. 1: Ready to write buffer 0: Not ready to write buffer
3	dma	R/W1C	0	**DMA Interrupt** Set when the SDHC encounters the DMA buffer boundary set in the *SDHC_BLK_SIZE*.trans field during a SDMA transfer. The Card Driver must update the *SDHC_SDMA* register with the address of the next block to transfer before the SDHC continues the transfer. 1: SDHC DMA Interrupt is generated 0: No SDHC DMA Interrupt
2	blk_gap_event	R/W1C	0	**Block Gap Event** If the Stop at Block Gap Request field is set in the Block Gap Control register (*SDHC_BLK_GAP*.stop), this bit is set when a read or write transaction is stopped at a block gap. If Stop at Block Gap Request is not set to 1, then this bit is not meaningless. 1: Transaction stopped at block gap 0: No Block Gap Event
1	trans_comp	R/W1C	0	**Transfer Complete** Set when a read/write transfer and a command with busy is complete. This bit has higher priority than Data Timeout Error. If both bits are set to 1, execution of a command is complete. See *Table 8-74* for Transfer Complete and Data Timeout Error priority and meaning. 1: Command execution is complete 0: Not complete *Note: This field is not set while performing a tuning procedure (SDHC_HOST_CN_2.execute = 1).*

Normal Interrupt Status Register			SDHC_INT_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
0	cmd_comp	R/W1C	0	**Command Complete** Set when the end bit of the command response is received. Auto CMD12 and Auto CMD23 consist of two responses. This flag is not set by the card's response to the CMD12 or CMD23, but by the card's response to the read or write command you send to complete the Auto CMD12 or Auto CMD23. See Command Inhibit (CMD) in the Present State (*SDHC_PRESENT.cmd*) register for how to control this bit. *Table 8-75* illustrates the relationship between Command Complete and Command Timeout Error bits. If both bits are set, then the response was not received within 64 SD clock cycles. 1: Command execution is complete 0: Not complete

Table 8-74: Transfer Complete and Data Timeout Error Priority and Status

Transfer Complete *SDHC_INT_STAT.trans_comp*	Data Timeout Error *SDHC_ER_INT_STAT.data_to*	Status
0	0	Interrupted by another event
0	1	Timeout occurred during transfer
1	N/A	Command execution complete

Table 8-75: Command Complete and Command Timeout Error Priority and Status

Transfer Complete *SDHC_INT_STAT.cmd_comp*	Data Time Error *SDHC_ER_INT_STAT.cmd_to*	Status
0	0	Interrupted by another event.
N/A	1	Response not received within 64 SD Clock cycles.
1	0	Response received.

8.5.6.2 Error Interrupt Status Register

The interrupts defined in this register are enabled by the corresponding fields in the Error Interrupt Status Enable (*SDHC_ER_INT_EN*) register. Setting any field in the *SDHC_ER_INT_SIGNAL* register enables SDHC error interrupt generation using the SDHC_IRQ. The interrupt occurs when any field in the *SDHC_ER_INT_STAT* register is set to 1.

Table 8-76: SDHC Error Interrupt Status Register

Error Interrupt Status Register			SDHC_ER_INT_STAT	[0x0032]
Bits	**Name**	**Access**	**Reset**	**Description**
15:13	-	R/W1C	0	**Reserved for Future Use** Do not modify this field.
12	dma	R/W1C	0	**DMA Error** Error in SDMA transaction 1: Error 0: No error
11:10	-	R/W1C	0	**Reserved for Future Use** Do not modify this field.

Error Interrupt Status Register	SDHC_ER_INT_STAT	[0x0032]

Bits	Name	Access	Reset	Description
9	adma	R/W1C	0	**ADMA Error** Set when the SDHC detects an error during an ADMA data transfer. The state of the ADMA when the error occurs is saved in the ADMA Error Status (*SDHC_ADMA_ER*) register. This bit is also set if the SDHC detects invalid descriptor data. If the *SDHC_ADMA_ER* register indicates an ADMA Error State, then an invalid descriptor was detected. 1: Error 0: No error
8	auto_cmd_12	R/W1C	0	**Auto CMD Error** Auto CMD12 and Auto CMD23 use this error status. This bit is set when detecting that one of the bits D00-D04 in Auto CMD Error Status (*SDHC_AUTO_CMD_ER*) register changed from a 0 to a 1. 1: Error 0: No error *Note: For Auto CMD12, this bit is set to 1 not only when an error occurs in Auto CMD12, but also when Auto CMD12 is not executed due to a previous command error.*
7	current_limit	R/W1C	0	**Current Limit Error** Not supported on MAX32665/MAX32666.
6	data_end_bit	R/W1C	0	**Data End Bit Error** Set if a 0 is detected at the end bit position of read data that uses the DAT line or the end-bit position of the CRC status. 1: Error 0: No error
5	data_crc	R/W1C	0	**Data CRC Error** Set when a CRC error is detected when receiving read data that uses the DAT line or when detecting a Write CRC status with a value other than 010. 1: Error 0: No error
4	data_to	R/W1C	0	**Data Timeout Error** Set for any of the following timeout conditions: • Busy timeout for R1b and R5b response types. See *Table 8-53* for more information about response types. • Busy timeout after Write CRC status • Write CRC status Timeout • Read Data Timeout 1: Error 0: No error
3	cmd_idx	R/W1C	0	**Command Index Error** Set if a Command Index error is detected in the Command Response. 1: Error 0: No error

Error Interrupt Status Register	SDHC_ER_INT_STAT	[0x0032]

Bits	Name	Access	Reset	Description
2	cmd_end_bit	R/W1C	0	**Command End Bit Error** Set if the end bit of a Command Response is 0. 1: Error 0: No error
1	cmd_crc	R/W1C	0	**Command CRC Error** Set for the following cases: 1) If a response is returned, and the Command Timeout Error is set to 0, then this error flag is set if a CRT error is detected in the Command Response. 2) The SDHC detects a CMD-line conflict by monitoring the *SDHC_CMD* line when a command is issued. The SDHC sets the Command Timeout Error flag if a CMD line conflict is detected. A CMD line conflict indicates the CMD line was driven to 1, and the SDHC detected a 0 on the CMD line on the next SDCLK. 1: Error 0: No error
0	cmd_to	R/W1C	0	**Command Timeout Error** Set if there is not response within 64 SDCLK cycles from the end bit of a command. 1: Error 0: No error *Note: If both the* SDHC_ER_INT_STAT.cmd_crc *and* SDHC_ER_INT_STAT.cmd_to *flags are set, then the SDHC detected a CMD-line conflict. See* SDHC_ER_INT_STAT.cmd_crc *for more information about a CMD-line conflict.*

Table 8-77: SDHC Normal Interrupt Status Register

Normal Interrupt Status Enable Register	SDHC_INT_EN	[0x0034]

Bits	Name	Access	Reset	Description
15:13	-	R/W	0	**Reserved for Future Use** Do not modify this field.
12	retuning	R/W		**Re-Tuning Event Status Enable** 1: Enabled 0: Disabled
11:9	-	R/W	0	**Reserved for Future Use** Do not modify this field.
8	card_int	R/W	0	**Card Interrupt Status Enable** Set to enable card-interrupt detection. The Card Driver should clear this bit prior to servicing a card interrupt status event and re-enable this bit after all interrupts from the card are serviced. 1: Enabled 0: Disabled
7	card_removal	R/W	0	**Card Removal Status Enable** Set to enable card removal event. 1: Enabled 0: Disabled

Normal Interrupt Status Enable Register			SDHC_INT_EN	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
6	card_insert	R/W	0	**Card Insertion Status Enable** Set to enable card insertion event. 1: Enabled 0: Disabled
5	buffer_rd	R/W	0	**Buffer Read Ready Status Enable** Set to enable Buffer Read Ready status. 1: Enabled 0: Disabled
4	buffer_wr	R/W	0	**Buffer Write Ready Status Enable** Set to enable Buffer Write Ready status. 1: Enabled 0: Disabled
3	dma	R/W	0	**DMA Interrupt Status Enable** Set to enable DMA status. 1: Enabled 0: Disabled
2	blk_gap	R/W	0	**Block Gap Event Status Enable** Set to enable Block Gap status. 1: Enabled 0: Disabled
1	trans_comp	R/W	0	**Transfer Complete Status Enable** Set to enable Transfer Complete status. 1: Enabled 0: Disabled
0	cmd_comp	R/W	0	**Command Complete Status Enable** Set to enable Command Complete status. 1: Enabled 0: Disabled

Table 8-78: SDHC Error Interrupt Status Enable Register

Error Interrupt Status Enable Register			SDHC_ER_INT_EN	[0x0036]
Bits	**Name**	**Access**	**Reset**	**Description**
15:13	-	R/W	0	**Reserved for Future Use** Do not modify this field.
12	vendor	R/W	0	**Target Response Error/Host Error Status Enable** Set to enable Target Response/Host Error status interrupts. 1: Enabled 0: Disabled
11	-	R/W	0	**Reserved for Future Use** Do not modify this field.
10	tuning	R/W	0	**Tuning Error Status Interrupt Enable** 1: Enabled 0: Disabled

Error Interrupt Status Enable Register			SDHC_ER_INT_EN	[0x0036]
Bits	**Name**	**Access**	**Reset**	**Description**
9	adma	R/W	0	**ADMA Error Status Interrupt Enable** 1: Enabled 0: Disabled
8	auto_cmd	R/W	0	**Auto CMD12 Error Status Interrupt Enable** 1: Enabled 0: Disabled
7	-	R/W	0	**Reserved for Future Use** Do not modify this field.
6	data_end_bit	R/W	0	**Data End Bit Error Status Interrupt Enable** 1: Enabled 0: Disabled
5	data_crc	R/W	0	**Data CRC Error Status Interrupt Enable** 1: Enabled 0: Disabled
4	data_to	R/W	0	**Data Timeout Error Status Interrupt Enable** 1: Enabled 0: Disabled
3	cmd_idx	R/W	0	**Command Index Error Status Interrupt Enable** 1: Enabled 0: Disabled
2	cmd_end_bit	R/W	0	**Command End Bit Error Status Interrupt Enable** 1: Enabled 0: Disabled
1	cmd_crc	R/W	0	**Command CRC Error Status Interrupt Enable** 1: Enabled 0: Disabled
0	cmd_to	R/W	0	**Command Timeout Error Status Interrupt Enable** 1: Enabled 0: Disabled

Table 8-79: SDHC Normal Interrupt Signal Enable Register

Normal Interrupt Signal Enable Register			SDHC_INT_SIGNAL	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**
15:13	-	R/W	0	**Reserved for Future Use** Do not modify this field.
12	retuning	R/W	0	**Re-Tuning Event Signal Enable** 1: Enabled 0: Disabled
11:9	-	R/W	0	**Reserved for Future Use** Do not modify this field.
8	card_int	R/W	0	**Card Interrupt Signal Enable** 1: Enabled 0: Disabled

Normal Interrupt Signal Enable Register			SDHC_INT_SIGNAL	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**
7	card_removal	R/W	0	**Card Removal Signal Enable** 1: Enabled 0: Disabled
6	card_insert	R/W	0	**Card Insertion Signal Enable** 1: Enabled 0: Disabled
5	buffer_rd	R/W	0	**Buffer Read Ready Signal Enable** 1: Enabled 0: Disabled
4	buffer_wr	R/W	0	**Buffer Write Ready Signal Enable** 1: Enabled 0: Disabled
3	dma	R/W	0	**DMA Interrupt Signal Enable** 1: Enabled 0: Disabled
2	blk_gap	R/W	0	**Block Gap Signal Enable** 1: Enabled 0: Disabled
1	trans_comp	R/W	0	**Transfer Complete Signal Enable** 1: Enabled 0: Disabled
0	cmd_comp	R/W	0	**Command Complete Signal Enable** 1: Enabled 0: Disabled

Table 8-80: SDHC Error Interrupt Signal Enable Register

Error Interrupt Signal Enable Register			SDHC_ER_INT_SIGNAL	[0x003A]
Bits	**Name**	**Access**	**Reset**	**Description**
15:13	-	R/W	0	**Reserved for Future Use** Do not modify this field.
12	tar_resp	R/W	0	**Target Response Error Signal Enable** 1: Enabled 0: Disabled
11	-	R/W	0	**Reserved for Future Use** Do not modify this field.
10	tuning	R/W	0	**Tuning Error Signal Enable** 1: Enabled 0: Disabled
9	adma	R/W	0	**ADMA Error Signal Enable** 1: Enabled 0: Disabled
8	auto_cmd	R/W	0	**Auto CMD12 Error Signal Enable** 1: Enabled 0: Disabled

Error Interrupt Signal Enable Register			SDHC_ER_INT_SIGNAL	[0x003A]
Bits	**Name**	**Access**	**Reset**	**Description**
7	curr_lim	R/W	0	**Current Limit Error Signal Enable** 1: Enabled 0: Disabled
6	data_end_bit	R/W	0	**Data End Bit Error Signal Enable** 1: Enabled 0: Disabled
5	data_crc	R/W	0	**Data CRC Error Signal Enable** 1: Enabled 0: Disabled
4	data_to	R/W	0	**Data Timeout Error Signal Enable** 1: Enabled 0: Disabled
3	cmd_idx	R/W	0	**Command Index Error Signal Enable** 1: Enabled 0: Disabled
2	cmd_end_bit	R/W	0	**Command End Bit Error Signal Enable** 1: Enabled 0: Disabled
1	cmd_crc	R/W	0	**Command CRC Error Signal Enable** 1: Enabled 0: Disabled
0	cmd_to	R/W	0	**Command Timeout Error Signal Enable** 1: Enabled 0: Disabled

8.5.6.3 Auto CMD Error Status Register

This register is used to indicate response errors for Auto CMD12 and Auto CMD23. The contents of this register are only valid when the Auto CMD Error is set (*SDHC_ER_INT_STAT.auto_cmd_12*). For Auto CMD23 errors, the error code is stored in *SDHC_AUTO_CMD_ER*[4:1].

Table 8-81: SDHC Auto CMD Error Status Register

Auto CMD Error Status Register			SDHC_AUTO_CMD_ER	[0x003C]
Bits	**Name**	**Access**	**Reset**	**Description**
15:8	-	R/W	0	**Reserved for Future Use** Do not modify this field.
7	not_issued	R	0	**Command Not Issued by Auto CMD12 Error** 1: Command not issued due to Auto CMD12 error as indicated in bits 4:1 of this register. 0: Auto CMD Error issued by Auto CMD23
6:5	-	R/W0	0	**Reserved for Future Use** Do not modify this field.
4	index	RC	0	**Auto CMD Index Error** Command Index error occurred in response to a command. 1: Command Index Error 0: No Error

Auto CMD Error Status Register			SDHC_AUTO_CMD_ER	[0x003C]
Bits	**Name**	**Access**	**Reset**	**Description**
3	end_bit	RC	0	**Auto CMD End Bit Error** Set if the end bit of the Command Response is 0. 1: End Bit Error 0: No Error
2	crc	RC	0	**Auto CMD CRC Error** Set if CRC error in command response. 1: CRC Error 0: No Error *Note: If both SDHC_AUTO_CMD_ER.crc and SDHC_AUTO_CMD_ER.to are set, then a CMD-line conflict occurred.*
1	to	RC	0	**Auto CMD Timeout Error** Set if no response is returned within 64 SDCLK cycles from the end bit of the command. If set, then ignore bits 4:2 of this register. 1: Timeout Error 0: No Error *Note: If both SDHC_AUTO_CMD_ER.crc and SDHC_AUTO_CMD_ER.to are set, then a CMD-line conflict occurred.*
0	not_executed	RC	0	**Auto CMD12 Not Executed Error** Auto CMD12 was not issued to stop a multi-block memory transfer due to an error with a prior command. 1: Not Executed 0: No Error or error generated by Auto CMD23

Table 8-82: SDHC Host Control 2 Register

Host Control 2 Register			SDHC_HOST_CN_2	[0x003E]
Bits	**Name**	**Access**	**Reset**	**Description**
15	preset_val_en	R/W	0	**Preset Value Enable** When set to 0, the following fields must be set by the Card Driver: • SDCLK Frequency Select (*SDHC_CLK_CN.sdclk_freq_sel*) • Clock Generator Select (*SDHC_CLK_CN.clk_gen_sel*) • Driver Strength Select (*SDHC_HOST_CN_2.driver_strength*) If set to 1, the Host Controller hardware sets the above fields based on the values in the Preset Value registers. 0: Card Driver must set the SDCLK Frequency Select, Clock Generator Select and Driver Strength Select fields. 1: The Host Controller hardware sets the above fields using the Preset Value register settings.
14	asynch_int	R/W	0	**Asynchronous Interrupt Enable** Always reads 0. Asynchronous Interrupt Enable is not supported by the MAX32665/MAX32666. Writes to this field have no effect.
13:8	-	R/W	0	**Reserved for Future Use** Do not modify this field.

Host Control 2 Register			SDHC_HOST_CN_2	[0x003E]
Bits	**Name**	**Access**	**Reset**	**Description**
7	sampling_clk	R/W	0	**Sampling Clock Select** This field is automatically set by hardware when Execute Tuning (*SDHC_HOST_CN_2*. execute) is cleared. 0: The fixed clock is used to sample data 1: The tuned clock is used to sample data *Note: The Card Driver cannot write 1 to this bit. Writing this bit to 0 can only be done if the Host Controller is not receiving a response or reading a data block.*
6	execute	R/W	0	**Execute Tuning** Setting this bit to 1 starts the tuning procedure and the bit is automatically cleared when the Host Controller completes the tuning procedure. Writing a 0 to this bit when it is set to 1 aborts the tuning procedure. 1: Execute tuning 0: Tuning complete or not tuned
5:4	driver_strength	R/W	0	**Driver Strength Select** If using 3.3V signaling, this field is ignored. For 1.8V signaling, the output driver strength is set using this field. If *SDHC_HOST_CN_2.preset_val_en* = 0, this field is controlled by the Host Driver. If *SDHC_HOST_CN_2.preset_val_en* = 1, this field is automatically set by the hardware using the Preset Value registers. 00b: Driver Type B is selected 01b: Driver Type A is selected 10b: Driver Type C is selected 11b: Driver Type D is selected
3	signal_v1_8	R/W	0	**1.8V Signaling Enable** If the card inserted supports UHS-I, this bit can be set to 1. No matter the value set, 3.3V is used for the card's supply. 1: 1.8V signaling 0: 3.3V signaling
2:0	uhs	R/W	0	**UHS Mode Select** Used to select the UHS-I mode. This field is only used if 1.8V signaling is set to 1 (*SDHC_HOST_CN_2.signal_v1_8* = 1). 000b: SDR12 001b: SDR25 010b: SDR50 011b: SDR104 (Not supported) 100b: DDR50 101b – 111b: Reserved for Future Use

Table 8-83: SDHC Capabilities Register 0

Capabilities Register 0			SDHC_CFG_0	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
31:30	slot_type	R	0b00	**Slot Type** 0b00: Support for a single slot with support for a removable card
29	async_int	R	1	**Asynchronous Interrupt Support** 1: Asynchronous Interrupt Supported
28	bit_64_sys_bus	R	0	**64-bit System Bus Support** 0: 64-bit system bus not supported

Capabilities Register 0			SDHC_CFG_0	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
27	-	R	0	**Reserved for Future Use**
26	v1_8	R	1	**Voltage Support 1.8V** 1: 1.8V supported
25	v3_0	R	1	**Voltage Support 3.0V** 1: 3.0V supported
24	v3_3	R	1	**Voltage Support 3.3V** 1: 3.3V supported
23	suspend	R	1	**Suspend/Resume Support** 1: Suspend / Resume functionality is supported
22	sdma	R	1	**SDMA Support** SDMA is supported and can transfer data between system memory and the SDHC directly. 1: SDMA supported
21	hs	R	1	**High Speed Support** The SDHC supports High Speed mode with $f_{PCLK} = 96MHz/2$. 1: High speed mode supported
20	-	R	0	**Reserved for Future Use** Do not modify this field.
19	adma2	R	1	**ADMA2 Support** The SDHC supports ADMA2. 1: ADMA2 supported
18	bit_8	R	0	**8-bit Support for Embedded Device** The SDHC supports 8-bit bus width mode. 0: 8-bit Bus width not supported
17:16	max_blk_len	R	0b10	**Max Block Length** This value indicates the maximum block size that the Host Driver can read and write to the buffer in the SDHC without wait cycles. The transfer block length is always 512 bytes for SD memory cards regardless of this field. 0b10: 2048 bytes
15:8	to_clk_freq	R	0x00	**Base Clock Frequency for SD Clock** 0x00: Get information using another method
7	to_clk_unit	R	1	**Timeout Clock Unit** 1: MHz base clock unit
6	-	R	0	**Reserved for Future Use** Do not modify this field.
5:0	clk_freq	R	0x01	**Timeout Clock Frequency** The base clock frequency used to detect Data Timeout errors. The Timeout Clock Unit defines the units of this field's value. 1: 1 [MHz]

Table 8-84: SDHC Capabilities Register 1

Capabilities Register 1			SDHC_CFG_1	[0x0044]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	R	1	**Reserved for Future Use** Do not modify this field.
23:16	clk_multi	R	0	**Clock Multiplier** Always reads 0x00. 0: Programmable clock generation is not supported.
15:14	retuning	R	0	**Re-Tuning Modes** Always reads 0b00. The SDHC supports Mode 1 Re-Tuning only with timer controlled by the host driver and a maximum of 4MB data length.
13	tuning_sdr50	R	0	**Use Tuning for SDR50** 1: Tuning required for SDR50 0: SDR50 does not require tuning
12	-	R	0	**Reserved for Future Use** Do not modify this field.
11:8	timer_cnt_tuning	R	0	**Timer Count for Re-Tuning** 0x0: Re-Tuning Timer disabled 0x1: 1 second 0x2: 2 seconds 0x3: 4 seconds 0x4: 8 seconds ….: …… n: $2^{(n-1)}$ $seconds$ ….: …… 0xB: 1024 seconds 0xC: Reserved 0xD: Reserved 0xE: Reserved 0xF: Get information from another source
7	-	R	0	**Reserved for Future Use** Do not modify this field.
6	driver_d	R	1	**Driver Type D Support** 1: Driver Type D is supported
5	driver_c	R	1	**Driver Type C Support** 1: Driver Type C is supported
4	driver_a	R	1	**Driver Type A Support** 1: Driver Type A is supported
3	-	R	0	**Reserved for Future Use** Do not modify this field.
2	ddr50	R	1	**DDR50 Support** 1: DDR50 is supported
1	sdr104	R	1	**SRD104** 1: SDR104 is supported
0	sdr50	R	1	**SDR50** 1: SDR50 is supported

Table 8-85: SDHC Maximum Current Capabilities Register

Maximum Current Capabilities Register			SDHC_MAX_CURR_CFG	[0x0048]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	R	0	**Reserved for Future Use** Do not modify this field.
23:16	v1_8	R	0	**Maximum Current for 1.8V** 0x00: System dependent
15:8	v3_0	R	0	**Maximum Current for 3.0V** 0x00: System dependent
7:0	v3_3	R	0	**Maximum Current for 3.3V** 0x00: System dependent

Table 8-86: SDHC Force Event Register for Auto CMD Error Status Register

Force Event Register for Auto CMD Error Status			SDHC_FORCE_CMD	[0x0050]
Bits	**Name**	**Access**	**Reset**	**Description**
15:8	-	W	0	**Reserved for Future Use** Do not modify this field.
7	not_issued	W	0	**Force Event for Command Not Issued By Auto CMD12 Error** 1: Interrupt is generated 0: No interrupt generated
6:5	-	W	0	**Reserved for Future Use** Do not modify field.
4	index	W	0	**Force Event for Auto CMD Index Error** 1: Interrupt is generated 0: No interrupt generated
3	end_bit	W	0	**Force Event for Auto CMD End Bit Error** 1: Interrupt is generated 0: No interrupt generated
2	crc	W	0	**Force Event for Auto CMD CRC Error** 1: Interrupt is generated 0: No interrupt generated
1	to	W	0	**Force Event for Auto CMD Timeout Error** 1: Interrupt is generated 0: No interrupt generated
0	not_excu	W	0	**Force Event for Auto CMD12 Not Executed** 1: Interrupt is generated 0: No interrupt generated

Table 8-87: SDHC Force Event Register for Error Interrupt Status

Force Event Register for Error Interrupt Status			SDHC_FORCE_EVENT_INT_STAT	[0x0052]
Bits	**Name**	**Access**	**Reset**	**Description**
15:12	vendor	R/W	0	**Force Event for Vendor Specific Error Status** 1: Interrupt is generated 0: No interrupt generated

Force Event Register for Error Interrupt Status		SDHC_FORCE_EVENT_INT_STAT		[0x0052]
Bits	**Name**	**Access**	**Reset**	**Description**
11:10	-	R/W	0	**Reserved for Future Use** Do not modify this field.
9	adma	R/W	0	**Force Event for ADMA Error** 1: Interrupt is generated 0: No interrupt generated
8	auto_cmd	R/W	0	**Force Event for Auto CMD Error** 1: Interrupt is generated 0: No interrupt generated
7	curr_limit	R/W	0	**Force Event for Current Limit Error** 1: Interrupt is generated 0: No interrupt generated
6	data_end_bit	R/W	0	**Force Event for Data End Bit Error** 1: Interrupt is generated 0: No interrupt generated
5	data_crc	R/W	0	**Force Event for Data CRC Error** 1: Interrupt is generated 0: No interrupt generated
4	data_to	R/W	0	**Force Event for Data Timeout Error** 1: Interrupt is generated 0: No interrupt generated
3	cmd_index	R/W	0	**Force Event for Command Index Error** 1: Interrupt is generated 0: No interrupt generated
2	cmd_end_bit	R/W	0	**Force Event for Command End Bit Error** 1: Interrupt is generated 0: No interrupt generated
1	cmd_crc	R/W	0	**Force Event for Command CRC Error** 1: Interrupt is generated 0: No interrupt generated
0	cmd_to	R/W	0	**Force Event for Command Timeout Error** 1: Interrupt is generated 0: No interrupt generated

Table 8-88: SDHC ADMA Error Status Register

ADMA Error Status Register		SDHC_ADMA_ER		[0x0054]
Bits	**Name**	**Access**	**Reset**	**Description**
7:3	-	R	0	**Reserved for Future Use** Do not modify this field.

<table>
<tr><th colspan="3">ADMA Error Status Register</th><th>SDHC_ADMA_ER</th><th>[0x0054]</th></tr>
<tr><th>Bits</th><th>Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td>2</td><td>len_mismatch</td><td>R/W0</td><td>0</td><td>ADMA Length Mismatch Error
This error occurs in the following two cases:
1) When setting Block Count Enable, the total data length specified by the Descriptor Table is different from that specified by the Block Count and Block Length fields.
2) Total data length is not divisible by the Block Length field.

1: Error
0: No Error</td></tr>
<tr><td>1:0</td><td>state</td><td>R/W0</td><td>0b00</td><td>ADMA Error State
The state of the ADMA when the error condition occurred. Only valid during data transfer for ADMA.

The following table shows the possible state values, the associated ADMA Error State, and the contents of the SDHC_SDMA register.
<table>
<tr><th>state</th><th>ADMA Error State when the error occurred</th><th>SYS_SDR register contents</th></tr>
<tr><td>0b00</td><td>ST_STOP (Stop DMA)</td><td>Points next to the error descriptor</td></tr>
<tr><td>0b01</td><td>ST_FDS (Fetch Descriptor)</td><td>Points to the error descriptor</td></tr>
<tr><td>0b10</td><td>N.A.</td><td>N.A.</td></tr>
<tr><td>0b11</td><td>ST_TFR (Transfer Data)</td><td>Points next to the error descriptor</td></tr>
</table>
Note: 0b10 is not a valid error state and is never set.</td></tr>
</table>

Table 8-89: SDHC ADMA System Address Register 0

ADMA System Address Register 0			SDHC_ADMA_ADDR_0	[0x0058]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**ADMA System Address 0** Holds the byte address of the executing command for the Descriptor Table. The Host Driver must set this address, made up of <*SDHC_ADMA_ADDR_1*>:<*SDHC_ADMA_ADDR_0*>, to the start address of the Descriptor Table. The ADMA increments this register address when fetching a descriptor line to point to the next address. When an ADMA Error Interrupt occurs, this register holds a valid descriptor address depending on the ADMA state. The following table shows the 64-bit System Address for ADMA using <*SDHC_ADMA_ADDR_1*>:<*SDHC_ADMA_ADDR_0*>.

SDHC_ADMA_ADDR_1	*SDHC_ADMA_ADDR_0*	64-bit System Address
0x0000 0000	0x0000 0000	0x00000000_00000000
0x0000 0000	0x0000 0004	0x00000000_00000004
0x0000 0000	0x0000 0008	0x00000000_00000008
0x0000 0000	0x0000 000C	0x00000000_0000000C
…….	…….	…….
0xFFFF FFFF	0xFFFF FFFC	0xFFFFFFFF_FFFFFFFC

Note: The Host Driver must program the Descriptor Table on 32-bit boundaries and set the 32-bit boundary address to this register. ADMA2 ignores the lower two bits of this register, assuming it to be 00b.

Table 8-90: SDHC ADMA System Address Register 1

ADMA System Address Register 1			SDHC_ADMA_ADDR_1	[0x005C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**ADMA System Address 1** Most-significant word for the 64-bit ADMA address. See *SDHC_ADMA_ADDR_0* for details.

8.5.6.4 Preset Value Registers

All Preset Value registers (*SDHC_PRESET_0* to *SDHC_PRESET_7*) contain the same fields as described in the *SDHC_PRESET_0* register. One of the Preset Value registers is automatically selected by the SDHC based on the selected bus-speed mode

Table 8-91 shows a group of preset values per card or device. One of the Preset Value registers (*SDHC_PRESET_1* – *SDHC_PRESET_7*) is selected by the SDHC hardware based on the Selected Bus Speed mode. *Table 8-92* defines the conditions to select one of the Preset Value registers.

Table 8-91: Preset Value Register Example

Offset	Preset Value Registers	Signal Voltage
[0x0060]	Preset Value for Initialization	3.3V or 1.8V
[0x0062]	Preset Value for Default Speed	3.3V
[0x0064]	Preset Value for High Speed	3.3V
[0x0066]	Preset Value for SDR12	1.8V

Offset	Preset Value Registers	Signal Voltage
[0x0068]	Preset Value for SDR25	1.8V
[0x006A]	Preset Value for SDR50	1.8V
[0x006C]	Preset Value for SDR104	1.8V
[0x006E]	Preset Value for DDR50	1.8V

Table 8-92: Preset Value Register Selection Conditions

Selected Bus Speed Mode	1.8V Signaling Enable *SDHC_HOST_CN_2*.signal_v1_8l	High Speed Enable *SDHC_HOST_CN_1*. hs_en	UHS-I Mode Selection *SDHC_HOST_CN_2*.uhs
Default Speed	0	0	N/A
High Speed	0	1	N/A
SDR12	1	N/A	0b000
SDR25	1	N/A	0b001
SDR50	1	N/A	0b010
SDR104	1	N/A	0b011
DDR50	1	N/A	0b100
Reserved	1	N/A	0b101 to 0b111

Table 8-93: SDHC Preset Value 0 to Preset Value 7 Registers

Preset Value 0 for Initialization	**SDHC_PRESET_0**	**[0x0060]**
Preset Value 1 for Initialization	**SDHC_PRESET_1**	**[0x0062]**
Preset Value 2 for Initialization	**SDHC_PRESET_2**	**[0x0064]**
Preset Value 3 for Initialization	**SDHC_PRESET_3**	**[0x0066]**
Preset Value 4 for Initialization	**SDHC_PRESET_4**	**[0x0068]**
Preset Value 5 for Initialization	**SDHC_PRESET_5**	**[0x006A]**
Preset Value 6 for Initialization	**SDHC_PRESET_6**	**[0x006C]**
Preset Value 7 for Initialization	**SDHC_PRESET_7**	**[0x006E]**

Bits	Name	Access	Reset	Description
15:14	driver_strength	R	1	**Driver Strength Select Value** Driver strength is supported by 1.8V signaling bus speed modes. This field is not used for 3.3V signaling. 0b00: Driver Type B is selected 0b01: Driver Type A is selected 0b10: Driver Type C is selected 0b11: Driver Type D is selected
13:11	-	R	0	**Reserved for Future Use** Do not modify this field.
10	clk_gen	R	0	**Clock Generator Select Value** 0: Programmable clock generator is not supported
9:0	sdclk_freq	R	-	**SDCLK Frequency Select Value** 10-bit preset value to set the SDCLK Frequency Select field in the Clock Control register (*SDHC_CLK_CN.upper_sdclk_freq_sel* and *SDHC_CLK_CN.sdclk_freq_sel*)

Table 8-94: SDHC Slot Interrupt Status Register

Slot Interrupt Status Register			SDHC_SLOT_INT	[0x00FC]
Bits	**Name**	**Access**	**Reset**	**Description**
15:8	-	R	0	**Reserved for Future Use** Do not modify this field.
7:1	-	R	0	**Reserved for Future Use** Do not modify this field.
0	int_signals	R	0	**Interrupt Signals** Indicates the logical OR of Interrupt Signal and Wakeup Signal for the single slot. Only one slot is defined for the MAX32665/MAX32666, slot 0. Reset by POR and by software reset for all (*SDHC_SW_RESET.reset_all*).

Table 8-95: SDHC Host Controller Version Register

Host Controller Version Register			SDHC_HOST_CN_VER	[0x00FE]
Bits	**Name**	**Access**	**Reset**	**Description**
15:8	vend_ver	R	-	**Vendor Version** This status is reserved for the vendor version number. The Host Driver should not use this status.
7:0	spec_ver	R	0x02	**Specification Version Number** This status indicates the Host Controller Specification Version. 0x02: SD Host Specification Version 3.00

9. Standard DMA (DMA)

The Standard Direct Memory Access controller (DMA) is a hardware feature that provides the ability to perform high-speed, block memory transfers of data independent of an Arm core. All DMA transactions consist of burst read from the source into the internal DMA FIFO followed by a burst write from the internal DMA FIFO to the destination.

DMA transfers are one of three types:

- From a receive FIFO to a memory address
- To a transmit FIFO from a memory address, or
- From a source memory address to a destination memory address.

The DMA supports multiple channels. Each channel provides the following features:

- Full 32-bit source and destination addresses with 24-bit (16 Mbytes) address increment capability
- Ability to chain DMA buffers when a count-to-zero (CTZ) condition occurs
- Up to 16 Mbytes for each DMA transfer
- 8 x 32 byte transmit and receive FIFO
- Programmable channel timeout period
- Programmable burst size
- Programmable priority
- Interrupt upon CTZ
- Abort on error

9.1 Instances

There are two instances of the DMA, generically referred to as DMAm. Each instance provides 8 channels, generically referred to as DMA_CHn. Each instance of the DMA has a set of interrupt registers common to all its channels, and a set of registers unique to each channel instance.

Table 9-1: MAX32665/MAX32666 DMA and Channel Instances

DMAm Instance	DMA_CHn Channel Instance
DMA0	DMA_CH0
	DMA_CH1
	DMA_CH2
	DMA_CH3
	DMA_CH4
	DMA_CH5
	DMA_CH6
	DMA_CH7
DMA1	DMA_CH0
	DMA_CH1
	DMA_CH2
	DMA_CH3
	DMA_CH4
	DMA_CH5
	DMA_CH6
	DMA_CH7

9.2 DMA Channel Operation (DMA_CH)

9.2.1 DMA Channel Arbitration and DMA Bursts

DMA contains an internal arbiter that allows enabled channels to access the AHB and move data. Once a channel is programmed and enabled, it generates a request to the arbiter immediately (for memory-to-memory DMA) or whenever its associated peripheral requests DMA (for memory-to-peripheral or peripheral-to-memory DMA).

Granting is done based on priority—a higher priority request is always granted. Within a given priority level, requests are granted on a round-robin basis. The *DMA_CHn_CFG.priority* field determines the DMA channel priority.

When a channel's request is granted, it runs a DMA transfer. The arbiter grants requests to a single channel at a time. Once the DMA transfer completes, the channel relinquishes its grant.

A DMA channel is enabled using the *DMA_CHn_CFG.chen* bit.

When disabling a channel, poll the *DMA_CHn_ST*.ch_st bit to determine if the channel is truly disabled. In general, *DMA_CHn_ST*.ch_st follows the setting of the *DMA_CHn_CFG*.chen bit. However, the *DMA_CHn_ST*.ch_st bit is automatically cleared under the following conditions:

- Bus error (cleared immediately)
- CTZ when the *DMA_CHn_CFG.rlden* = 0 (cleared at the end of the AHB R/W burst)
- *DMA_CHn_CFG.chen* bit transitions to 0 (cleared at the end of the AHB R/W burst)

Whenever *DMA_CHn_ST.ch_st* transitions from 1 to 0, the corresponding *DMA_CHn_CFG.chen* bit is also cleared. If an active channel is disabled during an AHB read/write burst, the current burst will continual until completed.

Only an error condition can interrupt an ongoing data transfer.

9.2.2 DMA Source and Destination Addressing

The source and destination for DMA transfers are dictated by the request select dedicated to the peripheral instance. The *DMA_CHn_CFG.reqsel*. field dictates the source and destination for a channel's DMA transfer as shown in *Table 9-2.* The *DMA_CHn_SRC* and *DMA_CHn_DST* registers hold the source and/or destination memory addresses, depending on the specific operation.

The DMA_CHn_CFG*.srcinc field is ignored when the DMA source is a peripheral memory, and the* DMA_CHn_CFG*.dstinc field is ignored when the DMA destination is a peripheral memory.*

Table 9-2: MAX32665/MAX32666 DMA Source and Destination by Peripheral

DMA_CHn_CFG.reqsel	Peripheral	DMA Source	DMA Destination
0x00	Memory-to-Memory	*DMA_CHn_SRC*	*DMA_CHn_DST*
0x01	SPI1	SPI1 Receive FIFO	*DMA_CHn_DST*
0x02	SPI2	SPI2 Receive FIFO	*DMA_CHn_DST*
0x03	Reserved		
0x04	UART0	UART0 Receive FIFO	*DMA_CHn_DST*
0x05	UART1	UART1 Receive FIFO	*DMA_CHn_DST*
0x06	Reserved		
0x07	I2C0	I2C0 Receive FIFO	*DMA_CHn_DST*
0x08	I2C1	I2C1 Receive FIFO	*DMA_CHn_DST*
0x09	ADC	*ADC FIFO*	*DMA_CHn_DST*
0x0A	I2C2	I2C2 Receive FIFO	*DMA_CHn_DST*

DMA_CHn_CFG.reqsel	Peripheral	DMA Source	DMA Destination
0x0B	Reserved		
0x0C	Reserved		
0x0D	Reserved		
0x0E	UART2	UART2 Receive FIFO	DMA_CHn_DST
0x0F	SPI0	SPI0 Receive FIFO	DMA_CHn_DST
0x10	Reserved		
0x1C	Reserved		
0x1D	Reserved		
0x1E	Reserved		
0x1F	Reserved		
0x20	Reserved		
0x21	SPI1	DMA_CHn_SRC	SPI1 Transmit FIFO
0x22	SPI2	DMA_CHn_SRC	SPI2 Transmit FIFO
0x23	Reserved		
0x24	UART0	DMA_CHn_SRC	UART0 Transmit FIFO
0x25	UART1	DMA_CHn_SRC	UART1 Transmit FIFO
0x26	Reserved		
0x27	I2C0	DMA_CHn_SRC	I2C0 Transmit FIFO
0x28	I2C1	DMA_CHn_SRC	I2C1 Transmit FIFO
0x29	Reserved		
0x2A	I2C2	DMA_CHn_SRC	I2C2 Transmit FIFO
0x2B	Reserved		
0x2C	Reserved		
0x2D	Reserved		
0x2E	UART2	DMA_CHn_SRC	UART2 Transmit FIFO
0x2F	SPI0	DMA_CHn_SRC	SPI0 Transmit FIFO
0x30	Reserved		
0x3C	Reserved		
0x3D	Reserved		
0x3E	Reserved		
0x3F	Reserved		

9.2.3 Data Movement from Source to DMA

Table 9-3 shows the fields that control the burst movement of data into the DMA FIFO. The source is a peripheral or memory.

Table 9-3: Data Movement from Source to DMA FIFO

Register/Field	Description	Comments
DMA_CHn_SRC	Source address	If the increment enable is set, this increments on every read cycle of the burst. This field is ignored when the DMA source is a peripheral.
DMA_CHn_CNT	Number of bytes to transfer before a CTZ condition occurs	This register is decremented on each read of the burst.
DMA_CHn_CFG.brst	Burst size (1-32)	This maximum number of bytes moved during the burst read.
DMA_CHn_CFG.srcwd	Source width	This determines the maximum data width used during each read of the AHB burst (byte, two bytes, or four bytes). The actual AHB width might be less if *DMA_CHn_CNT* is not great enough to supply all the needed bytes.
DMA_CHn_CFG.srcinc	Source increment enable	Increments *DMA_CHn_SRC*. This field is ignored when the DMA source is a peripheral.

9.2.4 Data Movement from the DMA to Destination

Table 9-4 shows the fields that control the burst movement of data out of the DMA FIFO. The destination is a peripheral or memory.

Table 9-4: Data Movement from the DMA FIFO to Destination

Register/Field	Description	Comments
DMA_CHn_DST	Destination address	If the increment enable is set, this increments on every write cycle of the burst. This field is ignored when the DMA destination is a peripheral.
DMA_CHn_CFG.brst	Burst size (1-32)	The maximum number of bytes moved during a single AHB read/write burst.
DMA_CHn_CFG.dstwd	Destination width	This determines the maximum data width used during each write of the AHB burst (one byte, two bytes, or four bytes).
DMA_CHn_CFG.dstinc	Destination increment enable	Increments *DMA_CHn_DST*. This field is ignored when the DMA destination is a peripheral.

9.3 Usage

Use the following procedure to perform a DMA transfer from a peripheral's receive FIFO to memory, from memory to a peripheral's transmit FIFO, or from memory to memory:

1. Ensure *DMA_CHn_CFG.chen*, *DMA_CHn_CFG.rlden* = 0, and *DMA_CHn_ST.ctz_st* = 0.
2. If using memory for the destination of the DMA transfer, configure *DMA_CHn_DST* to the starting address of the destination in memory.
3. If using memory for the source of the DMA transfer, configure *DMA_CHn_SRC* to the starting address of the source in memory.
4. Write the number of bytes to transfer to the *DMA_CHn_CNT* register.
5. Configure the following *DMA_CHn_CFG* register fields in one or more instructions. Do not set *DMA_CHn_CFG.chen* to 1 or *DMAn_CNT_RLD.rlden* to 1 in this step:
    a. Configure *DMA_CHn_CFG*.req_sel to select the transfer operation associated with the DMA channel.
    b. Configure *DMA_CHn_CFG*.burst for the desired burst size.
    c. Configure *DMA_CHn_CFG*.priority to set the channel priority relative to other DMA channels.
    d. Configure *DMA_CHn_CFG*.dst_width to dictate the number of bytes written in each transaction.
    e. If desired, set *DMA_CHn_CFG*.dst_inc to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.

f. Configure *DMA_CHn_CFG*.src_width to dictate the number of bytes read in each transaction.

g. If desired, set *DMA_CHn_CFG*.src_inc to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.

h. If desired, set *DMA_CHn_CFG*.*chd_ien* = 1 to generate an interrupt when the channel becomes disabled. The channel becomes disabled when the DMA transfer completes or a bus error occurs.

i. If desired, set *DMA_CHn_CFG*.*ctz_ien* 1 to generate an interrupt when the *DMA_CHn_CNT* register is decremented to zero.

6. If using the reload feature, configure the reload registers to set the destination, source, and count for the following DMA transaction.

   a. If desired, enable the channel timeout feature described in *Channel Timeout Detect*. Clear *DMA_CHn_CFG*.to_prescale to 0x0 to disable the channel timeout feature.

7. Set *DMAn_CNT_RLD*.*rl_den* to 1 to enable the reload feature.
8. Set *DMA_CHn_CFG*.*ch_en* = 1 to immediately start the DMA transfer.
9. Wait for the status bit to become 0 to indicate the completion of the DMA transfer.

9.4 Count-to-Zero (CTZ) Condition

When an AHB channel burst completes, a CTZ condition exists if *DMA_CHn_CNT* is decremented to 0.

At this point, there are two possible responses depending on the value of the *DMA_CHn_CFG*.*rlden*:

1. If *DMA_CHn_CFG*.*rlden* = 1, then the *DMA_CHn_SRC*, *DMA_CHn_DST*, and *DMA_CHn_CNT* registers are loaded from the reload registers, and the channel remains active and continues operating using the newly-loaded address/count values and the previously programmed configuration values.
2. If *DMA_CHn_CFG*.*rlden* = 0, then the channel is disabled, and *DMA_CHn_ST*.*ch_st* is cleared.

9.5 Chaining Buffers

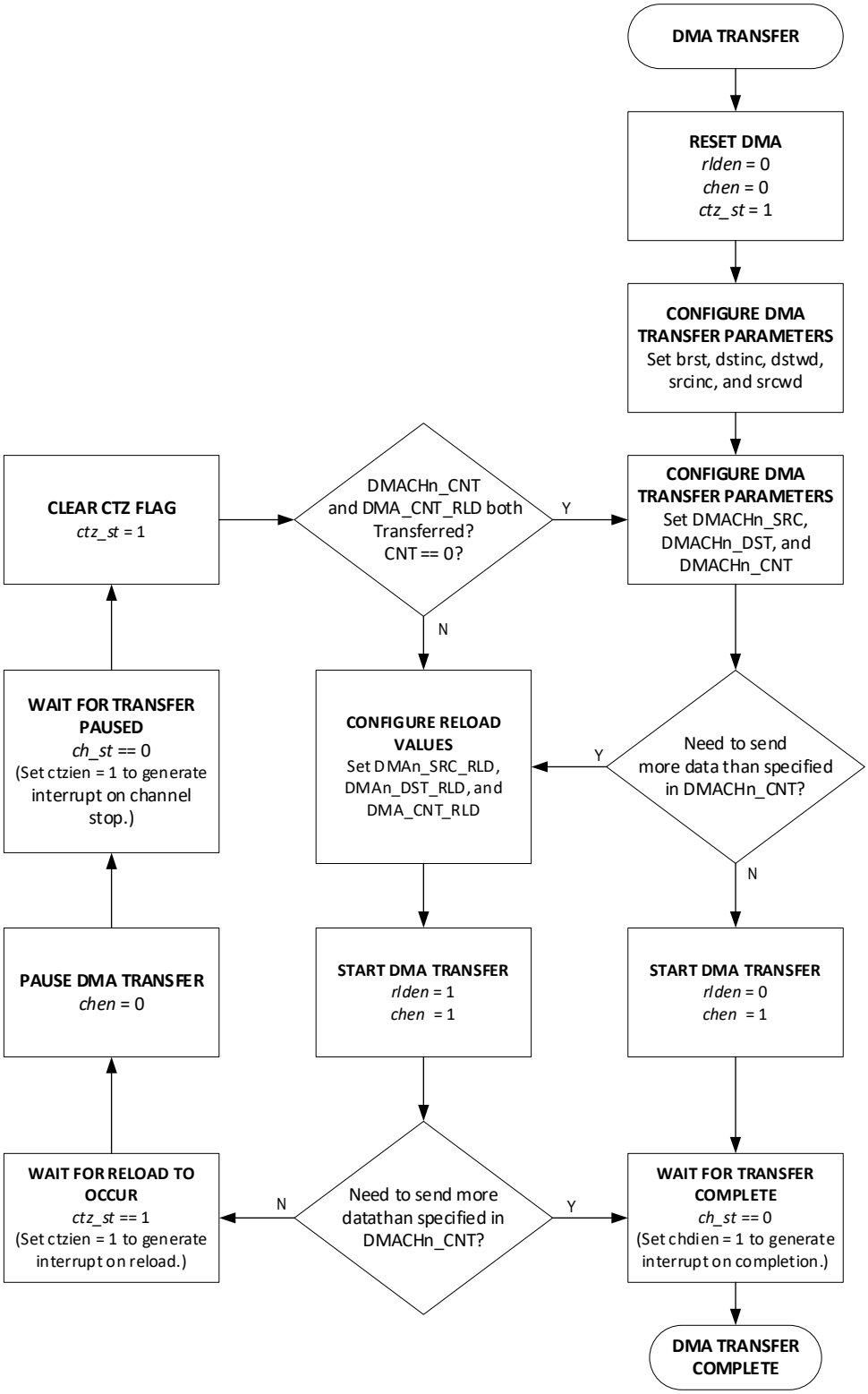
Chaining buffers reduces the DMA ISR response time and allows DMA to service requests without intermediate processing from the CPU. *Figure 9-1* shows the procedure for generating a DMA transfer using one or more chain buffers.

Configure the following reload registers to configure a channel for chaining:

- *DMA_CHn_CFG*
- *DMA_CHn_SRC*
- *DMA_CHn_DST*
- *DMA_CHn_CNT*
- *DMAn_SRC_RLD*
- *DMAn_DST_RLD*
- *DMAn_CNT_RLD*

Writing to any register while a channel is disabled is supported, but there are certain restrictions when a channel is enabled. The *DMA_CHn_ST*.ch_st bit indicates whether the channel is enabled or not. Because an active channel might be in the middle of an AHB read/write burst, do not write to the *DMA_CHn_SRC*, *DMA_CHn_DST*, or *DMA_CHn_CNT* registers while a channel is active (*DMA_CHn_ST*.ch_st = 1). To disable any DMA channel, clear the *DMAn_CN*.*ch<n>_ien* bit. Then, poll the *DMA_CHn_ST*.ch_st bit to verify that the channel is disabled.

Figure 9-1: DMA Block-Chaining Flowchart



9.6 DMA Interrupts

Enable interrupts for each channel by setting *DMAn_CN.ch<n>_ien*. When an interrupt for a channel is pending, the corresponding *lIDMAn_INTR.ch<n>_ipend* = 1. Set the corresponding enable bit to cause an interrupt when the flag is set.

A channel interrupt (*DMA_CHn_ST.ipend* = 1) is caused by:

- *DMA_CHn_CFG.ctzien* = 1
  - If enabled all CTZ occurrences set the *DMA_CHn_ST.ipend* bit.
- *DMA_CHn_CFG.chdien* = 1
  - If enabled, any clearing of the *DMA_CHn_ST.ch_st* bit sets the *DMA_CHn_ST.ipend* bit. Examine the *DMA_CHn_ST* register to determine which reasons caused the disable. The *DMA_CHn_CFG.chdien* bit also enables the *DMA_CHn_ST.to_st* bit. The *DMA_CHn_ST.to_st* bit does not clear the *DMA_CHn_ST.ch_st* bit.

To clear the channel interrupt, write 1 to the cause of the interrupt (the *DMA_CHn_ST.ctz_st*, *DMA_CHn_ST.rld_st*, *DMA_CHn_ST.bus_err*, or *DMA_CHn_ST.to_st* bits).

When running in normal mode without buffer chaining (*DMA_CHn_CFG.rlden* = 0), set the *DMA_CHn_CFG.chdien* bit only. An interrupt is generated upon DMA completion or an error condition (bus error or timeout error).

When running in buffer chaining mode (*DMA_CHn_CFG.rlden* = 1), set both the *DMA_CHn_CFG.chdien* and *DMA_CHn_CFG.ctzien* bits. The CTZ interrupts occur on completion of each DMA (count reaches zero and reload occurs). The setting of *DMA_CHn_CFG.chdien* ensures that an error condition generates an interrupt. If *DMA_CHn_CFG.ctzien* = 0, then the only interrupt occurs when the DMA completes and *DMA_CHn_CFG.rlden* = 0 (final DMA).

9.7 Channel Timeout Detect

Each channel can optionally generate an interrupt when its associated peripheral does not request a transfer in a user-configurable period of time. When the timeout start conditions are met, an internal 10-bit counter begins incrementing at a frequency determined by the AHB clock, *DMA_CHn_CFG.to_prescale*, and *DMA_CHn_CFG.to_period* shown in *Table 9-5*. A channel timeout event is generated if the timer is not reset by one of the events listed below before the timeout period expires.

Table 9-5: DMA Channel Timeout Configuration

DMA_CHn_CFG.prescale	Timeout Period (µs)
0x0	*Channel timeout disabled.*
0x1	$\frac{2^8 * [Value\ from\ \mathrm{DMA\_CFG}.to\_period]}{f_{\mathrm{HCLK}}}$
0x2	$\frac{2^{16} * [Value\ from\ \mathrm{DMA\_CFG}.to\_period]}{f_{\mathrm{HCLK}}}$
0x3	$\frac{2^{24} * [Value\ from\ \mathrm{DMA\_CFG}.to\_period]}{f_{\mathrm{HCLK}}}$

The start of the timeout period is controlled by *DMA_CHn_CFG.reqwait*:

- If *DMA_CHn_CFG.reqwait* = 0, the timer begins counting immediately after *DMA_CHn_CFG.to_sel* is configured to a value other than 0x0.
- If *DMA_CHn_CFG.reqwait* = 1, the timer begins counting when the first DMA request is received from the peripheral.

The timer is reset whenever:

- The DMA request line programmed for the channel is activated.
- The channel is disabled for any reason (*DMA_CHn_ST.ch_st* = 0).

If the timeout timer period expires, hardware will set *DMA_CHn_ST*.to_st = 1 to indicate a channel timeout event has occurred. A channel timeout will not disable the DMA channel.

9.8 Memory-to-Memory DMA

Memory-to-memory transfers are processed as if the request is always active. This means that the DMA channel generates an almost constant request for the bus until its transfer is complete. For this reason, assign a lower priority to channels executing memory-to-memory transfers to prevent starvation of other DMA channels.

9.9 DMA Registers

See *Table 3-3:* APB Peripheral Base Address Map for this peripheral/module's base address. If multiple instances are provided, each will have a unique base address. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific reset, if applicable.

Table 9-6: DMA Register Summary

Offset	Register	Description
[0x0000]	*DMAn_CN*	DMA Control register
[0x0004]	*llDMAn_INTR*	DMA Interrupt Status register

9.10 DMA Register Details

Table 9-7: DMAn Control Register

DMAn Control				**DMAn_CN**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	*ch<n>_ien*	R/W	0	**DMAn Channel Interrupt Enable** Each bit in this field enables the corresponding channel interrupt m in *llDMAn_INTR*. Register bits associated with unimplemented channels should not be changed from their default reset value. 0: Disabled. 1: Enabled.	

Table 9-8: DMAn Interrupt Register

DMAn Interrupt				**llDMAn_INTR**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	ch<n>_ipend	RO	0	**DMAHn Channel Interrupt Flag** Each bit in this field represents an interrupt for the corresponding channel interrupt m. To clear an interrupt, clear the corresponding active interrupt bit in the *DMA_CHn_ST* register. An interrupt bit in this field is set only if the corresponding interrupt enable field is set in the *DMAn_CN* register. Register bits associated with unimplemented channels should be ignored. 0: No interrupt 1: Interrupt pending	

9.11 DMA Channel Register Summary

Table 9-9: Standard DMA Channel 0 to Channel 7 Register Summary

Offset	DMA Channel	Description
[0x0100]	DMA_CH0	DMAm Channel 0
[0x0120]	DMA_CH1	DMAm Channel 1
[0x0140]	DMA_CH2	DMAm Channel 2

Offset	DMA Channel	Description
[0x0160]	DMA_CH3	DMAm Channel 3
[0x0180]	DMA_CH4	DMAm Channel 4
[0x01A0]	DMA_CH5	DMAm Channel 5
[0x01C0]	DMA_CH6	DMAm Channel 6
[0x01E0]	DMA_CH7	DMAm Channel 7

9.12 DMA Channel Registers

See *Table 3-3:* APB Peripheral Base Address Map for the DMA Peripheral Base Address

Table 9-10: DMA Channel Registers Summary

Offset	Register	Description
[0x0100]	*DMA_CHn_CFG*	DMA_CHn Channel Configuration Register
[0x0104]	*DMA_CHn_ST*	DMA_CHn Channel Status Register
[0x0108]	*DMA_CHn_SRC*	DMA_CHn Channel Source Register
[0x010C]	*DMA_CHn_DST*	DMA_CHn Channel Destination Register
[0x0110]	*DMA_CHn_CNT*	DMA_CHn Channel Count Register
[0x0114]	*DMAn_SRC_RLD*	DMA_CHn Channel Source Reload Register
[0x0118]	*DMAn_DST_RLD*	DMA_CHn Channel Destination Reload Register
[0x011C]	*DMAn_CNT_RLD*	DMA_CHn Channel Count Reload Register

9.13 DMA Channel Register Details

Table 9-11: DMA_CHn Configuration Register

DMA Channel n Configuration				**DMA_CHn_CFG**	**[0x0100]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31	*ctzien*	R/W	0	**CTZ Interrupt Enable** 0: Disabled 1: Enabled. *lIDMAn_INTR.ch<n>_ipend* is set to 1 whenever a CTZ event occurs.	
30	*chdien*	R/W	0	**Channel Disable Interrupt Enable** 0: Disabled 1: Enabled. *lIDMAn_INTR.ch<n>_ipend* bit is set to 1 whenever *DMA_CHn_ST.ch_st* changes from 1 to 0.	
29	-	RO	0	**Reserved**	
28:24	*brst*	R/W	0	**Burst Size** The number of bytes transferred into and out of the DMA FIFO in a single burst. 0b00000: 1 byte 0b00001: 2 bytes 0b00010: 3 bytes ... 0b11111: 32 bytes	
23	-	RO	0	**Reserved**	

DMA Channel n Configuration				DMA_CHn_CFG	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**	
22	*distinc*	R/W	0	**Destination Increment Enable** This bit enables the automatic increment of the *DMA_CHn_DST* register upon every AHB transaction. This bit is ignored for a DMA transmit to peripherals. 0: Disabled 1: Enabled	
21:20	*dstwd*	R/W	0	**Destination Width** Indicates the width of each AHB transaction to the destination peripheral or memory (the actual width might be less than this if there are insufficient bytes in the DMA FIFO for the full width). 0x0: One byte 0x1: Two bytes 0x2: Four bytes 0x3: Reserved	
19	-	RO	0	**Reserved**	
18	*srcinc*	R/W	0	**Source Increment on AHB Transaction Enable** This bit enables the automatic increment of the *DMA_CHn_SRC* register upon every AHB transaction. This bit is ignored for a DMA receive from peripherals. 0: Disabled 1: Enabled	
17:16	*srcwd*	R/W	0	**Source Width** Indicates the width of each AHB transaction from the source peripheral or memory. The actual width might be less than this if the *DMA_CHn_CNT* register indicates a smaller value. 0x0: One byte 0x1: Two bytes 0x2: Four bytes 0x3: Reserved	
15:14	*pssel*	R/W	0	**Timeout Timer Clock Pre-Scale Select** Selects the Pre-Scale divider for the timer clock input. 0x0: Timer disabled. 0x1: f_{HCLK} / 2^8 0x2: f_{HCLK} / 2^{16} 0x3: f_{HCLK} / 2^{24}	
13:11	*tosel*	R/W	0	**Timeout Period Select** Selects the number of pre-scaled clocks seen by the channel timer before a timeout condition is generated. The value is approximate because of synchronization delays between timers 0: 3-4 1: 7-8 2: 15-16 3: 31-32 4: 63-64 5: 127-128 6: 255-256 7: 511-512	
10	*reqwait*	R/W	0	**Request DMA Timeout Timer Wait Enable** 0: Start timer immediately when enabled. 1: Delay timer start until after the first DMA transaction occurs.	

DMA Channel n Configuration				DMA_CHn_CFG	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**	
9:4	*reqsel*	R/W	0	**Request Select** Selects the source and destination for the transfer as shown in *Table 9-2*.	
3:2	*pri*	R/W	0	**Channel Priority** Sets the priority of the channel relative to other channels of DMAm. Channels of the same priority are serviced in a round-robin fashion. 0x0: Highest priority 0x1: … 0x2: … 0x3: Lowest priority	
1	*rlden*	R/W	0	**Reload Enable** Setting this bit to 1 allows reloading the *DMA_CHn_SRC*, *DMA_CHn_DST*, and *DMA_CHn_CNT* registers with their corresponding reload registers upon CTZ. Note: This bit is also writeable in the DMAn_CNT_RLD register.	
0	*chen*	R/W	0	**Channel Enable** This bit is automatically cleared when *DMA_CHn_ST.ch_st* changes from 1 to 0. 0: Disabled 1: Enabled	

Table 9-12: DMA Status Register

DMA Channel n Status				DMA_CHn_ST	[0x0104]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:7	-	RO	0	**Reserved for Future Use** Do not modify this field from its reset default value.	
6	*to_st*	R/W1C	0	**Timeout Status** Timeout status field. Write 1 to clear. 0: No time out. 1: A channel time out has occurred	
5	-	RO	0	**Reserved**	
4	*bus_err*	R/W1C	0	**Bus Error** If this bit reads 1, an AHB abort occurred and the channel was disabled by hardware. Write 1 to clear. 0: No error found 1: An AHB bus error occurred	
3	*rld_st*	R/W1C	0	**Reload Status** Reload status field. Write 1 to clear. 0: Reload has not occurred. 1: Reload occurred.	
2	*ctz_st*	R/W1C	0	**CTZ Status** Write 1 to clear. 0: CTZ has not occurred. 1: CTZ has occurred.	
1	*ipend*	RO	0	**Channel Interrupt Pending** 0: No interrupt 1: Interrupt pending	

DMA Channel n Status				DMA_CHn_ST	[0x0104]
Bits	**Field**	**Access**	**Reset**	**Description**	
0	*ch_st*	RO	0	**Channel Status** This bit indicates when it is safe to change the configuration, address, and count registers for the channel. Whenever this bit is cleared by hardware, the *DMA_CHn_CFG.chen* bit is also cleared. 0: Disabled. 1: Enabled.	

Table 9-13: DMA_CHn Source Register

DMA Channel n Source				DMA_CHn_SRC	[0x0108]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	addr	R/W	0	**Source Device Address** For peripheral transfers, the actual address field is either ignored or forced to zero because peripherals only have one location to read/write data based on the request select chosen. If *DMA_CHn_CFG.srcinc* = 1, then this register is incremented on each AHB transfer cycle by one, two, or four bytes depending on the data width. If *DMA_CHn_CFG.srcinc* = 0, this register remains constant. If a CTZ condition occurs while *DMA_CHn_CFG.rlden* = 1, then this register is reloaded with the contents of the *DMAn_SRC_RLD* register.	

Table 9-14: DMA Channel n Destination Register

DMA Channel n Destination				DMA_CHn_DST	[0x010C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	addr	R/W	0	**Destination Device Address** For peripheral transfers, the actual address field is either ignored or forced to zero because peripherals only have one location to read/write data based on the request select chosen. If *DMA_CHn_CFG.dstinc* = 1, then this register is incremented on every AHB transfer cycle by one, two, or four bytes depending on the data width. If a CTZ condition occurs while *DMA_CHn_CFG.rlden* = 1, then this register is reloaded with the contents of the *DMAn_DST_RLD* register.	

Table 9-15: DMA Channel n Count Register

DMA Channel n Count				DMA_CHn_CNT	[0x0110]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:24	-	RO	0	**Reserved**	
23:0	cnt	R/W	0	**DMA Counter** Load this register with the number of bytes to transfer. This field decreases on every AHB access to the DMA FIFO. The decrement is one, two, or four bytes depending on the data width. When the counter reaches 0, a CTZ condition is triggered. If a CTZ condition occurs while *DMA_CHn_CFG.rlden* = 1, then this register is reloaded with the contents of the *DMAn_CNT_RLD* register.	

Table 9-16: DMA Channel n Source Reload Register

DMA Source Reload				DMAn_SRC_RLD	[0x0114]
Bits	**Field**	**Access**	**Reset**	**Description**	
31	-	RO	0	**Reserved**	
30:0	src_rld	R/W	0	**Source Address Reload Value** If *DMA_CHn_CFG.rlden* = 1, then the value of this register is loaded into *DMA_CHn_SRC* upon a CTZ condition.	

Table 9-17: DMA Channel n Destination Reload Register

DMA Destination Reload Register				DMAn_DST_RLD	[0x0118]
Bits	**Field**	**Access**	**Reset**	**Description**	
31	-	RO	0	**Reserved**	
30:0	dst_rld	R/W	0	**Destination Address Reload Value** If *DMA_CHn_CFG.rlden* = 1, then the value of this register is loaded into *DMA_CHn_DST* upon a CTZ condition.	

Table 9-18: DMA Channel n Count Reload Register

DMA Count Reload				DMAn_CNT_RLD	[0x011C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31	rlden	R/W	0	**Reload Enable.** Enables automatic loading of the *DMA_CHn_SRC*, *DMA_CHn_DST*, and *DMA_CHn_CNT* registers when a CTZ event occurs. Set this bit after the address reload registers are programmed. *Note: This bit is automatically cleared to 0 when reload occurs.* *Note: This bit is also seen in the DMA_CHn_CFG register.* 0: Reload disabled 1: Reload enabled	
30:24	-	RO	0	**Reserved**	
23:0	cnt_rld	R/W	0	**Count Reload Value.** If *DMAn_CNT_RLD.rlden* = 1, then the value of this register is loaded into *DMA_CHn_CNT* upon a CTZ condition.	

10. Analog-to-Digital Converter (ADC) and Comparators

The analog-to-digital Converter (ADC) is a 10-bit sigma-delta ADC with a single-ended input multiplexer and an integrated reference generator. The multiplexer selects an input channel from either of the 8 external analog input signals or the internal power supply inputs. The external analog input signals are defined as alternate functions on GPIO as shown in *Table 6-2*. The 10-bit ADC conversions are stored as a 16-bit value selectable as most-significant bit (MSB) or least-significant bit (LSB) aligned. The 8 external analog inputs can be configured as 4 two-input comparators with interrupt capabilities.

10.1 Features

- 8MHz maximum ADC clock rate
- Two reference sources, an internal 1.22V bandgap or the V_{DDA} analog supply
- 8 External analog inputs that can be configured as 4 two-input comparators
- 10 Internal power supply monitor inputs
- Fixed 10-bit word conversion time of 1024 ADC clock cycles
- Programmable out-of-range (limit) detection
- Interrupt generation for limit detection, conversion start, conversion complete, and internal reference powered on
- Serial ADC data measurements
- ADC conversion 10 output either MSB or LSB aligned

10.2 Instances

Table 10-1 lists the locations of the External ADC inputs for each package.

Table 10-1: MAX32665/MAX32666 ADC Peripheral Pins

ALTERNATE FUNCTION	MAPPING	109 WLP	121 CTBGA
AIN0/AIN0P	AF1	P0.16	P0.16
AIN1/AIN0N	AF1	P0.17	P0.17
AIN2/AIN1P	AF1	P0.18	P0.18
AIN3/AIN1N	AF1	P0.19	P0.19
AIN4/AIN2P	AF1	P0.20	P0.20
AIN5/AIN2N	AF1	P0.21	P0.21
AIN6/AIN3P	AF1	P0.22	P0.22
AIN7/AIN3N	AF1	P0.23	P0.23

10.3 Architecture

The ADC is a first-order sigma-delta converter with a 10-bit output. The ADC operates at a maximum frequency of 8MHz with a fixed-sample rate as shown in *Equation 10-1*. Details of selecting the ADC clock frequency, f_{adcclk}, are covered in the *Clock Configuration* section.

Equation 10-1: ADC 10-bit Word Sample Rate

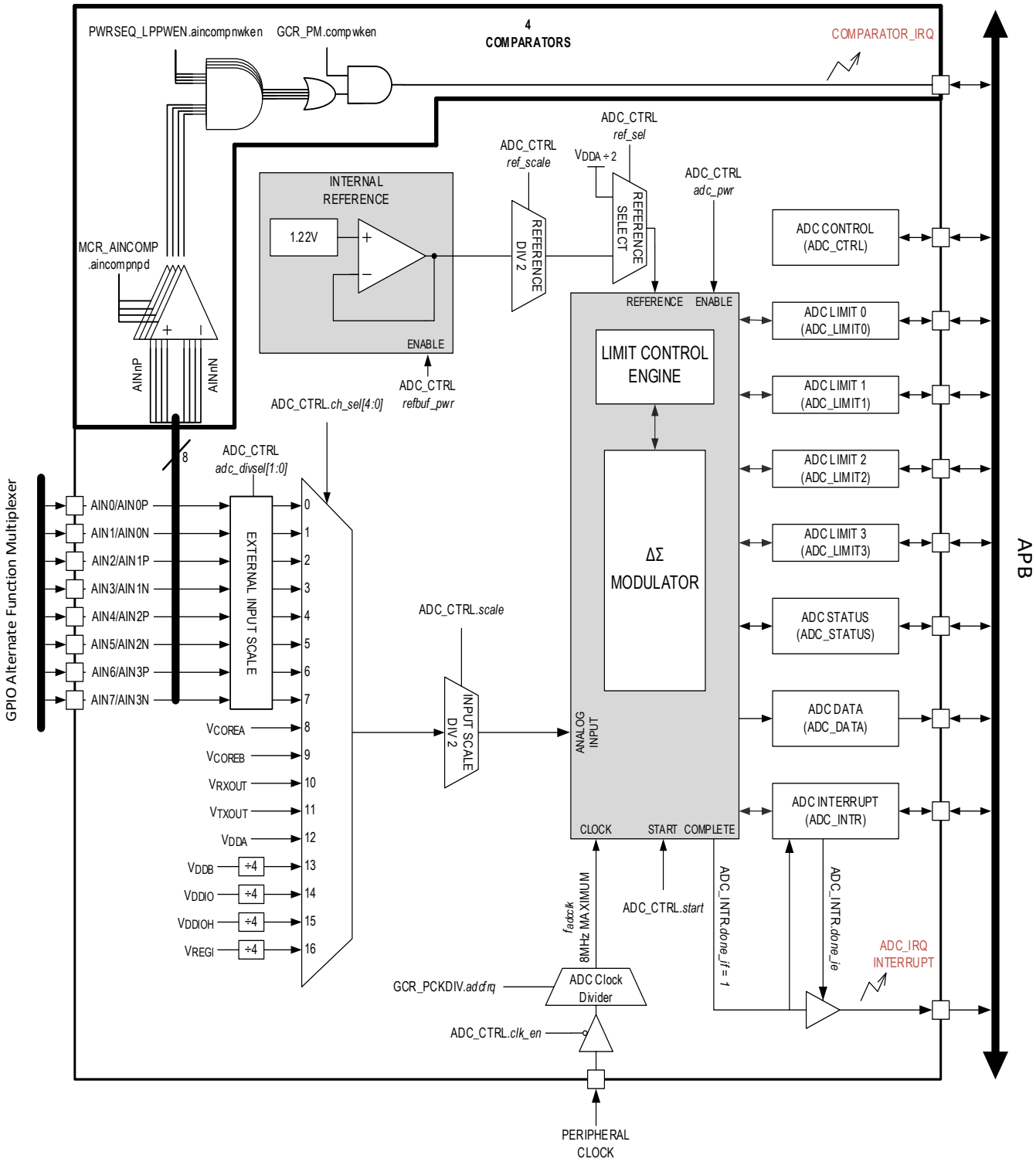
$$t_{adc\_sample} = 1024 \times \left(\frac{1}{f_{adcclk}}\right)$$

ADC offset is factory trimmed and automatically loaded into the ADC controller during system power-up.

The ADC uses a switched capacitor network to perform the conversion; this results in dynamic switching current and requires settling time for the external analog input signals (AIN0 – AIN7). This dynamic switching current sets the upper limit of the source impedance of the external analog input signals to approximately 10kΩ.

The ADC supports a gain of 2× to provide additional conversion resolution if the input signals are less than half the reference voltage.

Figure 10-1: Analog to Digital Converter Block Diagram



10.4 Clock Configuration

The ADC clock, *adcclk*, is controlled by the *GCR_PCKDIV.adcfrq* register field. Configure this field for the target ADC sample frequency. The maximum clock supported by the ADC is 8MHz. The divisor selection, *GCR_PCKDIV.adcfrq*, for the ADC depends on the peripheral clock. *Equation 10-2* shows the calculation for the ADC clock frequency, where:

$$f_{PCLK} = {}^{f_{SYSCLK}}/_{2}.$$

Equation 10-2: ADC Clock Frequency

$$f_{adcclk} = \frac{f_{PCLK}}{\text{GCR\_PCKDIV.adcfrq}}$$

The *GCR_PCKDIV.adcfrq* register field setting must result in a value for $f_{adcclk} \leq 8\text{MHz}$ as shown in *Table 10-2* with the System Clock set as the 96MHz high frequency oscillator.

Table 10-2: ADC Clock Frequency and ADC Conversion Time ($f_{SYSCLK} = 96MHz$, $f_{PCLK} = 48MHz$)

GCR_PCKDIV.adcfrq[3:0]	ADC Clock Frequency (Hz) f_{adcclk}	10-Bit Word Conversion Time (µs) t_{adc_sample}
0x– 0x7	Invalid	Invalid
0x8	6,000,000	171
0x9	5,333,333	192
0xA	4,800,000	214
0xB	4,363,636	235
0xC	4,000,000	256
0xD	3,692,308	278
0xE	3,428,571	299
0xF	3,200,000	320

10.5 Power-Up Sequence

Complete the following steps to configure the ADC:

1. Disable the ADC clock by setting *ADC_CTRL.clk_en* to 0.
2. Set the ADC clock (adcclk) using *GCR_PCKDIV.adcfrq*. See *Clock Configuration*
3. Enable the ADC clock by setting *ADC_CTRL.clk_en* to 1
4. Clear the ADC reference ready interrupt flag by writing a 1 to *ADC_INTR.ref_ready_if*.
5. Optionally enable the ADC reference ready interrupt (*ADC_INTR.ref_ready_ie* = 1), and enable the ADC interrupt vector (ADC IRQ).
6. Select one of the following ADC reference sources:
   a. Internal 1.22V bandgap reference (*ADC_CTRL.ref_sel* = 0).
   b. $V_{DDA} \div 2$ reference (*ADC_CTRL.ref_sel* = 1).
7. Complete the following steps to enable power:
   a. Set *ADC_CTRL.pwr* to 1 to turn on the ADC.
   b. Set *ADC_CTRL.refbuf_pwr* to 1 to turn on the internal reference buffer If using the internal bandgap reference.
   c. Wait until hardware sets the *ADC_INTR.ref_ready_if* bit to 1 indicating the charge pump is fully stabilized.
   d. Clear the ADC reference ready interrupt flag by writing 1 to *ADC_INTR.ref_ready_if*.

e. Optionally disable the ADC reference ready interrupt (*ADC_INTR.ref_ready_ie* = 0).

10.6 Conversion

After the power-up sequence is complete, the ADC is ready for data conversion. Complete the following steps to perform a data conversion:

1. Select the ADC input channel for the conversion by setting *ADC_CTRL.ch_sel* field. See *ADC Channel Select* for details.
2. Optionally set input and reference scaling. See *Reference Scaling and Input Scaling* for details on each input channel's scale requirements.
3. Set the data alignment for the conversion output data using the *ADC_CTRL.data_align* field, 0 for LSB alignment or 1 for MSB alignment. See *Table 10-4* for alignment details of the DATA register.
4. Clear the ADC done interrupt flag by writing 1 to the *ADC_INTR.done_if*.
5. Optionally enable the ADC done interrupt (*ADC_INTR.done_ie* = 1), and enable the ADC interrupt vector (ADC IRQ). See *Interrupts and Exceptions* for details.
6. Start the ADC conversion by setting *ADC_CTRL.start* to 1.
7. Poll the *ADC_INTR.done_if* flag until you read 1, or wait for the ADC interrupt to occur if enabled in step 5.
8. Read the data from the *ADC_DATA.data*, and clear the ADC done interrupt flag by writing 1 to *ADC_INTR.done_if*.

10.7 Reference Scaling and Input Scaling

For small signals, the ADC input, ADC reference or both can be scaled by 50%. This enables flexibility to achieve better resolution on the ADC conversion. Each input channel, supports the default of no scaling of the input (*ADC_CTRL.scale* = 0) and no scaling of the reference (*ADC_CTRL.ref_scale* = 0). The following sections describe the scale options for each of the ADC input channels.

10.7.1 AIN0 – AIN7 Scale Limitations

The external inputs, AIN0 through AIN7, support scaling of the input by 50%, the reference by 50%, or both by 50%. Also, the scaling can further be modified by additional factors of 2, 3, or 4 as defined by *ADC_CTRL.adc_divsel*. The scale settings for the given input signal and reference must satisfy the following equation to be valid:

Equation 10-3: Input and Reference Scale Requirements Equation

$$\frac{\mathrm{AINn}}{2^{\mathrm{scale}}} < \frac{\mathrm{V_{REF}}}{2^{\mathrm{ref\_scale}}}$$

10.7.2 Scale Limitations for All Other Input Channels

For the remaining internal input channels, the scale settings must either both be disabled, or both be enabled as shown in *Table 10-3*.

Table 10-3: Input and Reference Scale Support by ADC Input Channel

ADC Channel	ADC Input Signal	ADC_CTRL scale	ADC_CTRL ref_scale
8	V_{COREA}	0	0
		1	1
9	V_{COREB}	0	0
		1	1
10	V_{RXOUT}	0	0
		1	1
11	V_{TXOUT}	0	0
		1	1
12	V_{DDA}	0	0
		1	1
13	$V_{DDB}/4$	0	0
		1	1
14	$V_{DDIO}/4$	0	0
		1	1
15	$V_{DDIOH}/4$	0	0
		1	1
16	$V_{REGI}/4$	0	0
		1	1

10.7.3 Data Conversion Output Alignment

The ADC outputs a total of 10-bits per conversion and stores the data in the DATA register LSB justified by default. *Table 10-4* shows the ADC data alignment based on the value of the *ADC_CTRL.data_align* bit.

Table 10-4: ADC Data Register Alignment Options

***ADC_CTRL.data_align* = 0**																
	MSB															**LSB**
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADC_DATA	0	0	0	0	0	0	data									

***ADC_CTRL.data_align* = 1**																
	MSB															**LSB**
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADC_DATA	data										0	0	0	0	0	0

10.7.4 Data Conversion Value Equations

Use the following equations to calculate the ADC data value for a conversion for the selected channel. If using the internal reference, V_{REF} = 1.22V; otherwise V_{REF} = VDDA.

Equation 10-4: ADC Data Calculation for Input Signal ADC_CTRL.ch_sel = 0x00 thru 0x07 (AIN0 – AIN7)

$$\text{ADC\_DATA} = \text{round}\left\{\left(\frac{\left(\frac{\text{Input Signal}}{2^{\text{scale}} * (\text{adc\_divsel} + 1)}\right)}{\left(\frac{V_{REF}}{2^{\text{ref\_scale}}}\right)}\right) \times (2^{10} - 1)\right\}$$

Note: Must satisfy Equation 10-3.

Equation 10-5: ADC Data Equation for Input Signal ADC_CTRL.ch_sel = 0x08 thru 0x0C (V_{COREA}, V_{COREB}, V_{RXOUT}, V_{TXOUT}, V_{DDA})

$$\text{ADC\_DATA} = \text{round}\left\{\left(\frac{\left(\frac{\text{Input Signal}}{2^{\text{scale}}}\right)}{\left(\frac{V_{REF}}{2^{\text{ref\_scale}}}\right)}\right) \times (2^{10} - 1)\right\}$$

Note: See Table 10-3 for limitations.

Equation 10-6: ADC Data Calculation Input Signal ADC_CTRL.ch_sel = 0x0D thru 0x10 (V_{DDB}, V_{DDIO}, V_{DDIOH}, V_{REGI})

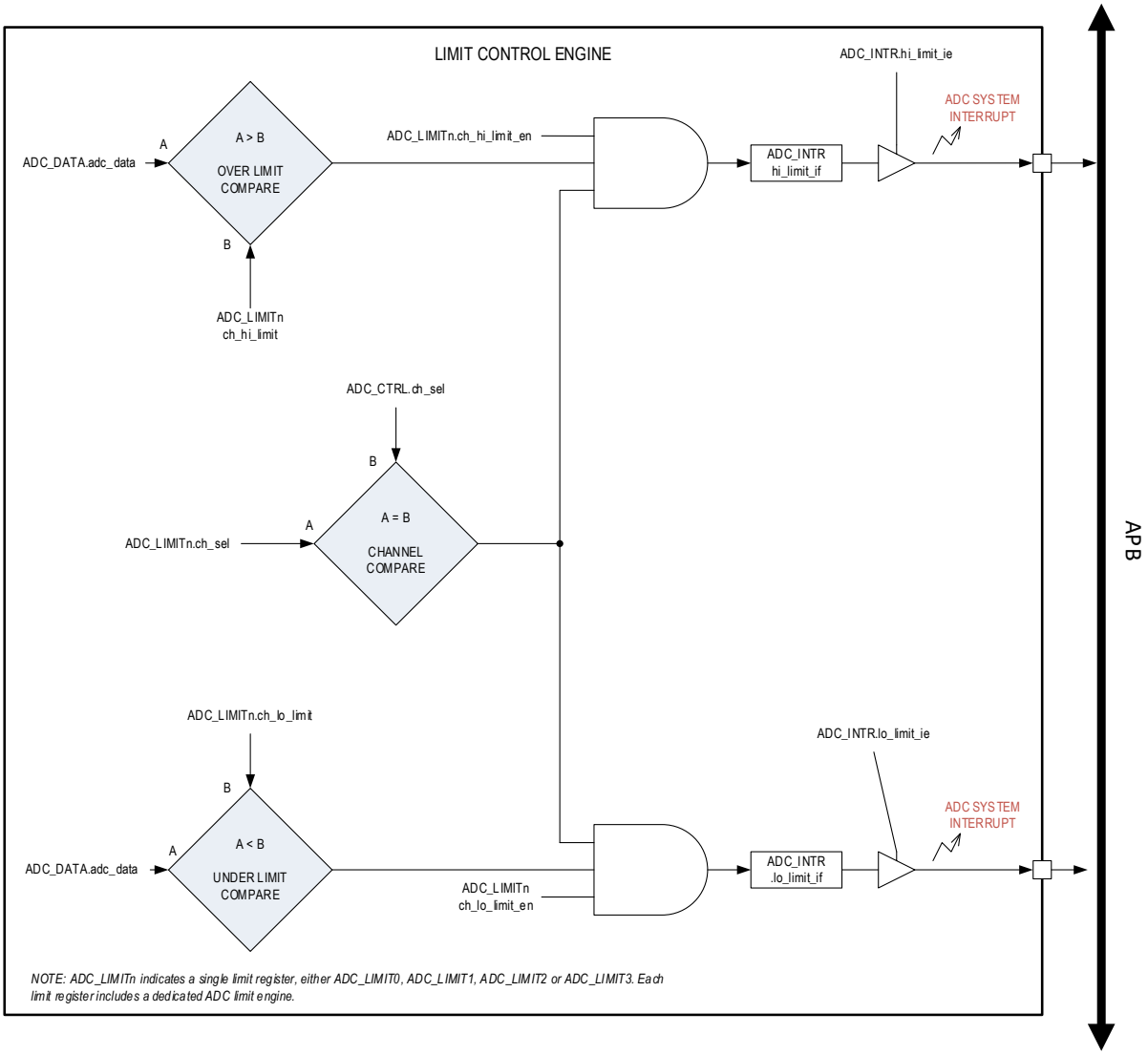
$$ADC\_DATA = round\left\{\left(\frac{\left(\frac{\frac{Input\ Signal}{4}}{2^{scale}}\right)}{\left(\frac{V_{REF}}{2^{ref\_scale}}\right)}\right) \times (2^{10} - 1)\right\}$$

Note: See Table 10-3 for limitations.

10.7.5 Data Limits and Out of Range Interrupts

Channel limits are implemented to minimize power consumption for power supply monitoring. The ADC includes four limit registers, *ADC_LIMIT0* to *ADC_LIMIT3*, that you can use to set a high limit, low limit, and the ADC channel number to apply the limits against. A block diagram of the limit engine for each of the four limit registers is shown in *Figure 10-2*.

Figure 10-2: ADC Limit Engine



When a measurement is taken on the ADC, the limit engine determines if the channel measured matches one of the channels selected by the limit registers. If it does and the data converted is above or below the high or low limit, an interrupt flag is set resulting in an ADC interrupt if the interrupt is enabled.

Complete the following steps to enable a high and low limit for an ADC input channel using the *ADC_LIMIT0* register. Perform these steps after the ADC is configured for measurement, and the configuration is identical for all four limit registers except for the limit register name:

1. Verify the ADC is not actively taking a measurement by checking *ADC_STATUS.active* until it reads 0.
2. Set *ADC_LIMIT0.ch_sel* field to the selected channel for the high and low limit.
3. Set the high limit, *ADC_LIMIT0.ch_hi_limit*, to the selected 10-bit trip point. When enabled, an ADC measurement greater than this field on the channel selected (*ADC_LIMIT0.ch_sel*) generates an ADC interrupt.
4. Set the low limit, *ADC_LIMIT0.ch_lo_limit*, to the selected 10-bit low trip point. When enabled, an ADC measurement lower than this field on the channel selected (*ADC_LIMIT0.ch_sel*) generates an ADC interrupt.

5. Enable the high limit, the low limit, or both interrupt signals by writing a 1 to *ADC_LIMIT0.ch_hi_limit_en*, *ADC_LIMIT0.ch_lo_limit_en*, or both. Note: Each limit register is independently enabled for high- and low-limit interrupts.
6. Clear the ADC interrupt high and low interrupt flags by writing 1 to *ADC_INTR.hi_limit_if* and *ADC_INTR.lo_limit_if*.
7. Enable the high, low, or both interrupts for the ADC by setting *ADC_INTR.hi_limit_if* to 1, *ADC_INTR.lo_limit_ie* to 1, or both.
8. If an ADC conversion occurs that is above or below the enabled limits, an ADC_IRQn is generated with the *ADC_INTR.hi_limit_if*, *ADC_INTR.lo_limit_if*, or both set to 1. The *ADC_CTRL.ch_sel* value indicates the channel that caused the interrupt, and the value of the ADC conversion that is out of bounds is in the *ADC_DATA.data* field.

10.7.6 Power-Down Sequence

Complete the following steps to power-down the ADC:

1. Set *ADC_CTRL.pwr* to 0, disabling the ADC converter power.
2. Set *ADC_CTRL.refbuf_pwr* to 0, disabling the internal reference buffer power.
3. Set *ADC_CTRL.clk_en* to 0, disabling the ADC internal clock.

10.8 Comparator Operation

Each comparator is individually enabled using *MCR_AINCOMP*.aincompnpd. When the inputs to any comparator cross their bias potential, the corresponding flag bit in the *PWRSEQ_LPPWST* register will be set. Should the user desire, an interrupt can be generated by setting the corresponding bit in the *PWRSEQ_LPPWEN* register. The interrupts must be globally enabled by setting the *GCR_PM*.compwken bit.

10.9 Registers

See *Table 3-3* for the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 10-5: ADC Registers Summary

Offset	Register	Description
[0x0000]	*ADC_CTRL*	ADC Control Register
[0x0004]	*ADC_STATUS*	ADC Status Register
[0x0008]	*ADC_DATA*	ADC Output Data Register
[0x000C]	*ADC_INTR*	ADC Interrupt Control Register
[0x0010]	*ADC_LIMIT0*	ADC Limit 0 Register
[0x0014]	*ADC_LIMIT1*	ADC Limit 1 Register
[0x0018]	*ADC_LIMIT2*	ADC Limit 2 Register
[0x001C]	*ADC_LIMIT3*	ADC Limit 3 Register

10.10 Register Details

Table 10-6: ADC Control Register

ADC Control			ADC_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0x050	**Reserved for Future Use** Do not modify this field.

ADC Control			ADC_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
20	data_align	R/W	0	**ADC Data Alignment** Selects the alignment of the 16-bit data conversion stored in the DATA register. 0: Data is LSB justified in 16-bit DATA register. DATA[15:10] = 0. 1: Data is MSB justified in 16-bit DATA register. DATA[5:0] = 0.
19	-	RO	0	**Reserved for Future Use** Do not modify this field.
18:17	adc_divsel	R/W	0	**External Input Scale** Scales the external inputs AIN0-AIN7. All eight of external inputs are scaled by the same value 0x0: No scaling. 0x1: Divide by 2 0x2: Divide by 3 0x3: Divide by 4
16:12	ch_sel	R/W	0	**ADC Channel Select** Selects the active channel for the next ADC conversion.

ch_sel	ADC Input Channel	Input
0x00	0	AIN0
0x01	1	AIN1
0x02	2	AIN2
0x03	3	AIN3
0x04	4	AIN4
0x05	5	AIN5
0x06	6	AIN6
0x07	7	AIN7
0x08	8	VCOREA
0x09	9	VCOREB
0x0A	10	VRXOUT
0x0B	11	VTXOUT
0x0C	12	VDDA
0x0D	13	VDDB / 4
0x0E	14	VDDIO / 4
0x0F	15	VDDIOH / 4
0x10	16	VREGI / 4
0x11 – 0x1F	Reserved for future use	Reserved for future use

11	clk_en	R/W	0	**ADC Clock Enable** 0: Disabled 1: Enabled
10	-	RO	0	**Reserved for Future Use** Do not modify this field.
9	scale	R/W	0	**ADC Input Scale** Scales ADC input by 50 percent. 0: ADC input is not scaled. 1: ADC input is scaled by ½. *Note: See Reference Scaling and Input Scaling for valid settings for each ADC input.*
8	ref_scale	R/W	0	**Reference Scale** Scales the internal bandgap reference by 50 percent. 0: Internal bandgap reference is not scaled. 1: Internal bandgap reference is scaled by ½. *Note: See Reference Scaling and Input Scaling for valid settings for each ADC input*

ADC Control			ADC_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
7:5	-	RO	0	**Reserved for Future Use** Do not modify this field.
4	ref_sel	R/W	0	**ADC Reference Select** 0: Internal bandgap reference is used for the ADC reference. 1: $V_{DDA} \div 2$ is used for the ADC reference.
3	refbuf_pwr	R/W	0	**Reference Buffer Power Enable** 0: Disabled 1: Enabled
2	-	RO	0	**Reserved for Future Use** Do not modify this field.
1	pwr	R/W	0	**ADC Power Enable** 0: Disabled 1: Enabled
0	start	R/W	0	**Start ADC Conversion** Write this bit to 1 to start an ADC conversion. When the conversion is complete, the hardware automatically sets this bit to 0 indicating the conversion is complete. 0: ADC inactive or data conversion complete. 1: Start ADC conversion and remains set until complete.

Table 10-7: ADC Status Register

ADC Status			ADC_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved for Future Use** Do not modify this field.
3	overflow	RO	0	**ADC Overflow Flag** 0: No overflow on last conversion 1: Overflow on last conversion
2	Afe_pwr_up_active	RO	0	**ADC Power-Up State** This field is set to 1 when the ADC charge pump is powering up. 0: AFE is not in power-up delay. 1: AFE is currently in the power-up delay state.
1	-	RO	0	**Reserved for Future Use** Do not modify this field.
0	active	RO	0	**ADC Conversion in Progress** 0: ADC is idle 1: ADC conversion is in progress

Table 10-8: ADC Data Register

ADC Data			ADC_DATA	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	data	RO	0	**ADC Data** This field contains the ADC conversion output data. See the *Data Conversion Output Alignment* for details.

Table 10-9: ADC Interrupt Control Register

ADC Interrupt Control			ADC_INTR	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:23	-	RO	0	**Reserved for Future Use** Do not modify this field.

ADC Interrupt Control			ADC_INTR	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
22	pending	RO	0	**ADC Interrupt Pending** 0: No ADC interrupt pending. 1: At least one ADC interrupt is pending, and the corresponding interrupt enable bit is set.
21	-	RO	0	**Reserved for Future Use** Do not modify this field.
20	overflow_if	R/W1C	0	**ADC Overflow Interrupt Flag** 1: The last conversion resulted in an overflow
19	lo_limit_if	R/W1C	0	**ADC Low Limit Interrupt Flag** 1: The last conversion resulted in a low-limit condition for one of the limit registers.
18	hi_limit_if	R/W1C	0	**ADC High Limit Interrupt Flag** 1: The last conversion resulted in a high-limit condition for one of the limit registers.
17	ref_ready_if	R/W1C	0	**ADC Reference Ready Interrupt Flag** 0: Not Ready 1: Ready.
16	done_if	R/W1C	0	**ADC Conversion Complete Interrupt Flag** Set by the ADC hardware when an ADC conversion is complete. 1: ADC conversion complete
15:5	-	RO	0	**Reserved for Future Use** Do not modify this field.
4	overflow_ie	R/W	0	**ADC Overflow Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR*.overflow_if.
3	lo_limit_ie	R/W	0	**ADC Low Limit Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets the *ADC_INTR*.lo_limit_if.
2	hi_limit_ie	R/W	0	**ADC High Limit Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR*.lo_limit_if.
1	ref_ready_ie	R/W	0	**ADC Reference Ready Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR*.ref_ready_if.
0	done_ie	R/W	0	**ADC Conversion Complete** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR*.done_if.

Table 10-10: ADC Limit 0 to 3 Registers

ADC Limit 0			ADC_LIMIT0	[0x0010]
ADC Limit 1			**ADC_LIMIT1**	**[0x0014]**
ADC Limit 2			**ADC_LIMIT2**	**[0x0018]**
ADC Limit 3			**ADC_LIMIT3**	**[0x001C]**
Bits	**Field**	**Access**	**Reset**	**Description**
31	-	RO	0	**Reserved for Future Use** Do not modify this field.

ADC Limit 0	ADC_LIMIT0	[0x0010]
ADC Limit 1	ADC_LIMIT1	[0x0014]
ADC Limit 2	ADC_LIMIT2	[0x0018]
ADC Limit 3	ADC_LIMIT3	[0x001C]

Bits	Field	Access	Reset	Description
30	ch_hi_limit_en	R/W	0	**High Limit Monitoring Enable** If set, then an ADC conversion that results in a value greater than the ch_high_limit field generates an ADC interrupt if the ADC high-limit interrupt is enabled. (*ADC_INTR*.hi_limit_ie = 1). 1: The high-limit comparison for the ch_sel channel is active. 0: The high-limit comparison is not enabled.
29	ch_lo_limit_en	R/W	0	**Low Limit Monitoring Enable** If set, then an ADC conversion that results in a value less than the ch_high_limit field generates an ADC interrupt if the ADC low-limit interrupt is enabled (*ADC_INTR*.lo_limit_ie = 1). 1: The low-limit comparison for the ch_sel channel is active. 0: The low-limit comparison is not enabled.
28:24	ch_sel	R/W	0	**ADC Channel for Limit Monitoring** Sets the ADC input channel for high- and low-limit thresholds. See *ADC_CTRL.ch_sel* for valid values for this field.
23:22	-	RO	0	**Reserved for Future Use** Do not modify this field.
21:12	ch_hi_limit	R/W	0x3FF	**High Limit Threshold** Sets the threshold for high-limit comparisons. This field is a 10-bit value compared against any ADC conversion on the channel set in the ch_sel field. ADC conversions greater than this field are over threshold and can result in interrupt assertion if the ch_hi_limit_en field is set. Valid values for this field are 0x000 to 0x3FF.
11:10	-	RO	0	**Reserved for Future Use** Do not modify this field.
9:0	ch_lo_limit	R/W	0	**Low Limit Threshold** Sets the threshold for low-limit comparisons. This field is a 10-bit value compared against any ADC conversion on the channel set in the ch_sel field. ADC conversions less than this field are under threshold and can result in interrupt assertion if the ch_lo_limit_en field is set. Valid values for this field are 0x000 to 0x3FF.

11. UART

The industry standard Universal Asynchronous Receiver/Transmitter (UART) peripheral communicates with external devices using a standard serial communications protocol. Each UART instance supports identical functionality and registers unless expressly noted otherwise.

The following features are provided:

- Flexible baud rate generation up to 4 Mbps with ±2% accuracy
- Programmable character size of 5-bits to 8-bits
- Stop bit settings of 1, 1.5, or 2-bits
- Parity settings of even, odd, mark (always 1), space (always 0), and no parity
- Automatic parity error detection with selectable parity bias
- Automatic framing error detection
- Separate 32-bytes deep transmit and receive FIFOs
- Flexible interrupt conditions
- Hardware flow control for RTS and CTS
- Null modem support
- Break generation and detection
- Wake-up from DEEPSLEEP on UART edge with no character loss
- Receive timeout detection
- DMA capable

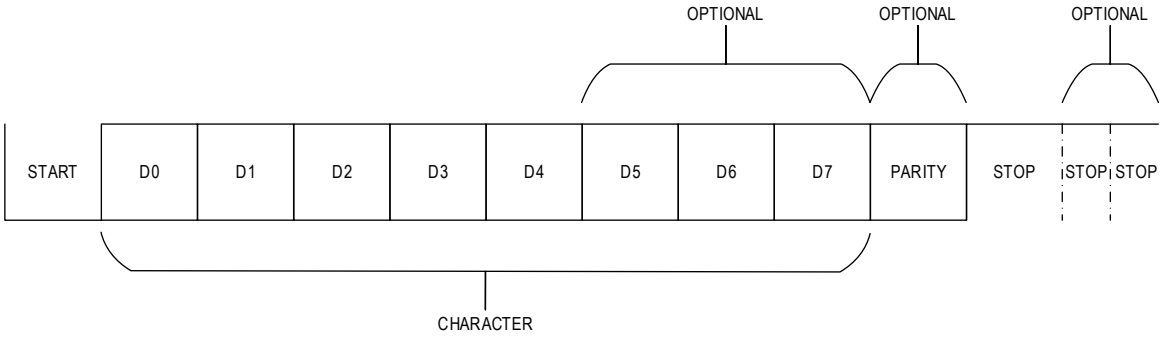
11.1 Instances

Three instances of the UART are provided: UART0, UART1, and UART2.

Each supports identical functionality and registers unless expressly noted otherwise. For simplicity, the UARTs are referred to UARTn where n = 0, 1, or 2.

11.2 UART Frame

Figure 11-1: UART Frame Diagram



Character sizes of 5 to 8 bits are supported. The field *UARTn_CTRL.char_size* is used to select the character size.

Stop bit support includes 1, 1.5, and 2 stop bits selected with the register field *UARTn_CTRL.stopbits.* Character sizes of 5 can have 1.5 stop bits. All other character sizes can have 1 or 2 stop bits.

Parity support includes even, odd, mark, space, or none. For no parity, set field *UARTn_CTRL.parity_en* to 0. For all other parity options, select one of the four parity options using the *UARTn_CTRL.parity* field and enable parity (*UARTn_CTRL.parity_en* = 1). Parity can be based on the number of logic-high bits or logic-low bits in the receive characters as set in the register bit *UARTn_CTRL.parmd*.

Break frames are transmitted by setting the field *UARTn_CTRL.break* to 1. A break sets all bits in the frame to 0.

When a break frame is received, two interrupts are available, *UARTn_INT_FL.break* is set to 1 when the break frame character is detected. and *UARTn_INT_FL.last_break* is set when the end of the break character is detected.

Note: A break condition does not set the frame error flag because breaks are not valid UART characters.

11.3 UART Interrupts

Interrupts can be generated for the for the conditions in the following table:

Table 11-1: UART Interrupt Conditions

Interrupt	Condition
TX FIFO Level	The transmit FIFO level transitions from being greater than to being equal to the set transmit threshold.
RX FIFO Level	The receive FIFO level equal to or greater than the set receive threshold.
RX FIFO Overrun	The receive FIFO is full but is still receiving data.
CTS State Change during hardware flow control	CTS is deasserted, which tells the UART to pause transmitting data. CTS is asserted, which tells the UART to resume transmitting data.
RX Parity	error
RX Frame	START or STOP bits were not detected.
RX Timeout	no characters were received within the set timeout period.
Break	Beginning and end of break.

11.4 UART Baud Rate Clock Source

The device can use multiple clock sources for baud rate generation. There are restrictions on the clock source based on whether the device is transmitting or receiving as shown in *Table 11-2*:

Table 11-2: Baud Rate Clock Sources in Transmit and Receive Operations

Clock Source	Transmit	Receive
Peripheral clock (PCLK)	Y	N
7.3728MHz Internal Oscillator	Y	Y

The UART bit rate clock is set using the *UARTn_CTRL.clksel* field. The fixed 7.3728MHz clock frequency should be used if the selected system clock (f_{SYS_CLK}) does not meet the bit rate requirements of the application.

The 7.3728MHz clock must be selected as the clock source before entering any of the low-power modes to use the UART wake-up function.

The UART always uses the peripheral clock for register access and logic operation.

11.5 UART Baud Rate Calculation

The UART peripheral clock, f_{PCLK} , is used as the input clock to the UART bit rate generator. The following fields are used to set the target bit rate for the UART instance.

- *UARTn_BAUD0.factor*: Selects the bit rate clock divisor.
- *UARTn_BAUD0.ibaud*: Sets the integer portion of the bit rate divisor.
- *UARTn_BAUD1.dbaud*: Sets the decimal portion of the bit rate divisor.

The equations below are used to determine the values for each of the bit rate fields required to achieve a target bit rate for the UART instance.

Equation 11-1: UART Bit Rate Divisor Equation

$$\text{DIV} = \frac{\text{f}_{\text{UART\_BIT\_RATE\_CLK}}}{(2^{(7-\text{UARTn\_BAUD0O}.clkdiv)} \times \text{Target Baud Rate})} \text{ where,}$$

Target Baud Rate is the desired UART interface speed

$\text{f}_{\text{UART\_BIT\_RATE\_CLK}}$ is the UART interface time base frequency. This frequency is either f_{PCLK} or the 7.3728MHz clock.

Note: UARTn_BAUD0.factor should be set to the lowest value that results in $\lfloor DIV \rfloor \geq 1$ to achieve the highest accuracy for the target bit rate. $\lfloor x \rfloor$ is a function that takes as input a real number x and gives as output the greatest integer less than or equal to x.

Equation 11-2: Bit Rate Integer Calculation

$$\text{UARTn\_BAUD0}.\text{ibaud} = \lfloor \text{DIV} \rfloor$$

Equation 11-3: Bit Rate Remainder Calculation

$$\text{y} = \lfloor (\text{DIV} - \text{UARTn\_BAUD0}.\text{ibaud}) \times 128 \rfloor$$

if (y > 3)

$$\text{UARTn\_BAUD1}.\text{dbaud} = \text{y} - 3$$

else

$$\text{UARTn\_BAUD1}.\text{dbaud} = \text{y} + 3$$

Example Baud Rate Calculation:

$$\text{Target Bit Rate} = 1{,}843{,}200 \text{ bits per second } (1.8 \text{ Mbps})$$

$$\text{f}_{\text{UART\_BIT\_RATE\_CLK}} = \text{f}_{\text{PCLK}} = 48 \text{ MHz}$$

$$\text{DIV} = \frac{48{,}000{,}000}{(2^{(7-\text{UARTn\_BAUD0}.factor)} \times 1{,}843{,}200)}$$

Table 11-3: Example Baud Rate Calculation Results, Target Bit Rate = 1.8Mbps

UARTn_BAUD0.factor	DIV	*UARTn_BAUD0.ibaud*	*UARTn_BAUD1.dbaud*
4	3.26	3	30
3	1.63	1	77
2	0.81	0 (Must be 1 or greater. The value selected for *UARTn_BAUD0*.factor is not valid)	
1	0.41	0 (Must be 1 or greater. The value selected for *UARTn_BAUD0*.factor is not valid)	
0	0.20	0 (Must be 1 or greater. The value selected for *UARTn_BAUD0*.factor is not valid)	

Table 11-3 shows the resulting DIV for each of the *UARTn_BAUD0.factor* field settings. With *UARTn_BAUD0.factor* set to 4 or3, the resulting DIV value is greater than 1. Setting *UARTn_BAUD0.factor* to 3 will generate the most accurate target bit rate because it is the smallest value that results in DIV ≥ 1. Using 3 for *UARTn_BAUD0.factor*, *UARTn_BAUD0.ibaud* is 1,

which is the integer portion of the 1.63 DIV calculation. The *UARTn_BAUD1.dbaud* field calculation based on *UARTn_BAUD0.factor* = 3, *UARTn_BAUD0.ibaud* = 1 and DIV = 1.63 is:

$$\text{UARTn\_BAUD1.dbaud} = \lfloor (1.63 - 1) \times 128 \rfloor - 3$$

The resulting field settings for the example 1,843,200 bps rate are:

- *UARTn_BAUD0.factor* = 3
- *UARTn_BAUD0.ibaud* = 1
- *UARTn_BAUD1*.dbaud = 77

11.6 FIFOs

Separate read (Rx FIFO) and write (Tx FIFO) FIFOs are implemented internally but both are accessed through the same *UARTn_FIFO.data* field. The current level of the transmit FIFO is read from *UARTn_STATUS.tx_fifo_cnt*, and the current level of the receive FIFO is read from UARTn_STATUS.*rx_num*.

11.6.1 Transmit FIFO Operation

Writing data to *UARTn_FIFO.data* increments the transmit FIFO pointer, *UARTn_STATUS.tx_fifo_cnt*, and loads the data into the transmit FIFO. Writes to the transmit FIFO will be ignored while *UARTn_STATUS.tx_fifo_cnt* = 8.

A special feature allows software to "peek" at the next character in the transmit FIFO using the *UARTn_TX_FIFO.data.* This allows software to examine the transmit FIFO without changing *UARTn_STATUS.tx_fifo_cnt*,

11.6.2 Receive FIFO Operation

Reads of *UARTn_FIFO.data* return the character values in the receive FIFO and decrement UARTn_STATUS.*rx_num*. Data for character sizes less than 7 bits are right justified. An overrun event will occur if a valid frame, including parity, is detected while UARTn_STATUS.*rx_num* = 8. In this case the frame will be discarded.

A parity error event indicates that the value read from *UARTn_FIFO.data* contains a parity error.

Note: The first character received after resetting the peripheral and changing the clock source to 7 MHz will automatically generate a framing error interrupt, regardless of the data sent. Software can either ignore the framing error or have a protocol in which the external transmitter always sends the first byte twice.

11.7 UART Configuration and Operation

To configure the UART, perform the following steps:

1. Clear *UARTn_CTRL.enable* to 0 to disable the peripheral.
2. Set *UARTn_CTRL* and *UARTn_THRESH_CTRL* registers for desired parity, character size, stop bits, flow control, and polarity.
3. If changing the clock source:
   a. Configure the UART_TX pins as GPIO.
   b. Change the clock source as desired.
   c. Reconfigure the UART_TX pin for the appropriate alternate function.
4. Configure the desired baud rate by setting *UARTn_BAUD0.factor* , *UARTn_BAUD0.ibaud* , and *UARTn_BAUD1.dbaud* and described in *UART Baud Rate Calculation*.
5. Set *UARTn_CTRL.rx_flush* to 1 and *UARTn_CTRL.tx_flush* to 1 to clear the receive and transmit FIFOs.
6. Set all bits in *UARTn_INT_FL*[9:0] to 1 to clear any pending UART interrupts.
7. Enable interrupts as desired by setting individual fields in *UARTn_INT_EN*.
8. Enable the UART by setting *UARTn_CTRL.enable* = 1.

11.8 Wake-up Time

Wake-up is configured by setting the UART I/O pins to GPIO with interrupt capability. Once a START bit is received, it generates a GPIO interrupt. The firmware interrupt handler must re-configure the GPIO pins to UART functionality to receive the first bit of the received character. The time for the operation to configure GPIO pins and return for interrupt service to do so is highly dependent upon the value of f_{SYS_CLK}.

11.9 Hardware Flow Control

When hardware flow control is enabled, the CTS (Clear-to-send) and RTS (Request-to-Send) external signals are directly managed by hardware without CPU intervention. RTS and CTS are active when flow control is enabled by setting the register bit *UARTn_CTRL.flow_ctl* = 1. The polarity of the CTS and RTS signals are configured with *UARTn_CTRL.flow_pol* and can be active low or active high.

In operation, the UART that wants to transmit data waits for its CTS input pin to be asserted. If CTS is asserted, then the UART begins transmitting data to the slave UART. If during the transmission the UART notices CTS is deasserted, the UART finishes transmitting the current character and then pauses to wait for CTS to return to an asserted level before transmitting more data.

If this UART is receiving data, and the receive FIFO reaches the level set in *UARTn_THRESH_CTRL.rts_fifo_thresh*, then the RTS signal of this UART is deasserted, informing the transmitting UART to stop sending data to this UART to prevent data overflow. Transmission resumes when the level of the receive FIFO drops below *UARTn_THRESH_CTRL.rts_fifo_thresh*, which automatically asserts RTS.

11.10 Registers

See *Table 3-3* for the base address of this peripheral/module. Each instance has its own independent set of the registers shown in *Table 11-4*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 11-4: UART Register Summary

Offset	Register	Description
[0x0000]	*UARTn_CTRL*	UARTn Control Register
[0x0004]	*UARTn_THRESH_CTRL*	UARTn Threshold Control Register
[0x0008]	*UARTn_STATUS*	UARTn Status Register
[0x000C]	*UARTn_INT_EN*	UARTn Interrupt Enable Register
[0x0010]	*UARTn_INT_FL*	UARTn Interrupt Flag Register
[0x0014]	*UARTn_BAUD0*	UARTn Baud Rate Integer Register
[0x0018]	*UARTn_BAUD1*	UARTn Baud Rate Decimal Register
[0x001C]	*UARTn_FIFO*	UARTn FIFO Read/Write Register
[0x0020]	*UARTn_DMA*	UARTn DMA Configuration Register
[0x0024]	*UARTn_TX_FIFO*	UARTn Transmit FIFO Peek Register

11.11 Register Details

Table 11-5: UART Control Register

UART Control			UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23:16	rx_to	R/W	0	**Receive Timeout Frame Count** Represents the number of frames to wait for a character. If the receive FIFO contains data, a Receive Timeout condition occurs if the number of frames in this register passes without the FIFO receiving any new data. If a timeout occurs, the hardware sets the receive timeout flag to 1 (*UARTn_INT_FL.rx_timeout* = 1).
15	clksel	R/W	0	**Bit Rate Clock Source Select** 0: $f_{UART\_BIT\_RATE\_CLK} = f_{PCLK}$ 1: $f_{UART\_BIT\_RATE\_CLK} = 7.3728MHz$.
14	break	R/W	0	**Transmit BREAK** Set this field to 1 to set the Tx line low during a character transmission. A character must be transmitting for this feature to operate. The Tx line remains low until this field is set to zero. 0: Normal UART operation. 1: Transmit zero during a character transfer.
13	null_modem	R/W	0	**Null Modem Support** 0: Normal operation for UARTn_RTS, CTS, and UARTn_TX/ UARTn_RX 1: Null Modem Mode: UARTn_RTS and UARTn_CTS pin signals are swapped. UARTn_RX and UARTn_TX pin signals are swapped.
12	flow_pol	R/W	0	**RTS/CTS Polarity** This field controls the polarity used for the RTS/CTS signals. Setting this field to 0 indicates active low assertion for the signals. Setting this field to 1 uses an active-high assertion. 0: RTS/CTS asserted is 0 1: RTS/CTS asserted is 1
11	flow_ctl	R/W	0	**Hardware Flow Control Enable** 0: Disabled. 1: Enabled.
10	stopbits	R/W	0	**Stop Bit Mode Select** Select the number of stop bits per character. Setting this field to 1 enables 1.5 stop bits for a 5-bit character and 2 stop bits for all other character sizes. 0: 1 stop bit. 1: 1.5 stop bits for 5-bit character size or 2 stop bits for all other character sizes
9:8	char_size	R/W	0	**Character Size** Set the number of bits per character. 0x0: 5 data bits 0x1: 6 data bits 0x2: 7 data bits 0x3: 8 data bits

UART Control			UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
7	bitacc	R/W	0	**Frame or Bit Accuracy Select** This field selects between either Frame Accuracy or Bit Accuracy for transmitting data. Frame Accuracy: Individual character bit durations may be varied by hardware to meet the target frame period. Bit accuracy: Character bit width is prioritized by the hardware. The overall frame accuracy may vary. For clock rates that cannot achieve the +/- 2% baud rate accuracy, this bit allows the software to select whether bit times are varied to approximate the desired baud rate. 0: Frame accuracy is prioritized. 1: Bit accuracy is prioritized. *Note: A frame includes the start, stop, all data bits, and parity bit/bits for the character being transmitted.*
6	rx_flush	R/W1O	0	**Receive FIFO Flush** Write 1 to flush the receive FIFO. Cleared to 0 by hardware when flush is completed
5	tx_flush	R/W1O	0	**Transmit FIFO Flush** Write 1 to flush the transmit FIFO. Cleared to 0 by hardware when flush is completed
4	parmd	R/W	1	**Parity Level Select** 0: Parity is based on the number of 0 bits in the character. 1: Parity is based on the number of 1 bits in the character.
3:2	parity	R/W	0	**Parity Mode Select** Set this field to the type of parity for the UART data. Mark parity always sets the parity bit to 1, and space parity always sets the parity bit to 0. For even parity, the parity bit is set to 1 if the number of 1 bits in the character is odd. Odd parity sets the parity bit to 1 if the number of 1 bits in the character is even. 0x0: Even parity 0x1: Odd Parity 0x2: Mark parity 0x3: Space parity *Note: For even and odd parity, the default uses the number of 1 bits in the character to calculate the parity bit. Clear UARTn_CTRL.parmd to 0 to base the parity on the number of 0 bits in the character.*
1	parity_en	R/W	0	**Parity Enable** If parity is enabled, parity is generated and verified based on the *UARTn_CTRL.parity* field. 0: No parity checking or generation. 1: Parity generation and checking are enabled.
0	enable	R/W	0	**UART Enable** Enabling the UART activates the bit rate generator. Disabling the UART flushes the transmit FIFO and the receive FIFO and disables the bit rate generator. 0: Disabled. 1: Enabled.

Table 11-6: UART Threshold Control Register

UART Threshold Control			UARTn_THRESH_CTRL	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:22	-	RO	0	**Reserved**
21:16	rts_fifo_thresh	R/W	0	**RTS Receive FIFO Threshold Level** When the receive FIFO level is equal to or greater than this value, deassert RTS output signal to inform the transmitting UART to stop sending data to this UART. Valid values are 1 to 7. When the receive FIFO level is equal to or greater than this value, de-assert RTS output signal to inform the transmitting UART to stop sending data. Valid values are 1 to 32.
15:14	-	RO	0	**Reserved**
13:8	tx_fifo_thresh	R/W	0	**Transmit FIFO Threshold Level** When the transmit FIFO level is less than or equal to this value, hardware sets the *UARTn_INT_FL.tx_fifo_thresh* interrupt flag. Valid values are 1 to 32. Set this field greater than 1 to avoid a stall condition when transmitting UART data. *Note: See Table 11-1 for description.*
7:6	-	RO	0	**Reserved**
5:0	rx_fifo_thresh	R/W	0	**Receive FIFO Threshold Level** When the receive FIFO level is equal to or greater than this value, hardware sets the *UARTn_INT_FL.rx_fifo_thresh* interrupt flag. Valid values are 1 to 32. Set this field to less than 32 to avoid a receive FIFO overrun condition. *Note: See Table 11-1 for description.*

Table 11-7: UART Status Register

UART Status			UARTn_STATUS	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:25	-	RO	0	**Reserved**
24	rx_to			**Receiver Timeout Status** 0: no timeout has occurred 1: timeout occurred
23:22	-	RO	0	**Reserved**
21:16	tx_fifo_cnt	RO	0	**Number of characters in the Transmit FIFO** Read this field to determine the number of characters in the transmit FIFO.
15:14	-	RO	0	**Reserved**
13:8	rx_num	RO	0	**Number of Characters in the Receive FIFO** Read this field to determine the number of characters in the receive FIFO.
7	tx_full	RO	0	**Transmit FIFO Full Status** 0: FIFO is not full. 1: FIFO is full.
6	tx_empty	RO	1	**Transmit FIFO Empty Status** 0: FIFO is not empty. 1: FIFO is empty.
5	rx_full	RO	0	**Receive FIFO Full Status** 0: FIFO is not full. 1: FIFO is full.
4	rx_empty	RO	1	**Receive FIFO Empty Status** 0: FIFO is not empty. 1: FIFO is empty.

UART Status			UARTn_STATUS	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
3	break	RO	0	**Break Status** Set while a break condition exists. 0: A break has not been received. 1: A break condition exists.
2	parity	RO	0	**Parity Bit State** This field returns the state of the parity bit. 0: Parity bit is 0. 1: Parity bit is 1.
1	rx_busy	RO	0	**Receive Busy** This field reads 1 when the UART is receiving data. 0: UART is not actively receiving data. 1: UART is actively receiving data.
0	tx_busy	RO	0	**Transmit Busy** This field reads 1 when the UART is transmitting data. 0: UART is not actively transmitting data. 1: UART is transmitting data.

Table 11-8: UART Interrupt Enable Register

UART Interrupt Enable			UARTn_INT_EN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	last_break	R/W	0	**BREAK End Interrupt Enable** Enables the BREAK End Detection interrupt. 0: Disabled. 1: Enabled.
8	rx_timeout	R/W	0	**Receive Timeout Interrupt Enable** 0: Disabled. 1: Enabled.
7	break	R/W	0	**Received BREAK Interrupt Enable** Enables the BREAK Detection interrupt. 0: Disabled. 1: Enabled.
6	tx_fifo_thresh	R/W	0	**Transmit FIFO Threshold Level Interrupt Enable** Enables the transmit FIFO threshold level interrupt. This interrupt occurs when the number of entries in the transmit FIFO is equal or less than the value set in *UARTn_THRESH_CTRL.tx_fifo_thresh*. 0: Disabled. 1: Enabled.
5	tx_fifo_almost_empty	R/W	0	**Transmit FIFO Almost Empty Interrupt Enable** This interrupt occurs when there is one byte remaining in the transmit FIFO. 0: Disabled. 1: Enabled.
4	rx_fifo_thresh	R/W	0	**Receive FIFO Threshold Level Interrupt Enable** 0: Disabled 1: Enabled *Note:* See *Table 11-1 for description.*

UART Interrupt Enable			UARTn_INT_EN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
3	rx_overrun	R/W	0	**Receive FIFO Overrun Interrupt Enable** 0: Disabled. 1: Enabled. *Note: See Table 11-1 for description.*
2	cts_change	R/W	0	**CTS State Change Interrupt Enable** Enable the CTS level change interrupt event. This is often referred to as Modem Status Interrupt. 0: Disabled. 1: Enabled.
1	rx_parity_error	R/W	0	**Receive Parity Error Interrupt Enable** 0: Disabled. 1: Enabled.
0	rx_frame_error	R/W	0	**Receive Frame Error Interrupt Enable** 0: Disabled. 1: Enabled.

Table 11-9: UART Interrupt Flags Register

UART Interrupt Flags			UARTn_INT_FL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	last_break	R/W1C	0	**BREAK End Interrupt Flag** When the UART receives a series of BREAK frames, this flag is set when the last BREAK frame is received. Write 1 to clear this field. 0: Last BREAK condition has not occurred. 1: Last BREAK condition has occurred. *Note:* See *Table 11-1 for description.*
8	rx_timeout	R/W1C	0	**Receive Frame Timeout Interrupt Flag** This field is set when a receive frame timeout occurs. Write 1 to clear this field. 0: Receive frame timeout has not occurred. 1: A receive frame timeout was detected by the UART.
7	break	R/W1C	0	**Received Break Interrupt Flag** When the UART receives a series of BREAK frames, this flag is set when the first BREAK frame is received. Write 1 to clear this field. 0: Break condition not occurred 1: Break condition occurred. *Note:* See *Table 11-1 for description.*
6	tx_fifo_thresh	R/W1C	0	**Transmit FIFO Threshold Interrupt Flag** Set when the number of entries in in the transmit FIFO is less than or equal to the transmit FIFO level set in *UARTn_THRESH_CTRL.tx_fifo_thresh*. Write 1 to clear. 0: The transmit FIFO level is below the threshold set in *UARTn_THRESH_CTRL.tx_fifo_thresh*. 1: The transmit FIFO level is equal to or greater than the *UARTn_THRESH_CTRL.tx_fifo_thresh*. *Note:* See *Table 11-1 for description.*

UART Interrupt Flags			UARTn_INT_FL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
5	tx_fifo_almost_empty	R/W1C	0	**Transmit FIFO Almost Empty Interrupt Flag** This field is set when there is one byte remaining in the transmit FIFO. Write 1 to clear. 0: Transmit FIFO level is greater than 1. 1: Transmit FIFO is Almost Empty.
4	rx_fifo_thresh	R/W1C	0	**Receive FIFO Threshold Interrupt Flag** Set when number of entries in the receive FIFO is equal to or greater than the receive FIFO threshold level as set in the *UARTn_THRESH_CTRL.rx_fifo_thresh* field. Data must be read from the receive FIFO to reduce the level below the threshold to guarantee this interrupt does not occur again after clearing the flag. Write 1 to clear this field. 0: The number of bytes in the receive FIFO is below the threshold level. 1: The number of bytes in the receive FIFO is equal to or greater than the threshold level. *Note:* See *Table 11-1 for description.*
3	rx_overrun	R/W1C	0	**Receive FIFO Overrun Interrupt Flag** This field is set if the receive FIFO is full and an additional byte is received resulting in a FIFO overrun condition. If this field is set at least one byte of received data has been lost. Write 1 to clear. 0: Receive FIFO overrun has not occurred. 1: Receive FIFO overrun occurred.
2	cts_change	R/W1C	0	**CTS Interrupt Flag** Also called Modem Status Interrupt. Write 1 to clear. 0: No state change 1: CTS has changed state.
1	rx_parity_error	R/W1C	0	**Receive Parity Error Status Flag** Set if a parity error is detected. This flag applies to data received only. Write 1 to clear. 0: Parity error has not been detected. 1: Parity error detected.
0	rx_frame_error	R/W1C	0	**Frame Error Status Flag** Set if a frame error occurs while receiving data. Write 1 to clear. 0: Frame error not occurred. 1: Frame error occurred.

Table 11-10: UART Rate Integer Register

UART Baud Rate Integer			UARTn_BAUD0	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:19	-	R/W	0	**Reserved**

UART Baud Rate Integer			UARTn_BAUD0	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
18:16	factor	R/W	0	**Bit Rate Clock Divisor** This field is used to divide the bit rate clock by the selected Clock Divider value. 0x0: 128 0x1: 64 0x2: 32 0x3: 16 0x4: 8 0x5: Reserved 0x6: Reserved 0x7: Reserved *Note: See the UART Baud Rate Calculation section for details of determining this field's value for a given UART bit rate.*
15:12	-	R/W	0	**Reserved**
11:0	ibaud	R/W	0	**Integer Portion of Baud Rate Divisor** This field contains the integer value of the bit rate divisor. See the *UART Baud Rate Calculation* section to calculate this field's value for a given UART bit rate.

Table 11-11: UART Baud Rate Decimal Register

UART Baud Rate Decimal Register			UARTn_BAUD1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	R0	0	**Reserved**
6:0	dbaud	R/W	0	**Decimal Portion of Baud Rate Divisor** This field contains the remainder portion of the bit rate divisor. See the *UART Baud Rate Calculation* section to calculate this field's value for a given UART bit rate.

Table 11-12: UART FIFO Register

UART FIFO Register			UARTn_FIFO	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	0	**Reserved**
7:0	fifo	R/W	N/A	**UART FIFO Register** Reading this field reads data from the receive FIFO and writes to this field write to the transmit FIFO.

Table 11-13: UART DMA Configuration Register

UART DMA Configuration Register			UARTn_DMA	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:22	-	R/W	0	**Reserved**
21:16	rxdma_level	R/W	0	**Receive FIFO Level DMA Trigger** If the receive FIFO level is greater than this value, the DMA channel transfers data from the receive FIFO. DMA transfers continue until the receive FIFO is empty. To avoid an receive FIFO overrun, do not set this value to 32. Values above 32 are reserved for future use.
15:14	-	R/W	0	**Reserved**

UART DMA Configuration Register			UARTn_DMA	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
13:8	txdma_level	R/W	0	**Transmit FIFO Level DMA Trigger** If the transmit FIFO level is less than this value, the DMA channel transfers data into the transmit FIFO. DMA transfers continue until the transmit FIFO is full. To avoid stalling a UART transmission, do not set this value to 1 or 0. *Note: Values above 32 are Reserved for Future Use.*
7:6	-	R/W	0	**Reserved**
5	rxdma_auto_to			**Receive DMA Timeout Start** If *UARTn_CTRL.rx_to* causes an receive timeout and *UARTn_STATUS.rx_num* is greater than zero, the DMA transfer will start. Auto-clear after set. 0: Start not initiated. 1: Start DMA transfer.
4	-	R/W	0	**Reserved**
3	rxdma_start	R/W1	0	**Receive DMA Start** Regardless of the setting of *UARTn_DMA.rxdma_level*, start the DMA transfer if *UARTn_STATUS.rx_num* is greater than zero. Auto-clear after set. 0: Start not initiated. 1: Start DMA transfer.
2	-	R/W	0	**Reserved**
1	rxdma_en	R/W	0	**Receive FIFO DMA Channel Enable** 0: Disabled. 1: Enabled.
0	txdma_en	R/W	0	**Transmit FIFO DMA Channel Enable** 0: Disabled. 1: Enabled.

Table 11-14: UART Transmit FIFO Data Output Register

UART Transmit FIFO Register			UARTn_TX_FIFO	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	0	**Reserved**
7:0	data	RO	0	**Transmit FIFO Peek Register** Reads from this register return the next character available for transmission at the end of the transmit FIFO. If no data is available, reads of this field return 0. Reads from this register do not affect the transmit FIFO state.

12. I²C Master/Slave Serial Communications Peripheral (I²C)

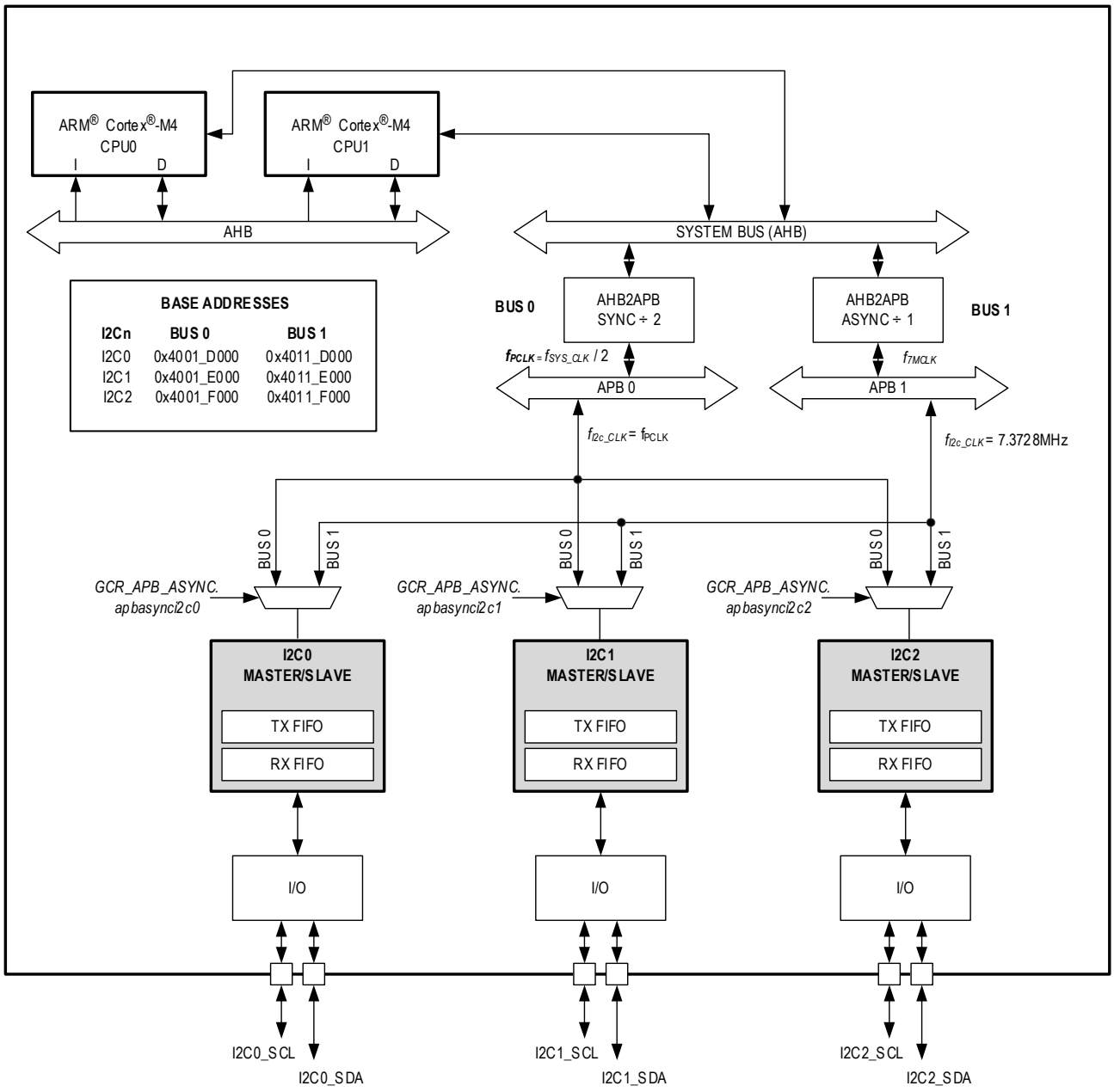
The I²C peripherals can be configured as either an I²C master or I²C slave at standard data rates. For simplicity, I2Cn is used throughout this section to refer to any of the I²C peripherals.

The MAX32665/MAX32666 can access the I2Cn peripherals through two different busses (clock sources). Bus 0 is the traditional I²C implementation where the peripheral timing is based on the synchronous APB clock ($f_{SYS\_CLK}/2$). If the system clock selection is changed, the timing for the I2Cn peripherals changes as well. Managing the timing for I²C can become complex if the system clock is ever changed, especially real-time, for example, when trying to maximize battery life. Bus 1, on the other hand, provides access to the I2Cn peripherals through an asynchronous APB running at a fixed clock of 7.3728MHz and simplifies the management of the I2Cn peripherals because their timing is not dependent on a clock which may change real-time.

The user selects the desired bus using either the I2Cn registers designated BUS 0 or designated BUS 1. The registers for each of the bus instances are identical except for the base address. Likewise, the registers for each of the I2Cn peripherals are identical except for the base addresses. The base addresses are illustrated in *Figure 12-1*.

For detailed information on I²C bus operation, refer to Application Note 4024 *SPI/ I2C Bus Lines to Control Multiple Peripherals.*

Figure 12-1: I²C Block Diagram



12.1 I²C Master and Slave Features

Each I²C master and slave is compliant with the I²C Bus Specification and includes the following features:

- Communicates through a serial data bus (SDA) and a serial clock line (SCL)
- Operates as either a master or slave device as a transmitter or receiver
- Supports I²C Standard Mode, Fast Mode, Fast Mode Plus, and High Speed (Hs) mode
- Transfers data at rates up to:
  - 100kbps in Standard Mode
  - 400kbps in Fast Mode
  - 1Mbps in Fast Mode Plus
  - 3.4Mbps in Hs Mode
- Supports multi-master systems, including support for arbitration and clock synchronization for Standard, Fast, and Fast Plus modes
- Supports 7- and 10-bit addressing
- Supports RESTART condition
- Supports clock stretching
- Supports transfer status interrupts and flags
- Supports DMA data transfer support
- Supports I²C timing parameters fully controllable through software
- Supports glitch filter and Schmitt trigger hysteresis on SDA and SCL
- Supports control, status, and interrupt events for maximum flexibility
- Supports independent an 8-byte receive FIFO and an 8-byte transmit FIFO
- Supports transmit FIFO preloading
- Supports programmable interrupt threshold levels for the transmit and receive FIFO

12.2 Instances

The three instances of the peripheral are shown in *Figure 12-1*. *Table 12-1* lists the locations of the SDA and SCL signals for each of the I2Cn peripherals per package.

Table 12-1: MAX32665/MAX32666 I²C Peripheral Pins

I2Cn	Alternate Function Name	Alternate Function Number	109 WLP	121 CTBGA
I2C0	I2C0_SCL	AF1	P0.6	P0.6
	I2C0_SDA	AF1	P0.7	P0.7
I2C1	I2C1_SCL	AF1	P0.14	P0.14
	I2C1_SDA	AF1	P0.15	P0.15
I2C2	I2C2_SCL	AF1	P1.14	P1.14
	I2C2_SDA	AF1	P1.15	P1.15

Note: The pins apply to both BUS 0 and BUS 1.

12.3 I²C Overview

12.3.1 I²C Bus Terminology

Table 12-2 contains terms and definitions used in this chapter for the I²C Bus Terminology.

Table 12-2: I²C Bus Terminology

Term	Definition
Transmitter	The device sending data on the bus.
Receiver	The device receiving data from the bus.
Master	The device that starts a transfer, generates the clock signal, and terminates a transfer.
Slave	The device addressed by a master.
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message.
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one can do so, and the resulting message is not corrupted.
Synchronization	The procedure to synchronize the clock signals of two or more devices.
Clock Stretching	When a slave device holds SCL low to pause a transfer until it is ready. Clock stretching is an optional feature according to the I²C specification; thus, a master does not have to support slave clock stretching if none of the slaves in the system are capable of clock stretching.

12.3.2 I²C Transfer Protocol Operation

The I²C protocol operates over a two-wire bus: a clock circuit (SCL) and a data circuit (SDA). I²C is a half-duplex protocol: only one device is allowed to transmit on the bus at a time.

Each transfer is initiated when the bus master sends a START or repeated START condition. It is followed by the I²C slave address of the targeted slave device plus a read/write bit. The master can transmit data to the slave (a 'write' operation) or receive data from the slave (a 'read' operation). Information is sent most significant bit (MSB) first. Following the slave address, the master indicates a read or write operation and then exchanges data with the addressed slave. An acknowledge bit is sent by the receiving device after each byte is transferred. When all necessary data bytes have been transferred, a STOP or RESTART condition is sent by the bus master to indicate the end of the transaction. After the STOP condition has been sent, the bus is idle and ready for the next transaction. After a RESTART condition is sent, the same master begins a new transmission. The number of bytes that can be transmitted per transfer is unrestricted.

12.3.3 START and STOP Conditions

A START condition occurs when a bus master pulls SDA from high to low while SCL is high, and a STOP condition occurs when a bus master allows SDA to be pulled from low to high while SCL is high. Because these are unique conditions that cannot occur during normal data transfer, they are used to denote the beginning and end of the data transfer.

12.3.4 Master Operation

I²C transmit and receive data transfer operations occur through the *I2Cn_FIFO* register. Writes to the register load the transmit FIFO and reads of the register return data from the receive FIFO. If a slave sends a NACK in response to a write operation, the I²C master generates an interrupt. The I²C controller can be configured to issue a STOP condition to free the bus.

The receive FIFO contains the received data. If the receive FIFO is full or the transmit FIFO is empty, the I²C master stops the clock to allow time to read bytes from the receive FIFO or load bytes into the transmit FIFO.

12.3.5 Acknowledge and Not Acknowledge

An acknowledge bit (ACK) is generated by the receiver, whether I²C master or slave, after every byte received by pulling SDA low. The ACK bit is how the receiver tells the transmitter that the byte was successfully received and another byte might be sent.

A Not Acknowledge (NACK) occurs if the receiver does not generate an ACK when the transmitter releases SDA. A NACK is generated by allowing SDA to float high during the acknowledge time slot. The I²C master can then either generate a STOP condition to abort the transfer or generate a repeated START condition (send a START condition without an intervening STOP condition) to start a new transfer.

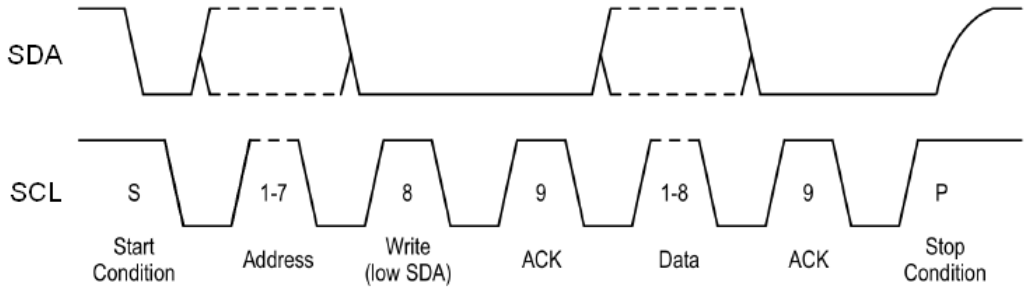
A receiver can generate a NACK after a byte transfer if any of the following conditions occur:

- No receiver is present on the bus with the transmitted address. In that case, no device responds with an acknowledge signal.
- The receiver cannot receive or transmit because it is busy and is not ready to start communication with the master.
- During the transfer, the receiver receives data or commands it does not understand.
- During the transfer, the receiver is unable to receive any more data.
- If an I²C master has requested data from a slave, it signals the slave to stop transmitting by sending a NACK following the last byte it requires.

12.3.6 Bit Transfer Process

Both SDA and SCL circuits are open-drain, bidirectional circuits. Each requires an external pullup resistor that ensures each circuit is high when idle. The I²C specification states that during data transfer, the SDA line can change state only when SCL is low and that SDA is stable and can be read when SCL is high, as shown in *Figure 12-2*.

Figure 12-2: I²C Write Data Transfer



An example of an I²C data transfer is as follows:

1. A bus master indicates a data transfer to a slave with a START condition.
2. The master then transmits one byte with a 7-bit slave address and a single read-write bit: a zero for a write or a one for a read.
3. During the next SCL clock following the read-write bit, the master releases SDA. During this clock period, the addressed slave responds with an ACK by pulling SDA low.
4. The master senses the ACK condition and begins transferring data. If reading from the slave, it floats SDA and allows the slave to drive SDA to send data. After each byte, the master drives SDA low to acknowledge the byte. If writing to the slave, the master drives data on the SDA circuit for each of the eight bits of the byte and then floats SDA during the ninth bit to allow the slave to reply with the ACK indication.
5. After the last byte is transferred, the master indicates the transfer is complete by generating a STOP condition. A STOP condition is generated when the master pulls SDA from a low to high while SCL is high.

12.4 I²C Configuration and Usage

12.4.1 SCL and SDA Bus Drivers

SCL and SDA are open-drain signals. In this device, once the I²C peripheral is enabled and the proper GPIO alternate function is selected, the corresponding pad circuits are automatically configured as open-drain outputs. However, SCL can also be optionally configured as a push-pull driver to conserve power and avoid the need for any pullup resistor. This should only be used in systems where no I²C slave device can hold SCL low, such as for clock stretching. Push-pull operation is enabled by setting *I2Cn_CTRL.scl_pp_mode* to 1. SDA, on the other hand, always operates in open-drain mode.

12.4.2 SCL Clock Configurations

The SCL frequency depends on the values of the I²C peripheral clock and the values of the external pullup resistor and trace capacitance on the SCL clock line.

Note: An external RC load on the SCL line affects the target SCL frequency calculation.

12.4.3 SCL Clock Generation for Standard, Fast and Fast-Plus Modes

The master generates the I²C clock on the SCL line. When operating as a master, software must configure the *I2Cn_CLK_HI* and *I2Cn_CLK_LO* registers for the desired I²C operating frequency.

The MAX32665/MAX32666 can select the source for the I²C peripheral clock. The software can select between the system peripheral clock, f_{PCLK} (accessed through I2Cn Bus 0 registers), or the 7.3728MHz oscillator (accessed through I2Cn Bus 1 registers). The frequency f_{PCLK} is f_{SYS_CLK} divided by 2. All three I2Cn peripherals default to Bus 0. Switching one or more of the peripherals to Bus 1 consists of first selecting the 7.3728MHz bus (*GCR_APBASYNC.apbasynci2c[n]*, where [n] = 0, 1, 2), followed by accessing it through the corresponding I2Cn Bus 1 registers. The base addresses are shown in the block diagram shown in *Figure 12-1*.

The SCL high time is configured in the I²C clock high time register field *I2Cn_CLK_HI.scl_hi using Equation 12-1.* The SCL low time is configured in the I²C Clock Low Time register field *I2Cn_CLK_LO.scl_lo using Equation 12-2*. Each of these fields is 8-bits. The value $\frac{1}{t_{I2C\_CLK}}$ is either f_{PCLK} or 7.3728MHz.

Equation 12-1: I²C Clock High Time Calculation

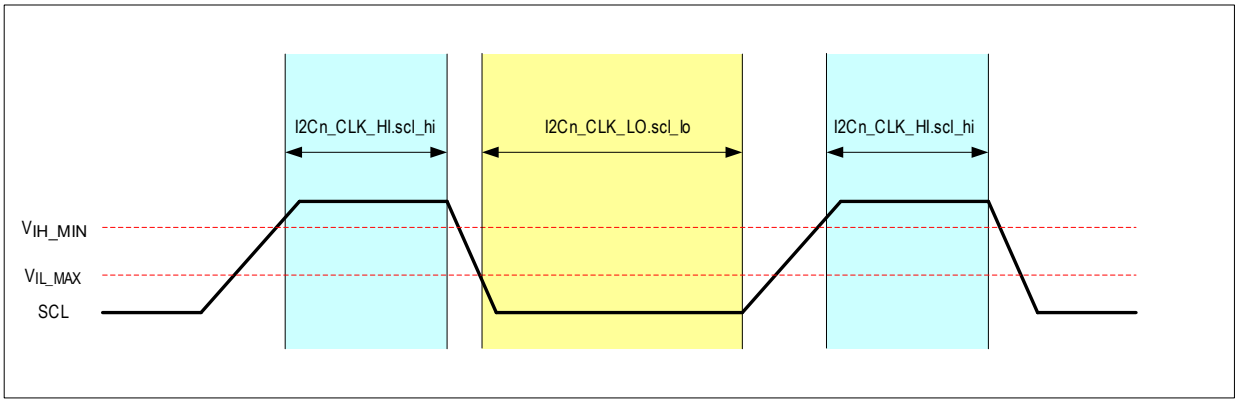
$$t_{SCL\_HI} = t_{I2C\_CLK} \times (\text{I2Cn\_CLK\_HI.scl\_hi} + 1)$$

Equation 12-2: I²C Clock Low Time Calculation

$$t_{SCL\_LO} = t_{I2C\_CLK} \times (\text{I2Cn\_CLK\_LO.scl\_lo} + 1)$$

Figure 12-3 shows the association between the SCL clock low and high times for Standard, Fast, and Fast Plus I²C frequencies.

Figure 12-3: I²C SCL Timing for Standard, Fast, and Fast-Plus Modes



During synchronization, external masters or external slaves may be driving SCL simultaneously. This affects the SCL duty cycle. By monitoring SCL, the controller can determine whether an external master or slave is holding SCL low. In either case, *the controller waits until SCL is high before starting to count the number of SCL high cycles*. Similarly, if an external master pulls SCL low before the controller has finished counting SCL high cycles, then the controller starts counting SCL low cycles and releases SCL once the time period, *I2Cn_CLK_LO.scl_lo*, has expired.

Because the controller does not start counting the high/low time until the input buffer detects the new value, the actual clock behavior is based on many factors. These include bus loading, other devices on the bus holding SCL low, and the filter delay time of this device.

12.4.4 SCL Clock Generation for Hs-Mode

The values programmed into the I2Cn_HS_CLK.hs_clk_lo register and I2Cn_HS_CLK.hs_clk_hi register must be determined to operate the I²C interface in Hs-Mode at its maximum speed (~3.4MHz). Since the Hs-Mode operation is entered by first using one of the lower speed modes for pre-amble, a relevant lower speed mode must also be configured. See SCL Clock Generation for Standard, Fast and Fast-Plus Modes for information regarding the configuration of lower speed modes.

12.4.4.1 Hs-Mode Timing

With I²C bus capacitances less than 100pf, the following specifications are extracted from the I²C-bus Specification User Manual Rev. 6 April 2014 https://www.nxp.com/docs/en/user-guide/UM10204.pdf
t_{LOW_MIN}, the minimum low time for the I²C bus clock, = 160ns
t_{HIGH_MIN}, the minimum high time for the I²C bus clock, = 60ns
t_{rCL_MAX}, the maximum rise time of the I²C bus clock, = 40ns
t_{fCL_MAX}, the maximum fall time of the I²C bus clock, = 40ns

12.4.4.2 Hs-Mode Clock Configuration

The maximum Hs-mode bus clock frequency can now be determined. The system clock frequency, f_{SYS_CLK}, must be known. Hs-Mode timing information from Hs-Mode Timing must be used.

Equation 12-3: I²C Target SCL Frequency

This is the desired target for the maximum I²C clock speed.

$$Target\ I^2C\ Clock\ Frequency,\ \ f_{SCL} = \frac{1}{t_{SCL}}.$$

Equation 12-4: I²C Peripheral Source Clock Period

$$t_{I2C\_CLK} = \begin{Bmatrix} \frac{2}{f_{SYS\_CLK}}, & \text{GCR\_APBASYNC}.apbasynci2c[n] = 0 \\ & [n] = 0,1,2 \\ 7.3728\text{MHz}, & \text{GCR\_APBASYNC}.apbasynci2c[n] = 1 \\ & [n] = 0,1,2 \end{Bmatrix}$$

In Hs-Mode, the analog glitch filter within the device adds a minimum delay of t_{AF_MIN} = 10ns.

Equation 12-5: Determining the I2Cn_HS_CLK.hs_clk_lo Register Value

$$\text{I2Cn\_HS\_CLK}.hs\_clk\_lo = \text{MAX}(\left\lfloor\left(\frac{t_{LOW\_MIN} + t_{FCL\_MAX} + t_{I2C\_CLK} - t_{AF\_MIN}}{t_{I2C\_CLK}}\right)\right\rfloor - 1,\quad \frac{t_{SCL}}{t_{I2C\_CLK}} - 1)$$

Equation 12-6: Determining the I2Cn_HS_CLK.hs_clk_hi Register Value

$$\text{I2Cn\_HS\_CLK}.hs_{clk_{hi}} = \left\lfloor\left(\frac{t_{HIGH_{MIN}} + t_{rCL_{MAX}} + t_{I2C_{CLK}} - t_{AF_{MIN}}}{t_{I2C_{CLK}}}\right)\right\rfloor - 1$$

Equation 12-7: The Calculated Frequency of the I²C Bus Clock Using the Results of Equation 12-5 and Equation 12-6

$$Calculated\ Frequency = ((\text{I2Cn\_HS\_CLK}.hs\_clk\_hi + 1) + (\text{I2Cn\_HS\_CLK}.hs\_clk\_lo + 1)) \times t_{I2C\_CLK}$$

Table 12-3 shows the I²C bus clock calculated frequencies given different f_{SYS_CLK} frequencies.

Table 12-3: Calculated I²C Bus Clock Frequencies

f_{SYS_CLK} (MHz)	I2Cn_HS_CLK.hs_clk_hi	I2Cn_HS_CLK.hs_clk_lo	Calculated Frequency (MHz)
96	4	9	3.2
48	2	4	3.0
24	1	2	2.4

12.4.5 Master Mode Addressing

After a START condition, an I²C slave address byte is transmitted. The I²C slave address is composed of a slave address followed by a read/write bit.

Table 12-4: I²C Slave Address Format

Slave Address Bits		R/W Bit	Description
0000	000	0	General Call Address
0000	000	1	START Condition
0000	001	x	CBUS Address
0000	010	x	Reserved for different bus format
0000	011	x	Reserved for future purposes
0000	1xx	x	Hs-Mode master code
1111	1xx	x	Reserved for future purposes
1111	0xx	x	10-bit slave addressing

In 7-bit addressing mode, the master sends one address byte. To address a 7-bit address slave, first clear *I2Cn_M.sl_ex_addr* = 0, then write the address to the transmit FIFO formatted as follows where A*n* is address A6:A0.

Master Writing to Slave : 7-bit address : [A6 A5 A4 A3 A2 A1 A0 0]

Master Reading from Slave : 7-bit address : [A6 A5 A4 A3 A2 A1 A0 1]

In 10-bit addressing mode (*I2Cn_M.sl_ex_addr* = 1), the first byte the master sends is the 10-bit slave addressing byte, including the first two bits of the 10-bit address, followed by a 0 for the R/W bit. That is followed by a second byte representing the remainder of the 10-bit address. If the operation is a write, this is followed by data bytes to be written to the slave. If the operation is a read, it is followed by a repeated START. The software then writes the 10-bit address again with a 1 for the R/W bit. This I²C then starts receiving data from the slave device.

12.4.6 Master Mode Operation

The peripheral operates in master mode when the master mode enable bit, *I2Cn_CTRL.mst*, is set to 1. To initiate a transfer, the master generates a START condition by setting *I2Cn_M.start* = 1. If the bus is busy, it does not generate a START condition until the bus is available.

A master can communicate with multiple slave devices without relinquishing the bus. Instead of generating a STOP condition after communicating with the first slave, the master generates a repeated START condition, or RESTART, by setting *I2Cn_M.restart* = 1. If a transaction is in progress, the peripheral completes the transaction before generating a RESTART. The peripheral then transmits the slave address stored in the transmit FIFO. The *I2Cn_M.restart* bit is automatically cleared to 0 as soon as the master begins a RESTART condition.

I2Cn_M.start is automatically cleared to 0 after the master has completed a transaction and sent a STOP condition.

The master can also generate a STOP condition by setting *I2Cn_M.stop* = 1.

If both START and RESTART conditions are enabled simultaneously, a START condition is generated first. Then, at the end of the first transaction, a RESTART condition is generated.

If both RESTART and STOP conditions are enabled simultaneously, a STOP condition is not generated. Instead, a RESTART condition is generated. After the RESTART condition is generated, both bits are cleared.

If START, RESTART, and STOP are all enabled simultaneously, a START condition is first generated. At the end of the first transaction, a RESTART condition is generated. The *I2Cn_M.stop* bit is cleared and ignored.

A slave cannot generate START, RESTART, or STOP conditions. Therefore, when master mode is disabled, the *I2Cn_M.start*, *I2Cn_M.restart*, and *I2Cn_M.stop* bits are all cleared to 0.

For master mode operation, the following registers should only be configured when either the I²C peripheral is disabled or the I²C bus is guaranteed to be idle or free. If this peripheral is the only master on the bus, then changing the registers outside of a transaction (*I2Cn_MASTER_CTRL.start* = 0) satisfies this requirement:

- *I2Cn_CTRL.mst*
- *I2Cn_CTRL.rx_mode*
- *I2Cn_CTRL.scl_pp_mode*
- *I2Cn_CTRL.hs_mode*
- *I2Cn_RX_CTRL1.rx_cnt*
- *I2Cn_MASTER_CTRL.sl_ex_addr*
- *I2Cn_MASTER_CTRL.mcode*
- *I2Cn_CLK_LO.scl_lo*
- *I2Cn_CLK_HI.scl_hi*
- *I2Cn_HS_CLK.hs_clk_lo*
- *I2Cn_HS_CLK.hs_clk_hi*

In contrast to the above set of registers, these registers below can be safely (re)programmed at any time:

- All interrupt flags and interrupt enables
- *I2Cn_TX_CTRL0.tx_thresh*
- *I2Cn_RX_CTRL0.rx_thresh*
- *I2Cn_TIMEOUT.to*
- *I2Cn_DMA.rxen*
- *I2Cn_DMA.txen*
- *I2Cn_FIFO.data*
- *I2Cn_MASTER_CTRL.start*
- *I2Cn_MASTER_CTRL.restart*
- *I2Cn_MASTER_CTRL.stop*

12.4.6.1 Master Mode Receiver Operation

When in master mode, initiating a Master Receiver operation begins with the following sequence:

1. Write the number of data bytes to receive to the I²C receive count field (*I2Cn_RX_CTRL1.rx_cnt).*
2. Write the I²C slave address byte to the *I2Cn_FIFO* register with the R/W bit set to 1
3. Send a START condition by setting *I2Cn_MASTER_CTRL.start* = 1
4. The slave address is transmitted by the controller from the *I2Cn_FIFO* register.
5. The I²C controller receives an ACK from the slave, and the controller sets the address ACK interrupt flag (*I2Cn_INT_FL0.addr_ack* = 1).
6. The I²C controller receives data from the slave and automatically ACKs each byte. The software must retrieve this data by reading the *I2Cn_FIFO* register.
7. Once *I2Cn_RX_CTRL1.rx_cnt* data bytes have been received, the I²C controller sends a NACK to the slave and sets the Transfer Done Interrupt Status Flag (*I2Cn_INT_FL0.done* = 1).
8. If *I2Cn_MASTER_CTRL.restart* or *I2Cn_MASTER_CTRL.stop* is set, then the I²C controller sends a repeated START or STOP, respectively.

12.4.6.2 Master Mode Transmitter Operation

When in master mode, initiating a master transmitter operation begins with the following sequence:

1. Write the I²C slave address byte to the *I2Cn_FIFO* register with the R/W bit set to 0.
2. Write the desired data bytes to the *I2Cn_FIFO* register, up to the depth of the transmit FIFO. (e.g., If the transmit FIFO size is 8 bytes, the software may write one address byte and seven data bytes before starting the transaction.)
3. Send a START condition by setting *I2Cn_MASTER_CTRL.start* = 1
4. The controller transmits the slave address byte written to the *I2Cn_FIFO* register.

5. The I²C controller receives an ACK from the slave, and the controller sets the address ACK interrupt flag (*I2Cn_INT_FL0.addr_ack* = 1).
6. The *I2Cn_FIFO* register data bytes are transmitted on the SDA line.
   a. The I²C controller receives an ACK from the slave after each data byte
   b. As the transfer proceeds, the software should refill the transmit FIFO by writing to the *I2Cn_FIFO* register as needed.
   c. If the transmit FIFO goes empty during this process, the controller pauses at the beginning of the byte and waits for the software to either write more data or instruct the controller to send a RESTART or STOP condition
7. Once the software writes all the desired bytes to the *I2Cn_FIFO* register; the software should set either *I2Cn_MASTER_CTRL.restart* or *I2Cn_MASTER_CTRL.stop*.
8. Once the controller sends all the remaining bytes and empties the transmit FIFO, it sets *I2Cn_INT_FL0.done* and proceeds to send out either a RESTART condition, if *I2Cn_MASTER_CTRL.restart* was set, or a STOP condition if *I2Cn_MASTER_CTRL.stop* was set.

12.4.6.3 Multi-Master Operation

The I²C protocol supports multiple masters on the same bus. When the bus is free, two (or more) masters might try to initiate communication simultaneously. This is a valid bus condition. If this occurs and the two masters want to transmit different data or address different slaves, only one master can remain in master mode and complete its transaction. The other master must back off transmission and wait until the bus is idle. This process by which the winning master is determined is called bus arbitration.

For each address or data bit, the master compares the data being transmitted on SDA to the value observed on SDA to determine which master wins the arbitration. If a master attempts to transmit a 1 on SDA (that is, the master lets SDA float) but senses a 0 instead, then that master loses arbitration, and the other master that sent a zero continues with the transaction. The losing master cedes the bus by switching off its SDA and SCL drivers.

Note: This arbitration scheme works with any number of bus masters: if more than two masters begin transmitting simultaneously, the arbitration continues as each master cedes the bus until only one master remains transmitting. Data is not corrupted because as soon as each master realizes it has lost arbitration, it stops transmitting on SDA, leaving the following data bits sent on SDA intact.

If the I²C master peripheral detects it has lost arbitration, it stops generating SCL; sets *I2Cn_INT_FL0.arb_er*; sets *I2Cn_INT_FL0.tx_lock_out*, flushing any remaining data in the transmit FIFO; and clears *I2Cn_MASTER_CTRL.start*, *I2Cn_MASTER_CTRL.restart*, and *I2Cn_MASTER_CTRL.stop* to 0. So long as the peripheral is not addressed by the winning master, the I²C peripheral stays in master mode (*I2Cn_CTRL.mst = 1)*. If *at any time* another master addresses this peripheral using the address programmed in *I2Cn_SLAVE_ADDR.slave_addr*, then the I²C peripheral clears *I2Cn_CTRL.mst* to 0 and begins responding as a slave. This can even occur during the same address transmission during which the peripheral lost arbitration.

Note: Arbitration loss is considered an error condition, and like the other error conditions, sets I2Cn_INT_FL0.tx_lock_out. Therefore, after an arbitration loss, the software needs to clear I2Cn_INT_FL0.tx_lock_out and reload the transmit FIFO.

Also, in a multi-master environment, software does *not* need to wait for the bus to become free before attempting to start a transaction (writing 1 to *I2Cn_MASTER_CTRL.start*). If the bus is free when *I2Cn_MASTER_CTRL.start* is set to 1, the transaction begins immediately. If, instead, the bus is busy, then the peripheral:

1. Waits for the other master to complete the transaction(s) by sending a STOP,
2. counts out the bus free time using $t_{BUF} = t_{SCL\_LO}$ (see *Equation 12-2), and then*
3. sends a START condition and begins transmitting the slave address byte(s) in the transmit FIFO, followed by the rest of the transfer.

The I²C master peripheral is compliant with all bus arbitration and clock synchronization requirements of the I²C specification; this operation is automatic, and no additional programming is required.

12.4.7 Slave Mode Operation

When in slave mode, the I2Cn peripheral operates as a slave device on the I²C bus and responds to an external master's requests to transmit or receive data. To configure the I2Cn peripheral as a slave, write the *I2Cn_CTRL.mst* bit to zero. The master drives the I2Cn clock on the bus, so the SCL device pin is driven by the external master, and *I2Cn_STATUS.clk_mode* remains a zero. The desired slave address must be set by writing to the *I2Cn_SLAVE_ADDR.slave_addr* register.

For slave mode operation, the following registers should be configured with the I²C peripheral disabled:

- *I2Cn_CTRL.mst* – 0 for slave operation.
- I²C slave address:
  - *I2Cn_SLAVE_ADDR.slave_addr* must be set to the desired address for the device on the bus
  - *I2Cn_SLAVE_ADDR.ex_addr* should be set to 1 for 10-bit addressing or 0 for 7-bit addressing
- *I2Cn_CTRL.gen_call_addr*
- *I2Cn_CTRL.rx_mode*
  - The recommended value for this field is 0. Also, note that a setting of 1 is incompatible with slave mode operation with clock stretching disabled (*I2Cn_CTRL.scl_clk_stretch_dis* = 1).
- *I2Cn_CTRL.scl_clk_stretch_dis*
- *I2Cn_CTRL.hs_mode*
- *I2Cn_RX_CTRL0.dnr* – SMBus/PMBus applications should set this to 0, while other applications should set this to 1.
- *I2Cn_TX_CTRL0.tx_nack_afd*
- *I2Cn_TX_CTRL0.tx_amr_afd*
- *I2Cn_TX_CTRL0.tx_amw_afd*
- *I2Cn_TX_CTRL0.tx_amgc_afd*
- *I2Cn_TX_CTRL0.tx_preload*
  - The recommended value is 0 for applications that can tolerate slave clock stretching (*scl_clk_stretch_dis* = 0).
  - The recommended value is 1 for applications that do not allow slave clock stretching (*scl_clk_stretch_dis* = 1).
- *I2Cn_CLK_HI.scl_hi*
  - Applies to slave mode when clock stretching is enabled (*I2Cn_CTRL.scl_clk_stretch_dis* = 0)
  - This is used to satisfy $t_{SU;DAT}$ after clock stretching; program it so that the value defined by *Equation 12-1* is ≥ $t_{SU;DAT(min)}$
- *I2Cn_HS_CLK.hs_clk_hi*
  - Applies to slave mode when clock stretching is enabled (*I2Cn_CTRL.scl_clk_stretch_dis* = 0)
    - This is used to satisfy $t_{SU;DAT}$ after clock stretching during Hs-Mode operation; program it so that the value defined by *Equation 12-6* is ≥ $t_{SU;DAT(min)}$ for Hs-Mode
- I2Cn_SLAVE_ADDR.*slave_addr*
- *I2Cn_SLAVE_ADDR.ex_addr*

In contrast to the above registers, these registers can be safely (re)programmed at any time:

- All interrupt flags and interrupt enables
- *I2Cn_TX_CTRL0.tx_thresh* and *I2Cn_RX_CTRL0.rx_thresh*
  - Transmit and receive FIFO threshold levels
- *I2Cn_TX_CTRL1.tx_ready*
  - Transmit Ready (Can only be cleared by hardware)
- *I2Cn_TIMEOUT.to*
  - Timeout control
- *I2Cn_DMA.rxen*/*I2Cn_DMA.txen*
  - Transmit and receive DMA enables
- *I2Cn_FIFO.data*
  - FIFO access register

12.4.7.1 Slave Transmitter

The device operates as a slave transmitter when the received address matches the device slave address with the R/W bit set to 1. The master is then reading from the device slave. There are two primary modes of slave transmitter operation: just-in-time mode and preload mode.

12.4.7.1.1 Just-In-Time Slave Transmitter

In just-in-time mode, the software waits to write the transmit data to the transmit FIFO until after the master addresses it for a READ transaction, "just in time" for the data to be sent to the master. This allows the software to defer the determination of what data should be sent until the time of the address match. For example, the transmit data could be based on an immediately preceding I2C WRITE transaction that requests a certain block of data to be sent. The data could represent the latest, most up-to-date value of a sensor reading. Clock stretching *must* be enabled (*I2Cn_CTRL*.*scl_clk_stretch_dis* = 0) for just-in-time mode operation.

Program flow for transmit operation in just-in-time mode is as follows:

1. With *I2Cn_CTRL*.*i2c_en* = 0, initialize all relevant registers, including:
   a. Set the *I2Cn_SLAVE_ADDR*.*slave_addr* field with the desired I^2C slave address.
   b. Set the *I2Cn_SLAVE_ADDR*.*ex_addr* for either 7-bit or 10-bit addressing.
   c. Just-in-time specific settings:
      i) *I2Cn_CTRL*. *scl_clk_stretch_dis* = 0
      ii) *I2Cn_TX_CTRL0*[5:2] = 0x8
      iii) *I2Cn_TX_CTRL0*.*tx_preload* = 0.
   d. Program *I2Cn_CLK_HI*.*scl_hi* and *I2Cn_HS_CLK*.*hs_clk_hi* with appropriate values satisfying $t_{SU;DAT}$ (and HS $t_{SU;DAT}$).
2. Software sets *I2Cn_CTRL*.*i2c_en* = 1.
   a. The controller is now listening for its address. The peripheral responds to its address with an ACK for either a transmit (R/W = 1) or receive (R/W = 0) operation.
   b. When the address match occurs, hardware sets *I2Cn_INT_FL0*.*addr_match* and *I2Cn_INT_FL0*.*tx_lock_out*.
3. Software waits for *I2Cn_INT_FL0*.*addr_match* = 1, either through polling the interrupt flag or setting *I2Cn_INT_EN0*.*addr_match* to interrupt the CPU.
4. After reading *I2Cn_INT_FL0*.*addr_match* = 1, software reads *I2Cn_CTRL*.*read* to determine whether the transaction is a transmit (read = 1) or receive (read = 0) operation. In this case, we assume read = 1, indicating transmit.

a. At this point, hardware holds SCL low until software clears *I2Cn_INT_FL0*.*tx_lock_out* and loads data into the FIFO.

5. Software clears *I2Cn_INT_FL0*.*addr_match* and *I2Cn_INT_FL0*.*tx_lock_out*. Now that *I2Cn_INT_FL0*.*tx_lock_out* is 0, software can begin loading the transmit data into *I2Cn_FIFO*.
6. As soon as there is data in the FIFO, hardware releases SCL (after counting out *I2Cn_CLK_HI*.*scl_hi*) and sends out the data on the bus.
7. While the master keeps requesting data and sending ACKs, *I2Cn_INT_FL0*.*done* remains 0, and the software should continue to monitor the transmit FIFO and refill it as needed.
   a. The FIFO level can be monitored synchronously through the transmit FIFO interrupt flags or asynchronously by setting *I2Cn_TX_CTRL0*.*tx_thresh* and setting *I2Cn_INT_EN0*.*tx_thresh* interrupt.
   b. If the transmit FIFO ever empties during the transaction, the hardware starts clock stretching and waits for it to be refilled.
8. The master ends the transaction by sending a NACK. Once this happens, the *I2Cn_INT_FL0*.*done* interrupt flag is set, and software can stop monitoring the transmit FIFO.

9. The transaction is complete. The software should "clean up," including clearing *I2Cn_INT_FL0.done* and clearing *I2Cn_INT_EN0.tx_thresh* interrupt. Return to step 3, waiting on an address match.
10. If the software needs to know how many data bytes were transmitted to the master, check the transmit FIFO level as soon as the software sees *I2Cn_INT_FL0.done* = 1 and use that to determine how many data bytes were successfully sent.

    Note: Any data remaining in the transmit FIFO is discarded before the next transmit operation; it is NOT necessary for software to manually flush the transmit FIFO for this to occur.

12.4.7.1.2 Preload Mode Slave Transmitter

The other mode of operation for slave transmit is preload mode. In this mode, it is assumed that the software knows before the transmit operation what data it should send to the master. This data is then "preloaded" into the transmit FIFO. Once the address match occurs, this data can be sent out without any software intervention. Preload mode can be used with clock stretching either enabled or disabled, but it is the only option if clock stretching must be disabled.

To use slave transmit preload mode:

1. With *I2Cn_CTRL.i2c_en* = 0, initialize all relevant registers, including:
   a. Set the *I2Cn_SLAVE_ADDR.slave_addr* field with the desired I^2C slave address.
   b. Set the *I2Cn_SLAVE_ADDR.ex_addr* for either 7-bit or 10-bit addressing.
   c. Preload mode-specific settings:
      i) *I2Cn_CTRL*. *scl_clk_stretch_dis* = 1
      ii) *I2Cn_TX_CTRL0*[5:2] = 0xF
      iii) *I2Cn_TX_CTRL0.tx_preload* = 1.
2. Software sets *I2Cn_CTRL.i2c_en* = 1.
   a. Even though the controller is enabled, at this point, it does not ACK an address match with R/W = 1 until software sets *I2Cn_TX_CTRL1.tx_ready* = 1.
3. Software prepares for the transmit operation by loading data into the transmit FIFO, enabling DMA, setting *I2Cn_TX_CTRL0.tx_thresh* and setting *I2Cn_INT_EN0.tx_thresh* interrupt, etc.
   a. If clock stretching is disabled, then an empty transmit FIFO during the transmit operation causes a transmit underrun error. Therefore, the software should take any necessary steps to avoid an underrun *before* setting *I2Cn_TX_CTRL1.tx_ready* = 1.
   b. If clock stretching is enabled, then an empty transmit FIFO does not cause a transmit underrun error. However, it is recommended to follow the same preparation steps to minimize the amount of time spent on clock stretching, letting the transaction complete as quickly as possible.
4. Once software is prepared for the transmit operation; the software sets *I2Cn_TX_CTRL1.tx_ready* = 1.
   a. The controller is now fully enabled and responds with an ACK to an address match.
   b. Hardware sets *I2Cn_INT_FL0.addr_match* once an address match has occurred. *I2Cn_INT_FL0.tx_lock_out*. is NOT set and remains 0.
5. Software waits for *I2Cn_INT_FL0.addr_match* = 1, either through polling the interrupt flag or setting *I2Cn_INT_EN0.addr_match* to interrupt the CPU.
6. After seeing *I2Cn_INT_FL0.addr_match* = 1, software reads *I2Cn_CTRL.read* to determine whether the transaction is a transmit (read = 1) or receive (read = 0) operation. In this case, assume *I2Cn_CTRL.read*, indicating transmit.
   a. At this point, the hardware begins sending out the data that was preloaded into the transmit FIFO.
   b. Once the first data byte is sent, the hardware automatically clears *I2Cn_TX_CTRL1.tx_ready* to 0.

7. While the master keeps requesting data and sending ACKs, *I2Cn_INT_FL0.done* remains 0, and the software should continue to monitor the transmit FIFO and refill it as needed.
    a. The FIFO level can be monitored synchronously through the transmit FIFO status/interrupt flags or asynchronously by setting *I2Cn_TX_CTRL0.tx_thresh* and setting *I2Cn_INT_EN0.tx_thresh* interrupt.
    b. If clock stretching is disabled and the transmit FIFO ever empties during the transaction, the hardware sets *I2Cn_INT_FL1.tx_underflow* = 1 and sends 0xFF for all following data bytes requested by the master.
8. The master ends the transaction by sending a NACK. Once this happens, the *I2Cn_INT_FL0.done* interrupt flag is set.
    a. If the transmit FIFO goes empty at the same time that the master indicates the transaction is complete by sending a NACK, this is not considered an underrun event, and the *I2Cn_INT_FL1.tx_underflow* flag remains 0.
9. The transaction is complete. The software should "clean up," which includes clearing *I2Cn_INT_FL0.done*. Return to step 3 and prepare for the next transaction.
    a. If software needs to know how many data bytes were transmitted to the master, it should check the transmit FIFO level as soon as software sees *I2Cn_INT_FL0.done* = 1 and use that to determine how many data bytes were successfully sent.
    b. By default, any data remaining in the transmit FIFO is NOT discarded, and instead, it is reused for the next transmit operation.
    c. If this is not desired, software can flush the transmit FIFO. The safest way to do this is by clearing and then setting *I2Cn_CTRL.i2c_en*. This flushes both the transmit and receive FIFOs.

Once a slave starts transmitting out of its *I2Cn_FIFO*, detection of out of sequence STOP, START, or RESTART condition terminates the current transaction. When a transaction is terminated in such a manner, *I2Cn_INT_FL0.start_er* or *I2Cn_INT_FL0.stop_er* is set to 1.

If the transmit FIFO is not ready (*I2Cn_TX_CTRL1.tx_ready* = 0) and the I^2C controller receives a data read request from the master, the hardware automatically sends a NACK at the end of the first address byte. The hardware ignores the setting of the do not respond field in this case because the only opportunity to send a NACK for an I^2C read transaction is after the address byte.

12.4.7.2 Slave Receiver

The device operates as a slave receiver when the received address matches the device slave address with the R/W bit set to 0. The external master is writing to the slave.

Program flow for a receive operation is as follows:

1. With *I2Cn_CTRL.i2c_en* = 0, initialize all relevant registers, including:
    a. Set the *I2Cn_SLAVE_ADDR.slave_addr* field with the desired I^2C slave address.
    b. Set the *I2Cn_SLAVE_ADDR.ex_addr* for either 7-bit or 10-bit addressing.
2. Set *I2Cn_CTRL.i2c_en* = 1.
    a. If an address match with R/W = 0 occurs, and the receive FIFO is empty, the peripheral responds with an ACK, and the *I2Cn_INT_FL0.addr_match* flag is set.
    b. If the receive FIFO is not empty, then depending on the value of *I2Cn_RX_CTRL0.dnr*, the peripheral NACKs either the address byte (*I2Cn_RX_CTRL0.dnr* = 1) or the first data byte (*I2Cn_RX_CTRL0.dnr* = 0).
3. Wait for *I2Cn_INT_FL0.addr_match* = 1, either by polling or by enabling the *wr_addr_match* interrupt. Once a successful address match occurs, hardware sets *I2Cn_INT_FL0.addr_match* = 1.
4. Read *I2Cn_CTRL.read* to determine whether the transaction is a transmit (*I2Cn_CTRL.read* = 1) or receive (*I2Cn_CTRL.read* = 0) operation. In this case we assume *I2Cn_CTRL.read* = 0, indicating receive. At this point, the device begins receiving data into the receive FIFO.

5. Clear *I2Cn_INT_FL0.addr_match*, and while the master keeps sending data, *I2Cn_INT_FL0.done* remains 0, and software should continue to monitor the receive FIFO and empty it as needed.
   a. The FIFO level can be monitored synchronously through the receive FIFO status/interrupt flags or asynchronously by setting *I2Cn_RX_CTRL0.rx_thresh* and enabling the *I2Cn_INT_FL0.rx_thresh* interrupt.
   b. If the receive FIFO ever fills up during the transaction, then hardware sets *I2Cn_INT_FL1.rx_overflow* and then either:
      1) if *I2Cn_CTRL.scl_clk_stretch_dis* = 0, start clock stretching and wait for software to read from the receive FIFO,

      or

      2) if *I2Cn_CTRL.scl_clk_stretch_dis* = 1, respond to the master with a NACK and the last byte is discarded.
6. The master ends the transaction by sending a RESTART or STOP. Once this happens, the *I2Cn_INT_FL0.done* interrupt flag is set, and software can stop monitoring the receive FIFO.
7. Once a slave starts receiving into its RX_FIFO, detection of out of sequence STOP, START or RESTART condition releases the I^2C bus to the Idle state, and hardware sets *I2Cn_INT_FL0.start_er* or *I2Cn_INT_FL0.stop_er* to 1.

If software has not emptied the data in the receive FIFO from the previous transaction by the time that a master addresses it for another write (i.e., receive) transaction, then the controller does *not* participate in the transaction, and no additional data is written into the FIFO. Although a NACK *is* sent to the master, software can control whether the NACK is sent with the initial address match or if instead it is sent at the end of the first data byte. Setting *I2Cn_RX_CTRL0.dnr* to 1 chooses the former while setting *I2Cn_RX_CTRL0.dnr* to 0 chooses the latter.

12.4.8 Interrupt Sources

The I^2C controller has a very flexible interrupt generator that generates an interrupt signal to the Interrupt Controller on any of several events. On recognizing the I^2C interrupt, the software determines the cause of the interrupt by reading the I^2C Interrupt Flags registers *I2Cn_INT_FL0* and *I2Cn_INT_FL1*. Interrupts can be generated for the following events:

- Transaction complete (master/slave)
- Address NACK received from slave (master)
- Data NACK received from slave (master)
- Lost arbitration (master)
- Transaction timeout (master/slave)
- FIFO is empty, not empty, full to a configurable threshold level (master/slave)
- Transmit FIFO locked out because it is being flushed (master/slave)
- Out of sequence START and STOP conditions (master/slave)
- Sent a NACK to an external master because the transmit or receive FIFO was not ready (slave)
- Address ACK or NACK received (master)
- Incoming address match (slave)
- Transmit underflow or receive overflow (slave)

Interrupts for each event can be enabled or disabled by setting or clearing the corresponding bit in the *I2Cn_INT_EN0* or *I2Cn_INT_EN1* interrupt enable register.

Note: Disabling the interrupt does not prevent the corresponding flag from being set by hardware but does prevent an IRQ when the interrupt flag is set.

Note: Before enabling an interrupt, the status of the corresponding interrupt flag should be checked and, if necessary, serviced or cleared. This prevents a previous interrupt event from interfering with a new I^2C communications session.

12.4.9 Transmit FIFO and Receive FIFO

There are separate transmit and receive FIFOs. Both are accessed using the FIFO data register *I2Cn_FIFO*. Writes to this register enqueue data into the transmit FIFO. Writes to a full transmit FIFO has no effect. Reads from *I2Cn_FIFO* de queue data from the receive FIFO. Writes to a full transmit FIFO has no effect and reads from an empty receive FIFO return 0xFF.

The transmit and receive FIFO only read or write one byte at a time. Transactions larger than 8 bits can still be performed, however. A 16- or 32-bit write to the transmit FIFO stores just the lowest 8 bits of the write data. A 16- or 32-bit read from the receive FIFO has the valid data in the lowest 8 bits and 0's in the upper bits. In any case, the transmit and receive FIFOs only accept 8 bits at a time for either read or write.

To offload work from the CPU, the DMA can read and write to each FIFO. See section *DMA Control* for more information on configuring the DMA.

During a receive transaction (which during master operation is a READ, and during slave operation is a WRITE), received bytes are automatically written to the receive FIFO. Software should monitor the receive FIFO level and unload data from it as needed by reading *I2Cn_FIFO*. If the receive FIFO becomes full during a master mode transaction, then the controller sets the *I2Cn_INT_FL1.rx_overflow* bit, and one of two things happen depending on the value of *I2Cn_CTRL.scl_clk_stretch_dis*:

- If clock stretching is enabled (*I2Cn_CTRL.scl_clk_stretch_dis* = 0), then the controller stretches the clock until software makes space available in the receive FIFO by reading from *I2Cn_FIFO*. Once space is available, the peripheral moves the data byte from the shift register into the receive FIFO, the SCL device pin is released, and the master is free to continue the transaction.
- If clock stretching is disabled (*I2Cn_CTRL.scl_clk_stretch_dis* = 1), then the controller responds to the master with a NACK and the data byte is lost. The master can return the bus to idle with a STOP condition or start a new transaction with a RESTART condition.

During a transmit transaction (which during master operation is a WRITE, and during slave operation is a READ), either software or the DMA can provide data to be transmitted by writing to the transmit FIFO. Once the peripheral finishes transmitting each byte, it removes it from the transmit FIFO and, if available, begins transmitting the next byte.

Interrupts can be generated for the following FIFO status:

- Transmit FIFO level less than or equal to the threshold
- Receive FIFO level greater than or equal to the threshold
- Transmit FIFO underflow
- Receive FIFO overflow
- Transmit FIFO locked for writing

The receive and transmit FIFO are flushed when the I^2C port is disabled by clearing *I2Cn_CTRL.i2c_en* = 0. While the peripheral is disabled, writes to the transmit FIFO has no effect and reads from the receive FIFO return 0xFF.

The transmit FIFO and receive FIFO can be flushed by setting the transmit FIFO flush bit (*I2Cn_TX_CTRL0.tx_flush* = 1) or the receive FIFO flush bit (*I2Cn_RX_CTRL0.rx_flush* = 1), respectively. In addition, under certain conditions, the transmit FIFO is automatically locked by hardware and flushed so stale data is not unintentionally transmitted. The transmit FIFO is automatically flushed and writes locked out from software under the following conditions:

- General Call Address Match. Automatic flushing and lockout can be disabled by setting *I2Cn_TX_CTRL0.tx_amgc_afd*.
- Slave Address Match Write. Automatic flushing and lockout can be disabled by setting *I2Cn_TX_CTRL0.tx_amw_afd*.
- Slave Address Match Read. Automatic flushing and lockout can be disabled by setting *I2Cn_TX_CTRL0.tx_amr_afd*.
- During operation as a slave transmitter, a NACK is received. Automatic flushing and lockout can be disabled by setting *I2Cn_TX_CTRL0.tx_nack_afd*.
- Any of the following interrupts: arbitration error, timeout error, master mode address NACK error, master mode data NACK error, start error, and stop error. Automatic flushing cannot be disabled for these conditions.

When the above conditions occur, the transmit FIFO is flushed so that data intended for a previous transaction is not unintentionally transmitted for a new transaction. In addition to flushing the transmit FIFO, the Transmit Lockout Flag is set (*I2Cn_INT_FL0.tx_lock_out* = 1) and writes to the transmit FIFO are ignored until the software acknowledges the external event by clearing *I2Cn_INT_FL0.tx_lock_out*.

12.4.10 Transmit FIFO Preloading

There may be situations during slave mode operation where software wants to preload the transmit FIFO before a transmission, such as when clock stretching is disabled. In this scenario, rather than responding to an external master

requesting data with an ACK and clock stretching while software writes the data to the transmit FIFO, the controller instead responds with a NACK until software has preloaded the requested data into the transmit FIFO.

When transmit FIFO preloading is enabled, the software controls ACKs to the external master using the transmit ready (*I2Cn_TX_CTRL1*.*tx_ready*) bit. When *I2Cn_TX_CTRL1*.*tx_ready* is set to 0, hardware automatically NACKs all read transactions from the master. Setting *I2Cn_TX_CTRL1*.*tx_ready* to 1 sends an ACK to the master on the next read transaction and transmits the data in the transmit FIFO. Preloading the transmit FIFO must be complete before setting the *I2Cn_TX_CTRL1*.*tx_ready* field to 1.

The required steps for implementing transmit FIFO Preloading in an application are as follow:

1. Enable transmit FIFO preloading by setting *I2Cn_TX_CTRL0*.*tx_preload* to 1.
   a. This automatically clears *I2Cn_TX_CTRL1*.*tx_ready* to 0
2. If the transmit FIFO lockout flag (*I2Cn_INT_FL0*.*tx_lock_out*) is set to 1, write 1 to clear the flag and enable writes to the transmit FIFO.
3. Enable DMA or Interrupts if required.
4. Load the transmit FIFO with the data to send when the master sends the next read request.
5. Set *I2Cn_TX_CTRL1*.*tx_ready* to 1 to automatically let the hardware send the preloaded FIFO on the next read from a master.
6. *I2Cn_TX_CTRL1*.*tx_ready* is cleared by hardware once it finishes transmitting the first byte, and data is transmitted from the transmit FIFO. Once cleared, the software may repeat the preloading process or disable transmit FIFO preloading.

Note: To prevent the preloaded data from being cleared when the master tries to read it, the software must set I2Cn_TX_CTRL0.tx_amr_afd to 1, disabling auto flush on a READ address match. The software determines whether the other auto flush disable bits should be set. For example, if a master uses I²C WRITE transactions to determine what data the slave should send in the following READ transactions, then software can clear I2Cn_TX_CTRL0.tx_amw_afd to 0. Then when a WRITE occurs, the transmit FIFO is flushed, giving the software time to load the new data. For the READ transaction, the external master can poll the slave address until the new data has been loaded and I2Cn_TX_CTRL1.tx_ready is set, at which point the peripheral responds with an ACK.

12.4.11 Interactive Receive Mode (IRXM)

In some situations, the I²C might want to inspect and respond to each byte of received data. In this case, IRXM can be used. IRXM is enabled by setting *I2Cn_CTRL*.*rx_mode* = 1. If IRXM is enabled, it must occur before any I²C transfer is initiated.

When IRXM is enabled, for every data byte received the I²C peripheral automatically holds SCL low before the ACK bit. Additionally, after the 8th SCL falling edge, the I²C peripheral sets the IRXM interrupt status flag (*I2Cn_INT_FL0*.*rx_mode* = 1). Software must read the data and generate a response (ACK or NACK) by setting the IRXM acknowledge (*I2Cn_CTRL*.*rx_mode_ack*) bit accordingly. Send an ACK by clearing the *I2Cn_CTRL*.*rx_mode_ack* bit to 0. Send a NACK by setting the *I2Cn_CTRL*.*rx_mode_ack* bit to 1.

After setting the *I2Cn_CTRL*.*rx_mode_ack* bit, clear the IRXM interrupt flag. Write 1 to *I2Cn_INT_FL0*.*rx_mode* to clear the interrupt flag. When the IRXM interrupt flag is cleared, the I²C peripheral hardware releases the SCL line and sends the *I2Cn_CTRL*.*rx_mode_ack* on the SDA line.

While the I²C peripheral is waiting for the software to clear the *I2Cn_INT_FL0*.*rx_mode* flag, software can disable IRXM and, if operating as a master, load the remaining number of bytes to be received for the transaction. This allows the software to examine the initial bytes of a transaction, which might be a command, and then disable IRXM to receive the remaining bytes in normal operation.

During IRXM, received data is not placed in the receive FIFO. Instead, the *I2Cn_FIFO* address is repurposed to directly read the receive shift register, bypassing the receive FIFO. Therefore, before disabling IRXM, the software must first read the data byte from *I2Cn_FIFO.data.* If the IRXM byte is not read, the byte is lost, and the next read from the receive FIFO returns 0xFF.

Note: IRXM does not apply to address bytes. IRXM only applies to data bytes.

Note: IRXM does not apply to general call address responses or START byte responses.

Note: When enabling IRXM and operating as a slave, clock stretching must remain enabled (I2Cn_CTRL.scl_clk_stretch_dis = 0).

12.4.12 Clock Stretching

When the I²C peripheral requires some response or intervention from the software to continue with a transaction, it holds SCL low, preventing the transfer from continuing. This is called 'clock stretching' or 'stretching the clock.' While the I²C Bus Specification defines the term 'clock stretching' to only apply to a slave device holding the SCL line low, this section describes situations where the I²C peripheral holds the SCL line low in either slave *or* master mode and refers to *both* as clock stretching.

When the I²C peripheral stretches the clock, it typically does so in response to either a full receive FIFO during a receive operation or an empty transmit FIFO during a transmit operation. Necessarily, this occurs before the next data byte begins, either between the ACK bit and the first data bit or, if at the beginning of a transaction, immediately after a START or RESTART condition. However, when operating in IRXM (*I2Cn_CTRL.rx_mode* = 1), the peripheral can also clock stretch *before* the ACK bit, allowing the software to decide whether to send an ACK or NACK.

For a transmit operation (as either master or slave), when the transmit FIFO is empty, SCL is automatically held low after the ACK bit and before the next data byte begins. The software must write data to the *I2Cn_FIFO.data* register to stop clock stretching and continue the transaction. However, if operating in master mode, instead of sending more data, the software may also set either *I2Cn_MASTER_CTRL.stop* or *I2Cn_MASTER_CTRL.restart* to send a STOP or RESTART condition, respectively.

For a receive operation (as either master or slave), when both the receive FIFO and the receive shift register are full, SCL is automatically held low until at least one data byte is read from the receive FIFO. The software must read data from the *I2Cn_FIFO.data* register to stop clock stretching and continue the transaction. If operating in master mode and this is the final byte of the transaction, as determined by *I2Cn_RX_CTRL1.rx_cnt*, then software must also set either *I2Cn_MASTER_CTRL.stop* or *I2Cn_MASTER_CTRL.restart* to send a STOP or RESTART condition, respectively. This must be done in addition to reading from the receive FIFO since the peripheral cannot start sending the STOP or RESTART until the last data byte has been moved from the receive shift register into the receive FIFO. (This automatically occurs once there is space in the receive FIFO.)

Note: Since some masters do not support other devices stretching the clock, it is possible to completely disable all clock stretching during slave mode by setting I2Cn_CTRL.scl_clk_stretch_dis to 1 and clearing I2Cn_CTRL.rx_mode to 0. In this case, instead of clock stretching, the I²C peripheral sends a NACK if receiving data or sends 0xFF if transmitting data.

Note: The clock synchronization required to support other I²C master or slave devices stretching the clock is built into the peripheral and requires no intervention from software to operate correctly.

12.4.13 Bus Timeout

The Timeout register, *I2Cn_TIMEOUT.to,* is used to detect bus errors. *Equation 12-8* and *Equation 12-9* show equations for calculating the maximum and minimum timeout values based on the value loaded into the *I2Cn_TIMEOUT.to* field.

Equation 12-8: I²C Timeout Maximum

$$t_{TIMEOUT} \leq \left(\frac{1}{f_{I2C\_CLK}}\right) \times \left((\text{I2Cn\_TIMEOUT.to} \times 32) + 3\right)$$

Due to clock synchronization, the timeout is guaranteed to meet the following minimum time calculation shown in *Equation 12-9*.

Equation 12-9: I²C Timeout Minimum

$$t_{TIMEOUT} \leq \left(\frac{1}{f_{I2C\_CLK}}\right) \times \left((I2Cn\_TIMEOUT.\text{to} \times 32) + 2\right)$$

The timeout feature is disabled when *I2Cn_TIMEOUT.to* = 0 and is enabled for any non-zero value. When the timeout is enabled, the timeout timer starts counting when the I²C peripheral hardware drives SCL low and is reset by the I²C peripheral hardware when the SCL line is released.

The timeout counter only monitors if the I²C peripheral hardware is driving the SCL line low. It does not monitor if an external I²C device is actively holding the SCL line low. The timeout counter also does not monitor the status of the SDA line.

If the timeout timer expires, a bus error condition has occurred. When a timeout error occurs, the I²C peripheral hardware releases the SCL and SDA lines and sets the timeout error interrupt flag to 1 (*I2Cn_INT_FL0.to_er* = 1).

For applications where the device may hold the SCL line low longer than the maximum timeout supported, the timeout can be disabled by setting the timeout field to 0 (*I2Cn_TIMEOUT.to* = 0).

12.4.14 DMA Control

There are independent DMA channels for the transmit FIFO and the receive FIFO. DMA activity is triggered by the transmit FIFO (*I2Cn_TX_CTRL0.tx_thresh*) and receive FIFO (*I2Cn_RX_CTRL0.rx_thresh*) threshold levels.

When the transmit FIFO byte count (*I2Cn_TX_CTRL1.txfifo*) is less than or equal to the transmit FIFO Threshold Level *I2Cn_TX_CTRL0.tx_thresh*, then the DMA transfers data into the transmit FIFO according to the DMA configuration.

Set the DMA burst size as shown in *Equation 12-10* to ensure the DMA does not overflow the transmit FIFO.

Equation 12-10: DMA Burst Size Calculation for I²C Transmit

$$DMA\ Burst\ Size \leq TX\ FIFO\ Depth - I2Cn\_TX\_CTRL0.tx\_thresh = 8 - I2Cn\_TX\_CTRL0.tx\_thresh$$

$$where\ 0 \leq I2Cn\_TX\_CTRL0.tx\_thresh \leq 7$$

Applications trying to avoid transmit underflow and clock stretching should use a smaller burst size and higher *I2Cn_TX_CTRL0.tx_thresh* setting. This fills up the FIFO more frequently but increases internal bus traffic.

When the receive FIFO count (*I2Cn_RX_CTRL1.rx_fifo*) is greater than or equal to the receive FIFO Threshold Level *I2Cn_RX_CTRL0.rx_thresh*, the DMA transfers data out of the receive FIFO according to the DMA configuration. Set the DMA burst size as shown in *Equation 12-11* to ensure the DMA does not underflow the receive FIFO.

Equation 12-11: DMA Burst Size Calculation for I²C Receive

$$DMA\ Burst\ Size \leq \mathrm{I2Cn\_RX\_CTRL0}.rx\_thresh$$

$$\text{where } 1 \leq \mathrm{I2Cn\_RX\_CTRL0}.rx\_thresh \leq 8$$

Applications trying to avoid receive overflow and clock stretching should use a smaller burst size and lower *I2Cn_RX_CTRL0.rx_thresh*. This results in reading from the Receive FIFO more frequently but increases internal bus traffic.

Note for receive operations, the length of the DMA transaction (in bytes) must be an integer multiple of I2Cn_RX_CTRL0.rx_thresh. Otherwise, the receive transaction ends with some data still in the receive FIFO, but not enough to trigger an interrupt to the DMA, leaving the DMA transaction incomplete. One easy way to ensure this for all transaction lengths is to set burst size to 1 (I2Cn_RX_CTRL0.rx_thresh = 1).

To enable DMA transfers, enable the transmit DMA channel (*I2Cn_DMA.txen*) and/or the receive DMA channel (*I2Cn_DMA.rxen*).

12.5 Registers

See *Table 3-3* for the base address of this peripheral/module. Each instance has its own independent set of the registers shown in *Table 12-5*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 12-5: I²C Registers

Offset	Register	Description
[0x0000]	*I2Cn_CTRL*	I²C Control Register
[0x0004]	*I2Cn_STATUS*	I²C Status Register
[0x0008]	*I2Cn_INT_FL0*	I²C Interrupt Flags 0 Register
[0x000C]	*I2Cn_INT_EN0*	I²C Interrupt Enable 0 Register
[0x0010]	*I2Cn_INT_FL1*	I²C Interrupt Flags 1 Register
[0x0014]	*I2Cn_INT_EN1*	I²C Interrupt Enable 1 Register
[0x0018]	*I2Cn_FIFO_LEN*	I²C FIFO Length Register
[0x001C]	*I2Cn_RX_CTRL0*	I²C Receive Control 0 Register
[0x0020]	*I2Cn_RX_CTRL1*	I²C Receive Control 1 Register
[0x0024]	*I2Cn_TX_CTRL0*	I²C Transmit Control 0 Register
[0x0028]	*I2Cn_TX_CTRL1*	I²C Transmit Control 1 Register
[0x002C]	*I2Cn_FIFO*	I²C Transmit and Receive FIFO Register
[0x0030]	*I2Cn_MASTER_CTRL*	I²C Master Mode Register
[0x0034]	*I2Cn_CLK_LO*	I²C Clock Low Time Register
[0x0038]	*I2Cn_CLK_HI*	I²C Clock High Time Register
(0x003C)	*I2Cn_HS_CLK*	I²C Hs-Mode Clock Control Register
[0x0040]	*I2Cn_TIMEOUT*	I²C Timeout Register
[0x0048]	*I2Cn_DMA*	I²C DMA Enable Register
[0x004C]	*I2Cn_SLAVE_ADDR*	I²C Slave Address Register

12.6 Register Details

Table 12-6: I²C Control Register

I²C Control			**I2Cn_CTRL**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	hs_mode	R/W	0	**Hs-Mode Enable** I²C high speed mode operation 0: Disable 1: Enable
14	-	RO	0	**Reserved**
13	scl_pp_mode	R/W	0	**Single Master Only** When set to 1, the device MUST ONLY be used in a single master application with slave devices that are NOT going to hold SCL low (i.e., the slave devices can never clock stretch)
12	scl_clk_stretch_dis	R/W	0	**Slave Mode Clock Stretching** 0: Enabled 1: Disabled
11	read	R	0	**Slave Read/Write Bit Status** Returns the logic level of the R/W bit on a received address match (*I2Cn_INT_FL0.addr_match* = 1) or general call match (*I2Cn_INT_FL0.gen_call_addr* = 1). This bit is valid three system clock cycles after the address match status flag is set.

I²C Control			I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
10	sw_out_en	R/W	0	**Software Output Control Enabled** Setting this field to 1 enables software bit-bang control of I²C. 0: The I²C controller manages the SDA and SCL pins in hardware. 1: SDA and SCL are controlled by the software using the *I2Cn_CTRL.sda_out* and *I2Cn_CTRL.scl_out* fields.
9	sda	R	-	**SDA Status** 0: SDA pin is logic low. 1: SDA pin is logic high.
8	scl	R	-	**SCL Status** 0: SCL pin is logic low. 1: SCL pin is logic high.
7	sda_out	R/W	0	**SDA Pin Output Control** Set the state of the SDA hardware pin (actively pull low or float). 0: Pull SDA Low 1: Release SDA *Note: Only valid when I2Cn_CTRL.sw_out_en = 1*
6	scl_out	R/W	0	**SCL Pin Output Control** Set the state of the SCL hardware pin (actively pull low or float). 0: Pull SCL low 1: Release SCL *Note: Only valid when I2Cn_CTRL.sw_out_en = 1*
5	-	RO	0	**Reserved**
4	rx_mode_ack	R/W	0	**IRXM Acknowledge** If IRXM is enabled (*I2Cn_CTRL.rx_mode* = 1), this field determines if the hardware sends an ACK or a NACK to an IRXM transaction. 0: Respond to IRXM with ACK 1: Respond to IRXM with NACK
3	rx_mode	R/W	0	**IRXM Enable** When receiving data, this field allows for an IRXM interrupt event after each received byte of data. The I²C peripheral hardware can be enabled to send either an ACK or NACK for IRXM. See the *Interactive Receive Mode* section for detailed information. 0: Disable 1: Enable *Note: Only set this field when the I²C bus is inactive.*
2	gen_call_addr	R/W	0	**General Call Address Enable** 0: Ignore General Call Address 1: Acknowledge General Call Address
1	mst	R/W	0	**Master Mode Enable** 0: Slave mode enabled. 1: Master mode enabled.
0	i2c_en	R/W	0	**I²C Peripheral Enable** 0: Disabled 1: Enabled

Table 12-7: I²C Status Register

I²C Status			I2Cn_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**
5	clk_mode	RO	0	**Master Mode I²C Bus Transaction Active** The peripheral is operating in master mode, and a valid transaction beginning with a START command is in progress on the I²C bus. This bit reads 1 until the master ends the transaction with a STOP command. This bit continues to read 1 while a slave performs clock stretching. 0: Device not actively driving SCL clock cycles. 1: Device operating as master and actively driving SCL clock cycles.
4	tx_full	RO	0	**Transmit FIFO Full** 0: Not full 1: Full
3	tx_empty	RO	1	**Transmit FIFO Empty** 0: Not empty 1: Empty
2	rx_full	RO	0	**Receive FIFO Full** 0: Not full 1: Full
1	rx_empty	RO	1	**Receive FIFO Empty** 0: Not empty 1: Empty
0	bus	RO	0	**Master or Slave Mode I²C Bus Transaction Active** The peripheral is operating in master or slave mode, and a valid transaction beginning with a START command is in progress on the I²C bus. This bit reads 1 until the peripheral acting as a master or an external master ends the transaction with a STOP command. This bit continues to read 1 while a slave performs clock stretching. 0: I²C bus is idle. 1: I²C bus transaction in progress.

Table 12-8: I²C Interrupt Flag 0 Register

I²C Interrupt Flag 0			I2Cn_INT_FL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23	wr_addr_match	R/W1C	0	**Slave Write Address Match Interrupt Flag** If set, the device has been accessed for a write (i.e., receive) transaction in slave mode, and the address received matches the device slave address. 0: No address match. 1: Address match.
22	rd_addr_match	R/W1C	0	**Slave Read Address Match Interrupt Flag** If set, the device has been accessed for a read (i.e., transmit) transaction in slave mode, and the address received matches the device slave address. 0: No address match. 1: Address match.
21:16	-	RO	0	**Reserved**

I²C Interrupt Flag 0			I2Cn_INT_FL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
15	tx_lock_out	R/W1C	0	**Transmit FIFO Locked Interrupt Flag** If set, the transmit FIFO is locked, and writes to the transmit FIFO are ignored. When set, the transmit FIFO is automatically flushed. Writes to the transmit FIFO are ignored until this flag is cleared. Write 1 to clear. 0: Transmit FIFO not locked. 1: Transmit FIFO is locked, and all writes to the transmit FIFO are ignored.
14	stop_er	R/W1C	0	**Out of Sequence STOP Interrupt Flag** This flag is set if a STOP condition occurs out of its expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence STOP condition occurred.
13	start_er	R/W1C	0	**Out of Sequence START Interrupt Flag** This flag is set if a START condition occurs out of its expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence START condition occurred.
12	do_not_resp_er	R/W1C	0	**Slave Mode Do Not Respond Interrupt Flag** This flag is set if an address match is made, but the transmit FIFO or receive FIFO is not ready. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: I²C address match has occurred, and either the transmit or receive FIFO is not configured.
11	data_er	R/W1C	0	**Master Mode Data NACK from External Slave Interrupt Flag** This flag is set by hardware if a NACK is received from a slave. This flag is only valid if the I2Cn peripheral is configured for master mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Data NACK received from a slave.
10	addr_nack_er	R/W1C	0	**Master Mode Address NACK from Slave Error Flag** This flag is set by hardware if an Address NACK is received from a slave bus. This flag is only valid if the I2Cn peripheral is configured for master mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Address NACK received from a slave.
9	to_er	R/ W1C	0	**Timeout Error Interrupt Flag** This occurs when this device holds SCL low longer than the programmed timeout value. This field applies to both master and slave mode. Write 1 to clear. Write 0 has no effect. 0: Timeout error has not occurred. 1: Timeout error occurred.
8	arb_er	R/ W1C	0	**Master Mode Arbitration Lost Interrupt Flag** Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: Condition occurred.

I²C Interrupt Flag 0			I2Cn_INT_FL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
7	addr_ack	R/ W1C	0	**Master Mode Address ACK from External Slave Interrupt Flag** This field is set when a slave address ACK is received. Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: The slave device ACK for the address was received.
6	stop	R/ W1C	0	**Slave Mode STOP Condition Interrupt Flag** This flag is set by hardware when a STOP condition is detected. Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: Condition occurred.
5	tx_thresh	RO	1	**Transmit FIFO Threshold Level Interrupt Flag** This field is set by hardware if the number of bytes in the transmit FIFO is less than or equal to the transmit FIFO threshold level. Write 1 to clear. This field is automatically cleared by hardware when the transmit FIFO contains fewer bytes than the transmit threshold level. 0: Transmit FIFO contains more bytes than the transmit threshold level. 1: Transmit FIFO contains transmit threshold level or fewer bytes (Default).
4	rx_thresh	R/W1C	1	**Receive FIFO Threshold Level Interrupt Flag** This field is set by hardware if the number of bytes in the Receive FIFO is greater than or equal to the Receive FIFO threshold level. This field is automatically cleared when the receive FIFO contains fewer bytes than the receive threshold setting. 0: Receive FIFO contains fewer bytes than the receive threshold level. 1: Receive FIFO contains at least receive threshold level of bytes (Default).
3	addr_match	R/W1C	0	**Slave Mode Incoming Address Match Status Interrupt Flag** Write 1 to clear. Writing 0 has no effect. 0: Slave address match has not occurred. 1: Slave address match occurred.
2	gen_call_addr	R/W1C	0	**Slave Mode General Call Address Match Received Interrupt Flag** Write 1 to clear. Writing 0 has no effect. 0: General call address match has not occurred. 1: General call address match occurred.
1	rx_mode	R/W1C	0	**IRXM Interrupt Flag** Write 1 to clear. Writing 0 is ignored. 0: Interrupt condition has not occurred. 1: Interrupt condition occurred.
0	done	R/W1C	0	**Transfer Complete Interrupt Flag** This flag is set for both master and slave mode once a transaction completes. Write 1 to clear. Writing 0 has no effect. 0: Transfer is not complete. 1: Transfer complete.

Table 12-9: I²C Interrupt Enable 0 Register

I²C Interrupt Enable 0			I2Cn_INT_EN0	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**

I²C Interrupt Enable 0			I2Cn_INT_EN0	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
23	wr_addr_match	R/W	0	**Slave Write Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in slave mode, and the address received matches the device slave addressed for a write transaction. 0: Disabled. 1: Enabled.
22	rd_addr_match	R/W	0	**Slave Read Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in slave mode, and the address received matches the device slave addressed for a read transaction. 0: Disabled. 1: Enabled.
21:16	-	RO	0	**Reserved**
15	tx_lock_out	R/W	0	**Transmit FIFO Lock Out Interrupt Enable** 0: Disabled. 1: Enabled.
14	stop_er	R/W	0	**Out of Sequence STOP Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
13	start_er	R/W	0	**Out of Sequence START Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
12	do_not_resp_er	R/W	0	**Slave Mode Do Not Respond Interrupt Enable** Set this field to enable interrupts in slave mode when the "Do Not Respond" condition occurs. 0: Interrupt disabled. 1: Interrupt enabled.
11	data_er	R/W	0	**Master Mode Received Data NACK from Slave Interrupt Enable** 0: Disabled. 1: Enabled.
10	addr_nack_er	R/W	0	**Master Mode Received Address NACK from Slave Interrupt Enable** 0: Disabled. 1: Enabled.
9	to_er	R/W	0	**Timeout Error Interrupt Enable** 0: Disabled. 1: Enabled.
8	arb_er	R/W	0	**Master Mode Arbitration Lost Interrupt Enable** 0: Disabled. 1: Enabled.
7	addr_ack	R/W	0	**Received Address ACK from Slave Interrupt Enable** Set this field to enable interrupts for master mode slave device address ACK events. 0: Interrupt disabled. 1: Interrupt enabled.
6	stop	R/W	0	**STOP Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.

I²C Interrupt Enable 0			I2Cn_INT_EN0	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
5	tx_thresh	R/W	0	**Transmit FIFO Threshold Level Interrupt Enable** 0: Disabled. 1: Enabled.
4	rx_thresh	R/W	0	**Receive FIFO Threshold Level Interrupt Enable** 0: Disabled. 1: Enabled.
3	addr_match	R/W	0	**Slave Mode Incoming Address Match Interrupt Enable** 0: Disabled. 1: Enabled.
2	gen_call_addr	R/W	0	**Slave Mode General Call Address Match Received Interrupt Enable** 0: Disabled. 1: Enabled.
1	rx_mode	R/W	0	**Interactive Receive Interrupt Enable** 0: Disabled. 1: Enabled.
0	done	R/W	0	**Transfer Complete Interrupt Enable** 0: Disabled. 1: Enabled.

Table 12-10: I²C Interrupt Flag 1 Register

I²C Interrupt Status Flags 1			I2Cn_INT_FL1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	start	R/W1C	0	**START Condition Status Flag** If set, a device START condition has been detected. 0: START condition not detected. 1: START condition detected.
1	tx_underflow	R/W1C	0	**Slave Mode: Transmit FIFO Underflow Status Flag** In slave mode operation, the hardware sets this flag automatically if the transmit FIFO is empty and the master requests more data by sending an ACK after the previous byte is transferred. 0: Slave mode transmit FIFO underflow condition has not occurred. 1: Slave mode transmit FIFO underflow condition occurred.
0	rx_overflow	R/W1C	0	**Slave Mode: Receive FIFO Overflow Status Flag** In slave mode operation, the hardware sets this flag automatically when a receive FIFO overflow occurs. Write 1 to clear. Writing 0 has no effect. 0: Slave mode receive FIFO overflow event has not occurred. 1: Slave mode receive FIFO overflow condition occurred (data lost).

Table 12-11: I²C Interrupt Enable 1 Register

I²C Interrupt Enable 1			I2Cn_INT_EN1	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**

I²C Interrupt Enable 1			I2Cn_INT_EN1	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
2	start	R/W	0	**START Condition Interrupt Enable** 0: Disabled. 1: Enabled.
1	tx_underflow	R/W	0	**Slave Mode Transmit FIFO Underflow Interrupt Enable** 0: Disabled. 1: Enabled.
0	rx_overflow	R/W	0	**Slave Mode Receive FIFO Overflow Interrupt Enable** 0: Disabled. 1: Enabled.

Table 12-12: I²C FIFO Length Register

I²C FIFO Length			I2Cn_FIFO_LEN	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:8	tx_len	RO	8	**Transmit FIFO Length** Returns the length of the transmit FIFO. 8: 8-byte transmit FIFO.
7:0	rx_len	RO	8	**Receive FIFO Length** This field returns the length of the receive FIFO. 8: 8-byte receive FIFO.

Table 12-13: I²C Receive Control 0 Register

I²C Receive Control 0			I2Cn_RX_CTRL0	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	rx_thresh	R/W	0	**Receive FIFO Threshold Level** Set this field to the required number of bytes to trigger a receive FIFO threshold event. When the number of bytes in the receive FIFO is equal to or greater than this field, the hardware sets the *I2Cn_INT_FL0.rx_thresh* bit indicating a receive FIFO threshold level event. 0: 0 bytes or more in the receive FIFO causes a threshold event. 1: 1+ bytes in the receive FIFO triggers a receive threshold event (recommended minimum value). … 8: Receive FIFO threshold event only occurs when the receive FIFO is full.
7	rx_flush	R/W1O	0	**Flush Receive FIFO** Write 1 to this field to initiate a receive FIFO flush, clearing all data in the receive FIFO. This field is automatically cleared by hardware when the receive FIFO flush completes. Writing 0 has no effect. 0: Normal operation 1: Flush the receive FIFO
6:1	-	RO	0	**Reserved**

I²C Receive Control 0			I2Cn_RX_CTRL0	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
0	dnr	R/W	0	**Slave Mode Do Not Respond** Slave mode operation only. If the device has been addressed for a write operation, and there is still data in the receive FIFO, then: 0: Always respond to an address match with an ACK but then always respond to data bytes with a NACK. 1: NACK the address.

Table 12-14: I²C Receive Control 1 Register

I²C Receive Control 1			I2Cn_RX_CTRL1	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	rx_fifo	R	0	**Receive FIFO Byte Count Status** 0 – 8: Number of bytes in the receive FIFO.
7:0	rx_cnt	R/W	1	**Receive FIFO Transaction Byte Count Configuration** When in master mode, write the number of bytes to be received in a transaction from 1 to 256. 0x00 represents 256. 0: 256 byte receive transaction. 1: 1 byte receive transaction. 2: 2 byte receive transaction. … 255: 255 byte receive transaction. *This field is ignored when I2Cn_CTRL.rx_mode = 1. To receive more than 256 bytes, use I2Cn_CTRL.rx_mode = 1*

Table 12-15: I²C Transmit Control 0 Register

I²C Transmit Control 0			I2Cn_TX_CTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	*tx_thresh*	R/W	0	**Transmit FIFO Threshold Level** This field sets the level for a transmit FIFO threshold event interrupt. If the number of bytes remaining in the transmit FIFO falls to this level or lower, the interrupt flag *I2Cn_INT_FL0.tx_thresh* is set, indicating a transmit FIFO threshold event occurred. 0: 0 bytes remaining in the transmit FIFO triggers a transmit FIFO threshold event. 1: 1 byte or fewer remaining in the transmit FIFO triggers a transmit FIFO threshold event (recommended minimum value). … 7: 7 or fewer bytes remaining in the transmit FIFO triggers a transmit FIFO threshold event
7	tx_flush	R/W1O	0	**Transmit FIFO Flush** A transmit FIFO flush clears all remaining data from the transmit FIFO. 0: Normal operation 1: Flush the transmit FIFO *Note: Hardware automatically clears this bit to 0 after it is written to 1 when the flush is completed.* If *I2Cn_INT_FL0.tx_lock_out* = 1, then *I2Cn_TX_CTRL0.tx_flush* = 1.

I²C Transmit Control 0			I2Cn_TX_CTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
6	-	RO	0	**Reserved**
5	tx_nack_afd	R/W	0	**Transmit FIFO received NACK Auto Flush Disable** Various situations or conditions are described in this user guide that leads to the transmit FIFO being flushed and locked out (*I2Cn_INT_FL0.tx_lock_out* = 1). 0: Received NACK at the end of a slave transmit operation enabled. 1: Received NACK at the end of slave transmit operation disabled. *Note: Upon entering transmit preload mode, the hardware automatically sets this bit to 0. The software can subsequently set to any value desired (i.e., hardware does not continuously force the bitfield to this value).*
4	tx_amr_afd	R/W	0	**Transmit FIFO Slave Address Match Read Auto Flush Disable** Various situations or conditions are described in this user guide that leads to the transmit FIFO being flushed and locked out (*I2Cn_INT_FL0.tx_lock_out* = 1). 0: Enabled. 1: Disabled. *Note: Upon entering transmit preload mode, the hardware automatically sets this bit to 1. The software can subsequently set to any value desired (i.e., hardware does not continuously force the bitfield to this value).*
3	tx_amw_afd	R/W	0	**Transmit FIFO Slave Address Match Write Auto Flush Disable** Various situations or conditions are described in this user guide that leads to the transmit FIFO being flushed and locked out (*I2Cn_INT_FL0.tx_lock_out* = 1). 0: Enabled 1: Disabled. *Note: Upon entering transmit preload mode, the hardware automatically sets this bit to 1. The software can subsequently set to any value desired (i.e., hardware does not continuously force the bitfield to this value).*
2	tx_amgc_afd	R/W	0	**Transmit FIFO General Call Address Match Auto Flush Disable** Various situations or conditions are described in this user guide that leads to the transmit FIFO being flushed and locked out (*I2Cn_INT_FL0.tx_lock_out* = 1). 0: Enabled. 1: Disabled. *Note: Upon entering transmit preload mode, the hardware automatically sets this bit to 1. The software can subsequently set to any value desired (i.e., hardware does not continuously force the bitfield to this value).*
1	tx_ready_mode	R/W	0	**Transmit FIFO Ready Manual Mode** 0: Hardware controls *I2Cn_TX_CTRL1.tx_ready* 1: Software controls *I2Cn_TX_CTRL1.tx_ready*
0	tx_preload	R/W	0	**Transmit FIFO Preload Mode Enable** 0: Normal operation. An address match in slave mode, or a general call address match, flushes and locks the transmit FIFO so it cannot be written and set *I2Cn_INT_FL0.tx_lock_out.* 1: Enabled. An address match in slave mode or a general call address match does not lock the transmit FIFO and does not set *I2Cn_INT_FL0.tx_lock_out*. This allows the software to preload data into the transmit FIFO. The status of the I²C is controllable at *I2Cn_TX_CTRL1.tx_ready.*

Table 12-16: I²C Transmit Control 1 Register

I²C Transmit Control Register 1			I2Cn_TX_CTRL1	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	tx_fifo	R	0	**Transmit FIFO Byte Count Status** 0- 8: Number of bytes in the transmit FIFO
7:1	-	RO	0	**Reserved**
0	tx_ready	R/1	1	**Transmit FIFO Preload Ready Status** When transmit FIFO preload mode is enabled, *I2Cn_TX_CTRL0.tx_preload* = 1, this bit is automatically cleared to 0. While this bit is 0, if the I²C hardware receives a slave address match, a NACK is sent. Once the I²C hardware is ready (the software has preloaded the transmit FIFO, configured the DMA, etc.), software must set this bit to 1 so the I²C hardware can send an ACK on a slave address match. When transmit FIFO preload mode is disabled, *I2Cn_TX_CTRL0.tx_preload* = 1, this bit is forced to 1, and the I²C hardware behaves normally.

Table 12-17: I²C Data Register

I²C Data			I2Cn_FIFO	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	data	R/W	0xFF	**I²C FIFO Data Register** Reads from this register pop data off the receive FIFO. Writes to this register push data in the transmit FIFO. Reading from an empty receive FIFO returns 0xFF. Writes to a full transmit FIFO are ignored.

Table 12-18: I²C Master Control Register

I²C Master Control			I2Cn_MASTER_CTRL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved**
10:8	mcode	R/W	0	**MCODE** This field sets the master code used in Hs-Mode operation
7	sl_ex_addr	R/W	0	**Slave Extended Addressing Enable** 0: Send a 7-bit address to the slave. 1: Send a 10-bit address to the slave.
6:3	-	RO	0	**Reserved**
2	stop	R/W1O	0	**Send STOP Condition** 1: Send a STOP Condition at the end of the current transaction. *Note: This bit is automatically cleared by hardware when the STOP condition begins.*

I²C Master Control			I2Cn_MASTER_CTRL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
1	restart	R/W1O	0	**Send Repeated START Condition** After sending data to a slave, the master may send another START to retain control of the bus. 1: Send a repeated START condition to the slave instead of sending a STOP condition at the end of the current transaction. *Note: This bit is automatically cleared by hardware when the repeated START condition begins.*
0	start	R/W1O	0	**Start Master Mode Transfer** 1: Start master mode transfer *Note: This bit is automatically cleared by hardware when the transfer is completed or aborted.*

Table 12-19: I²C SCL Low Control Register

I²C Clock Low Control			I2Cn_CLK_LO	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	scl_lo	R/W	0x001	**Clock Low Time** In master mode, this configures the SCL low time. $t_{SCL\_LO} = f_{I2C\_CLK} \times (scl\_lo + 1)$ *Note: 0 is not a valid setting for this field.*

Table 12-20: I²C SCL High Control Register

I²C Clock High Control			I2Cn_CLK_HI	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	scl_hi	R/W	0x001	**Clock High Time** In master mode, this configures the SCL high time. $t_{SCL\_HI} = {}^{1}/_{f_{I2C\_CLK}} \times (scl\_hi + 1)$ In both master and slave mode, this also configures the time SCL is held low after new data is loaded from the transmit FIFO or after the software clears *I2Cn_INT_FL0.rx_mode* during IRXM. *Note: 0 is not a valid setting for this field.*

Table 12-21: I²C Hs-Mode Clock Control Register

I²C Hs-Mode Clock Control			I2Cn_HS_CLK	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field.

I²C Hs-Mode Clock Control			I2Cn_HS_CLK	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
15:8	hs_clk_hi	R/W	0	**Hs-Mode Clock High Time** This field sets the Hs-Mode clock high count. In Slave mode, this is the time SCL is held high after data is output on SDA. *Note: See section SCL Clock Generation for Hs-Mode for details on the requirements for the Hs-Mode clock high and low times.*
7:0	hs_clk_lo	R/W	0	**Hs-Mode Clock Low Time** This field sets the Hs-Mode clock low count. In Slave mode, this is the time SCL is held low after data is output on SDA. *Note: See section SCL Clock Generation for Hs-Mode for details on the requirements for the Hs-Mode clock high and low times.*

Table 12-22: I²C Timeout Register

I²C Timeout			I2Cn_TIMEOUT	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	to	R/W	0	**Bus Error SCL Timeout Period** Set this value to the number of I²C clock cycles desired to cause a bus timeout error. The peripheral timeout timer starts when it pulls SCL low. After the peripheral releases the line, if the line is not pulled high before the timeout number of I²C clock cycles, a bus error condition is set (*I2Cn_INT_FL0.to_er* = 1) and the peripheral releases the SCL and SDA lines 0: Timeout disabled. All other values result in a timeout calculation of: $t_{BUS\_TIMEOUT} = {}^{1}/_{f_{I2C\_CLK}} \times to$ *Note: The timeout counter monitors the I2Cn peripheral's driving of the SCL pin, not an external I²C device driving the SCL pin.*

Table 12-23: I²C DMA Register

I²C DMA			I2Cn_DMA	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	rxen	R/W	0	**Receive DMA Channel Enable** 0: Disabled 1: Enable
0	txen	R/W	0	**Transmit DMA Channel Enable** 0: Disabled 1: Enable

Table 12-24: I²C Slave Address Register

I²C Slave Address			I2Cn_SLAVE_ADDR	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**

I²C Slave Address			I2Cn_SLAVE_ADDR	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
15	ex_addr	R/W	0	**Slave Mode Extended Address Length Select** 0: 7-bit addressing. 1: 10-bit addressing.
14:10	-	RO	0	**Reserved**
9:0	slave_addr	R/W	0	**Slave Mode Slave Address** In slave mode operation, *I2Cn_CTRL.mstr* = 0, set this field to the slave address for the I2Cn port. For 7-bit addressing, the address occupies the least significant 7 bits. For 10-bit addressing, the 9-bits of address occupies the most significant 9 bit, and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE_ADDR.ex_addr controls if this field is a 7-bit or 10-bit address.*

13. Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) is a highly configurable, flexible, and efficient synchronous interface between multiple SPI devices on a single bus. The SPI bus uses a single clock signal, single, dual, or quad data lines, and one or more slave select lines for communication with external SPI devices. An SPI network uses a single master and one or more slaves for any given transaction.

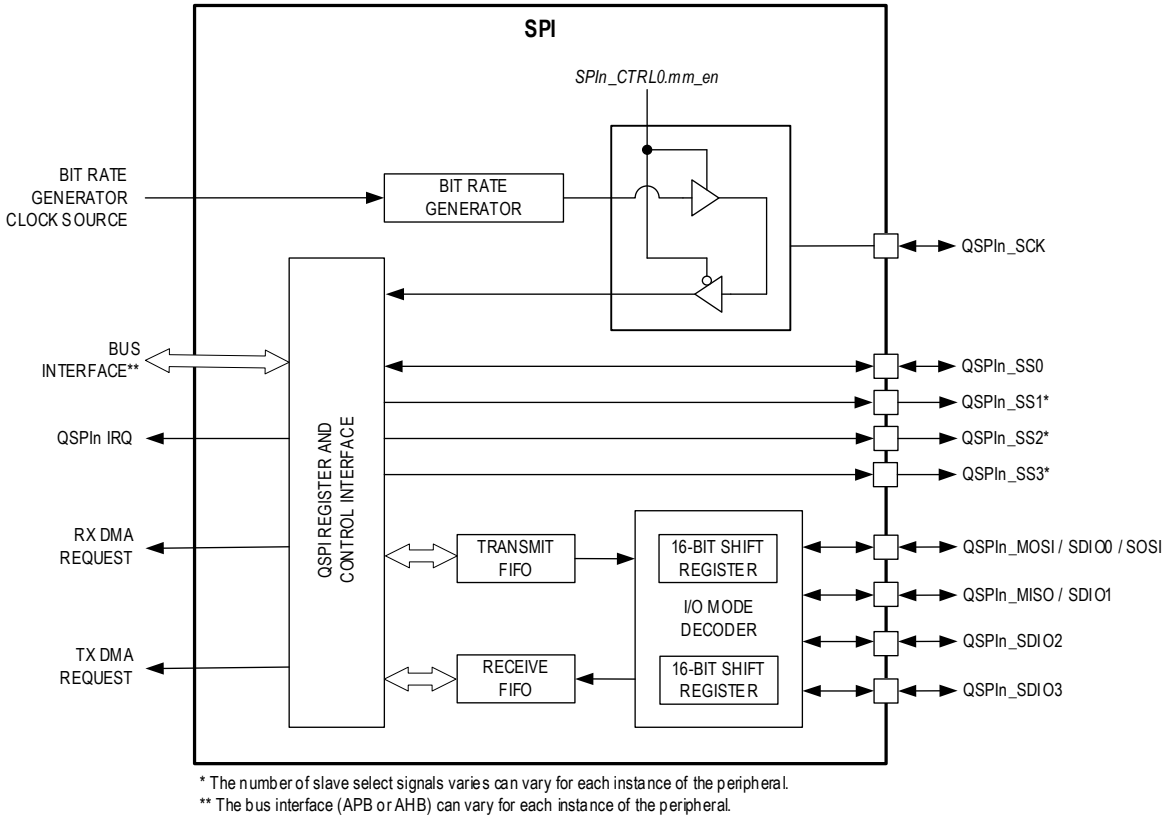
The provided SPI ports support full-duplex, bi-direction I/O, and each SPI includes a Bit Rate Generator (BRG) for generating the clock signal when operating in master mode. Each SPI port operates independently and requires minimal processor overhead. All instances of the SPI peripheral support both master and slave modes and support single-master and multi-master networks.

Features include:

- Dedicated Bit Rate Generator for precision serial clock generation in Master Mode
  - Up to $\frac{f_{PCLK}}{2}$ for instances on the APB bus
  - Up to $\frac{f_{HCLK}}{2}$ for instances on the AHB bus
  - Programmable SCK duty cycle timing
- Full-duplex, synchronous communication of 1 to 16-bit characters
- 3-wire and 4-wire SPI operation for single-bit communication
- Single, Dual, and Quad I/O
- Byte-wide Transmit and Receive FIFOs with 32-byte depth
  - For character sizes greater than 8, each character uses 2 entries per character resulting in 16 entries for the Transmit and Receive FIFO
- Transmit and Receive DMA support
- SPI modes 0, 1, 2, 3
- Configurable slave select lines
  - Programmable slave select level
- Programmable slave select timing for SCK starting edge and ending edge
- Multi-master mode fault detection

Figure 13-1 shows the structure of the peripheral. See *Table 13-1* for the peripheral-specific peripheral bus assignment and bit-rate-generator clock source.

Figure 13-1: SPI Block Diagram



13.1 Instances

The following instances of the peripheral are provided. For a specific instance, replace n in register names with either 0, 1, or 2 depending on the peripheral instance.

Table 13-1: MAX32665/MAX32666 SPI Instances

Name	Formats				Bus Assignment	Bit-Rate-Generator Clock Source Frequency	Slave Select Signals	
	3-Wire	4-Wire	Dual	Quad data			109 WLP	121 CTBGA
SPI0	Yes	Yes	Yes	Yes	AHB	f_{HCLK}	3	3
SPI1	Yes	Yes	Yes	Yes	APB	f_{PCLK}	3	3
SPI2	Yes	Yes	Yes	Yes	APB	f_{PCLK}	3	3

Table 13-2 shows the mapping of the SPI alternate functions.

Table 13-2: MAX32665/MAX32666 SPI Signal Mapping

SPI Signal Name	Alternate Function Name	SPI Port to Pin Mapping			
		SPI0		SPI1	SPI2
		AF1	AF2	AF2	AF2
SCK	QSPIn_SCK	P1.11	P0.11	P0.19	P0.27
MOSI/SDIO0	QSPIn_MOSI/SDIO0	P1.9	P0.9	P0.17	P0.25
MISO/SDIO1	QSPIn_MISO/SDIO1	P1.10	P0.10	P0.18	P0.26
SDIO2	QSPIn_SDIO2	P1.12	P0.12	P0.20	P0.28
SDIO3	QSPIn_SDIO3	P1.13	P0.13	P0.21	P0.29
SS0	QSPIn_SS0	P1.8	P0.8	P0.16	P0.24
SS1	QSPIn_SS1	-	P0.14	P0.22	P0.30
SS2	QSPIn_SS2	-	P0.15	P0.23	P0.31
Note: All Alternate Function Names for SPI device pins are prefixed with "Q." (i.e., QSPIn_SSO is the Slave Select 0 signal for SPIn).					

13.2 SPI Formats

13.2.1 Four-Wire SPI

SPI devices operate as either master or slave devices. In four-wire SPI, four signals are required for communication, as shown in *Table 13-3*.

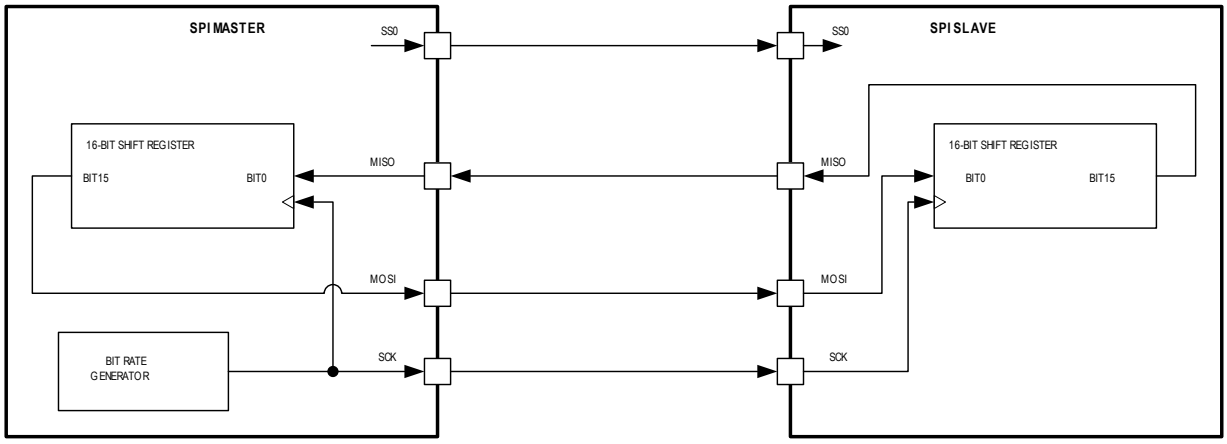
Table 13-3: Four-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The master generates the Serial Clock signal, an output from the master and input to the slave.
MOSI	Master Output Slave Input	In master mode, this signal is used as an output for sending data to the slave. In slave mode, this is the input data from the master.
MISO	Master Input Slave Output	In master mode, this signal is used as an input for receiving data from the slave. In slave mode, this signal is output for transmitting data to the master.
SS	Slave Select	In master mode, this signal is an output used to select a slave device before communication. Peripherals may have multiple slave select outputs to communicate with one or more external slave In slave mode, QSPIn_SS0 is a dedicated input that indicates an external master starts communication. Other slave select signals into the peripheral are ignored in slave mode.

The MAX32665/MAX32666 supports up to three slave select lines for each instance, QSPIn_SS0, QSPIn_SS1, and QSPIn_SS2.

In a typical SPI network, the master device selects the slave device using the slave select output. The master starts the communication by selecting the slave device by asserting the slave select output. The master then starts the SPI clock via the SCK output pin. When a slave device's slave select pin is de-asserted, the device must put the SPI pins in tri-state mode.

Figure 13-2: 4-Wire SPI Connection Diagram



13.2.2 Three-Wire SPI

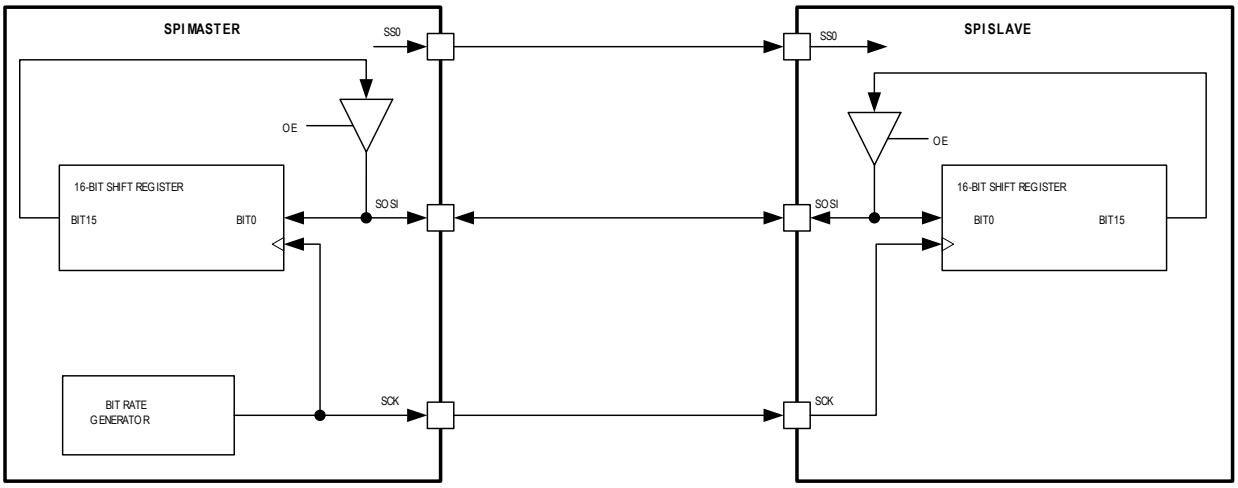
The signals in the three-wire SPI operation are shown in *Table 13-4*. The MOSI signal is used as a bi-directional, half-duplex I/O referred to as Slave Input Slave Output (SISO). Three-wire SPI also uses a serial clock signal generated by the master and a slave select pin controlled by the master.

Table 13-4: Three-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The master generates the serial clock signal, an output from the master and input to the slave.
SISO	Slave Input Slave Output	This is a half-duplex, bidirectional I/O pin used for communication between the SPI master and. This signal is used to transmit data from the master to the slave and to receive data from the slave by the master.
SS	Slave Select	In master mode, this signal is an output used to select a slave device before communication. In slave mode, QSPIn_SS0 is a dedicated input that indicates an external master will start communication. Other slave select signals into the peripheral are ignored in slave mode

A three-wire SPI network is shown in *Figure 13-3*. The master device selects the slave device using the slave select output. The communication starts with the master asserting the slave select line and then the clock (SCK). In three-wire SPI communication, the master and slave must both know the intended direction of the data to prevent bus contention. For a write, the master drives the data out of the SISO pin. For a read, the master must release the SISO line and let the slave drive the SISO line, usually on the second edge of a clock cycle. The direction of transmission is controlled using the FIFO. Writing to the FIFO starts the three-wire SPI write, and reading from the FIFO starts a three-wire SPI read transaction.

Figure 13-3: Generic 3-Wire SPI Master to Slave Connection



13.3 Pin Configuration

Before configuring the SPIn peripheral, first, disable any SPI activity for the port by setting the *SPIn_CTRL0.en* field to 0.

13.3.1 SPIn Alternate Function Mapping

Pin selection and configuration is required to use the SPIn port. The following information applies to SPI master and slave operation and three-wire, four-wire, dual, and quad mode communications. *Table 13-2* maps standard SPI signal names to the Alternate Function name. Determine the pins required for the SPI type and mode in the application and configure the required GPIO as described below.

The Alternate Function Name column in *Table 13-2* maps the standard SPI signal name to the Alternate Function name. Refer to the data sheet for pin availability for a specific package.

When the SPIn port is disabled, *SPIn_CTRL0.en* = 0, the GPIO pins enabled for SPI alternate function are placed in high-impedance input mode.

13.3.2 Four-Wire Format Configuration

Four-wire SPI uses SCK, MISO, MOSI, and one or more SS pins. Four-wire SPI may use more than one slave select pin for a transaction, resulting in more than four wires total, however, the communication is referred to as four-wire for legacy reasons. The following steps set up SPI1 for four-wire SPI with three Slave Select lines.

Note: SPI0 provides SPI pins on Alternate Function 1 and Alternate Function 2. Select the pins mapped to the SPI external device in the design and modify the setup accordingly. There is no restriction on which alternate function is used for a specific SPI pin, and each SPI pin can be used independently from the other pins chosen.

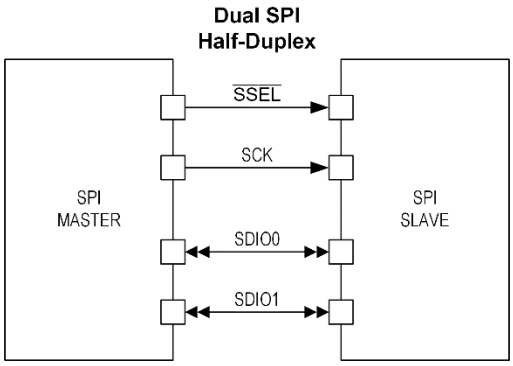
13.3.3 Three-Wire Format Configuration

Three-wire SPI uses SCK, MOSI (SOSI), and one or more slave select pins for an SPI transaction. The three-wire SPI configuration is identical to the four-wire configuration, except QSPIn_MISO does not need set up for the SPI alternate function. The direction of communication in three-wire SPI mode is controlled by the SPIn Transmit and Receive FIFO enables. Enabling the Receive FIFO and disabling the Transmit FIFO indicates a read transaction. Enabling the Transmit FIFO and disabling the Receive FIFO indicates a write transaction. It is an illegal condition to enable both the Transmit and Receive FIFOs in three-wire SPI operation.

13.3.4 Dual-Mode Format Configuration

In Dual mode, two I/O pins are used to transmit 2-bits of data per SCK clock cycle. The communication is half-duplex, and the direction of the data transmission must be known by both the master and slave for a given transaction. Dual-mode SPI uses SCK, SDIO0, SDIO1 and one or more slave select lines as shown in *Figure 13-4*. The configuration of the GPIO pins for Dual-mode SPI is identical to four-wire SPI, and the mode is controlled by setting *SPIn_CTRL2.data_width* to 1, indicating to the SPIn hardware to use SDIO0 and SDIO1 for half-duplex communication rather than full-duplex communication.

Figure 13-4: Dual-Mode SPI Connection Diagram



13.3.5 Quad-Mode Format Pin Configuration

Quad-mode SPI uses four I/O pins to transmit four bits of data per transaction. In Quad-mode SPI, the communication is half-duplex, and the master and slave must know the direction of transmission for each transaction. Quad-mode SPI uses SCK, SDIO0, SDIO1, SDIO2, SDIO3, and one or more slave select pins.

Quad-mode SPI transmits four bits per SCK cycle. Selection of Quad-mode SPI is selected by setting *SPIn_CTRL2.data_width* to 2.

13.4 SPI Clock Configuration

13.4.1 Serial Clock

The SCK signal synchronizes data movement in and out of the device. The master drives SCK as an output to the slave's SCK pin. When SPIn is set to master mode, the SPIn bit rate generator creates the serial clock and outputs it on the configured QSPIn_SCK pin. When SPIn is configured for slave operation, the QSPIn_SCK pin is an input from the external master, and the SPIn hardware synchronizes communications using the SCK input. Operating as a slave, if a SPIn slave select input is not asserted, the SPIn ignores any signals on the serial clock and serial data lines.

When SPIn is configured for slave operation, the maximum SCK input frequency supported is $f_{SCK} = {}^{f_{PCLK}}/_{8}$. For example, if $f_{PCLK} = 48\text{MHz}$, the maximum SPI clock frequency supported in slave mode for SPIn is 6MHz.

In both master and slave devices, data is shifted on one edge of the SCK and is sampled on the opposite edge where data is stable. Data availability and sampling time are controlled using the SPI phase control field, *SPIn_CTRL2.cpha*. The SCK clock polarity field, *SPIn_CTRL2.cpol*, controls if the SCK signal is active high or active low.

The SPIn peripheral supports four combinations of SCK phase and polarity referred to as SPI Modes 0, 1, 2, and 3. Clock Polarity (*SPIn_CTRL2.cpol*) selects an active low/high clock and does not affect the transfer format. Clock Phase (*SPIn_CTRL2.cpha*) selects one of two different transfer formats.

The clock phase and polarity must be identical for the SPI master and slave for proper data transmission. The master always places data on the MOSI line a half-cycle before the SCK edge for the slave to latch the data. See section *Clock Phase and Polarity Control* for additional details.

13.4.2 SPI Peripheral Clock

The System Peripheral Clock, PCLK, drives the SPIn peripheral clock. The SPI0 provides an internal clock, SPI0_CLK, used within the SPI peripheral for the base clock to control the module and generate the SCK clock when in master mode. Set the SPI0 internal clock using the field *SPIn_CLK_CFG.scale* as shown in *Equation 13-1*. Valid settings for *SPIn_CLK_CFG.scale* are 0 to 8, allowing a divisor of 1 to 256.

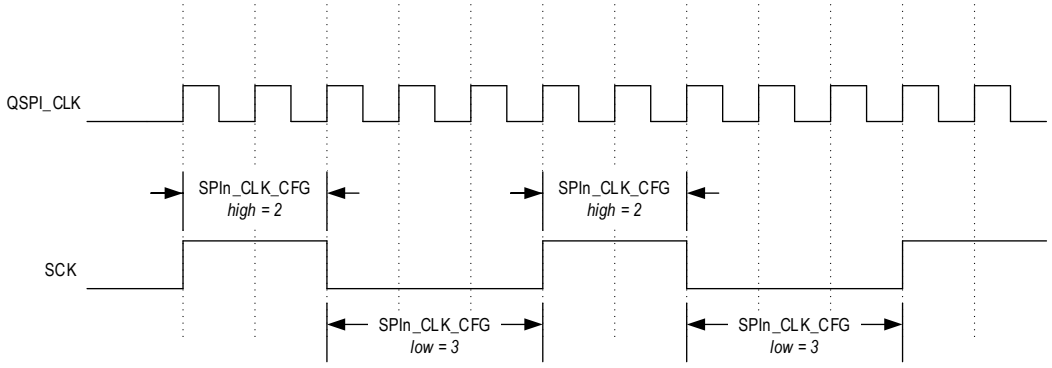
Equation 13-1: SPI Peripheral Clock

$$f_{SPI\_CLK} = \frac{f_{PCLK}}{2^{scale}}$$

13.4.3 Master Mode Serial Clock Generation

In master and multi-master mode, the SCK clock is generated by the master. The SPI0 provides a control for both the high time and low time of the SCK clock. This control allows setting the high and low times for the SCK to duty cycles other than 50% if required. The SCK clock uses the SPIn peripheral clock as a base value, and the high and low values are a count of the number of f_{SPI_CLK} clocks. *Figure 13-5* visually represents the use of the *SPIn_CLK_CFG.hi* and *SPIn_CLK_CFG.lo* fields for a non-50% duty cycle serial clock generation. See *Equation 13-2* and *Equation 13-3* for calculating the SCK high and low time from the *SPIn_CLK_CFG.hi* and *SPIn_CLK_CFG.lo* field values.

Figure 13-5: SCK Clock Rate Control



Equation 13-2: SCK High Time

$$t_{SCK\_HI} = t_{SPInCLK} \times \text{SPIn\_CLK\_CFG.hi}$$

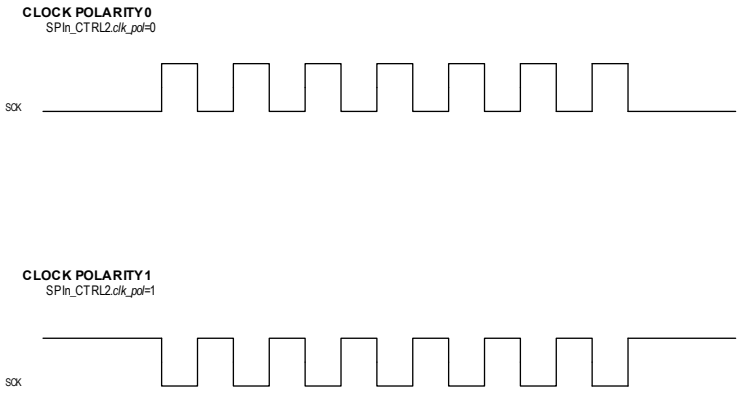
Equation 13-3: SCK Low Time

$$t_{SCK\_LOW} = t_{SPInCLK} \times \text{SPIn\_CLK\_CFG.lo}$$

13.4.4 Clock Phase and Polarity Control

SPIn supports four combinations of clock and phase polarity, as shown in *Table 13-5*. Clock polarity is controlled using the bit *SPIn_CTRL2.cpol* and determines if the clock is active high or active low, as shown in *Figure 13-6*. Clock polarity does not affect the transfer format for SPI. The clock phase determines when the data must be stable for sampling. Setting the clock phase to 0, *SPIn_CTRL2.cpha* = 0, dictates the SPI data is sampled on the initial SPI clock edge regardless of clock polarity. Phase 1, *SPIn_CTRL2.cpha* = 1, results in a data sample occurring on the second edge of the clock regardless of clock polarity.

Figure 13-6: SPI Clock Polarity



The clock phase and polarity must be identical for the SPI master and slave for proper data transmission. The master always places data on the MOSI line a half-cycle before the SCK edge for the slave to latch the data.

Table 13-5: SPI Modes Clock Phase and Polarity Operation

SPI Mode	*SPIn_CTRL2 cpha*	*SPIn_CTRL2 cpol*	SCK Transmit Edge	SCK Receive Edge	SCK Idle State
0	0	0	Falling	Rising	Low
1	0	1	Rising	Falling	High
2	1	0	Rising	Falling	Low
3	1	1	Falling	Rising	High

13.4.5 SPIn FIFOs

The Transmit FIFO hardware is 32 bytes deep. The write data width can be 8-, 16- or 32-bits wide. A 16-bit write queues a 16-bit word to the FIFO hardware. A 32-bit write queues two 16-bit words to the FIFO hardware with the least significant word dequeued first. Bytes must be written to two consecutive byte addresses, with the odd byte as the most significant and the even byte as the least significant. The FIFO logic waits for both the odd and even bytes to be written to this register space before dequeuing the 16-bit result to the FIFO.

The Receive FIFO hardware is 32 bytes deep. Read data width can be 8-, 16- or 32-bits. A byte read from this register dequeues one byte from the FIFO. A 16-bit read from this register dequeues two bytes from the FIFO, least significant byte first. A 32-bit read from this register dequeues four bytes from the FIFO, the least significant byte first.

13.4.6 SPI Interrupts and Wakeups

The SPIn supports multiple interrupt sources. Status flags for each interrupt are set regardless of the state of the interrupt enable bit for that event. The event happens once when the condition is satisfied. The status flag must be cleared by software by writing a 1 to the interrupt flag.

The following FIFO interrupts are supported:

- Transmit FIFO Empty
- Transmit FIFO Threshold
- Receive FIFO Full
- Receive FIFO threshold
- Transmit FIFO Underrun
  - Slave mode only, master mode stalls the serial clock
- Transmit FIFO Overrun
- Receive FIFO Underrun
- Receive FIFO Overrun (Slave Mode only, Master Mode stalls the clock)

SPIn supports interrupts for the internal state of the SPI as well as external signals. The following transmission interrupts are supported:

- SSn asserted or de-asserted
- SPI transaction complete
- Slave mode transaction aborted
- Multi-master fault

The SPIn port can wake up the microcontroller for low-power modes when the wake event is enabled. SPIn events that can wake the microcontroller are:

- Receive FIFO full
- Transmit FIFO empty
- Receive FIFO threshold
- Transmit FIFO threshold

13.5 Registers

See *Table 3-3* for the base address of this peripheral/module. Each instance has its own, independent set of the registers shown in *Table 13-6*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 13-6: SPIn Base Address Offsets, Register Names, Access and Descriptions

Offset	Register Name	Description
[0x0000]	*SPIn_DATA*	SPIn FIFO Data Register
[0x0004]	*SPIn_CTRL0*	SPIn Master Signals Control Register
[0x0008]	*SPIn_CTRL1*	SPIn Transmit Packet Size Register
[0x000C]	*SPIn_CTRL2*	SPIn Static Configuration Register
[0x0010]	*SPIn_SS_TIME*	SPIn Slave Select Timing Register
[0x0014]	*SPIn_CLK_CFG*	SPIn Master Clock Configuration Register
[0x001C]	*SPIn_DMA*	SPIn DMA Control Register
[0x0020]	*SPIn_INT_FL*	SPIn Interrupt Flag Register
[0x0024]	*SPIn_INT_EN*	SPIn Interrupt Enable Register
[0x0028]	*SPIn_WAKE_FL*	SPIn Wakeup Flags Register
[0x002C]	*SPIn_WAKE_EN*	SPIn Wakeup Enable Register
[0x0030]	*SPIn_STAT*	SPIn Status Register

13.6 Register Details

Table 13-7: SPIn FIFO Data Register

SPIn FIFO Data Register			SPIn_DATA	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	qspififo	R/W	0	**SPIn FIFO Data Register** This register is used for the SPI Transmit and Receive FIFO. Reading from this register returns characters from the Receive FIFO, and writing to this register adds characters to the Transmit FIFO. Read and write this register in either 1-byte, 2-byte, or 4-byte widths only.

Table 13-8: SPIn Control 0 Register

SPIn Control 0 Register			SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:19	-	R/W	0	**Reserved for Future Use** Do not modify this field.
18:16	ss	R/W	0	**Master Slave Select** The SPIn includes up to three slave select lines for each port. This field selects which slave select pin is active when the next SPI transaction is started (*SPIn_CTRL0.start* = 1). One or more slave select pins can be selected for each SPI transaction by setting the bit for each slave select pin. For example, use QSPIn_SS0 and QSPIn_SS2 by setting this field to 0b101. 0b001: QSPIn_SS0 0b010: QSPIn_SS1 0b100: QSPIn_SS2 *Note: This field is only used when the SPIn is configured for Master Mode (SPIn_CTRL0.master = 1).*
15:9	-	R/W	0	**Reserved for Future Use** Do not modify this field.
8	ss_ctrl	R/W	0	**Master Slave Select Control** This field controls the behavior of the slave select pins after a transaction. The default behavior, *ss_ctrl* = 0, de-asserts the slave select pin after the transaction. Set this field to 1 to leave the slave select pins asserted after the transaction. If the external device supports this behavior, leaving the slave select pins asserted allows multiple transactions without delay associated with de-assertion of the slave select pin between transactions. 0: Slave Select is de-asserted at the end of a transmission 1: Slave Select stays asserted at the end of a transmission
7:6	-	R/W	0	**Reserved for Future Use** Do not modify this field.
5	start	R/W1O	0	**Master Start Data Transmission** Set this field to 1 to start an SPI master mode transaction. 0: No master mode transaction active. 1: Master initiates data transmission. Ensure that all pending transactions are complete before setting this field to 1. *Note: This field is only used when the SPIn is configured for Master Mode (SPIn_CTRL0.master = 1).*

SPIn Control 0 Register			SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
4	ss_io	R/W	0	**Master-Slave Select Signal Direction** Set the I/O direction for 0: Slave Select is an output 1: Slave Select is an input *Note: This field is only used when the SPIn is configured for Master Mode (SPIn_CTRL0.master = 1).*
3:2	-	R/W	0	**Reserved for Future Use** Do not modify this field.
1	master	R/W	0	**SPI Master Mode Enable** This field selects between slave mode and master mode operation for the SPI port. Write this field to 0 to operate as an SPI slave. Setting this field to 1 sets the port as an SPI master. 0: Slave mode SPI operation. 1: Master mode SPI operation.
0	en	R/W	0	**SPI Enable/Disable** This field enables and disables the SPIn port. Disable the SPIn port by setting this field to 0. Disabling the SPIn port does not affect the SPIn FIFOs or register settings. 0: SPIn port is disabled 1: SPIn port is enabled *Note: If this bit is set without setting SPIn_CTRL0.master, the SPIn peripheral operates in Slave mode. This could cause the peripheral to immediately start a transaction (depending upon the state of the peripheral slave select device pin), thus setting the SPIn_STAT.busy bit to 1.*

Table 13-9: SPIn Transmit Packet Size Register

SPIn Transmit Packet Size Register			SPIn_CTRL1	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	rx_num_char	R/W	0	**Number of Receive Characters** The number of characters to receive in receive FIFO. Note: If the SPIn port is set to operate in 4-wire mode, this field is ignored, and the SPIn_CTRL1.tx_num_chars field is used for both the number of characters to receive and transmit.
15:0	tx_num_char	R/W	0	**Number of Transmit Characters** The number of characters to transmit from transmit FIFO. Note: If the SPIn port is set to operate in 4-wire mode, this field is used for the number of characters to receive and transmit.

Table 13-10: SPIn Control 2 Register

SPIn Control 2 Register			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved for Future Use** Do not modify this field.
19:16	ss_pol	R/W	0	**Slave Select Polarity** Controls the polarity of each SS signal where each bit position corresponds to a SS signal. QSPIn_SS0 is controlled with bit position 0, and QSPIn_SS2 is controlled with bit position 2. For each bit position, 0: SS is active low 1: SS is active high
15	three_wire	R/W	0	**Three-Wire SPI Enable** Set this field to 1 to enable three-wire SPI communication. Set this field to 0 for four-wire full-duplex SPI communication. 0: Four-wire full-duplex mode enabled. 1: Three-wire mode enabled *Note: This field is ignored for Dual SPI, SPIn_CTRL2.data_width = 1, and Quad SPI, SPIn_CTRL2.data_width = 2, this field is ignored.*
14	-	R/W	0	**Reserved for Future Use** Do not modify this field.
13:12	data_width	R/W	0b00	**SPI Data Width** This field controls the number of data lines used for SPI communications. **Three-wire SPI**, *data_width* = 0 Set this field to 0, indicating QSPIn_MOSI (SISO) is used for half-duplex communication. **Four-wire full-duplex SPI operation**, *data_width* = 0 Set this field to 0, indicating QSPIn_MOSI and QSPIn_MISO are used for the SPI data output and input, respectively. **Dual SPI**, *data_width* = 1, uses QSPIn_SDIO0 and QSPIn_SDIO1 for half-duplex communication. **Quad SPI**, *data_width* = 2, uses QSPIn_SDIO0, QSPIn_SDIO1, QSPIn_SDIO2, and QSPIn_SDIO3 for half-duplex communication. 0: 1-bit per SCK cycle (Three-wire half-duplex SPI and Four-wire full-duplex SPI) 1: 2-bits per SCK cycle (Dual SPI) 2: 4-bits per SCK cycle (Quad SPI) 3: Reserved *Note: When this field is set to 0, use SPIn_CTRL2.three_wire to select either Three-Wire SPI or Four-Wire SPI operation.*

SPIn Control 2 Register			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
11:8	numbits	R/W	0	**Number of Bits per Character** Set this field to the number of bits per character for the SPI transaction. Setting this field to 0 indicates a character size of 16. 0: 16-bits per character 1: 1-bit per character 2: 2-bits per character … 14: 14-bits per character 15: 15-bits per character *Note: 1-bit and 9-bit character lengths are not supported in Slave Mode, SPIn_CTRL0.master = 0.* *Note: For Dual and Quad mode SPI, the character size should be divisible by the number of bits per SCK cycle.*
7:2	-	R/W	0	**Reserved for Future Use** Do not modify this field.
1	cpol	R/W	0	**Clock Polarity** This field controls the SCK polarity. The default clock polarity is for SPI Mode 0 and Mode 1 operation and is active high. Invert the SCK polarity for SPI Mode 2 and Mode 3 operation. 0: Standard SCK for use in SPI Mode 0 and Mode 1 1: Inverted SCK for use in SPI Mode 2 and Mode 3
0	cpha	R/W	0	**Clock Phase** 0: Data sampled on clock rising edge. Use when in SPI Mode 0 and Mode 2 1: Data sampled on clock falling edge. Use when in SPI Mode 1 and Mode 3

Table 13-11: SPIn Slave Select Timing Register

SPIn Slave Select Timing Register			SPIn_SS_TIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	R/W	0	**Reserved for Future Use** Do not modify this field.
23:16	inact	R/W	0	**Inactive Stretch** This field controls the number of system clocks the bus is inactive between the end of a transaction (Slave Select inactive) and the start of the next transaction (Slave Select active). 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255

SPIn Slave Select Timing Register			SPIn_SS_TIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
15:8	post	R/W	0	**Slave Select Hold Post Last SCK** The number of system clock cycles that SS remains active after the last SCK edge. 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255
7:0	pre	R/W	0	**Slave Select Delay to First SCK** Set the number of system clock cycles the Slave Select is held active before the first SCK edge. 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255

Table 13-12: SPIn Master Clock Configuration Registers

SPIn Master Clock Configuration Register			SPIn_CLK_CFG	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved for Future Use** Do not modify this field.
19:16	scale	R/W	0	**SPI Peripheral Clock Scale** Scales the SPI input clock (PCLK for SPI1/SPI2 and HCLK for SPI0) by 2^{scale} to generate the SPIn peripheral clock. $f_{SPInCLK} = \frac{f_{SPIn\_INPUT\_CLK}}{2^{scale}}$ Valid values for scale are 0 to 8 inclusive. Values greater than 8 are reserved for future use. *Note: If SPIn_CLK_CFG.scale = 0, SPIn_CLK_CFG.hi = 0, and SPIn_CLK_CFG.lo = 0, character sizes of 2 and 10 bits are not supported.*
15:8	hi	R/W	0	**SCK Hi Clock Cycles Control** 0: Hi duty-cycle control disabled. Only valid if scale = 0. 1 to 15: The number of SPIn peripheral clocks, $f_{SPInCLK}$, that SCK is high. *Note: If SPIn_CLK_CFG.scale = 0, SPIn_CLK_CFG.hi = 0, and SPIn_CLK_CFG.lo = 0, character sizes of 2 and 10 bits are not supported.*

SPIn Master Clock Configuration Register			SPIn_CLK_CFG	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
7:0	lo	R/W	0	**SCK Low Clock Cycles Control** This field controls the SCK low clock time and controls the overall SCK duty cycle in combination with the *SPIn_CLK_CFG.hi* field. 0: Low duty cycle control disabled. Setting this field to 0 is only valid if *SPIn_CLK_CFG.scale* = 0. 1 to 15: The number of SPIn peripheral clocks, $f_{SPInCLK}$ that the SCK signal is low. *Note: If SPIn_CLK_CFG.scale = 0, SPIn_CLK_CFG.hi = 0, and SPIn_CLK_CFG.lo = 0, character sizes of 2 and 10 bits are not supported.*

Table 13-13: SPIn DMA Control Registers

SPIn DMA Control Register			SPIn_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31	rx_dma_en	R/W	0	**Receive DMA Enable** 0: Disabled. Any pending DMA requests are cleared 1: Enabled
30	-	R/W	0	**Reserved for Future Use** Do not modify this field.
29:24	rx_fifo_cnt	R	0	**Number of Bytes in the receive FIFO** Read returns the number of bytes currently in the receive FIFO
23	rx_fifo_clear	W	-	**Clear the receive FIFO** 1: Clear the receive FIFO and any pending receive FIFO flags in SPIn_INTFL. This should be done when the receive FIFO is inactive. Writing a 0 has no effect.
22	rx_fifo_en	R/W	0	**Receive FIFO Enabled** 0: Disabled 1: Enabled
21	-	R/W	0	**Reserved for Future Use** Do not modify this field.
20:16	rx_fifo_level	R/W	0x00	**Receive FIFO Threshold Level** Set this value to the desired receive FIFO threshold level. When the receive FIFO contains the number of bytes or greater than this field, a DMA request is triggered, and *SPIn_INT_FL.rx_thresh* is set. Valid values are 0 to 30. *Note: 31 is an invalid setting and reserved for future use.*
15	tx_dma_en	R/W	0	**Transmit DMA Enable** 0: Disabled. Any pending DMA requests are cleared 1: Transmit DMA is enabled
14	-	R/W	0	**Reserved for Future Use** Do not modify this field.
13:8	tx_fifo_cnt	RO	0	**Number of Bytes in the Transmit FIFO** Read this field to determine the number of bytes currently in the transmit FIFO.

SPIn DMA Control Register		SPIn_DMA		[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
7	tx_fifo_clear	R/W	0	**T FIFO Clear** Set this bit to clear the transmit FIFO and all transmit FIFO flags in the *SPIn_INT_FL* register. *Note: The transmit FIFO should be disabled (SPIn_DMA.tx_fifo_en = 0) before setting this field.* *Note: Setting this field to 0 has no effect.*
6	tx_fifo_en	R/W	0	**Transmit FIFO Enabled** 0: Disabled 1: Enabled
5	-	R/W	0	**Reserved for Future Use** Do not modify this field.
4:0	tx_fifo_level	R/W	0x10	**Transmit FIFO Threshold Level** When the transmit FIFO count (*SPIn_DMA.tx_fifo_cnt*) falls below this value, a DMA request is triggered, and *SPIn_INT_FL.tx_thresh* is set.

Table 13-14: SPIn Interrupt Status Flags Registers

SPIn Interrupt Status Flags Register		SPIn_INT_FL		[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field.
15	rx_und	R/1	0	**Receive FIFO Underrun Flag** Set when a read is attempted from an empty receive FIFO.
14	rx_ovr	R/W1C	0	**Receive FIFO Overrun Flag** Set if SPI is in Slave Mode, and a write to a full receive FIFO is attempted. If the SPI is in Master Mode, this bit is not set as the SPI stalls the clock until data is read from the receive FIFO.
13	tx_und	R/W1C	0	**Transmit FIFO Underrun Flag** Set if SPI is in Slave Mode, and a read from empty transmit FIFO is attempted. If SPI is in Master Mode, this bit is not set as the SPI stalls the clock until data is written to the empty transmit FIFO.
12	tx_ovr	R/W1C	0	**Transmit FIFO Overrun Flag** Set when a write is attempted to the full transmit FIFO.
11	m_done	R/W1C	0	**Master Data Transmission Done Flag** Set if SPI is in Master Mode and all transactions have been completed.
10	-	R/W	0	**Reserved for Future Use** Do not modify this field.
9	abort	R/W1C	0	**Slave Mode Transaction Abort Detected Flag** Set if the SPI is in Slave Mode, and SS is de-asserted before receiving a complete character.

SPIn Interrupt Status Flags Register		SPIn_INT_FL		[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
8	fault	R/W1C	0	**Multi-Master Fault Flag** Set if the SPI is in Master Mode, Multi-Master Mode is enabled, and a Slave Select input is asserted. A collision also sets this flag.
7:6	-	R/W	0	**Reserved for Future Use** Do not modify this field.
5	ssd	R/W1C	0	**Slave Select Deasserted Flag**
4	ssa	R/W1C	0	**Slave Select Asserted Flag**
3	rx_full	R/W1C	0	**Receive FIFO Full Flag**
2	rx_thresh	R/W1C	0	**Receive FIFO Threshold Level Crossed Flag** Set when the receive FIFO exceeds the value in *SPIn_DMA.rx_fifo*_level.
1	tx_empty	R/W1C	1	**Transmit FIFO Empty Flag**
0	tx_thresh	R/W1C	0	**Transmit FIFO Threshold Level Crossed Flag** Set when the transmit FIFO is less than the value in *SPIn_DMA.tx_fifo*_level.

Table 13-15: SPIn Interrupt Enable Registers

SPIn Interrupt Enable Register		SPIn_INT_EN		[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field.
15	rx_und	R/W	0	**Receive FIFO Underrun Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled
14	rx_ovr	R/W	0	**Receive FIFO Overrun Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled
13	tx_und	R/W	0	**Transmit FIFO Underrun Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled
12	tx_ovr	R/W	0	**Transmit FIFO Overrun Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled
11	m_done	R/W	0	**Master Data Transmission Done Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled
10	-	R/W	0	**Reserved for Future Use** Do not modify this field.
9	abort	R/W	0	**Slave Mode Abort Detected Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled

SPIn Interrupt Enable Register		SPIn_INT_EN		[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
8	fault	R/W	0	**Multi-Master Fault Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled
7:6	-	R/W	0	**Reserved for Future Use** Do not modify this field.
5	ssd	R/W	0	**Slave Select Deasserted Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled
4	ssa	R/W	0	**Slave Select Asserted Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled
3	rx_full	R/W	0	**Receive FIFO Full Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled
2	rx_thresh	R/W	0	**Receive FIFO Threshold Level Crossed Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled
1	tx_empty	R/W	0	**Transmit FIFO Empty Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled
0	tx_thresh	R/W	0	**Transmit FIFO Threshold Level Crossed Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled

Table 13-16: SPIn Wakeup Status Flags Registers

SPIn Wakeup Flags Register		SPIn_WAKE_FL		[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved for Future Use** Do not modify this field.
3	rx_full	R/W1C	0	**Wake on receive FIFO Full Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
2	rx_thresh	R/W1C	0	**Wake on receive FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
1	tx_empty	R/W1C	0	**Wake on Transmit FIFO Empty Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
0	tx_thresh	R/W1C	0	**Wake on Transmit FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.

Table 13-17: SPIn Wakeup Enable Registers

SPIn Wakeup Enable Register			SPIn_WAKE_EN	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved for Future Use** Do not modify this field.
3	rx_full	R/W	0	**Wake on receive FIFO Full Enable** 0: Wake event is disabled 1: Wake event is enabled.
2	rx_thresh	R/W	0	**Wake on receive FIFO Threshold Level Crossed Enable** 0: Wake event is disabled 1: Wake event is enabled.
1	tx_empty	R/W	0	**Wake on Transmit FIFO Empty Enable** 0: Wake event is disabled 1: Wake event is enabled.
0	tx_thresh	R/W	0	**Wake on Transmit FIFO Threshold Level Crossed Enable** 0: Wake event is disabled 1: Wake event is enabled.

Table 13-18: SPIn Slave Select Timing Registers

SPIn Status Register			SPIn_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	R/W	0	**Reserved for Future Use** Do not modify this field.
0	busy	R	0	**SPI Active Status** 0: SPIn is not active. In master-mode, the *busy* flag is cleared when the last character is sent. In slave-mode, the *busy* field is cleared when the configured slave select input is de-asserted. 1: SPIn is active. In master-mode, the *busy* flag is set when a transaction starts. In slave-mode the *busy* flag is set when a configured slave select input is asserted.

14. HTimer (HTMR)

14.1 Overview

The HTimer (HTMR) is a 40-bit binary timer similar to the real-time clock but is driven by a high-speed internal clock source. The timer provides a short-interval, auto-reload alarm and a long-interval alarm. Configurable alarm settings allow it to be used as a low-power wakeup timer.

The HT clock source is the IBRO.

Two registers combine to create the timer. The *HTMR_SEC.rts* field contains the most significant bits and the *HTMR_SSEC.rtss* field contains the least significant bits. The *HTMR_SEC.rts* field is incremented each time *HTMR_SSEC.rtss* rolls over.

The peripheral does not have a dedicated clock output equivalent to the 32KCAL.

A programmable long-interval alarm is usable with *HTMR_SEC.rts* to provide a single event/alarm timer. When the counter is started, it counts continuously unless it is disabled, and reads of the counter registers do not affect the count.

A separate 32-bit auto-reload short-interval alarm counter register (*HTMR_RSSA*) can generate repeating interval alarms.

14.2 Alarm Functions

The timer provides two alarm functions:

The long interval alarm is generated when *HTMR_RAS.ras* matches *HTMR_SEC.rts*[19:0].

The short-interval alarm provides an internal 32-bit auto-reload counter that increments on each transition of *HTMR_SSEC.rtss*. The counter always increments from the value in *HTMR_RSSA.rssa* up to the maximum value of *HTMR_SSEC.rtss*. When the internal counter rolls over to 0, an alarm is generated, the internal counter is reloaded with *HTMR_RSSA.rssa* and continues incrementing.

14.2.1 Long-Interval Alarm

The long interval counter increments once each time *HTMR_SSEC.rtss* rolls over to 0. The alarm is triggered when the *HTMR_SEC.rts[19:0]* matches *HTMR_RAS.ras*. Hardware will then set the *HTMR_RAS.tod_fl* bit and an interrupt will be generated if software has set *HTMR_RAS.tod_en*.

You must disable the long-interval alarm before changing the *HTMR_RAS.tod* field.

14.2.2 Short-Interval Alarm

The *HTMR_RSSA.rts* and *HTMR_CTRL.ase* fields control the short-interval alarm. Writing *HTMR_RSSA* sets the starting value for the short-interval alarm counter. Writing the Short-Interval Alarm Enable (*HTMR_CTRL.ase*) bit to 1 enables the short-interval alarm. Once enabled, the short-interval alarm begins up-counting from the *HTMR_RSSA* value. When the counter rolls over from 0xFFFF FFFF to 0x0000 0000, hardware sets the *HTMR_CTRL.alsf* bit triggering the alarm. At the same time, hardware also reloads the counter with the value previously written to *HTMR_RSSA.rssa*.

You must disable the short-interval interval alarm, *HTMR_CTRL.ase*, prior to changing the interval alarm value, *HTMR_RSSA*.

The delay (uncertainty) associated with enabling the short-interval alarm is propagated to the first interval alarm. Thereafter, if the interval alarm remains enabled, the alarm triggers after each short-interval interval as defined without the first alarm uncertainty because the short-interval alarm is an auto-reload timer. Enabling the short-interval alarm with the short-interval alarm register set to 0 (*HTMR_RSSA.rssa* = 0) results in the maximum short-interval alarm interval.

14.3 Register Access Control

The hardware provides a collision-protection mechanism that prevents software from reading registers at the same time they are being updated by hardware, and vice versa.

14.3.1 Register Write Protection

The *HTMR_CTRL.busy* bit is a read-only status bit controlled by hardware and set when any of the following conditions occur:

- System Reset.
- Software writes to the *HTMR_SEC.rts*.
- Software modifies the *HTMR_CTRL.hten*, *HTMR_CTRL.ade*, or *HTMR_CTRL.ase* bits.

When the *HTMR_CTRL.busy* bit is set by hardware, writes to the above bits and count registers are blocked by hardware. The *HTMR_CTRL.busy* bit remains active until the register or bit is synchronized by hardware. The synchronization by hardware occurs on the next timer tick. The *HTMR_CTRL.busy* bit is set for a maximum of one timer tick. Therefore, a software write is not complete until hardware clears the *HTMR_CTRL.busy* bit.

Once the *HTMR_CTRL.busy* bit is cleared to 0, additional writes are completed as permitted by individual count or alarm-enable bits.

14.3.2 Register Read Protection

The *HTMR_CTRL.rdy* bit indicates when the count registers contain valid data. Hardware clears the *HTMR_CTRL.rdy* bit approximately one timer tick before the ripple occurs through the counter registers (*HTMR_SEC.rts* and *HTMR_SSEC*) and is set once again immediately after the ripple occurs. The period of the *HTMR_CTRL.rdy* bit set/clear activity provides a large window during which the counter registers are readable. Software can clear the *HTMR_CTRL.rdy* bit at any time and the bit remains clear until set by hardware when the next ripple occurs. A separate Ready Enable (*HTMR_CTRL.rdye*) bit is provided to generate an interrupt when hardware sets the *HTMR_CTRL.rdy* bit. You can use this interrupt to signal the start of a new timer read window.

14.3.3 Count Register Access

Values read from the count registers (*HTMR_SEC.rts* and *HTMR_SSEC*) are valid only when the *HTMR_CTRL.rdy* = 1.

To write the count registers, disable the timer by clearing (*HTMR_CTRL.hten*) to 0. Clearing the *HTMR_CTRL.hten* bit is permitted only when the Write Enable (*HTMR_CTRL.we*) bit is set to 1 and is governed by the *HTMR_CTRL.busy* bit signaling process (that is, the *HTMR_CTRL.busy* bit is 0). Writes to each count register must occur only when the *HTMR_CTRL.busy* bit reads 0.

14.3.4 Alarm Register Access

The alarm registers *HTMR_RAS* and *HTMR_RSSA* are readable at any time.

Set *HTMR_CTRL.ase* = 0 before writing to *HTMR_RSSA.rssa*. Set *HTMR_CTRL.ade* = 0 before writing to *HTMR_RAS*.

Clearing these bits requires monitoring the *HTMR_CTRL.busy* bit to assess completion of the write. Once the alarm is disabled, update the associated alarm registers using software.

14.4 Registers

See *Table 3-3* for the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 14-1: HTimer Registers Summary

Offset	Register	Description
[0x0000]	*HTMR_SEC*	HTimer Long-Interval Counter Register
[0x0004]	*HTMR_SSEC*	HTimer Short-Interval Counter Register
[0x0008]	*HTMR_RAS*	HTimer Long-Interval Alarm Register
[0x000C]	*HTMR_RSSA*	HTimer Short-Interval Alarm Register
[0x0010]	*HTMR_CTRL*	HTimer Control Register

14.5 Register Details

Table 14-2: HTimer Long-Interval Counter Register

HTimer Long-Interval Counter			HTMR_SEC	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	rts	R/W	-	**Long-Interval Counter** This register increments each time *HTMR_SSEC.rtss* rolls over.

Table 14-3: HTimer Short-Interval Counter Register

HTimer Short-Interval Counter			HTMR_SSEC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	R/W	0	**Reserved**
11:0	rtss	R/W	-	**Short-Interval Counter** This register increments once per t_{HTCLK} tick.

Table 14-4: HTimer Long-Interval Alarm Register

HTimer Long-Interval Alarm			HTMR_RAS	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved for Future Use** Do not modify this field from its default value.
19:0	ras	R/W	0	**Long-Interval Alarm** Sets the long-interval alarm. A system interrupt is generated when *HTMR_SEC.rts[19:0]* matches *HTMR_RAS.ras*.

Table 14-5: HTimer Short-Interval Alarm Register

HTimer Short-Interval Alarm			HTMR_RSSA	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	rssa	R/W	0	**Short-Interval Alarm** Sets the starting and reload values for the short-interval alarm.

Table 14-6: HTimer Control Register

HTimer Control			HTMR_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field from its default value.
15	we	R/W	0	**Write Enable** Software must set this bit to 1 before writing to *HTMR_CTRL.hten*. 1: Writes to *HTMR_CTRL.hten* are allowed. 0: Writes to *HTMR_CTRL.hten* are ignored.
14:8	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
7	alsf	R/W	0	**Short-Interval Alarm Interrupt Flag** This flag is a wake-up source for the processor. 0: No short-interval alarm pending. 1: Short-interval interrupt pending.
6	aldf	R/W	0	**Long-Interval Alarm Interrupt Flag** This flag is a wake-up source for the processor. 0: No long-interval alarm interrupt pending. 1: Long-interval interrupt pending.

HTimer Control			HTMR_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
5	rdye	R/W	0	**Timer Ready Interrupt Enable** This interrupt flag is set when the *HTMR_CTRL.rdy* is set by hardware. 0: Interrupt disabled. 1: Interrupt enabled.
4	rdy	R/W0O	0	**Timer Ready** This bit is set to 1 by hardware when *HTMR_SEC.rts* is updated. Software can clear this bit at any time. Hardware automatically clears this bit just prior to updating the *HTMR_SEC.rts*, indicating the timer is busy. 0: *HTMR_SEC* register not updated. 1: *HTMR_SEC* register updated.
3	busy	RO	0	**Timer Busy Flag** This bit is set by hardware when changes to the registers are synchronized. The bit is automatically cleared by hardware when the synchronization is complete. Software should poll this field for 0 after changing registers to ensure the change is complete prior to making any other register changes. 0: Not busy. 1: Busy.
2	ase	R/W	0	**Short-Interval Alarm Interrupt Enable** Set this bit to 1 to enable the short-interval alarm interrupt. Check the *HTMR_CTRL.busy* flag after writing this bit to determine when the register synchronizations are complete. 0: Short-interval alarm interrupt disabled. 1: Short-interval alarm interrupt disabled.
1	ade	R/W	0	**Long-Interval Alarm Interrupt Enable** Set this bit to 1 to enable the long-interval alarm interrupt. Check the *HTMR_CTRL.busy* flag after writing to this bit to determine when the timer synchronization is complete. 0: Long-interval alarm interrupt is disabled. 1: Long-interval alarm interrupt is enabled.
0	hten	R/W	0	**HT Enable** Enables and disables the timer. Software must write *HTMR_CTRL.we* = 1 before changing this field. *HTMR_CTRL.busy* must read 0 before writing to this bit. After writing to this bit, check the *HTMR_CTRL.busy* flag for 0 to determine when the HT synchronization is complete. 0: HT disabled. 1: HT enabled.

15. Timers

Multiple 32-bit, reloadable timers are provided. Each timer provides multiple operating modes:

- One-Shot: Timer counts up to terminal value then halts.
- Continuous: Timer counts up to terminal value then repeats.
- Counter: Timer counts input edges received on timer input pin.
- Pulse Width Modulated (PWM).
- Capture: Captures a snapshot of the current timer count when timer input edge transitions.
- Compare: Timer pin toggles when timer exceeds terminal count.
- Gated: Timer increments only when timer input pin is asserted.
- Capture/Compare: Timer counts when timer input is asserted, captures timer count when input is deasserted.

The MAX32665/MAX32666 provide six instances of the timer peripheral (TMR0, TMR1, TMR2, TMR3, TMR4, TMR5).

15.1 Features

- 32-bit reload counter
- Programmable prescaler with values from 1 to 4096
- Non-overlapping PWM output generation with configurable off-time
- Capture, compare, and capture/compare capability
- Timer pin available as alternate function
- Configurable Input pin for event triggering, clock gating, or capture signal
- Timer output pin for event output and PWM signal generation
- Independent interrupt

15.2 Basic Operation

The timer modes operate by incrementing the *TMRn_CNT* register, driven by either the timer clock, an external stimulus on the timer pin, or a combination of both. The *TMRn_CNT* register is always readable, even while the timer is enabled and counting.

Each timer mode has a user-configurable timer period, which terminates on the timer clock cycle following the end of timer period condition. Each timer mode has a different response at the end of a timer period, which can include changing the state of the timer pin, capturing a timer value, reloading *TMRn_CNT* with a new starting value, or disabling the counter. The end of a timer period will always set the corresponding interrupt bit and can generate an interrupt, if enabled.

In most modes the timer peripheral automatically sets *TMRn_CNT* to 0x0000 0001 at the end of a timer period, but *TMRn_CNT* is set to 0x0000 0000 following a system reset. This means the first timer period following a system reset will be one timer clock longer than subsequent timer periods if *TMRn_CNT* is not initialized to 0x0000 0001 during the timer configuration step.

Clocking of timer functions is driven by the timer clock frequency, f_{CNT_CLK}. The timer clock frequency is a user-configurable, division of the system peripheral clock, PCLK. Each timer has an independent prescaler, allowing timers to operate at different frequencies. The prescaler can be set from 1 to 4096 using the *TMRn_CN.pres3*:*TMRn_CN.pres* fields. Unless otherwise mentioned, the timer clock is generated as follows:

Equation 15-1*: Timer Peripheral Clock Equation*

$$f_{CNT\_CLK} = \frac{f_{PCLK}}{prescaler}$$

Software writes to the timer registers and external events on timer pins will be asynchronous events to the slower timer clock frequency. These events are latched on the next rising edge of the timer clock. Since it is not possible to observe the timer clock directly, input events may have up to 0.5 timer clock delay before being recognized.

15.3 Timer Pin Functionality

Most timers have an associated timer pin that can function as an optional input or output depending on the selected timer mode. The timer pin functionality is mapped as an alternate function that is shared with a GPIO. Timer pin assignments are detailed in the data sheet for the specific device.

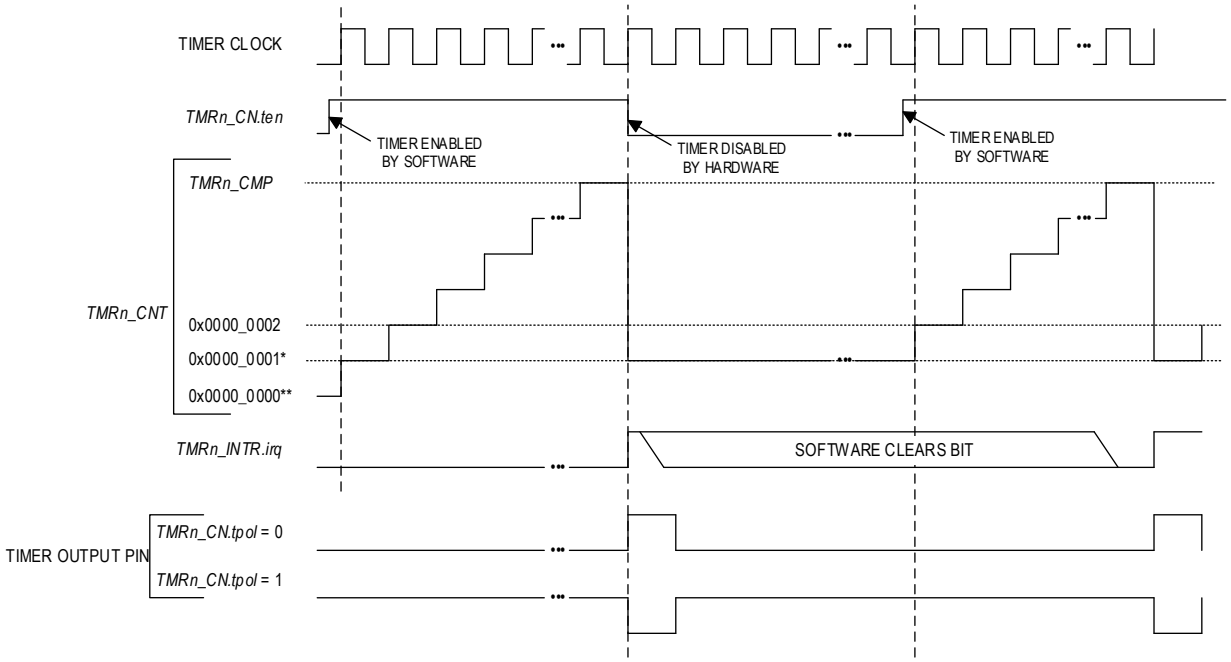
When the timer pin alternate function is enabled, the timer pin will have the same electrical characteristics, such as pullup/pulldown strength, drive strength, etc. as the GPIO mode settings for that pin. When configured as an output, the corresponding bit in the GPIO_OUT register should be configured to match the inactive state of the timer pin for that mode. The pin characteristics must be configured before enabling the timer. Consult the GPIO section for details on how to configure the electrical characteristics for the pin.

Each timer has a dedicated interrupt flag, *TMRn_INTR.irq*, which is set at the end of a timer period. If enabled, an interrupt will be generated. The interrupt flag can be cleared by writing any value to *TMRn_INTR.irq*.

15.4 One-Shot Mode (0)

In One-Shot mode the timer peripheral increments *TMRn_CNT* until it matches *TMRn_CMP* and then stops incrementing and disables the timer. The timer can optionally output a pulse on the timer pin at the end of the timer period. In this mode, the timer must be re-enabled to start another one-shot mode event.

Figure 15-1: One-Shot Mode Diagram



* TMRn_CNT defaults to 0x00000000 on a timer reset. TMRn_CNT reloads to 0x00000001 for all following timer periods

15.4.1 One-Shot Mode Timer Period

The timer period ends on the timer clock following *TMRn_CNT* = *TMRn_CMP*.

The timer peripheral automatically performs the following actions at the end of the timer period:

1. *TMRn_CNT* is reset to 0x0000 0001.

2. The timer is disabled by setting *TMRn_CN.ten* = 0.
3. If the timer output is enabled, the timer pin is driven to its active state for one timer clock. It then returns to its inactive state.
4. The timer interrupt bit *TMRn_INTR.irq* will be set. An interrupt will be generated if enabled.

15.4.2 One-Shot Mode Configuration

Configure the timer for One-Shot mode by doing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer. Set *TMRn_CN.tmode* to 0 to select One-Shot mode.
2. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
3. If using the timer pin:
   a. Configure the pin as a timer output and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired (inactive) state.
4. If using the timer interrupt, enable the interrupt and set the interrupt priority.
5. Write an initial value to *TMRn_CNT*, if desired. This effects only the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
6. Write the compare value to *TMRn_CMP*.
7. Set *TMRn_CN.ten* = 1 to enable the timer.

The timer period is calculated using the following equation:

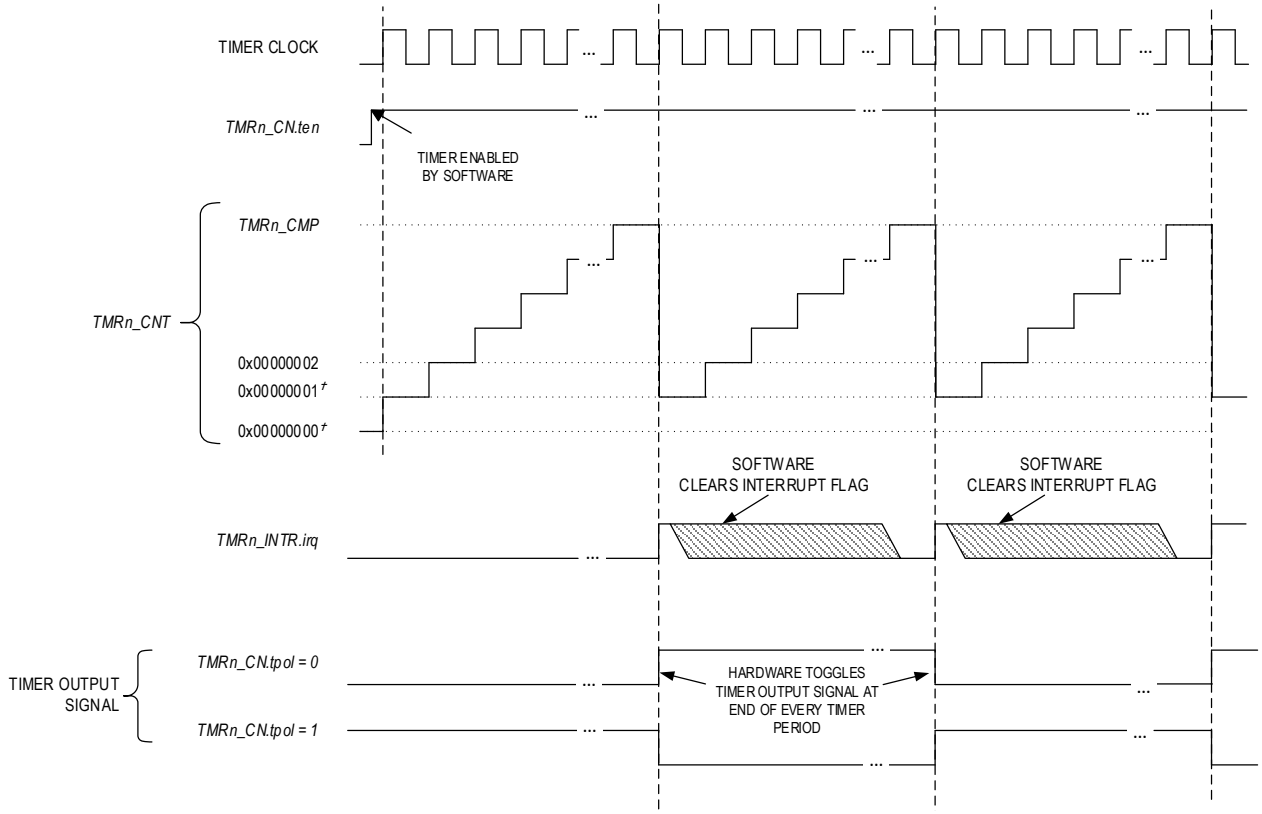
Equation 15-2*: One-Shot Mode Timer Period*

$$One\text{-}Shot\ mode\ timer\ period\ in\ seconds = \frac{TMR\_CMP - TMR\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

15.5 Continuous Mode (1)

In Continuous mode, the timer peripheral increments *TMRn_CNT* until it matches *TMRn_CMP*, resets *TMRn_CNT* to 0x0000 0001, and continues incrementing. The timer peripheral can optionally toggle the state of the timer pin at the end of the timer period.

Figure 15-2: Continuous Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CN.tmode = 1 (Continuous)

† *TMRn_CNT* defaults to 0x00000000 on a timer reset. *TMRn_CNT* reloads to 0x00000001 for all following timer periods.

15.5.1 Continuous Mode Timer Period

The timer period ends on the timer clock following *TMRn_CNT* = *TMRn_CMP*.

The timer peripheral automatically performs the following actions at the end of the timer period:

1. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing.
2. If the timer output is enabled, the timer pin toggles state (low to high or high to low).
3. The timer interrupt bit *TMRn_INTR.irq* will be set. An interrupt will be generated if enabled.

15.5.2 Continuous Mode Configuration

Configure the timer for Continuous mode by performing the steps following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 1 to select Continuous mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency, f_{CNT_CLK}.
4. If using the timer pin:
   a. Configure the pin as a timer output and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired inactive state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.

6. Write an initial value to *TMRn_CNT*, if desired. The initial value is only used for the first period; subsequent timer periods always reset the *TMRn_CNT* register to 1.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* to 1 to enable the timer.

The Continuous Mode Timer Period is calculated using *Equation 15-3*.

Equation 15-3*: Continuous Mode Timer Period*

$$Continuous\ mode\ timer\ period\ in\ seconds = \frac{TMRn\_CMP - TMRn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

15.6 Counter Mode (2)

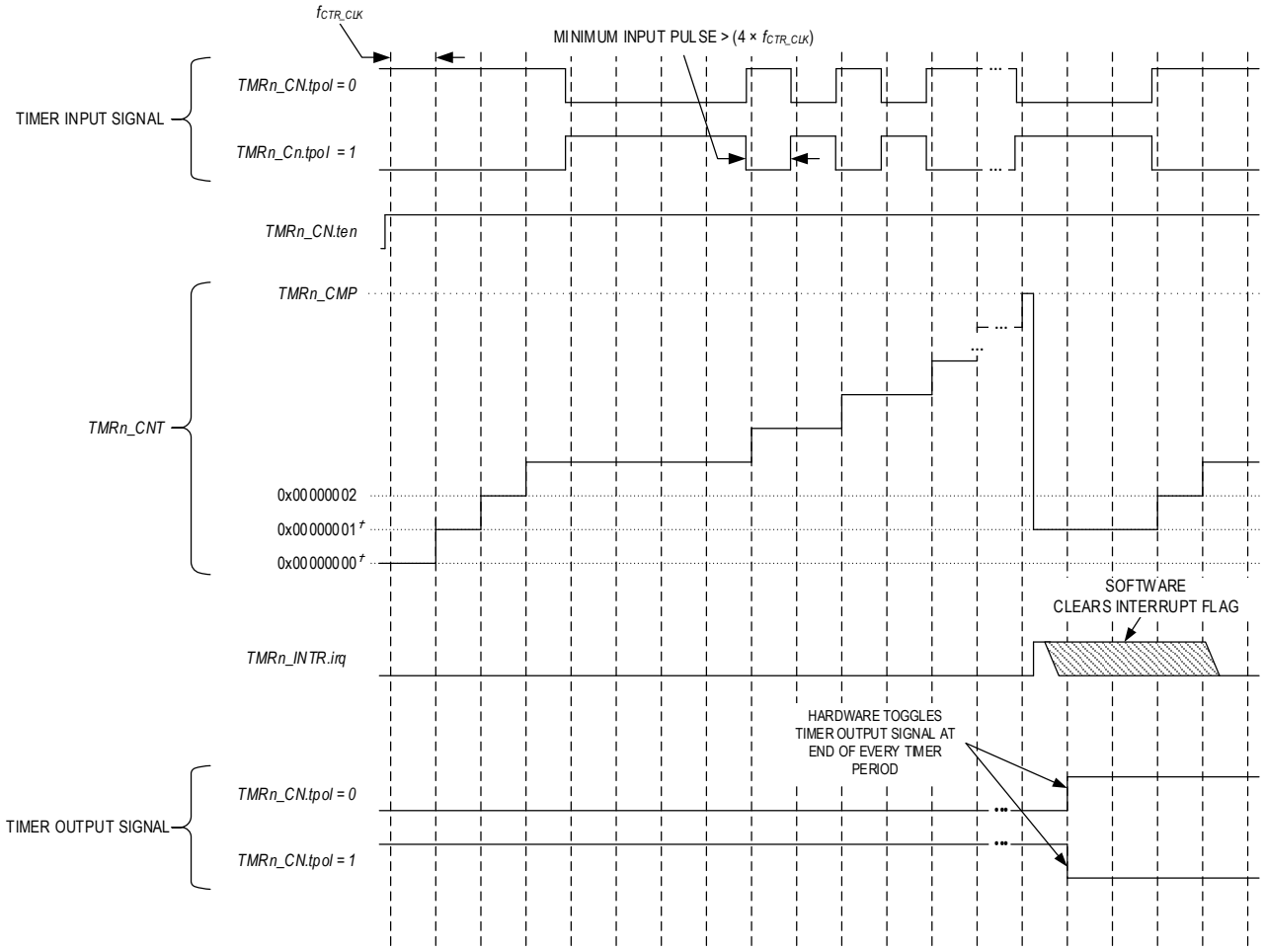
In Counter mode, the timer peripheral increments *TMRn_CNT* when a transition occurs on the timer pin. When *TMRn_CNT* = *TMRn_CMP*, the interrupt bit is set and the *TMRn_CNT* register is set to 0x0000 0001 and continues incrementing. The timer can be configured to increment on either the rising edge or the falling edge, but not both.

The timer prescaler setting has no effect in this mode. The frequency of the timer's input signal (f_{CTR_CLK}) must not exceed 25 percent of the PCLK frequency as shown in the following equation:

Equation 15-4: Counter Mode Maximum Clock Frequency

$$f_{CTR\_CLK} \leq \frac{f_{PCLK}\ (Hz)}{4}$$

Figure 15-3: Counter Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CN.tmode = 2 (Counter)

† *TMRn_CNT* defaults to 0x00000000 on a timer reset. *TMRn_CNT* reloads to 0x00000001 for all following timer periods.

15.6.1 Counter Mode Timer Period

The timer period ends on the rising edge of PCLK following *TMRn_CNT* = *TMRn_CMP*.

The timer peripheral automatically performs the following actions at the end of the timer period:

1. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing on selected transitions of the timer pin.
2. The timer interrupt bit *TMRn_INTR.irq* will be set. An interrupt will be generated if enabled.

15.6.2 Counter Mode Configuration

Configure the timer for Counter mode by doing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.

2. Set *TMRn_CN.tmode* to 2 to select Counter mode.
3. Configure the timer pin:
   a. Configure the pin as a timer input and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired initial (inactive) state.
4. If using the timer interrupt, enable the interrupt and set the interrupt priority.
5. Write an initial value to *TMRn_CNT*, if desired. The initial value is only used for the first timer period; subsequent timer periods always reset *TMRn_CNT* to 1.
6. Write the compare value to *TMRn_CMP*.
7. Set *TMRn_CN.ten* = 1 to enable the timer.

In Counter mode, the number of timer input transitions since timer start is calculated using the following equation:

Equation 15-5: Counter Mode Timer Input Transitions

$$Counter\ mode\ timer\ input\ transitions = TMR\_CNT_{CURRENT\_VALUE} - TMR\_CNT_{INITIAL\_VALUE}$$

15.7 PWM Mode (3)

In PWM mode, the timer sends a Pulse-Width Modulated (PWM) output using the timer's output signal. The timer first counts up to the match value stored in the *TMRn_PWM* register. At the end of the cycle where the *TMRn_CNT* value matches the *TMRn_PWM* value, the timer's output toggles state. The timer continues counting until it reaches the *TMRn_CMP* value.

15.7.1 PWM Mode Timer Period

The timer period ends on the rising edge of PCLK following *TMRn_CNT* = *TMRn_CMP*.

The timer peripheral automatically performs the following actions at the end of the timer period:

1. The *TMRn_CNT* is reset to 0x0000 0001, and the timer resumes counting.
2. The timer output signal is toggled.
3. The timer interrupt bit *TMRn_INTR.irq* will be set. An interrupt will be generated if enabled.

When *TMRn_CN.tpol* = 0, the timer output signal starts low and then transitions to high when the *TMRn_CNT* value matches the *TMRn_PWM* value. The timer output signal remains high until the *TMRn_CNT* value reaches the *TMRn_CMP* value, resulting in the timer output signal transitioning low, and the *TMRn_CNT* value resetting to 0x0000 0001.

When *TMRn_CN.tpol* = 1, the Timer output signal starts high and transitions low when the *TMRn_CNT* value matches the *TMRn_PWM* value. The timer output signal remains low until the *TMRn_CNT* value reaches the *TMRn_CMP* value, resulting in the timer output signal transitioning high, and the *TMRn_CNT* value resetting to 0x0000 0001.

15.7.2 PWM Mode Configuration

Complete the following steps to configure a timer for PWM mode and initiate the PWM operation:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 3 to select PWM mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. Configure the timer pin:
5. Configure the pin as a timer input and configure the electrical characteristics as needed.
6. Set *TMRn_CN.tpol* to match the desired initial (inactive) state.
   a. Set *TMRn_CN.tpol* to select the initial logic level (high or low) and PWM transition state for the timer's output.
   b. Set *TMRn_CNT* to the starting count, typically 0x0000 0001. The initial *TMRn_CNT* value only effects the initial period in PWM mode with subsequent periods always setting *TMRn_CNT* to 0x0000 0001.

c. Set the *TMRn_PWM* value to the transition period count.

7. Set the *TMRn_CMP* value for the PWM second transition period. Note: *TMRn_CMP* must be greater than the *TMRn_PWM* value.
8. Optionally enable the timer's interrupt in the Interrupt Controller, and set the timer's interrupt priority.
9. Set *TMRn_CN.ten* to 1 to enable the timer and start the PWM.

The PWM period is calculated using the following equation:

Equation 15-6: Timer PWM Period

$$PWM\ period\ in\ seconds = \frac{TMR\_CNT}{f_{CNT\_CLK}\ (Hz)}$$

If an initial starting value other than 0x0000 0001 is loaded into the *TMRn_CNT* register, use the One-Shot mode equation, *Equation 15-2*, to determine the initial PWM period.

If *TMRn_CN.tpol* is 0, the ratio of the PWM output high time to the total period is calculated using *Equation 15-7*.

Equation 15-7: Timer PWM Output High Time Ratio with Polarity 0

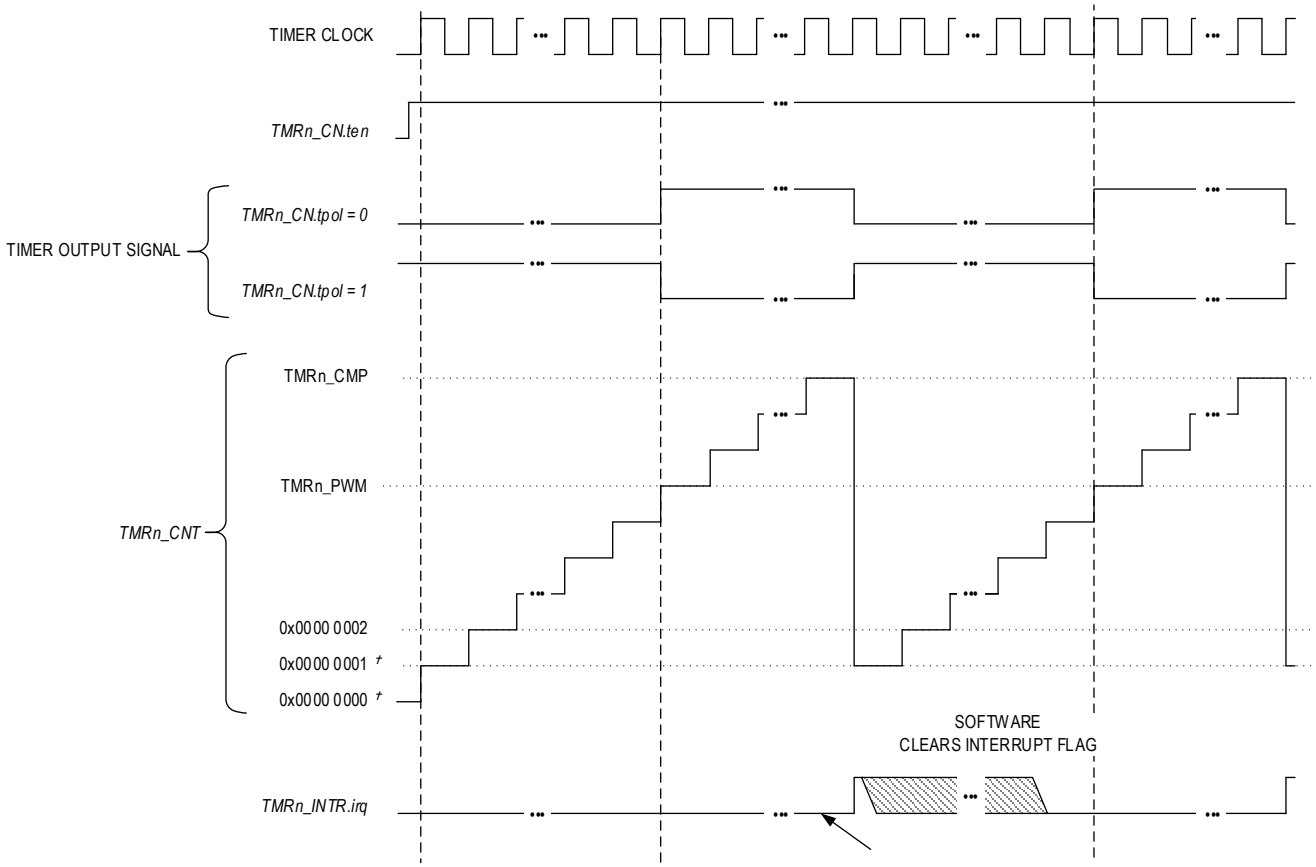
$$PWM\ output\ high\ time\ ratio\ (\%) = \frac{(TMR\_CMP - TMR\_PWM)}{TMR\_CMP} \times 100$$

If *TMRn_CN.tpol* is set to 1, the ratio of the PWM output high time to the total period is calculated using *Equation 15-8*.

Equation 15-8: Timer PWM Output High Time Ratio with Polarity 1

$$PWM\ output\ high\ time\ ratio\ (\%) = \frac{TMR\_PWM}{TMR\_CMP} \times 100$$

Figure 15-4: PWM Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CN.tmode = 3 (PWM)

† *TMRn_CNT* defaults to 0x00000000 on a timer reset. *TMRn_CNT* reloads to 0x00000001 for all following timer periods.

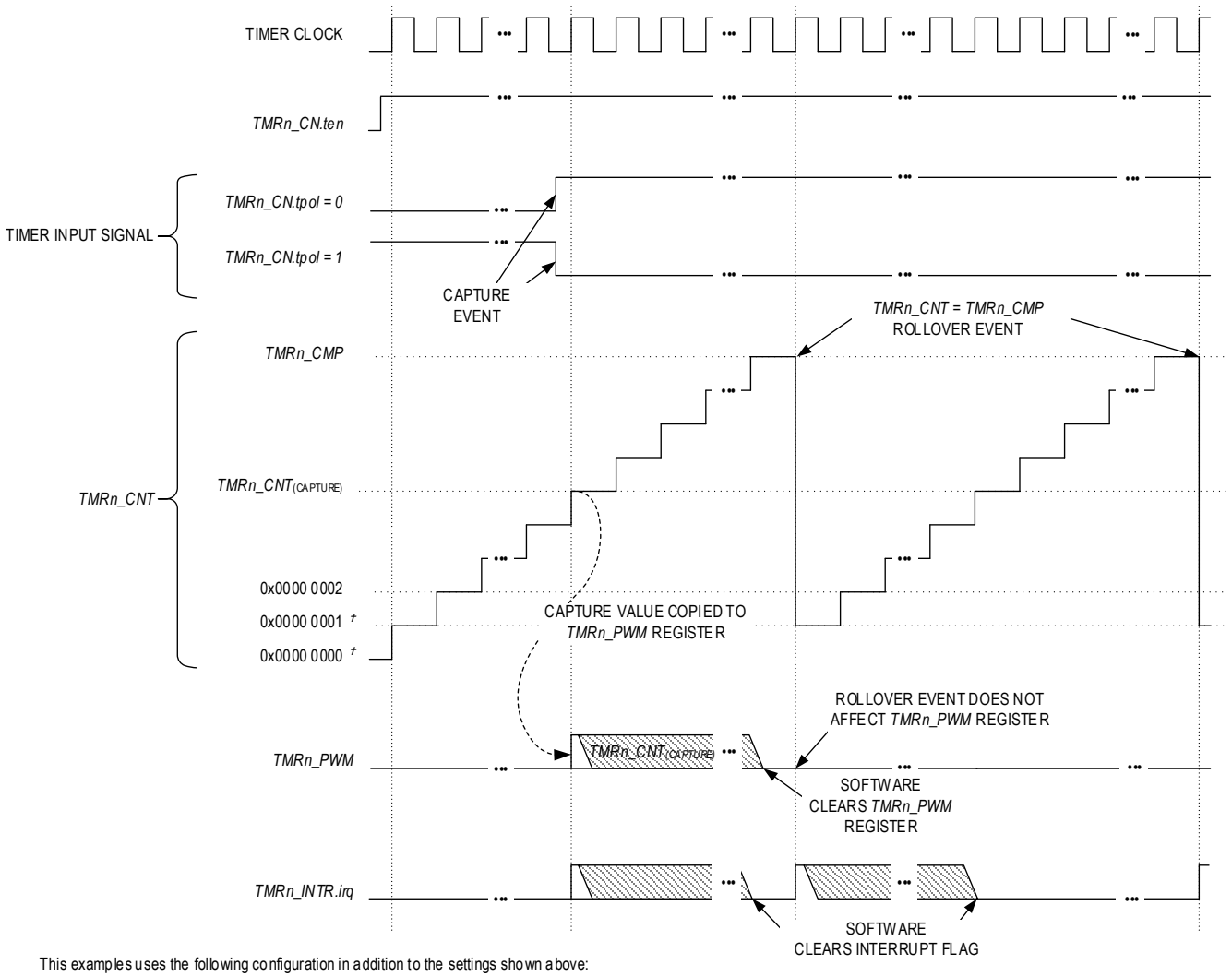
15.8 Capture Mode (4)

Capture mode is used to measure the time between application determined events. The timer starts incrementing the timer's count field until a transition occurs on the timer's input pin or a rollover event occurs. A capture event is triggered by hardware when the timer's input pin transitions state. When a capture event occurs, the timer hardware copies the timer's count value (*TMRn_CNT*) to the timer's PWM register (*TMRn_PWM*).

If a capture event does not occur prior to the timer's count value reaching the timer's compare value (*TMRn_CNT* = *TMRn_CMP*), a rollover event occurs. Both the capture event and the rollover event set the timer's interrupt flag, *TMRn_INTR.irq*, to 1 and result in an interrupt if the timer's interrupt is enabled.

A capture event can occur before or after a rollover event. Software must track the number of rollover events that occur prior to a capture event to determine the elapsed time of the capture event. When a capture event occurs, software should reset the count of rollover events.
Note: A capture event does not stop the timer's counter from incrementing and does not reset the timer's count value; a rollover event still occurs when the timer's count value reaches the timer's compare value.

Figure 15-5: Capture Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CN.tmode = 4 (Capture)

† *TMRn_CNT* defaults to 0x00000000 on a timer reset. *TMRn_CNT* reloads to 0x00000001 for all following timer periods.

15.8.1 Capture Mode Timer Period

Two timer period events are possible in Capture Mode:

The Capture event occurs on the timer clock following the selected transition on the timer pin. The timer peripheral automatically performs the following actions:

1. The value in *TMRn_CNT* is copied to *TMRn_PWM*
2. The timer interrupt bit *TMRn_INTR.irq* will be set. An interrupt will be generated if enabled.
3. The timer remains enabled and continues incrementing.
4. The timer period ends on the timer clock following *TMRn_CNT* = *TMRn_CMP*.

The timer period event occurs on the timer clock *TMRn_CNT* = *TMRn_CMP*, a roll-over event. The timer peripheral automatically performs the following actions when an end of timer period event occurs:

1. The value in *TMRn_CNT* is reset to 0x0000 00001. The timer remains enabled and continues incrementing.
2. The timer interrupt bit *TMRn_INTR.irq* will be set. An interrupt will be generated if enabled.

15.8.2 Capture Mode Configuration

Configure the timer for Capture mode by doing the following:

1. Disable the timer by setting *TMRn_CN.ten* to 0.
2. Select Counter mode by setting *TMRn_CN.tmode* to 4.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. If using the timer pin:
   a. Configure the pin as a timer output and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired (inactive) state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write the initial value to *TMRn_CNT*. This effects only the first period; subsequent periods always begin with 0x0000 0001.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* = 1 to enable the timer.

The timer period is calculated using the following equation:

Equation 15-9: Capture Mode Elapsed Time Calculation in Seconds

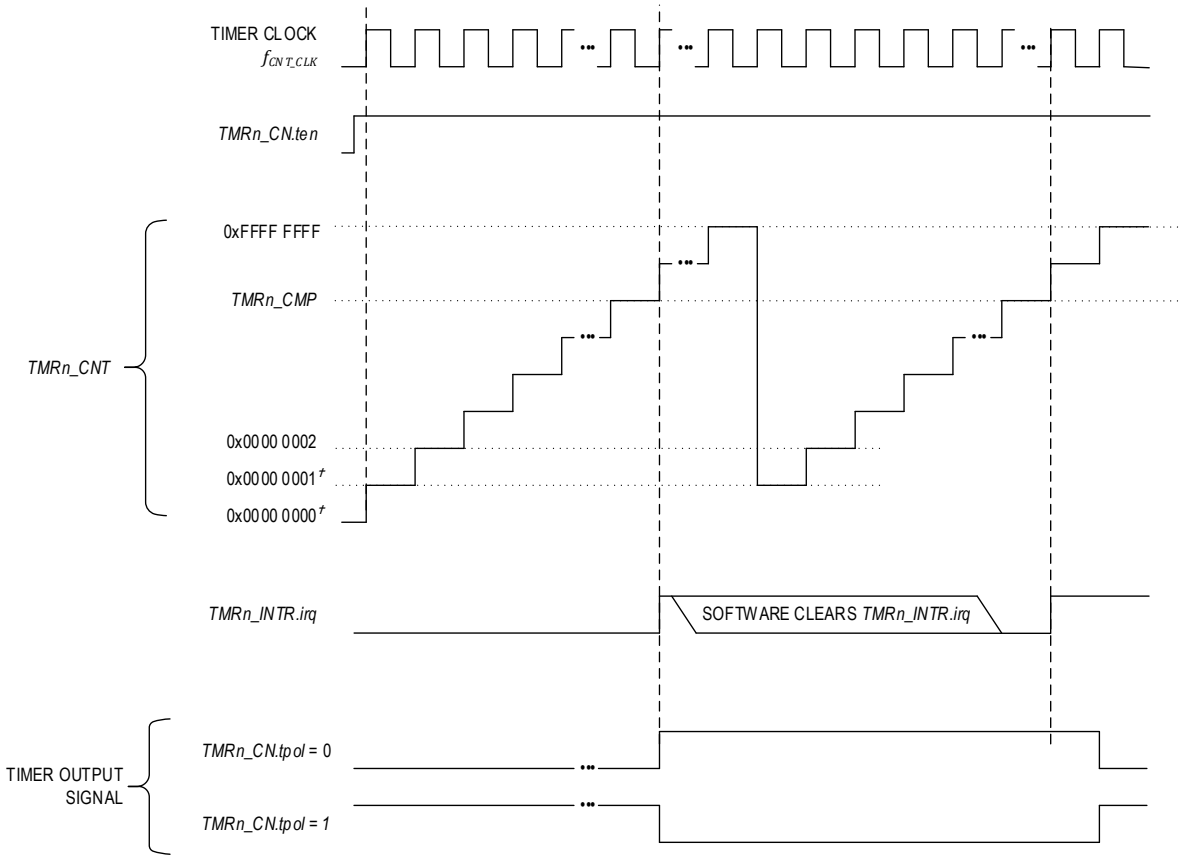
$$Capture\ elapsed\ time = \frac{(TMRn\_PWM - TMRn\_CNT) + \left((Number\ of\ rollover\ events) \times \left(TMRn\_CMP - TMR\_CNT_{INITIAL\_VALUE}\right)\right)}{f_{CNT\_CLK}}$$

Note: The capture elapsed time calculation is only valid after the capture event occurs, and the timer stores the captured count in the TMRn_PWM *register.*

15.9 Compare Mode (5)

In Compare mode the timer peripheral increments continually, allowing the timer to be a programmable 32-bit programmable period timer. The end of timer period event occurs when the timer value matches the compare value, but the timer continues to increment until the count reaches 0xFFFF FFFF. The timer counter then rolls over and continues counting from 0x0000 0000.

Figure 15-6: Counter Mode Diagram



This example uses the following configuration in addition to the settings shown above:
TMRn_CN.tmode = 5 (Compare)

† *TMRn_CNT* defaults to 0x00000000 on a timer reset. *TMRn_CNT* reloads to 0x00000001 for all following timer periods.

15.9.1 Compare Mode Timer Period

The timer period ends on the timer clock following *TMRn_CNT* = *TMRn_CMP*.

The timer peripheral automatically performs the following actions at the end of the timer period:

1. The timer remains enabled and continues incrementing. Unlike other modes, *TMRn_CNT* is not reset to 0x0000 0001 at the end of the timer period.
2. If the timer output is enabled, then the timer pin toggles state (low to high or high to low).
3. The timer interrupt bit *TMRn_INTR.irq* will be set. An interrupt is generated if enabled.

15.9.2 Compare Mode Configuration

Configure the timer for Compare mode by doing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 5 to select Compare mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.

4. If using the timer pin:
   a. Configure the pin for the timer output alternate function and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired (inactive) state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write the initial value to *TMRn_CNT*. This effects only the first period as the counter increments continuously, rolling over to 0x0000 0000 and continuing.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* = 1 to enable the timer.

The Compare Mode timer period is calculated using *Equation 15-10*.

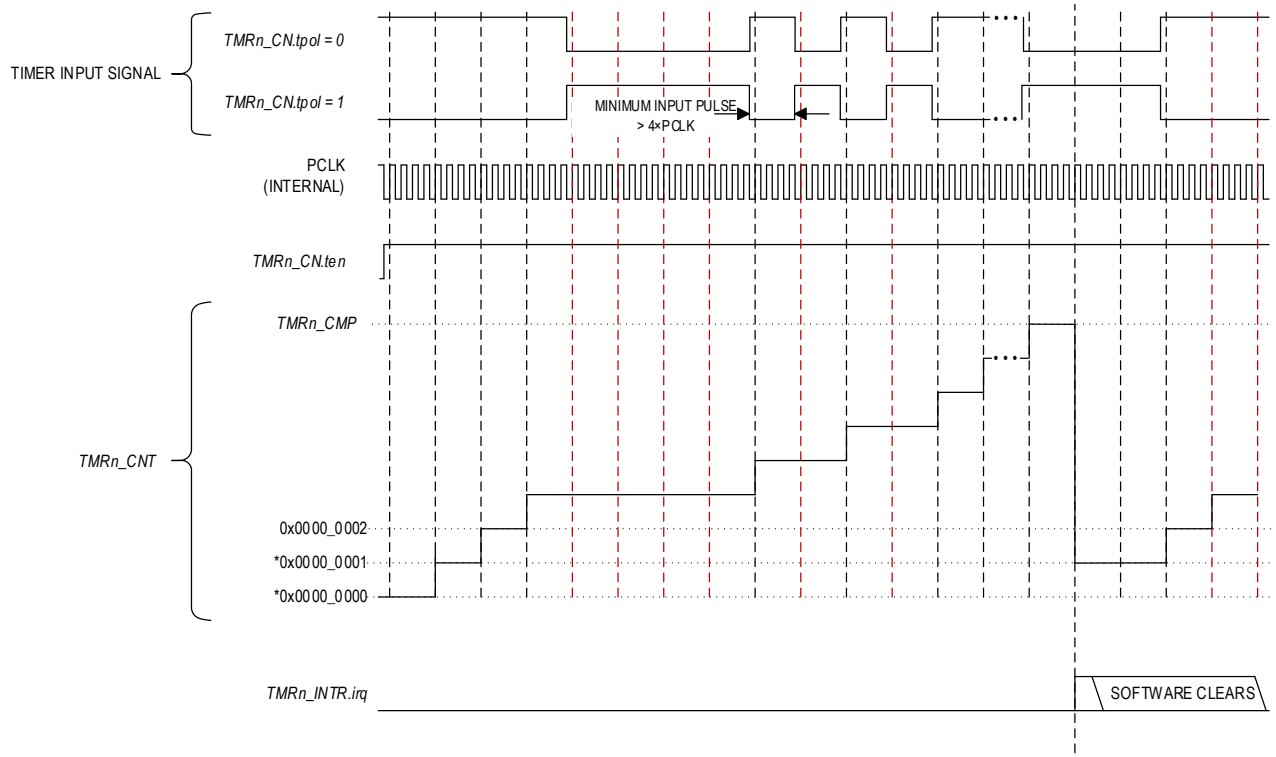
Equation 15-10: Compare Mode Timer Period

$$Compare\ mode\ timer\ period\ in\ seconds = \frac{TMR\_CMP - TMR\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

15.10 Gated Mode (6)

Gated mode is similar to continuous mode, except that *TMRn_CNT* only increments when the timer pin is in its active state.

Figure 15-7: Gated Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CN.tmode = 6 (Gated)

† *TMRn_CNT* defaults to 0x00000000 on a timer reset. *TMRn_CNT* reloads to 0x00000001 for all following timer periods.

15.10.1 Gated Mode Timer Period

The timer period ends when *TMRn_CNT* = *TMRn_CMP* and the timer automatically performs the following actions:

1. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing.
2. The timer interrupt bit *TMRn_INTR.irq* will be set. An interrupt will be generated if enabled.

15.10.2 Gated Mode Configuration

Configure the timer for Gated mode by doing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 6 to select Gated mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. Configure the timer pin:
   a. Configure the pin as a timer input and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired initial (inactive) state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write an initial value to *TMRn_CNT*, if desired. This effects only the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* = 1 to enable the timer.

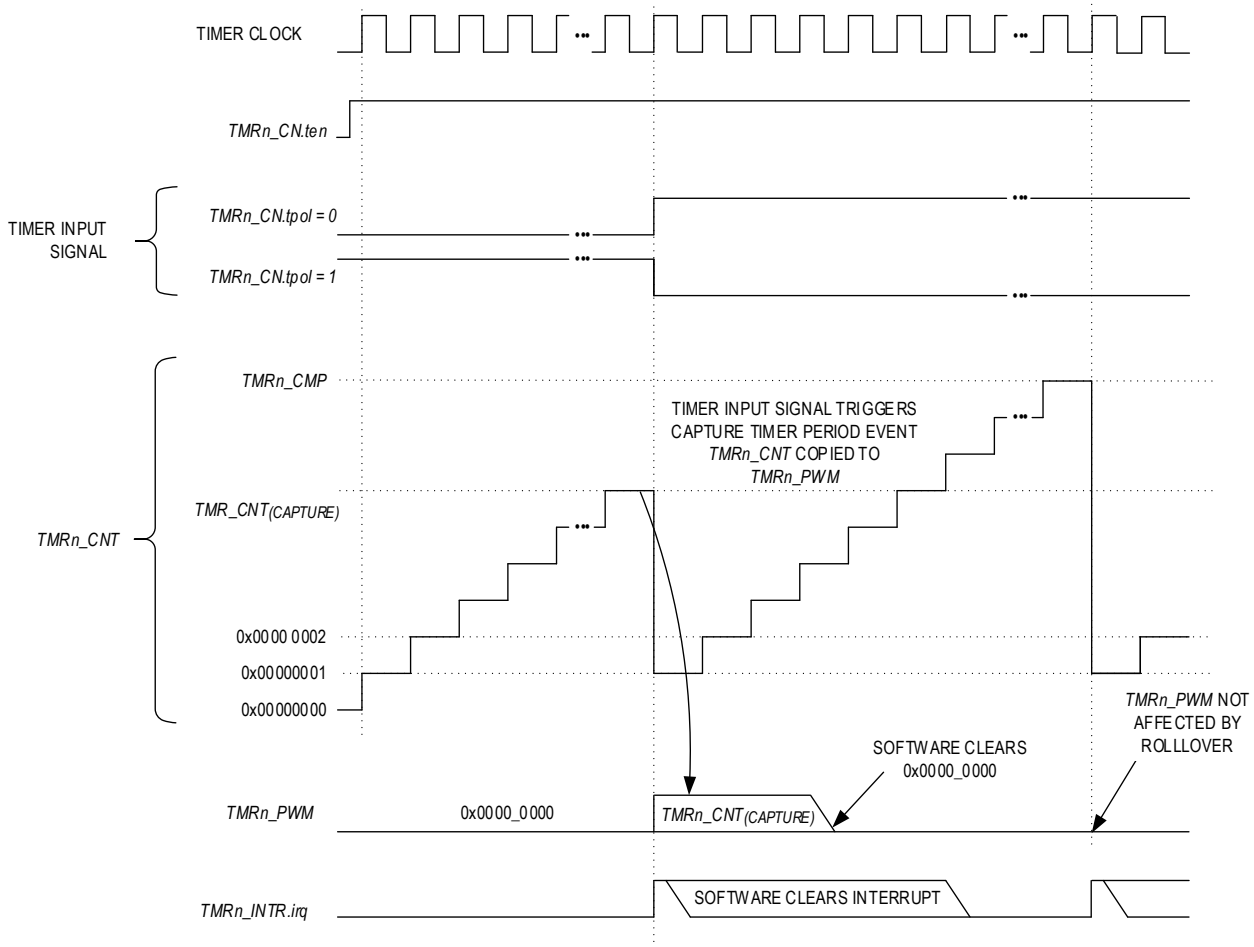
15.11 Capture/Compare Mode (7)

In Capture/Compare mode, the timer starts counting after the first external timer input transition occurs. The transition, a rising edge or falling edge on the timer's input signal, is set using the *TMRn_CN.tpol* bit.

Each subsequent transition, after the first transition of the timer input signal, captures the *TMRn_CNT* value, writing it to the *TMRn_PWM* register (capture event). When a capture event occurs, a timer interrupt is generated, the *TMRn_CNT* value is reset to 0x0000 0001, and the timer resumes counting.

If no capture event occurs, the timer counts up to the *TMRn_CMP* value. At the end of the cycle where the *TMRn_CNT* equals the *TMRn_CMP* value, a timer interrupt is generated, the *TMRn_CNT* value is reset to 0x0000 0001, and the timer resumes counting.

Figure 15-8: Capture/Compare Mode Diagram



This example uses the following configuration in addition to the settings shown above:
TMRn_CN.tmode = 7 (Capture/Compare)

† *TMRn_CNT* defaults to 0x00000000 on a timer reset. *TMRn_CNT* reloads to 0x00000001 for all following timer periods.

15.11.1 Capture/Compare Timer Period

The timer period ends when the selected transition occurs on the timer pin, or on the clock cycle following *TMRn_CNT* = *TMRn_CMP*.

The timer peripheral automatically performs the following actions at the end of the timer period:

If the end of the timer period was caused by a transition on the timer pin:

1. The value in *TMRn_CNT* is copied to *TMRn_PWM*.
2. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing.
3. The timer interrupt bit, *TMRn_INTR.irq*, is set. If the timer's interrupt is enabled a Timer IRQ is generated automatically.
4. If the end of the timer period was caused by a transition on the timer pin:
5. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing..
6. The timer interrupt bit *TMRn_INTR.irq* will be set. An interrupt will generated if enabled.

15.11.2 Capture/Compare Configuration

Configure the timer for Capture/Compare mode by doing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 7 to select Capture/Compare mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. Configure the timer pin:
   a. Configure the pin as a timer input and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to select the positive edge (*TMRn_CN.tpol* = 0) or negative edge (*TMRn_CN.tpol* = 0) transition causes the capture event.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write an initial value to *TMRn_CNT*, if desired. This effects only the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
7. Set *TMRn_CN.ten* to 1 to enable the timer. Counting starts after the first transition of the timer's input signal.

Note: No interrupt is generated by the first transition of the input signal.

In Capture/Compare mode, the elapsed time from the timer start to the capture event is calculated using *Equation 15-11*.

Equation 15-11*: Capture Mode Elapsed Time*

$$Capture\ elapsed\ time\ in\ seconds = \frac{TMR\_PWM - TMR\_CNT_{INITIAL\_CNT\_VALUE}}{f_{CNT\_CLK}\ (Hz)}$$

15.12 Registers

See *Table 3-3* for the the base address of this peripheral/module. Each instance has its own, independent set of the registers shown in *Table 15-1*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 15-1: Timer Register Summary

Offset	Register Name	Description
[0x0000]	*TMRn_CNT*	Timer Counter Register
[0x0004]	*TMRn_CMP*	Timer Compare Register
[0x0008]	*TMRn_PWM*	Timer PWM Register
[0x000C]	*TMRn_INTR*	Timer Interrupt Register
[0x0010]	*TMRn_CN*	Timer Control Register
[0x0014]	*TMRn_NOLCMP*	Timer Non-Overlapping Compare Register

15.13 Register Details

Table 15-2: Timer Count Registers

Timer Count Register			TMRn_CNT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	count	R/W	0	**Timer Count Value** The current count value for the timer. This field increments as the timer counts. Reads of this register are always valid. Prior to writing this field, disable the timer by clearing bit *TMRn_CN.ten*.

Table 15-3: Timer Compare Registers

Timer Compare Register			TMRn_CMP	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	compare	R/W	0	**Timer Compare Value** The value in this register is used as the compare value for the timer's count value. The compare field meaning is determined by the specific mode of the timer. See the timer mode's detailed configuration section for *compare* usage and meaning.

Table 15-4: Timer PWM Registers

Timer PWM Register			TMRn_PWM	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	pwm	R/W	0	**Timer PWM Match** In PWM mode, this field sets the count value for the first transition period of the PWM cycle. At the end of the cycle where *TMRn_CNT* equals *TMRn_CMP*, the PWM output transitions to the second period of the PWM cycle. The second PWM period count is stored in the *TMRn_CMP* register. The value set for *TMRn_PWM* must be less than the value set in *TMRn_CMP* for PWM mode operation. **Timer Capture Value** In Capture, Compare, and Capture/Compare modes, this field is used to store the *TMRn_CNT* value when a Capture, Compare, or Capture/Compare event occurs.

Table 15-5: Timer Interrupt Registers

Timer Interrupt Register			TMRn_INTR	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
0	irq_clr	R/W	0	**Timer Interrupt** If set, this field indicates a timer interrupt condition occurred. Writing any value to this bit clears the timer's interrupt. 0: Timer interrupt is not active. 1: Timer interrupt occurred.

Table 15-6: Timer Control Registers

Timer Control Register			TMRn_CN	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:13	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
12	pwmckbd	RO	1	**Reserved**
11	nollpol	RO	0	**Reserved**
10	nolhpol	RO	0	**Reserved**
9	pwmsync	RO	0	**Reserved**
8	pres3	R/W	0	**Timer Prescale Select MSB** See *TMRn_CN.pres* for details on this field's usage.
7	ten	R/W	0	**Timer Enable** 1: Timer enabled 0: Timer disabled

<table>
<tr><th colspan="3">Timer Control Register</th><th>TMRn_CN</th><th>[0x0010]</th></tr>
<tr><th>Bits</th><th>Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td>6</td><td>tpol</td><td>R/W</td><td>0</td><td>Timer Polarity
Selects the polarity of the timer's input and output signal. This setting is not used if the GPIO is not configured for the alternate function. The tpol field meaning is determined by the specific mode of the timer. See the mode's detailed configuration section for tpol usage.</td></tr>
<tr><td>5:3</td><td>pres</td><td>R/W</td><td>0</td><td>Timer Prescaler Select
Sets the timer's prescaler value. The prescaler divides the PCLK input to the timer and sets the timer's count clock, $f_{CNT\_CLK} = {PCLK\ (Hz)}/{prescaler}$. The timer's prescaler setting is a 4-bit value with pres3 as the most significant bit and pres as the three least significant bits. The table below shows the prescaler values based on pres3:pres.
<table>
<tr><th>pres3</th><th>pres</th><th>Prescaler</th><th>f_{CNT_CLK}</th></tr>
<tr><td>0</td><td>0b000</td><td>1</td><td>$f_{PCLK}\ (Hz)/_{1}$</td></tr>
<tr><td>0</td><td>0b001</td><td>2</td><td>$f_{PCLK}\ (Hz)/_{2}$</td></tr>
<tr><td>0</td><td>0b010</td><td>4</td><td>$f_{PCLK}\ (Hz)/_{4}$</td></tr>
<tr><td>0</td><td>0b011</td><td>8</td><td>$f_{PCLK}\ (Hz)/_{8}$</td></tr>
<tr><td>0</td><td>0b100</td><td>16</td><td>$f_{PCLK}\ (Hz)/_{16}$</td></tr>
<tr><td>0</td><td>0b101</td><td>32</td><td>$f_{PCLK}\ (Hz)/_{32}$</td></tr>
<tr><td>0</td><td>0b110</td><td>64</td><td>$f_{PCLK}\ (Hz)/_{64}$</td></tr>
<tr><td>0</td><td>0b111</td><td>128</td><td>$f_{PCLK}\ (Hz)/_{128}$</td></tr>
<tr><td>1</td><td>0b000</td><td>256</td><td>$f_{PCLK}(Hz)/_{256}$</td></tr>
<tr><td>1</td><td>0b010</td><td>512</td><td>$f_{PCLK}\ (Hz)/_{512}$</td></tr>
<tr><td>1</td><td>0b011</td><td>1024</td><td>$f_{PCLK}\ (Hz)/_{1024}$</td></tr>
<tr><td>1</td><td>0b100</td><td>2048</td><td>$f_{PCLK}\ (Hz)/_{2048}$</td></tr>
<tr><td>1</td><td>0b101</td><td>4096</td><td>$f_{PCLK}\ (Hz)/_{4096}$</td></tr>
<tr><td>1</td><td>0b110</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>1</td><td>0b111</td><td>Reserved</td><td>Reserved</td></tr>
</table>
</td></tr>
</table>

Timer Control Register			TMRn_CN	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
2:0	tmode	R/W	0	**Timer Mode Select** Sets the timer's operating mode. *tmode* / Timer Mode 0b000 One-Shot 0b001 Continuous 0b010 Counter 0b011 PWM 0b100 Capture 0b101 Compare 0b110 Gated 0b111 Capture/Compare

Table 15-7: Timer Non-Overlapping Compare Registers

Timer Non-Overlapping Compare Register			TMRn_NOLCMP	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:8	nolhcmp	RO	0	**Reserved**
7:0	nollcmp	RO	0	**Reserved**

16. Wake-Up Timer (WUT0)

The WUT is a unique instance of a 32-bit timer.

- Uses the 32.768kHz for its clock source
- Programmable prescaler with values from 1 to 4096
- Supports three timer modes
  - One-Shot: Timer counts up to terminal value then halts.
  - Continuous: Timer counts up to terminal value then repeats.
  - Compare: The timer counts up to the terminal value, generates a wake-up timer event, resets the count, and continues counting.
- Independent interrupt

16.1 Basic Operation

The timer modes operate by incrementing the *WUTn_CNT* register. The *WUTn_CNT* register is always readable, even while the timer is enabled and counting.

Each timer mode has a user-configurable timer period, which terminates on the timer clock cycle following the end of timer period condition. The end of a timer period will always set the corresponding interrupt bit and can generate an interrupt, if enabled.

The timer peripheral automatically sets *WUTn_CNT* to 0x0000 0001 at the end of a timer period, but *WUTn_CNT* is set to 0x0000 0000 following a system reset. This means the first timer period following a system reset will be one timer clock longer than subsequent timer periods if *WUTn_CNT* is not initialized to 0x0000 0001 during the timer configuration step.

The timer clock frequency, f_{CNT_CLK}. is a divided version of the 32.768kHz RTC clock. The divisor/prescaler can be set from 1 to 4096 using the *WUTn_CTRL.pres3*: *WUTn_CTRL.pres,* as shown in *Table 16-1*.

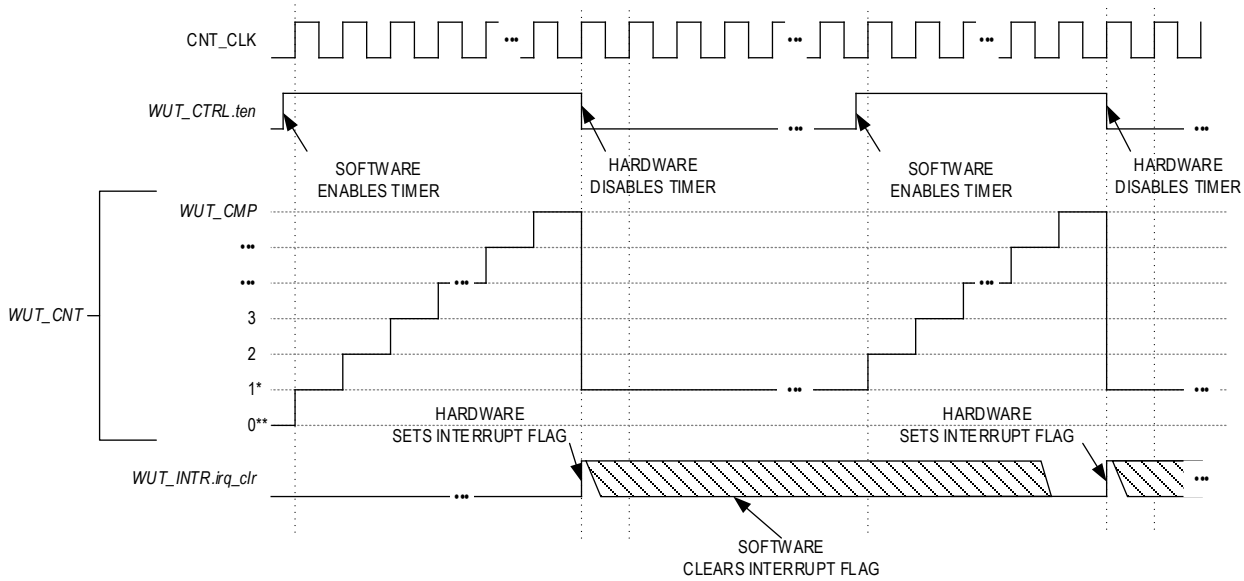
Table 16-1: MAX32665/MAX32666 WUT Clock Period

pres3	pres	Prescaler	f_{CNT_CLK} (Hz)
0	0b000	1	32,768
0	0b001	2	16,384
0	0b010	4	8,192
0	0b011	8	4,096
0	0b100	16	2,048
0	0b101	32	1,024
0	0b110	64	512
0	0b111	128	256
1	0b000	256	128
1	0b010	512	64
1	0b011	1024	32
1	0b100	2048	16
1	0b101	4096	8
1	0b110	Reserved	Reserved
1	0b111	Reserved	Reserved

16.2 One-Shot Mode (000b)

In One-Shot mode, the timer peripheral increments *WUTn_CNT* until it matches *WUTn_CMP* and then stops incrementing and disables the timer. In this mode, the timer must be re-enabled to start another One-Shot mode event.

Figure 16-1: One-Shot Mode Diagram



* *WUT_CNT* AUTOMATICALLY RELOADS WITH 1 AT THE END OF THE WUT PERIOD, BUT SOFTWARE CAN WRITE ANY INITIAL VALUE TO *WUT_CNT* PRIOR TO ENABLING THE TIMER.

** THE DEFAULT VALUE OF *WUT_CNT* FOR THE FIRST PERIOD AFTER A SYSTEM RESET IS 0 UNLESS CHANGED BY SOFTWARE.

16.2.1 One-Shot Mode Timer Period

The timer period ends on the timer clock following *WUTn_CNT* = *WUTn_CMP*.

The timer peripheral automatically performs the following actions at the end of the timer period:

1. *WUTn_CNT* is reset to 0x0000 0001.
2. The timer is disabled by setting *WUTn_CTRL.ten* = 0.
3. The timer interrupt bit *WUTn_INTFL.irq_clr* will be set. An interrupt will be generated if enabled.

16.2.2 One-Shot Mode Configuration

Configure the timer for One-Shot mode by performing the following steps:

1. Set *WUTn_CTRL.ten* = 0 to disable the timer.
2. Set *WUTn_CTRL.tmode* = 0 to select One-Shot mode.
3. Set *WUTn_CTRL.pres3*: *WUTn_CTRL.pres* to determine the timer period.
4. Enable the interrupt and set the interrupt priority.
5. Write an initial value to *WUTn_CNT*, if desired. This only the first period; subsequent timer periods always reset *WUTn_CNT to* 0x0000 0001.
6. Write the compare value to *WUTn_CMP*.
7. Set *WUTn_CTRL.ten* = 1 to enable the timer.

The timer period is calculated using the following equation:

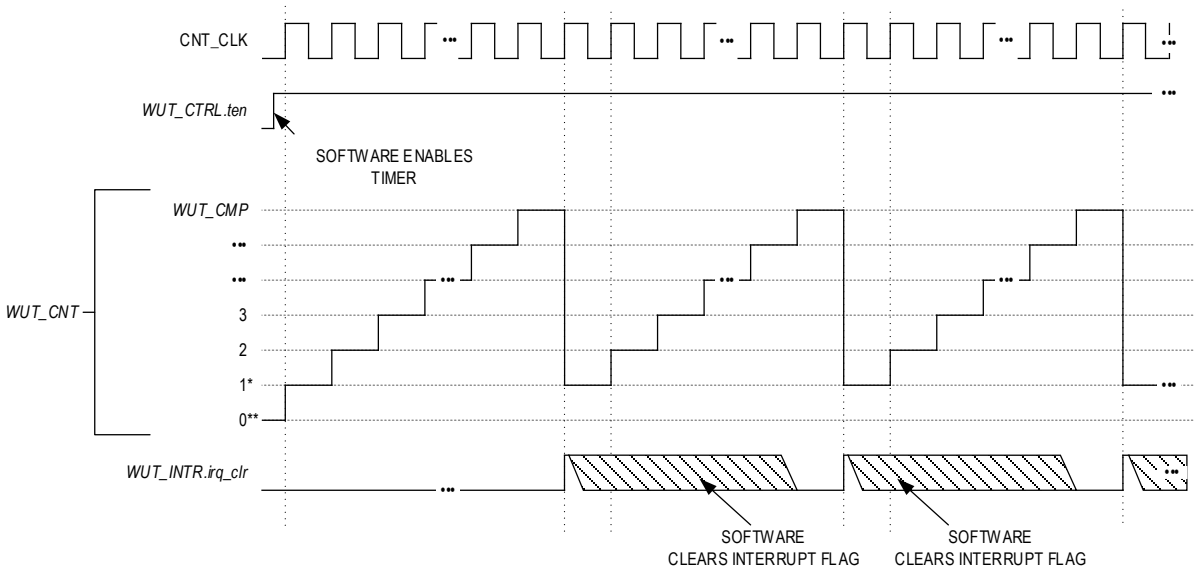
Equation 16-1*: One-Shot Mode Timer Period*

$$One\text{-}Shot\ mode\ timer\ period\ in\ seconds = \frac{WUTn\_CMP - WUTn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

16.3 Continuous Mode (1)

In Continuous mode, the timer peripheral increments *WUTn_CNT* until it matches *WUTn_CMP*, resets *WUTn_CNT* to 0x0000 0001, and continues incrementing.

Figure 16-2*: Continuous Mode Diagram*



* *WUT_CNT* AUTOMATICALLY RELOADS WITH 1 AT THE END OF THE WAKEUP TIMER PERIOD, BUT SOFTWARE CAN WRITE ANY INITIAL VALUE TO *WUT_CNT* PRIOR TO ENABLING THE WAKEUP TIMER.

** THE VALUE OF *WUT_CNT* FOR THE FIRST PERIOD AFTER A SYSTEM RESET IS 0 UNLESS CHANGED BY SOFTWARE.

16.3.1 Continuous Mode Timer Period

The timer period ends on the timer clock following *WUTn_CNT* = *WUTn_CMP*.

The timer peripheral automatically performs the following actions at the end of the timer period:

1. *WUTn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing.
2. The timer interrupt bit *WUTn_INTFL.irq_clr* will be set. An interrupt will be generated if enabled.

16.3.2 Continuous Mode Configuration

Configure the timer for Continuous mode by performing the following steps:

1. Set *WUTn_CTRL*.ten = 0 to disable the timer.
2. Set *WUTn_CTRL.tmode* = 1 to select continuous mode.
3. Set *WUTn_CTRL.pres3*: *WUTn_CTRL.pres* to determine the timer period.
4. Enable the interrupt and set the interrupt priority.

5. Write an initial value to *WUTn_CNT*, if desired. The initial value is only used for the first period; subsequent timer periods always reset the *WUTn_CNT* register = 1.
6. Write the compare value to *WUTn_CMP*.
7. Set *WUTn_CTRL.ten* = 1 to enable the timer.

The Continuous mode timer period is calculated using *Equation 15-3*.

Equation 16-2*: Continuous Mode Timer Period*

$$Continuous\ mode\ timer\ period\ in\ seconds = \frac{WUTn\_CMP - WUTn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

16.3.3 Compare Mode (5)

In Compare mode, the timer peripheral increments continually from 0x0000 0000 (after the first timer period) to the maximum value, then rolls over to 0x0000 0000 and continues incrementing. The end of timer period event occurs when the timer value matches the compare value, but the timer continues to increment until the count reaches 0xFFFF FFFF. The timer counter then rolls over and continues counting from 0x0000 0000.

The timer period ends on the timer clock following *WUTn_CNT* = *WUTn_CNT*.

The timer peripheral automatically performs the following actions when a timer period event ends:

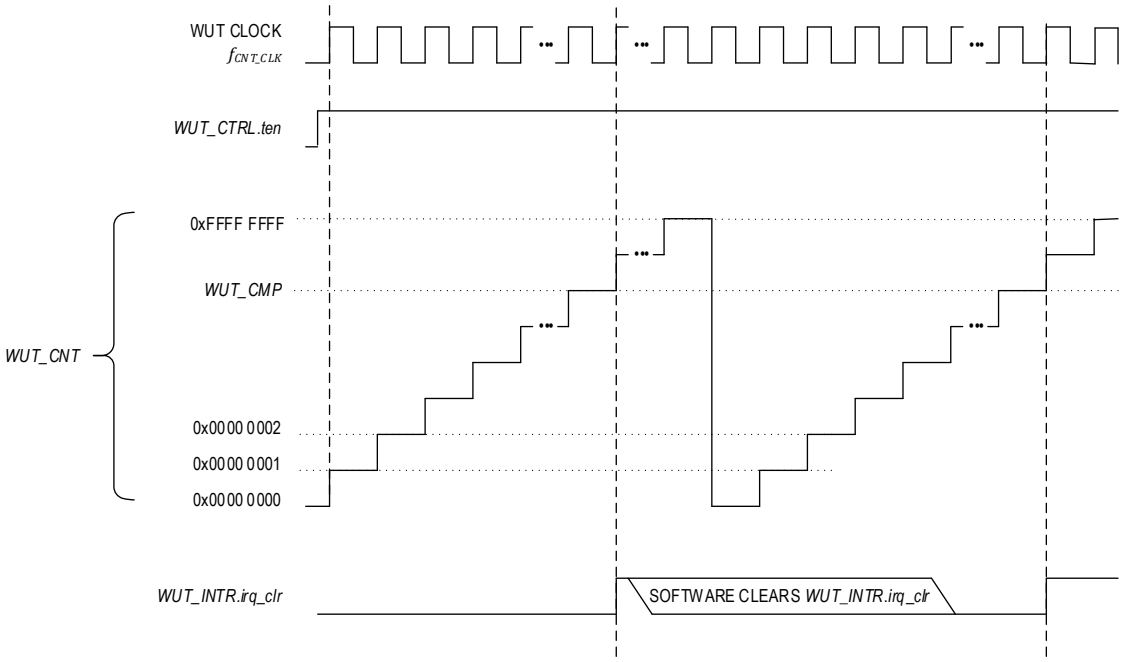
- *WUTn_CNT* is reset to 0x0000 00000.
- The *WUTn_INTFL.irq_clr* field is set to 1 to indicate a timer interrupt event occurred.
- The timer remains enabled and continues incrementing.

The initial Compare mode timer period is calculated using *Equation 15-11*. Subsequent Compare mode timer periods are always 0xFFFF FFFF.

Equation 16-3: Compare Mode Timer Initial Period

$$Compare\ mode\ timer\ period\ in\ seconds = \frac{(WUT\_CMP - WUT\_CNT_{INITIAL\_VALUE} + 1)}{f_{CNT\_CLK}(Hz)}$$

Figure 16-3: Compare Mode Diagram



THIS EXAMPLE USES THE FOLLOWING CONFIGURATION IN ADDITION TO THE SETTINGS SHOWN ABOVE:
WUT_CTRL.TMODE = 5 (COMPARE)

Configure the timer for Compare mode by doing the following:

1. Set *WUTn_CTRL*.*ten* = 0 to disable the timer.
2. Set *WUTn_CTRL*.*tmode* = 1 to select continuous mode.
3. Set *WUTn_CTRL*.*pres3*: *WUTn_CTRL*.*pres* to determine the timer period.
4. Enable the wake-up timer as a wake-up source by setting *GCR_PM*.*wutwken* = 1.
   a. If desired, register a wake-up interrupt handler (WUT_IRQn).
5. Write the compare value to the *WUTn_CMP* register.
6. If desired, write an initial value to *WUTn_CTRL* register.
7. Clear the wake-up timer interrupt flag by writing 0 to *WUTn_INTFL*.*irq_clr*.
8. Set *WUTn_CTRL*.*ten* = 1 to enable the timer.
9. Enter a low-power sleep mode. See *Operating Modes* for details.

16.4 Registers

See *Table 3-3* for the the base address of this peripheral/module. Each instance has its own, independent set of the registers shown in *Table 16-2*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 16-2: Timer Register Summary

Offset	Register Name	Description
0x0000	*WUTn_CNT*	Timer Counter Register
0x0004	*WUTn_CMP*	Timer Compare Register
0x0008	Reserved	
0x000C	*WUTn_INTFL*	Timer Interrupt Register
0x0010	*WUTn_CTRL*	Timer Control Register
0x0014	*Reserved*	
0x0018	*Reserved*	
0x001C	*Reserved*	
0x0020	*Reserved*	

16.5 Register Details

Table 16-3: Timer Count Register

Timer Count Register			WUTn_CNT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	count	R/W	0	**Timer Count Value** The current count value for the timer. This field increments as the timer counts. Reads of this register are always valid. Prior to writing this field, disable the timer by clearing bit *WUTn_CTRL.ten*.

Table 16-4: Timer Compare Register

Timer Compare Register			WUTn_CMP	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	compare	R/W	0	**Timer Compare Value** The value in this register is used as the compare value for the timer's count value. The compare field meaning is determined by the specific mode of the timer. See the timer mode's detailed configuration section for *compare* usage and meaning.

Table 16-5: Timer Interrupt Register

Timer Interrupt Register			WUTn_INTFL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	irq_clr	RW	0	**Timer Interrupt** If set, this field indicates a timer interrupt condition occurred. Writing any value to this bit clears the timer's interrupt. 0: Normal operation. 1: Timer interrupt occurred.

Table 16-6: Timer Control Register

Timer Control Register			WUTn_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:13	-	RO	0	**Reserved**
12:9	-	DNM	0	**Reserved. Do Not Modify.**
8	pres3	R/W	0	**Timer Prescale Select MSB** See *WUTn_CTRL.pres* for details on this field's usage.
7	ten	R/W	0	**Timer Enable** 1: Timer enabled 0: Timer disabled

Timer Control Register			WUTn_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
6	-	DNM	0	**Reserved. Do Not Modify.**
5:3	pres	R/W	0	**Timer Prescaler Select** Sets the timer's prescaler value. The prescaler divides the PCLK input to the timer and sets the timer's count clock, $$f_{CNT\_CLK} = \frac{f_{CLK\_SOURCE}}{prescaler}$$ The timer's prescaler setting is a 4-bit value with pres3 as the most significant bit and pres as the three least significant bits. The table below shows the prescaler values based on *pres3*:*pres*.
2:0	tmode	R/W	0	**Timer Mode Select** Sets the timer's operating mode.

pres3	pres	Prescaler
0	0b000	1
0	0b001	2
0	0b010	4
0	0b011	8
0	0b100	16
0	0b101	32
0	0b110	64
0	0b111	128
1	0b000	256
1	0b010	512
1	0b011	1024
1	0b100	2048
1	0b101	4096
1	0b110	Reserved
1	0b111	Reserved

tmode	Timer Mode
0b000	One-Shot
0b001	Continuous
0b010	Reserved
0b011	Reserved
0b100	Reserved
0b101	Compare
0b110	Reserved
0b111	Reserved

17. Pulse Train Engine (PT)

Each independent pulse train engine operates either in Square Wave mode which generates a continuous 50% duty-cycle square wave, or pulse train mode which generates a continuous programmed bit pattern from 2- to 32-bits in length. Pulse train engines are used independently or may be synchronized together to generate signals in unison. The frequency of each generated output can be set separately based on a divisor of the Peripheral Clock.

17.1 Instances

The device provides 16 instances of the pulse train engine peripheral.

- PT0
- PT1
- PT2
- PT3
- PT4
- PT5
- PT6
- PT7
- PT8
- PT9
- PT10
- PT11
- PT12
- PT13
- PT14
- PT15

All peripheral registers share a common register set.

17.2 Pulse Train Engine Features

The pulse train outputs with individually programmable modes, patterns and output enables. The pulse train engine uses the Peripheral Clock (PCLK), $f_{PTE\_CLK} = f_{PCLK}$, ensuring all pulse train outputs use the same clock source.

- Independent or synchronous pulse train output operation
- Atomic Enable and Atomic Disable
- Synchronous enable or disable of pulse train output(s) without modification to non-intended pulse train outputs
- Multiple Output Modes:
  - Square Wave Output mode generates a repeating square wave (50% duty cycle)
  - Pattern Output mode for generating a customizable output wave based on a programmable bit pattern from 2 to 32 output cycles
- Global clock for all generated outputs
- Individual rate configuration for each pulse train output
- Configuration registers are modifiable while the pulse train engine is running
- Pulse train outputs can be halted and resumed at the same point

17.3 Engine

The pulse train engine uses the Peripheral Clock as the peripheral input clock,. Each pulse train output is individually configurable and independently controlled.

The following sections describe the available configuration options for each individual pulse train output.

17.3.1 Pulse Train Output Modes

Each pulse train output supports the following modes:

- Pulse Train Mode
- Bit Patter Length
- Square Wave Mode

17.3.1.1 Pulse Train Mode

When pulse train x (PTn) is configured in pulse train mode, the configuration also includes the bit length (up to 32-bits) of the custom pulse train. This is configured using the 5-bit field *PTn_RATE_LENGTH.mode*.

PTn_RATE_LENGTH.mode = 1 (PTn configured in Square Wave mode)
PTn_RATE_LENGTH.mode > 1 (PTn configured in pulse train mode. The value of mode is the pattern bit length.)
PTn_RATE_LENGTH.mode = 0 (PTn bit length configured for pulse train mode, 32-bit pattern)

If in pulse train Mode, Set the Bit Pattern

If an output is set to pulse train mode, then configure a custom bit pattern from 2-bits to 32-bits in length in the 32-bit register *PTn_TRAIN*. The pattern is shifted out least significant bit (LSB) first. If the output is configured in Square Wave mode, then the *PTn_TRAIN* register is ignored.

Equation 17-1: Pulse Train Mode Output Function

$$\text{PTn\_TRAIN} = [\text{Bit pattern for PTn}]$$

Synchronize Two or More Outputs, if Needed

The write-only register *PTG_RESYNC* “PT Global Resync” allows two or more outputs to be reset and synchronized. Write to any bit in *PTG_RESYNC* to simultaneously reset any outputs in pulse train mode to the beginning of the pattern (the LSB) set in the *PTn_TRAIN* bit-pattern register, and reset the output to 0 for outputs in Square Wave mode.

17.3.1.2 Pulse Train Loop Mode

By default, a pulse train engine runs indefinitely until it is disabled by firmware.

A pulse train engine can be configured to repeat its pattern a specified number of times, called Loop mode. To select Loop mode, write a non-zero value to the 16-bit field *PTn_LOOP.count*. When the pulse train engine is enabled, this field decrements by 1 each time a complete pattern is shifted through the output pin. When the count reaches 0, the output is halted, and the corresponding flag in the *PTG_INTFL* register is set.

17.3.1.3 Pulse Train Loop Delay

If the pulse train is configured in Loop mode, a delay can be inserted after each repeated output pattern. To enable a delay, write the 12-bit field *PTn_LOOP.delay* with the number of Peripheral Clock cycles to delay between the most significant bit (MSB) of the last pattern to the least-significant bit (LSB) of the next pattern. During this delay, the output is held at the MSB of the last pattern. If the loop counter has not reached 0, then it is decremented when the next pattern starts.

17.3.1.4 Pulse Train Automatic Restart Mode

When an engine in pulse train mode is in Loop mode and stops when the loop count reaches 0, this is called a Stop Event. A Stop Event can optionally trigger one or more pulse trains to restart from the beginning. This is called Automatic Restart mode. While only pulse train engines operating in pulse train mode can operate in Loop mode and can optionally restart a pulse train engine, Automatic Restart mode can trigger pulse train engines operating in pulse train mode or in Square Wave mode.

If a running pulse train engine is triggered by another pulse train’s Stop Event, Automatic Restart restarts the running pulse train engine from the beginning of its pattern. If a pulse train engine is triggered by another pulse train’s Stop Event, and it is not running, Automatic Restart sets the enable bit to 1, and starts the pulse train engine.

The settings for this mode are contained in the *PTn_RESTART* register for each pulse train engine. Note that the configuration for automatic restart is set using the pulse engine(s) triggered by the automatic restart, not the pulse train

engine(s) that trigger the automatic restart. For example, the PT8_RESTART register configures which pulse train engine triggers PT8 to restart.

Each pulse train engine can be configured to perform an Automatic Restart when it detects a Stop Event from one or two pulse trains.

If *PTn_RESTART.on_pt_x_loop_exit* = 1, then pulse train engine n automatically restarts when it detects a Stop Event from pulse train x, where x is the value in the 5-bit field *PTn_RESTART.pt_x_select*.

If *PTn_RESTART.on_pt_*y*_loop_exit* = 1, then pulse train engine n automatically restarts when it detects a Stop Event from pulse train y, where y is the value in 5-bit field *PTn_RESTART.pt_y_select*.

A pulse train engine can be configured to restart on its own Stop Event, allowing the pulse train to run indefinitely.

Each individual pulse train can be configured for:

17.3.1.5 No Automatic Restart

Automatic Restart triggered by a stop event from pulse train x only

Automatic Restart triggered by a stop event from pulse train y only

Automatic Restart triggered by a stop event from both pulse train x and pulse train y

17.4 Enabling and Disabling a Pulse Train Output

The *PTG_ENABLE* register is used to enable and disable each of the individual pulse train outputs. Enable a given pulse train output by setting the respective bit in the *PTG_ENABLE* register. Halt a pulse train output by clearing the respective bit in the *PTG_ENABLE* register.

Note: Prior to changing a pulse train output's configuration the corresponding pulse train output should be halted to prevent unexpected behavior.

17.5 Atomic Pulse Train Output Enable and Disable

Deterministic enable and disable operations are critical for pulse train outputs that must be synchronized in an application. The *PTG_ENABLE* register does not perform atomic access directly. Atomic operations are supported using the registers *PTG_SAFE_EN*, *PTG_SAFE_DIS*.

For most pulse train peripherals, enabling and disabling individual pulse trains is performed by setting and clearing bits in the global enable/disable register, which for this peripheral is *PTG_ENABLE*. For most Arm Cortex-M microcontrollers, this is usually done by bit banding. Because bit banding performs a read, modify, write (RMW), some pulse trains could start and end during the RMW operation, often with unpredictable results.

To ensure safe and predictable operation, two additional registers are used to enable and disable the outputs.

17.5.1 Pulse Train Atomic Enable

PTG_SAFE_EN "Global Safe Enable" is a write-only register. To safely enable outputs without a RMW, write a 32-bit value to this register with a 1 in the bit positions corresponding to the pulse train engines to be enabled. This immediately sets to 1 the corresponding bits in the *PTG_ENABLE* register to 1, which enables the corresponding pulse train engine. Writing a 0 to any bit position in the *PTG_SAFE_EN* register has no effect on the state of the corresponding pulse train enable bit. If the corresponding pulse train engine is already enabled and running, writing a 1 to that bit position in the *PTG_SAFE_EN* register has no effect.

17.5.2 Pulse Train Atomic Disable

PTG_SAFE_DIS "Global Safe Disable" is a write-only register for disabling a pulse train engine without performing a RMW. To safely disable pulse train engines, write a 32-bit value to this register with a 1 in the bit positions corresponding to the pulse train engines to be disabled. This immediately clears to 0 the corresponding bits in *PTG_ENABLE* which disables the corresponding pulse train engines. Writing a 0 to any bit position in the *PTG_SAFE_DIS* register has no effect on the state of the corresponding pulse train enable bit.

Bit banding is not supported for the *PTG_ENABLE*, *PTG_SAFE_EN*, and *PTG_SAFE_DIS* registers and can have unpredictable results.

17.6 Pulse Train Halt and Disable

Once a pulse train engine is enabled and running, it continues to run until one of the following events stops the output:

The corresponding enable bit in the *PTG_ENABLE* register is cleared to 0 to halt the output.

A 1 is written to the corresponding disable bit in the *PTG_SAFE_DIS* register to halt the output.

The corresponding resync bit in the *PTG_RESYNC* register is cleared to 0 to halt and reset the output.

PTn_LOOP was initialized to a non-zero value, and the loop count has reached 0 (this has no effect in Square Wave mode; it only applies to pulse train mode).

When a pulse train is halted, the corresponding enable bit in *PTG_ENABLE* is automatically cleared to 0.

17.7 Pulse Train Interrupts

Each pulse train can generate an interrupt only if it is configured in pulse train mode, and the loop counter *PTG_SAFE_DIS* was initialized to a non-zero number. When *PTG_SAFE_DIS* counts down to 0, the corresponding status flag in the *PTG_INTFL* register is set. If the corresponding interrupt enable bit in the *PTG_INTEN* register is set, the event also generates an interrupt.

17.8 Registers

See *Table 3-3* for the the base address of this peripheral/module. Each instance has its own, independent set of the registers shown in *Table 17-1*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 17-1: Pulse Train Engine Register Summary

Offset	Register	Description
[0x0000]	*PTG_ENABLE*	PT Global Enable/Disable Control
[0x0004]	*PTG_RESYNC*	PT Global Resync
[0x0008]	*PTG_INTFL*	PT Stopped Global Status Flags
[0x000C]	*PTG_INTEN*	PT Global Interrupt Enable
[0x0010]	*PTG_SAFE_EN*	PT Global Safe Enable
[0x0014]	*PTG_SAFE_DIS*	PT Global Safe Disable
[0x0020]	*PTn_RATE_LENGTH*	PT0 Configuration
[0x0024]	*PTn_TRAIN*	PT0 Pulse Train Mode Bit Pattern
[0x0028]	*PTn_LOOP*	PT0 Loop Control
[0x002C]	*PTn_RESTART*	PT0 Automatic Restart
[0x0030]	*PTn_RATE_LENGTH*	PT1 Configuration
[0x0034]	*PTn_TRAIN*	PT1 Pulse Train Mode Bit Pattern
[0x0038]	*PTn_LOOP*	PT1 Loop Control
[0x003C]	*PTn_RESTART*	PT1 Automatic Restart
[0x0040]	*PTn_RATE_LENGTH*	PT2 Configuration
[0x0044]	*PTn_TRAIN*	PT2 Pulse Train Mode Bit Pattern
[0x0048]	*PTn_LOOP*	PT2 Loop Control
[0x004C]	*PTn_RESTART*	PT2 Automatic Restart
[0x0050]	*PTn_RATE_LENGTH*	PT3 Configuration

Offset	Register	Description
[0x0054]	*PTn_TRAIN*	PT3 Pulse Train Mode Bit Pattern
[0x0058]	*PTn_LOOP*	PT3 Loop Control
[0x005C]	*PTn_RESTART*	PT3 Automatic Restart
[0x0060]	*PTn_RATE_LENGTH*	PT4 Configuration
[0x0064]	*PTn_TRAIN*	PT4 Pulse Train Mode Bit Pattern
[0x0068]	*PTn_LOOP*	PT4 Loop Control
[0x006C]	*PTn_RESTART*	PT4 Automatic Restart
[0x0070]	*PTn_RATE_LENGTH*	PT5 Configuration
[0x0074]	*PTn_TRAIN*	PT5 Pulse Train Mode Bit Pattern
[0x0078]	*PTn_LOOP*	PT5 Loop Control
[0x007C]	*PTn_RESTART*	PT5 Automatic Restart
[0x0080]	*PTn_RATE_LENGTH*	PT6 Configuration
[0x0084]	*PTn_TRAIN*	PT6 Pulse Train Mode Bit Pattern
[0x0088]	*PTn_LOOP*	PT6 Loop Control
[0x008C]	*PTn_RESTART*	PT6 Automatic Restart
[0x0090]	*PTn_RATE_LENGTH*	PT7 Configuration
[0x0094]	*PTn_TRAIN*	PT7 Pulse Train Mode Bit Pattern
[0x0098]	*PTn_LOOP*	PT7 Loop Control
[0x009C]	*PTn_RESTART*	PT7 Automatic Restart
[0x00A0]	*PTn_RATE_LENGTH*	PT8 Configuration
[0x00A4]	*PTn_TRAIN*	PT8 Pulse Train Mode Bit Pattern
[0x00A8]	*PTn_LOOP*	PT8 Loop Control
[0x00AC]	*PTn_RESTART*	PT8 Automatic Restart
[0x00B0]	*PTn_RATE_LENGTH*	PT9 Configuration
[0x00B4]	*PTn_TRAIN*	PT9 Pulse Train Mode Bit Pattern
[0x00B8]	*PTn_LOOP*	PT9 Loop Control
[0x00BC]	*PTn_RESTART*	PT9 Automatic Restart
[0x00C0]	*PTn_RATE_LENGTH*	PT10 Configuration
[0x00C4]	*PTn_TRAIN*	PT10 Pulse Train Mode Bit Pattern
[0x00C8]	*PTn_LOOP*	PT10 Loop Control
[0x00CC]	*PTn_RESTART*	PT10 Automatic Restart
[0x00D0]	*PTn_RATE_LENGTH*	PT11 Configuration
[0x00D4]	*PTn_TRAIN*	PT11 Pulse Train Mode Bit Pattern
[0x00D8]	*PTn_LOOP*	PT11 Loop Control
[0x00DC]	*PTn_RESTART*	PT11 Automatic Restart
[0x00E0]	*PTn_RATE_LENGTH*	PT12 Configuration
[0x00E4]	*PTn_TRAIN*	PT12 Pulse Train Mode Bit Pattern
[0x00E8]	*PTn_LOOP*	PT12 Loop Control
[0x00EC]	*PTn_RESTART*	PT12 Automatic Restart
[0x00F0]	*PTn_RATE_LENGTH*	PT13 Configuration
[0x00F4]	*PTn_TRAIN*	PT13 Pulse Train Mode Bit Pattern
[0x00F8]	*PTn_LOOP*	PT13 Loop Control
[0x00FC]	*PTn_RESTART*	PT13 Automatic Restart
[0x0100]	*PTn_RATE_LENGTH*	PT14 Configuration
[0x0104]	*PTn_TRAIN*	PT14 Pulse Train Mode Bit Pattern
[0x0108]	*PTn_LOOP*	PT14 Loop Control
[0x010C]	*PTn_RESTART*	PT14 Automatic Restart

Offset	Register	Description
[0x0110]	*PTn_RATE_LENGTH*	PT15 Configuration
[0x0114]	*PTn_TRAIN*	PT15 Pulse Train Mode Bit Pattern
[0x0118]	*PTn_LOOP*	PT15 Loop Control
[0x011C]	*PTn_RESTART*	PT15 Automatic Restart

17.9 Register Details

Table 17-2: Pulse Train Engine Global Enable/Disable Register

PT Global Enable/Disable Control			PTG_ENABLE	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
15	enable_pt15	R/W	0	**Enable PT15** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
14	enable_pt14	R/W	0	**Enable PT14** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
13	enable_pt13	R/W	0	**Enable PT13** 0: Disable 1: Enable *Note: Disabling an active pulse train halts the output and does not generate a Stop Event.*
12	enable_pt12	R/W	0	**Enable PT12** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
11	enable_pt11	R/W	0	**Enable PT11** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
10	enable_pt10	R/W	0	**Enable PT10** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
9	enable_pt9	R/W	0	**Enable PT9** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.

PT Global Enable/Disable Control			PTG_ENABLE	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
8	enable_pt8	R/W	0	**Enable PT8** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
7	enable_pt7	R/W	0	**Enable PT7** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
6	enable_pt6	R/W	0	**Enable PT6** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
5	enable_pt5	R/W	0	**Enable PT5** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
4	enable_pt4	R/W	0	**Enable PT4** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
3	enable_pt3	R/W	0	**Enable PT3** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
2	enable_pt2	R/W	0	**Enable PT2** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
1	enable_pt1	R/W	0	**Enable PT1** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.
0	enable_pt0	R/W	0	**Enable PT0** 0: Disable 1: Enable Note: Disabling an active pulse train halts the output and does not generate a Stop Event.

If *PTn_LOOP.count* loop counter is set to a non-zero number, when the loop counter counts down to zero then the pulse train engine stops, and the corresponding enable bit is cleared.

Table 17-3: Pulse Train Engine Resync Register

PT Resync Register			PTG_RESYNC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
15	pt15	WO	-	**Resync Control for PT15** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
14	pt14	WO	-	**Resync Control for PT14** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
13	pt13	WO	-	**Resync Control for PT13** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
12	pt12	WO	-	**Resync Control for PT12** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
11	pt11	WO	-	**Resync Control for PT11** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*

PT Resync Register			PTG_RESYNC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
10	pt10	WO	-	**Resync Control for PT10** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
9	pt9	WO	-	**Resync Control for PT9** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
8	pt8	WO	-	**Resync Control for PT8** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
7	pt7	WO	-	**Resync Control for PT7** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
6	pt6	WO	-	**Resync Control for PT6** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*

PT Resync Register			PTG_RESYNC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
5	pt5	WO	-	**Resync Control for PT5** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
4	pt4	WO	-	**Resync Control for PT4** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
3	pt3	WO	-	**Resync Control for PT3** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
2	pt2	WO	-	**Resync Control for PT2** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
1	pt1	WO	-	**Resync Control for PT1** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*

PT Resync Register			PTG_RESYNC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
0	pt0	WO	-	**Resync Control for PT0** Write 1 to reset the output of the pulse train. For pulse train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*

Table 17-4: Pulse Train Engine Stopped Interrupt Flag Register

PT Stopped Interrupt Flag Register			PTG_INTFL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
15	pt15	R/W1C	0	**PT15 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
14	pt14	R/W1C	0	**PT14 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
13	pt13	R/W1C	0	**PT13 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
12	pt12	R/W1C	0	**PT12 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
11	pt11	R/W1C	0	**PT11 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
10	pt10	R/W1C	0	**PT10 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.

PT Stopped Interrupt Flag Register		PTG_INTFL		[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
9	pt9	R/W1C	0	**PT9 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
8	pt8	R/W1C	0	**PT8 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
7	pt7	R/W1C	0	**PT7 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
6	pt6	R/W1C	0	**PT6 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
5	pt5	R/W1C	0	**PT5 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
4	pt4	R/W1C	0	**PT4 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
3	pt3	R/W1C	0	**PT3 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
2	pt2	R/W1C	0	**PT2 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
1	pt1	R/W1C	0	**PT1 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.

PT Stopped Interrupt Flag Register			PTG_INTFL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
0	pt0	R/W1C	0	**PT0 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.

Table 17-5: Pulse Train Engine Interrupt Enable Register

PT Interrupt Enable Register			PTG_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
15	pt15	R/W	0	**PT15 Interrupt Enable** 0: Disabled. 1: Enabled.
14	pt14	R/W	0	**PT14 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
13	pt13	R/W	0	**PT13 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
12	pt12	R/W	0	**PT12 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
11	pt11	R/W	0	**PT11 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
10	pt10	R/W	0	**PT10 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
9	pt9	R/W	0	**PT9 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.

PT Interrupt Enable Register			PTG_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
8	pt8	R/W	0	**PT8 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
7	pt7	R/W	0	**PT7 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
6	pt6	R/W	0	**PT6 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
5	pt5	R/W	0	**PT5 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
4	pt4	R/W	0	**PT4 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
3	pt3	R/W	0	**PT3 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
2	pt2	R/W	0	**PT2 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
1	pt1	R/W	0	**PT1 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.
0	pt0	R/W	0	**PT0 Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register. 0: Disabled. 1: Enabled.

17.9.2 Pulse Train Engine Safe Enable Register

A 32-bit value written to this register performs an immediate binary OR with the contents of *PTG_ENABLE*. The result is immediately stored in the *PTG_ENABLE*.

Table 17-6: Pulse Train Engine Safe Enable Register

Pulse Train Engine Safe Enable Register		PTG_SAFE_EN		[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
15	pt15	WO	-	**Safe Enable Control for PT15** Writing a 1 sets *PTG_ENABLE.enable_pt15*. 1: Enable corresponding pulse train 0: No effect
14	pt14	WO	-	**Safe Enable Control for PT14** Writing a 1 sets *PTG_ENABLE.enable_pt14*. 1: Enable corresponding pulse train 0: No effect
13	pt13	WO	-	**Safe Enable Control for PT1** Writing a 1 sets *PTG_ENABLE.enable_pt13*. 1: Enable corresponding pulse train 0: No effect
12	pt12	WO	-	**Safe Enable Control for PT12** Writing a 1 sets *PTG_ENABLE.enable_pt12*. 1: Enable corresponding pulse train 0: No effect
11	pt11	WO	-	**Safe Enable Control for PT11** Writing a 1 sets *PTG_ENABLE.enable_pt11*. 1: Enable corresponding pulse train 0: No effect
10	pt10	WO	-	**Safe Enable Control for PT10** Writing a 1 sets *PTG_ENABLE.enable_pt10*. 1: Enable corresponding pulse train 0: No effect
9	pt9	WO	-	**Safe Enable Control for PT9** Writing a 1 sets *PTG_ENABLE.enable_pt9*. 1: Enable corresponding pulse train 0: No effect
8	pt8	WO	-	**Safe Enable Control for PT8** Writing a 1 sets *PTG_ENABLE.enable_pt8*. 1: Enable corresponding pulse train 0: No effect
7	pt7	WO	-	**Safe Enable Control for PT7** Writing a 1 sets *PTG_ENABLE.enable_pt7*. 1: Enable corresponding pulse train 0: No effect

Pulse Train Engine Safe Enable Register		PTG_SAFE_EN		[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
6	pt6	WO	-	**Safe Enable Control for PT6** Writing a 1 sets *PTG_ENABLE.enable_pt6*. 1: Enable corresponding pulse train 0: No effect
5	pt5	WO	-	**Safe Enable Control for PT5** Writing a 1 sets *PTG_ENABLE.enable_pt5*. 1: Enable corresponding pulse train 0: No effect
4	pt4	WO	-	**Safe Enable Control for PT4** Writing a 1 sets *PTG_ENABLE.enable_pt4*. 1: Enable corresponding pulse train 0: No effect
3	pt3	WO	-	**Safe Enable Control for PT3** Writing a 1 sets *PTG_ENABLE.enable_pt3*. 1: Enable corresponding pulse train 0: No effect
2	pt2	WO	-	**Safe Enable Control for PT2** Writing a 1 sets *PTG_ENABLE.enable_pt2*. 1: Enable corresponding pulse train 0: No effect
1	pt1	WO	-	**Safe Enable Control for PT1** Writing a 1 sets *PTG_ENABLE.enable_pt1*. 1: Enable corresponding pulse train 0: No effect
0	pt0	WO	-	**Safe Enable Control for PT0** Writing a 1 sets *PTG_ENABLE.enable_pt0*. 1: Enable corresponding pulse train 0: No effect

17.9.3 Pulse Train Engine Safe Disable Register

A 32-bit value written to this register immediately disables the corresponding pulse train in the *PTG_ENABLE* register. The result is immediately stored in *PTG_ENABLE*. Setting a field to 1 disables the corresponding pulse train immediately.

Table 17-7: Pulse Train Engine Safe Disable Register

Pulse Train Engine Safe Disable Register		PTG_SAFE_DIS		[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
15	pt15	WO	-	**Safe Disable Control for PT15** Writing a 1 clears *PTG_ENABLE.enable_pt15*. 1: Disable corresponding pulse train 0: No effect

Pulse Train Engine Safe Disable Register	PTG_SAFE_DIS	[0x0014]

Bits	Field	Access	Reset	Description
14	pt14	WO	-	**Safe Disable Control for PT14** Writing a 1 clears *PTG_ENABLE.enable_pt14*. 1: Disable corresponding pulse train 0: No effect
13	pt13	WO	-	**Safe Disable Control for PT1** Writing a 1 clears *PTG_ENABLE.enable_pt13*. 1: Disable corresponding pulse train 0: No effect
12	pt12	WO	-	**Safe Disable Control for PT12** Writing a 1 clears *PTG_ENABLE.enable_pt12*. 1: Disable corresponding pulse train 0: No effect
11	pt11	WO	-	**Safe Disable Control for PT11** Writing a 1 clears *PTG_ENABLE.enable_pt11*. 1: Disable corresponding pulse train 0: No effect
10	pt10	WO	-	**Safe Disable Control for PT10** Writing a 1 clears *PTG_ENABLE.enable_pt10*. 1: Disable corresponding pulse train 0: No effect
9	pt9	WO	-	**Safe Disable Control for PT9** Writing a 1 clears *PTG_ENABLE.enable_pt9*. 1: Disable corresponding pulse train 0: No effect
8	pt8	WO	-	**Safe Disable Control for PT8** Writing a 1 clears *PTG_ENABLE.enable_pt8*. 1: Disable corresponding pulse train 0: No effect
7	pt7	WO	-	**Safe Disable Control for PT7** Writing a 1 clears *PTG_ENABLE.enable_pt7*. 1: Disable corresponding pulse train 0: No effect
6	pt6	WO	-	**Safe Disable Control for PT6** Writing a 1 clears *PTG_ENABLE.enable_pt6*. 1: Disable corresponding pulse train 0: No effect
5	pt5	WO	-	**Safe Disable Control for PT5** Writing a 1 clears *PTG_ENABLE.enable_pt5*. 1: Disable corresponding pulse train 0: No effect
4	pt4	WO	-	**Safe Disable Control for PT4** Writing a 1 clears *PTG_ENABLE.enable_pt4*. 1: Disable corresponding pulse train 0: No effect

Pulse Train Engine Safe Disable Register		PTG_SAFE_DIS		[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
3	pt3	WO	-	**Safe Disable Control for PT3** Writing a 1 clears *PTG_ENABLE.enable_pt3*. 1: Disable corresponding pulse train 0: No effect
2	pt2	WO	-	**Safe Disable Control for PT2** Writing a 1 clears *PTG_ENABLE.enable_pt2*. 1: Disable corresponding pulse train 0: No effect
1	pt1	WO	-	**Safe Disable Control for PT1** Writing a 1 clears *PTG_ENABLE.enable_pt1*. 1: Disable corresponding pulse train 0: No effect
0	pt0	WO	-	**Safe Disable Control for PT0** Writing a 1 clears *PTG_ENABLE.enable_pt0*. 1: Disable corresponding pulse train 0: No effect

Table 17-8: Pulse Train Engine Configuration Register

Pulse Train Configuration Register		PTn_RATE_LENGTH		[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:27	mode	R/W	0b00001	**Square Wave or Pulse Train Output Mode** Sets either pulse train mode with length, or Square Wave mode. 0: Pulse train mode, 32-bits long 1: Square Wave mode 2: Pulse train mode, 2-bits long 3: Pulse train mode, 3-bits long … etc … 31: Pulse train mode, 31-bits long Note: If this field is set to 1, Square Wave mode, the PTn_LENGTH register is not used.
26:0	rate_control	R/W	0	**Pulse Train Enable and Rate Control** Defines the rate at which the output for PTn changes state by setting the divisor of the PT Clock, where: $f_{PTn} = {f_{PTE\_CLK}}/{rate\_control}$ 0: Output halted 1: $f_{PTn} = f_{PTE\_CLK}$ 2: $f_{PTn} = {f_{PTE\_CLK}}/{2}$ 3: $f_{PTn} = {f_{PTE\_CLK}}/{3}$ … 0x7FFFF: $f_{PTn} = {f_{PTE\_CLK}}/{0x7FFFF}$

Table 17-9: Pulse Train Mode Bit Pattern Register

Pulse Train Mode Bit Pattern			PTn_TRAIN	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	ptn_train	R/W	0	**Pulse Train Mode Bit Pattern** Write the repeating bit pattern that is shifted out, LSB first, when configured in pulse train mode. Set the bit pattern length with the *PTn_RATE_LENGTH.mode* field. Note: This register is ignored in Square Wave mode. Note: 0x0000 0000and 0x0001 0000are invalid values for this register.

Table 17-10: Pulse Train n Loop Configuration Register

Pulse Train Loop Configuration			PTn_LOOP	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:28	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
27:16	delay	R/W	0	**Pulse Train Delay Between Loops** Sets the delay, in number of Peripheral Clock cycles, that the output pauses between loops. The bitfield count is decremented after the delay. If firmware writes a 0 to bitfield count, this field is ignored.
15:0	count	R/W	0	**Pulse Train Loop Countdown** Sets the number of times a pulse train pattern is repeated until it automatically stops. Reading this field returns the number of loops remaining. When this field counts down to zero, the corresponding PTn_INTFL flag is set. Write 0 to have the pulse train pattern repeat indefinitely. Ignored in Square Wave mode.

Table 17-11: Pulse Train n Automatic Restart Configuration Register

Pulse Train Automatic Restart Configuration			PTn_RESTART	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
15	on_pt_y_loop_exit	R/W	0	**Enable Automatic Restart for This Pulse Train on PTy Stop Event** 0: Disable automatic restart 1: When PTy has a Stop Event, automatically restart this pulse train from the beginning of its pattern.
14:11	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.

Pulse Train Automatic Restart Configuration			PTn_RESTART	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
12:8	pt_y_select	R/W	0	**Select PTy** Select the pulse train number to be associated with PTy. This engine must be in pulse train mode. 0b00000: PT0 0b00001: PT1 0b00010: PT2 0b00011: PT3 0b00100: PT4 0b00101: PT5 0b00110: PT6 0b00111: PT7 0b01000: PT8 0b01001: PT9 0b01010: PT10 0b01011: PT11 0b01100: PT12 0b01101: PT13 0b01110: PT14 0b01111: PT15 0b1xxxx: Reserved.
7	on_pt_x_loop_exit	R/W	0	**Enable Automatic Restart for this Pulse Train on a PTn Stop Event** 0: Disable automatic restart 1: When PTn has a Stop Event, automatically restart pulse train from the beginning of its pattern.
6:5	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
4:0	pt_x_select	R/W	0	**Select PTn** Select the pulse train number to be associated with PTn. This engine must be in pulse train mode. 0b00000: PT0 0b00001: PT1 0b00010: PT2 0b00011: PT3 0b00100: PT4 0b00101: PT5 0b00110: PT6 0b00111: PT7 0b01000: PT8 0b01001: PT9 0b01010: PT10 0b01011: PT11 0b01100: PT12 0b01101: PT13 0b01110: PT14 0b01111: PT15 0b1xxxx: Reserved.

18. Real-Time Clock (RTC)

18.1 Overview

The Real-Time Clock (RTC) is a 32-bit binary timer that keeps the time of day up to 136 years. It provides time-of-day and sub-second alarm functionality in the form of system interrupts.

The RTC operates on an external 32.768 time base. It can be generated from the internal crystal oscillator driving an external 32.768kHz crystal between the 32KIN and 32KOUT pins, or a 32.768kHz square wave driven directly into the 32KIN pin. Refer to the data sheet for the required electrical characteristics of the external crystal.

A user-configurable, digital frequency trim is provided for applications requiring higher accuracy.

The 32-bit seconds register *RTC_SEC* is incremented ever time there is a rollover of the *RTC_SSEC*.ssec sub-seconds field.

Two alarm functions are provided:

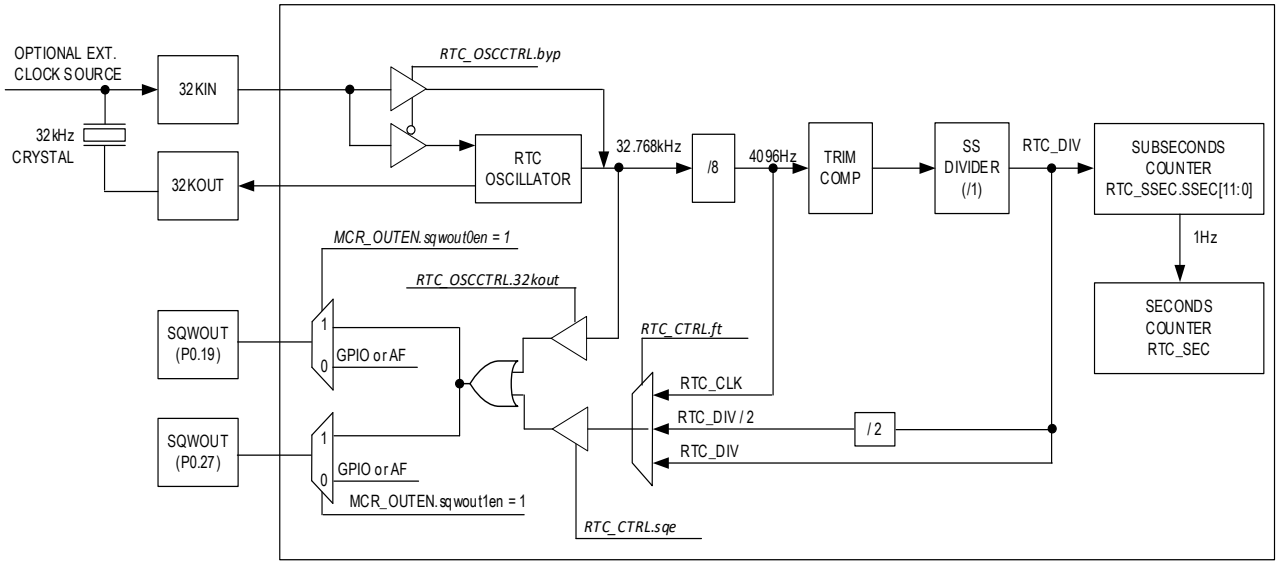
- A programmable time-of-day alarm provides a single event, alarm timer using the *RTC_TODA* alarm register, *RTC_SEC* register, and *RTC_CTRL.ade* field.
- A programmable sub-second provides a recurring alarm using the *RTC_SSECA* and *RTC_CTRL.ase* field

The RTC is powered in the always-on domain or if applicable, while there is a valid voltage on the V_{COREA} device pin.

Disabling the RTC stops incrementing *RTC_SSEC*, *RTC_SEC*, and the internal RTC sub-second counter, but preserves their current value. The 32kHz oscillator is not affected by the *RTC_CTRL.rtce* field.

The RTC increments the *RTC_TRIM.vrtc_tmr* field every 32 seconds while the RTC is enabled.

Figure 18-1: MAX32665/MAX32666 RTC Block Diagram (12-bit Sub-Second Counter)



18.2 Instances

One instance of the RTC peripheral is provided.

The RTC counter and alarm registers are shown in *Table 18-1*.

Table 18-1: MAX32665/MAX32666 RTC Counter and Alarm Registers

Field	Length	Counter Increment	Minimum	Maximum	Description
RTC_SEC	32	1s	1s	136yrs	Seconds Counter Register
RTC_SSEC	12	244µs (1/4kHz)	244µs	1s	Sub-Seconds Counter Register

Field	Length	Counter Increment	Minimum	Maximum	Description
RTC_TODA	20	1s	1s	12days	Time-of-Day Alarm Register
RTC_SSECA	32	244µs (1/4kHz)	244µs	12days	Sub-Second Alarm Register

18.3 Register Access Control

Access protection mechanisms prevent software from accessing critical registers and fields while RTC while hardware is updating them. Monitoring the *RTC_CTRL.busy* and *RTC_CTRL.rdy* fields allows software to determine when it is safe to write to registers and when registers will return valid results.

Table 18-2: RTC Register Access

Register	Field	Read Access	Write Access	Busy = 1 during write	Description
RTC_SEC	All	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1 (See *RTC_SEC and RTC_SSEC Read Access Control* for details)	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1 (See *RTC_SEC and RTC_SSEC Read Access Control* for details)	Y	Seconds Counter Register
RTC_SSEC	ssec	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1 (See *RTC_SEC and RTC_SSEC Read Access Control* for details)	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1 (See *RTC_SEC and RTC_SSEC Read Access Control* for details)	Y	Sub-Seconds Counter Register
RTC_TODA	All	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.ade* = 0 *RTC_CTRL.rtce = 0*	Y	Time-of-Day Alarm Register
RTC_SSECA	All	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.ase* = 0 *RTC_CTRL.rtce = 0*	Y	Sub-Second Alarm Register
RTC_TRIM	All	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.we* = 1	Y	Trim Register
RTC_OSCCTRL	All	Always	*RTC_CTRL.we* = 1	N	Oscillator Control Register
RTC_CTRL	rtce	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.we* = 1	Y	RTC Enable Field
	All other bits	Always	See *RTC_CTRL.busy* for limitations on specific bits	Y	

18.3.1 RTC_SEC and RTC_SSEC Read Access Control

Software reads of the *RTC_SEC* and *RTC_SSEC* registers will return invalid results if the read operation occurs on the same cycle that the register is being updated by hardware. To avoid this, hardware sets *RTC_CTRL.rdy* to 1 for 120 µs when the *RTC_SEC* and *RTC_SSEC* registers are valid and can be read.

Alternatively, the software can set *RTC_CTRL.acre* to 1 to allow asynchronous reads of both the *RTC_SEC* and *RTC_SSEC* registers.

Software can use three methods to ensure valid results when reading *RTC_SEC* and *RTC_SSEC*:

- Software clears *RTC_CTRL.rdy* to 0. Software polls *RTC_CTRL.rdy* until it reads 1 before reading the registers. The software has approximately 120µs to read the *RTC_SEC* and *RTC_SSEC* registers to ensure accuracy.
- Set the *RTC_CTRL.rdye* field to 1 to generate an RTC interrupt when the next update cycle is complete and hardware sets *RTC_CTRL.rdy* to 1. RTC interrupt must be serviced and *RTC_SEC* and/or *RTC_SSEC* registers must be read while *RTC_CTRL.rdy* = 1 to ensure accuracy. This avoids the software overhead associated with polling to observe the state of *RTC_CTRL.rdy*.
- Set *RTC_CTRL.acre* to 1 to allow asynchronous reads of both the *RTC_SEC* and *RTC_SSEC* registers. Multiple consecutive reads of *RTC_SEC* and *RTC_SSEC* must be executed until two consecutive reads are identical to ensure data accuracy.

18.3.2 RTC Write Access Control

The read-only status field *RTC_CTRL.busy* is set to 1 by hardware following a software instruction that writes to specific registers. The bit remains 1 while the software updates are being synchronized into the RTC. Software should not write to any of the registers until hardware indicates the synchronization is complete by clearing *RTC_CTRL.busy* to 0.

18.4 RTC Alarm Functions

The RTC provides time-of–day and sub-second interval alarm functions. The time-of-day alarm is implemented by matching the count values in the counter register with the value stored in the alarm register. The sub-second interval alarm provides an auto-reload timer that is driven by the trimmed RTC clock source.

18.4.1 Time-of-Day Alarm

Program the RTC Time-of-Day Alarm register (*RTC_TODA*) to configure the time-of-day-alarm. The alarm triggers when the value stored in *RTC_TODA.tod_alarm* matches the lower 20 bits of the *RTC_SEC* seconds count register. This allows programming the time-of-day-alarm to any future value between 1 second and 12 days relative to the current time with a resolution of 1 second. You must disable the time-of-day alarm before changing the *RTC_TODA .tod* field.

When the alarm occurs, a single event sets the Time-of-Day Alarm Interrupt Flag (*RTC_CTRL.aldf*) to 1.

Setting the *RTC_CTRL.aldf* bit to 1 in software results in an interrupt request to the processor if the Alarm Time-of-Day Interrupt Enable (*RTC_CTRL.ade*) bit is set to 1, and the RTC's system interrupt enable is set.

18.4.2 Sub-Second Alarm

The *RTC_SSECA* and *RTC_CTRL.ase* field control the sub-second alarm. Writing *RTC_SSECA* sets the starting value for the sub-second alarm counter. Writing the Sub-Second Alarm Enable (*RTC_CTRL.ase*) bit to 1 enables the sub-second alarm. Once enabled, an internal alarm counter begins incrementing from the *RTC_SSECA* value. When the counter rolls over from 0xFFFF FFFF to 0x0000 0000, hardware sets the *RTC_CTRL.alsf* bit triggering the alarm. At the same time, hardware also reloads the counter with the value previously written to *RTC_SSECA.rssa*.

You must disable the sub-second interval alarm, *RTC_CTRL.ase*, prior to changing the interval alarm value, *RTC_SSECA*.

The delay (uncertainty) associated with enabling the sub-second alarm is up to one period of the sub-second clock. This uncertainty is propagated to the first interval alarm. Thereafter, if the interval alarm remains enabled, the alarm triggers after each sub-second interval as defined without the first alarm uncertainty because the sub-second alarm is an auto-reload timer. Enabling the sub-second alarm with the sub-second alarm register set to 0 (*RTC_SSECA* = 0) results in the maximum sub-second alarm interval.

18.4.3 RTC Interrupt and Wakeup Configuration

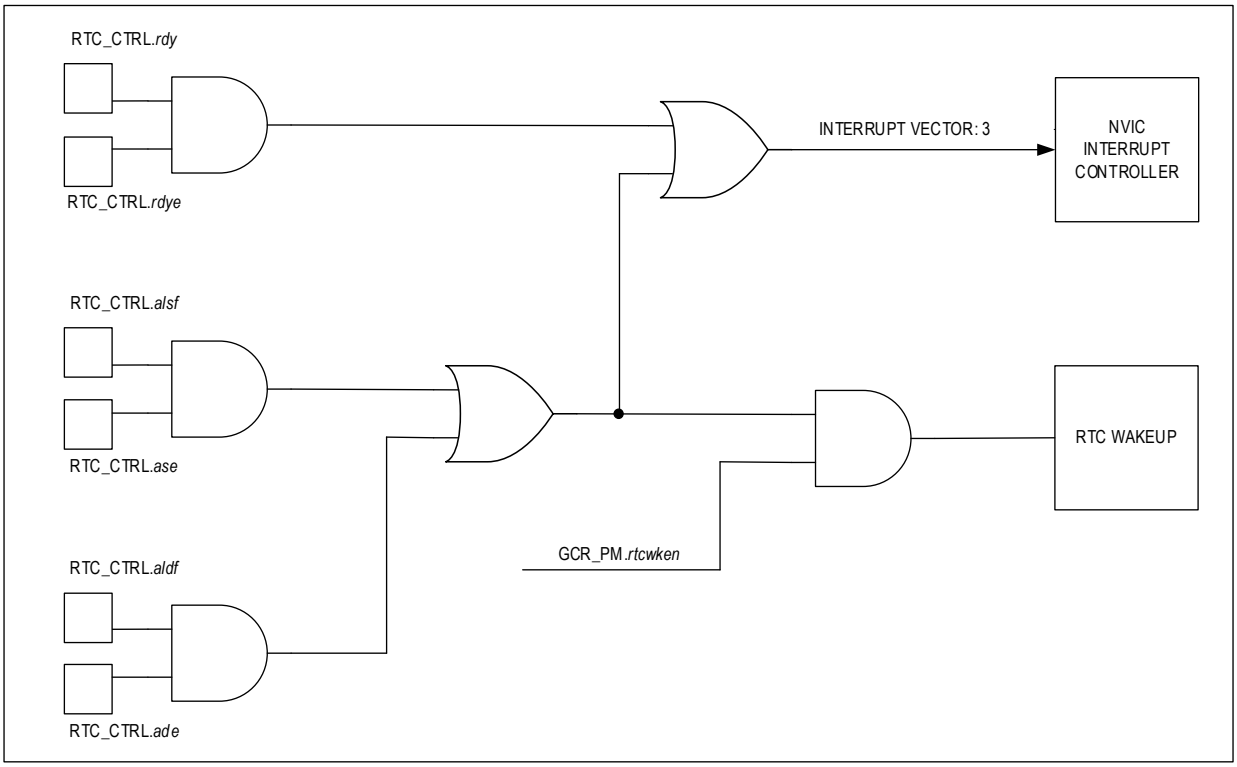
The following are a list of conditions that, when enabled, can generate an RTC interrupt.

- Time-of-Day Alarm
- Sub-second Alarm
- *RTC_CTRL*.rdy field asserted high, signaling read access permitted

RTC can be configured so the time-of-day and sub-second alarms are a wakeup source for exiting the following low power modes:

- BACKUP
- DEEPSLEEP

Figure 18-2: RTC Interrupt/Wakeup Diagram Wakeup Function



Use this procedure to enable the RTC as a wakeup source:

1. Software configures the RTC interrupt enable bits so one or more interrupt conditions will generate an RTC interrupt.
2. Create a RTC IRQ handler function and register the address of the RTC IRQ handler using the NVIC.
3. Software sets *GCR_PM*.*rtcwken* to 1 to enable the system wakeup for by the RTC.
4. Software places the device into the desired low power mode. Refer to section Operating Modes for details on entering DEEPSLEEP or BACKUP mode.

18.4.4 Square-Wave Output

The RTC can output a 50% duty cycle square-wave signal derived from the 32kHz oscillator on a selected device pin. Refer to *Table 18-3*: for the device pins, frequency options, and control fields specific to this device. Frequencies noted as compensated are used during the RTC frequency calibration procedure because they incorporate the frequency adjustments provided by the digital trim function.

Table 18-3: MAX32665/MAX32666 RTC Square-Wave Output Configuration

Function	Option	Control Field
Output Pin*	P0.19: SQWOUT	*MCR_OUTEN*.sqw*out0en* = 1
	P0.27: SQWOUT	*MCR_OUTEN*.sqw*out1en* = 1
Frequency Select	1Hz (Compensated)	*RTC_OSCCTRL*.fre*q_sel* = 0b00
	512Hz (Compensated)	*RTC_OSCCTRL*.fre*q_sel* = 0b01
	4kHz	*RTC_OSCCTRL*.fre*q_sel* = 0b10
	32kHz	*RTC_OSCCTRL*.fre*q_sel* = 0bxx *RTC_OSCCTRL*.32*k_out* = 1
Output Enable	1Hz (Compensated)	*RTC_CTRL*.*sqe* = 1 *RTC_OSCCTRL*.32*k_out* = 0
	512Hz (Compensated)	*RTC_CTRL*.*sqe* = 1 *RTC_OSCCTRL*.32*k_out* = 0
	4kHz	*RTC_CTRL*.*sqe* = 1 *RTC_OSCCTRL*.32*k_out* = 0
	32kHz	*RTC_OSCCTRL*.32*k_out* = 1

**Note: The square-wave output can be enabled on more than one pin if desired.*

Use the following procedure to generate the square wave:

Software configures the fields shown in *Table 18-3* to select the desired frequency.

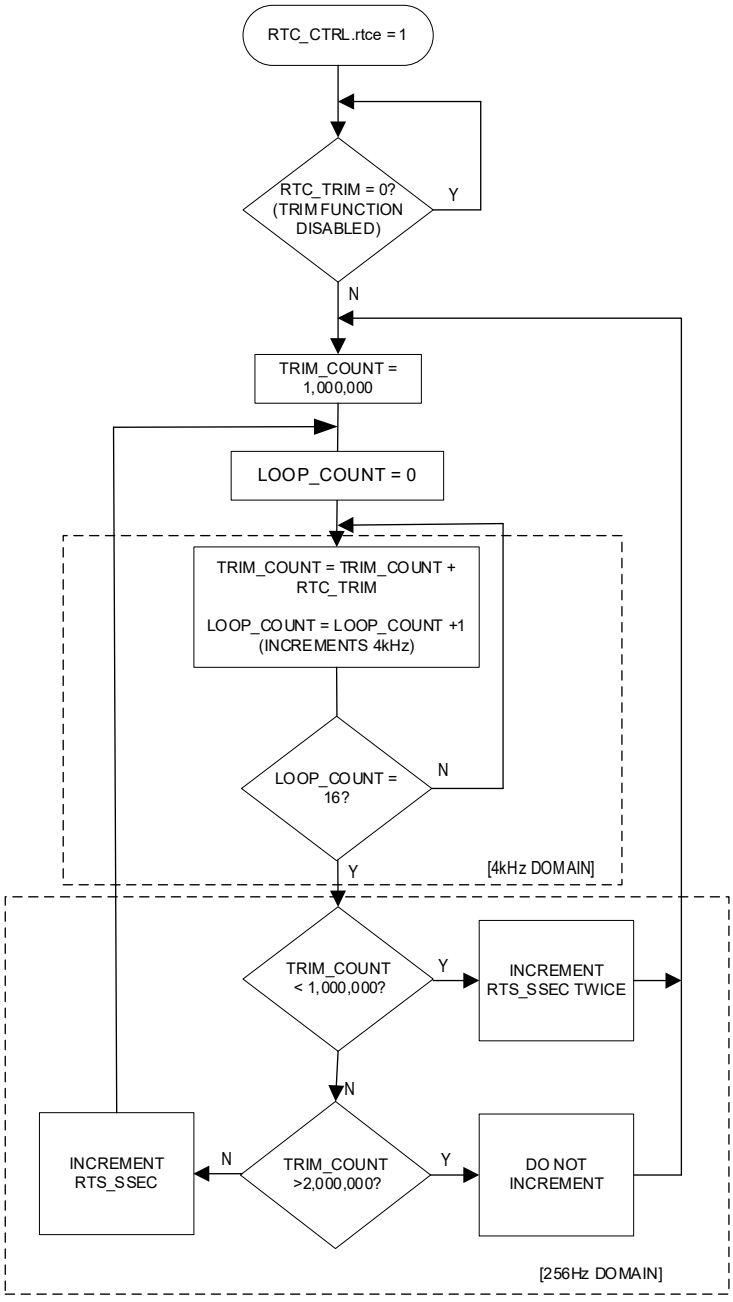
If more than one output pin is available, software configures the fields shown in *Table 18-3* to select the output pin.

Software configures the fields shown in *Table 18-3:* to enable the square-wave output.

18.5 RTC Calibration

A digital trim facility provides the ability to compensate for RTC inaccuracies of up to ± 127ppm when compared against an external reference clock. The trimming function utilizes an independent, dedicated timer which increments the sub-second register based on a user-supplied, two's complement value in the *RTC_TRIM* register as shown in *Figure 18-3*.

Figure 18-3: Internal Implementation of Digital Trim, 4kHz



Complete the following steps to perform an RTC calibration:

1. If not already, software configures and enables one of the compensated calibration frequencies as described in section *Square-Wave Output*.
2. Measure the frequency on the square-wave output pin and compute the deviation from an accurate reference clock.
3. Software clears *RTC_CTRL.rdy* to 0.
4. Wait for *RTC_CTRL.rdy* = 1 by:
   a. Software sets *RTC_CTRL.rdye* to 1 to generate an interrupt when *RTC_CTRL.rdy* = 1, or

b. Software polls *RTC_CTRL.rdy* until the field = 1.

5. Software polls until *RTC_CTRL.busy* = 0 to make sure any previous operations are complete
6. Software sets *RTC_CTRL.we* = 1 to allow access to *RTC_TRIM*.
7. Software writes to *RTC_TRIM* register as desired to correct for measured inaccuracy.
8. Hardware clears *RTC_CTRL*.busy = 0.
9. Software clears *RTC_CTRL.we* to 0.
10. Repeat the process as needed until the desired accuracy is achieved

18.6 Registers

See *Table 3-3* for the the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 18-4: RTC Register Summary

Offset	Register	Description
[0x0000]	*RTC_SEC*	RTC Seconds Counter Register
[0x0004]	*RTC_SSEC*	RTC Sub-Second Counter Register
[0x0008]	*RTC_TODA*	RTC Time-of-Day Alarm Register
[0x000C]	*RTC_SSECA*	RTC Sub-Second Alarm Register
[0x0010]	*RTC_CTRL*	RTC Control Register
[0x0014]	*RTC_TRIM*	RTC 32kHz Oscillator Digital Trim Register
[0x0018]	*RTC_OSCCTRL*	RTC 32kHz Oscillator Control Register

18.7 Register Details

Table 18-5: RTC Seconds Counter Register

RTC Seconds Counter			RTC_SEC	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	sec	R/W	0	**Seconds Counter** This register is a binary count of seconds.

Table 18-6: RTC Sub-Second Counter Register (12-bit)

RTC Sub-Seconds Counter			RTC_SSEC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	R/W	0	**Reserved for Future Use** Do not modify this field.
11:0	ssec	R/W	0	**Sub-Seconds Counter (12-bit)** *RTC_SEC* increments when this field rolls from 0x0FFF to 0x0000

Table 18-7: RTC Time-of-Day Alarm Register

RTC Time-of-Day Alarm			RTC_TODA	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved for Future Use** Do not modify this field.
19:0	tod_alarm	R/W	0	**Time-of-Day Alarm** Sets the time-of-day alarm from 1 second up to 12 days. When this field matches *RTC_SEC*[19:0], an RTC system interrupt is generated.

Table 18-8: RTC Sub-Second Alarm Register

RTC Sub-Second Alarm			RTC_SSECA	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	ssec_alarm	R/W	0	**Sub-second Alarm (4kHz)** Sets the starting and reload value of the internal sub-second alarm counter. The internal counter increments and generates an alarm when the internal counter rolls from 0xFFFF FFFF to 0x0000 0000.

Table 18-9: RTC Control Register

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field.
15	we	R/W	0*	**Write Enable** This field controls access to the *RTC_TRIM* register, the RTC enable (*RTC_CTRL.rtce*) fields. 1: Writes to the *RTC_TRIM* register and the *RTC_CTRL.rtce* field are allowed. 0: Writes to the *RTC_TRIM* register and the *RTC_CTRL.rtce* field are ignored. **Note: Reset on System Reset, Soft Reset, and GCR_RSTR0.rtc assertion.*
14	acre	R/W	0	**Asynchronous Counter Read Enable** Set this field to 1 to allow direct read access of the *RTC_SEC* and *RTC_SSEC* registers without waiting for *RTC_CTRL.rdy.* Multiple consecutive reads of *RTC_SEC* and *RTC_SSEC* must be executed until two consecutive reads are identical to ensure data accuracy. *0: RTC_SEC* and *RTC_SSEC* registers are synchronized and should only be accessed while *RTC_CTRL.rdy* = 1. *1: RTC_SEC* and *RTC_SSEC* registers are asynchronous and will require software interaction to ensure data accuracy.
13	-	R/W	0	**Reserved for Future Use** Do not modify this field.
12:11	-	R/W	0	**Reserved for Future Use** Do not modify this field.
10:9	ft	R/W	0*	**Frequency Output Select** Selects the RTC-derived frequency to output on the square wave output pin. See *Table 18-3* for configuration details. 0b00: 1Hz (Compensated) 0b01: 512Hz (Compensated) 0b1x: 4kHz **Note: Reset on POR only.*
8	sqe	R/W	0*	**Square Wave Output Enable** Enables the square wave output. *Table 18-3* 0: Disabled. 1: Enabled. *Note: Reset on POR only.
7	alsf	R/W	0*	**Sub-second Alarm Interrupt Flag** This interrupt flag is set when a sub-second alarm condition occurs. This flag is a wake-up source for the processor. 0: No sub-second alarm pending. 1: Sub-second interrupt pending. **Note: Reset on POR only.*

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
6	aldf	R/W	0*	**Time-of-Day Alarm Interrupt Flag** This interrupt flag is set by hardware when a time-of-day alarm occurs. 0: No Time-of-Day alarm interrupt pending. 1: Time-of-day interrupt pending. **Note: Reset on POR only.*
5	rdye	R/W	0*	**RTC Ready Interrupt Enable** 0: Disabled. 1: Enabled. **Note: Reset on System Reset, Soft Reset, and GCR_RSTR0.rtc assertion.*
4	rdy	R/W0	0*	**RTC Ready** This bit is set to 1 for 120µs by hardware once a hardware update of the *RTC_SEC* and *RTC_SSEC* registers has occurred. Software should read *RTC_SEC* and *RTC_SSEC* while this hardware bit is set to 1. Software can clear this bit at any time. An RTC interrupt will be generated if *RTC_CTRL.rdye* = 1. 0: Software reads of *RTC_SEC* and *RTC_SSEC* are invalid. 1: Software reads of *RTC_SEC* and *RTC_SSEC* are valid. **Note: Reset on System Reset, Soft Reset, and GCR_RSTR0.rtc assertion.*
3	busy	RO	0*	**RTC Busy Flag** This field is set to 1 by hardware to indicate a register update is in progress when software writes to: • *RTC_SEC* register • *RTC_SSEC* register • *RTC_TRIM* register The following bits cannot be written if this field is set: • *RTC_CTRL*.rtce • *RTC_CTRL*.ade • *RTC_CTRL*.ase • *RTC_CTRL*.rdye • *RTC_CTRL*.aldf • *RTC_CTRL*.alsf • *RTC_CTRL*.sqe • *RTC_CTRL*.ft • *RTC_CTRL*.acre This field is automatically cleared by hardware when the update is complete. Software should poll this field for 0 after changing RTC registers before making any other RTC register changes. 0: RTC not busy. 1: RTC busy. **Note: Reset on POR only.*
2	ase	R/W	0*	**Sub-Second Alarm Interrupt Enable** Check the *RTC_CTRL.busy* flag after writing to this field to determine when the RTC synchronization is complete. 0: Disable. 1: Enable. **Note: Reset on POR only.*

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
1	ade	R/W	0*	**Time-of-Day Alarm Interrupt Enable** Check the *RTC_CTRL.busy* flag after writing to this field to determine when the RTC synchronization is complete. 0: Disable. 1: Enable. **Note: Reset on POR only.*
0	rtce	R/W	0*	**Real-Time Clock Enable** The RTC write enable (*RTC_CTRL.we*) bit must be set and RTC Busy (*RTC_CTRL.busy*) must read 0 before writing to this field. After writing to this bit, check the*RTC_CTRL.busy* flag for 0 to determine when the RTC synchronization is complete. 0: Disabled. 1: Enabled. **Note: Reset on POR only.*

Table 18-10: RTC 32KHz Oscillator Digital Trim Register

RTC 32KHz Oscillator Digital Trim			RTC_TRIM	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	vrtc_tmr	R/W	0*	**VRTC Time Counter** Hardware increments this field every 32s while the RTC is enabled. **Note: Reset on POR only.*
7:0	trim	R/W	0*	**RTC Trim** This field specifies the two's complement value of the trim resolution. Each increment or decrement of the field adds or subtracts 1ppm at each 4kHz clock value with a maximum correction of ± 127ppm. **Note: Reset on POR only.*

Table 18-11: RTC 32KHz Oscillator Control Register

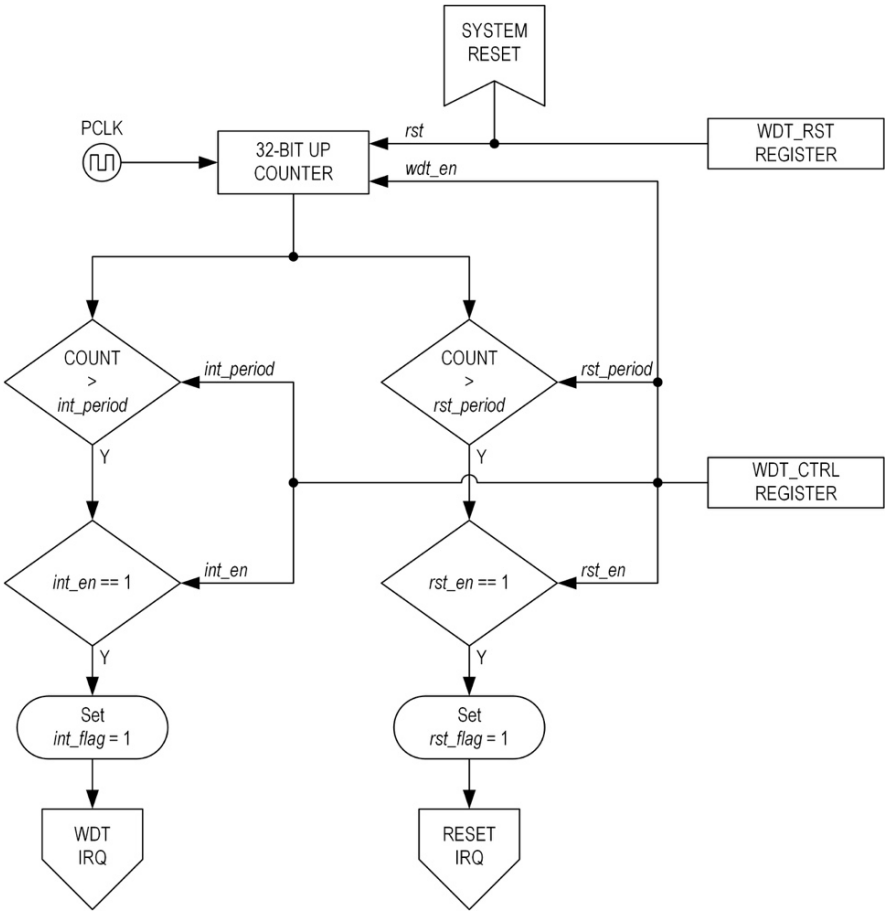
RTC Oscillator Control			RTC_OSCCTRL	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	R/W	0	**Reserved for Future Use** Do not modify this field.
5	32kout	R/W	0	**RTC Square-Wave Output** 0: Disabled. 1: Enables the 32kHz oscillator output or the external clock source is output on square-wave output pin. See *Table 18-3* for configuration details. **Note: Reset on POR only.*
4	bypass	R/W	0	**RTC Crystal Bypass** This field disables the RTC oscillator and allows an external clock source to be driven on the 32KIN pin. 0: Disable bypass. RTC timebase is external 32kHz crystal. 1: Enable bypass. RTC timebase is external square wave driven on 32KIN. **Note: Reset on POR only.*
3:0	-	R/W	0b1001	**Reserved for Future Use** Software must not modify this field from its current value.

19. Watchdog Timer (WDT)

The watchdog timer protects against corrupt or unreliable software, power faults, and other system-level problems, which may place the microcontroller into an improper operating state. When the application is executing properly, application software periodically resets the watchdog counter. If the watchdog timer interrupt is enabled and the software does not reset the counter within the interrupt period (*WDTn_CTRL.int_period*), the watchdog timer generates a watchdog timer interrupt. If the watchdog timer reset is enabled and the software does not reset the counter within the reset period (*WDTn_CTRL.rst_period*), the watchdog timer generates a system reset.

Figure 19-1 shows the block diagram of the watchdog timers.

Figure 19-1: Watchdog Timer Block Diagram



19.1 Features

- Sixteen programmable time periods for the watchdog interrupt 2^{16} through 2^{31} PCLK cycles
- Sixteen programmable time periods for the watchdog reset 2^{16} through 2^{31} PCLK cycles
- The watchdog timer counter is reset on all forms of reset

19.2 Usage

Utilizing the watchdog timer in the application software is straightforward. As early as possible in the application software, enable the watchdog timer interrupt and the watchdog timer reset. Periodically the application software must write to the *WDTn_RST* register to reset the watchdog counter. If program execution becomes lost, the watchdog timer interrupt

occurs, giving the system a “last chance” to recover from whatever circumstance caused the improper code execution. The interrupt routine may either attempt to repair the situation or allow the watchdog timer reset to occur. If a system software failure occurs, the interrupt is not executed, and the watchdog system reset recovers operation.

As soon as possible after a reset, the application software should interrogate the *WDTn_CTRL.rst_flag* to determine if the reset event resulted from a watchdog timer reset. If so, the application software should assume that a program execution error occurred and take whatever steps necessary to guard against a software corruption issue.

19.3 Interrupt and Reset Period Timeout Configuration

Each watchdog timer supports two independent timeout periods, the interrupt period timeout and reset period timeout.

- **Interrupt Period Timeout:** *WDTn_CTRL.int_period* sets the number of PCLK cycles until a watchdog timer interrupt is generated. This period must be less than the Reset Period Timeout for the watchdog timer interrupt to occur.
- **Reset Period Timeout:** *WDTn_CTRL.rst_period* sets the number of PCLK cycles until a system reset event occurs.

The interrupt and reset period timeouts are calculated using *Equation 19-1* and *Equation 19-2,* respectively, where $f_{PCLK} = {}^{f_{SYSCLK}}/_{2}$. *Table 19-1* shows example interrupt period timeout values.

Equation 19-1: Watchdog Timer Interrupt Period

$$T_{INT\_PERIOD} = \left(\frac{1}{f_{PCLK}}\right) \times 2^{(31-WDT\_CTRL.int\_period)}$$

Equation 19-2. Watchdog Timer Reset Period

$$T_{RST\_PERIOD} = \left(\frac{1}{f_{PCLK}}\right) \times 2^{(31-WDT\_CTRL.rst\_period)}$$

Table 19-1*: Watchdog Timer Interrupt Period f_{SYS_CLK} = 96MHz and f_{PCLK} = 48MHz*

WDTn_CTRL .int_period	T_{INT_PERIOD} (SECONDS)
15	0.001
14	0.003
13	0.005
12	0.011
11	0.022
10	0.044
9	0.087
8	0.175
7	0.350
6	0.699
5	1.398
4	2.796
3	5.592
2	11.185
1	22.370
0	Disabled

19.4 Timed Access Protection

The WDT is a critical system safeguard protected against random accesses that would enable, disable, or reset the watchdog timer.

During consecutive instruction cycles, the timed-access protection requires software to write two specific values to the timed-access register *WDTn_RST.wdt_rst*. This simultaneously resets the WDT and unlocks access to *WDTn_CTRL.wdt_en* for a period of one PCLK cycle. Non-consecutive writes to *WDTn_RST.wdt_rst* does not cause a reset of the WDT. Attempts to modify *WDTn_CTRL.wdt_en* after the one PCLK cycle window are ignored.

1. Write *WDTn_RST.wdt_rst*: 0xA5
2. Write *WDTn_RST.wdt_rst*: 0x5A

19.5 Enabling the Watchdog Timer

The watchdog timers are free-running and require a protected sequence of writes to enable the watchdog timers to prevent an unintended reset during the enable process.

19.5.1 Enable sequence

1. Write *WDTn_RST.wdt_rst*: 0xA5
2. Write *WDTn_RST.wdt_rst*: 0x5A
3. Set *WDTn_CTRL.wdt_en* to 1

19.6 Disabling the Watchdog Timer

The watchdog timers can be disabled by software manually or by the microcontroller automatically.

19.6.1 Manual Disable

Setting *WDTn_CTRL.wdt_en* to 0 disables the watchdog timer.

19.6.2 Automatic Disable

A POR event automatically disables the watchdog timers by setting *WDTn_CTRL.wdt_en* to 0.

Note: The watchdog timers remain enabled during all other types of reset.

19.7 Resetting the Watchdog Timer

To prevent a WDT interrupt, reset, or both, application software must use the timed-access procedure above to reset the WDT before an interrupt or reset timeout occurring.

19.7.1 Reset Sequence

1. Write *WDTn_RST.wdt_rst*: 0xA5
2. Write *WDTn_RST.wdt_rst*: 0x5A

19.8 Detection of a Watchdog Reset Event

Multiple hardware and software events can cause a system reset. If the watchdog timer is being used, the application software should check the *WDTn_CTRL.rst_flag* to determine if the reset resulted from a watchdog reset. Application software is responsible for taking appropriate actions if a watchdog reset occurred.

19.9 Registers

See *Table 3-3* for the the base address of this peripheral/module. Each instance has its own, independent set of the registers shown in *Table 19-2*. Register names for a specific instance are defined by replacing "n" with the instance number.

For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 19-2*: Watchdog Timer Register Offsets, Names and Descriptions*

Offset	Register Name	Description
[0x0000]	*WDTn_CTRL*	Watchdog Timer 0 Control Register
[0x0004]	*WDTn_RST*	Watchdog Timer 0 Reset Register

19.10 Register Details

Table 19-3*: Watchdog Timer Control Register*

Watchdog Timer Control Register			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31	rst_flag	R/W	0	**WDT Reset Flag** If this field is set, a watchdog system reset occurred. This field is set to 0 on a POR and is not affected by other resets. 0: Normal operation 1: Watchdog reset occurred.
30:12	-	DNM	0	**Reserved, Do Not Modify**
11	rst_en	R/W	0	**WDT Reset Enable** Enable/disable system reset if the *rst_period* expires. This field is set to 0 on a POR and is not affected by other resets. 0: Disabled 1: Enabled.
10	int_en	R/W	0	**WDT Interrupt Enable** Enable or disable the watchdog interrupt. This field is set to 0 on a POR and is not affected by other resets. 0: Disabled 1: Enabled
9	int_flag	R/W1C	0	**WDT Interrupt Flag** If set, the watchdog interrupt period has occurred. This field is set to 0 on a POR and is not affected by other resets. 0: Normal operation 1: Interrupt period expired. Generates an interrupt if *WDTn_CTRL.int_en* = 1.
8	wdt_en	R/W	0	**WDT Enable** Enable or disable the watchdog timer. The following sequence of writes must be performed to enable the watchdog timer. This field is set to 0 on a POR and is not affected by other resets. 1. Write *WDTn_RST.wdt_rst*: 0xA5 2. Write *WDTn_RST.wdt_rst*: 0x5A 3. Write *wdt_en*: 1 0: Disabled 1: Enabled

Watchdog Timer Control Register			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
7:4	rst_period	R/W	0	**WDT Reset Period** Sets the number of PCLK cycles until a system reset occurs if the watchdog timer is not reset. This field is set to 0 on a POR and is not affected by other resets. 0xF: $2^{16} \times t_{PCLK}$ 0xE: $2^{17} \times t_{PCLK}$ 0xD: $2^{18} \times t_{PCLK}$ 0xC: $2^{19} \times t_{PCLK}$ 0xB: $2^{20} \times t_{PCLK}$ 0xA: $2^{21} \times t_{PCLK}$ 0x9: $2^{22} \times t_{PCLK}$ 0x8: $2^{23} \times t_{PCLK}$ 0x7: $2^{24} \times t_{PCLK}$ 0x6: $2^{25} \times t_{PCLK}$ 0x5: $2^{26} \times t_{PCLK}$ 0x4: $2^{27} \times t_{PCLK}$ 0x3: $2^{28} \times t_{PCLK}$ 0x2: $2^{29} \times t_{PCLK}$ 0x1: $2^{30} \times t_{PCLK}$ 0x0: $2^{31} \times t_{PCLK}$
3:0	int_period	R/W	0	**WDT Interrupt Period** Sets the number of PCLK cycles until a watchdog timer interrupt is generated. This field is set to 0 on a POR and is not affected by other resets. 0xF: $2^{16} \times t_{PCLK}$ 0xE: $2^{17} \times t_{PCLK}$ 0xD: $2^{18} \times t_{PCLK}$ 0xC: $2^{19} \times t_{PCLK}$ 0xB: $2^{20} \times t_{PCLK}$ 0xA: $2^{21} \times t_{PCLK}$ 0x9: $2^{22} \times t_{PCLK}$ 0x8: $2^{23} \times t_{PCLK}$ 0x7: $2^{24} \times t_{PCLK}$ 0x6: $2^{25} \times t_{PCLK}$ 0x5: $2^{26} \times t_{PCLK}$ 0x4: $2^{27} \times t_{PCLK}$ 0x3: $2^{28} \times t_{PCLK}$ 0x2: $2^{29} \times t_{PCLK}$ 0x1: $2^{30} \times t_{PCLK}$ 0x0: $2^{31} \times t_{PCLK}$

Table 19-4: Watchdog Timer Reset Register

Watchdog Timer Reset Register			WDTn_RST	[0x0004]
Bits	**Register Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	wdt_rst	R/W	-	**Reset Register** The following is the required reset sequence to reset the watchdog and prevent a watchdog timer interrupt or watchdog system reset. This field is set to 0 on a POR and is not affected by other resets. • Write *WDTn_RST.wdt_rst*: 0x000000A5 • Write *WDTn_RST.wdt_rst*: 0x0000005A

20. Semaphores

The semaphore peripheral allows multiple cores in a system to cooperate when accessing shared resources. The peripheral contains eight semaphore registers that can be atomically set and cleared. Reading the status field of a semaphore register returns the current state of the status field and, if the field is 0, it automatically sets the status to 1. The semaphore status register reflects the state of each of the semaphore's statuses. The status register enables checking each semaphore's states, but it is not guaranteed that the semaphore status fields cannot change after checking the status register's value.

It is left to the discretion of the software architect to decide how and when the semaphores are used and how they are allocated. Existing hardware does not have to be modified for this type of cooperative sharing, and the use of semaphores is exclusively within the software domain.

20.1 Instances

There is one instance of the semaphore peripheral, as shown in *Table 20-1*.

Table 20-1: MAX78000 Semaphore Instances

Instance	Number of Semaphores
SEMA	8

20.2 Multiprocessor Communications

The software running on CPU0 and CPU1 must know semaphore operation. Each processor must poll the *SEMA_STATUS* register to understand the desired communication between them.

20.2.1 Reset

Globally reset the semaphore peripheral by setting *GCR_RSTR1.smphr* to 1.

20.3 Registers

See *Table 3-3* for the the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets. If multiple instances are provided, each will have a unique base address. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific reset, if applicable.

Table 20-2: Semaphore Register Summary

Offset	Register	Name
[0x0000]	*SEMA_SEMAPHORES0*	Semaphore 0 Register
[0x0004]	*SEMA_SEMAPHORES1*	Semaphore 1 Register
[0x0008]	*SEMA_SEMAPHORES2*	Semaphore 2 Register
[0x000C]	*SEMA_SEMAPHORES3*	Semaphore 3 Register
[0x0010]	*SEMA_SEMAPHORES4*	Semaphore 4 Register
[0x0014]	*SEMA_SEMAPHORES5*	Semaphore 5 Register
[0x0018]	*SEMA_SEMAPHORES6*	Semaphore 6 Register
[0x0020]	*SEMA_SEMAPHORES7*	Semaphore 7 Register
[0x0100]	*SEMA_STATUS*	Semaphore Status Register

20.4 Register Details

Table 20-3: Semaphore 0 Register

Semaphore 0			SEMA_SEMAPHORES0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and, if 0, it automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status0* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 20-4: Semaphore 1 Register

Semaphore 1			SEMA_SEMAPHORES1	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and, if 0, it automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status1* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 20-5: Semaphore 2 Register

Semaphore 2			SEMA_SEMAPHORES2	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and, if 0, it automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status2* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 20-6: Semaphore 3 Register

Semaphore 3			SEMA_SEMAPHORES3	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and, if 0, it automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status3* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 20-7: Semaphore 4 Register

Semaphore 4			SEMA_SEMAPHORES4	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and, if 0, it automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status4* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 20-8: Semaphore 5 Register

Semaphore 5			SEMA_SEMAPHORES5	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and, if 0, it automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status5* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 20-9: Semaphore 6 Register

Semaphore 6			SEMA_SEMAPHORES6	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and, if 0, it automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status6* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 20-10: Semaphore 7 Register

Semaphore 7			SEMA_SEMAPHORES7	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	status	*	0	**Semaphore Status** Reading this field returns its current value and, if 0, it automatically sets the field to 1. Write 0 to clear this field. Modifications to this field are mirrored in the *SEMA_STATUS.status7* field. 0: Semaphore is available. 1: Semaphore is taken.

Table 20-11: Semaphore Status Register

Semaphore Status			SEMA_STATUS	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**

Semaphore Status			SEMA_STATUS	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
7	status7	R	0	**Semaphore 7 Status** This field mirrors the semaphore 7 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMAPHORES7.status* is 0 1: *SEMA_SEMAPHORES7.status* is 1
6	status6	R	0	**Semaphore 6 Status** This field mirrors the semaphore 6 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMAPHORES6.status* is 0 1: *SEMA_SEMAPHORES6.status* is 1
5	status5	R	0	**Semaphore 5 Status** This field mirrors the semaphore 5 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMAPHORES5.status* is 0 1: *SEMA_SEMAPHORES5.status* is 1
4	status4	R	0	**Semaphore 4 Status** This field mirrors the semaphore 4 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMAPHORES4.status* is 0 1: *SEMA_SEMAPHORES4.status* is 1
3	status3	R	0	**Semaphore 3 Status** This field mirrors the semaphore 3 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMAPHORES3.status* is 0 1: *SEMA_SEMAPHORES3.status* is 1
2	status2	R	0	**Semaphore 2 Status** This field mirrors the semaphore 2 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMAPHORES2.status* is 0 1: *SEMA_SEMAPHORES2.status* is 1
1	status1	R	0	**Semaphore 1 Status** This field mirrors the semaphore 1 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMAPHORES1.status* is 0 1: *SEMA_SEMAPHORES1.status* is 1
0	status0	R	0	**Semaphore 0 Status** This field mirrors the semaphore 0 status field. Reads from this field do not affect the corresponding semaphore's status field. 0: *SEMA_SEMAPHORES0.status* is 0 1: *SEMA_SEMAPHORES0.status* is 1

21. 1-Wire Master (OWM)

The device provides a 1-Wire master (OWM) that you can use to communicate with one or more external 1-Wire slave devices using a single-signal, combined clock, data protocol. The OWM is contained in the OWM module. The OWM module handles the lower-level details (including timing and drive modes) required by the 1-Wire protocol, allowing the CPU to communicate over the 1-Wire bus at a logical data level.

The OWM provides the following features:

- Flexible, 1-Wire timing generation (required 1MHz timing base) using the OWM module clock frequency, which is in turn derived from the current system clock source. You can also prescale the OWM module clock to allow proper 1-Wire timing generation using a range of base frequencies.
- Automatic generation of proper 1-Wire time slots for both standard and overdrive timing modes.
- Flexible configuration for 1-Wire line pullup modes: options for internal pullup, external fixed pullup, and optional external strong pullup are available.
- Long-line compensation and bit banging (direct firmware drive) modes.
- 1-Wire reset generation and presence-pulse detection.
- Generation of 1-Wire read and write time slots for single-bit and eight-bit byte transmissions.
- Search ROM Accelerator (SRA) mode, which simplifies the generation of multiple-bit time slots and discrepancy resolution required when completing the Search ROM function to determine the IDs of multiple, unknown 1-Wire slaves on the bus.
- Transmit data completion, received data available, presence pulse detection, and 1-Wire line-error condition interrupts.

For more information about the Analog Devices 1-Wire protocol and supporting devices, refer to the following resources:

- *AN937: The Book of iButton® Standards*
- *AN1796: Overview of 1-Wire Technology and Its Use*
- *AN187: 1-Wire Search Algorithm*

21.1 Instances

There is one instance of this peripheral.

iButton is a registered trademark of Maxim Integrated Products, Inc.

21.2 Pins and Configuration

The OWM pin mapping is shown in *Table 21-1*.

Table 21-1: OWM Pin to Alternate Function Mapping

Alternate Function	Alternate Function 1	Alternate Function 2	Alternate Function 4	Pin Name	Direction	Signal Description
OWM_IO	SPIXF_SDIO2	-	TMR4	P0.4	I/O	1-Wire I/O
	SPIXR_SDIO2	QSPI0_SDIO2	TMR0	P0.12	I/O	1-Wire I/O
	AIN0/AIN0P	QSPI1_SS0	TMR4	P0.16	I/O	1-Wire I/O
	PCM_LRCLK	QSPI2_SS0	TMR0	P0.24	I/O	1-Wire I/O
OWM_PE	SPIXF_SDIO3	-	TMR5	P0.5	O	Pullup Enable Output
	SPIXR_SDIO3	QSPI0_SDIO3	TMR1	P0.13	O	Pullup Enable Output
	AIN1/AIN0P	QSPI1_MOSI/ SDIO0	TMR5	P0.17	O	Pullup Enable Output
	PCM_DOUT	QSPI2_MOSI/ SDIO0	TMR1	P0.25	O	Pullup Enable Output

21.2.1 Pin Configuration

Perform the following steps to configure the GPIO for OWM peripheral usage:

1. Enable the alternate function mode for pins P1.30 and P1.31 by setting GPIO1_EN[30:31] to 0.
2. Set alternate function 1 (AF1) by setting GPIO1_AF_SEL[30:31] to 0.

21.2.2 1-Wire I/O (OWM_IO)

The 1-Wire IO signal is a bidirectional I/O that is used to directly drive the external 1-Wire bus. As described in the 1-Wire interface specification, this I/O is generally driven as an open-drain output. The 1-Wire bus requires a common pullup to return the 1-Wire bus line to an idle high state when no master or slave device is actively driving the line low. This pullup can consist of a fixed resistor pullup (connected to the 1-Wire bus outside the microcontroller), an internal pullup enabled by setting *OWM_CFG.int_pullup_enable* to 1, or an OWM module controlled external pullup enabled by setting *OWM_CFG.ext_pullup_mode* to 1.

21.2.3 Pullup Enable (OWM_PE)

The 1-Wire pullup enable (PE) signal is an active high output used to enable an optional external pullup on the 1-Wire bus. This pullup is intended to provide a stronger (lower impedance) pullup on the 1-Wire bus under certain circumstances, such as during overdrive mode.

21.2.4 Clock Configuration

To correctly generate the timing required by the 1-Wire protocol in Standard or Overdrive timing modes, the OWM clock must be set to achieve $f_{owmclk} = 1\text{MHz}$. This clock generates both the Standard and Overdrive timing, so it does not need adjustment when transitioning from Standard to Overdrive mode or vice versa.

The OWM peripheral uses the system peripheral clock, PCLK, divided by the value in the *OWM_CLK_DIV_1US.divisor* field as shown in *Equation 21-1*, where $f_{PCLK} = {}^{f_{SYSCLK}}/_{2}$.

Equation 21-1: OWM 1MHz Clock Frequency

$$f_{owmclk} = 1MHz = \frac{f_{PCLK}}{OWM\_CLK\_DIV\_1US.divisor}$$

If the system clock is set to 120MHz, $f_{PCLK} = 60MHz$, the *OWM_CLK_DIV_1US.divisor* field should be set to 60 as shown in *Equation 21-2*.

Equation 21-2: OWM Clock Divisor for $f_{SYSCLK} = 120MHz$

$$OWM\_CLK\_DIV\_1US.divisor = \frac{60MHz}{1MHz} = 60$$

21.3 1-Wire Protocol

The general timing and communication protocols used by the OWM interface are those standardized for the 1-Wire network.

Because the 1-Wire interface is a master interface, it initiates and times all communication on the 1-Wire bus. Except for the present pulse generation when a device first connects to the 1-Wire bus, 1-Wire slave devices complete 1-Wire bus communication only as directed by the 1-Wire bus master. From a firmware perspective, the lowest-level timing and electrical details of how the 1-Wire network operates are unimportant. The application can configure the OWM module properly and direct it to complete low-level operations such as reset, read, and write bit/byte operations. Thus, the OWM module on the microcontroller is designed to interface to the 1-Wire bus at a low level.

21.3.1 Networking Layers

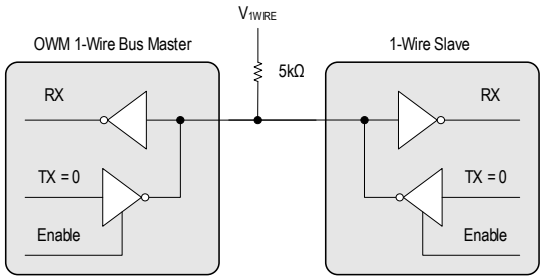
In the *Book of iButton Standards*, the 1-Wire communication protocol is described in terms of the ISO-OSI model (International Organization of Standardization (ISO) Open System Interconnection (OSI) network layer model). Network layers that apply to this description are the Physical, Link, Network, and Transport layers. The Presentation layer would correspond to higher-level application software functions (such as library layers) that implement communication protocols using the 1-Wire layers as a foundation.

21.3.1.1 Bus Interface (Physical Layer)

The 1-Wire communication bus consists of a single data/power line plus ground. devices (either master or slave) that interface to the 1-Wire communication bus using an open-drain (active low) connection, which means that the 1-Wire bus normally idles in a high state.

An external pullup resistor is used to pull the 1-Wire line high when no master or slave device is driving the line. This means that 1-Wire devices do not actively drive the 1-Wire line high. Instead, they either drive the line low or release it (set their output to high impedance) to allow the external resistor to pull the line high. This allows the 1-Wire bus to operate in a wired-AND manner as shown in *Figure 21-1* and avoids bus contention if more than one device attempts to drive the 1-Wire bus at the same time.

Figure 21-1: 1-Wire Signal Interface



21.3.1.2 Reset, Presence Detect, and Data Transfer (Link Layer)

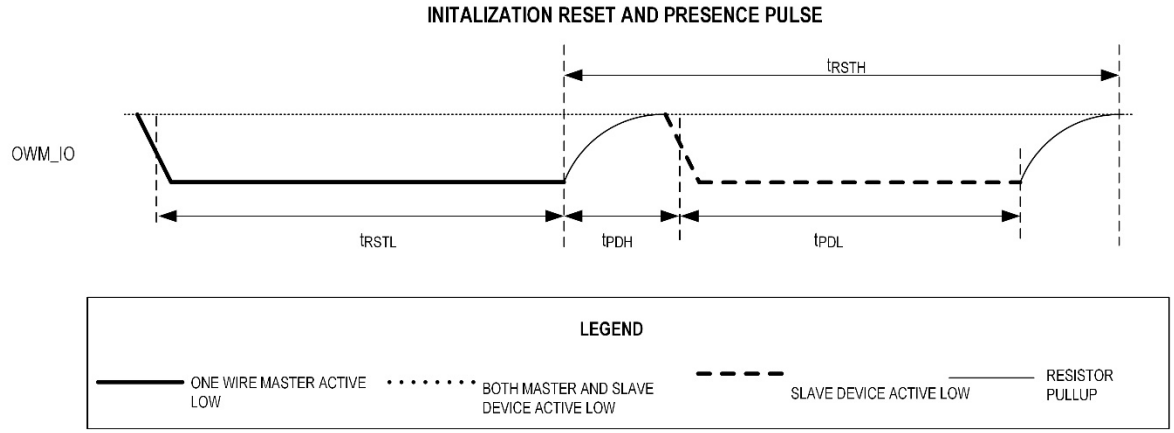
The 1-Wire Bus supports a single master and one or more slave devices (multidrop). Slave devices can connect to and disconnect from the 1-Wire Bus dynamically (as is typically the case with iButton devices that operate using an intermittent

touch contact interface), which means that it is the master's responsibility to poll the bus as needed to determine the number and types of 1-Wire devices that are connected to the bus.

All communication sequences on the 1-Wire Bus are initiated by the OWM. The OWM determines when 1-Wire data transmissions begin, as well as the overall communication speed that is used. There are three different communication speeds supported by the 1-Wire specification: standard speed, overdrive speed, and hyperdrive speed. However, only standard speed and overdrive speed are supported by the OWM peripheral in the devices.

The OWM begins each communication sequence by sending a reset pulse as shown in *Figure 21-2*. This pulse resets all 1-Wire slave devices on the line to their initial states and causes them all to begin monitoring the line for a command from the OWM. Each 1-Wire slave device on the line responds to the reset pulse by sending out a presence pulse. These pulses from multiple 1-Wire slave devices are combined in wired-AND fashion, resulting in a pulse whose length is determined by the slowest 1-Wire slave device on the bus.

Figure 21-2: 1-Wire Reset Pulse



In general, the 1-Wire line must idle in a high state when communication is not taking place. It is possible for the master to pause communication in between time slots. There is no overall "timeout" period that causes a slave to revert to the reset state if the master takes too long between one time slot and the next time slot.

The 1-Wire communication protocol relies on the fact that the maximum allowable length for a bit transfer (write 0/1 or read bit) time slot is less than the minimum length for a 1-Wire reset. At any time, if the 1-Wire line is held low (by the master or by any slave device) for more than the minimum reset pulse time, all slave devices on the line interpret this as a 1-Wire reset pulse.

21.3.1.2.1 Read and Write Time Slots

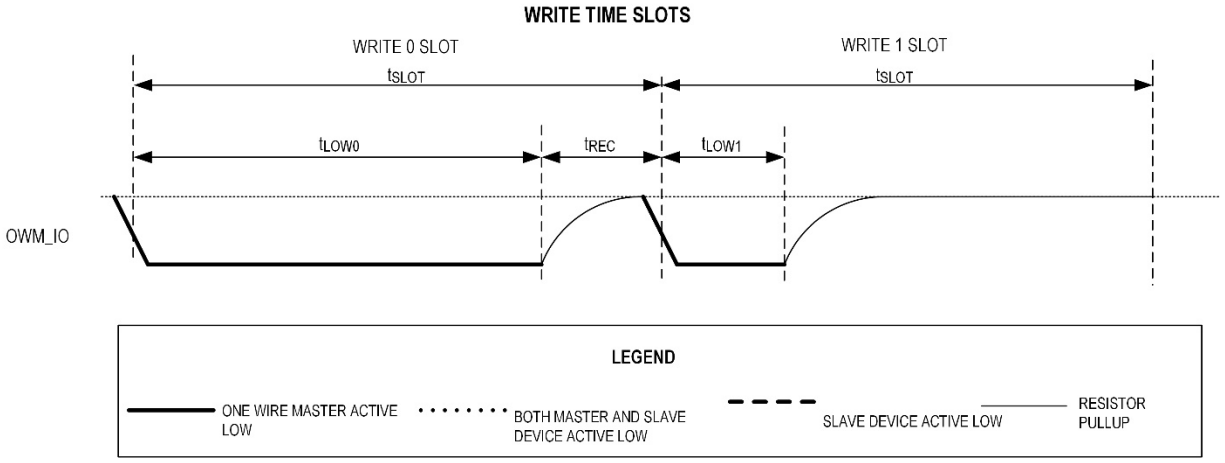
OWM Write Time Slot

All 1-Wire bit time slots are initiated by the 1-Wire bus master and begin with a single falling edge. There is no indication given by the beginning of a time slot whether a read bit or write bit operation is intended, as the time slots all begin in the same manner. Rather, the 1-Wire command protocol enforces agreement between the OWM and slave as to which time slots are used for bit writes and which time slots are used for bit reads.

When multiple bits of a value are transmitted (or read) in sequence, the least significant bit of the value is always sent or received first. The 1-Wire bus is a half-duplex bus, so data is transmitted in only one direction (from master to slave or from slave to master) at any given time.

As shown in *Figure 21-3*, the time slots for writing a 0 bit and writing a 1 bit begin identically, with the falling edge and a minimum-width low pulse sent by the master. To write a one bit, the master releases the line after the minimum low pulse, allowing it to be pulled high. To write a zero bit, the master continues to hold the line low until the end of the time slot.

Figure 21-3: 1-Wire Write Time Slot



From the slave's perspective, the initial falling edge of the time slot triggers the start of an internal timer, and when the proper amount of time has passed, the slave samples the 1-Wire line that is driven by the master. This sampling point is in between the end of the minimum-width low pulse and the end of the time slot.

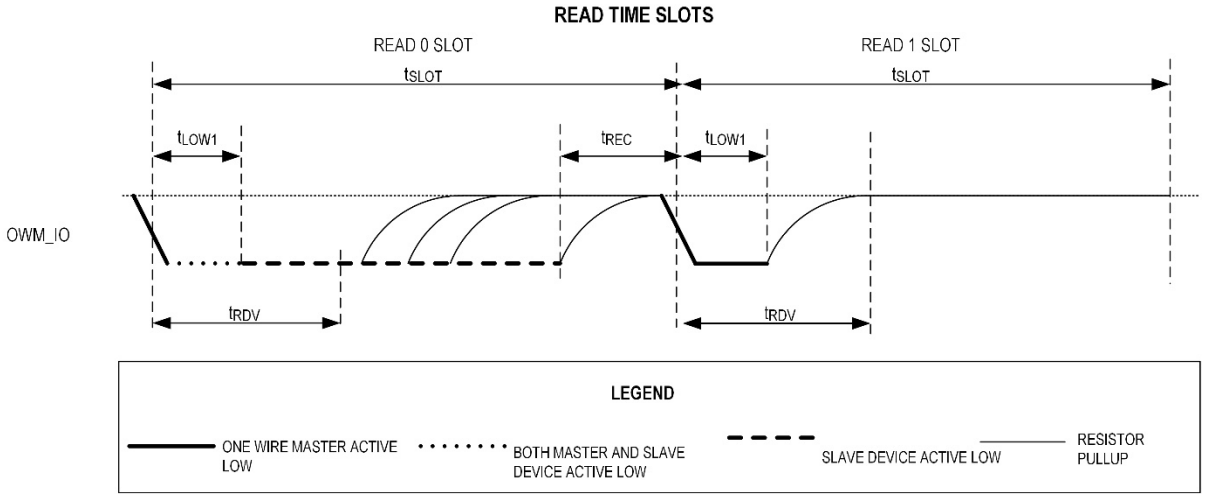
OWM Read Time Slot

As with all 1-Wire transactions, the master initiates all bit read time slots. Like the bit write time slots, the bit read time slot begins with a falling edge. From the master's perspective, this time slot is transmitted identically to the "Write 1 Bit" time slot shown in *Figure 21-3*. The master begins by transmitting a falling edge, holds the line low for a minimum-width period, and then releases the line.

The difference here is that instead of the slave sampling the line, the slave begins transmitting either a 0 (by holding the line low) or a 1 (by leaving the line to float high) after the initial falling edge. The master then samples the line to read the bit value that is transmitted by the slave device.

As an example, *Figure 21-4* shows a sequence in which the slave device transmits data back to the 1-Wire bus master upon request. Note that to transmit a 1 bit, the slave device does not need to do anything. It simply leaves the line alone (to float high) and waits for the next time slot. To transmit a 0 bit, the slave device holds the line low until the end of the time slot.

Figure 21-4: 1-Wire Read Time Slot



21.3.1.2.2 Standard Speed and Overdrive Speed

By default, all 1-Wire communications following reset begin at the lowest rate of speed (that is, standard speed). For 1-Wire devices that support it, it is possible for the OWM to increase the rate of communication from standard speed to overdrive speed by sending the appropriate command.

The protocols and time slots operate identically for standard and overdrive speeds. The difference comes in the widths of the time slots and pulses. The OWM automatically adjusts the timings based on the setting of the *OWM_CFG.overdrive* field.

If a 1-Wire slave device receives a standard speed reset pulse, it resets and reverts to standard speed communication. If the device is already communicating in overdrive mode, and it receives a reset pulse at the overdrive speed, it resets but remains in overdrive mode.

21.3.1.3 ROM Commands (Network Layer)

Following the initial 1-Wire reset pulse on the bus, all slave 1-Wire devices are active, which means that they are monitoring the bus for commands. Because the 1-Wire bus can have multiple slave devices present on the bus at any time, the OWM must go through a process (defined by the 1-Wire command protocol) to activate only the 1-Wire slave device that it intends to communicate with and deactivate all others. This is the purpose of the ROM commands (network layer) shown in Table 21-2.

Table 21-2: 1-Wire ROM Commands

ROM Command	Hex Value
Read ROM	0x33
Match ROM	0x55
Search ROM	0xF0
Skip ROM	0xCC
Overdrive Skip ROM	0x3C
Overdrive Match ROM	0x69
Resume Communication	0xA5

The ROM command layer relies on the fact that all 1-Wire slave devices are assigned a globally-unique, 64-bit ROM ID. This ROM ID value is factory programmed to ensure that no two 1-Wire slave devices have the same value.

Figure 21-5 is a visual representation of the 1-Wire ROM ID fields and shows the organization of the fields within the 64-bit ROM ID for a device.

Figure 21-5: 1-Wire ROM ID Fields

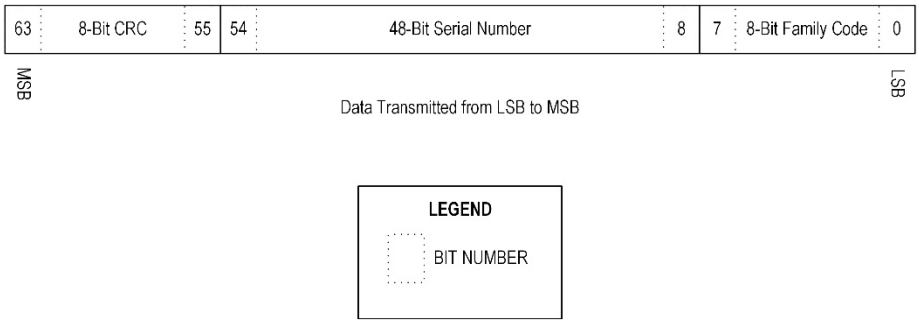


Table 21-3 provides a detailed description of each of the ROM ID fields.

Table 21-3: 1-Wire Slave Device ROM ID Field

Field	Bit Number	Description
Family code	0-7	This 8-bit value is used to identify the type of a 1-Wire slave device.
Unique ID	8-55	This 48-bit value is factory-programmed to give each 1-Wire slave device (within a given family code group) a globally unique identifier.
CRC	56-63	This is the 8-bit, 1-Wire CRC as defined in the *Book of iButton Standards*. The CRC is generated using the polynomial ($x^8 + x^5 + x^4 + 1$).

Note: For certain operations that consist only of writing from the OWM to the slave, it is technically possible for the master to communicate with more than one slave at a time on the same 1-Wire bus. For this to work, the exact same data must be transmitted to all slave devices, and any values read back from the slaves must either be identical as well or must be disregarded by the master device (because different slaves can attempt to transmit different values). The descriptions below assume, however, that the master is communicating with only one slave device at a time because this is the method that is normally used.

As explained above, the ROM ID contents play an important role in addressing and selecting devices on the 1-Wire bus. All devices except one are in an idle/inactive state after the Match ROM command or the Search ROM command is executed. They return to the active state only after receiving a 1-Wire reset pulse.

Devices with overdrive capability are distinguished from others by their family code and two additional ROM commands: Overdrive Skip ROM and Overdrive Match ROM. The first transmission of the ROM command itself takes place at the normal speed that is understood by all 1-Wire devices. After a device with overdrive capability is addressed and set into overdrive mode (that is, after the appropriate ROM command is received), further communication to that device must occur at overdrive speed. Because all deselected devices remain in the idle state as long as no reset pulse of regular duration is detected, even multiple overdrive components can reside on the same 1-Wire bus. A reset pulse of regular duration resets all 1-Wire devices on the bus and simultaneously sets all overdrive-capable devices back to the default standard speed.

21.3.2 Read ROM Command

The Read ROM command allows the OWM to obtain the 8-byte ROM ID of any slave device connected to the 1-Wire bus. Each slave device on the bus responds to this command by transmitting all eight bytes of its ROM ID value starting with the least significant byte (Family Code) and ending with the most significant byte (CRC).

Because this command is addressed to all 1-Wire devices on the bus, if more than one slave is present on the bus there is a data collision as multiple slaves attempt to transmit their ROM IDs at once. This condition is detectable by the OWM because the CRC value does not match the ROM ID value received. In this case, the OWM should reset the 1-Wire bus and select a single slave device on the bus to continue either by using the Match ROM command (if the ROM ID values are already known) or the Search ROM command (if the master has not yet identified some or all devices on the bus).

After the Read ROM command is complete, all slave devices on the 1-Wire bus are selected or active, and communication proceed to the Transport layer.

21.3.3 Skip ROM and Overdrive Skip ROM Commands

The Skip ROM command is used to activate all slave devices present on the 1-Wire bus regardless of their ROM ID. Normally, this command is used when only a single 1-Wire slave device is connected to the bus. After the Skip ROM command is complete, all slave devices on the 1-Wire bus are selected or active and communication proceeds to the Transport layer.

The Overdrive Skip ROM command operates in an identical manner except that running it also causes the receiving slave devices to shift communication speed from standard speed to overdrive speed. The Overdrive Skip ROM command byte itself (0x3C) is transmitted at standard speed. All subsequent communication is sent at overdrive speed.

21.3.4 Match ROM and Overdrive Match ROM Commands

The Match ROM command is used by the OWM to select one and only one slave 1-Wire device when the ROM ID of the device has already been determined. When transmitting this command, the master sends the command byte (that is, 55h for standard speed and 69h for overdrive speed) and then sends the entire 64-bit ROM ID for the device selected, least significant bit first.

During the transmission of the ROM ID by the master, all slave devices monitor the bus. As each bit is transmitted, each of the slave devices compares it against the corresponding bit of their ROM ID. If the bits match, the slave device continues to monitor the bus. If the bits do not match, the slave device transitions to the inactive state (waiting for a 1-Wire reset) and stops monitoring the bus.

At the end of the transmission, at most one slave device is active, which is the slave device whose ROM ID matched the ROM ID that was transmitted. All other slave devices are inactive. Communication then proceeds to the Transport layer for the device that was selected.

The Overdrive Match ROM command operates in an identical manner except that it also causes the slave device that is selected by the command to shift communication speed from standard speed to overdrive speed. The Overdrive Match ROM command byte (69h) and the 64-bit ROM ID bits are transmitted at standard speed. All subsequent communication is sent at overdrive speed.

21.3.5 Search ROM Command

The Search ROM command allows the OWM to determine the ROM ID values of all 1-Wire slave devices connected to the bus using an iterative search process. Each execution of the Search ROM command reveals the ROM ID of one slave device on the bus.

The operation of the Search ROM command resembles a combination of the Read ROM and Match ROM commands. First, all slaves on the bus transmit the least significant bit (Bit 0) of their ROM IDs. Next, all slaves on the bus transmit a complement of the same bit. By analyzing the two bits received, the master can determine whether the Bit 0 values were 0 for all slaves, 1 for all slaves, or a combination of the two. Next, the master selects which slaves remain activated for the next step in the Search ROM process by transmitting the Bit 0 value for the slaves it selects. All slaves whose Bit 0 matches the value transmitted by the master remain active, while slaves with a different Bit 0 value go to the inactive state and do not participate in the remainder of the Search ROM command.

Next, the same process is followed for Bit 1, then Bit 2, and so on until the 63rd bit (most significant bit) of the ROM ID is transmitted. At this point only one slave device remains active, and the master can either continue with communication at the Transport layer or issue a 1-Wire reset pulse to go back for another pass at the Search ROM command.

The *Book of iButton Standards* goes into more detail about the process that is used by the master to obtain ROM IDs of all devices on the 1-Wire bus using multiple executions of the Search ROM command. The algorithm resembles a binary tree search and is used regardless of how many devices are on the bus.

There is no overdrive equivalent version of the Search ROM command.

21.3.6 Search ROM Accelerator Operation

To allow the Search ROM command to process more quickly, the OWM module provides a special accelerator mode for use with the Search ROM command. This mode is activated by setting *OWM_CTRL_STAT.sra_mode* to 1.

When this mode is active, ROM IDs being processed by the Search ROM command are broken into 4-bit nibbles where the current 64-bit ROM ID varies with each pass through the search algorithm. Each 4-bit processing step is initiated by writing the 4-bit value to *OWM_DATA.tx_rx*. This causes the generation of twelve 1-Wire time slots by the OWM as each bit in the 4-bit value (starting with the LSB) results in a read of two bits (all active slaves transmitting bit N of their ROM IDs, then all active slaves transmitting the complement of bit N of their ROM ID), and then a write of a single bit by the OWM.

After the 4-bit processing stage is complete, the return value loaded into *OWM_DATA.tx_rx* consists of 8 bits. The low nibble (bits 0 through 3) contains the four discrepancy flags: one for each ID bit processed. If the discrepancy bit is set to 1, it means that either two slaves with differing ID bits in that position both responded (the 2 bits read were both zero), or

that no slaves responded (the 2 bits read were both 1). If the discrepancy bit is set to 0, then the 2 bits read were complementary (either 0, 1 or 1, 0) meaning that there was no bus conflict.

In this way, at each step in the Search ROM command, the master either follows the ID of the responding slaves or deselects some of the slaves on the bus in case of a conflict. By the time the end of the 64-bit ROM ID is reached (the sixteenth 4-bit group processing step), the combination of all bits from the high nibbles of the received data are equal to the ROM ID of one of the slaves remaining on the bus. Subsequent passes through the Search ROM algorithm are used to determine additional slave ROM ID values until all slaves are identified. Refer to the *Book of iButton Standards* for a detailed explanation of the search function and possible variants of the search algorithm applicable to specific circumstances.

21.3.7 Resume Communication Command

If more than one 1-Wire slave device is on the bus, then the master must specify which one it wishes to communicate with each time a new 1-Wire command (starting with a reset pulse) begins. Using the commands discussed previously, this would normally involve sending the Match ROM command each time, which means the master must explicitly specify the full 64-bit ROM ID of the part it communicates with for each command.

The Resume Communication command provides a shortcut for this process by allowing the master to repeatedly select the same device for multiple commands without having to transmit the full ROM ID each time.

When the OWM selects a single device (using the Match ROM or Search ROM commands), an internal flag called the RC (for Resume Communication) flag is set in the slave device. (Only one device on the bus has this flag set at any one time; the Skip ROM command selects multiple devices, but the RC flag is not set by the Skip ROM command.)

When the master resets the 1-Wire bus, the RC flag remains set. At this point, it is possible for the master to send the Resume Communication command. This command does not have a ROM ID attached to it, but the device that has the RC flag set responds to this command by going to the active state while all other devices deactivate and drop off the 1-Wire bus.

Issuing any other ROM command clears the RC flag on all devices. So, for example, if a Match ROM command is issued for device A, its RC flag is set. The Resume Communication command can then be used repeatedly to send commands to device A. If a Match ROM command is then sent with the ROM ID of device B, the RC flag on device A will clear to 0, and the RC flag on device B is set.

21.4 1-Wire Operation

Once the OWM peripheral is correctly configured, then using the OWM peripheral to communicate with the 1-Wire network involves directing the OWM to generate the proper reset, read, and write operations to communicate with the 1-Wire slave devices used in a specific application.

The OWM handles the following 1-Wire protocol primitives directly in either Standard or Overdrive mode:

- 1-Wire bus reset (including detection of presence pulse from responding slave devices)
- Write single bit (a single write time slot)
- Write 8-bit byte, least significant bit first (eight write time slots)
- Read single bit (a single write-1 time slot)
- Read 8-bit byte, least significant bit first (eight write-1 time slots)
- Search ROM Acceleration Mode allowing the generation of four groups of three time slots (read, read, and write) from a single 4-bit register write to support the Search ROM command

21.4.1 Resetting the OWM

The first step in any 1-Wire communication sequence is to reset the 1-Wire bus. To direct the OWM module to complete a 1-Wire reset, write *OWM_CTRL_STAT.start_ow_reset* to 1. This generates a reset pulse and checks for a replying presence pulse from any connected slave devices.

Once the reset time slot is complete, the *OWM_CTRL_STAT.start_ow_reset* field is automatically cleared to zero. Then, the interrupt flag *OWM_INTFL.ow_reset_done* is set to 1 by hardware. This flag must be cleared by writing a 1 bit to the flag.

If a presence pulse is detected on the 1-Wire bus during the reset sequence (that should normally be the case unless no 1-Wire slave devices are present on the bus), the *OWM_CTRL_STAT.presence_detect* flag is also set to 1. This flag does not result in the generation of an interrupt.

21.5 1-Wire Data Reads

21.5.1 Reading a Single Bit Value from the 1-Wire Bus

The procedure for reading a single bit is like the procedure for writing a single bit because the operation is completed by writing a 1 bit that the slave device either leaves unchanged (to transmit a 1 bit) or overrides by forcing the line low (to transmit a 0 bit).

To read a single bit value from the 1-Wire Bus, complete the following steps:

1. Set *OWM_CFG.single_bit_mode* to 1. This setting causes the OWM to transmit/receive a single bit of data at a time instead of the default 8 bits.
2. Write *OWM_DATA.tx_rx* to 1. Only bit 0 of this field is used in this instance; the other bits in the field are ignored. Writing to the *OWM_DATA* register initiates the read of the bit on the 1-Wire bus.
3. Once the single-bit transmission is complete, hardware sets the interrupt flag *OWM_INTFL.tx_data_empty* to 1. This flag (that triggers an OWM module interrupt if *OWM_INTEN.tx_data_empty* is also set to 1) is cleared by writing a 1 to the flag.
4. As the hardware shifts the bit value out, it also samples the value returned from the slave device. Once this value is ready to read, the interrupt flag *OWM_INTFL.rx_data_ready* is set to 1. If *OWM_INTEN.rx_ready* is set to 1, an OWM module interrupt occurs.
5. Read *OWM_DATA.tx_rx* (only bit 0 is used) to determine the value returned by the slave device. Note that if no slave devices are present or the slaves are not communicating with the master, bit 0 remains set to 1.

21.5.2 Reading an 8-Bit Value from the 1-Wire Bus

The procedure for reading an 8-bit byte is like the procedure for writing an 8-bit byte because the operation is completed by writing eight 1 bits that the slave device either leaves unchanged (to transmit 1 bits) or overrides by forcing the line low (to transmit 0 bits).

1. Set *OWM_CFG.single_bit_mode* to 0. This setting causes the OWM to transmit/receive in the default 8-bit mode.
2. Write *OWM_DATA.tx_rx* to 0x0FFh.
3. Once the 8-bit transmission completes, the hardware sets the interrupt flag *OWM_INTFL.tx_data_empty* to 1. This flag (that triggers an OWM module interrupt if *OWM_INTEN.tx_data_empty* is also set to 1) is cleared by writing a 1 to the flag.
4. As the hardware shifts the bit values out, it also samples the values returned from the slave device. Once the full 8-bit value is ready to be read, the interrupt flag *OWM_INTFL.rx_data_ready* is set to 1. If *OWM_INTEN.rx_ready* is set to 1, an OWM module interrupt occurs.
5. Read *OWM_DATA.tx_rx* to determine the 8-bit value returned by the slave device. Note that if no slave devices are present or the slave devices are not communicating with the master, the return value 0x0FF is the same as the transmitted value.

21.6 Registers

See *Table 3-3* for the the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

Table 21-4: OWM Register Summary

Offset	Register	Description
[0x0000]	*OWM_CFG*	OWM Configuration Register
[0x0004]	*OWM_CLK_DIV_1US*	OWM Clock Divisor Register
[0x0008]	*OWM_CTRL_STAT*	OWM Control/Status Register
[0x000C]	*OWM_DATA*	OWM Data Buffer Register
[0x0010]	*OWM_INTFL*	OWM Interrupt Flag Register
[0x0014]	*OWM_INTEN*	OWM Interrupt Enable Register

21.7 Register Details

Table 21-5: OWM Configuration Register

OWM Configuration Register		OWM_CFG		[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	0	**Reserved for Future Use** Do not modify this field.
7	int_pullup_enable	R/W	0	**Internal Pullup Enable** Set this field to enable the internal pullup resistor. 0: Internal pullup disabled. 1: Internal pullup enabled.
6	overdrive	R/W	0	**Overdrive Enable** Set this field to 1 to enable overdrive mode for 1-Wire communications. Clearing this field sets 1-Wire communications to standard speed. 0: Overdrive mode disabled, standard speed mode. 1: Overdrive mode enabled.
5	single_bit_mode	R/W	0	**Bit Mode Enable** When set to 1, only a single bit at a time is transmitted and received (LSB of *OWM_DATA*) rather than the whole byte. 0: Byte mode enabled, single bit mode disabled. 1: Single bit mode enabled, byte mode disabled.
4	ext_pullup_enable	R/W	0	**External Pullup Enable** Enables external FET pullup when the 1-Wire master is idle. FET is designed to pull the wire high regardless of its enable state (that is, high or low). Idle means the 1-Wire master is idle, and there are no 1-Wire accesses in progress. 0: External pullup pin is not driven to high. 1: External pullup pin is driven high when the 1-Wire bus is idle, actively pulling the 1-Wire IO high.
3	ext_pullup_mode	R/W	0	**External Pullup Mode** Provides an extra output to control an external pullup. For long wires, a pullup resistor strong enough to pull the wire high in a reasonable amount of time might need to be so strong that it would be difficult to drive the line low. In this case, implement an external FET to actively drive the wire high for a brief amount of time. Then, let the resistor keep the line high.

OWM Configuration Register		OWM_CFG		[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
2	bit_bang_en	R/W	0	**Bit-Bang Mode Enable** Enable bit-bang control of the I/O pin. If this bit is set to 1, *OWM_CTRL_STAT.bit_bang_oe* controls the state of the I/O pin. 0: Bit-bang mode disabled. 1: Bit-bang mode enabled.
1	force_pres_det	R/W	1	**Presence Detect Force** Setting this bit to 1 drives the OWM_IO pin low during presence detection. Use this bit field to prevent a large number of 1-Wire slaves on the bus from all responding at different times, which might cause ringing. When this bit is set to 1, the *OWM_CTRL_STAT.presence_detect* bit is always set as the result of a 1-Wire reset even if no slave devices are present on the bus. 0: OWM_IO pin floats during presence detection portion of 1-Wire reset. 1: OWM_IO pin is driven low during presence detection portion of 1-Wire reset.
0	long_line_mode	R/W	0	**Long Line Mode Enable** Selects alternate timings for 1-Wire communication. The recommended setting depends on the length of the wire. For lines less than 40 meters, 0 should be used. Setting this bit to 0 leaves the write one release, the data sampling, and the time-slot recovery times at approximately 5µs, 15µs, and 7µs, respectively. Setting this bit to 1 enables long line mode timings during standard mode communications. This mode moves the write one release, the data sampling, and the time-slot recovery times out to approximately 8µs, 22µs, and 14µs, respectively. 0: Standard operation for lines less than 40 meters. 1: Long line mode enabled, see description above.

Table 21-6: OWM Clock Divisor Register

OWM Clock Divisor Register		OWM_CLK_DIV_1US		[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R	0	**Reserved for Future Use** Do not modify this field.
7:0	divisor	R/W	0	**OWM Clock Divisor** Divisor for the OWM peripheral clock. The target is to achieve a 1MHz clock. See the *Clock Configuration* section for details. 0x00: OWM clock disabled. 0x01: $f_{owmclk} = \frac{f_{PCLK}}{1}$ 0x02: $f_{owmclk} = \frac{f_{PCLK}}{2}$ . . . 0xFF: $f_{owmclk} = \frac{f_{PCLK}}{255}$

Table 21-7: OWM Control/Status Register

OWM Control/Status Register		OWM_CTRL_STAT		[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved for Future Use** Do not modify this field.

OWM Control/Status Register			OWM_CTRL_STAT	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
7	presence_detect	RO	0	**Presence Detect Flag** Set to 1 when a presence pulse is detected from one or more slaves during the 1-Wire reset sequence. 0: No presence detect pulse during previous 1-Wire reset sequence. 1: Presence detect pulse on bus during previous 1-Wire reset sequence.
6:5	-	RO	0	**Reserved for Future Use** Do not modify this field.
4	od_spec_mode	RO	0	**Overdrive Spec Mode** Returns the version of the overdrive spec.
3	ow_input	RO	-	**OWM_IN State** Returns the current logic level on the OWM_IO pin. 0: OWM_IO pin is low. 1: OWM_IO pin is high.
2	bit_bang_oe	R/W	0	**OWM Bit-Bang Output** When bit-bang mode is enabled (*OWM_CFG.bit_bang_en* = 1), this bit sets the state of the OWM_IO pin. Setting this bit to 1 drives the OWM_IO pin low. Setting this bit to 0 releases the line, allowing the OWM_IO pin to be pulled high by the pullup resistor or held low by a slave device. 0: OWM_IO pin floating. 1: Drive OWM_IO pin to low state.
1	sra_mode	R/W	0	**Search ROM Accelerator Enable** Enable Search ROM Accelerator mode. This mode is used to identify slaves and their addresses that are attached to the 1-Wire bus. 0: Search ROM accelerator mode disabled. 1: Search ROM accelerator mode enabled.
0	start_ow_reset	R/W	0	**Start 1-Wire Reset Pulse** Write 1 to start a 1-Wire reset sequence. Automatically cleared by the OWM hardware when the reset sequence is complete. 0: 1-Wire reset sequence complete or inactive. 1: Start a 1-Wire reset sequence.

Table 21-8: OWM Data Register

OWM Data Register			OWM_DATA	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	0	**Reserved for Future Use** Do not modify this field.
7:0	tx_rx	R/W	0	**OWM Data Field** Writing to this field sets the transmit data and initiates a 1-Wire data transmit cycle. Reading from this field returns the data received by the master during the last 1-Wire data transmit cycle.

Table 21-9: OWM Interrupt Flag Register

OWM Interrupt Flag Register			OWM_INTFL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	R/W	0	**Reserved for Future Use** Do not modify this field.

OWM Interrupt Flag Register		OWM_INTFL		[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
4	line_low	R/W1C	0	**Line Low Flag** If this flag is set, the OWM_IO pin was in a low state. Write 1 to clear this flag.
3	line_short	R/W1C	0	**Line Short Flag** The OWM hardware detected a short on the OWM_IO pin. Write 1 to clear this flag.
2	rx_data_ready	R/W1C	0	**RX Data Ready** Data received from the 1-Wire bus and is available in the *OWM_DATA.tx_rx* field. Write 1 to clear this flag. 0: RX data not available. 1: Data received and is available in the *OWM_DATA.tx_rx* field.
1	tx_data_empty	R/W1C	0	**TX Empty** The OWM hardware automatically sets this interrupt flag when the data transmit is complete. Write 1 to clear this flag. 0: Either no data was sent or the data in the *OWM_DATA.tx_rx* field has not completed transmission. 1: Data in the *OWM_DATA.tx_rx* field was transmitted.
0	ow_reset_done	R/W1C	0	**Reset Complete** This flag is set when a 1-Wire reset sequence completes. To start a 1-Wire reset sequence, see *OWM_CTRL_STAT.start_ow_reset*. Write 1 to clear this flag. 0: 1-Wire reset sequence not complete or bus idle. 1: 1-Wire reset sequence complete.

Table 21-10: OWM Interrupt Enable Register

OWM Interrupt Enable Register		OWM_INTEN		[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved for Future Use** Do not modify this field.
4	line_low	R/W	0	**Line Low Interrupt Enable** I/O pin low detected interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
3	line_short	R/W	0	**Line Short Interrupt Enable** I/O pin short detected interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
2	rx_data_ready	R/W	0	**Receive Data Ready Interrupt Enable** RX data ready interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
1	tx_data_empty	R/W	0	**Transmit Data Empty Interrupt Enable** TX data empty interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
0	ow_reset_done	R/W	0	**1-Wire Reset Sequence Complete Interrupt Enable** 1-Wire reset sequence completed. 0: Interrupt disabled. 1: Interrupt enabled.

22. USB 2.0 High-Speed (USBHS) Host Interface with PHY

The microcontroller includes one Universal Serial Bus (USB) Host communications peripheral with a USB physical interface (PHY). The USB Host is USB 2.0 High-Speed (USBHS) compliant, capable of transfers at 480Mbps. It supports Host mode with 12 USB buffers called endpoints.

The following features are supported:

- USB Device Mode
- USB 2.0 Full Speed (FS) 12Mbps transfers
- USB 2.0 Hi-Speed (HS) 480Mbps transfers
- Bulk transfers
- Isochronous transfers
- 11 endpoints plus Endpoint 0, each with dedicated FIFOs
- Packet splitting and combining
- High bandwidth IN and OUT Isochronous endpoints

Each endpoint has an associated FIFO with the following sizes:

- Endpoint 0 FIFO: 64 bytes deep
- Endpoints 1 through 7 FIFOs: 512 bytes deep
- Endpoints 8 and 9 FIFOs: 2048 bytes deep
- Endpoints 10 and 11 FIFOs: 4096 bytes deep

Supported interrupts include:

- Interrupts for each IN endpoint from Endpoint 0 to Endpoint 11
- Interrupts for each OUT endpoint from Endpoint 1 to Endpoint 11
- Start of Frame (SOF)
- RESET bus state
- RESUME bus state
- SUSPEND Mode bus state
- STALL sent
- Control byte received
- Control transfer ended early
- Packet transmitted
- Packet received
- Data underrun
- Data overrun
- Invalid token received
- Empty data packet sent

This chapter includes a simplified description of USB bus states. Refer to the USB 2.0 Specification for a complete description of USB operation.

The USB device hardware behavior is controlled by the internal serial interface engine (SIE). The SIE is a small control processor that manages the USB port's behavior. When referring to behavior of the USB hardware, it is the SIE doing the work.

22.1 Instances

There is one instance of the peripheral. See *Table 22-1* for signal definition.

Table 22-1: USB Instance Table

Instance	USB Signal	Package Signal Name
USBHS	D+	DP
	D-	DM

Instance	USB Signal	Package Signal Name
	VBUS	V_{DDB}
	GND	V_{SSB}

22.2 USBHS Bus Signals

A USB cable connects a USB Host, which controls the transfer, and a USB Device, which is controlled by the Host. The USBHS Peripheral is a USB Device. A USB cable has four conductors (three hardware signals plus ground). These signals can be duplicated more than once in a physical USB cable. The signals in a USB cable are as follows:

- **D+ (DPLUS):** Positive line of the differential data pair.
- **D- (DMINUS):** Negative line of the differential data pair.
- **VBUS:** Bus voltage supplied by Host.
- **Ground**

When a USB Device is attached to a USB Host, the USB Host identifies the speed of the USB Device by the presence of a pullup resistor on either D+ or D-. The Host then begins the Enumeration sequence. The Enumeration sequence allows the Host to identify the type and characteristics of the Device attached to the Host so that it can load the proper drivers for the Device. Enumeration always uses Endpoint 0. The Host requests and reads from the Device the contents of the USB Endpoint Descriptor Table that tells the Host everything it needs to know about the capabilities of the USB Device. The Host then assigns the USB Device an address, which firmware writes to the *USBHS_FADDR.faddr* bit field.

Table 22-2 shows the USB Bus states seen by the Host indicated by the differential pair.

Table 22-2: USB Bus States Indicated by the Differential Pair (D+, D-)

Bus State	Condition	D+	D-	Notes
Differential 1	Host or Device is driving the bus	Hi	Lo	
Differential 0	Host or Device is driving the bus	Lo	Hi	
Single-Ended Zero (SE0)	Cable Detached	Lo	Lo	No Device plugged in.
Single-Ended One (SE1)	Illegal State	Hi	Hi	Illegal state. This state should never occur on a properly configured USB bus.
IDLE State, Full Speed	No Host or Device is driving the bus	Hi	Lo	USB Device is Full Speed. No activity on bus.
IDLE State, Low Speed	No Host or Device is driving the bus	Lo	Hi	USB Device is Low Speed. No activity on bus.
DISCONNECT	Device wants to disconnect from Host	Lo	Lo	Held for 2.5µs or longer.
RESET	Host is initiating communication with a Device	Lo	Lo	Held for 10ms or longer.

USB communication is based on the above basic conditions, which are used to generate the following states:

- **Data J State** – Same as IDLE state, but bus is actively driven by either the Host or the Device.
- **Data K State** – Opposite of J state. Bus is actively driven by the Host or the Device.
- **RESUME** – Data K State. Tells Device to exit SUSPEND mode.
- **START OF PACKET (SOP)** – Bus switches from IDLE to K state.
- **END OF PACKET (EOP)** – SE0 for two bit periods, then J state for one bit period.
- **KEEP ALIVE Signal** – EOP sent every 1 millisecond.

22.3 USBHS Device Endpoints

Each USB Device supports one or more endpoints. Endpoints serve as a source or destination for data and are supported by memory buffers. This USB controller supports 12 endpoints, each with its own set of descriptors and data buffers. These Endpoints are referenced as Endpoint 0 through Endpoint 11.

Endpoints support four different types of data transfers:

- **Control Endpoint** – Always uses Endpoint 0, this endpoint is used by the USB Host to setup the USB Device for the USB Device to receive operational status from the USB Host.
- **Interrupt Endpoints** – Used to send and receive non-time critical data to and from a USB Device. An application example is a USB keyboard or a USB mouse.
- **Bulk Endpoints** – Used to send and receive high-volume data that does not require real-time processing. An application example is a USB flash drive which transfers high volume data.
- **Isochronous Endpoints** – Used to send or receive real-time data that requires a guaranteed bandwidth to or from a Host. An application example is a video camera used for real-time video streaming.

The USBHS supports Control, Interrupt, Bulk, and Isochronous Endpoints. Per the USB 2.0 Specification, Endpoint 0 is dedicated to Control Transfers only.

Endpoint directions are always defined from a USB Host to a USB Device. OUT Endpoint 1 refers to a Device Endpoint holding data sent out from a USB Host to a USB Device. IN Endpoint 2 refers to a Device Endpoint holding data sent from a USB Device to a USB Host.

Each USBHS Data Endpoint supports the following features:

- Single or double buffered
- Programmable and flexible interrupts
- Ability to send a STALL packet to the Host to indicate an error with the data
- Ability to automatically send an ACK packet to the Host to acknowledge a successful data transfer
- Ability to send a NYET (Not Yet) packet to the Host for Hi-Speed transfers to indicate it is not yet ready to receive more data
- Configurable response to Status Stage of Control transfer

22.4 USBHS Reset and Clock

When a RESET state is detected on the bus, the USBHS performs the following actions:

1. Sets *USBHS_FADDR.addr* = 0
2. Sets *USBHS_INDEX* = 0
3. All endpoint FIFOs are flushed
4. All control and status registers are reset
5. The USB PHY is electrically disconnected from the bus
6. All endpoint interrupts are enabled
7. Generates a USB Reset IRQ

For correct operation of the USBHS interface, the system clock, f_{SYS_CLK}, must be no less than 32MHz.

CAUTION: When switching SYS_OSC or modifying the SYS_OSC prescaler (GCR_CLK_CTRL.sysclk_prescale), any device peripherals using SYS_CLK, APB clock, or AHB clock may become unstable. Software should disable the USB peripheral before switching SYS_OSC or modifying the SYS_OSC prescaler.

22.5 USBHS SUSPEND Mode and RESUME States

When the USBHS sees no activity on the bus for 3ms, and if SUSPEND mode is allowed (*USBHS_POWER.suspendm* = 1), then the USBHS goes into low-power SUSPEND mode. The SUSPEND status flag is set (*USBHS_INT.suspend* = 1), and a SUSPEND interrupt is generated if enabled (*USBHS_IN.suspend* = 1).

Firmware can exit SUSPEND mode by sending a RESUME state on the bus by setting the bit field *USBHS_POWER.resume* = 1. Firmware must leave this bit set between 2ms and 15ms with 10ms being the optimal time after which firmware must clear the resume bitfield.

If the external Host generates a RESUME state on the bus, a RESUME interrupt is generated. A RESUME interrupt is not generated if the RESUME state on the bus is caused by firmware setting the *USBHS_POWER.resume* bit.

22.6 Packet Size

For all transfers the packet size is specified in the *USBHS_INMAXP* register for IN endpoints and the *USBHS_OUTMAXP* register for OUT endpoints. These registers specify the size of the entire transactions.

22.7 Endpoint 0 Control Transactions

Endpoint 0 (EP0) is the main control endpoint and handles all USB Standard Device Requests for control transfers. There are three types of Standard Device Requests:

1. In Zero Data Requests, all the information for the request is included in an 8-byte command.
2. In Write Requests, the command from the USBHS is followed by additional data.
3. In Read Requests, the USBHS is communicating with a USB Device that is required to send data back to the Host.

22.7.1 Endpoint 0 Error Handling

The USBHS can detect and generate interrupts for control transfers errors. It sends a STALL packet on the bus and generates an interrupt if the incorrect amount of data is transferred over the bus. This can happen under the following conditions:

1. The Host sends more data during the OUT Data phase of a write request than the amount specified in the command. This condition is detected when the Host sends an OUT token after the Data End bit *USBHS_CSR0.dataend* is set by firmware.
2. The Host requests more data during the IN Data phase of a read request than the amount specified in the command. This condition is detected when the Host sends an IN token after the DataEnd bit *USBHS_CSR0.dataend* is set by firmware.
3. The Host sends more data in an OUT data packet than the amount specified in the *USBHS_OUTMAXP* register.
4. The Host sends a non-zero length DATA1 packet during the STATUS phase of a read request.

An error occurs if a control transaction ends prematurely. This can happen if the USB Host enters the STATUS phase before all data has been transferred. This can also occur if a USB Host transmits a new SETUP packet before finishing the current control transaction. In both cases, the *USBHS_CSR0.setupend* bit is set, which generates an Endpoint 0 interrupt.

If the *USBHS_CSR0.outpktrdy* bit is set, this indicates that the Host has sent another SETUP packet. Firmware should then process the command in that packet.

22.8 Bulk Endpoints Operation and Options

22.8.1 Bulk IN Endpoints

A Bulk IN endpoint is used to transfer high-volume data that does not require real-time processing. Five features are available for use with a Bulk IN endpoint as shown in *Table 22-3*.

Table 22-3: USB Bulk IN Endpoints Options

Bulk IN Endpoint Option	Description
Double Packet Buffering	When the value written to the *USBHS_INMAXP* register is less than or equal to half the size of the FIFO allocated to the endpoint, and double packet buffering is allowed (*USBHS_INCSRU.dpktbufdis* = 0), double packet buffering is enabled. This allows up to two packets to be stored in the FIFO for transmission to the Host.
AutoSet	When the AutoSet feature is enabled (*USBHS_INCSRU.autoset* = 1) for a Bulk IN endpoint, the IN Packet Ready bit field *USBHS_INCSRL.inpktrdy* is automatically set when a packet of *USBHS_INMAXP* bytes is loaded into the FIFO.
Automatic Packet Splitting	For some USB transfers, it might be necessary to write larger amounts of data to an endpoint than you can transfer in a single USB operation. For these transfers, the USBHS supports split transactions where large data packets that are written to Bulk endpoints are split into multiple smaller packets. The necessary packet size information is set in the *USBHS_INCSRU* register.
Error Handling	A STALL packet is used to indicate that an endpoint has had an error. To shut down the Bulk IN endpoint transfer, set the *USBHS_INCSRL.sendstall* bit field. When the USBHS receives the next IN token, it then sends a STALL to the Host, sets the *USBHS_INCSRL.sentstall* bit field, and generates an interrupt.

22.8.2 Bulk OUT Endpoints

A Bulk OUT endpoint is used to transfer non-periodic data from the Host to the function controller. Five optional features are available for use with a Bulk OUT endpoint.

Bulk OUT Endpoint Option	Description
Double Packet Buffering	When the value written to the *USBHS_OUTMAXP* register is less than or equal to half the size of the FIFO allocated to the endpoint, and double packet buffering is allowed (*USBHS_OUTCSRU.dpktbufdis* = 0), double packet buffering is enabled. This allows storage of up to two packets in the FIFO for transmission to the Host.
AutoClear	When the AutoClear feature is enabled (*USBHS_OUTCSRU.autoclear* = 1), the *USBHS_OUTCSRL.outpktrdy* bit is automatically cleared when a packet of *USBHS_OUTMAXP* bytes is unloaded from the FIFO.
Automatic Packet Combining	For some USB transfers, it might be necessary to receive larger amounts of data from an endpoint than can be received in a single USB operation. For these transfers, the USBHS supports automatically combining packets received by split transactions, where large data packets received by Bulk endpoints had been split into multiple smaller packets. The necessary packet size information is set in the *USBHS_OUTMAXP* register.
Error Handling	A STALL packet is used to indicate that an endpoint has an error. To shut down the Bulk OUT endpoint transfer, set *USBHS_OUTCSRL.sendstall* = 1. When the USBHS receives the next packet, it then sends a STALL to the Host, sets the *USBHS_OUTCSRL.sentstall* bit, and generates an interrupt.

22.9 Interrupt Endpoints

22.9.1 Interrupt IN Endpoints

Interrupt IN endpoints use the same protocols as Bulk IN endpoints and are used the same way.

One feature supported by Interrupt IN endpoints and not Bulk IN endpoints is continuous toggle of the Data-Toggle bit. This feature is enabled by setting bit *USBHS_INCSRU.frcdatatog* = 1. When continuous toggling of the Data-Toggle bit is enabled, USBHS always considers a transmitted Interrupt packet as successfully sent and toggles Data-Toggle regardless of whether an ACK was received from the Host.

22.9.2 Interrupt OUT Endpoints

Interrupt OUT endpoints use almost the same protocols as Bulk OUT endpoints and are used in the same way.

One feature not supported by Interrupt OUT endpoints that is supported by Bulk OUT endpoints is PING flow control. Because of this, Interrupt OUT endpoints cannot respond with NYET (Not Yet) handshakes. Instead, they can only respond with ACK, NAK, or STALL.

22.10 Isochronous Endpoints

22.10.1 Isochronous IN Endpoints

An Isochronous IN endpoint is used to transfer time-sensitive but loss-tolerant data from a USB Device to a USB Host. Five optional features are available for use with an Isochronous IN endpoint as shown in *Table 22-4*.

Table 22-4: USB Isochronous IN Endpoint Options

Isochronous IN Endpoint Option	Description
Double Packet Buffering	When the value written to the *USBHS_INMAXP* register is less than or equal to half the size of the FIFO allocated to the endpoint, and double packet buffering is allowed (*USBHS_INCSRU.dpktbufdis* = 0), double packet buffering is enabled. This allows the storage of up to two packets in the FIFO for transmission to the Host. This is recommended to avoid data underrun.
AutoSet	When the AutoSet feature is enabled (*USBHS_INCSRU.autoset* = 1), the *USBHS_INCSRL.inpktrdy* bit is automatically set when a packet of *USBHS_INMAXP* bytes is loaded into the FIFO. However, there is little benefit with Isochronous endpoints because the packets transferred are often not the maximum packet size. In this situation, the *USBHS_INCSRL.underrun* bit would have to be checked for underrun errors after each packet.
Error Handling	If an Isochronous IN endpoint receives an IN Token while the IN FIFO is empty, it creates an underrun condition. This automatically sets the *USBHS_INCSRL.underrun* bit and results in the USBHS sending a null packet to the USB Host. If firmware is loading the IN Endpoint FIFO one packet per frame, it should check that there is room in the IN FIFO by making sure the *USBHS_INCSRL.inpktrdy* bit is cleared before loading the next packet. If this bit is set, it indicates that a data packet is still in the FIFO and has not been sent, possibly from a corrupt IN Token. This error condition must be handled by firmware, for example, firmware might flush the unsent packet, or skip the current packet.
Error Handling – High Bandwidth Isochronous IN Endpoints Only	High-bandwidth Isochronous IN endpoints can transfer three 1024-byte packets in one payload. To the USB bus, it appears to be a single packet of 3072 bytes with a data transfer rate of up to 24MBps. If a high-bandwidth isochronous data transfer is split into more than one packet but has not received enough IN tokens from the Host to send all the packets, an error condition exists. In this case, the Incomplete Split Transfer Error Status bit *USBHS_INCSRL.incomPTn*, is automatically set. This also automatically flushes the remainder of the packet from the IN FIFO. If a second packet is in the IN FIFO, it is not flushed. Because the packet was lost, the *USBHS_INCSRL.inpktrdy* bit is cleared.

22.10.2 Isochronous OUT Endpoints

An Isochronous OUT endpoint is used to transfer time-sensitive but loss-tolerant data from the Host to the function controller. Five optional features are available for use with an Isochronous OUT endpoint as shown in *Table 22-5*.

Table 22-5: USB Isochronous OUT Endpoint Options

Isochronous OUT Endpoint Option	Description
Double Packet Buffering	When the value written to the *USBHS_OUTMAXP* register is less than or equal to half the size of the FIFO allocated to the endpoint, and double packet buffering is allowed (*USBHS_OUTCSRU.dpktbufdis* = 0), double packet buffering is enabled. This allows the storage of up to two packets in the FIFO for transmission to the Host. Double packet buffering is recommended for isochronous OUT endpoints to avoid data overrun errors.
AutoClear	When the AutoClear feature is enabled (*USBHS_OUTCSRU.autoclear* = 1), the *USBHS_OUTCSRL.outpktrdy* bit is automatically cleared when a packet of *USBHS_OUTMAXP* bytes is unloaded from the FIFO. However, there is little benefit with Isochronous endpoints because the packets transferred are often not the maximum packet size. In this situation, the *USBHS_INCSRL.underrun* bit would need to be checked for underrun errors after each packet.
Error Handling	If a packet is received from a USB Host, but the OUT FIFO is full, it creates an overrun error condition. The register bit *USBHS_OUTCSRL.overrun* is automatically set. This error condition usually means that firmware is not unloading the OUT FIFO fast enough. This error condition must be handled by firmware. If a received packet has a CRC error the packet is stored in the OUT FIFO, and both the *USBHS_OUTCSRL.dataerror* bit and the *USBHS_OUTCSRL.outpktrdy* bit are set. This error condition must be handled by firmware.
Error Handling – High Bandwidth Isochronous OUT Endpoints Only	High-bandwidth Isochronous OUT endpoints can transfer three 1024-byte packets in one payload. To the USB bus, it appears to be a single packet of 3072 bytes. If a high-bandwidth isochronous data transmission is split into more than one packet, but if less than the expected number of packets is received by the OUT endpoint, an error condition exists. In this case, the Incomplete Isochronous Packet Received Error Status bit *USBHS_OUTCSRU.incomprx* is automatically set to indicate that the data received in the OUT FIFO is incomplete. If a packet of the wrong data type is received during a high-bandwidth Isochronous OUT transaction, then the PID Error Status bit *USBHS_OUTCSRU.piderror* is automatically set.

22.11 USBHS Device Registers

See *Table 3-3*: APB Peripheral Base Address Map for the USBHS Peripheral Base Address.

Table 22-6: USBHS Device Register Offsets, Names, Access, and Descriptions

Offset	Register Name	Description
[0x0000]	*USBHS_FADDR*	USBHS Device Address Register
[0x0001]	*USBHS_POWER*	USBHS Power Management Register
[0x0002]	*USBHS_INTRIN*	USBHS IN Endpoint Interrupt Status Register
[0x0004]	*USBHS_INTROUT*	USBHS OUT Endpoint Interrupt Status Flags Register
[0x0006]	*USBHS_INTRINEN*	USBHS IN Endpoint Interrupt Enable Register
[0x0008]	*USBHS_INTROUTEN*	USBHS OUT Endpoint Interrupt Enable Register
[0x000A]	*USBHS_INTRUSB*	USBHS Signaling Interrupt Status Flags Register
[0x000B]	*USBHS_INTRUSBEN*	USBHS Signaling Interrupt Enable Register
[0x000C]	*USBHS_FRAME*	USBHS Frame Number Register
[0x000E]	*USBHS_INDEX*	USBHS Endpoint and Status Register Index Register
[0x000F]	*USBHS_TESTMODE*	USBHS Test Mode Register
[0x0010]	*USBHS_INMAXP*	USBHS IN Endpoint Maximum Packet Size Register
[0x0012]	*USBHS_CSR0*	USBHS Endpoint 0 Control Status Register
[0x0012]	*USBHS_INCSRL*	USBHS IN Endpoint Lower Control and Status Register

Offset	Register Name	Description
[0x0013]	*USBHS_INCSRU*	USBHS IN Endpoint Upper Control and Status Register
[0x0014]	*USBHS_OUTMAXP*	USBHS OUT Endpoint Maximum Packet Sizes Register
[0x0016]	*USBHS_OUTCSRL*	USBHS OUT Endpoint Lower Control Status Register
[0x0017]	*USBHS_OUTCSRU*	USBHS OUT Endpoint Upper Control Status Register
[0x0018]	*USBHS_COUNT0*	USBHS Endpoint 0 IN FIFO Byte Count Register
[0x0018]	*USBHS_OUTCOUNT*	USBHS Endpoint OUT FIFO Byte Count Register
[0x0020]	*USBHS_FIFO0*	USBHS FIFO for Endpoint 0 Register
[0x0024]	*USBHS_FIFO1*	USBHS FIFO for Endpoint 1 Register
[0x0028]	*USBHS_FIFO2*	USBHS FIFO for Endpoint 2 Register
[0x002C]	*USBHS_FIFO3*	USBHS FIFO for Endpoint 3 Register
[0x0030]	*USBHS_FIFO4*	USB HS FIFO for Endpoint 4 Register
[0x0034]	*USBHS_FIFO5*	USBHS FIFO for Endpoint 5 Register
[0x0038]	*USBHS_FIFO6*	USBHS FIFO for Endpoint 6 Register
[0x003C]	*USBHS_FIFO7*	USBHS FIFO for Endpoint 7 Register
[0x0040]	*USBHS_FIFO8*	USBHS FIFO for Endpoint 8 Register
[0x0044]	*USBHS_FIFO9*	USBHS FIFO for Endpoint 9 Register
[0x0048]	*USBHS_FIFO10*	USBHS FIFO for Endpoint 10 Register
[0x004C]	*USBHS_FIFO11*	USBHS FIFO for Endpoint 11 Register
[0x0078]	*USBHS_EPINFO*	USBHS Endpoint Count Info Register
[0x0079]	*USBHS_RAMINFO*	USBHS RAM and MAInfo Register
[0x007A]	*USBHS_SOFTRESET*	USBHS Soft Reset Control Register
[0x0080]	*USBHS_CTUCH*	USBHS Hi-Speed Chirp Timeout Register
[0x0082]	*USBHS_CTHSRTN*	USBHS Hi-Speed RESUME Delay Register
[0x0410	*USBHS_M31_PHY_PONRST*	USBHS Power-On Reset Register
[0x0414]	*USBHS_M31_PHY_NONCRY_RSTB*	USBHS Hi-Speed VBUS Reset Register
[0x0418]	*USBHS_M31_PHY_NONCRY_EN*	USBHS Non-Clock Recovery Enable
[0x0430]	*USBHS_M31_PHY_PLL_EN*	USBHS PLL Enable Register
[0x043C]	*USBHS_M32_PHY_OSCOUTEN*	USBHS Oscillator Output Enable Register
[0x0448]	*USBHS_M32_PHY_CORECLKIN*	USBHS Hi-Speed Core Clock Input Register
[0x044C]	*USBHS_M32_PHY_XTLSEL*	USBHS Hi-Speed Clock Source Frequency Select Register
[0x045C]	*USBHS_M32_PHY_OUTCLKSEL*	USBHS Hi-Speed Reference Clock Select Register
[0x0498]	*USBHS_MXM_INT*	USBHS Hi-Speed VBUS Interrupt Register
[0x049C]	*USBHS_MXM_INT_EN*	USBHS Hi-Speed VBUS Interrupt Enable Register
[0x04A0]	*USBHS_MXM_SUSPEND*	USBHS Suspend Register
[0x04A4]	*USBHS_MXM_REG_A4*	USBHS Hi-Speed VBUS State Register

22.12 USBHS Device Register Details

Table 22-7: USBHS Device Address Register

USBHS Device Address Register		USBHS_FADDR		[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
7	update	RO	0	**Read USBHS Device Update Status** 0: The Device address in the bit field addr is presently used. 1: New address written to the bit field addr is pending. New address takes effect at the end of the current transfer.
6:0	faddr	R/W	0	**USBHS Device Address** This is the USB Device address specified by the external USB Host during the enumeration process. It must be written with the address value contained in the SET_ADDRESS Device request when received during a Control Transaction.

Table 22-8: USBHS Power Management Register

USBHS Power Management		USBHS_POWER		[0x0001]
Bits	**Name**	**Access**	**Reset**	**Description**
7	iso_update	R/W	0	**Isochronous Update** 1: If an SOF token is received from the Host and a packet is in the IN FIFO (*USBHS_INCSRL.inpktrdy* = 1), then send the packet. However, if an IN token is received from the Host before an SOF token, then send a zero-length data packet. *Note: This register is only applicable in Isochronous Mode and ignored in all other modes.*
6	softconn	R/W	0	**Soft Connect/Disconnect PHY** 0: The USB D+/D- lines of the PHY are tri-stated, and this USB is electrically disconnected from the USB bus. 1: The USB D+/D- lines of the PHY are enabled.
5	hs_enable	R/W	1	**Enable Hi-Speed (HS) Mode** 0: USB remains in Full Speed Mode even if connected to a USB HS port. 1: USB always negotiates for HS mode on the bus.
4	hs_mode	RO	0	**Read Hi-Speed Mode Status Flag** 0: USB in Full Speed Mode. 1: USB in Hi-Speed Mode.
3	power_reset	RO	0	**Read RESET Mode Status Flag** 0: Normal operation. 1: RESET state is on the bus.
2	resume	R/W	0	**Generate RESUME State** Set to generate a RESUME state on the bus. Once set, it should be left set for at least 10ms and no more than 15ms, then cleared.
1	suspend	RO	0	**Read SUSPEND Mode Status** 0: Normal operation. 1: USBHS is in SUSPEND Mode. *Note: Automatically cleared when a SUSPEND Mode interrupt occurs, or if the resume bit (above) is set to 1.*

USBHS Power Management			USBHS_POWER	[0x0001]
Bits	**Name**	**Access**	**Reset**	**Description**
0	suspendm	R/W	0	**SUSPEND Mode Enable** 0: SUSPEND Mode disabled. USB will not enter SUSPEND Mode. 1: SUSPEND Mode allowed. If no activity is detected on the bus for more than 3.0ms, this USB enters low-power SUSPEND Mode.

Table 22-9: USBHS IN Endpoint Interrupt Flags Register

USBHS IN Endpoint Interrupt Flags			USBHS_INTRIN	[0x0002]
Bits	**Name**	**Access**	**Reset**	**Description**
16:12	-	ROC	0	**Reserved for Future Use** Do not modify this field.
11	ep11_in	ROC	0	**IN EP11 Interrupt Status Flag** 0: IN Endpoint 11 not active. 1: IN Endpoint 11 occurred.
10	ep10_in	ROC	0	**IN EP10 Interrupt Status Flag** 0: IN Endpoint 10 not active. 1: IN Endpoint 11 occurred.
9	ep9_in	ROC	0	**IN EP9 Interrupt Status Flag** 0: IN Endpoint 9 not active. 1: IN Endpoint 9 occurred.
8	ep8_in	ROC	0	**IN EP8 Interrupt Status Flag** 0: IN Endpoint 8 not active. 1: IN Endpoint 8 occurred.
7	ep7_in	ROC	0	**IN EP7 Interrupt Status Flag** 0: IN Endpoint 7 not active. 1: IN Endpoint 7 occurred.
6	ep6_in	ROC	0	**IN EP6 Interrupt Status Flag** 0: IN Endpoint 6 not active. 1: IN Endpoint 6 occurred.
5	ep5_in	ROC	0	**IN EP5 Interrupt Status Flag** 0: IN Endpoint 5 not active. 1: IN Endpoint 5 occurred.
4	ep4_in	ROC	0	**IN EP4 Interrupt Status Flag** 0: IN Endpoint 4 not active. 1: IN Endpoint 4 occurred.
3	ep3_in	ROC	0	**IN EP3 Interrupt Status Flag** 0: IN Endpoint 3 not active. 1: IN Endpoint 3 occurred.
2	ep2_in	ROC	0	**IN EP2 Interrupt Status Flag** 0: IN Endpoint 2 not active. 1: IN Endpoint 2 occurred.
1	ep1_in	ROC	0	**IN EP1 Interrupt Status Flag** 0: IN Endpoint 1 not active. 1: IN Endpoint 1 occurred.

USBHS IN Endpoint Interrupt Flags		USBHS_INTRIN		[0x0002]
Bits	**Name**	**Access**	**Reset**	**Description**
0	ep0	ROC	0	**EP0 Interrupt Status Flag** 0: In Endpoint 0 not active. 1: In Endpoint 0 occurred.

Table 22-10: USBHS OUT Endpoint Interrupt Flags Register

USBHS OUT Endpoint Interrupt Flags		USBHS_INTROUT		[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
16:12	-	ROC	0	**Reserved for Future Use** Do not modify this field.
11	ep11_out	ROC	0	**OUT EP11 Interrupt Status Flag** 0: OUT Endpoint 11 interrupt not active. 1: OUT Endpoint 1 1interrupt active.
10	ep10_out	ROC	0	**OUT EP10 Interrupt Status Flag** 0: OUT Endpoint 10 interrupt not active. 1: OUT Endpoint 10 interrupt active.
9	ep9_out	ROC	0	**OUT EP9 Interrupt Status Flag** 0: OUT Endpoint 9 interrupt not active. 1: OUT Endpoint 9 interrupt active.
8	ep8_out	ROC	0	**OUT EP8 Interrupt Status Flag** 0: OUT Endpoint 8 interrupt not active. 1: OUT Endpoint 8 interrupt active.
7	ep7_out	ROC	0	**OUT EP7 Interrupt Status Flag** 0: OUT Endpoint 7 interrupt not active. 1: OUT Endpoint 7 interrupt active.
6	ep6_out	ROC	0	**OUT EP6 Interrupt Status Flag** 0: OUT Endpoint 6 interrupt not active. 1: OUT Endpoint 6 interrupt active.
5	ep5_out	ROC	0	**OUT EP5 Interrupt Status Flag** 0: OUT Endpoint 5 interrupt not active. 1: OUT Endpoint 5 interrupt active.
4	ep4_out	ROC	0	**OUT EP4 Interrupt Status Flag** 0: OUT Endpoint 4 interrupt not active. 1: OUT Endpoint 4 interrupt active.
3	ep3_out	ROC	0	**OUT EP3 Interrupt Status Flag** 0: OUT Endpoint 3 interrupt not active. 1: OUT Endpoint 3 interrupt active.
2	ep2_out	ROC	0	**OUT EP2 Interrupt Status Flag** 0: OUT Endpoint 2 interrupt not active. 1: OUT Endpoint 2 interrupt active.
1	ep1_out	ROC	0	**OUT EP1 Interrupt Status Flag** 0: OUT Endpoint 1 interrupt not active. 1: Interrupt occurred.
0	-	ROC	0	**Reserved for Future Use** Do not modify this field.

Table 22-11: USBHS IN Endpoint Interrupt Enable Register

USBHS IN Endpoint Interrupt Enable			USBHS_INTRINEN	[0x0006]
Bits	**Name**	**Access**	**Reset**	**Description**
16:12	-	R/W	0	**Reserved for Future Use** Do not modify this field.
11	ep11_in	R/W	0	**IN EP11 Interrupt Enable** Set to 1 to enable the interrupt for IN Endpoint 11. 0: Interrupt disabled. 1: Interrupt enabled.
10	ep10_in	R/W	0	**IN EP10 Interrupt Enable** Set to 1 to enable the interrupt IN Endpoint 10. 0: Interrupt disabled. 1: Interrupt enabled.
9	ep9_in	R/W	0	**IN EP9 Interrupt Enable** Set to 1 to enable the interrupt IN Endpoint 9. 0: Interrupt disabled. 1: Interrupt enabled.
8	ep8_in	R/W	0	**IN EP8 Interrupt Enable** Set to 1 to enable the interrupt IN Endpoint 8. 0: Interrupt disabled. 1: Interrupt enabled.
7	ep7_in	R/W	0	**IN EP7 Interrupt Enable** Set to 1 to enable the interrupt IN Endpoint 7. 0: Interrupt disabled. 1: Interrupt enabled.
6	ep6_in	R/W	0	**IN EP6 Interrupt Enable** Set to 1 to enable the interrupt IN Endpoint 6. 0: Interrupt disabled. 1: Interrupt enabled.
5	ep5_in	R/W	0	**IN EP5 Interrupt Enable** Set to 1 to enable the interrupt IN Endpoint 5. 0: Interrupt disabled. 1: Interrupt enabled.
4	ep4_in	R/W	0	**IN EP4 Interrupt Enable** Set to 1 to enable the interrupt IN Endpoint 4. 0: Interrupt disabled. 1: Interrupt enabled.
3	ep3_in	R/W	0	**IN EP3 Interrupt Enable** Set to 1 to enable the interrupt IN Endpoint 3. 0: Interrupt disabled. 1: Interrupt enabled.
2	ep2_in	R/W	0	**IN EP2 Interrupt Enable** Set to 1 to enable the interrupt IN Endpoint 2. 0: Interrupt disabled. 1: Interrupt enabled.

USBHS IN Endpoint Interrupt Enable			USBHS_INTRINEN	[0x0006]
Bits	**Name**	**Access**	**Reset**	**Description**
1	ep1_in	R/W	0	**IN EP1 Interrupt Enable** Set to 1 to enable the interrupt IN Endpoint 1. 0: Interrupt disabled. 1: Interrupt enabled.
0	ep0	R/W	0	**EP0 Interrupt Enable** Set to 1 to enable the interrupt IN Endpoint 0. 0: Interrupt disabled. 1: Interrupt enabled.

Table 22-12: USBHS OUT Endpoint Interrupt Enable Register

USBHS OUT Endpoint Interrupt Enable			USBHS_INTROUTEN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
16:12	-	R/W	0	**Reserved for Future Use** Do not modify this field.
11	ep11_out	R/W	1	**OUT EP11 Interrupt Enable** Set to 1 to enable the interrupt for OUT Endpoint 11. 0: Interrupt disabled. 1: Interrupt enabled.
10	ep10_out	R/W	1	**OUT EP10 Interrupt Enable** Set to 1 to enable the interrupt for OUT Endpoint 10. 0: Interrupt disabled. 1: Interrupt enabled.
9	ep9_out	R/W	1	**OUT EP9 Interrupt Enable** Set to 1 to enable the interrupt for OUT Endpoint 9. 0: Interrupt disabled. 1: Interrupt enabled.
8	ep8_out	R/W	1	**OUT EP8 Interrupt Enable** Set to 1 to enable the interrupt for OUT Endpoint 8. 0: Interrupt disabled. 1: Interrupt enabled.
7	ep7_out	R/W	1	**OUT EP7 Interrupt Enable** Set to 1 to enable the interrupt for OUT Endpoint 7. 0: Interrupt disabled. 1: Interrupt enabled.
6	ep6_out	R/W	1	**OUT EP6 Interrupt Enable** Set to 1 to enable the interrupt for OUT Endpoint 6. 0: Interrupt disabled. 1: Interrupt enabled.
5	ep5_out	R/W	1	**OUT EP5 Interrupt Enable** Set to 1 to enable the interrupt for OUT Endpoint 5. 0: Interrupt disabled. 1: Interrupt enabled.

USBHS OUT Endpoint Interrupt Enable			USBHS_INTROUTEN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
4	ep4_out	R/W	1	**OUT EP4 Interrupt Enable** Set to 1 to enable the interrupt for OUT Endpoint 4. 0: Interrupt disabled. 1: Interrupt enabled.
3	ep3_out	R/W	1	**OUT EP3 Interrupt Enable** Set to 1 to enable the interrupt for OUT Endpoint 3. 0: Interrupt disabled. 1: Interrupt enabled.
2	ep2_out	R/W	1	**OUT EP2 Interrupt Enable** Set to 1 to enable the interrupt for OUT Endpoint 2. 0: Interrupt disabled. 1: Interrupt enabled.
1	ep1_out	R/W	1	**OUT EP1 Interrupt Enable** Set to 1 to enable the interrupt for OUT Endpoint 1. 0: Interrupt disabled. 1: Interrupt enabled.
0	-	R	0	**Reserved for Future Use** Do not modify this field.

Table 22-13: USBHS Signaling Interrupt Status Flag Register

USBHS Signaling Interrupt Status Flags			USBHS_INTRUSB	[0x000A]
Bits	**Name**	**Access**	**Reset**	**Description**
8:4	-	R/W	0	**Reserved for Future Use** Do not modify this field.
3	sof	R	0	**Start Of Frame Detected Status Flag**
2	reset	R	0	**RESET State Detected Status Flag**
1	resume	R	0	**RESUME State Detected Status Flag** Set when a RESUME state is detected while in SUSPEND Mode.
0	suspend	R	0	**SUSPEND Mode Status Flag** Reads 1 when SUSPEND mode is active.

Table 22-14: USBHS Signaling Interrupt Enable Register

USBHS Signaling Interrupt Enable			USBHS_INTRUSBEN	[0x000B]
Bits	**Name**	**Access**	**Reset**	**Description**
8:4	-	R/W	0	**Reserved for Future Use** Do not modify this field.
3	sof	R/W	0	**Start Of Frame (SOF) Detected Interrupt Enable** 0: Interrupt event disabled. 1: Interrupt event enabled.
2	reset	R/W	1	**RESET State Detected Interrupt Enable** 0: Interrupt event disabled. 1: Interrupt event enabled.

USBHS Signaling Interrupt Enable			USBHS_INTRUSBEN	[0x000B]
Bits	**Name**	**Access**	**Reset**	**Description**
1	resume	R/W	1	**RESUME State Detected Interrupt Enable** 0: Interrupt event disabled. 1: Interrupt event enabled.
0	suspend	R/W	0	**SUSPEND Mode Interrupt Enable** 0: Interrupt event disabled. 1: Interrupt event enabled.

Table 22-15: USBHS Frame Number Register

USBHS Frame Number			USBHS_FRAME	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
15:11	framenum	R	0	**Frame Number** Always reads 0.
10:0	framenum	R	0	**Read Last Received Frame Number** This is the 11-bit frame number received in the SOF packet.

Table 22-16: USBHS Register Index Select Register

USBHS Register Index Select			USBHS_INDEX	[0x000E]
Bits	**Name**	**Access**	**Reset**	**Description**
7:5	-	R/W	0	**Reserved for Future Use** Do not modify this field.
4:0	index	R/W	0x0	**Index Register Access Selector** Each IN and OUT endpoint has memory-mapped control and status registers in addresses from 0x400B 1010 to 0x400B 1018. Only one endpoint's registers are addressable in the memory map at time. This bit field selects which endpoint's registers are present in the memory map where: 0x0: Endpoint 0 IN/OUT status registers addressable. 0x1: Endpoint 1 IN/OUT status registers addressable. 0x2: Endpoint 2 IN/OUT status registers addressable. … 0xB: Endpoint 11 IN/OUT status registers addressable.

Table 22-17: USBHS Test Mode Register

USBHS Test Mode Register			USBHS_TESTMODE	[0x000F]
Bits	**Name**	**Access**	**Reset**	**Description**
8:6	-	R/W	0	**Reserved for Future Use** Do not modify this field.
5	force_fs	R/W	0	**Force FS Mode** When the USBHS receives a RESET from the host, the USBHS is forced into Full-Speed mode.
4	force_hs	R/W	0	**Force HS Mode** When the USBHS receives a RESET from the host, the USBHS is forced into Hi-Speed mode.

USBHS Test Mode Register			USBHS_TESTMODE	[0x000F]
Bits	**Name**	**Access**	**Reset**	**Description**
3	test_packet	R/W	0	**Test Packet Mode** To enter this test mode, firmware must write the standard 53-byte test packet to the Endpoint 0 FIFO, then set *USBHS_INCSRL.inpktrdy* = 1, then set this bit. The DATA0 PID is automatically added to the head of the packet and the CRC to the end of the packet. The USBHS will continue to send the test packet until this bit is cleared.
2	test_k	R/W	0	**HS Data K State Test Mode** The USBHS transmits a continuous Data K State
1	test_j	R/W	0	**HS Data J State Test Mode** The USBHS transmits a continuous Data J State
0	test_se0_nak	R/W	0	**SE0 NAK Test Mode** The USBHS responds to any valid IN token with a NAK

22.12.2 Endpoint Register Access Control

Each IN and OUT endpoint from Endpoint 0x1 to 0xB uses memory mapped access to the registers in *Table 22-18*. Selecting a specific endpoint, using the *USBHS_INDEX* register, maps each of the registers in *Table 22-18* to the selected endpoint.

Table 22-18: USB Memory Mapped Register Access for Endpoints 1 to 11

Endpoint Register
USBHS_INMAXP
USBHS_INCSRL
USBHS_INCSRU
USBHS_OUTMAXP
USBHS_OUTCSRL
USBHS_OUTCSRU
USBHS_OUTCOUNT

22.12.3 USBHS IN Endpoint Maximum Packet Size Registers

Endpoints 1 to 11 have a memory mapped version of this register selected using the *USBHS_INDEX* register.

Table 22-19: USBHS IN Endpoint Maximum Packet Size Register

USBHS IN Endpoint Maximum Packet Size			USBHS_INMAXP	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
15:11	numpackminus1	R/W	0	**Number of Split Packets - 1** Defines the maximum number of packets minus 1 that a USB payload can be split into. This must be an exact multiple of *maxpacketsize*. The total number of bytes transferred in the payload is: $N_{BYTES\_TRANSFERED} = maxpacketsize \times (numpackminus1 + 1)$ This must match the *maxpacketsize f*ield of the Standard Endpoint Descriptor for the associated endpoint. For HS High Bandwidth Isochronous endpoints, the multiplier can only be 2 or 3, so this field can only be 0x01 or 0x02. For Bulk endpoints, the max multiplier is 32, so the maximum value for this register is 31 (0x1F). *Note: Only applicable for High-Speed (HS), High-Bandwidth Isochronous endpoints, and Bulk endpoints. Ignored in all other cases.*
10:0	maxpacketsize	R/W	0	**Maximum Packet Size in a Single Transaction** This is the maximum packet size, in bytes, that is transmitted for each microframe. The maximum value is 1024, subject to the limitations for the endpoint type set in the *USB 2.0 Specification, Chapter 9.* For Bulk Transfers: The USB 2.0 Specification requires this to be 8, 16, 32, or 64. HS Bulk Transfer also supports 512.

22.12.4 USBHS IN Endpoint Lower Control and Status Registers

Endpoints 1 to 11 have a memory mapped version of this register selected using the *USBHS_INDEX* register.

Table 22-20: USBHS IN Endpoint Lower Control and Status Register

USBHS IN Endpoint Lower Control and Status			USBHS_INCSRL	[0x0012]
Bits	**Name**	**Access**	**Reset**	**Description**
7	incompt	R/W0C	0	**Read Incomplete Split Transfer Error Status** High-Bandwidth Isochronous transfers: Automatically set when a payload is split into multiple packets but insufficient IN tokens were received to send all packets. The current packet is flushed from the IN FIFO. Write a 0 to clear. Bulk and Interrupt Transfers: Ignored.
6	clrdatatog	R/W1O	0	**Clear IN Endpoint Data Toggle** 1: Clear the IN Endpoint data toggle to 0. *Note: Automatically cleared after set.*
5	sentstall	R/W0C	0	**Read STALL Handshake Sent Status** Automatically set when a STALL handshake is transmitted, at which time the IN FIFO is flushed, and *USBHS_INCSRL.inpktrdy* is cleared. *Note: Write a 0 to clear.*

USBHS IN Endpoint Lower Control and Status				USBHS_INCSRL	[0x0012]
Bits	**Name**	**Access**	**Reset**	**Description**	
4	sendstall	R/W	0	**Send STALL Handshake** 1: Respond to an IN token with a STALL handshake. 0: Terminate STALL handshake *Note: Ignored for Isochronous transfers.*	
3	flushfifo	R/W1O	0	**Flush Next Packet from IN FIFO** 1: Flush the next packet to be transmitted from the IN FIFO. This also clears the bit field *USBHS_INCSRL.inpktrdy*. This must only be set when *USBHS_INCSRL.inpktrdy* = 1, or FIFO data corruption might occur. *Note: If the IN FIFO contains two packets, set the flushfifo field twice to clear both packets.* *Note: Automatically cleared when the packet is flushed.*	
2	underrun	R/W0C	0	**Read IN FIFO Underrun Error Status** Isochronous Mode: Automatically set if the IN FIFO is empty (*inpktrdy* = 0), an IN token has been received, and a zero-length data packet has been sent. Bulk or Interrupt Modes: Automatically set when an IN token is received, and a NAK is sent. *Note: Write a 0 to clear.*	
1	fifonotempty	R/W0C	0	**Read FIFO Not Empty Status** Automatically set when there is at least one packet in the IN FIFO. Note: Write a 0 to clear.	
0	inpktrdy	R/W1O	0	**IN Packet Ready** 1: Write a 1 after writing a data packet to the IN FIFO. Automatically cleared when the data packet is transmitted. If double-buffering is enabled, this bit is automatically cleared when there is space for a second packet in the FIFO. *Note: This bit field is also controlled by USBHS_INCSRU.autoset.*	

22.12.5 USBHS Endpoint 0 Control Status Register

Table 22-21: USBHS Endpoint 0 Control Status Register

USBHS Endpoint 0 Control Status				USBHS_CSR0	[0x0012]
Bits	**Name**	**Access**	**Reset**	**Description**	
7	servicedsetupend	R/W1C	0	**Clear EP0 Setup End Bit** Write a 1 to clear the *setupend* bit. *Note: Automatically cleared after being set.*	
6	servicedoutpktrdy	R/W1C	0	**Clear EP0 Out Packet Ready Bit** Write a 1 to clear the *outpktrdy* bit (below). *Note: Automatically cleared after being set.*	

USBHS Endpoint 0 Control Status			USBHS_CSR0	[0x0012]
Bits	**Name**	**Access**	**Reset**	**Description**
5	sendstall	R/W1O	0	**Send EP0 STALL Handshake** Write a 1 to this bit to terminate the current Control Transaction by sending a STALL handshake. Automatically cleared after being set. *Note: This behavior is different from the sendstall bits associated with IN/OUT endpoints.*
4	setupend	RO	0	**Read Setup End Status** Automatically set when a Control Transaction ends before the *dataend* bit has been set by software. An interrupt is generated when this bit is set. Write a 1 to *servicedsetupend* (above) to clear.
3	dataend	R/W1O	0	**Control Transaction Data End** Write a 1 to this bit after software completes any of the following three transactions: 1) Set *inpktrdy* = 1 for the last data packet. 2) Set *inpktrdy* = 1 for a zero-length data packet. 3) Clear *outpktrdy* = 0 after unloading the last data packet. *Note: Automatically cleared after being set.*
2	sentstall	R/W0C	0	**Read EP0 STALL Handshake Sent Status** Automatically set when a STALL handshake is transmitted. Write a 0 to clear.
1	inpktrdy	R/W1O	0	**EP0 IN Packet Ready** Set this bit to indicate a packet is ready to transmit from the IN FIFO. Hardware automatically clears this bit when the packet transmit is complete. 0: Packet was transmitted or no packet transmit pending. Read only. 1: Write a 1 after writing a data packet to the IN FIFO to indicate the EP0 IN packet is ready. *Note: An interrupt is generated when this bit is cleared.*
0	outpktrdy	RO	0	**EP0 OUT Packet Ready Status** Automatically set when a data packet is received in the OUT FIFO. An interrupt is generated when this bit is set. Write a 1 to the *servicedoutpktrdy* bit (above) to clear after the packet is unloaded from the OUT FIFO.

22.12.6 USBHS IN Endpoint Upper Control Registers

Endpoint 1 to 11 have a memory mapped version of this register selected using the *USBHS_INDEX* register.

Table 22-22: USBHS IN Endpoint Upper Control Register

USBHS IN Endpoint Upper Control		USBHS_INCSRU		[0x0013]
Bits	**Name**	**Access**	**Reset**	**Description**
7	autoset	R/W	0	**Auto Set inpktrdy** 0: *USBHS_INCSRL.inpktrdy* must be set by firmware 1: *USBHS_INCSRL.inpktrdy* is automatically set when data that is of the maximum packet size specified in the *USBHS_INMAXP* register is loaded into the IN FIFO.
6	iso	R/W	0	**Isochronous Transfer Enable** 0: Enable IN Bulk and IN Interrupt transfers 1: Enable IN Isochronous transfers
5	mode	R/W	0	**Endpoint Direction Mode** 0: Endpoint direction is OUT 1: Endpoint direction is IN *Note: Ignored if endpoint is not used for both IN and OUT transactions.*
4	-	-	0	**Reserved**
3	frcdatatog	R/W	0	**Force IN Data-Toggle** 0: Toggle data-toggle only when an ACK is received 1: Toggle data-toggle regardless of whether an ACK is received *Note: Useful for Interrupt IN endpoints that are communicating rate feedback to Isochronous endpoints.*
2	-	-	0	**Reserved**
1	dpktbufdis	R/W	0	**Double Packet Buffering Disable** 0: Enable double packet buffering. Firmware must also configure the FIFO and packet size. 1: Disable double packet buffering
0	-	R/W	0	**Reserved for Future Use** Do not modify this field.

Table 22-23: USBHS OUT Endpoint Maximum Packet Size Register

USBHS OUT Endpoint Maximum Packet Size			USBHS_OUTMAXP	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
15:11	numpackminus1	R/W	0x00	**Number of Split Packets** Defines the maximum number of packets minus 1 that a USB payload is combined into. The value must be an exact multiple of *maxpacketsize*. The total number of bytes transferred in the payload is *maxpacketsize* x (numpackminus1+1). This must match the *maxpacketsize* field of the Standard Endpoint Descriptor for the associated endpoint. Only applicable for Hi-Speed (HS), High Bandwidth Isochronous endpoints, and Bulk endpoints. Ignored in all other cases. **HS High Bandwidth Isochronous Endpoints** The multiplier can only be 2 or 3, so this bit field value can only be 0x01 or 0x02. **Bulk Endpoints** The max multiplier is 32, so the maximum value for this register is 31 (0x1F).

USBHS OUT Endpoint Maximum Packet Size			USBHS_OUTMAXP	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
10:0	maxpacketsize	R/W	0x000	**Maximum Packet Size in a Single Transaction** This is the maximum packet size, in bytes, that is transmitted for each microframe. The maximum value is 1024, subject to the limitations for the endpoint type set in the USB 2.0 Specification, Chapter 9. For all Bulk Transfers, the USB 2.0 Specification requires this to be 8, 16, 32, or 64. HS Bulk Transfer also supports 512.

Table 22-24: USBHS OUT Endpoint Lower Control Status Register

USBHS OUT Endpoint Lower Control Status			USBHS_OUTCSRL	[0x0016]
Bits	**Name**	**Access**	**Reset**	**Description**
7	clrdatatog	R/W1O	0	**Clear OUT Endpoint Data Toggle** 1: Clear the OUT Endpoint data toggle to 0. *Note: Automatically cleared.*
6	sentstall	R/W0C	0	**STALL Handshake Sent Status** Automatically set when a STALL handshake is transmitted. Write a 0 to clear.
5	sendstall	R/W	0	**Send STALL Handshake** 1: Send a STALL handshake to a data packet 0: Terminate STALL handshake Ignored for Isochronous transfers. Write a 0 to clear.
4	flushfifo	R/W1O	0	**Flush OUT FIFO Packet** 1: Flush the next packet to be read from the OUT FIFO. This also clears the *outpktrdy* bit. This must only be set when *outpktrdy* = 1, or data corruption in the FIFO might occur. If the out FIFO contains two packets, *flushfifo* might need to be set twice to completely clear the FIFO. *Note: Automatically cleared when the packet is flushed.*
3	dataerror	RO	0	**OUT Packet CRC Error Status** Isochronous Mode: Automatically set if a data packet is received (*outpktrdy* = 1), and the data packet has a CRC error. Automatically cleared when *outpktrdy* = 0. Bulk or Interrupt Modes: Always returns 0.
2	overrun	R/W0C	0	**OUT FIFO Overrun Error Status** Isochronous Mode: Automatically set if the OUT FIFO is full (*fifofull* = 1), and an OUT packet arrives. In this case, the OUT packet is lost. Bulk or Interrupt Modes: Always reads 0. *Note: Write a 0 to clear.*
1	fifofull	RO	0	**FIFO Full Status** Set when the OUT FIFO is full. *Note: Automatically cleared when the FIFO is no longer full.*

USBHS OUT Endpoint Lower Control Status		USBHS_OUTCSRL		[0x0016]
Bits	**Name**	**Access**	**Reset**	**Description**
0	outpktrdy	R/W0C	0	**OUT Packet Ready Status** Automatically set when a data packet is received in the OUT FIFO. Write a 0 to clear after the packet is unloaded from the OUT FIFO. *Note: Write 0 to clear.*

Table 22-25: USBHS OUT Endpoint Upper Control Status Register

USBHS OUT Endpoint Upper Control Status Register			USBHS_OUTCSRU	[0x0017]
Bits	**Name**	**Access**	**Reset**	**Description**
7	autoclear	R/W	0	**Auto Clear outpktrdy** 0: *USBHS_OUTCSRL.outpktrdy* must be cleared by firmware. 1: *USBHS_OUTCSRL.outpktrdy* is automatically cleared when data that is of the maximum packet size specified in the *USBHS_OUTMAXP* register is unloaded from the OUT FIFO. If packets less than the maximum packet size are unloaded, *outpktrdy* must be cleared by firmware. *Note: Do not set for High Bandwidth Isochronous endpoints.*
6	iso	R/W	0	**Isochronous Transfer Enable** 0: Enable OUT Bulk and OUT Interrupt transfers. 1: Enable OUT Isochronous transfers.
5	-	-	0	**Reserved**
4	disnyet/piderror	R/W R/W0C	0 0	**Disable NYET Packets (HS Bulk and HS Interrupt Modes)** 0: If the OUT FIFO is full, respond to newly received OUT packets with a NYET (Not Yet) packet to indicate the FIFO is full. 1: Disable NYET packets. Respond to all received OUT packets with an ACK even when the FIFO is full. **PID Error Status (Isochronous Mode only)** Automatically set if there is a PID (Packet ID) error in the received OUT packet. *Note: Write 0 to clear.* *Note: Ignored in all other modes.* *Note: Bit 4 is dual-use and can be addressed by two different names depending on the endpoint mode.*
3	-	-	0	**Reserved**
2	-	R/W	0	**Reserved for Future Use** Do not modify this field.
1	dpktbufdis	R/W	0	**Double Packet Buffering Disable** 0: Enable double packet buffering. Firmware must configure the FIFO and packet size. 1: Disable double packet buffering.

USBHS OUT Endpoint Upper Control Status Register		USBHS_OUTCSRU		[0x0017]
Bits	**Name**	**Access**	**Reset**	**Description**
0	incomprx	R	0	**Incomplete Isochronous Packet Received Error Status** High Bandwidth Isochronous Mode: Automatically set if an incomplete packet is received in the OUT FIFO. Automatically cleared when *USBHS_OUTCSRL.outpktrdy* is cleared. Bulk or Interrupt Modes: Always reads 0.

Note: Endpoints 1 to 11 have a memory-mapped version of this register selected using the USBHS_INDEX register.

Table 22-26: USBHS Endpoint OUT FIFO Byte Count Register

USBHS Endpoint OUT FIFO Byte Count			USBHS_OUTCOUNT	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
15:13	-	RO	0	**Reserved for Future Use** Do not modify this field.
12:0	outcount	RO	0	**Read Number of Data Bytes in OUT FIFO** Returns the number of data bytes in the packet that are read next in the OUT FIFO. *Note: This value changes as the contents of the FIFO change.* *Note: This value is only valid when a packet is in the OUT FIFO (USBHS_OUTCSRL.outpktrdy = 1).*

Table 22-27: USBHS Endpoint 0 IN FIFO Byte Count Register

USBHS Endpoint 0 IN FIFO Byte Count			USBHS_COUNT0	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
15:7	-	RO	0	**Reserved for Future Use** Do not modify this field.
6:0	count0	RO	0	**Read Number of Data Bytes in the Endpoint 0 FIFO** Returns the number of data bytes in the Endpoint 0 FIFO. *Note: This field changes as the contents of the FIFO change.* *Note: This field is only valid when USBHS_OUTCSRL.outpktrdy = 1.*

Table 22-28: USBHS FIFO for Endpoint n Register

USBHS FIFO for Endpoint 0	**USBHS_FIFO0**	**[0x0020]**
USBHS FIFO for Endpoint 1	**USBHS_FIFO1**	**[0x0024]**
USBHS FIFO for Endpoint 2	**USBHS_FIFO2**	**[0x0028]**
USBHS FIFO for Endpoint 3	**USBHS_FIFO3**	**[0x002C]**
USBHS FIFO for Endpoint 4	**USBHS_FIFO4**	**[0x0030]**
USBHS FIFO for Endpoint 5	**USBHS_FIFO5**	**[0x0034]**
USBHS FIFO for Endpoint 6	**USBHS_FIFO6**	**[0x0038]**
USBHS FIFO for Endpoint 7	**USBHS_FIFO7**	**[0x003C]**
USBHS FIFO for Endpoint 8	**USBHS_FIFO8**	**[0x0040]**
USBHS FIFO for Endpoint 9	**USBHS_FIFO9**	**[0x0044]**
USBHS FIFO for Endpoint 10	**USBHS_FIFO10**	**[0x0048]**
USBHS FIFO for Endpoint 11	**USBHS_FIFO11**	**[0x004C]**

Bits	Name	Access	Reset	Description
31:0	usbhs_fifo	R/W	-	**USBHS Endpoint FIFO Read/Write Register** Reads from this register unload data from the OUT FIFO for the corresponding endpoint. Writes to this register load data into the IN FIFO for the corresponding endpoint. FIFO reads and writes may be 8-bit, 16-bit, 24-bit or 32-bit. Any combination is allowed provided the data accessed is contiguous. However, all reads and writes for a packet must be of the same width so that the data is consistently byte-, word- or double-word-aligned. The last transfer can contain fewer bytes than the previous transfers when completing an odd-byte or odd-word transfer. *Note: The value of these registers at reset is undetermined.*

Table 22-29: USBHS Endpoint Count Info Register

USBHS Endpoint Count Info	**USBHS_EPINFO**	**[0x0078]**

Bits	Name	Access	Reset	Description
7:4	outendpoints	RO	0xB	**Number of OUT Endpoints** There are 11 OUT endpoints in this USBHS peripheral. 0xB: 11 OUT Endpoints.
3:0	inendpoints	RO	0xB	**Number of IN Endpoints** Returns the number of IN endpoints in this USBHS peripheral. 0xB: 11 IN Endpoints

Table 22-30: USBHS RAM Info Register

USBHS RAM Info	**USBHS_RAMINFO**	**[0x0079]**

Bits	Name	Access	Reset	Description
7:4	-	RO	0	**Reserved for Future Use** Do not modify this field.

USBHS RAM Info			USBHS_RAMINFO	[0x0079]
Bits	**Name**	**Access**	**Reset**	**Description**
3:0	rambits	RO	0xC	**Number of RAM Address Bits** The width of the RAM address bus in this USBHS module. The width is 12 bits. 0xC: 12-bit-wide RAM address supported in the USB HS peripheral.

Table 22-31: USBHS Soft Reset Control Register

USBHS Soft Reset Control			USBHS_SOFTRESET	[0x007A]
Bits	**Name**	**Access**	**Reset**	**Description**
7:2	-	R/W	0	**Reserved for Future Use** Do not modify this field.
1	rstxs	R/W1O	0	**Reset the USB PHY.** Write a 1 to reset the USB PHY. This field is cleared by hardware automatically after a 1 is written and the USB PHY is reset. 0: USB PHY reset complete or not initiated. 1: Write 1 to reset the USB PHY.
0	rsts	R/W1O	0	**Reset the USB Controller.** Write 1 to reset the USBHS controller. This field is cleared by hardware automatically after a 1 is written and the USBHS controller is reset. 0: USBHS controller reset complete or not initiated. 1: Write 1 to reset the USBHS controller.

Table 22-32: USBHS Hi-Speed Chirp Timeout Register

USBHS Hi-Speed Chirp Timeout			USBHS_CTUCH	[0x0080]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	c_t_uch	R/W	0x203A	**HS Chirp Timeout Clock Cycles** This configures the chirp timeout used by this device to negotiate a HS connection with a FS host. $t_{CHIRP\_TIMEOUT}(PHY\ clock\ cycles) = c\_t\_uch \times 4$ The timeout value represents the number of 30MHz PHY clock cycles (66.7ns) before the chirp timeout occurs.

Table 22-33: USBHS Hi-Speed RESUME Delay Register

USBHS Hi-Speed RESUME Delay			USBHS_CTHSRTN	[0x0082]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	c_t_hsrtn	R/W	0x0019	**Hi-Speed RESUME Delay Clock Cycles** This configures the delay from when the RESUME state on the bus ends, to when the USBHS resumes normal operation. $t_{HI\_SPEED\_DELAY}(PHY\ clock\ cycles) = c\_t\_hsrtn \times 4$ The delay value represents the number of 30MHz PHY clock cycles (66.7ns) from the end of the RESUME state to when normal USBHS operation begins.

Table 22-34: USBHS Power-On Reset Register

USBHS Power-On Reset			USBHS_M31_PHY_PONRST	[0x0410]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	phy_reset	R/W	1	**USBHS PHY Reset** Set this bit to zero to generate a reset of the USBHS PHY. The USBHS PHY dissipates minimum power while it is in the reset state. System reset also resets the USBHS PHY. 0: The USBHS PHY is held in reset. 1: The USBHS reset is controlled by system reset.

Table 22-35: USBHS Hi-Speed VBUS Reset Register

USBHS VBUS Reset			USBHS_M31_PHY_NONCRY_RSTB	[0x0414]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	noncry_rstb	R/W	0	**USBHS VBUS Reset** The software control of this bit is determined by the type of USB device desired. For a bus-powered device, this bit should be set to 1. For a self-powered device, this bit should be set to 1 when VBUS is on and 0 when VBUS is off.

	noncry_rstb	VBUS
Bus-Powered Device	Set to 1	Don't Care
Self-Powered Device	Set to 1	ON
	Set to 0	OFF

Table 22-36: USBHS Non-Clock Recovery Enable

USBHS Hi-Speed RESUME Delay			USBHS_M31_PHY_NONCRY_EN	[0x0418]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	noncry_en	R/W	0	**Non-Clock Recovery Enable** Set this field to 1 to ensure the USBHS interface uses clock recovery from the USB data as its data clock source. 0: Invalid 1: Clock recovery enabled

Table 22-37: USBHS PLL Enable Register

USBHS PLL Enable			USBHS_M31_PHY_PLL_EN	[0x0430]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**

USBHS PLL Enable			USBHS_M31_PHY_PLL_EN	[0x0430]
Bits	**Name**	**Access**	**Reset**	**Description**
0	pll_en	R/W	0	**PLL Enable** Set this bit to 1 to ensure the PLL is enabled for clock recovery from the USB data. See *USBHS_M31_PHY_NONCRY_EN.noncry_en.* 0: Invalid 1: Clock recovery enabled

Table 22-38: USBHS Oscillator Output Enable Register

USBHS Oscillator Output Enable			USBHS_M32_PHY_OSCOUTEN	[0x043C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	oscouten	R/W	0	**Oscillator Output Enable** Set this bit to 1 to ensure the PLL is enabled for clock recovery from the USB data. See *USBHS_M31_PHY_NONCRY_EN.noncry_en.* 0: Invalid 1: Clock recovery enabled

Table 22-39: USBHS Hi-Speed Core Clock Input Register

USBHS Hi-Speed Core Clock Input			USBHS_M32_PHY_CORECLKIN	[0x0448]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1		RO	0	Reserved
0	coreclkin	R/W	0	**Core Clock Input** 0: Clock recovery from the data 1: Invalid

Table 22-40: USBHS Hi-Speed Clock Source Frequency Select Register

USBHS Hi-Speed Clock Source Frequency Select			USBHS_M32_PHY_XTLSEL	[0x044C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3		RO	0	**Reserved**
2:0	xtlsel	R/W	0	**Clock Source Frequency Select** 0: 10Mhz 1: 12MHz 2: 25MHz 3: 30MHz 4: 19.2MHz 5: 24MHz 6: Reserved 7: Reserved

Table 22-41: USBHS Hi-Speed Reference Clock Select Register

USBHS Hi-Speed Reference Clock Select			USBHS_M32_PHY_OUTCLKSEL	[0x045C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1		RO	0	**Reserved**

USBHS Hi-Speed Reference Clock Select			USBHS_M32_PHY_OUTCLKSEL	[0x045C]
Bits	**Name**	**Access**	**Reset**	**Description**
0	outclksel	R/W	0	**Reference Clock Select** 0: Clock recovery from the data 1: Invalid

Table 22-42: USBHS Hi-Speed VBUS Interrupt Register

USBHS Hi-Speed VBUS Interrupt			USBHS_MXM_INT	[0x0498]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2		RO	0	**Reserved**
1	rise	R/W1C	0x	**VBUS Rising Edge Detect Flag** This flag will generate an interrupt when set if *USBHS_MXM_INT_EN.rise_en is set.* 0: VBUS not detected 1: VBUS rising edge detected
0	fall	R/W1C	0x	**VBUS Falling Edge Detect Flag** This flag will generate an interrupt when set if *USBHS_MXM_INT_EN.fall_en is set.* 0: VBUS not detected 1: VBUS falling edge detected

Table 22-43: USBHS Hi-Speed VBUS Interrupt Enable Register

USBHS Hi-Speed VBUS Interrupt Enable			USBHS_MXM_INT_EN	[0x049C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2		RO	0	**Reserved**
1	rise_en	R/W	0	**VBUS Rising Edge Detect Enable** An interrupt will be generated if *USBHS_MXM_INT.rise* is set. 0: Disabled 1: Enabled
0	fall_en	R/W	0	**VBUS Falling Edge Detect Enable** An interrupt will be generated if *USBHS_MXM_INT.fall* is set. 0: Disabled 1: Enabled

Table 22-44: USBHS Suspend Register

USBHS Suspend			USBHS_MXM_SUSPEND	[0x04A0]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	suspend	R/W	0	**Soft Connect/Disconnect PHY** The hardware or the software can control the SUSPEND operation of the USBHS PHY. The *USBHS_MXM_SUSPEND.sel* field enables this field to provide control of the SUSPEND operation to the software. 0: The USBHS PHY is not suspended. 1: The USBHS PHY is suspended.

USBHS Suspend			USBHS_MXM_SUSPEND	[0x04A0]
Bits	**Name**	**Access**	**Reset**	**Description**
0	sel	R/W	1	**Suspend Control Select** The SUSPEND operation of the USBHS PHY can be controlled by the hardware or the firmware. This field selects the operating mode of the SUSPEND operation. 0: The USBHS controller hardware will control the PHY suspend operation. 1: The PHY SUSPEND operation is controlled by software using the *USBHS_MXM_SUSPEND*.*suspend* bit.

Table 22-45: USBHS Hi-Speed VBUS State Register

USBHS Hi-Speed VBUS State			USBHS_MXM_REG_A4	[0x04A4]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	vddb	RO	0	**VBUS State** 0: VBUS is not present 1: VBUS is present

23. Audio Subsystem

The audio interface allows the device to communicate with external audio devices using standard I^2S pulse-coded modulation (PCM) and pulse density modulation (PDM) audio interfaces. Application software allows audio algorithms such as dynamic speaker management (DSM) or ambient noise cancellation (ANC).

PCM mode Features:

- Single PCM interface with 4 pins (BCLK, LRCKLK, DIN, DOUT) allowing transmission and reception of audio data
- 4 transmit channels and 8 receive channels support 4-channel DSM
- Support 4 PDM receive channels at the same time as PCM

PDM mode features:

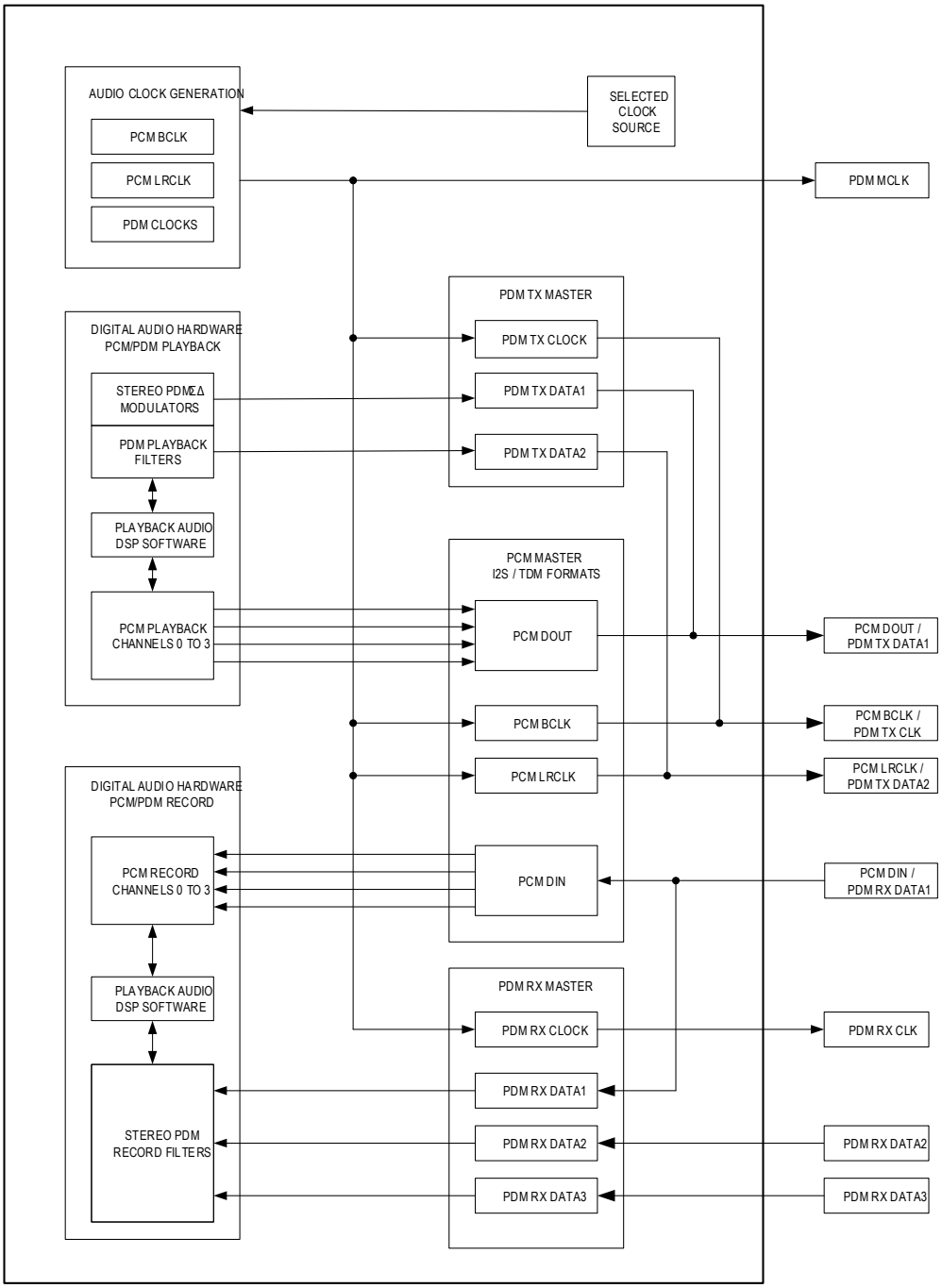
- Up to 2 PDM transmit channels
- Up to 6 PDM receive channels
- Interpolation and Decimation filtering

Special considerations when using the Audio Subsystem:

- The use of an external clock source is required for most applications to meet sampling rate stability requirements.
- The peripheral does not support DMA.
- The FIFO depth is four levels. The software must prioritize the peripheral interrupts, which can occur as frequently as one-half the sample rate if using the FIFO half-full interrupt to generate service requests.

A block diagram of the audio subsystem is shown below.

Figure 23-1: Audio Subsystem Block Diagram



23.1 Instances

There is one instance of the Audio Subsystem peripheral.

Table 23-1: MAX32665/MAX32666 Audio Digital Signals

GPIO	Signal
P0.23	External clock input for Audio Subsystem
P0.24	PCM_LRCLK PDM TX DATA 2
P0.25	PCM_DOUT PDM TX DATA 1
P0.26	PCM_DIN PDM RX DATA 1
P0.27	PCM_BCLK PDM TX CLK
P0.28	PDM_DATA2
P0.29	PDM_DATA3
P0.30	PDM_RX_CLK
P0.31	PDM_MCLK

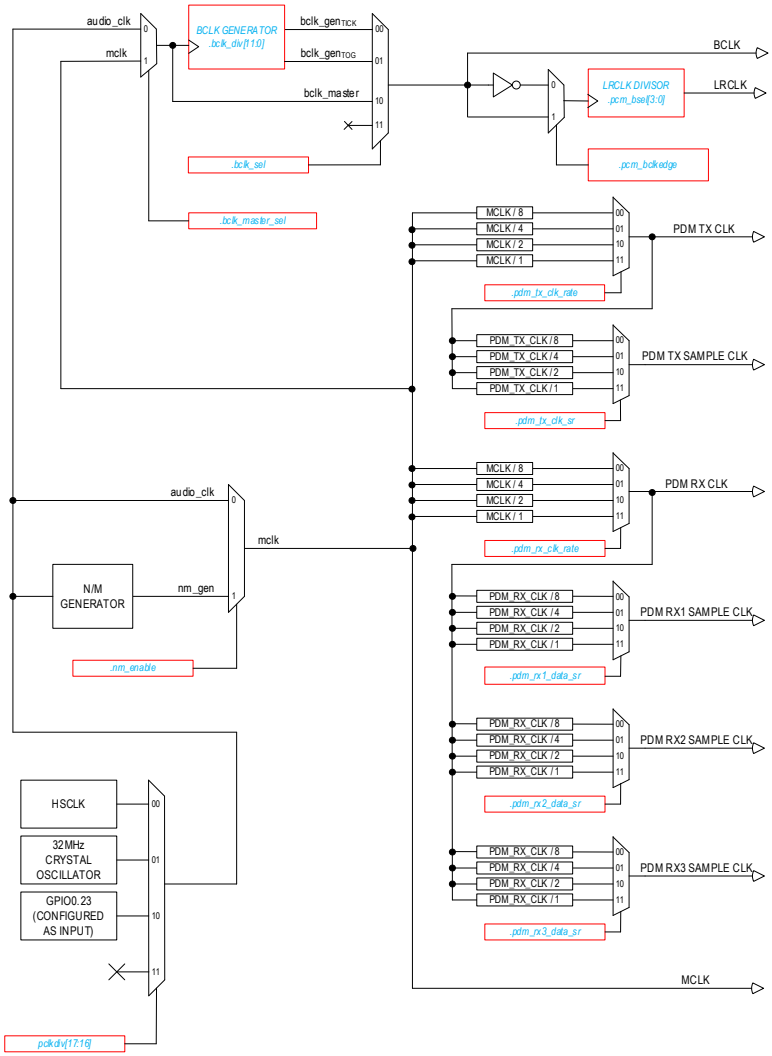
23.2 Clocking

The audio subsystem gets its clock from an external source to faithfully provide sample rate accuracy. This external source should be a CMOS-level signal. Its specification for accuracy is left up to the user to determine.

1. Clear *GLOBAL_ENABLE*.*enable* to 0 to disable the audio subsystem.
2. Set *GCR_PCKDIV*.*audclksel* to 2 to select the external clock source.
   a. External clock source on GPIO P0.23. The pin must be configured as an input.
3. Set *GLOBAL_ENABLE*.*enable* to 1 to enable the audio subsystem.

The Audio_Master_Clock output (MCLK) output is available in PCM mode for any slave devices that require it.

Figure 23-2: Audio Subsystem Clock Tree



23.2.1 Setting the Master Clock for the Audio Peripheral

The Audio peripheral requires an Audio_Master_Clock at 12.288MHz or 11.2896MHz, which are common audio master clock rates. The peripheral has an integrated N/M clock divider to create this clock.

The N/M ratio is calculated as in the following example.

f_{IN} =	96,000,000Hz
f_{OUT} = Audio_Master_Clock =	12,288,000Hz
GCD(f_{OUT}, f_{IN}) =	768,000
M = f_{IN} / GCD(f_{IN}, f_{OUT}) =	125
N = M * f_{OUT} / f_{IN} =	16
f_{OUT} (check) =	12,288,000Hz

23.3 FIFO and Data Management

Each audio channel has a 4-deep FIFO to give a small data buffer. The depth of the FIFO is deliberately small to minimize system latency.

The FIFOs need to be continuously read/from and written to maintain the flow of audio data to/from the peripheral. Software must manage the audio data to avoid dropped or repeated samples in two ways:

1. Scheduling FIFO reads/writes at an appropriate rate to prevent FIFO overflow/underflow.
2. Using the FIFO interrupts to determine when data is required to be written or read.

With the 4-deep FIFO on each channel, the CPU can allow a small build-up of data before servicing the FIFO. For example, the CPU could schedule reads/writes every 2*L clock cycles and transfer 2x the data.

The write and read pointers are updated only after ALL of the enabled channels (in this case, channel 0 and channel 1) have been received. This maintains the phase relationship between the 2 (or more) channels.

Note that the Rx FIFO is 4 levels deep. The almost-full flag is set while there are three OR four samples available. If enabled, the almost-full interrupt asserts when the FIFO goes from two unread samples to three unread samples. In the case of a slow ISR, it is possible for a fourth sample to be received before the third can be read. Any ISR triggered by the almost-full flag should read three samples from the FIFO, and if there is a fourth sample, it will be the first sample the next time the almost flag is set again.

There are 3 interrupts for each FIFO:

- Almost Empty – indicates that the FIFO has 0 or 1 samples
- Half Full – indicates that there are 2 samples stored in the FIFO
- Almost Full – indicates that there are 3 or 4 samples stored in the FIFO

For Rx channels, the following actions should be considered:

- Almost Empty – data is available and can be read at the next opportune time
- Half Full - data is available and can be read at the next opportune time
- Almost Full – data should be read immediately to avoid loss of data

For Tx channels, the following actions should be considered:

- Almost Empty – data needs to be supplied immediately to avoid the FIFO running out of data
- Half Full – data can be written if required
- Almost Full – one more sample can be written if required

Configuring the IRQs

Each channel has an Almost Empty read-only bitfield which is named

- AE_[PDM|PCM]_[RX|TX]_CH[n]

Each channel has a Half Full read-only bitfield, which is named

- HF_[PDM|PCM]_[RX|TX]_CH[n]

Each channel has an Almost Full read-only bitfield which is named

- AF_[PDM|PCM]_[RX|TX]_CH[n]

Reading these registers gives the status of each of the FIFOs. The read-only registers can be read at any time.

23.4 PCM Master Interface

The PCM master interface supports the standard stereo I^2S data format and three TDM data format variations with up to 8 channels each.

PCM mode supports up to 4 transmit and 8 receive channels. Multiple formats are supported.

The PCM interface receives serial data for all channels on the PCM DIN pin and transmits it on the PCM DOUT pin.

The peripheral generates the Bit Clock (BCLK) and Frame Clock (LRCLK, also called Left/Right Clock) for the PCM interface.

The 4-wire I^2S PCM interface allows audio data to transmit and receive and is intended to connect to speaker driver devices.

23.4.1 PCM Master Interface Clock Configuration

The PCM master interface is available in PCM mode and is automatically disabled in PDM Mode. When enabled, the PCM master interface supplies both a bit clock (PCM_BCLK) and frame clock (PCM_LRCLK) to an attached PCM slave device. To correctly configure the PCM master interface clock timing, the host must program both the PCM data sample rates (playback path and record path) and the PCM clock ratio (PCM_BCLK to PCM_LRCLK ratio).

In PCM mode, a system-generated audio master clock output (MCLK) is also optionally available for any slave devices that require it.

23.4.2 Setting the Sample Rate for PCM Mode

The PCM mode sample rate is equivalent to the LRCLK frequency. The LRCLK frequency is expressed as a number of BCLKs/LRCLK, i.e., the number of Bit Clocks per LRCLK period. The PCM_BSEL register field sets this.

This then depends on the BCLK frequency.

BCLK is divided down from the audio clock source according to the BCLK_DIV register field. BCLK_DIV can be any integer value, including 1. The BCLK frequency is

F_BCLK = Audio Master Clock / BCLK_DIV.

The user should follow this process to decide the clocking setup:

1. Set the desired sample rate. For this illustration, it will be 48000Hz.
2. Determine the Audio_Master_Clock Rate.
3. Determine the maximum number of channels needed in either Tx or Rx direction.
    a. Total_Chans = MAX(RX_CHANS, TX_CHANS)
    b. If Total_chans > 2, TDM format must be used
    c. If Total_chans ≤ 2, either I^2S or TDM can be used.
4. Determine the number of BCLKs/LRCLK
    a. BCLKs/LRCLK must be greater than or equal to (Total_Chans x Word Size).
    b. Word size is either 16, 24, or 32 bits and is set by the PCM_CHANSZ register field.
    c. For example, 4 channels at 16 bits require at least 64 BCLKs/LRCLK. So we could use PCM_BSEL set to 4, 5, 6, 7, 8, 9, 10, or 12.
5. Set PCM_BSEL. For this illustration, we will consider PCM_BSEL = 4 (64 BCLKs/LRCLK). Our desired BCLK frequency is (Sample rate x BCLKs/LRCLK)

a. F_BCLK = 48000 x 64 = 3072000

b. BCLK_DIV = Audio_Master_Clock / F_BCLK = 12288000 / 3072000 = 4.

The number of audio channels possible in Tx or Rx direction is calculated by:

MAX_CHANS = 12288000 / (WORD_SIZE * SAMPLE_RATE)

23.4.3 PCM Data Sample Rate and Clock Ratio Configuration

The base sample rate of the PCM master interface is always equal to that of the PCM playback data and must be configured to match the frequency of the PCM frame clock (PCM_LRCLK). The PCM playback path contains 4 channels internally that all run at the configured playback data sample rate, and these can be routed to any channel of the PCM interface data output (PCM_DOUT)

The PCM record data sample rate must be set to either the same sample rate as the PCM playback data or to a lower integer ratio sample rate (relative to the PCM playback data sample rate) according to the restrictions in the table below. The PCM record path contains 4 channels internally that all run at the configured record data sample rate, and each of these can accept data from any single channel of the PCM data input (PCM_DIN). When the record path is set to a lower sample rate than the playback path, the PCM input data (from PCM_DIN channels) may populate extra frames (based on the playback to record path sample rate ratio) with either repeated data or zero code samples.

The device supports a range of bit clock (PCM_BCLK) to frame clock (PCM_LRCLK) ratios ranging from 32 to 256. The PCM Record Path Sample Rate can be equal to or an integer divider of the PCM Interface/Playback Path Sample Rate.

Table 23-2: PCM Record and Playback Sample Rates

N/A = Not Available N/S = Not Supported ## = Supported Clock Ratio		**PCM Record Path Sample Rate (kHz) (PCM_DIN)**									
		48	**44.1**	**32**	**24**	**22.05**	**16**	**12**	**11.025**	**8**	**4**
PCM Playback Path Sample Rate (kHz) (PCM_DOUT)	192	4	N/S	6	8	N/S	12S	16	N/S	24	N/S
	176.4	N/S	4	N/S	N/S	8	N/S	N/S	16	N/S	N/S
	96	2	N/S	3	4	N/S	6	8	N/S	12	N/S
	88.2	N/S	2	N/S	N/S	4 S	N/S	N/S	8	N/S	N/S
	48	1	N/S	N/S	2	N/S	3	4	N/S	6	N/S
	44.1	N/A	1	N/S	N/S	2	N/S	N/S	4	N/S	N/S
	32	N/A	N/A	1	N/S	N/S	2	N/S	N/S	4	N/S
	24	N/A	N/A	N/A	1	N/S	N/S	2	N/S	3	N/S
	22.05	N/A	N/A	N/A	N/A	1	N/S	N/S	2	N/S	N/S
	16	N/A	N/A	N/A	N/A	N/A	1	N/S	N/S	2	N/S
	12	N/A	N/A	N/A	N/A	N/A	N/A	1	N/S	N/S	N/S
	11.025	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1	N/S	N/S
	8	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1	N/S

23.5 I²S Mode Operation

The I²S data format supports two channels that can be 16, 24, or 32 bits in length. The PCM_BCLK to PCM_LRCLK ratio must be configured to be twice the desired channel length to accommodate both channels. The audio data word size is configurable to 16, 24, or 32 bits in length and must be programmed to be less than or equal to the channel length set by the clock ratio as shown in *Table 23-3*. If the resulting channel length exceeds the configured data word size, then the PCM data input (PCM_DIN) LSBs are truncated, and the PCM data output (PCM_DOUT) LSBs are padded with either zero or Hi-Z data.

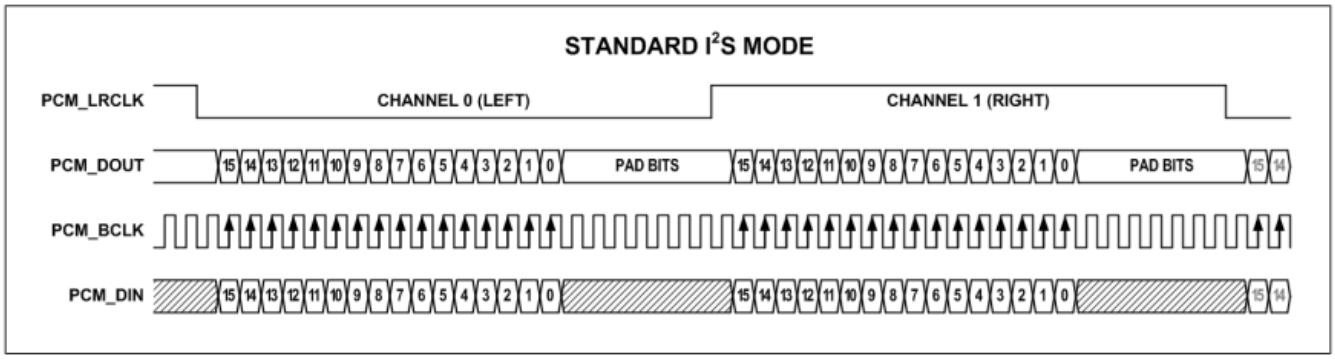
Table 23-3: Supported I²S Combinations

Number of Channels	Channel Length	PCM_BCLK to PCM_LRCLK ratio	Supported Data Word Sizes
2	16	32	16
	24	48	16, 24
	32	64	16, 24, 32

In I²S mode, the active PCM_LRCLK edge denotes the start of a new frame and the left channel data (Channel 0), and a PCM_LRCLK edge precedes the right channel data (Channel 1). The MSB of the audio data word latches on the second active PCM_BCLK edge after a PCM_LRCLK transition.

The active PCM_BCLK edge that is used for data capture (from PCM_DIN) and data output (PCM_DOUT) can be configured as rising or falling. All PCM_LRCLK edge transitions, including the edge that indicates the start of a new frame (channel 0), always align with the inactive PCM_BCLK edge. The data output is valid on the same active PCM_BCLK edge as the data input. The data output also transitions on the same edge as the data input. An example of the baseline I²S data format is shown below.

Figure 23-3: Standard I²S Mode Data Format Example (16-bit Audio Data Word, 24-bit Channel Length)



23.6 PCM Slave Interface Internal Data Routing

The PCM master Interface supports both the I²S and TDM data formats for external data transmit (PCM_DOUT) and receive (PCM_DIN). In I²S mode, there are 2 external data channels denoted as channel 0 (left) and channel 1 (right). In TDM modes, there can be either 2, 4, or 8 external data channels numbered sequentially from 0 to 7. Data is always transmitted and received MSB to LSB.

The internal PCM playback and record paths each support 4 internal 32-bit data channels. Each individual external data channel at the PCM master interface (up to 8 in TDM mode) can be routed to/from any one of the internal playback/record path data channels. The only exception to this is in channel sharing mode, where two paired external data channels can share a single internal data channel.

For the record path, if the input data word size is less than 32 bits, the internal record data channel will pad out the extra bits with zeroes. If the output data channel word size is less than 32 bits for the playback path, the lowest N bits of the internal playback data channel will be truncated down to the selected output word size. The PDM Tx Interface is automatically disabled in PCM mode.

23.6.1 PCM Input Data Channel Sharing (PCM_DIN)

PCM input data (received into PCM_DIN) may be combined into a single evenly split 32-bit internal record channel. This occurs when two channels of PCM input data (PCM_DIN) are assigned to a single internal record path channel. However, this is only possible for the specific paired input channels shown in *Table 23-4*.

Table 23-4: PCM Input Data Channel Sharing Pairs

Pair	First Input Data Channel (PCM_DIN)	Second Input Data Channel (PCM_DIN)	Structure of Shared 32-bit Internal Record Channel
1	Channel 0	Channel 1	16 bits of Channel 0 data then 16 bits of Channel 1 data
2	Channel 2	Channel 3	16 bits of Channel 2 data then 6 bits of Channel 3 data
3	Channel 4	Channel 5	16 bits of Channel 4 data then 16 bits of Channel 5 data
4	Channel 6	Channel 7	16 bits of Channel 6 data Then 16 bits of Channel 7 data

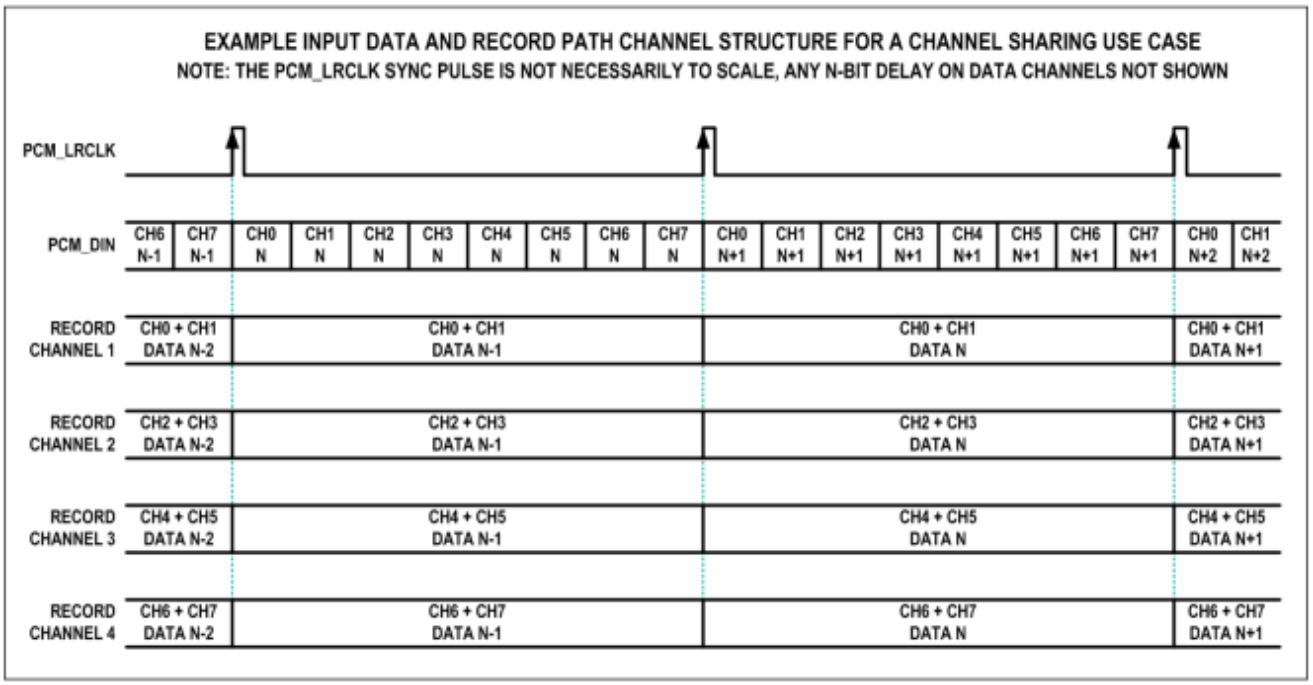
When both PCM input data channels of a given pair are assigned to the same internal record path channel, the data from the lower numbered channel occupies the first half of the internal record channel, followed by the data from the higher numbered channel. When an internal record channel is being shared, the data word is evenly split between the two input channels. If the combined data from the two input channels exceeds the internal channel length, then the data is evenly truncated to fit (lowest bits are dropped) as shown in *Table 23-5*. Attempting to assign 2 or more non-paired PCM input data channels to a single internal record channel is not supported, and only the data from the lowest-numbered PCM input data (PCM_DIN) channel is assigned to the internal record path channel.

Table 23-5: Channel Truncation Based on Data Input Word Size

Data Input Word Size	Truncation Needed
16-bits per Input Channel	None
24-bits per Input Channel	Lowest 8-bits of Each Input Channel
32-bits per Input Channel	Lowest 16-bits of Each Input Channel

Figure 23-4 is an example case for a 4-speaker system. There are 8 channels (TDM Mode) of input data using all 4 of the internal 32-bit record path channels.

Figure 23-4: PCM Channel Sharing Example



23.7 TDM Mode Operation

The provided TDM modes each support up to 8 audio channels that are 16-bits, 24-bits, or 32-bits in length each. In TDM mode, the channel length and data word size are always equal. The number of TDM channels is determined by both the selected bit clock (PCM_BCLK) to frame clock (PCM_LRCLK) ratio and the selected size. The formula is as follows:

Number of Channels = floor(PCM_BCLK to PCM_LRCLK Ratio / Channel Length)

The configured sample rate and clock ratio cannot require a PCM_BCLK frequency less than 256kHz or greater than 12.288MHz, as shown in *Table 23-6*.

Table 23-6: Supported TDM Mode Configurations

Number of Channels	Channel Length/Data Word Size	PCM_BCLK to PCM_LRCLK CLock Ratio	Maximum Supported Sample Rate (f_{PCM_LRCLK})
2	16	32	192kHz
	24	48	
	32	64	
4	16	64	96kHz
	24	96	
	32	128	
8	16	128	48kHz
	24	192	
	32	256	

With the default PCM interface settings, in TDM mode, a rising frame clock (PCM_LRCLK) edge acts as the frame sync pulse and indicates the start of a new frame. The frame sync pulse width (high pulse) will be equal to the width of at least one bit clock period. The active PCM_LRCLK edge (sync pulse) used to start a TDM frame can be inverted to a falling edge (low pulse).

In TDM mode, the MSB of the first audio word can be latched on the first, second, or third active PCM_BCLK edge after the sync pulse (TDM mode 0, 1, and 2). Additionally, the active PCM_BCLK transition that is used for data capture and data output can be configured to either the rising or falling edge as shown in *Figure 23-5*. The data output is valid on the same active PCM_BCLK edge as the data input. The data output also transitions on the same edge as data input.

Figure 23-5: TDM Mode 0 Data Format, Normal and Inverted BCLK

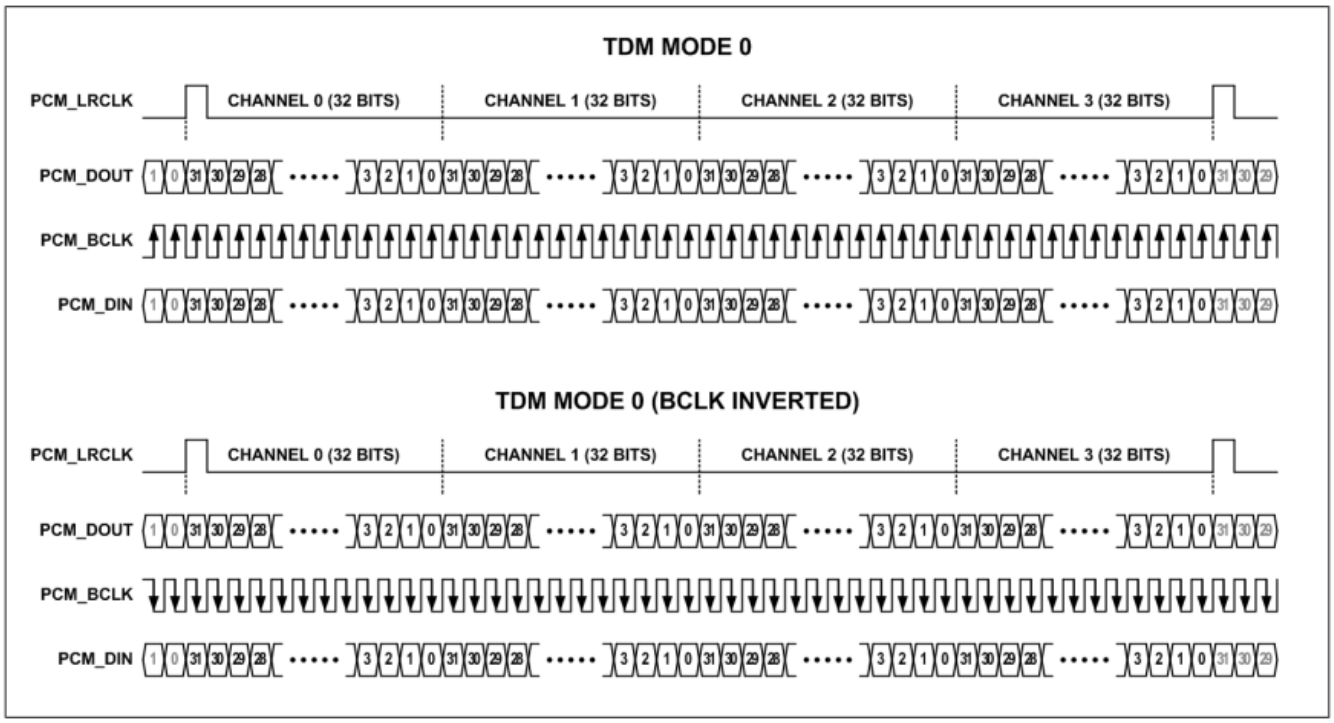
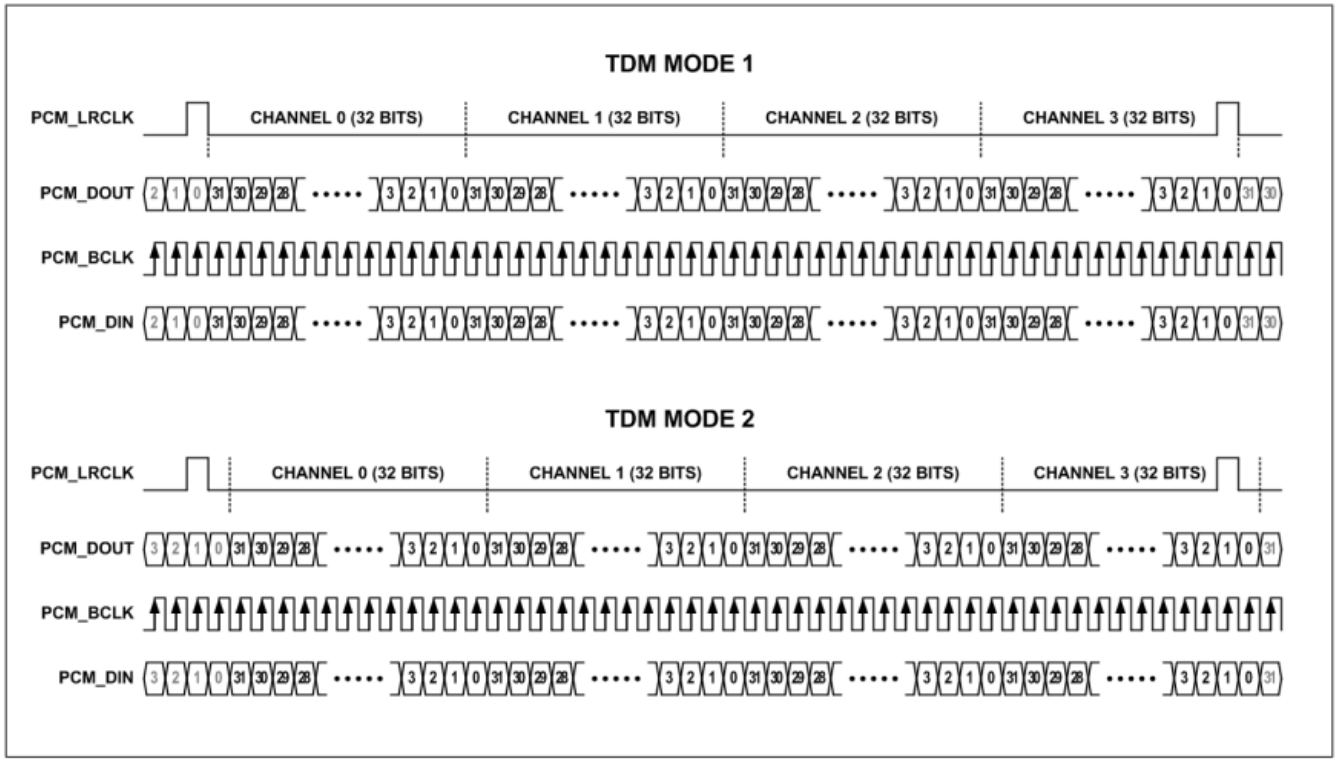


Figure 23-6: TDM Mode 1 and Mode 2



23.8 PDM Master Interface Transmitter (Tx) and Receiver (Rx)

PDM mode supports up to 2 transmit and 6 receive channels.

The peripheral up-samples the data using a CIC filter and then modulates it to 1-bit or 1.5-bit using a sigma-delta modulator.

Receive channels are 1-bit data with two channels per data pin. The data received is pulse-density modulated data (usually created by a sigma-delta modulator). This data is down-sampled by a CIC filter. Reads from each channel should be equally spaced at the sample rate.

The PDM master interface transmitter (Tx) is automatically disabled in PCM mode. When enabled, the PDM master interface Tx supplies both the PDM transmit clock (PDM_TX_CLOCK) and the PDM playback data outputs (PDM_TX_DATA1 and PDM_TX_DATA2) to attached PDM slave device receivers.

The PDM master interface receiver (Rx) data input 1 (PDM_RX_DATA1) is only available in PDM mode and is automatically disabled in PCM mode. However, the PDM receiver data inputs 2 and 3 (PDM_RX_DATA2 and PDM_RX_DATA3) are available in both PDM and PCM mode. When enabled, the PDM master interface Rx can operate from the same clock as the PDM Tx (PDM_TX_CLOCK), or it can supply a separate PDM receiver clock (PDM_RX_CLOCK).

23.8.1 PDM Master Interface Operation in PDM Mode

In PDM mode, the PCM interface is inactive, and all audio data is transmitted/received over the PDM interface transmitter (Tx) and receiver (Rx). In this mode, there are two primary cases - one where a single clock can be shared for transmitting/receiving and a second where the receiver clock is either an integer divider of the transmit clock or the system clock. Each of these cases needs to work for mono or stereo playback data in 1-bit or 1.5-bit data formats.

Record data may be mono or stereo for each data input but is always in the 1-bit format.

The onboard audio hardware contains 6.144MHz PDM modulators for both 1.5-bit and 1-bit use cases; however, a bypass mode is provided to allow for audio software DSP implemented modulator options. As a result, the PDM Tx clock needs to operate at 6.144MHz, as well as integer ratios of this value.

23.8.2 PDM Master Interface Operation in PCM Mode

In PCM mode, the PDM interface receiver may still be needed for external digital microphones. As a result, only the PDM receiver data input 1 (PDM_RX_DATA1) is automatically disabled (primarily as it shares a pin with the PCM data input - PCM_DIN). In PCM mode, both the PDM interface receiver clock output (PDM_RX_CLOCK) and data outputs 2 and 3 (PDM_RX_DATA2/PDM_RX_DATA3) are available to support potential digital microphones. In this use case, the PDM receiver clock (PDM_RX_CLOCK) can be derived either from the same source as the 6.144MHz PDM Tx clock or directly from the high-speed system clock.

23.9 PDM Master Interface Data Encoding Format

The master PDM interface transmitter sends both 1-bit and 1.5-bit PDM data formats, while the PDM receiver only accepts the 1-bit data format.

In 1-bit PDM mode, a single bit of data is transmitted or received for each sample on the selected Tx/Rx channel. In this case, a logic high decodes to +1, and a logic low decodes to -1.

In 1.5-bit PDM mode, two bits of data are transmitted or received for each data sample. The table below shows the 1.5-bit PDM data codes and the decoded values for each case. The invalid code will never be transmitted through the PDM Tx data outputs by the hardware-based modulators but is technically possible if explicitly programmed to occur in the audio DSP software. If the invalid code is received by the PDM Rx inputs, it will be treated as a "0" decode value. While matching the Analog Devices default audio DAC 1.5-bit data format would be convenient; it has potential interface lock-on issues. Therefore, an alternate Analog Devices 1.5-bit encoding scheme is used (see below).

Table 23-7: Analog Devices-Specific 1.5-bit Encoding Scheme

PDM Interface 1.5-bit Data Encoding	11	00	10	01
Decoded Value	-1	0	1	Invalid (0)

23.9.1 PDM Master Interface General Frame Structure

By default, PDM interface Tx/Rx data frames begin on a rising edge, and each clock period contains two bits of data. The PDM data for each frame is received starting with the following falling edge, and a single frame may require one or more clock periods.

Figure 23-7 and *Figure 23-8* illustrate the general frame structure using the PDM receiver.

Figure 23-7: PDM Receiver Frame ($f_{PDMCLK} = f_{S\_PDMDATA}$)

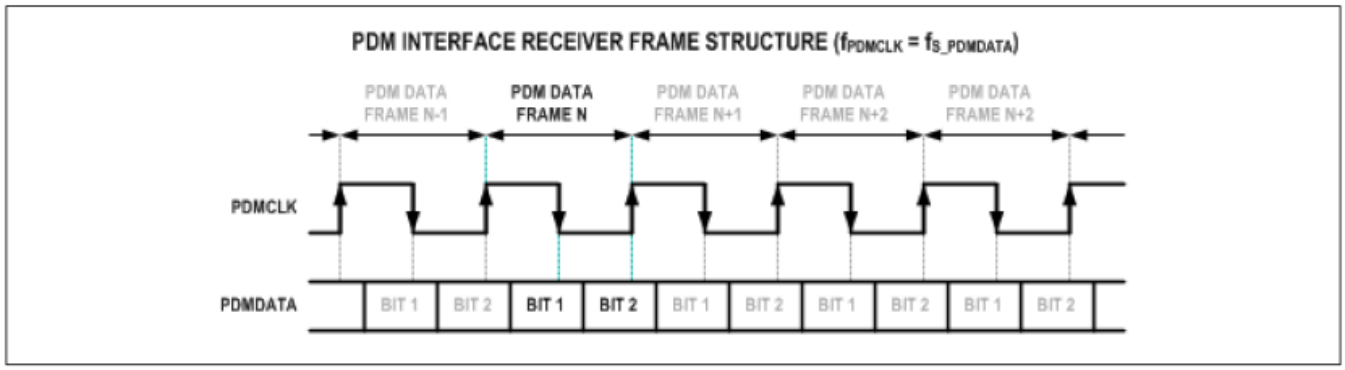
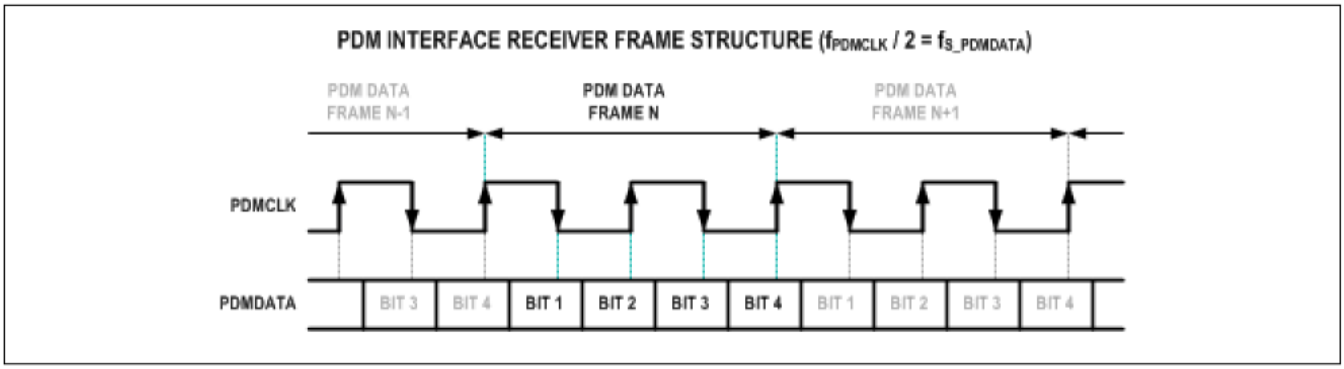


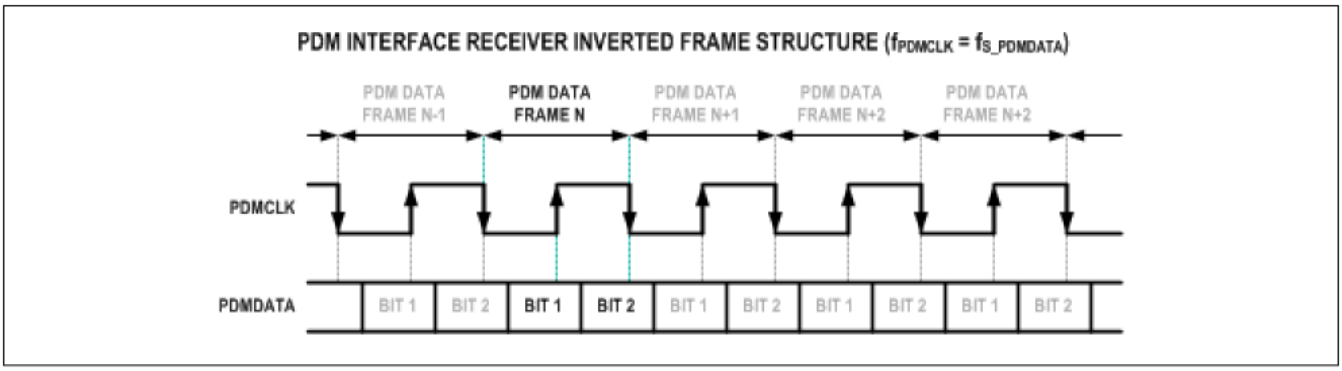
Figure 23-8: PDM Receiver Frame (f_{PDMCLK} / 2 = $f_{S\_PDMDATA}$)



23.9.2 PDM Master Interface Inverted Frame Structure

The PDM interface Tx/Rx can also be configured to accept data frames that begin on a falling edge, as shown in *Figure 23-9*. In this operating mode, the frame structure is identical to the default cases with the single exception that the PDM Tx/Rx Clock is inverted. The example below illustrates this with the PDM master interface receiver.

Figure 23-9: PDM Receiver Inverted Frame (f_{PDMCLK} = $f_{S\_PDMDATA}$)



23.9.3 PDM Master Interface Automatic Frame Alignment and Synchronization (Receiver)

In several supported configurations, the ratio between the PDM clock frequency and PDM data sample rate requires the insertion of extra repeated data to pad out the frame length. The PDM interface transmitter will send the appropriate pattern. However, the PDM interface receiver must lock onto the appropriate patterns.

If the PDM receiver locked blindly onto the first active edge, it might not be the beginning of a frame. Therefore, the PDM interface receiver uses the programmed settings (clock and sample rate), in combination with the resulting PDM frame structure, to lock onto and synchronize with the beginning of each frame.

In 1-bit PDM data mode (Rx only supports 1-bit data format), the PDM receiver must identify the transition between one sample and the next. In this case, to guarantee successful synchronization, the received data channel must contain transitions from -1 ("0" code) to 1 ("1" code) data samples. If the data contains only one code or the other, synchronization is not possible.

23.9.4 PDM Master Interface: All Supported Frame Structures

Frame structures are supported that align with the top-level use cases (1-bit or 1.5-bit data, Mono or Stereo, with clock frequency and data sample rates both at 6.144MHz). Support of a 3.072MHz clock (or lower) and matching / lower data sample rates are helpful but can only be implemented via software in hardware modulator bypass mode.

Figure 23-10: PDM Master Interface Tx/Rx Frame Structure for Mono 1-bit PDM Data (f_{PDMCLK} = 6.144MHz)

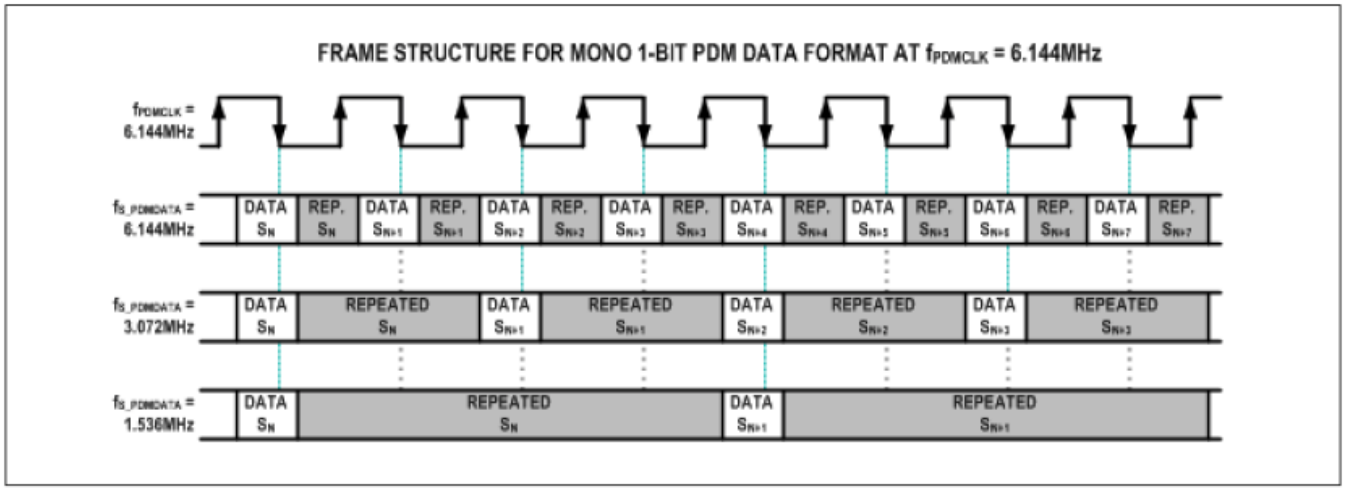


Figure 23-11: PDM Master Interface Tx/Rx Frame Structure for Mono 1-bit PDM Data (f_{PDMCLK} = 3.072MHz)

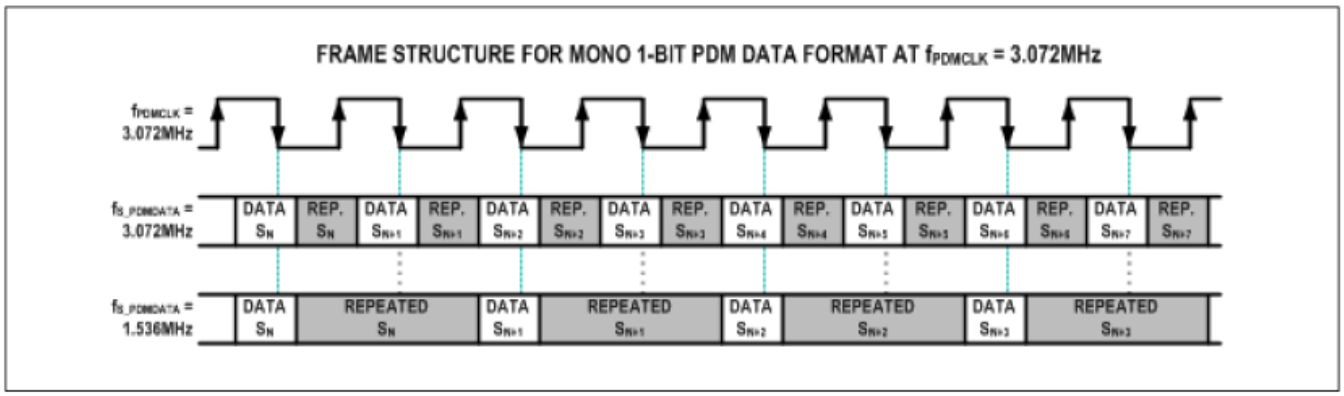


Figure 23-12: PDM Master Interface Tx/Rx Frame Structure for Stereo 1-bit PDM Data (f_{PDMCLK} = 6.144MHz)

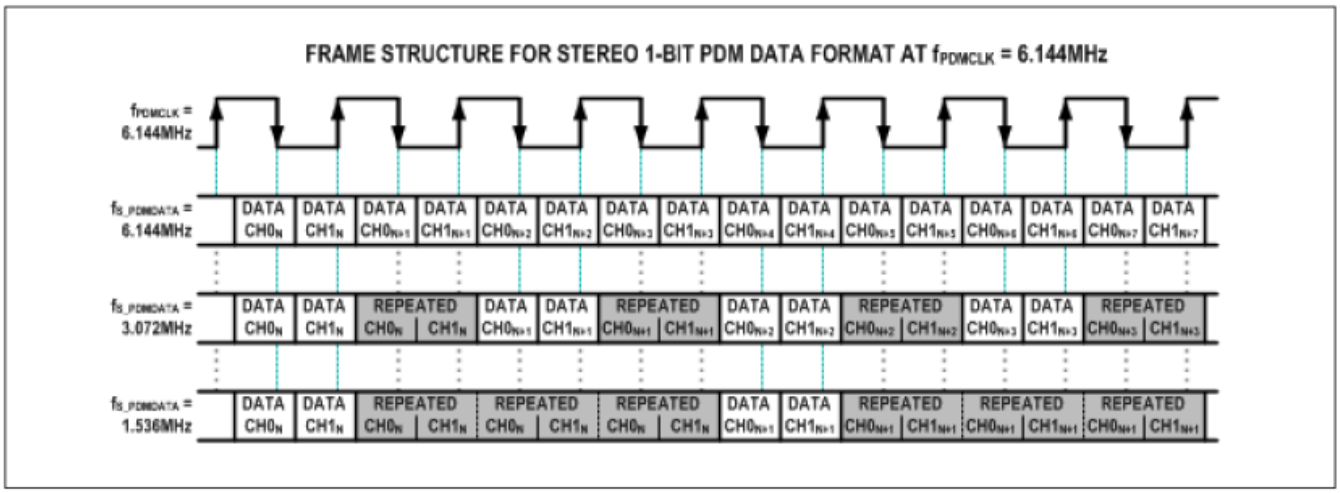


Figure 23-13: PDM Master Interface Tx/Rx Frame Structure for Stereo 1-bit PDM Data (f_{PDMCLK} = 3.072MHz)

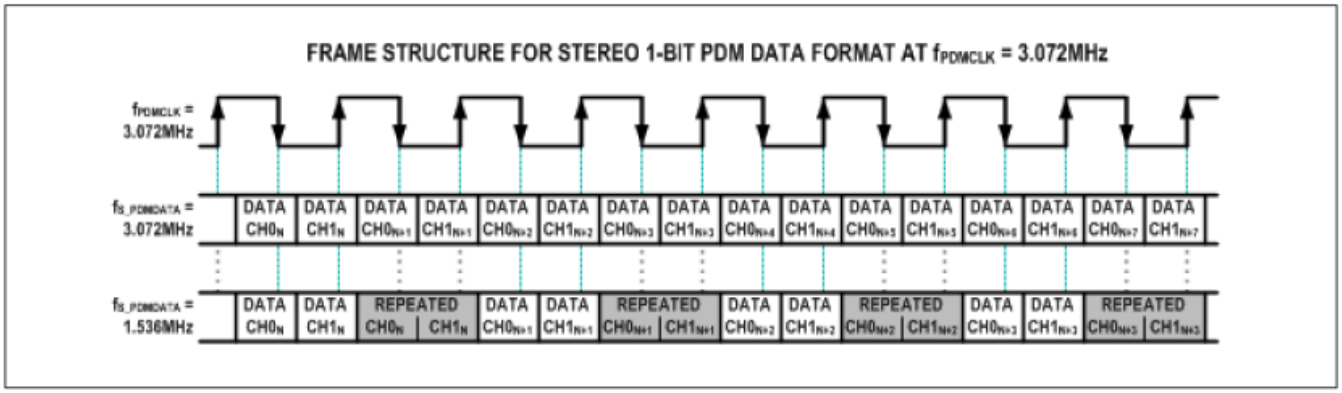


Figure 23-14: PDM Master Interface Tx/Rx Frame Structure for Mono 1.5-bit PDM Data (f_{PDMCLK} = 6.144MHz)

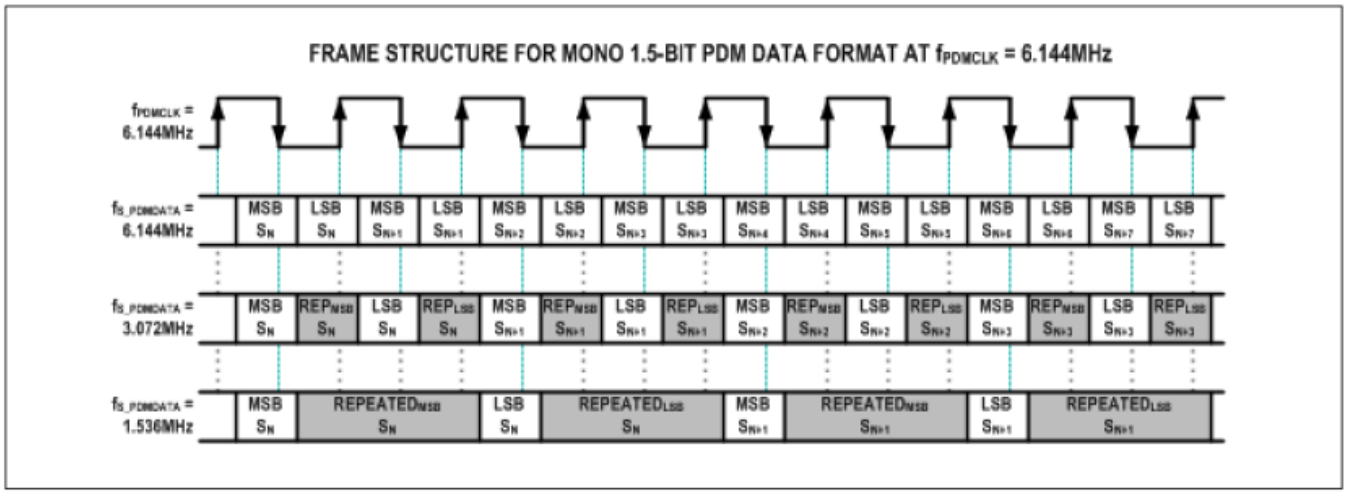


Figure 23-15: PDM Master Interface Tx/Rx Frame Structure for Mono 1.5-bit PDM Data (f_{PDMCLK} = 3.072MHz)

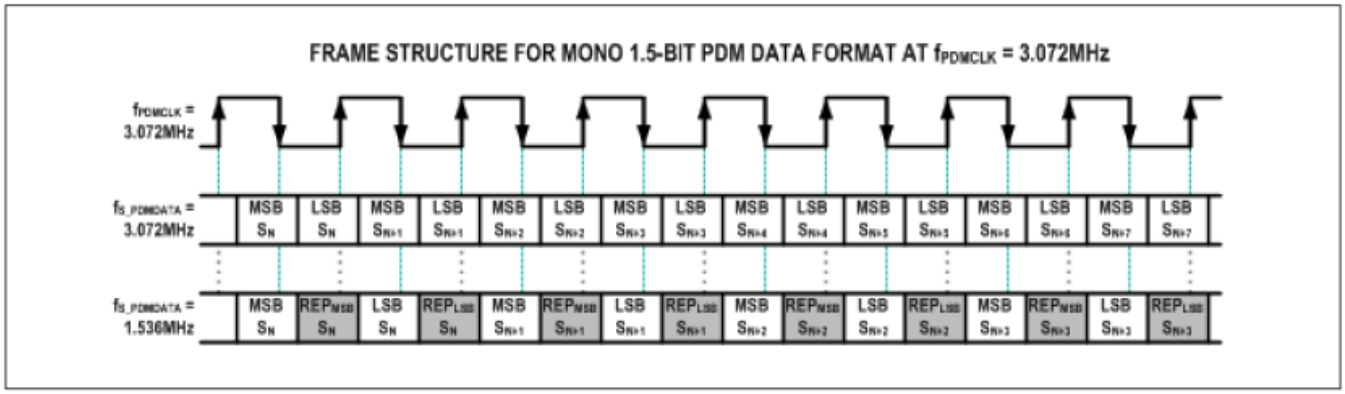


Figure 23-16: PDM Master Interface Tx/Rx Frame Structure for Stereo 1.5-bit PDM Data (f_{PDMCLK} = 6.144MHz)

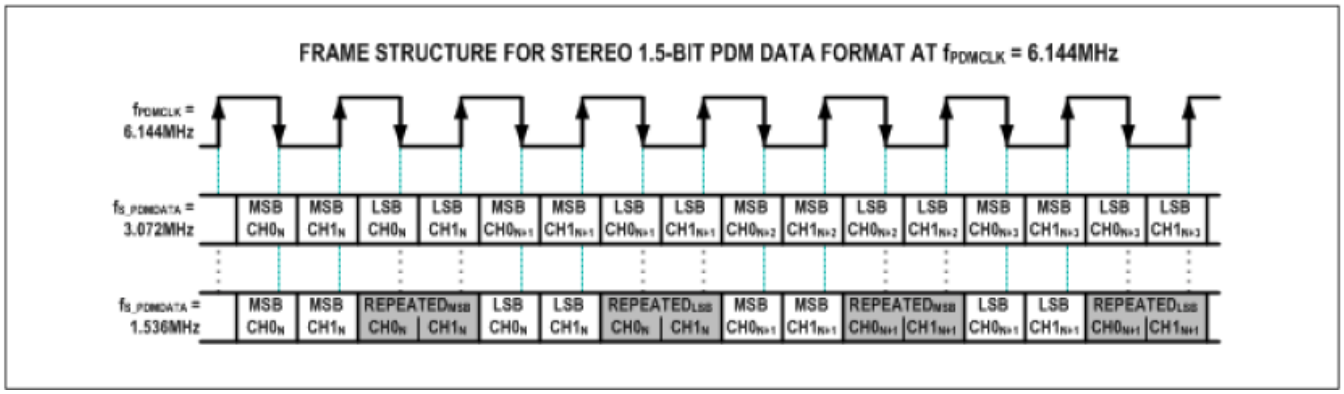
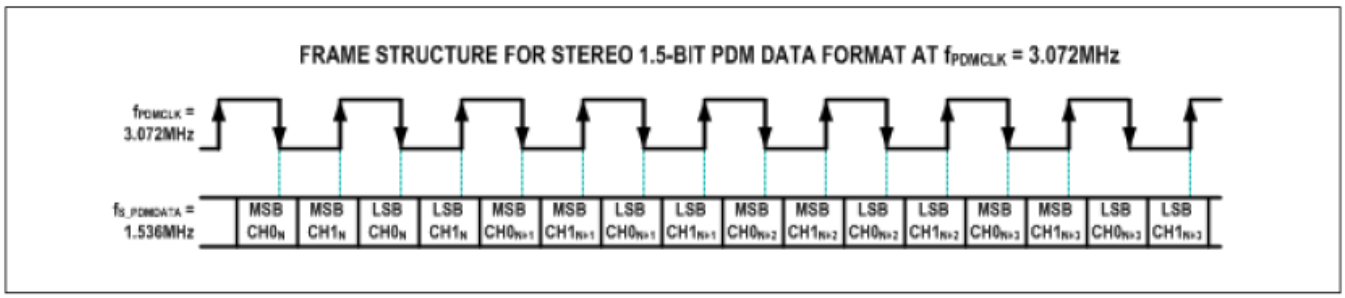


Figure 23-17: PDM Master Interface Tx/Rx Frame Structure for Stereo 1.5-bit PDM Data (f_{PDMCLK} = 3.072MHz)



23.10 Registers

See *Table 3-3* for this peripheral's base address. If multiple instances are provided, each has a unique base address. See *Table 3-1* to explain the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 23-8: Audio Subsystem Register Summary

Offset	Register	Description
[0x0000	*TX_PDM_CH0_ADDR*	PDM Tx FIFO Data Channel 0 Register
[0x0004]	*TX_PDM_CH1_ADDR*	PDM Tx FIFO Data Channel 1 Register
[0x0008]	*RX_PDM_1_CH0_ADDR*	PDM Rx1 FIFO Data Channel 0 Register
[0x000C]	*RX_PDM_1_CH1_ADDR*	PDM Rx1 FIFO Data Channel 1 Register
[0x0010]	*RX_PDM_2_CH0_ADDR*	PDM Rx2 FIFO Data Channel 0 Register
[0x0014]	*RX_PDM_2_CH1_ADDR*	PDM Rx2 FIFO Data Channel 1 Register
[0x0018]	*RX_PDM_3_CH0_ADDR*	PDM Rx3 FIFO Data Channel 0 Register
[0x001C]	*RX_PDM_3_CH1_ADDR*	PDM Rx3 FIFO Data Channel 1 Register
[0x0020]	*TX_PCM_CH0_ADDR*	PCM Tx FIFO Data Channel 0 Register
[0x0024]	*TX_PCM_CH1_ADDR*	PCM Tx FIFO Data Channel 1 Register
[0x0028]	*TX_PCM_CH2_ADDR*	PCM Tx FIFO Data Channel 2 Register
[0x002C]	*TX_PCM_CH3_ADDR*	PCM Tx FIFO Data Channel 3 Register
[0x0030]	*RX_PCM_CH0_ADDR*	PCM Rx FIFO Data Channel 0 Register
[0x0034]	*RX_PCM_CH1_ADDR*	PCM Rx FIFO Data Channel 1 Register
[0x0038]	*RX_PCM_CH2_ADDR*	PCM Rx FIFO Data Channel 2 Register

Offset	Register	Description
[0x003C]	*RX_PCM_CH3_ADDR*	PCM Rx FIFO Data Channel 3 Register
[0x0040]	*RX_PCM_CH4_ADDR*	PCM Rx FIFO Data Channel 4 Register
[0x0044]	*RX_PCM_CH5_ADDR*	PCM Rx FIFO Data Channel 5 Register
[0x0048]	*RX_PCM_CH6_ADDR*	PCM Rx FIFO Data Channel 6 Register
[0x004C]	*RX_PCM_CH7_ADDR*	PCM Rx FIFO Data Channel 7 Register
[0x0050]	*INT_PDM_STATUS*	PDM FIFO Status Register
[0x0054]	*INT_PDM_CLR*	PDM Interrupt Clear Register
[0x0058]	*INT_PCM_TX_STATUS*	PCM Tx FIFO Status Register
[0x005C]	*INT_PCM_TX_CLR*	PCM Tx Interrupt Clear Register
[0x0060]	*INT_PCM_RX_STATUS*	PCM Rx FIFO Status Register
[0x0064]	*INT_PCM_RX_CLR*	PCM Rx Interrupt Clear Register
[0x0068]	*INT_EN*	Interrupt Enable Register
[0x0080]	*M_VAL*	NM Generator Denominator M Register
[0x0088]	*N_VAL*	NM Generator Numerator N Register
[0x0094]	*PDM_TX_RATE_SETUP*	PDM Tx Rate Setup Register
[0x0098]	*MODULATOR_CONTROLS*	Modulator Configuration Register
[0x009C]	*PDM_TX_CONTROL_1*	PDM Tx Configuration 0 Register
[0x00A0]	*PDM_RX_RATE_SETUP*	PDM Rx Clock Configuration Register
[0x00A4]	*PDM_TX_CONTROL_2*	PDM Tx Configuration 1 Register
[0x00A8]	*PDM_RX_1_CONTROL*	PDM Rx1 Configuration Register
[0x00AC]	*PDM_RX_2_CONTROL*	PDM Rx2 Configuration Register
[0x00B0]	*PDM_RX_3_CONTROL*	PDM Rx3 Configuration Register
[0x00B4]	*PCM_CLOCK_DIVIDERS_MSB*	PCM BCLK Configuration 0 Register
[0x00B8]	*PCM_CLOCK_DIVIDERS_LSB*	PCM BCLK Configuration 1 Register
[0x00BC]	*PCM_CLOCK_SET_UP*	PCM Clock Configuration Register
[0x00C0]	*PCM_CONFIG*	PCM Configuration Register
[0x00C4]	*PCM_TX_SAMPLE_RATES*	PCM Tx Sample Rate Configuration Register
[0x00C8]	*PCM_RX_SAMPLE_RATES*	PCM Rx Sample Rate Configuration Register
[0x00CC]	*PCM_RX_ENABLES_BYTE_0*	PCM Rx Channel Enable 0 Register
[0x00D0]	*PCM_RX_ENABLES_BYTE_1*	PCM Rx Channel Enable 1 Register
[0x00D4]	*PCM_TX_ENABLES_BYTE_0*	PCM Tx Channel Enable 0 Register
[0x00D8]	*PCM_TX_ENABLES_BYTE_1*	PCM Tx Channel Enable 1 Register
[0x00DC]	*PCM_TX_HIZ_BYTE_0*	PCM Tx Channel HI-Z Enable 0 Register
[0x00E0]	*PCM_TX_HIZ_BYTE_1*	PCM Tx Channel HI-Z Enable 1 Register
[0x00E4]	*DATAPORT_1_SLOT_MAPPING_CHANNEL_0*	Dataport 1 Slot Mapping Channel 0 Register
[0x00E8]	*DATAPORT_1_SLOT_MAPPING_CHANNEL_1*	Dataport 1 Slot Mapping Channel 1 Register
[0x00EC]	*DATAPORT_1_SLOT_MAPPING_CHANNEL_2*	Dataport 1 Slot Mapping Channel 2 Register
[0x00F0]	*DATAPORT_1_SLOT_MAPPING_CHANNEL_3*	Dataport 1 Slot Mapping Channel 3 Register
[0x00F4]	*DATAPORT_2_SLOT_MAPPING_CHANNEL_0*	Dataport 2 Slot Mapping Channel 0 Register
[0x00F8]	*DATAPORT_2_SLOT_MAPPING_CHANNEL_1*	Dataport 2 Slot Mapping Channel 1 Register

Offset	Register	Description
[0x00FC]	*DATAPORT_2_SLOT_MAPPING_CHANNEL_2*	Dataport 2 Slot Mapping Channel 2 Register
[0x0100]	*DATAPORT_2_SLOT_MAPPING_CHANNEL_3*	Dataport 2 Slot Mapping Channel 3 Register
[0x0104]	*DATAPORT_2_SLOT_MAPPING_CHANNEL_4*	Dataport 2 Slot Mapping Channel 4 Register
[0x0108]	*DATAPORT_2_SLOT_MAPPING_CHANNEL_5*	Dataport 2 Slot Mapping Channel 5 Register
[0x010C]	*DATAPORT_2_SLOT_MAPPING_CHANNEL_6*	Dataport 2 Slot Mapping Channel 6 Register
[0x0110]	*DATAPORT_2_SLOT_MAPPING_CHANNEL_7*	Dataport 2 Slot Mapping Channel 7 Register
[0x0114]	*PCM_DATA_CONTROLS*	PCM Data Controls Register
[0x0118]	*GLOBAL_ENABLE*	Audio Enable Register

23.11 Register Details

Table 23-9: PDM Tx FIFO Data Channel 0 Register

PDM Tx FIFO Data Channel 0			TX_PDM_CH0_ADDR	[0x0000
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	FIFO data for corresponding channel.

Table 23-10: PDM Tx FIFO Data Channel 1 Register

PDM Tx FIFO Data Channel 1			TX_PDM_CH1_ADDR	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	FIFO data for corresponding channel.

Table 23-11: PDM Rx1 FIFO Data Channel 0 Register

PDM Rx1 FIFO Data Channel 0			RX_PDM_1_CH0_ADDR	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-12: PDM Rx1 FIFO Data Channel 1 Register

PDM Rx1 FIFO Data Channel 1			RX_PDM_1_CH1_ADDR	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-13: PDM Rx2 FIFO Data Channel 0 Register

PDM Rx2 FIFO Data Channel 0			RX_PDM_2_CH0_ADDR	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-14: PDM Rx2 FIFO Data Channel 1 Register

PDM Rx2 FIFO Data Channel 1			RX_PDM_2_CH1_ADDR	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-15: PDM Rx3 FIFO Data Channel 0 Register

PDM Rx3 FIFO Data Channel 0			RX_PDM_3_CH0_ADDR	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-16: PDM Rx3 FIFO Data Channel 1 Register

PDM Rx3 FIFO Data Channel 1			RX_PDM_3_CH1_ADDR	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-17: PCM Tx FIFO Data Channel 0 Register

PCM Tx FIFO Data Channel 0			TX_PCM_CH0_ADDR	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-18: PCM Tx FIFO Data Channel 1 Register

PCM Tx FIFO Data Channel 1			TX_PCM_CH1_ADDR	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-19: PCM Tx FIFO Data Channel 2 Register

PCM Tx FIFO Data Channel 2			TX_PCM_CH2_ADDR	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-20: PCM Tx FIFO Data Channel 3 Register

PCM Tx FIFO Data Channel 3			TX_PCM_CH3_ADDR	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-21: PCM Rx FIFO Data Channel 0 Register

PCM Rx FIFO Data Channel 0			RX_PCM_CH0_ADDR	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-22: PCM Rx FIFO Data Channel 1 Register

PCM Rx FIFO Data Channel 1			RX_PCM_CH1_ADDR	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R	0	FIFO data for corresponding channel.

Table 23-23: PCM Rx FIFO Data Channel 2 Register

PCM Rx FIFO Data Channel 2				RX_PCM_CH2_ADDR	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	data	R	0	FIFO data for corresponding channel.	

Table 23-24: PCM Rx FIFO Data Channel 3 Register

PCM Rx FIFO Data Channel 3				RX_PCM_CH3_ADDR	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	data	R	0	FIFO data for corresponding channel.	

Table 23-25: PCM Rx FIFO Data Channel 4 Register

PCM Rx FIFO Data Channel 4				RX_PCM_CH4_ADDR	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	data	R	0	FIFO data for corresponding channel.	

Table 23-26: PCM Rx FIFO Data Channel 5 Register

PCM Rx FIFO Data Channel 5				RX_PCM_CH5_ADDR	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	data	R	0	FIFO data for corresponding channel.	

Table 23-27: PCM Rx FIFO Data Channel 6 Register

PCM Rx FIFO Data Channel 6				RX_PCM_CH6_ADDR	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	data	R	0	FIFO data for corresponding channel.	

Table 23-28: PCM Rx FIFO Data Channel 7 Register

PCM Rx FIFO Data Channel 7				RX_PCM_CH7_ADDR	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	data	R	0	FIFO data for corresponding channel.	

Table 23-29: PDM FIFO Status Register

PDM FIFO Status				INT_PDM_STATUS	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**	
31	af_pdm_tx_ch0	R	0	**PDM Tx FIFO Channel 0 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	

30	af_pdm_tx_ch1	R	0	**PDM Tx FIFO Channel 1 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.
29	af_pdm_rx1_ch0	R	0	**PDM Rx1 FIFO Channel 0 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.
28	af_pdm_rx1_ch1	R	0	**PDM Rx1 FIFO Channel 1 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.
27	af_pdm_rx2_ch0	R	0	**PDM Rx2 FIFO Channel 0 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.
26	af_pdm_rx2_ch1	R	0	**PDM Rx2 FIFO Channel 1 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.
25	af_pdm_rx3_ch0	R	0	**PDM Rx3 FIFO Channel 0 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.
24	af_pdm_rx3_ch1	R	0	**PDM Rx3 Channel 1 FIFO Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.
23	af_pdm_tx_ch0	R	0	**PDM Tx FIFO Channel 0 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
22	af_pdm_tx_ch1	R	0	**PDM Tx FIFO Channel 1 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
21	af_pdm_rx1_ch0	R	0	**PDM Rx1 FIFO Channel 0 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.

20	af_pdm_rx1_ch1	R	0	**PDM Rx1 FIFO Channel 1 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
19	af_pdm_rx2_ch0	R	0	**PDM Rx2 FIFO Channel 0 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
18	af_pdm_rx2_ch1	R	0	**PDM Rx2 FIFO Channel 1 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
17	af_pdm_rx3_ch0	R	0	**PDM Rx3 FIFO Channel 0 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
16	af_pdm_rx3_ch1	R	0	**PDM Rx3 FIFO Channel 1 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
15	ae_pdm_tx_ch0	R	0	**PDM Tx FIFO Channel 0 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
14	ae_pdm_tx_ch1	R	0	**PDM Tx FIFO Channel 1 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
13	ae_pdm_rx1_ch0	R	0	**PDM Rx1 FIFO Channel 0 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
12	ae_pdm_rx1_ch1	R	0	**PDM Rx1 FIFO Channel 1 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
11	ae_pdm_rx2_ch0	R	0	**PDM Rx2 FIFO Channel 0 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.

10	ae_pdm_rx2_ch1	R	0	**PDM Rx2 FIFO Channel 1 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
9	ae_pdm_rx3_ch0	R	0	**PDM Rx3 FIFO Channel 0 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
8	af_pdm_rx3_ch1	R	0	**PDM Rx3 FIFO Channel 1 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
7:0	-	RO	0	**Reserved**

Table 23-30: PDM Interrupt Clear Register

PDM Interrupt Clear				**INT_PDM_CLR**	**[0x0054]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	irq_clear	R/W1C	0	**PDM Interrupt Clear** Setting one or more bits in this field to 1 will immediately clear the interrupt generated by the corresponding FIFO status flag(s). Both the CH0 and CH1 interrupt conditions must be cleared for the given status condition.	

Table 23-31: PCM Tx FIFO Status Register

PCM Tx FIFO Status				**INT_PCM_TX_STATUS**	**[0x0058]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31	af_pcm_tx_ch0	R	0	**PCM Tx FIFO Channel 0 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
30	af_pcm_tx_ch1	R	0	**PCM Tx FIFO Channel 1 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
29	af_pcm_tx_ch2	R	0	**PCM Tx FIFO Channel 2 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
28	af_pcm_tx_ch3	R	0	**PCM Tx FIFO Channel 3 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
27:24	-	RO	0	**Reserved**	

23	hf_pcm_tx_ch0	R	0	**PCM Tx FIFO Channel 0 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
22	hf_pcm_tx_ch1	R	0	**PCM Tx FIFO Channel 1 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
21	hf_pcm_tx_ch2	R	0	**PCM Tx FIFO Channel 2 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
20	hf_pcm_tx_ch3	R	0	**PCM Tx FIFO Channel 3 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
19:16	-	RO	0	**Reserved**
15	ae_pcm_tx_ch0	R	0	**PCM Tx FIFO Channel 0 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
14	ae_pcm_tx_ch1	R	0	**PCM Tx FIFO Channel 1 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
13	ae_pcm_tx_ch2	R	0	**PCM Tx FIFO Channel 2 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
12	ae_pcm_tx_ch3	R	0	**PCM Tx FIFO Channel 3 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
11:0	-	RO	0	**Reserved**

Table 23-32: PCM Tx Interrupt Clear Register

PCM Tx Interrupt Clear				**INT_PCM_TX_CLR**	**[0x005C]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	irq_clear	R/W1C	0	Setting one or more bits in this field to 1 will immediately clear the interrupt generated by the corresponding FIFO status flag(s). Both the CH0 and CH1 interrupt conditions must be cleared for the given status condition.	

Table 23-33: PCM RX FIFO Status Register

PCM RX FIFO Status				INT_PCM_RX_STATUS	[0x0060]
Bits	**Field**	**Access**	**Reset**	**Description**	
31	af_pcm_rx_ch0	R	0	**PCM Rx FIFO Channel 0 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
30	af_pcm_rx_ch	R	0	**PCM Rx FIFO Channel 1 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
29	af_pcm_rx_ch2	R	0	**PCM Rx FIFO Channel 2 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
28	af_pcm_rx_ch3	R	0	**PCM Rx FIFO Channel 3 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
27	af_pcm_rx_ch4	R	0	**PCM Rx FIFO Channel 4 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
26	af_pcm_rx_ch5	R	0	**PCM Rx FIFO Channel 5 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
25	af_pcm_rx_ch6	R	0	**PCM Rx FIFO Channel 6 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
24	af_pcm_rx_ch7	R	0	**PCM Rx FIFO Channel 7 Almost Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, or 2 samples. 1: FIFO contains 3 or 4 samples.	
23	hf_pcm_rx_ch0	R	0	**PCM Rx FIFO Channel 0 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.	

22	hf_pcm_rx_ch1	R	0	**PCM Rx FIFO Channel 1 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
21	hf_pcm_rx_ch2	R	0	**PCM Rx FIFO Channel 2 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
20	hf_pcm_rx_ch3	R	0	**PCM Rx FIFO Channel 3 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
19	hf_pcm_rx_ch4	R	0	**PCM Rx FIFO Channel 4 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
18	hf_pcm_rx_ch5	R	0	**PCM Rx FIFO Channel 5 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
17	hf_pcm_rx_ch6	R	0	**PCM Rx FIFO Channel 6 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
16	hf_pcm_rx_ch7	R	0	**PCM Rx FIFO Channel 7 Half Full** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.0: FIFO contains 0, 1, 3, or 4 samples. 1: FIFO contains 2 samples.
15	ae_pcm_rx_ch0	R	0	**PCM Rx FIFO Channel 0 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 samples
14	ae_pcm_rx_ch1	R	0	**PCM Rx FIFO Channel 1 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
13	ae_pcm_rx_ch2	R	0	**PCM Rx FIFO Channel 2 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.

12	ae_pcm_rx_ch3	R	0	**PCM Rx FIFO Channel 3 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
11	ae_pcm_rx_ch4	R	0	**PCM Rx FIFO Channel 4 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
10	ae_pcm_rx_ch5	R	0	**PCM Rx FIFO Channel 5 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
9	ae_pcm_rx_ch6	R	0	**PCM Rx FIFO Channel 6 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
8	ae_pcm_rx_ch7	R	0	**PCM Rx FIFO Channel 7 Almost Empty** This field indicates the number of samples in the corresponding FIFO. The CH0 and CH1 channel flags are updated simultaneously and are always the same value. 0: FIFO contains 2, 3, or 4 samples. 1: FIFO contains 0 or 1 sample.
7:0	-	RO	0	**Reserved**

Table 23-34: PCM RX Interrupt Clear Register

PCM RX Interrupt Clear				**INT_PCM_RX_CLR**	**[0x0064]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	irq_clear	R/W1C	0	Setting one or more bits in this field to 1 will immediately clear the interrupt generated by the corresponding FIFO status flag(s). Both the CH0 and CH1 interrupt conditions must be cleared for the given status condition.	

Table 23-35: FIFO Interrupt Enable Register

FIFO Interrupt Enable				**INT_EN**	**[0x0068]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31	en_af_pdm_tx	R/W	0	**PDM Tx FIFOs Almost Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost-full flags from 0 to 1.	
30	en_af_pdm_rx1	R/W	0	**PDM Rx1 FIFOs Almost Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost-full flags from 0 to 1.	
29	en_af_pdm_rx2	R/W	0	**PDM Rx2 FIFOs Almost Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost-full flags from 0 to 1.	

28	en_af_pdm_rx3	R/W	0	**PDM Rx3 FIFOs Almost Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost-full flags from 0 to 1.
27	en_af_pcm_tx	R/W	0	**PCM Tx FIFOs Almost Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost-full flags from 0 to 1.
26	en_af_pcm_rx	R/W	0	**PCM Rx FIFOs Almost Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost-full flags from 0 to 1.
25:24	-	RO	0	**Reserved**
23	en_hf_pdm_tx	R/W	0	**PDM Tx Channel FIFOs Half Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding half-full flags from 0 to 1.
22	en_hf_pdm_rx1	R/W	0	**PDM Rx1 Channel FIFOs Half Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding half-full flags from 0 to 1.
21	en_hf_pdm_rx2	R/W	0	**PDM Rx2 Channel FIFOs Half Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding half-full flags from 0 to 1.
20	en_hf_pdm_rx3	R/W	0	**PDM Rx3 Channel FIFOs Half Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding half-full flags from 0 to 1.
19	en_hf_pcm_tx	R/W	0	**PCM Tx Channel FIFOs Half Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding half-full flags from 0 to 1.
18	en_hf_pcm_rx	R/W	0	**PCM Rx Channel FIFOs Half Full Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding half-full flags from 0 to 1.
17:16	-	RO	0	**Reserved**
15	en_ae_pdm_tx	R/W	0	**PDM Tx Channel FIFOs Almost Empty Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost empty flags from 0 to 1.
14	en_ae_pdm_rx1	R/W	0	**PDM Rx1 Channel FIFO Almost Empty Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost empty flags from 0 to 1.
13	en_ae_pdm_rx2	R/W	0	**PDM Rx2 Channel FIFO Almost Empty Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost empty flags from 0 to 1.

12	en_ae_pdm_rx3	R/W	0	**PDM Rx3 Channel FIFO Almost Empty Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost empty flags from 0 to 1.
11	en_ae_pcm_tx	R/W	0	**PCM Tx Channel FIFOs Almost Empty Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost empty flags from 0 to 1.
10	en_ae_pcm_rx	R/W	0	**PCM Rx Channel FIFOs Almost Empty Interrupt Enable** 0: Disabled. 1: Interrupt will be asserted when hardware sets the corresponding almost empty flags from 0 to 1.
9:0	-	RO	0	**Reserved**

Table 23-36: NM Generator Denominator M Register

NM Generator Denominator M				**M_VAL**	**[0x0080]**
Bits	**Field**	**Access**	**Reset**	**Description**	
15:0	value	R/W	0x0C35	This value is the M_VALUE	

Table 23-37: NM Generator Denominator N Register

NM Generator Numerator N				**N_VAL**	**[0x0088]**
Bits	**Field**	**Access**	**Reset**	**Description**	
14:0	value	R/W	0x0180	This value is the N_VALUE	

Table 23-38: PDM Tx Rate Setup Register

PDM Tx Rate Setup				**PDM_TX_RATE_SETUP**	**[0x0094]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:4	-	RO	0	**Reserved**	
3:2	pdm_tx_data_sr	R/W	0b01	**PDM Tx Data Sample Rate Divisor** Allows the data sample rate to be the same as or lower than the PDM Tx clock rate. 00: PDM_TX_CLK / 8 01: PDM_TX_CLK / 4 10: PDM_TX_CLK / 2 11: PDM_TX_CLK	
1:0	pdm_tx_clk_rate	R/W	0b10	**PDM Transmit Clock Rate** 00: MCLK / 8 01: MCLK / 4 10: MCLK / 2 11: MCLK	

Table 23-39: Modulator Configuration Register

Modulator Configuration				**MODULATOR_CONTROLS**	**[0x0098]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:8	-	RO	0	**Reserved**	

7	pdm_pcm_select	R/W	0	**PCM Format Select** 0: PDM 1: PCM
6:4	-	RO	0	**Reserved**
3	pdm_tx_alt_encode	R/W	0	**PDM Alternate Encoding Enable** 0: Normal encoding: +1 = 0b01, 0 = 0b00, -1 = 0b11 1: Alternate encoding:+1 = 0b11, 0 = 0b01, -1 = 0b00
2	tx_mod_bypass	R/W	0	**Tx Modulator Bypass** 0: Use the audio subsystem modulator. 1: Use a software-implemented modulator.
1	tx_mono_mode	R/W	0	**Tx Mono Mode** 0: Disabled 1: Enabled
0	tx_mod_enable	R/W	1	**Tx Modulator Enable** 0: Disabled 1: Enabled

Table 23-40: PDM Tx Configuration 0 Register

PDM Tx Configuration 0				**PDM_TX_CONTROL_1**	**[0x009C]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:8	-	RO	0	**Reserved**	
7	pdm_tx_2_ch1_hiz	R/W	0	**PDM Tx2 Hi-Z Enable Channel Slot 1** This field controls the function of the Tx output driver during the corresponding PDM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding frame slot driven active low when the channel is disabled. 1: Corresponding frame slot is in high-impedance mode when the channel is disabled.	
6	pdm_tx_2_ch0_hiz	R/W	0	**PDM Tx2 Hi-Z Enable Channel Slot 0** This field controls the function of the Tx output driver during the corresponding PDM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding frame slot driven active low when the channel is disabled. 1: Corresponding frame slot is in high-impedance mode when the channel is disabled.	
5	pdm_tx_2_ch1_en	R/W	0	**PDM Tx2 Channel 1 Enable** 0: Disabled 1: Enabled This field is ignored if PDM_Tx_MONO_MODE == 1.	
4	pdm_tx_2_ch0_en	R/W	0	**PDM Tx2 Channel 0 Enable** 0: Disabled 1: Enabled	
3	pdm_tx_1_ch1_hiz	R/W	0	**PDM Tx Hi-Z Slot CH1 Enable** This field controls the function of the Tx output driver during the corresponding PDM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding frame slot driven active low when the channel is disabled. 1: Corresponding frame slot is in high-impedance mode when the channel is disabled.	

2	pdm_tx_1_ch0_hiz	R/W	0	**PDM Tx1 Hi-Z Slot CH0 Enable** This field controls the function of the Tx output driver during the corresponding PDM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding frame slot driven active low when the channel is disabled. 1: Corresponding frame slot is in high-impedance mode when the channel is disabled.
1	pdm_tx_1_ch1_en	R/W	0	**PDM Tx1 Channel 1 Enable** 0: Disabled 1: Enabled This field is ignored if PDM_TX_MONO_MODE == 1.
0	pdm_tx_1_ch0_en	R/W	0	**PDM Tx1 Channel 0 Enable** 0: Disabled 1: Enabled

Table 23-41: PDM RX Clock Configuration Register

PDM RX Clock Configuration			**PDM_RX_RATE_SETUP**	**[0x00A0]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:6	pdm_rx_3_data_sr	R/W	0b01	**Data Sample Rate Division for PDM Rx3 Interface** Allows the data sample rate to be the same as or lower than the PDM RX clock rate. 00: PDM_RX_CLK / 8 01: PDM_RX_CLK / 4 10: PDM_RX_CLK / 2 11: PDM_RX_CLK
5:4	pdm_rx_2_data_sr	R/W	0b01	**Data Sample Rate Division for PDM Rx2 Interface** Allows the data sample rate to be the same as or lower than the PDM RX clock rate. 00: PDM_RX_CLK / 8 01: PDM_RX_CLK / 4 10: PDM_RX_CLK / 2 11: PDM_RX_CLK
3:2	pdm_rx_1_data_sr	R/W	0b01	**Data Sample Rate Division for PDM Rx1 Interface** Allows the data sample rate to be the same as or lower than the PDM RX clock rate. 00: PDM_RX_CLK / 8 01: PDM_RX_CLK / 4 10: PDM_RX_CLK / 2 11: PDM_RX_CLK
1:0	pdm_rx_clk_rate	R/W	0b10	**Data Sample Rate Division for PDM Rx Interface** 00: MCLK / 8 01: MCLK / 4 10: MCLK / 2 11: MCLK / 1

Table 23-42: PDM Tx Configuration 1 Register

PDM Tx Configuration 1			**PDM_TX_CONTROL_2**	**[0x00A4]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**

1	pdm_tx_edge_sel	R/W	0	**1.5 Bit First Edge Detect PDM Tx** 0: Falling 1: Rising
0	pdm_tx_mode	R/W	1	**PDM Tx Data Length** 0: 1 bit data 1: 1.5 bit data

Table 23-43: PDM RX1 Configuration Register

PDM RX1 Configuration				**PDM_RX_1_CONTROL**	**[0x00A8]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:4	-	RO	0	**Reserved**	
3	pdm_rx_1_edge_sel	R/W	0	**1.5 Bit First Edge Detect PDM Rx1** 0: Falling 1: Rising	
2	pdm_rx_1_mode	R/W	1	**PDM Rx1 Data Length** 0: 1 bit data 1: 1.5 bit data	
1	pdm_rx_1_ch1_en	R/W	0	**PDM Rx3 Channel 1 Enable** 0: Disabled 1: Enabled	
0	pdm_rx_1_ch0_en	R/W	0	**PDM Rx1 Channel 0 Enable** 0: Disabled 1: Enabled	

Table 23-44: PDM RX2 Configuration Register

PDM RX2 Configuration				**PDM_RX_2_CONTROL**	**[0x00AC]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:4	-	RO	0	**Reserved**	
3	pdm_rx_2_edge_sel	R/W	0	**1.5 Bit First Edge Detect PDM Rx2** 0: Falling 1: Rising	
2	pdm_rx_2_mode	R/W	1	**PDM Rx3 Data Length** 0: 1 bit data 1: 1.5 bit data	
1	pdm_rx_2_ch1_en	R/W	0	**PDM Rx2 Channel 1 Enable** 0: Disabled 1: Enabled	
0	pdm_rx_2_ch0_en	R/W	0	**PDM Rx2 Channel 0 Enable** 0: Disabled 1: Enabled	

Table 23-45: PDM RX3 Configuration Register

PDM RX3 Configuration				**PDM_RX_3_CONTROL**	**[0x00B0]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:4	-	RO	0	**Reserved**	

3	pdm_rx_3_edge_sel	R/W	0	**1.5 Bit First Edge Detect PDM Rx3** 0: Falling 1: Rising "Select the first edge for 1.5-bit data"
2	pdm_rx_3_mode	R/W	1	**PDM Rx3 Data Length** 0: 1 bit data 1: 1.5 bit data
1	pdm_rx_3_ch1_en	R/W	0	**PDM Rx3 Channel 1 Enable** 0: Disabled 1: Enabled
0	pdm_rx_3_ch0_en	R/W	0	**PDM Rx3 Channel 0 Enable** 0: Disabled 1: Enabled

Table 23-46: PCM BCLK Configuration 0 Register

PCM BCLK Configuration 0			**PCM_CLOCK_DIVIDERS_MSB**	**[0x00B4]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	bclk_master_sel	R/W	1	**BCLK Clock Source** 0: f_{AUDIO_CLK} 1: f_{AUDIO_CLK} * (N_VAL / M_VAL)
6:5	bclk_sel	R/W	0b01	**BCLK Source Select** 00: BCLK generator, tick out 01: BCLK generator, toggle out 10: BCLK master 11: Reserved
4:2	-	RO	0	**Reserved**
1:0	bclk_div_msb	R/W	0b00	**BCLK Divisor MSb** Bits [10:11] of BCLK divisor.

Table 23-47: PCM BCLK Configuration 1 Register

PCM BCLK Configuration 1			**PCM_CLOCK_DIVIDERS_LSB**	**[0x00B8]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9:0	bclk_div_lsb	R/W	0x001	**BCLK Divisor LSb** Bits [9:0] of BCLK divisor.

Table 23-48: PCM Clock Configuration Register

PCM Clock Configuration			**PCM_CLOCK_SET_UP**	**[0x00BC]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	pcm_bclkedge	R/W	0	**BCLK Polarity** 0: Data transmitted falling edge, received rising edge. 1: Data transmitted rising edge, received falling edge.

3:0	pcm_bsel	R/W	0	**LRCLK Divider** Defines the number of BCLK pulses per LRCLK 0x0: Reserved 0x1: Reserved 0x2: 32 0x3: 48 0x4: 64 0x5: 96 0x6: 128 0x7: 192 0x8: 256 0x9: 384 0xA: 512 0xB: 320 Other: Reserved

Table 23-49: PCM Configuration Register

PCM Configuration			**PCM_CONFIG**	**[0x00C0]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:6	pcm_chansz	R/W	0	**PCM Channel Size** 00: Reserved 01: 16-bit 10: 24-bit 11: 32-bit
5:3	pcm_formatx	R/W	0	**PCM Format** 0x0: I^2S 0x1: PCM justified 0x2: PCM right justified 0x3: TDM mode 1 0x4: TDM mode 2 0x5: TDM mode 3
2	pcm_chansel	R/W	0	**LRCLK Polarity** 0: The start of the frame is the rising edge in TDM mode, falling edge in I^2S mode. 1: The start of the frame is falling edge in TDM mode, rising edge in I^2S mode.
0	pcm_tx_extra_hiz	R/W	0	**PCM Tx Extra Bits Format** 0: Extra bits at the end of a PCM transmission will be transmitted as 0. 1: Extra bits at the end of a PCM transmission will be transmitted as high impedance.

Table 23-50: PCM Tx Sample Rate Configuration Register

PCM Tx Sample Rate Configuration			**PCM_TX_SAMPLE_RATES**	**[0x00C4]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**

7:4	pcm_tx_dataport_sr	R/W	0x8	**PCM Tx Sample Rate** The sample rate must be equal to or less than the value for PCM_TX_INTERFACE_SR. If less than PCM_TX_INTERFACE_SR, then each output data sample will be repeated N times where N = PCM_TX_INTERFACE_SR / PCM_TX_DATAPORT_SR. 0x0: 8kHz 0x1: 11.025kHz 0x2: 12kHz 0x3: 16kHz 0x4: 22.05kHz 0x5: 24kHz 0x6: 32kHz 0x7: 44.1kHz 0x8: 48kHz 0x9: 88.2kHz 0xA: 96kHz 0xB: 176.4kHz 0xC: 192kHz 0xD: Reserved 0xE: Reserved 0xF: Reserved
3:0	pcm_tx_interface_sr	R/W	0x~~0~~8	**PCM Tx Interface Sample Rate** This is the rate of LRCLK. 0x0: 8kHz 0x1: 11.025kHz 0x2: 12kHz 0x3: 16kHz 0x4: 22.05kHz 0x5: 24kHz 0x6: 32kHz 0x7: 44.1kHz 0x8: 48kHz 0x9: 88.2kHz 0xA: 96kHz 0xB: 176.4kHz 0xC: 192kHz 0xD: Reserved 0xE: Reserved 0xF: Reserved

Table 23-51: PCM RX Sample Rate Configuration Register

PCM RX Sample Rate Configuration			**PCM_RX_SAMPLE_RATES**	**[0x00C8]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**

7:4	pcm_rx_dataport_sr	R/W	0x8	**PCM Rx Sample Rate** It can be the same or less than the value for PCM_RX_INTERFACE_SR. If less than PCM_RX_INTERFACE_SR, then only 1 sample in N will be received, and the other (N-1) samples will be ignored. N = PCM_RX_INTERFACE_SR / PCM_RX_DATAPORT_SR. 0x0: 8kHz 0x1: 11.025kHz 0x2: 12kHz 0x3: 16kHz 0x4: 22.05kHz 0x5: 24kHz 0x6: 32kHz 0x7: 44.1kHz 0x8: 48kHz 0x9: 88.2kHz 0xA: 96kHz 0xB: 176.4kHz 0xC: 192kHz 0xD: Reserved 0xE: Reserved 0xF: Reserved
3:0	pcm_rx_interface_sr	R/W	0x8	**PCM Rx Interface Sample Rate** This is the rate of LRCLK. 0x0: 8kHz 0x1: 11.025kHz 0x2: 12kHz 0x3: 16kHz 0x4: 22.05kHz 0x5: 24kHz 0x6: 32kHz 0x7: 44.1kHz 0x8: 48kHz 0x9: 88.2kHz 0xA: 96kHz 0xB: 176.4kHz 0xC: 192kHz 0xD: Reserved 0xE: Reserved 0xF: Reserved

Table 23-52: PCM RX Channel Enable 0 Register

PCM RX Channel Enable 0				**PCM_RX_ENABLES_BYTE_0**	**[0x00CC]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:8	-	RO	0	**Reserved**	
7	pcm_rx_ch7_en	R/W	0	**PCM Rx Channel 7 Enable** 0: Disabled 1: Enabled	
6	pcm_rx_ch6_en	R/W	0	**PCM Rx Channel 6 Enable** 0: Disabled 1: Enabled	

5	pcm_rx_ch5_en	R/W	0	**PCM Rx Channel 5 Enable** 0: Disabled 1: Enabled
4	pcm_rx_ch4_en	R/W	0	**PCM Rx Channel 4 Enable** 0: Disabled 1: Enabled
3	pcm_rx_ch3_en	R/W	0	**PCM Rx Channel 3 Enable** 0: Disabled 1: Enabled
2	pcm_rx_ch2_en	R/W	0	**PCM Rx Channel 2 Enable** 0: Disabled 1: Enabled
1	pcm_rx_ch1_en	R/W	0	**PCM Rx Channel 1 Enable** 0: Disabled 1: Enabled
0	pcm_rx_ch0_en	R/W	0	**PCM Rx Channel 0 Enable** 0: Disabled 1: Enabled

Table 23-53: PCM RX Channel Enable 1 Register

PCM RX Channel Enable 1			**PCM_RX_ENABLES_BYTE_1**	**[0x00D0]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_rx_ch15_en	R/W	0	**PCM Rx Channel 15 Enable** 0: Disabled 1: Enabled
6	pcm_rx_ch14_en	R/W	0	**PCM Rx Channel 14 Enable** 0: Disabled 1: Enabled
5	pcm_rx_ch13_en	R/W	0	**PCM Rx Channel 13 Enable** 0: Disabled 1: Enabled
4	pcm_rx_ch12_en	R/W	0	**PCM Rx Channel 12 Enable** 0: Disabled 1: Enabled
3	pcm_rx_ch11_en	R/W	0	**PCM Rx Channel 11 Enable** 0: Disabled 1: Enabled
2	pcm_rx_ch10_en	R/W	0	**PCM Rx Channel 10 Enable** 0: Disabled 1: Enabled
1	pcm_rx_ch9_en	R/W	0	**PCM Rx Channel 9 Enable** 0: Disabled 1: Enabled
0	pcm_rx_ch8_en	R/W	0	**PCM Rx Channel 8 Enable** 0: Disabled 1: Enabled

Table 23-54: PCM Tx Channel Enable 0 Register

PCM Tx Channel Enable 0				PCM_TX_ENABLES_BYTE_0	[0x00D4]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:8	-	RO	0	**Reserved**	
7	pcm_tx_ch7_en	R/W	0	**PCM Tx Channel 7 Enable** 0: Disabled 1: Enabled	
6	pcm_tx_ch6_en	R/W	0	**PCM Tx Channel 6 Enable** 0: Disabled 1: Enabled	
5	pcm_tx_ch5_en	R/W	0	**PCM Tx Channel 5 Enable** 0: Disabled 1: Enabled	
4	pcm_tx_ch4_en	R/W	0	**PCM Tx Channel 4 Enable** 0: Disabled 1: Enabled	
3	pcm_tx_ch3_en	R/W	0	**PCM Tx Channel 3 Enable** 0: Disabled 1: Enabled	
2	pcm_tx_ch2_en	R/W	0	**PCM Tx Channel 2 Enable** 0: Disabled 1: Enabled	
1	pcm_tx_ch1_en	R/W	0	**PCM Tx Channel 1 Enable** 0: Disabled 1: Enabled	
0	pcm_tx_ch0_en	R/W	0	**PCM Tx Channel 0 Enable** 0: Disabled 1: Enabled	

Table 23-55: PCM Tx Channel Enable 1 Register

PCM Tx Channel Enable 1				PCM_TX_ENABLES_BYTE_1	[0x00D8]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:8	-	RO	0	**Reserved**	
7	pcm_tx_ch15_en	R/W	0	**PCM Tx Channel 15 Enable** 0: Disabled 1: Enabled	
6	pcm_tx_ch14_en	R/W	0	**PCM Tx Channel 14 Enable** 0: Disabled 1: Enabled	
5	pcm_tx_ch13_en	R/W	0	**PCM Tx Channel 13 Enable** 0: Disabled 1: Enabled	
4	pcm_tx_ch12_en	R/W	0	**PCM Tx Channel 12 Enable** 0: Disabled 1: Enabled	

3	pcm_tx_ch11_en	R/W	0	**PCM Tx Channel 11 Enable** 0: Disabled 1: Enabled
2	pcm_tx_ch10_en	R/W	0	**PCM Tx Channel 10 Enable** 0: Disabled 1: Enabled
1	pcm_tx_ch9_en	R/W	0	**PCM Tx Channel 9 Enable** 0: Disabled 1: Enabled
0	pcm_tx_ch8_en	R/W	0	**PCM Tx Channel 8 Enable** 0: Disabled 1: Enabled

Table 23-56: PCM Tx Channel HI-Z Enable 0 Register

PCM Tx Channel HI-Z Enable 0			**PCM_TX_HIZ_BYTE_0**	**[0x00DC]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_tx_ch7_hiz	R/W	0	**PCM Tx Hi-Z Slot CH7 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
6	pcm_tx_ch6_hiz	R/W	0	**PCM Tx Hi-Z Slot CH6 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
5	pcm_tx_ch5_hiz	R/W	0	**PCM Tx Hi-Z Slot CH5 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
4	pcm_tx_ch4_hiz	R/W	0	**PCM Tx Hi-Z Slot CH4 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.

3	pcm_tx_ch3_hiz	R/W	0	**PCM Tx Hi-Z Slot CH3 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
2	pcm_tx_ch2_hiz	R/W	0	**PCM Tx Hi-Z Slot CH2 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
1	pcm_tx_ch1_hiz	R/W	0	**PCM Tx Hi-Z Slot CH1 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
0	pcm_tx_ch0_hiz	R/W	0	**PCM Tx Hi-Z Slot CH0 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.

Table 23-57: PCM Tx Channel HI-Z Enable 1 Register

PCM Tx Channel HI-Z Enable 1			**PCM_TX_HIZ_BYTE_1**	**[0x00E0]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_tx_ch15_hiz	R/W	0	**PCM Tx Hi-Z Slot CH15 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
6	pcm_tx_ch014_hiz	R/W	0	**PCM Tx Hi-Z Slot CH14 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.

5	pcm_tx_ch13_hiz	R/W	0	**PCM Tx Hi-Z Slot CH13 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
4	pcm_tx_ch12_hiz	R/W	0	**PCM Tx Hi-Z Slot CH12 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
3	pcm_tx_ch11_hiz	R/W	0	**PCM Tx Hi-Z Slot CH11 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
2	pcm_tx_ch10_hiz	R/W	0	**PCM Tx Hi-Z Slot CH10 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
1	pcm_tx_ch9_hiz	R/W	0	**PCM Tx Hi-Z Slot CH9 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.
0	pcm_tx_ch8_hiz	R/W	0	**PCM Tx Hi-Z Slot CH8 Enable** This field controls the function of the Tx output driver during the corresponding PCM frame slot when the channel is disabled. It has no effect on an enabled channel. 0: Corresponding PCM frame slot driven active low when the channel is disabled. 1: Corresponding PCM frame slot is in high-impedance mode when the channel is disabled.

Table 23-58: Dataport 1 Slot Mapping Channel 0 Register

Dataport 1 Slot Mapping Channel 0			**DATAPORT_1_SLOT_MAPPING_CHANNEL_0**	**[0x00E4]**
Bits	**Field**	**Access**	**Reset**	**Description**

31:8	-	RO	0	**Reserved**
7	pcm_dport1_ch0_en	R/W	0	**PCM Transmit Dataport 1 Channel 0 Enable** This enables the PCM TRANSMIT dataport channel 0, which is connected to APB Channel TX_PCM_CH0 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**
3:0	dport1_slot_map_ch0	R/W	0	**PCM Dataport 1 Channel 0 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-59: Dataport 1 Slot Mapping Channel 1 Register

Dataport 1 Slot Mapping Channel 1			DATAPORT_1_SLOT_MAPPING_CHANNEL_1	[0x00E8]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_dport1_ch1_en	R/W	0	**PCM Transmit Dataport 1 Channel 1 Enable** This enables the PCM TRANSMIT dataport channel 1, which is connected to APB Channel TX_PCM_CH1 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**
3:0	dport1_slot_map_ch1	R/W	0	**PCM Dataport 1 Channel 1 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-60: Dataport 1 Slot Mapping Channel 2 Register

Dataport 1 Slot Mapping Channel 2			DATAPORT_1_SLOT_MAPPING_CHANNEL_2	[0x00EC]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_dport1_ch2_en	R/W	0	**PCM Transmit Dataport 1 Channel 2 Enable** This enables the PCM TRANSMIT dataport channel 2, which is connected to APB Channel TX_PCM_CH2 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**
3:0	dport1_slot_map_ch2	R/W	0	**PCM Dataport 1 Channel 2 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-61: Dataport 1 Slot Mapping Channel 3 Register

Dataport 1 Slot Mapping Channel 3			DATAPORT_1_SLOT_MAPPING_CHANNEL_3	[0x00F0]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_dport1_ch3_en	R/W	0	**PCM Transmit Dataport 1 Channel 3 Enable** This enables the PCM TRANSMIT dataport channel 3, which is connected to APB Channel TX_PCM_CH3 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**
3:0	dport1_slot_map_ch3	R/W	0	**PCM Dataport 1 Channel 3 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-62: Dataport 2 Slot Mapping Channel 0 Register

Dataport 2 Slot Mapping Channel 0	DATAPORT_2_SLOT_MAPPING_CHANNEL_0	[0x00F4]

Bits	Field	Access	Reset	Description
31:8	-	RO	0	**Reserved**
7	pcm_dport2_ch0_en	R/W	0	**PCM Receive Dataport 2 Channel 0 Enable** 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**
3:0	dport2_slot_map_ch0	R/W	0	**PCM Dataport 2 Channel 0 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-63: Dataport 2 Slot Mapping Channel 1 Register

Dataport 2 Slot Mapping Channel 1			DATAPORT_2_SLOT_MAPPING_CHANNEL_1	[0x00F8]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_dport2_ch1_en	R/W	0	**PCM Receive Dataport 2 Channel 1 Enable** 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**
3:0	dport2_slot_map_ch1	R/W	0x1	**PCM Dataport 2 Channel 1 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-64: Dataport 2 Slot Mapping Channel 2 Register

Dataport 2 Slot Mapping Channel 2			DATAPORT_2_SLOT_MAPPING_CHANNEL_2	[0x00FC]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_dport2_ch2_en	R/W	0	**PCM Receive Dataport 2 Channel 2 Enable** 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**
3:0	dport2_slot_map_ch2	R/W	0x2	**PCM Dataport 2 Channel 2 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-65: Dataport 2 Slot Mapping Channel 3 Register

Dataport 2 Slot Mapping Channel 3			DATAPORT_2_SLOT_MAPPING_CHANNEL_3	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_dport2_ch3_en	R/W	0	**PCM Receive Dataport 2 Channel 3 Enable** 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**

3:0	dport2_slot_map_ch3	R/W	0x3	**PCM Dataport 2 Channel 3 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-66: Dataport 2 Slot Mapping Channel 4 Register

Dataport 2 Slot Mapping Channel 4			**DATAPORT_2_SLOT_MAPPING_CHANNEL_4**	**[0x0104]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_dport2_ch4_en	R/W	0	**PCM Receive Dataport 2 Channel 4 Enable** 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**
3:0	dport2_slot_map_ch4	R/W	0x4	**PCM Dataport 2 Channel 4 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-67: Dataport 2 Slot Mapping Channel 5 Register

Dataport 2 Slot Mapping Channel 5			**DATAPORT_2_SLOT_MAPPING_CHANNEL_5**	**[0x0108]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_dport2_ch5_en	R/W	0	**PCM Receive Dataport 2 Channel 5 Enable** 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**
3:0	dport2_slot_map_ch5	R/W	0x5	**PCM Dataport 2 Channel 5 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-68: Dataport 2 Slot Mapping Channel 6 Register

Dataport 2 Slot Mapping Channel 6			**DATAPORT_2_SLOT_MAPPING_CHANNEL_6**	**[0x010C]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	pcm_dport2_ch6_en	R/W	0	**PCM Receive Dataport 2 Channel 6 Enable** 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**
3:0	dport2_slot_map_ch6	R/W	0x6	**PCM Dataport 2 Channel 6 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-69: Dataport 2 Slot Mapping Channel 7 Register

Dataport 2 Slot Mapping Channel 7			**DATAPORT_2_SLOT_MAPPING_CHANNEL_7**	**[0x0110]**
Bits	**Field**	**Access**	**Reset**	**Description**

31:8	-	RO	0	**Reserved**
7	pcm_dport2_ch7_en	R/W	0	**PCM Receive Dataport 2 Channel 7 Enable** 0: Disabled 1: Enabled
6:4	-	RO	0	**Reserved**
3:0	dport2_slot_map_ch7	R/W	0x7	**PCM Dataport 2 Channel 7 Slot Mapping** Selects the serial data slot on the PCM bus to output data for the corresponding channel when that channel is enabled.

Table 23-70: PCM Data Controls Register

PCM Data Controls				**PCM_DATA_CONTROLS**	**[0x0114]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:6	-	RO	0	Reserved	
5	pcm rx interleave	R/W	0	**PCM Rx Interleave Mode Enable** In this mode the DATAPORT_SR must be 1/2 of the INTERFACE_SR. Samples for two channels are received on the same slot in a TDM fashion. The LSB indicates which channel is received. 0: Disabled 1: Enabled	
4:2	-	RO	0	Reserved	
1	pcm_tx_interleave	R/W	0	**PCM Tx Interleave Mode Enable** In this mode, the DATAPORT_SR must be 1/2 of the INTERFACE_SR. Samples for two channels are transmitted on the same slot in a TDM fashion. The LSB indicates which channel is transmitted. 0: Disabled 1: Enabled	
0	-	RO	0	Reserved	

Table 23-71: Audio Enable Register

Audio Enable				**GLOBAL_ENABLE**	**[0x0118]**
Bits	**Field**	**Access**	**Reset**	**Description**	
15:1	-	RO	0	**Reserved**	
0	global_en	R/W	0	**Audio Subsystem Enable** 0: Disabled 1: Enabled	

24. Bluetooth 5 Low Energy (LE) Radio

Bluetooth 5 Low Energy (LE) radio is the latest version of the Bluetooth wireless communication standard. It is used for wireless headphones and other audio hardware and communication between various smart home and Internet of Things (IoT) devices.

The system operates in the 2.4GHz ISM (Industrial, Scientific, Medical) band at 2400MHz–2483.5MHz. It uses 40 RF channels. These RF channels have center frequencies 2402 + k x 2MHz, where k = 0, ..., 39. A frequency-hopping transceiver is used to combat interference and fading.

Bluetooth 5 technology provides:

- 1Mbps, 2Mbps, and Long Range coded (125kbps and 500kbps) data rates
- Increased broadcast capability
- Advertising packet up to 255 bytes
- On-chip matching network to the antenna
- Provides hardware on-the-fly encryption and decryption for lower power consumption
- Supports mesh networking
- Supports high-quality audio streaming (isochronous)
- The low-power proprietary mode that supports 20kbps, 40kbps, 500kbps-MSK/GFSK, 1Mpbs-GFSK

24.1 Power-Efficient Design

The provided Bluetooth Low Energy radio is optimized for low-power operation.

- Higher transmit power up to +9.5dbm
- Low transmit current of 2.5mA at 0dbm at 3.3V
- Low receive current of 1.5mA at 3.3V

24.2 Bluetooth Hardware Accelerator

The dedicated Bluetooth hardware accelerator eliminates the need for application software to accommodate the strict timing requirements and restrictions. It transparently increases system performance while reducing the overall power consumption of the Bluetooth system.

24.3 Packetcraft Software Stack

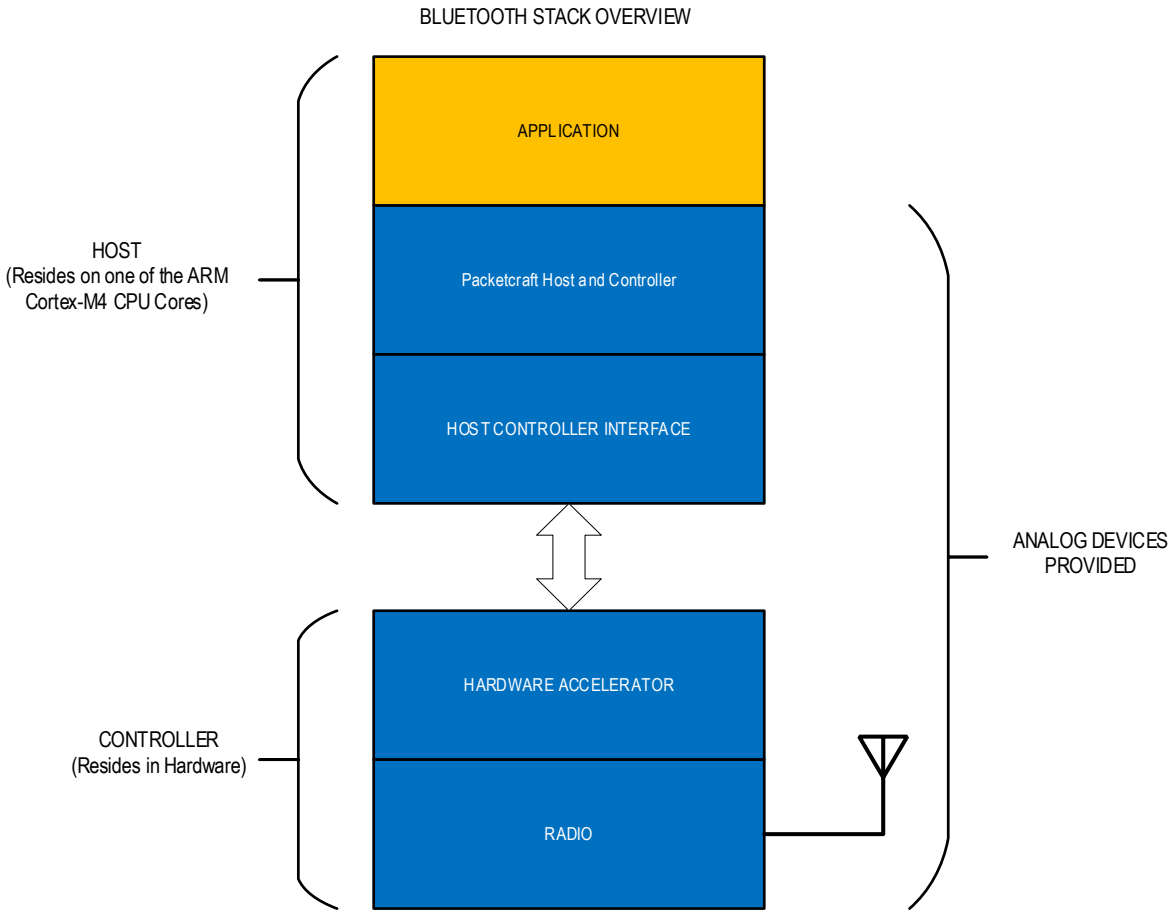
Analog Devices provides the Packetcraft Host and Controller in library form. This provides application developers access to Bluetooth without validation and development of a software stack.

The Packetcraft software stack interfaces to the Bluetooth link layer running on dedicated hardware. The dedicated hardware for the stack enables the ultimate power management for IoT applications.

The Packetcraft Host and Controller feature the following:

- C library for linking directly into an application development tool
- The host selects the PHY it needs to use at any given time, enabling long-range or higher bandwidth only when required
- LE 1M
- LE Coded S = 2
- LE Coded S = 8
- LE 2M
- Bluetooth 5 advertising extension support for enabling next-generation Bluetooth beacons
- Larger packets and advertising channel offloading
- Packets up to 255 octets long
- Advertising packet chaining
- Advertising sets
- Periodic advertising
- High-duty cycle non-connectable advertising
- Sample applications using standard profiles built on the Arm Cordio-B50 software framework

Figure 24-1: MAX32665/MAX32666 Bluetooth Stack Overview



24.4 Pins

The interface has two device pins to connect to the off-chip user-provided antenna. The ANT device pin is the radio frequency signal pin, and it should be routed through the ANT THRU device pin to the antenna. The ANT device pin is a 50Ω source impedance driver.

24.5 Configuration

The Radio and Hardware Accelerator requires a 32MHz external crystal specified in the data sheet. The CPU core hosting the Packetcraft Host and Controller stack must run at a core speed greater than 32MHz.

Perform the following steps to enable the Bluetooth radio:

1. Set *GCR_BTLELDOCN*.ldowen and set to *GCR_BTLELDOCN*.ldowoen 1 to enable the internal Bluetooth LDO.
2. Set *GCR_CLKCN.*x32M_en to 1.
3. Wait for *GCR_CLKCN.*x32M_rdy to be set to 1. Do not access any Bluetooth registers beforehand.

24.6 Documentation

The Packetcraft Host and Controller Product Sheet and Profiles can be found at:

https://www.packetcraft.com/

The Arm MBED documentation repository can be found at:

https://os.mbed.com/docs/mbed-cordio

25. Trust Protection Unit (TPU)

The trust protection unit (TPU) is a collection of hardware and software mechanisms that provide advanced cryptographic security. Dedicated hardware engines greatly increase the speed of computationally intensive cryptographic algorithms.

The dedicated symmetric block cipher engine provides the following features:

- AES-128, 192, and 256 (FIPS 197).
- DES and 3DES/TDEA (NIST SP800-67).
- Support for NIST-approved block modes (SP800-38).
- Parallel calculation of block cipher and hash functions

The requirements for meeting security validations are often updated. Contact Analog Devices before starting any secure product design to ensure that the cryptographic features of this device are compatible with the most recent requirements.

The dedicated hash function accelerator computes SHA-1, 224, 256, 384, and 512 (FIPS 180-3) values used in CMAC and HMAC.

Hamming code generator provides the ability to calculate an error correction code (ECC) on a block of data that can detect single or two-bit errors.

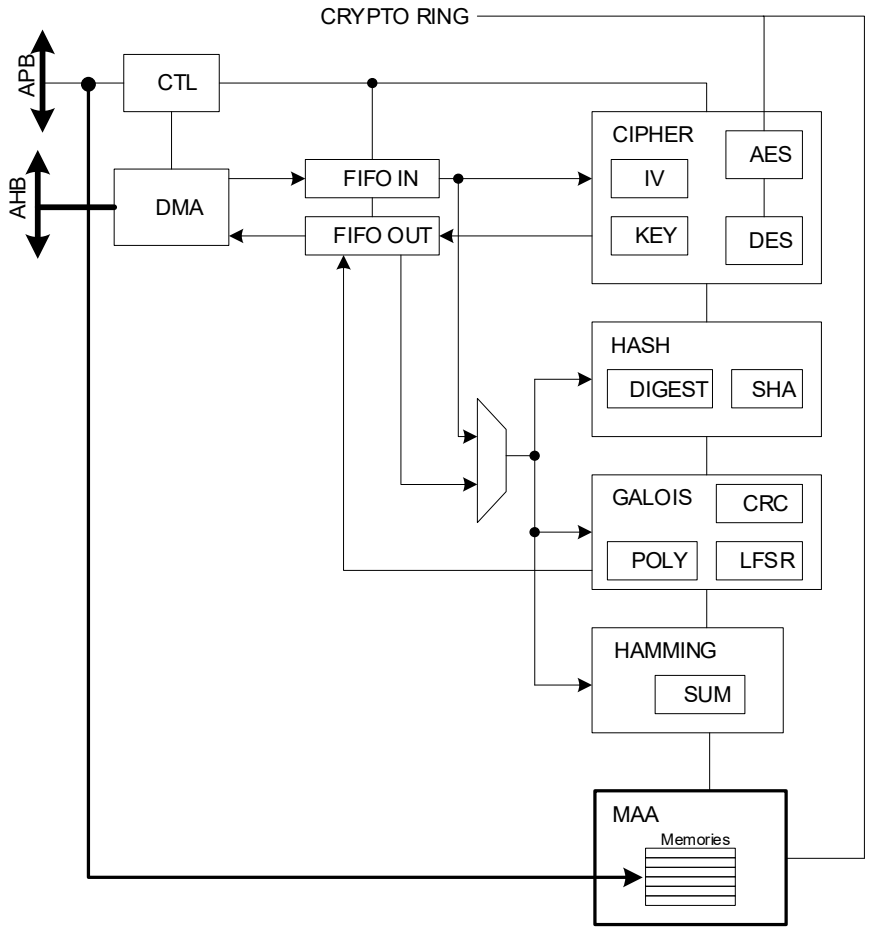
The cryptographic accelerator also provides a dedicated modular arithmetic accelerator (MAA). It provides high speed calculations of asymmetrical keys used in DSA, RSA, ECDSA and other cryptographic algorithms with modulus and operands up to 2048 bits in length. The Analog Devices Universal Cryptographic Library (UCL) available for some devices, supports modulus and operands up to 4096 bits using the provided MAA. The MAA has a dedicated memory space for the operands and operates independently of the CPU except when loading or unloading the operands.

Most functions are configurable for big- or little-endian operations.

The cryptographic accelerator interfaces with both the APB and AHB busses.

All cryptographic operations begin by resetting the cryptographic block. The cryptographic accelerators functions each have their own done bit, as well a global done bit for the cryptographic block. The cryptographic accelerators can generate an interrupt if enabled.

Figure 25-1: Cryptographic Accelerator Block Diagram



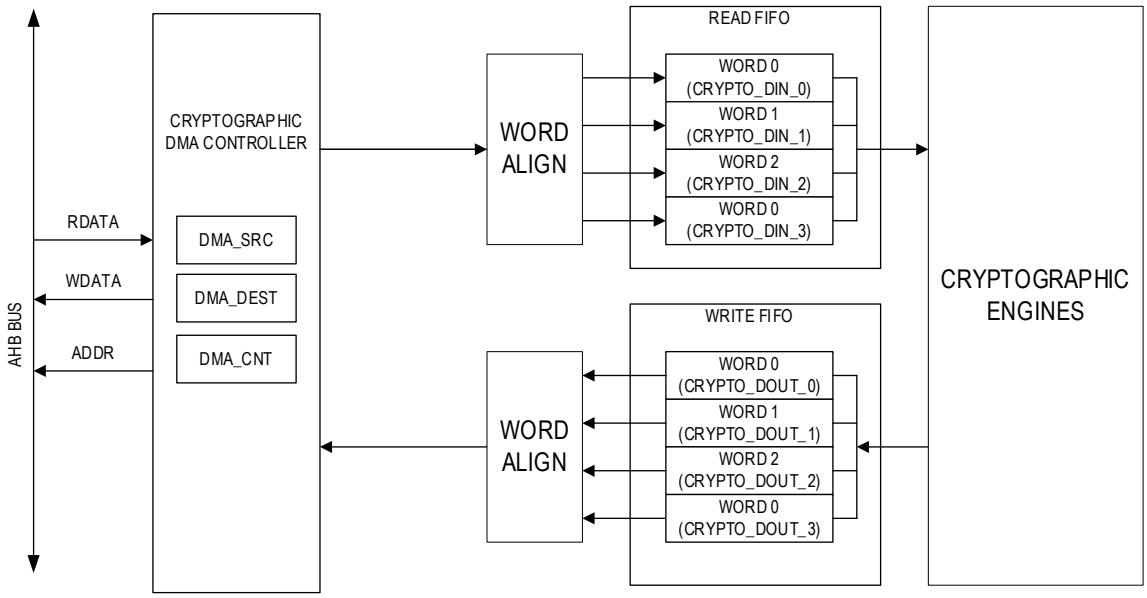
25.1 Dedicated Cryptographic DMA Engine (CDMA)

A dedicated DMA engine performs high-speed accesses between the TPU and memory on the AHB bus. This greatly improves performance during data-intensive operations such as encryption/decryption and hashing. The source, destination, and count registers are located in the cryptographic accelerator register space. The source and destination of the DMA engine can point to the same memory location to encrypt or decrypt the data in situ.

While the cryptographic accelerator is busy encrypting or hashing data, the DMA prefetches the data for the next operation and stores it in the read FIFO. Once the cipher or hash generator is done, the data for the next operation is immediately available. Data output is buffered in the write FIFO so the next cipher or hash operation can immediately begin calculating on the next block. This keeps the cipher and hash generator running continuously without having to wait for data to be written to or read from the bus.

The block cipher can operate in parallel with the hash accelerator as long as only one operation uses the DMA.

Figure 25-2: DMA Block Diagram



25.1.1 FIFOs

The read FIFO and write FIFO have programmable sources as shown in *Table 25-1* to allow flexibility in their operation.

Table 25-1: Cryptographic Accelerator DMA Sources

READ FIFO SOURCES	WRITE FIFO SOURCES
Read FIFO APB AHB DMA Random Number Generator	Write FIFO None Cipher output

During cryptographic operations, a typical setup is to use the AHB DMA as the read FIFO source and the cipher output as the write FIFO source. Data written to the write FIFO is always written out to the AHB DMA. This setup reads data from memory and writes the encrypted or decrypted result back to memory.

A Cipher-based Message Authentication Code (CMAC) is similar to a digital signature or a Keyed-Hash Message Authentication Code (HMAC). CMACs use a cipher in a block-chaining mode to form a cryptographic checksum. In this mode, the AHB DMA is the read FIFO source, but the cipher output is not written back to memory. Only the final cipher block is of interest, so set the write FIFO source to none.

You can use DMA to copy memory, similar to the memcpy standard C function, by setting the write FIFO source to the read FIFO. If the Hamming ECC generator is enabled, you can copy flash memory pages to memory while simultaneously calculating the error correction code.

You can fill memory with a block of data similar to the memset() standard C function by pointing the write FIFO source to the read FIFO and setting the read FIFO to the APB. Similarly, you can fill memory with random data by pointing the read FIFO source to the random number generator and the write FIFO source to the read FIFO.

To decrypt or encrypt data, set the write FIFO source to the cipher output. To implement memcpy() or memset() functions, or to fill memory with random data, you should set the write FIFO source to the read FIFO. When calculating a hash or CMAC, disable the write FIFO.

The setting of the read and write FIFOs sources are detailed in each section of specific operations.

For cipher, hash, or Galois operations most operations are finished when the DMA transfer is complete. If the *TPU_CTRL*.*rdsrc* is configured for DMA, both the *TPU_CTRL*.*dma_done* and the associated *.done* flag are set after the entire DMA operation is complete. In most cases only the *TPU_CTRL*.*done* flag is required. Setting the *TPU_CTRL*.*dmadnemsk* field will prevent the *TPU_CTRL.dma_done* field from setting the *TPU_CTRL*.*done* bit. This reduces software overhead by only using the specific operation's *.done* flag and masking the DMA from interrupting the CPU.

For cipher operations, when *TPU_CTRL*.*wrsrc* is configured for cipher output, the *TPU_CTRL*.*cph_done* is set only after the last cipher text has completed the DMA transfer out to memory.

After the cryptographic accelerator reset, the *TPU_CTRL*.*rdy* must be polled in the software before any other actions can start.

25.1.2 Direct FIFO Access

The read and write FIFOS are directly accessible via the TPU_DIN_[3:0] and TPU_DOUT_[3:0] registers, respectively. In general, however, the CMDA is much more efficient and requires less interaction.

If direct access is required, only 32-bit accesses to the TPU_DIN_0 and TPU_DOUT_0 registers should be used. Do not use the registers [3:1] for direct access.

25.1.3 Cache Security

Cryptographic operands and results may be stored in cached memory as part of normal device operation. For increased security, invalidate the cache memory associated with memory used in cryptographic operations.

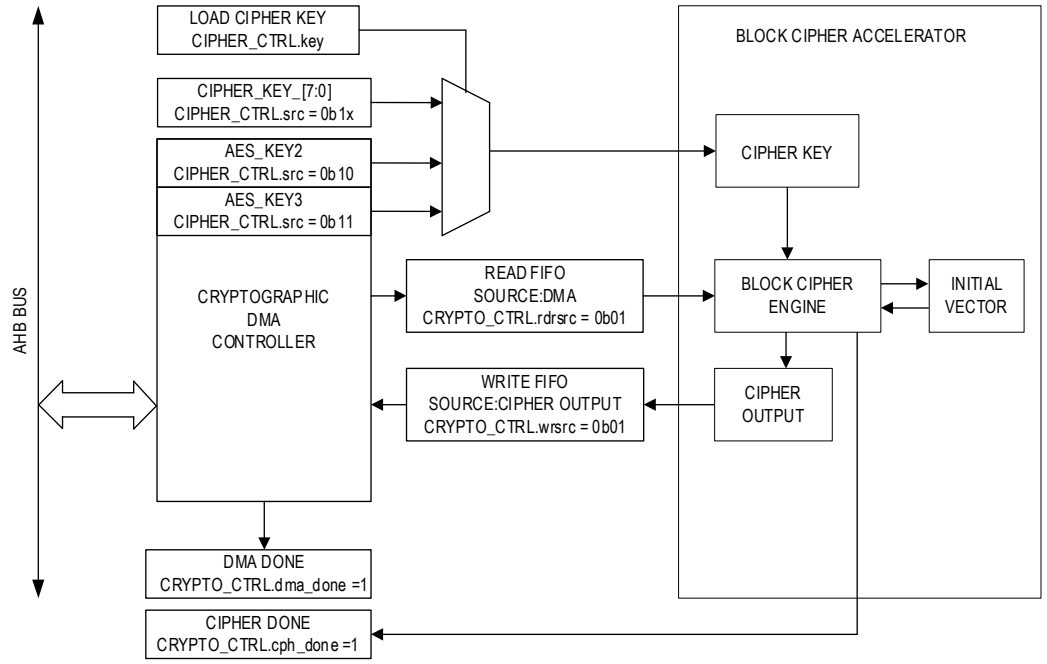
25.2 Block Cipher Accelerator

The block cipher accelerator is a dedicated hardware module that accelerates the computation of the following algorithms:

- AES-128
- AES-192
- AES-256
- Data Encryption Standard (DES)
- Triple Data Encryption Algorithm (TDEA/3DES)
- DES and TDEA have been deprecated by NIST are only provided for legacy support.

Figure 25-3 is a block diagram of the block cipher accelerator and its interface to the CMDA.

Figure 25-3: Block Cipher Block Diagram



The symmetric block ciphers encrypt or decrypt data in blocks. The block sizes for each cipher are shown in *Table 25-2*.

Table 25-2: Symmetric Block Ciphers

CIPHER	KEY SIZE	USED KEY BITS	EFFECTIVE STRENGTH (NIST SP800-57)	BLOCK SIZE
TDEA	192-bits	CIPHER_KEY[167:0] (168-bits)	112-bits	64-bits
AES-128	128-bits	CIPHER_KEY[127:0] 128-bits	128-bits	128-bits
AES-192	192-bits	CIPHER_KEY[191:0] 192-bits	192-bits	128-bits
AES-256	256-bits	CIPHER_KEY[255:0] 256-bits	256-bits	128-bits

The accelerator supports the block cipher modes approved by NIST SP800-38A.

- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter (CTR)

In the simplest mode Electronic Code Book (ECB), each data block is simply encrypted or decrypted using the cipher. A side effect of this is that identical data blocks will encrypt to the same ciphertext. Various modes of operation are used that chain or feedback ciphertext from the previous block to seed the next encryption operation. This causes identical, plaintext data blocks to encrypt to different ciphertexts.

For the CFB mode of operation, the mode size is equal to the block size. 128-bit CFB is supported for AES, and 64-bit CFB is supported for TDEA. 1-bit CFB and 8-bit CFB are not supported. For the CTR mode of operation, the lower 32-bits of the initial vector increment.

Block cipher operations set *TPU_CTRL.cph_done* = 1 and when complete.

25.2.1 Cipher Key Storage and Initialization

Block cipher operations requires a user-supplied cipher key to be loaded into CIPHER_KEY[7:0] before any algorithm can be executed. The cipher key is loaded into non-volatile memory during a secure bootloader session, and restored to the AES_KEY[3:2], registers following a power-on reset.

The length of the key is dependent on the specific algorithm used.

The following procedure is required to load a key of 128 bits or less:

1. Clear *TPU_CTRL.done = 0* and *TPU_CTRL.dma_done* = 0.
2. Set *TPU_CTRL.src* = 0b01 to copy the contents of AES_KEY_2 into *CIPHER_KEY_0*.
3. Poll until hardware sets *TPU_CTRL.dma_done* = 1.
4. Clear *TPU_CTRL.done* = 0 and *TPU_CTRL.dma_done* = 0.

The following procedure is required for key lengths greater than 128 bits:

1. Clear *TPU_CTRL*.done = 0 and *TPU_CTRL.dma_done* = 0.
2. Set *CIPHER_CTRL.src* = 0b01 to copy the contents of AES_KEY_2 into *CIPHER_KEY_0*.
3. Poll until hardware sets *TPU_CTRL.dma_done* = 1.
4. Clear *TPU_CTRL.done* = 0 and *TPU_CTRL.dma_done* = 0.
5. Set *CIPHER_CTRL.src* = 0b11 to copy the contents of AES_KEY_3 into *CIPHER_KEY_0*.
6. Poll until hardware sets *TPU_CTRL.dma_done* = 1.
7. Clear *TPU_CTRL.done* = 0 and *TPU_CTRL.dma_done* = 0.

25.2.2 Operation

The cipher algorithm and mode of operation are set in the cipher control register. The cipher key must be loaded before starting a block cipher operation. The cipher starts operating once the FIFO is full.

1. Reset the cryptographic accelerator by setting *TPU_CTRL.rst* = 1.
2. Poll until hardware sets *TPU_CTRL.rdy* = 1.
3. Select the cipher algorithm operation using *CIPHER_CTRL.cipher*.
4. Select the mode of operation using *CIPHER_CTRL.mode*.
5. Select encryption or decryption mode *CIPHER_CTRL.enc*.
6. Load *CIPHER_INIT_0*, *CIPHER_INIT_1*, *CIPHER_INIT_2*, *CIPHER_INIT_3* with the initial vector if using CBC, CFB, OFB, or counter modes.
7. Set *TPU_CTRL.rdsrc* = 0b01 to select the read FIFO source as DMA.
8. Set *TPU_CTRL.wrsrc* to 0b01 to select the cipher output FIFO source as DMA.
9. Load DMA source address to *TPU_DMA_SRC*.
10. Load DMA destination address to *TPU_DMA_SRC*.
11. Load DMA count to *TPU_DMA_CNT*.

At the end of the DMA count:

- *TPU_CTRL.done* = 1
- *TPU_CTRL.dma_done* = 1
- *TPU_CTRL.cph_done* = 1

An interrupt will be generated if *TPU_CTRL.int* = 1.

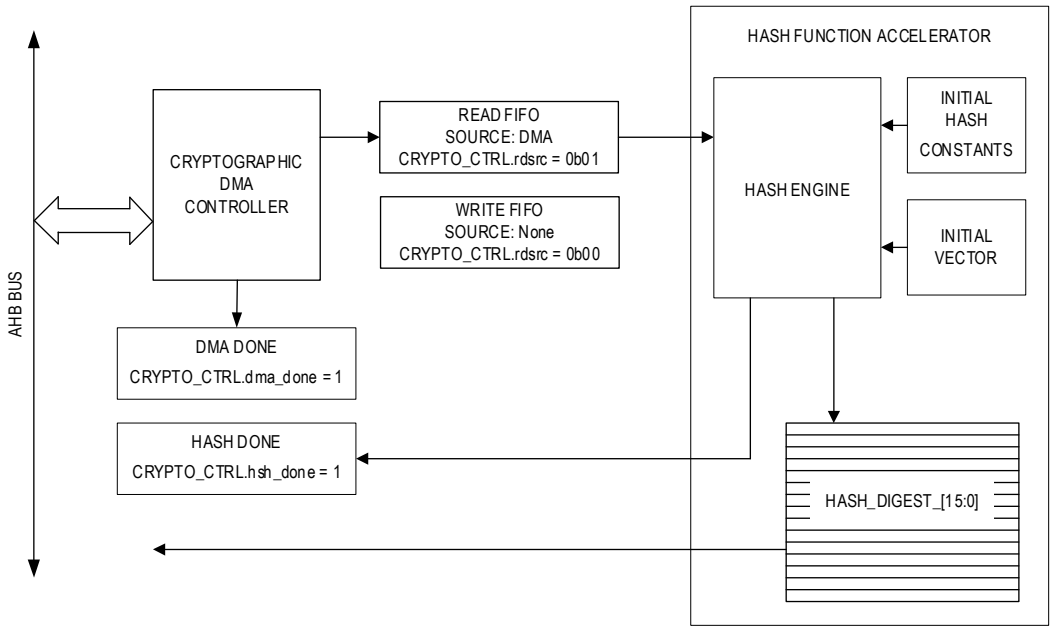
25.3 Hash Function Accelerator

A hash algorithm takes an input message of arbitrary length and summarizes it in a fixed-length message. The hash function accelerator executes the SHA algorithms listed in *Table 25-3*.

Table 25-3: Hash Functions

ALGORITHM	DIGEST LENGTH	EFFECTIVE STRENGTH (NIST SP800-57)	BLOCK SIZE
SHA-1	160 bits	63 bits	512 bits
SHA-224	224 bits	112 bits	512 bits
SHA-256	256 bits	128 bits	512 bits
SHA-384	384 bits	192 bits	1024 bits
SHA-512	512 bits	256 bits	1024 bits

Figure 25-4: Block Cipher Diagram



Data is processed in 512-bit or 1024-bit blocks. After every 16 or 32 words are written to the hash block, several cycles are needed to complete the hash of that block. The number of cycles needed to compute the hash is dependent upon the number of rounds in the algorithm. The first 16 rounds are calculated as the 16 or 32 words are written to the hash block. The remaining rounds are calculated at a rate of one clock cycle per round.

The integrated DMA can fetch data for the hash accelerator. Calculation of the hash can occur in parallel with the block cipher as long as only one of the operations uses the DMA.

Setting the *HASH_CTRL.init* bit will seed the hash accelerator with the secure hash constants required by security validation requirement . Contact Analog Devices for specific information about the seeding process and its comparability with the latest security requirements for NIST FIPS and other certifications.

25.3.1 Last Message Block Padding

Once all data is written to the cryptographic data register, the final message block must be padded according to the FIPS Publication 180 standard. The standard requires that the bit length be appended to the end of the message. The length of the message in binary representation is contained in the HASH Message Size Registers *HASH MSG_SZ_0*. Prior to hashing the last message data, set *HASH_CTRL.last* along with the HASH_MSG_SZ registers. The HASH_MSG_SZ value is automatically padded to the last message block.

For SHA-1 and SHA-256, a single bit equal to 1 is appended to the end of the message. The message is then padded using software with zeros until 64 bits remain in the last 512-bit block. If less than 64 bits remain, then zeros are appended to the message until 64 bits remain in the next 512-bit block. The *HASH_CTRL.last* field is set along with HASH_MSG_SZ_[1:0]. Hardware appends values in these registers to the last 64 bits of the final message block.

For SHA-384 and SHA-512, a single bit equal to 1 is appended to the end of the message. The message is then padded with zeros until 128 bits remain in the last 1024-bit block. If less than 128 bits remain, then zeros are appended to the message until 128 bits remain in the next 1024-bit block. The *HASH_CTRL.last* is set along with *HASH MSG_SZ_0*.

The automatic padding feature is used in terms of message bytes, not bits. Therefore, the *HASH MSG_SZ_0* are expressed in bytes. In addition, the feature automatically generates an additional padding-only block if the last block of message data cannot accommodate the 64- or 128-bit padding block.

As an exception, attempting to hash a 0 message-size block must include a dummy write to the HASH message digest register.

Hash operations using the CMDA set *TPU_CTRL.hsh_done* = 1 and *TPU_CTRL.dma_done* = 1, and *TPU_CTRL.done* = 1 when complete.

25.4 Hamming Code Accelerator

The Hamming code accelerator calculates an Error Correction Code (ECC) on a block of data. Hamming codes are capable of correcting single-bit errors. You can include an extra parity bit to detect two-bit errors. This is commonly referred to as Signal Error Correction, Double Error Detection (SEC-DED). Three errors masquerade as a correctable, single-bit error.

Multi-level Cell (MLC) Flash memories require Error Correction Codes that can correct multiple-bit errors since a corrupt cell can have multiple bit errors.

The hardware can calculate ECCs for a block of data up to 216 bits in length (8kB), but software implementations can extend this to any length. Because the Hamming code can only correct a single-bit error, increasing the block size increases the likelihood of multiple errors that are uncorrectable. The Hamming code generator in the cryptographic accelerator generates even parity on even halves of bit groups.

If you want the parity of the odd halves of the bit groups, XOR the parity of the even halves with the parity of the entire array. If the parity of the entire array is odd, the parity of the odd halves is the inverse of the parity of the even halves. If the parity of the entire array is even, the parity of the odd halves is identical to the parity of the even halves.

Figure 25-5: Hamming XOR Calculations

For a block of data 2n bits in length, n+1 ECC bits are needed for single-bit error correction (ECC B0 through Bn = n+1 ECC bits). ECC bit n indicates parity of the entire array. If it is different than the stored ECC bit n, it indicates an error in the data array. You can determine the location of the error using the lower n ECC bits (B0 to Bn-1).

To determine if an error occurred, XOR the saved and calculated ECC bits. If the result is zero, no error occurred. If the result of the ECC XOR operation is not zero, then the location of the failing bit is given by the inverse of the ECC XOR result. If the failing bit location is greater than or equal to the size of the data block (bit n of the inverted ECC XOR is set), then the error is in the ECC bits. The ECC bit that is corrupt is the bit that is set as a result of the XOR of the two sets of ECC bits.

failing_bit_location = ~(ecc_saved ^ ecc_calculated)

An error in the most significant ECC bit n masquerades as an error in the most significant bit of the data. To properly detect and correct an error in the MSB of the ECC, include an extra ECC bit (n+2 ECC bits). These two most significant ECC bits should be identical if they are error free. If they are different, you can determine the failing ECC bit by XORing the saved and calculated ECC bits.

You can save this extra ECC bit at the cost of an additional parity calculation. If the MSB of the data is set, the ECC parity bits should be XOR'd with 2n+1-1 (n 1s). This effectively swaps the MSB of the data with the MSB of the ECC for purposes of

parity calculation. When examining the result, if the ECC calculation indicates the MSB of the data is corrupt (the result is 2n+1-1), then the error is really in the MSB of the ECC. If the result indicates the error is in the MSB of the ECC (the result is 2n+1), then the error is really in the MSB of the data.

ECC bits n and above are all identical. They all indicate the parity of the entire array. To use a block size of 64k bits (8k bytes), the parity bit of the entire array (bit 16) can be duplicated to obtain the higher-order ECC bits.

You can generate Hamming codes over larger blocks using software. After the Hamming code is generated for an 8k block of data, software should examine the parity of the block of data just calculated (bit 16 of the ECC register). If the parity of the block is odd (parity bit is set), software should XOR additional software-maintained ECC bits with the inverse of the block address. If the parity of the block is even (parity bit of the block is clear), software does not need to modify the additional ECC bits. You should reset the parity of the block (bit 16 of the ECC register) for each block, but the remaining ECC bits (B0-B15) should remain unchanged and accumulate their respective XORs throughout the entire array.

To achieve Double Error Detection (but not correction), include another ECC parity bit. If the result of the ECC XOR is not zero, then this extra ECC bit should also be set indicating an error has occurred. If this bit is clear after the ECC XOR operation, it means an even number of errors has occurred, and the data is not correctable. This does not mean this extra ECC bit detects an even number of bit errors. It just indicates that an even number of errors have occurred. It is possible for an even number of bit errors to masquerade as valid data. This extra ECC bit is capable of detecting two-bit errors. If an odd number of bit errors occur, they always masquerade as a single correctable bit error. This is a limitation of using parity to indicate error. Parity can only detect an odd number of errors. For stronger error detection, a Cyclic Redundancy Check (CRC) is used.

The lower n ECC bits (B0 to Bn-1) are all that is needed to determine the location of a single-bit error in the data. The next ECC bit (which is the parity of the entire array) is simply used to indicate there is an error in the data that needs to be corrected. Subsequent ECC bits are only needed to detect an error in this added ECC bit or to detect double-bit errors. If an alternative error detection scheme is used, such as a CRC, then only n ECC bits are needed to find the location of a single-bit error in the data array.

Because all CRCs with at least two terms in the generator polynomial will detect all single-bit errors, ECC bit n, which is used to determine if there is a single-bit error in the array by checking parity is unnecessary. A CRC polynomial with an even number of terms has the parity polynomial (x+1) as one of its factors and checks parity as well. A CRC of sufficient length is also capable of detecting all two-bit errors, making an ECC bit added for double-error detection unnecessary as well.

Because a single-bit error in the CRC is catastrophic, the CRC should also be protected with ECC bits. If the CRC is appended to the data for the Hamming code calculation, you can protect it with the same set of ECC bits used to protect the data by including one additional ECC bit to account for the increase in block size.

Most manufacturers of NAND Flash memories have application notes that describe the calculation of a variation of Hamming codes. These application notes calculate parity on both the odd and even halves of the bit groups. As a result, they require 2*n ECC bits, almost twice as many as necessary. If the even half is XOR'd with the odd half, the result is the parity of the entire array. Instead of storing parity for both the even halves and odd halves of all bit groups (which requires 2*n ECC bits), an implementation only really needs to store the parity of the entire array (n+1 ECC bits). You can determine the odd half of ECC bits by XORing the even half of ECC bits with the parity of the entire array. The parity of the entire array is just the next bit in the ECC register, so saving n+1 bits from this register saves the parity of the entire array. Storing both the odd and even parity halves offers no more error protection than the methods described above. For both methods, an odd number of errors masquerades as a correctable single-bit error, and all double-bit errors are detected.

25.5 Modular Arithmetic Accelerator

The Modular Arithmetic Accelerator (MAA) is a dedicated hardware module that performs high-speed calculations that are key parts of asymmetric cryptographic algorithms based on DSA, RSA, and ECDSA algorithms.

The MAA features include:

- Supports up to a 2048-bit operand size
- Performs up to 2048-bit modular exponentiation (ae mod m)
- Performs up to 2048-bit modular multiplication (a*b mod m)
- Performs up to 2048-bit modular square (b2 mod m)
- Performs up to 2048-bit modular square followed by modular multiply ((b2 mod m) * a mod m)
- Performs up to 2048-bit modular addition (a+b mod m)
- Performs up to 2048-bit modular subtraction (a-b mod m)
- Optimized calculation mode to maximize speed
- Non-optimized calculation mode to maximize security

These operations can be combined to perform modular inversion and modular reduction.

The MAA operates independently from the processor except when the processor is reading or writing the control register, or when it is used to load/unload the data in the specified data memory segment.

The MAA does not use the CMDA engine because its operands and parameters are stored in a dedicated 768x16 data memory segment.

25.5.1 Operation

The MAA control register *MAA_CTRL* provides option control on arithmetic operations, data partition, and status bits for start/busy. This starting address of most parameters is configurable within the memory instance is configurable using the Memory Assignment (AMA, BMA, RMA and TMA) fields. Per FIPS' big-endian data convention, the most significant byte or sub-word of a multi-byte word is loaded first and stored at the lowest storage location.

Table 25-4: Cryptographic Memory Segments

SEGMENT	MEMORY SEGMENT ASSIGNMENT	DESCRIPTION
a	*MAA_CTRL.ama*	Multiplier/Operand A
b	*MAA_CTRL.bma*	Multiplicand/Operand B
e	Fixed	Exponent
m	Fixed	Modulus (only required for exponentiation operations)
t	*MAA_CTRL.tma*	Temporary storage. The data in this segment is undefined.
r	*MAA_CTRL.rma*	Result value

The Modular Accelerator Word Size field *MAA_MAWS.msgsz* defines the calculation size of a modular operation. The content of the register presents the number of bits for the modular operation. For example, to perform a 1007-bit (03EFh) modular operation, you would need to set this register to 03EFh prior to setting STC.

Valid word size is from 1 to 2048. The accelerator does not start if *MAA_MAWS.msgsz* is invalid.

The *TPU_CTRL.maa_done* bit is set to 1 after the completion of an MAA operation or when an error occurs. An interrupt is generated if *TPU_CTRL.int* is set. Software clears the bit by writing a 0.

25.5.2 MAA Memory

A dedicated, 1535-byte MAA memory begins at offset 0x0100. This memory holds the operands and intermediate values for MAA calculations. The memory space is subdivided into logical segments based on the size of the calculation.

The MAA memory makes a distinction between memory instances and memory segments when assigning parameter location. The following restrictions apply when storing parameters in the MAA memory.

Only one parameter can be located in each memory instance.

The modulus (m) is always stored in memory instance 5.

When an exponentiation operation is selected, the exponent (e) is always stored in memory instance 4. If another operation is selected, memory instance 4 is free to hold another parameter.

Parameters m, b, t and r must be stored in different memory instances (not segments) even if *MAA_MAWS.msgsz* < 1024 bits. For example, if b is stored in memory instance 0, then neither t nor r is stored in memory instance 1 when word size is smaller than 1024. Each memory instance is 256 bytes.

Table 25-5: MAA Memory Segments and Locations

	MEMORY INSTANCE	MEMORY SEGMENT (MAWS ≥ 1024)	MEMORY SEGMENT (MAWS < 1024)	DEDICATED FUNCTION	ADDRESS OFFSET
xMA[3:0]	0	0	0	None	0x0100 – 0x017F
			1		0x0180 – 0x01FF
	1	1	2	None	0x0200 – 0x027F
			3		0x0280 – 0x02FF
	2	2	4	None	0x0300 – 0x037F
			5		0x0380 – 0x03FF
	3	3	6	None	0x0400 – 0x047F
			7		0x0480 – 0x04FF
	4	4	8	Exponent (if needed)	0x0500 – 0x057F
			9		0x0580 – 0x05FF
	5	5	-	Modulus	0x0600 – 0x06FF

25.5.2.1 Memory Blinding

Memory blinding is an effective cryptographic technique that increases the difficulty of side channel attacks against cryptographic operations. In a poorly designed application, the MAA memory segments are in a fixed location, leaving them vulnerable to timing and power analysis attacks.

The memory blinding features provides the option of shifting the starting position of the a, b, e, and m parameters within their respective memory instances. Although you can alter the starting position, the entire parameter is still stored within a single memory instance. Each parameter can be independently configured with the default “unblinded” and three blinded starting positions. Memory blinding for each parameter is controlled by its corresponding Memory Select bits (AMS, BMS, EMS, MMS).

Table 25-6: MAA Memory Blinding Example (Memory Instance 0, MAWS > 1024)

xMS[1:0]	SHIFT	ADDRESS OFFSET
00	00x0000	0x0100 ... 0x01FF (no blinding)
01	00x0040	0x0140 ... 0x01FF, 0x0100 ... 0x013F
10	00x0080	0x0180 ... 0x01FF, 0x0100 ... 0x017F 1
11	00x00C0	0x01C0 ... 0x01FF, 0x0100 ... 0x01BF

25.5.2.2 MAA Clock Source

The MAA operates at either the LPCLK or LPCLK/2 frequency as determined by the GCR_CLKCL.ccd field and the frequency specified in the relevant data sheet.

25.5.2.3 Optimized Calculations

The optimized calculation control bit (*MAA_CTRL.ocal*) allows the software to optimize the speed of an exponentiation by skipping unnecessary multiply operations when the corresponding exponent bit is a 0. The bit defaults to 0, forcing the MAA to operate in a non-optimized mode. The non-optimized mode skips the leading zeros of the exponent and starts square/multiply operations when a 1 is detected. From this point on, multiply operation are completed for every exponent bit, regardless of its logical value.

Optimization is highly discouraged, as it leaves the device vulnerable to power analysis attacks.

25.5.2.4 MAA Operand Sizes

MAA operand size is specified by *MAA_MAWS.msgsz*. Valid values are from 1 to 2048. This value specifies valid ranges for a, b, e, and m.

a, b, and e, can have any values between 0 and 2MAWS-1 inclusive as long as their number of bits are less than those of m. For exponentiation operation, however, b memory must contain the value of 1, and e cannot be 0.

The m memory, which holds the modulus, has a value from 2MAWS-1 up to 2MAWS-1. For example, for a 16-bit *MAA_MAWS.msgsz* = 0x10, it has a value from 0x8000 (32768) up to 0xFFFF (65535).

If any parameter exceeds Word Size (MAWS), an invalid result is generated without issuing an MAA error status. It is up to application to check for invalid word sizes.

The MAA operands are stored as 64-bit blocks. For all the memories, operands must be zero padded to the 64-bit block boundary. There is no restriction on values stored in the memories beyond this boundary. For example, *MAA_MAWS.msgsz* = 65, and operand = 01 xxxx xxxx xxxx xxxxh. In this example, MAA operands are stored as two 64-bit blocks. Bit[63:0] contains the lower 64 bits. Bit[127:66] are zero padded while bit[65] = 1. Data in the words above where the 0000 0000 0000 0001h is stored can have any value.

For most calculations, memory segments t and r are used to store intermediate values during round operations and can contain the same value at the final operation.

For square-multiply and exponentiation, memory segment b is an additional temporary storage area during round operations.

25.6 Cyclic Redundancy Check (CRC) Engine

The CRC can perform CRC functions on data stored in SRAM. The CRC engine cannot be used to perform a CRC of data stored in flash memory.

An n-bit CRC can detect the following types of errors:

- Single-bit errors
- Two-bit errors for block lengths less than 2*k* where *k* is the order of the longest irreducible factor of the polynomial
- Odd numbers of errors for polynomials with the parity polynomial (x+1) as one of its factors (polynomials with an even number of terms)
- Burst errors less than *n* bits

Overall, all except 1 out of 2n errors are detected:

- 99.998% for a 16-bit CRC
- 99.99999998% for a 32-bit CRC

The hardware accelerator calculates the CRC of a block of data. Data is written to the *TPU_DOUT_3*:*TPU_DOUT_0*.

The starting initial CRC value is typically preset to all ones. If the starting initial value is preset to all zeros and an initial stream of all zeros is processed as the data, the CRC does not change.

Historically, CRCs were calculated on serial bit streams. Most serial bit streams were sent least significant bit (LSB) first. The CRC was calculated as each bit was transmitted or received. This resulted in the CRC being calculated on the LSB of the data first.

The CRC is typically appended to the end of the data. If the receiver calculates the CRC on both the data and received CRC, the result should be all zeros if the data and CRC were received error-free. Most implementations do not like to check

against an all-zero checksum. Therefore, most implementations invert the CRC before transmitting it. By inverting the CRC on the transmitting end, the resulting CRC on the receiving end should be a constant. The specific constant is dependent upon the CRC polynomial. This works because the non-inverted CRC calculated at the end of the data XOR'd with the received inverted CRC is all ones ($CRC \oplus \sim CRC = 1s$). Shifting all ones through the polynomial results in the same constant for each message, and the constant is dependent upon the polynomial.

Because the receiving end calculates a new CRC on both the data and received CRC, you must send the received CRC in the correct order, so the highest-order term of the CRC is shifted through the generator first. Because data is typically shifted through the generator LSB first, this means the CRC is reversed bitwise, with the highest-order term of the remainder in the LSB position. Software CRC algorithms typically handle this by calculating everything backwards. They reverse the polynomial and do right shifts on the data. The resulting CRC ends up being bit-swapped and in the correct format.

The CRC generator has a programmable polynomial up to 32 bits. The polynomial should be written to the *TPU_CRC_POLY* register. The largest term x^n defines the length of the CRC. When calculating the CRC on data LSB first, the polynomial should be reversed so that the coefficient of the highest power term is in the LSB position. The largest term x^n is implied (always one) and should be omitted when writing to the *TPU_CRC_POLY* register. This is necessary because the polynomial is always one bit larger than the resulting CRC, so a 32-bit CRC has a polynomial with 33 terms (x^0 … x^{32}).

CRC polynomials with good error-detection properties should be irreducible (the polynomial should not be factorable). Therefore, the constant term x^0 or 1 should always be present, otherwise, the polynomial would be factorable by x. If the constant term x^0 or 1 were not present, the resulting CRC would be cyclic with a subgroup smaller than x^n. The effective length of the CRC would be the difference between the highest and lowest order terms. Therefore, the highest and lowest order terms $x^n$and x^0 should always appear in the polynomial.

When found in literature, sometimes the LSB or MSB of the polynomial is omitted when the polynomial is written in binary. It is more common to see CRC polynomials with the MSB implied because that is the bit that is shifted off, XOR'd with the data, and tested to see if the result is set. Some literature assumes the reader knows that an *n*-bit CRC must have the x^n term set, or else it would be a smaller length CRC.

Some common CRC polynomials and their check constants are shown in *Table 25-7.* The polynomial register resets to the 32-bit CRC polynomial used by Ethernet, PPP, and file compression utilities such as zip or gzip.

By default, the CRC accelerator does right shifts and calculates the CRC on the LSB of the data first. The CRC can be calculated on the most significant bit (MSB) of the data first by setting the bit-swap control bit to 1 (*CRC_CTRL.msb* = 1). To calculate the CRC MSB first, you must left justify the polynomial in the *TPU_CRC_POLY* register. The hardware implies the MSB of the polynomial just as it did when shifting the LSB first. The LSB of the polynomial should be set—this defines the length of the CRC. The initial state of the CRC should also be left justified. When the CRC calculation is complete, it is necessary to right shift the CRC to right justify it if the polynomial is less than 32 bits.

Table 25-7: Common CRC Polynomials

Algorithm	Polynomial Expression	Order	Polynomial (TPU_CRC_POLY)	Check
CRC-32 Ethernet	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+x^0$	LSB	0xEDB8 8320	0xDEBB 20E3
CRC-CCITT	$x^{16}+x^{12}+x^5+x^0$	LSB	0x0000 8408	0x0000 F0B8
CRC-16	$x^{16}+x^{15}+x^2+x^0$	LSB	0x0000 A001	0x0000 B001
USB Data	$x^{16}+x^{15}+x^2+x^0$	MSB	0x8005 0000	0x800D 0000
Parity	x^1+x^0	LSB	0x0000 0001	

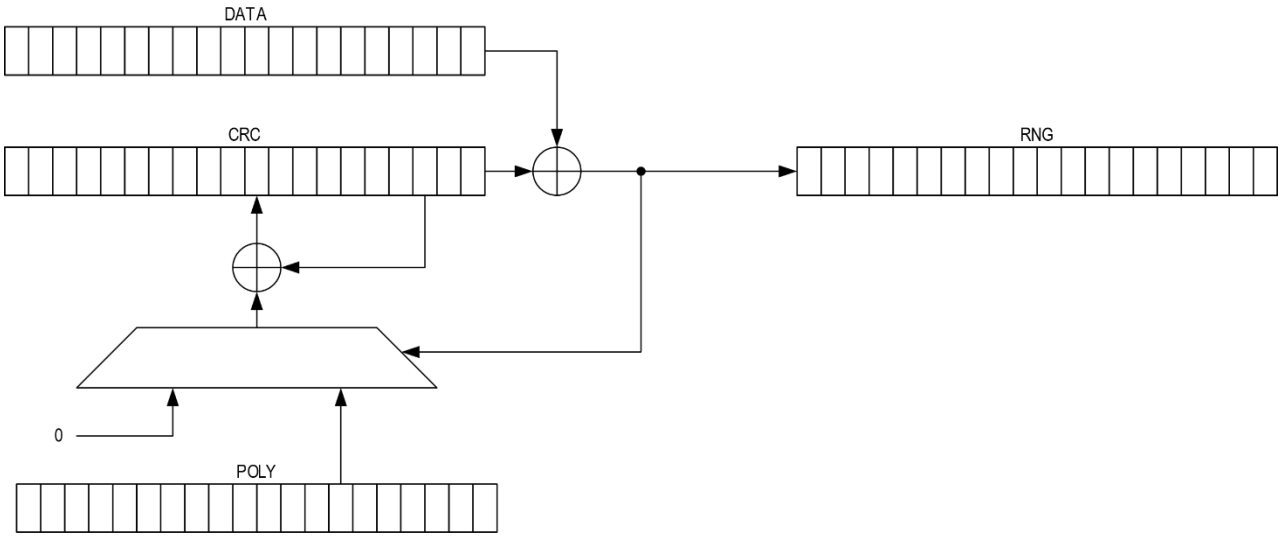
25.7 Instances

There is one instance of this peripheral.

25.8 Linear Feedback Shift Register (LFSR)

Linear Feedback Shift Registers (LFSRs) are commonly used to implement Pseudo-Random Number Generators (PRNGs). An LFSR polynomial can be written to the *TPU_CRC_POLY* register to generate pseudo-random data. The starting state or seed for the pseudo-random sequence should be written to the CRC register. The lockup state of all zeros is detected, and the LFSR substitutes the value 1 to prevent lockup.

Figure 25-6: Galois Field CRC and LFSR Architecture



Different polynomials generate different sequences of random data. Ideally, an n-bit polynomial generates a random sequence of $2^n - 1$ bits. Not all polynomials are maximal length. Some repeat before the theoretical maximum length of $2^n - 1$. There are thousands of different maximal length 32-bit LFSR polynomials. You can use any length of an LFSR polynomial up to 32 bits. Some tables of maximal length LFSR polynomials omit the MSB (x^n) term or the LSB $(x^0 = 1)$ term. Fibonacci LFSRs feed back the XOR of all the taps to the constant term $x^0 = 1$. It is often implied when listing the taps but must be present when writing the polynomial to the *TPU_CRC_POLY* register.

The crypto accelerator automatically generates the next sequence of 32 bits whenever the CRYPTO_LFSR register is read. If the PRNG control bit is set, the incoming data is forced to zero. You can use the DMA to quickly fill a block of memory with pseudo-random data.

25.9 True Random Number Generation

The Analog Devices-supplied Universal Cryptographic Library (UCL), implements a random bit generator that may be compliant with the entropy requirements of commonly used data security validations. The general information in this section is provided only for completeness; Analog Devices will work directly with the customer's chosen security validation laboratory and provide that all the information required for validation directly to that agency.

Entropy is continuously generated by the TRNG. It can be retrieved in groups of 32- and 128 bits through one or four consecutive reads of the 32-bit *TRNG_DATA* register. Hardware sets *TRNG_CN.rng_is* = 1 when 32 bits are available in the TRNG FIFO, and sets *TRNG_CN.rng_4is* = 1 when 128 bits are in the FIFO. Software must set *TRNG_CN.rng_isc* = 1, which clears *TRNG_CN.rng_is* and *TRNG_CN.rng_i4s*, before reading from the TRNG register.

An optional interrupt can be generated if *TRNG_CN.rng_ie* = 1 when hardware sets *TRNG_CN.rng_4is* = 1.

25.10 Registers

Access to specific registers is controlled to prevent software reading from or writing to the registers while they are performing specific functions.

25.10.1 Write Access

The *MAA_CTRL* register can only be written to when the MAA is idle (*MAA_CTRL.stc* = 0). See the bit description for more details on its write access limitation.

The *TPU_CTRL.flag_mode* field determines the method used to clear the *TPU_CTRL.dma_done*, *TPU_CTRL.gls_done*, *TPU_CTRL.hsh_done*, and *TPU_CTRL.cph_done* flags.

Writing to the *MAA_MAWS* or *MAA_CTRL* registers while *MAA_CTRL.stc* = 1 will generate an error and set the *MAA_CTRL.maaer* bit. The current operation will be terminated and the *MAA_CTRL.stc* bit will be cleared.

25.10.2 Read Access

Reading from the MAA memory while *MAA_CTRL.stc* = 0 will return invalid results, but it will not generate an error.

25.11 Registers

See *Table 3-3* for the the base address of this peripheral/module. See *Table 3-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 25-8: TPU Offsets, Register Names, Access, and Descriptions

Offset	Register	Description
0x0000	*TPU_CTRL*	TPU Control Register
0x0004	*CIPHER_CTRL*	Cipher Control Register
0x0008	*HASH_CTRL*	Hash Control Register
0x000C	*CRC_CTRL*	CRC Control Register
0x0010	*TPU_DMA_SRC*	TPU DMA Source Address Register
0x0014	*TPU_DMA_DEST*	TPU DMA Destination Address Register
0x0018	*TPU_DMA_CNT*	TPU DMA Byte Count Register
0x001C	*MAA_CTRL*	MAA Control Register
0x0020	*TPU_DIN_0*	TPU Data In [31:0]
0x0024	*TPU_DIN_1*	TPU Data In [63:32]
0x0028	*TPU_DIN_2*	TPU Data In [95:64]
0x002C	*TPU_DIN_3*	TPU Data In [127:96]
0x0030	*TPU_DOUT_0*	TPU Data Out [31:0]
0x0034	*TPU_DOUT_1*	TPU Data Out [63:32]
0x0038	*TPU_DOUT_2*	TPU Data Out [95:64]
0x003C	*TPU_DOUT_3*	TPU Data Out [127:96]
0x0040	*TPU_CRC_POLY*	TPU Polynomial Register
0x0044	*TPU_CRC_VAL*	TPU Value Register
0x0048	*TPU_CRC_PRNG*	TPU Pseudo-Random Number Register
0x004C	*HAM_ECC*	Hamming ECC Register
0x0050	*CIPHER_INIT_0*	Cipher Initial Vector[31:0]
0x0054	*CIPHER_INIT_1*	Cipher Initial Vector[63:32]
0x0058	*CIPHER_INIT_2*	Cipher Initial Vector[95:64]
0x005C	*CIPHER_INIT_3*	Cipher Initial Vector[127:96]
0x0060	*CIPHER_KEY_0*	Cipher Key [31:0]
0x0064	*CIPHER_KEY_1*	Cipher Key [63:32]
0x0068	*CIPHER_KEY_2*	Cipher Key [95:64]

Offset	Register	Description
0x006C	*CIPHER_KEY_3*	Cipher Key [127:96]
0x0070	*CIPHER_KEY_4*	Cipher Key [159:128]
0x0074	*CIPHER_KEY_5*	Cipher Key [191:160]
0x0078	*CIPHER_KEY_6*	Cipher Key [223:192]
0x007C	*CIPHER_KEY_7*	Cipher Key [255:224]
0x0080	*HASH_DIGEST_0*	Hash Message Digest [31:0]
0x0084	*HASH_DIGEST_1*	Hash Message Digest [63:32]
0x0088	*HASH_DIGEST_2*	Hash Message Digest [95:64]
0x008C	*HASH_DIGEST_3*	Hash Message Digest [127:96]
0x0090	*HASH_DIGEST_4*	Hash Message Digest [159:128]
0x0094	*HASH_DIGEST_5*	Hash Message Digest [191:160]
0x0098	*HASH_DIGEST_6*	Hash Message Digest [223:192]
0x009C	*HASH_DIGEST_7*	Hash Message Digest [255:224]
0x00A0	*HASH_DIGEST_8*	Hash Message Digest [287:256]
0x00A4	*HASH_DIGEST_9*	Hash Message Digest [319:288]
0x00A8	*HASH_DIGEST_10*	Hash Message Digest [351:320]
0x00AC	*HASH_DIGEST_11*	Hash Message Digest [383:352]
0x00B0	*HASH_DIGEST_12*	Hash Message Digest [415:384]
0x00B4	*HASH_DIGEST_13*	Hash Message Digest [447:416]
0x00B8	*HASH_DIGEST_14*	Hash Message Digest [479:448]
0x00BC	*HASH_DIGEST_15*	Hash Message Digest [511:480]
0x00CC	*HASH MSG_SZ_0*	Hash Message Size [31:0]
0x00C4	*HASH MSG_SZ_1*	Hash Message Size [63:32]
0x00C8	*HASH MSG_SZ_2*	Hash Message Size [95:64]
0x00CC	*HASH MSG_SZ_3*	Hash Message Size [127:96]
0x00D0	*MAA_MAWS*	MAA Word Size Register

25.12 Register Details

Table 25-9: TPU Control Register

TPU Control			TPU_CTRL	[0x0000]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31	done	R/W	0	**Cryptographic Operation Done** This bit is set whenever hardware completes an MMA, cipher or hash operation and sets the corresponding “done” bit in TPU_CTRL.[27:25]. This bit remains set until cleared by software. Writing 0 to one or more of the bits in TPU_CTRL.[27:24] will not effect this bit. Setting the *TPU_CTRL.dmanemsk* bit to 1 will cause this bit to be set to 1 when hardware sets the *TPU_CTRL.dma_done* bit. 0: No cryptographic operations have completed since this bit was cleared. 1: One or more cryptographic operations are complete.

TPU Control			TPU_CTRL	[0x0000]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
30	rdy	RO	1	**Cryptographic Block Ready** Hardware clears this status bit to 0 when software initiates a reset of the cryptographic accelerator by setting the *TPU_CTRL.rst*.bit. Software must poll this bit until it is set to 1 by hardware, indicating cryptographic accelerator is again ready for use. 0: Cryptographic accelerator reset in progress. 1: Cryptographic accelerator ready for use
29	err	R	0	**AHB Bus Error** This bit is set if the DMA attempts to access non-existent or protected memory on the AHB bus. This bit can only be cleared by resetting the cryptographic accelerator block. 0: No error 1: Cryptographic accelerator DMA bus error
28	maa_done	R/W	0	**MAA Operation Done** Clear this bit before starting a new MAA operation. This bit is read only while the MAA is in progress. This bit is the opposite of *MAA_CTRL.stc*. 0: MAA operation in progress 1: Last MAA operation done
27	cph_done	R/W	0	**Cipher Done** This bit is set when a block cipher encryption/decryption operation is complete. Clear this bit before starting a cipher operation. 0: Not done 1: Last cipher operation done
26	hsh_done	R/W	0	**Hash Done** This bit is set to 1 when they cryptographic accelerator has completed a SHA operation is complete. Clear this bit before starting the next hash operation. 0: Not done 1: SHA/hash operation done
25	gls_done	R/W	0	**Galois Done** FIFO is full, and CRC or the Hamming Code Generator is enabled. Clear this bit before starting a CRC operation. 0: Not done 1: Operation done
24	dma_done	R/W	0	**DMA Done** DMA write/read operation is complete. Clear this bit before starting a DMA operation. 0: Not done 1: Operation done
23:16	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
15	dmadnemsk	R/W	0	**DMA Done Flag Mask** This field prevents the *TPU_CTRL.dma_done* flag from setting the *TPU_CTRL.done* flag. The *TPU_CTRL.dma_done* flag will still be set. 0: *TPU_CTRL.dma_done* flag will not set *TPU_CTRL.done* 1: *TPU_CTRL.dma_done* flag will also set *TPU_CTRL.done*
14	flag_mode	R/W	0	**Done Flag Mode** This bit provides legacy support for the access behavior of the done flags. It should not be changed from its default value. 0: (Default) Unrestricted write(0 or 1) of *TPU_CTRL*[27:24] 1: Access to *TPU_CTRL*[27:24] is “write 1 to clear/write 0 no effect”
13:12	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.

TPU Control			TPU_CTRL	[0x0000]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
11:10	rdsrc	R/W	0	**Read FIFO Source Select** This field selects the source of the read FIFO data. 0b00: DMA disabled 0b01: DMA or APB 0b10: RNG 0b11: Reserved
9:8	wrsrc	R/W	0	**Write FIFO Source Select** This field determines the source of the write FIFO data. 0b00: None 0b01: Cipher Output 0b10: Read FIFO 0b11: Reserved
7	wait_pol	R/W	0	**Wait Pin Polarity** This feature is not implemented in this device. Do not change this bit from its default value. 0: Active low 1: Active high
6	wait_en	R/W	0	**Wait Pin Enable** This feature is not implemented in this device. Do not change this bit from its default value. 0: Disabled 1: Enabled. CMDA will be halted when the pin is in its active state.
5	bsi	R/W	0	**Byte Swap Input** *Note: No byte swap occurs if there is not a full word.* 0: No effect. 1: Byte swap input.
4	bso	R/W	0	**Byte Swap Output** *Note: No byte swap occurs if there is not a full word.* 0: No effect. 1: Byte swap output.
3	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
2	src	R/W	0	**Source Select** This bit selects the hash function and CRC generator input source. 0: Input FIFO 1: Output FIFO
1	int	R/W	0	**Interrupt Enable** Generates an interrupt when done or error set. 0: Interrupt disabled 1: Interrupt asserted when *TPU_CTRL.done* is set.
0	rst	R/W	0	**Reset Cryptographic Accelerator** Setting this bit initiates an internal reset of the cryptographic accelerator. Software must poll the *TPU_CTRL.rdy* bit to determine when the reset process is complete. All cryptographic internal states and related registers are reset to their default reset values. Control register such as *TPU_CTRL*, *CIPHER_CTRL*, *HASH_CTRL, CRC_CTRL*, MAA_CTRL (with the exception of the STC bit). *HASH MSG_SZ_3:0*, and *MAA_MAWS* retain their values. This bit automatically clears itself after one cycle. 0: No effect 1: Reset cryptographic accelerator

Table 25-10: Cipher Control Register

Cipher Control			CIPHER_CTRL	[0x0004]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:11	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
10:8	mode	R/W	0	**Mode Select** Operating mode of block cipher or memory operation. 0b000: ECB 0b001: CBC 0b010: CFB 0b011: OFB 0b100: CTR Others: Reserved
7	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
6:4	cipher	R/W	0	**Block Cipher Operation Select** Symmetric Block Cipher algorithm selection or memory operation.. Clear these bits before starting any other operation of the cryptographic accelerator. 0b000: Disabled 0b001: AES-128 0b010: AES-192 0b011: AES-256 0b100: DES 0b101: TDEA 0b110: Reserved 0b111: Reserved
3:2	src	R/W	0	**Source of Cipher Key** This indicates the source of the key data to be loaded into *CIPHER_KEY_7:0* registers when the *CIPHER_CTRL.key* bit is set. Each field setting loads 128 bits. The load cipher key operation must be performed twice (once with 0b10 and again with 0b11) to load a full 256-bit key. 0b00: *CIPHER_KEY_7* 0b01: *CIPHER_KEY_7* 0b10: AES_KEY2 0b11: AES_KEY3
1	key	R/W1O	0	**Load Cipher Key Using CMDA** Setting this bit initiates loading of the CIPHER_KEY[7:0] registers with the contents pointed to by *CIPHER_CTRL.src*. The bit must be cleared and the *TPU_CTRL.dma_done* bit after the DMA completes loading the key. 0: NOP 1: Initiate key loading from DMA
0	enc	R/W	0	**Encryption Mode** Select encryption or decryption of block cipher data. 0: Encrypt 1: Decrypt

Table 25-11: Hash Control Register

Hash Control			HASH_CTRL	[0x08]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:6	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.

Hash Control			HASH_CTRL	[0x08]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
5	last	R/W	0	**Last Message Bit** This bit is set along with the *HASH MSG_SZ_3:0* register prior to hashing the last 512 or 1024-bit block of the message data. It allows automatic preprocessing of the last message padding, which includes the trailing bit 1 followed by the respective number of zero bits for the last block size and the message length represented in bytes. 0: No effect 1: Last message data
4:2	hash	R	0	**Hash Function Selection.** Select the hash mode algorithm. Clear these bits before starting any other operation of the cryptographic accelerator. 0b000: Hash disabled 0b001: SHA-1 0b010: SHA-224 0b011: SHA-256 0b100: SHA-384 0b101: SHA-512 0b110: Reserved 0b111: Reserved
1	xor	R/W	-	**XOR IV and Cipher Block** Useful when calculating HMAC to XOR the input pad and output pad. Use the feature to Load Key from CMDA. This bit is automatically cleared by hardware after the DMA completes loading the key. When the DMA operation is done, it sets the appropriate CMDA Done flag. 0: No XOR 1: XOR input with IV
0	init	R/W	0	**Initialize** Load *HASH_DIGEST_0* HASH_DIGEST_[15:0] with the initial hash values based on the message digest size the initial hash values corresponding to the selected hash function. 0: NOP 1: Initialize hash values

Table 25-12: CRC Control Register

CRC Control			CRC_CTRL	[0x000C]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:6	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
5	hrst	R/W	0	**Hamming Reset** Reset the Hamming code ECC generator for the next block. 0: NOP 1: Reset Hamming Register
4	ham	R/W	0	**Hamming Code Enable** Enable Hamming code calculation. 0: Hamming disabled 1: Hamming enabled
3	ent	R/W	0	**Entropy Enable** This feature is not implemented in this device. Do not change this bit from its default value.

CRC Control			CRC_CTRL	[0x000C]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
2	prng	R/W	0	**PRNG Enable** This feature is not implemented in this device. Do not change this bit from its default value.
1	msb	R/W	0	**CRC MSB select** This bit selects the order of calculating CRC on data. 0: LSB data first 1: MSB data first
0	crc	R/W	0	**Cyclic Redundancy Check Enable** This feature is not implemented in this device. Do not change this bit from its default value. 0: CRC disabled 1: CRC enabled

Table 25-13: TPU DMA Source Register

TPU DMA Source			TPU_DMA_SRC	[0x0010]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:0	addr	R/W	0	**DMA Source Address** DMA source address for cryptographic operations.

Table 25-14: TPU DMA Destination Register

TPU DMA Destination			TPU_DMA_DEST	[0x0014]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:0	addr	R/W	0	**DMA Destination Address** DMA destination address for cryptographic operations.

Table 25-15: TPU DMA Count Byte Register

TPU DMA Byte Count			TPU_DMA_CNT	[0x0018]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:0	addr	R/W	0	**DMA Byte Counter** DMA counter for cryptographic operations. Writing a non-zero value to this register initiates DMA-based operations.

Table 25-16: MAA Control Register

MAA Control			MAA_CTRL	[0x001C]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:28	tma	R/W	0	**Temporary MAA Memory Assignment** These bits select the logical cryptographic RAM segment for parameter t.

SETTINGS		SEGMENT
MAWS < 1024	**MAWS ≥ 1024**	
0b0000	0b0000	0
0b0001	0b0010	1
0b0010	0b0100	2
0b0011	0b0110	3
0b0100	0b1000	4
0b0101	N/A	5
0b0111	N/A	6

MAA Control			MAA_CTRL		[0x001C]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**	
			0b1000	N/A	7
			0b1001	N/A	8
			0b1010	N/A	9
27:24	rma	R/W	0	**Result Memory Assignment** These bits select the logical cryptographic RAM segment for parameter r.	
			SETTINGS		**SEGMENT**
			MAWS < 1024	**MAWS ≥ 1024**	
			0b0000	0b0000	0
			0b0001	0b0010	1
			0b0010	0b0100	2
			0b0011	0b0110	3
			0b0100	0b1000	4
			0b0101	N/A	5
			0b0111	N/A	6
			0b1000	N/A	7
			0b1001	N/A	8
			0b1010	N/A	9
23:20	bma	R/W	0	**Multiplicand / Operand B Memory Assignment** These bits select the logical cryptographic RAM segment for parameter b.	
			SETTINGS		**SEGMENT**
			MAWS < 1024	**MAWS ≥ 1024**	
			0b0000	0b0000	0
			0b0001	0b0010	1
			0b0010	0b0100	2
			0b0011	0b0110	3
			0b0100	0b1000	4
			0b0101	N/A	5
			0b0111	N/A	6
			0b1000	N/A	7
			0b1001	N/A	8
			0b1010	N/A	9
19:16	ama	R/W	0	**Multiplier / Operand A Memory Assignment** These bits select the logical cryptographic RAM segment for parameter a.	
			SETTINGS		**SEGMENT**
			MAWS < 1024	**MAWS ≥ 1024**	
			0b0000	0b0000	0
			0b0001	0b0010	1
			0b0010	0b0100	2
			0b0011	0b0110	3
			0b0100	0b1000	4
			0b0101	N/A	5
			0b0111	N/A	6
			0b1000	N/A	7
			0b1001	N/A	8
			0b1010	N/A	9

MAA Control			MAA_CTRL	[0x001C]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
15:14	mms	R/W	0	**Modulus Memory Select** These bits select the starting position of parameter m within logical segment 5. **SETTING** / **OFFSET WITHIN LOGICAL SEGMENT** 0b00 / None 0b01 / 0x0040 0b10 / 0x0080 0b11 / 0x00C0
13:12	ems	R/W	0	**Exponent Memory Select** These bits select the starting position of parameter e within the logical segment specified by ema. **SETTING** / **OFFSET WITHIN LOGICAL SEGMENT** 0b00 / None 0b01 / 0x0040 0b10 / 0x0080 0b11 / 0x00C0
11:10	bms	R/W	0	**Multiplicand B Memory Select** These bits select the starting position of parameter b within the logical segment specified by bma. **SETTING** / **OFFSET WITHIN LOGICAL SEGMENT** 0b00 / None 0b01 / 0x0040 0b10 / 0x0080 0b11 / 0x00C0
9:8	ams	R/W	0	**Multiplier A Memory Select** These bits select the starting position of parameter a within the logical segment specified by ama. **SETTING** / **OFFSET WITHIN LOGICAL SEGMENT** 0b00 / None 0b01 / 0x0040 0b10 / 0x0080 0b11 / 0x00C0
7	maaer	R/W0C	0	**MAA Error** This bit is set by hardware if software writes to the MAA_CTRL or MAA_MAWS when MAA is in progress. This also clears STC and terminates the MAA operation. Once set, it must be cleared by software otherwise no new operation is initiated. 0: No error 1: Error occurs
6:5	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
4	ocalc	R/W	0	**Optimized Calculation Control** Setting this bit skips unnecessary multiply operations after normalizing the exponent are skipped. 0: No optimization 1: Optimize calculation

MAA Control			MAA_CTRL	[0x001C]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
3:1	clc	R/W	0	**Calculation Configuration** These bits select the MAA calculation. 0b000: Modular exponentiation 0b001: Square operation 0b010: Multiplication 0b011: Square followed by a multiplication 0b100: Addition 0b101: Subtraction 0b110: Reserved 0b111: Reserved
0	stc	R/W	0	**Start Calculation** This bit functions as both the control and the status of the MAA. Setting this bit initiates a calculation. It remains set while the calculation is in progress, and is cleared by hardware when the calculation is finished. The bit is cleared by hardware if the *MAA_MAWS.msgsz* value is invalid or the MAAER bit is set. Clearing this bit in software resets the controller to its default state. 0: No operation 1: Start calculation specified by CLC

Table 25-17: TPU Data Input Register

TPU Data Input 0 [31:0]			TPU_DIN_0	[0x0020]
TPU Data Input 1 [63:32]			TPU_DIN_1	[0x0024]
TPU Data Input 2 [95:64]			TPU_DIN_2	[0x0028]
TPU Data Input 3 [127:96]			TPU_DIN_3	[0x002C]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:0	DATA	W	0	**TPU Data Input** These registers form the read FIFO for the CMDA. The endian swap input control bit (*TPU_CTRL.bsi*) affects this register.

Table 25-18: TPU Data Output Register

TPU Data Out 0 [31:0]			TPU_DOUT_0	[0x0030]
TPU Data Out 1 [63:32]			TPU_DOUT_1	[0x0034]
TPU Data Out 2 [95:64]			TPU_DOUT_2	[0x0038]
TPU Data Out 3 [127:96]			TPU_DOUT_3	[0x003C]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:0	data	W	0	**TPU Data Output** These registers form the write FIFO for the CMDA. Data is placed in the lower words of these four registers depending on the algorithm. For block cipher modes, this register holds the result of most recent encryption or decryption operation. These registers are affected by the endian swap bit (*TPU_CTRL.bso*).

Table 25-19: TPU Polynomial Register

TPU Polynomial			TPU_CRC_POLY	[0x0040]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:0	data	W	0xEDB88320	**TPU Polynomial** This register holds the polynomial used for CRC calculations. The reset value of this register is the CRC-32 Ethernet polynomial. This register is affected by the MSB control bit.

Table 25-20: TPU Value Register

TPU Value			TPU_CRC_VAL	[0x0044]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:0	val	R/W	0xFFFFFFFF	**CRC Value** This register holds the result of a CRC calculation. The register will immediately be set to 0x0001 if the invalid value of 0x0000 is detected. This register is affected by the MSB control bit.

Table 25-21: TPU Pseudo-Random Number Register

TPU Pseudo-Random Number			TPU_CRC_PRNG	[0x0048]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:0	prng	R/W	0	**Pseudo Random Value** Output of the Galois Field shift register. This holds the resulting pseudo-random number if entropy is disabled or the true random number if entropy is enabled.

Table 25-22: Hamming Error Correction Code Register

Hamming Error Correction Code			HAM_ECC	[0x004C]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:17	-	RO	-	**Reserved for Future Use** Do not modify this field from its default value.
16	par	R/W	0	**Parity** This is the parity of the entire array. 0: Even parity 1: Odd parity
15:0	ecc	R/W	0	**Hamming ECC Value** These bits are the even parity of their corresponding bit groups

Table 25-23: Cipher Initial Vector Register [3:0]

Cipher Initial Vector [31:0]			CIPHER_INIT_0	[0x0050]
Cipher Initial Vector [63:32]			CIPHER_INIT_1	[0x0054]
Cipher Initial Vector [95:64]			CIPHER_INIT_2	[0x0058]
Cipher Initial Vector [127:96]			CIPHER_INIT_3	[0x005C]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:0	ivec	R/W	0	**Block Cipher Initial Vector** These registers hold the initial value for cipher operations that use CBC, CFB, OFB, or CNTR modes. This register is updated with each encryption or decryption operation. This register is affected by the endian swap bits.

Table 25-24: Cipher Key Register [7:0]

Cipher Key 0 [31:0]	**CIPHER_KEY_0**	**[0x0060]**
Cipher Key 1 [63:32]	**CIPHER_KEY_1**	**[0x0064]**
Cipher Key 2 [95:64]	**CIPHER_KEY_2**	**[0x0068]**
Cipher Key 3 [127:96]	**CIPHER_KEY_3**	**[0x006C]**
Cipher Key 4 [159:128]	**CIPHER_KEY_4**	**[0x0070]**
Cipher Key 5 [191:160]	**CIPHER_KEY_5**	**[0x0074]**
Cipher Key 6 [223:192]	**CIPHER_KEY_6**	**[0x0078]**
Cipher Key 7 [255:224]	**CIPHER_KEY_7**	**[0x007C]**

BITS	NAME	ACCESS	RESET	DESCRIPTION
31:0	key	R/W	0	**Cipher Key** These registers hold the cipher key used for block cipher operations. The number of bits used depends on the specific operation. See the *CIPHER_CTRL.key* field for information on loading this register. This register is affected by the endian swap input control bits.

Table 25-25: HASH Message Digest Register [15:0]

Hash Message Digest 0 [31:0]	**HASH_DIGEST_0**	**[0x0080]**
Hash Message Digest 1 [63:32]	**HASH_DIGEST_1**	**[0x0084]**
Hash Message Digest 2 [95:64]	**HASH_DIGEST_2**	**[0x0088]**
Hash Message Digest 3 [127:96]	**HASH_DIGEST_3**	**[0x008C]**
Hash Message Digest 4 [159:128]	**HASH_DIGEST_4**	**[0x0090]**
Hash Message Digest 5 [191:160]	**HASH_DIGEST_5**	**[0x0094]**
Hash Message Digest 6 [223:192]	**HASH_DIGEST_6**	**[0x0098]**
Hash Message Digest 7 [255:224]	**HASH_DIGEST_7**	**[0x009C]**
Hash Message Digest 8 [287:256]	**HASH_DIGEST_8**	**[0x00A0]**
Hash Message Digest 9 [319:288]	**HASH_DIGEST_9**	**[0x00A4]**
Hash Message Digest 10 [351:320]	**HASH_DIGEST_10**	**[0x00A8]**
Hash Message Digest 11 [383:352]	**HASH_DIGEST_11**	**[0x00AC]**
Hash Message Digest 12 [415:384]	**HASH_DIGEST_12**	**[0x00B0]**
Hash Message Digest 13 [447:416]	**HASH_DIGEST_13**	**[0x00B4]**
Hash Message Digest 14 [479:448]	**HASH_DIGEST_14**	**[0x00B8]**
Hash Message Digest 15 [511:480]	**HASH_DIGEST_15**	**[0x00BC]**

BITS	NAME	ACCESS	RESET	DESCRIPTION
31:0	hash	R/W	0	**Calculated Hash Value** These 16 registers hold the 512-bit calculated hash value. This register is affected by the endian swap bits.

Table 25-26: Hash Message Size Registers

Hash Message Size 0 [31:0]	**HASH MSG_SZ_0**	**[0x00C0]**
Hash Message Size 1 [63:32]	**HASH MSG_SZ_1**	**[0x00C4]**
Hash Message Size 2 [95:64]	**HASH MSG_SZ_2**	**[0x00C8]**
Hash Message Size 3 [127:96]	**HASH MSG_SZ_3**	**[0x00CC]**

BITS	NAME	ACCESS	RESET	DESCRIPTION
31:0	msgsz	R/W	0	**Hash Message Size** These four registers hold the 128-bit message size in bytes.

Table 25-27: MAA Word Size Register

MAA Word Size			MAA_MAWS	[0x00D0]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:12	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
11:0	msgsz	R/W	0	**MAA Word Size** This register defines the number of bits for a modular operation. Valid values are from 1 to 2048. Invalid values are ignored and do not initiate a MAA operation. You can only write to this register when *MAA_CTRL.stc* = 0 (MAA idle).

Table 25-28: TRNG Control Register

TRNG Control			TRNG_CN	[0x0000]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:7	-	RO	1	**Reserved for Future Use** Do not modify this field from its default value.
6	aeskg	R/W	0	**AES Key Generate** When enabled, the key for securing NVSRAM is generated and transferred to the secure key register automatically without user visibility or intervention. This bit is cleared by hardware once the key has been transferred to the secure key register.
5	rng_is	RO	0	**Random Number Ready Status** This bit is set when a new 32 bit random number is available in *TRNG_DATA*. This bit is cleared by hardware if all the random words have been read. It is needed to poll this bit before reading the TRNG Data Register
4	rng_i4s	RO	0	**128-bit Random Number Ready Status** This bit is set when a new 128 bit random number is ready to be read (using 4 consecutive reads of *TRNG_DATA*. When set, an interrupt will be generated if *TRNG_CN.rng_ie = 1*. This bit is cleared by setting *TRNG_CN.rng_isc*.
3	rng_isc	W	0	**Random Number Interrupt Status Clear** Setting this bit to 1 clears *TRNG_CN.rng_i4s* and acknowledges the interrupt, if enabled. This it is a write only bit and always reads as zero.
2	rng_ie	R/W	0	**Random Number Interrupt Enable** This bit enables an interrupt to be generated when *TRNG_CN.rng_i4s* = 1.
1:0	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.

Table 25-29: TRNG Data Register

MAA Word Size			TRNG_DATA	[0x0004]
BITS	**NAME**	**ACCESS**	**RESET**	**DESCRIPTION**
31:0	data	RO	0	**TRNG Data** The function of this register is dependent on the rng_is and rng_i4s bits

26. Secure Communication Protocol Bootloader (SCPBL)

The security of the secure boot and SCP communication relies on public key infrastructure (PKI) using a manufacturer root key (MRK) authority, a customer root key (CRK), and a certificate. The certificate is a signed CRK using the MRK. Both the signed CRK and MRK values are stored in nonvolatile memory for use by the SCPBL.

The SCPBL authenticates SCP transactions and verifies the integrity of internal program memory using digital signatures based on the MRK and CRK. SCP packets are signed at the session level to authenticate the sender and verify the integrity of the communication. After every reset, the secure boot verifies the digital signature of the downloaded program in memory to ensure it is not altered.

The features described in this chapter only apply to versions of the device with the secure boot or secure bootloader described in the data sheet Ordering Information table.

26.1 Development Tools

The information in this chapter is provided to customers who wish to understand details of the SCP protocol and the secure boot process. However, Analog Devices provides a software development kit (SDK) that builds a complete application image, digitally signs it, and creates the packets used by the SCP. These tools automatically implement most of the low-level functions described in this document.

The evaluation kit for the device comes preloaded with a developer/test CRK (DCRK) and demonstration software.

26.2 Instances

There is only one instance of the SCPBL.

The features of the SCPBL are in *Table 26-1*.

Table 26-1: MAX32666 Bootloader Instances

Part	SCP Bootloader	Secure Boot	SCPBL Interfaces UART0	Digital Signature Options
MAX32666	Yes	Yes	UART0 USB	ECDSA-256

Table 26-2 shows the interfaces associated with the SCPBL.

Table 26-2: MAX32666 Bootloader Physical Interfaces

Part	Interface	Default Stimulus Pin	Interface Pins Required For SCPBL	Stimulus Pin Reassignment Options	Timeout Period/ Conditions
MAX32666	UART (default 115200bps, 8-N-1)	No default stimulus pin. If a stimulus pin was not previously defined by the *WRITE_STIMULUS* command, the part listens on UART0 for 6s before attempting to boot the application.	UART0_RX (P0.10) UART0_TX (P0.9)	All GPIO except: UART0_RX UART0_TX UART0_CTS UART0_TX	20s (fixed)
	USB	None. Must be first configured by the *WRITE_STIMULUS* command through the UART0 interface.	DP DM	All GPIO except: UART0_RX UART0_TX UART0_CTS UART0_TX	20s (fixed)

The SCPBL stimulus pin is tested after the events shown in *Table 26-3*.

Table 26-3: Bootloader Stimulus Test Events

Event	Stimulus Tested?
Peripheral Reset	Yes
Soft Reset	Yes
System Reset	Yes
External Reset	Yes
POR	Yes
External Reset	Yes
Watchdog Reset	Yes
Exit from SLEEP Mode	N
Exit from DEEPSLEEP Mode	N
Exit from BACKUP Mode	Yes

26.3 Secure Product Life Cycle Management

26.3.1 Purpose

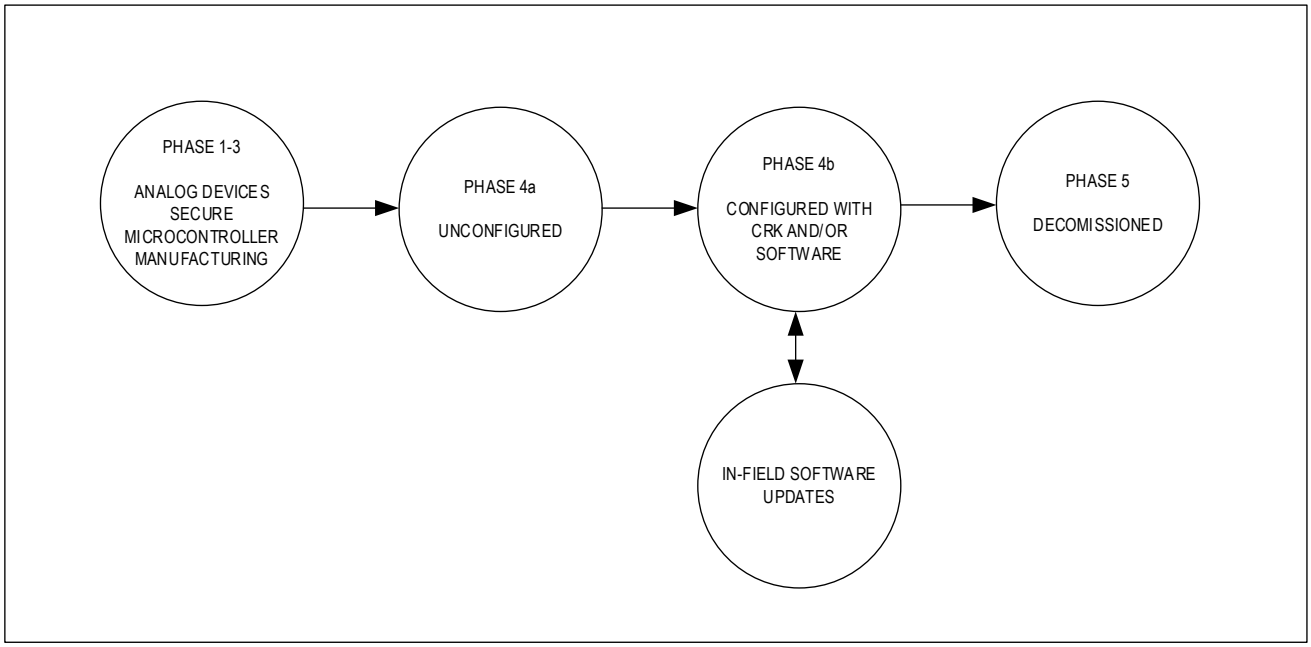
The secure product life cycle scheme implemented by the SCPBL protects the security of the product in multiple ways:

1. It verifies all the security measures on a device before being released to the customer.
2. It prevents a device from properly operating until configured by the customer.
3. It enables device programming in an unsecured environment (manufacturing location or in-field updates).
4. It provides end-of-life decommissioning to control the application life cycle securely.

26.3.2 Life Cycle Phases

The device supports multiple life cycle phases, as shown in *Figure 26-1*.

Figure 26-1: Life Cycle Phases



The phases are detailed below.

1. Phases 1–3. These phases are executed in a controlled manufacturing facility to guarantee that the device is secure, protocol-controlled before delivery to the customer.
2. Phase 4a. The device is not loaded with the CRK. Devices are delivered to the customer in this state.
3. Phase 4b. The device transitions to this phase after the customer has loaded the CRK signed by Analog Devices. At this point, a properly signed application image can be loaded for development and testing. The device is deployed in this state and remains in this phase throughout the product's active life. The SCPBL is active, allowing in-the-field software updates, and protecting against tampering and unauthorized access.
4. Phase 5: The device transitions to this phase using a dedicated command. Previously loaded software is not exectued, and program memory cannot be loaded or verified.

The life-cycle phase is read with the HELLO_REPLY command, which provides status information on the device.

26.4 MAX32666 Bootloader Activation

Two conditions must be met following a bootloader stimulus test event listed in *Table 26-3* to activate the bootloader:

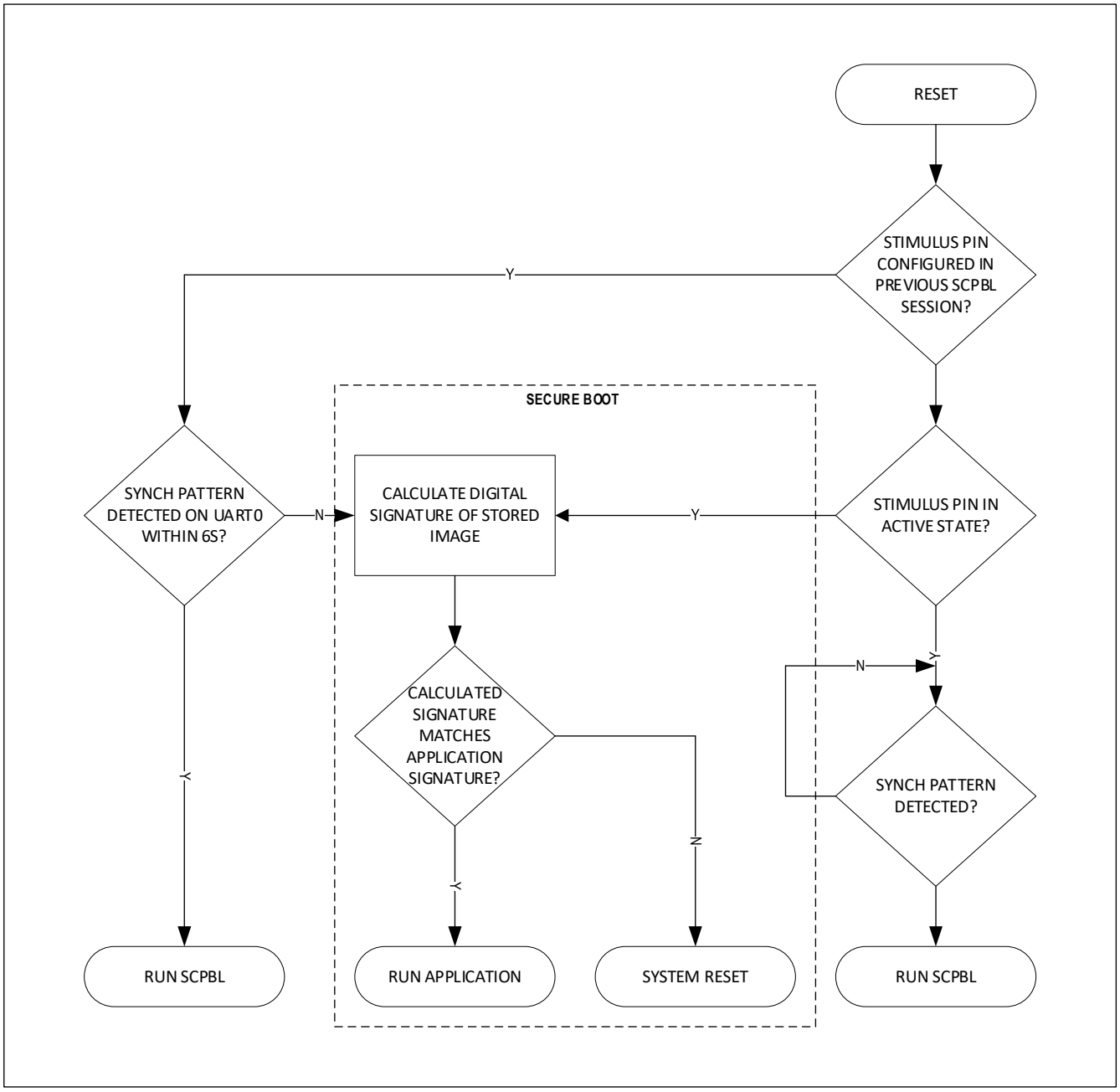
- Assertion of the stimulus pin
- Detection of the synchronization pattern on the active interface.

Unlike other SCPBL instances, this device does not have a default stimulus pin. Following a bootloader stimulus test event listed in *Table 26-3*, the device checks to see if a stimulus pin has been explicitly assigned in a previous SCPBL session using the *WRITE_STIMULUS* command. If no stimulus pin is assigned yet, the device monitors UART0 to detect the SYNCH pattern. If the pattern is not seen within 6s, the device performs the secure boot sequence and attempts to execute the software.

If a stimulus pin has been assigned, the device tests the stimulus pin once after a bootloader stimulus test event. If the stimulus pin is not asserted, the device performs the secure boot sequence and attempts to execute the application code. If the stimulus pin is asserted, the device will monitor the assigned interface for the SYNCH pattern in perpetuity and will never time out.

Because the device's GPIO pins default to a high-impedance input, the application must externally drive the stimulus pin or pins to the desired state. The stimulus pin must be held in its active state until the host terminates the session.

Figure 26-2: MAX32666 Bootloader Activation Flow



The format of the UART data is 115200 bps, 8 data bits, no parity, and one stop bit. Although this device transmits one stop bit, the host must be configured to send two stop bits for proper operation of the bootloader. The specific errata sheet for each revision describes the issue.

26.5 Secure Boot

If the stimulus pin is not active following a reset, the device executes a secure boot from the ROM to verify the integrity of the code in flash. The secure boot establishes the chain of trust that a secure application execution relies on:

- The platform knows the application software has not been accidentally or maliciously modified.
- The platform knows the application software is from a trusted, authorized source.
- The application software ensures it is running on a trusted platform.

Initialize a device with the secure boot feature before it executes the application software:

- The CRK must be loaded.
- The application image must be created and signed with the CRK.
- Load the application image through the SCPBL.

The device calculates the digital signature of the application image in flash before executing any code in flash. If the calculated value matches the digital signature stored at the end of the application image, the device begins the execution of the application software code.

If the calculated value does not match the digital signature, the device halts before executing any code. The secure boot calculation is always incorrect before the installation of the correct.

26.6 Root Key Management

The root of trust is a public-key infrastructure involving several key pairs. *Table 26-1* shows the key type for each device.

- The manufacturer root key (MRK) pair
- The customer root key (CRK) pair
- The developer or test customer rook key (DCRK) pair

26.6.1 Manufacturer Root Key (MRK)

The factory owns the MRK pair. The public key is used for digital signature verification and is stored in nonvolatile memory. It is used to sign the CRK. This certificate authenticates and identifies the source.

Analog Devices secures the MRK private key in a restricted-access hardware security module (HSM) compliant with most security requirements.

26.6.2 Customer Root Key (CRK)

The CRK is used to sign production-ready end products. The customer generates their own CRK keypair using (typically) PCI-PTS compliant tools. The public key used for the digital signature is securely sent to the factory and is signed by the MRK. Refer to Application Note 7494 *Secure Information Exchange and CRK Certification Guide* for details of the key exchange and signing procedures.

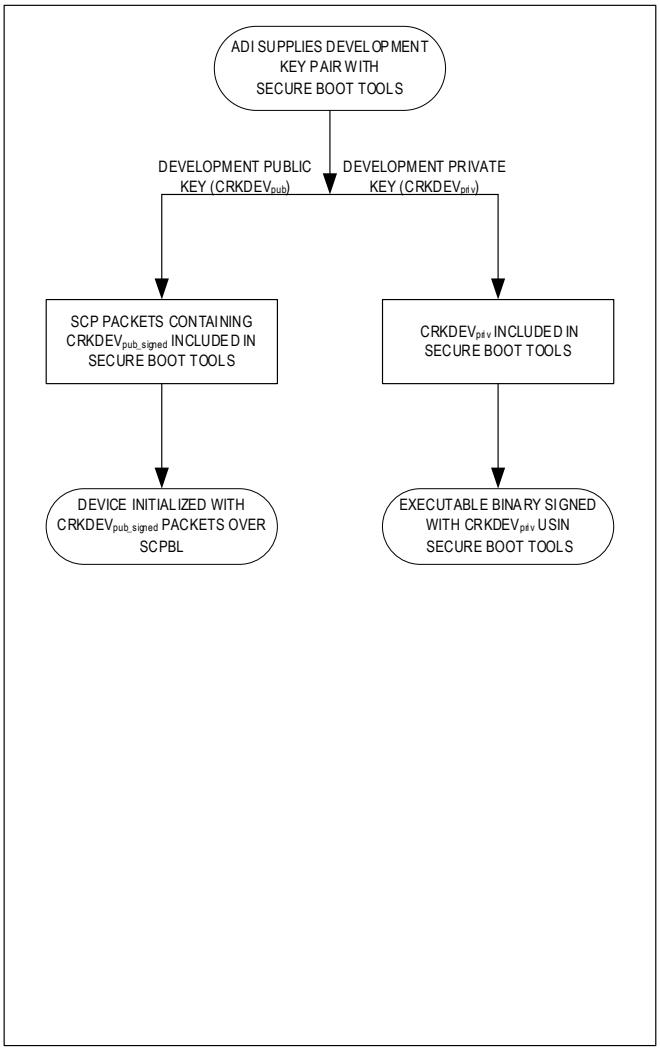
The customer must protect the private part of its key pair. The chain of trust cannot be guaranteed if this key is compromised. The customer should use an HSM or equivalent hardware device to provide physical protection to the ECDSA and strong key protection requirements.

26.6.3 Developer/Test CRK (DCRK)

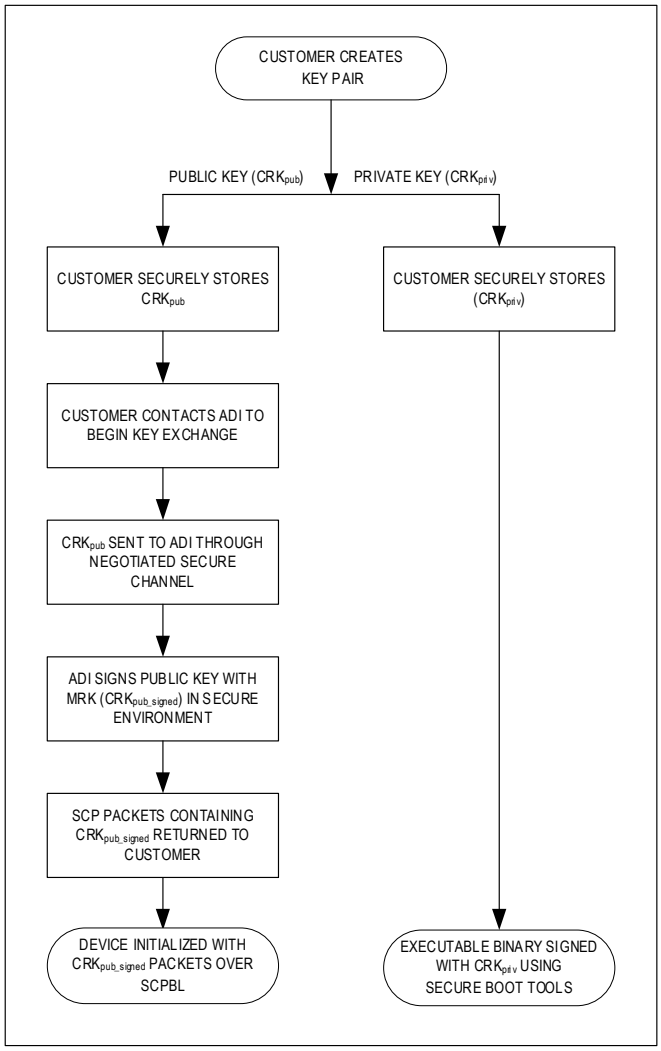
The SDK provides a particular version of the CRK, called the developer or test root key, used during development without generating a custom CRK and having it signed by the manufacturer. The development key is common to all customers and is not secure. A customer must load devices with a signed CRK before deployment to ensure the security of the end-customer product. Because it is a version of the CRK, a second key can be loaded once with the *REWRITE_CRK* command.

The evaluation kits for the device come preloaded with the developer or test root key.

Figure 26-3: Customer Root and Development Key Generation and Usage



KEY MANAGEMENT USING ANALOG DEVICES-SUPPLIED DEVELOPMENT KEY



KEY MANAGEMENT USING ANALOG DEVICES-SUPPLIED CUSTOMER ROOT KEY

26.7 MAX32666 Checksum/Signature Options

There are four fields in an SCP transaction that provide integrity and authentication of the data:

- Application image signature
- Transport layer header checksum
- Transport layer data checksum
- Session layer signature

The options for these fields appear in *Table 26-4*. The application digital signature and session-layer profile encryption types should be the same.

Table 26-4: MAX32666 Supported Checksum/Signatures

Element	Field Coding
Application Image Digital Signature	0x45: ECDSA-256 0x46: RSA-2048 (deprecated) 0x47: RSA-4096 (deprecated)
Header Checksum	(All SCP versions) The 1-byte header checksum is generated from AES-128 encryption with a 16-byte key of 0x00. The data input is the first 7 bytes of the transport header, then padded with 9 bytes of 0x00 to form the required minimum 16-byte input value for AES. The calculated header checksum is the MSB of the 16-byte digest.
Data Checksum	(All SCP versions) The 1-byte header checksum is generated from AES-128 encryption with a 16-byte key of 0x00. The data input is the first 7 bytes of the transport header, then padded with 9 bytes of 0x00 to form the required minimum 16-byte input value for AES. The calculated header checksum is the 4 MSB of the 16-byte digest.
Session Layer Protection Profile	0x0: none 0x8: RSA-4096 (deprecated) 0x9: RSA-2048 (deprecated) 0xA: ECDSA-256
MRK/CRK	These are based on the selected signature type.

26.8 Building the MAX32666 Application Image

The application image gets physically programmed into program memory, regardless of the transmission protocol.

The application image has the specific format shown in *Figure 26-4*.

Figure 26-4: Application Image

The application image contains multiple parts:

- Application image header, containing information about the length of the execution binary and identification of the cryptographic protection method
- Executable binary, which is the compiled customer code
- A digital signature or checksum to verify the integrity and/or authenticate the application image header and the executable binary
- Padding as needed to ensure that the start and end of the executable binary aligns with the required boundaries

The image elements appear in *Table 26-5*. All values are fixed, except where noted.

Table 26-5: Application Image Structure

Element	Absolute Start Address within Application Image	Length (Bytes)	Value
Synchronization Pattern	0x0000 0000	8	ECDSA-256: 0x48,0x49,0x53,0x57,0x45,0x44,0x47,0x44 RSA-2048: 0x46,0x49,0x53,0x57,0x45,0x44,0x47,0x44 RSA-4096: 0x47,0x49,0x53,0x57,0x45,0x44,0x47,0x44 Note: The RSA-2048 and RSA-4096 encryption types have been deprecated.
Format Version (ROM_REF)	0x0000 0008	4	0x0100 0000 (default defined by device version)
Application Image Start Address	0x0000 000C	4	0x1000 0000
Application Image Signature Offset	0x0000 0010	32 bits	This is the start address of the digital signature. The length is calculated from the application image start address to the end of the padding (if necessary) after the binary executable. It does not include the length of the digital signature.
Executable Binary Offset	0x0000 0014	32 bits	0x0000 0400 (fixed)
Argument Size	0x0000 0018	32 bits	0x0000 0000 (fixed)
Application Version (Corresponding to SLA_VERSION Command)	0x0000 001C	32 bits	User-defined (default convention used by the SBT is 0x01000000 corresponding to Version 1.0.0)
Padding	0x0000 0020	992	Padded with 0xFF out to 0x3FFF
Executable Binary	0x0000 0400	Var	Executable binary
Terminal Padding to 4B Boundary if Necessary	End of executable binary	0 1B 2B 3B	N/A 0xFF 0xFFFF 0xFFFFFF
Application Image Signature	End of executable binary + terminal padding	256 bits	ECDSA-256 signature

26.9 Selecting the Programming Interface

Product development and debugging are often performed over the simpler, faster JTAG interface. After development, the initial code can be programmed in a secure customer facility.

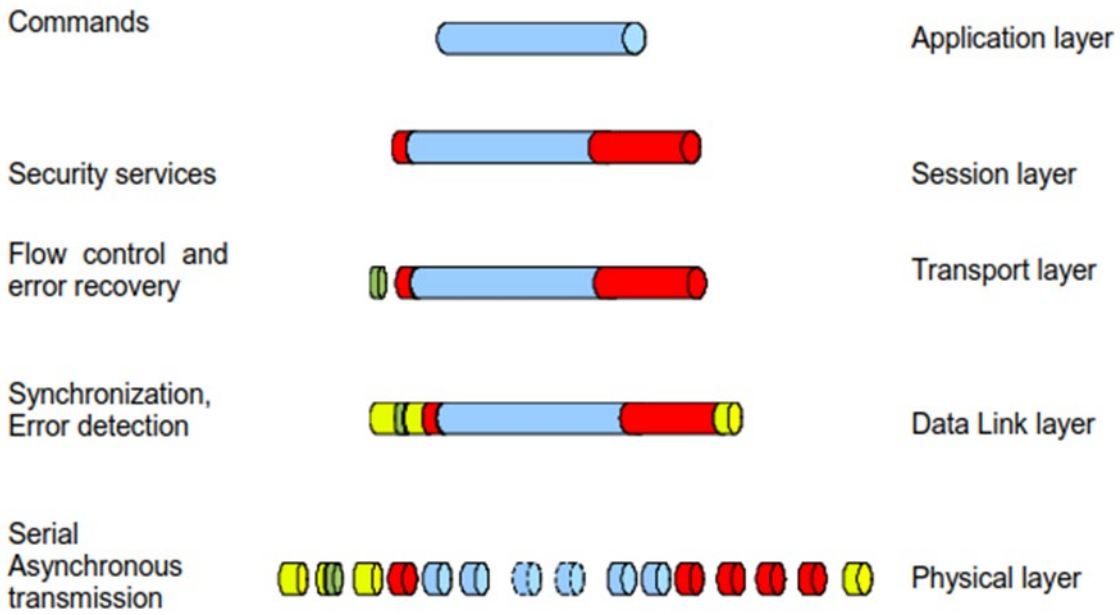
The SCPBL is most often used after a product has been deployed to the field for software updates. This ensures trusted and authenticated communication in an insecure environment.

26.10 SCP Session

The SCP is a session-based protocol that securely exchanges data packets between the host and the SCPBL.

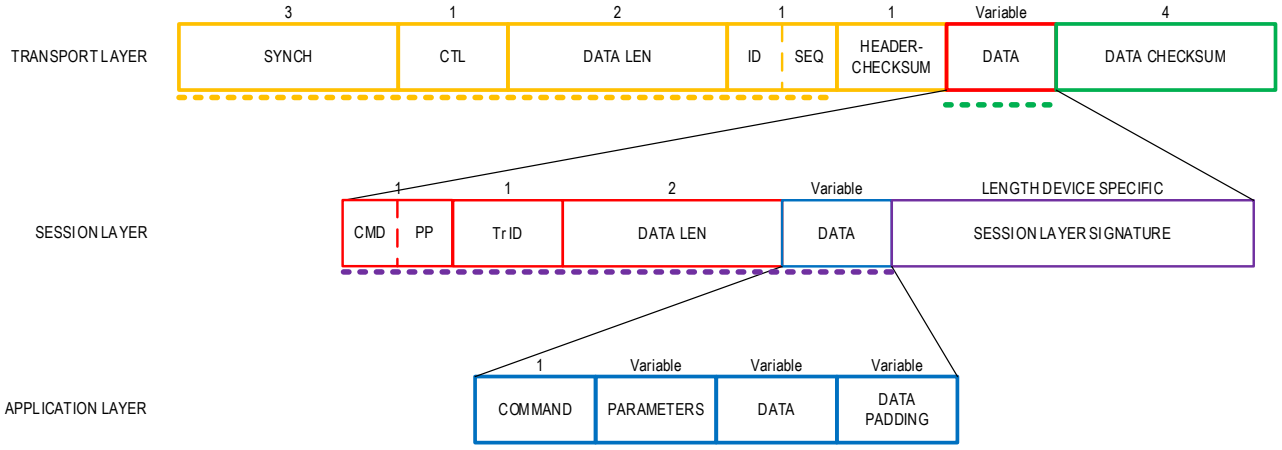
The SCP is a stack of layers based on the OSI model shown in *Figure 26-5*. The application layer contains basic commands to configure the SCPBL. It also includes the signed customer application and a set of WRITE data commands used for the SCPBL, including the write command that loads the customer code (the executable binary) into program memory.

Figure 26-5: SCPBL Implementation of OSI Model



The general structure of a packet is shown in *Figure 26-6*. Simpler commands only require a subset of these elements.

Figure 26-6: SCP Packet Structure



26.10.1 Physical Layer

At this layer, data between the SCPBL and the host is transferred over the selected communication interface shown in *Table 26-6*. The stream of data is subdivided into Protocol Data Units (PDUs) called bytes. When a framing error occurs, the data byte is discarded.

26.10.2 Data Link Layer

The Data Link Layer accumulates bytes from the Physical layer into a Protocol Data Unit (PDU) called a frame (logical, structured packets for data) to ensure the data integrity.

Each frame contains a header that identifies the type of packet shown in yellow in *Figure 26-6*, followed by an optional data portion.

Table 26-6: Transport/Session Layer Header Structure

Segment	Offset	Length	Value
Synchronization Pattern	0x0000 0000	3B	0xBEEFED (fixed)
Segment Type (Control)	0x0000 0002	1B	Data link segment type
Data Length	0x0000 0003	1B	Data link segment length
Channel ID	0x0000 0003	Upper 4 bits	The channel identifier is provided by the host to identify an active connection and must be the same in both ingoing and outgoing segments. This "channel identifier" is fixed for the whole session.
Sequence Number		Lower 4 bits	The lower 4 bits of this byte serve as an acknowledgment from the SCPBL that the last segment was received as described in the appropriate section. This field is incremented by one modulo 16 by the sender for each new data segment (a segment represents each data exchange one way, i.e., an ACK is not a new segment). The "sequence number" initial value is 0, used for the first data exchange after opening the session (i.e., after the HELLO-REPLY).
Header Checksum	0x0000 0004	1B	An AES-CRC hash with an initial key of 0x00 is performed on the preceding 7 segments plus 9 bytes of 0x00 to make the required 16B. This field contains the LSB of the 4-byte result.

26.10.3 Transport Layer

The transport layer provides security, data recovery, and flow control. It establishes, manages, and terminates connections with clear phases of link establishment, information transfer, and link termination. This layer implements a reliable message service through several mechanisms:

- The sequential numbering of the segments,
- Positive acknowledgment (ACK command) of command receipt
- Timeout and a retry strategy

26.10.4 Session Layer

The session layer manages security issues by encrypting and signing the data. The PDU is called Data at this layer.

Data has a variable length.

26.11 Sequence and Transaction ID

The SCP integrates two sequencing parameters as error-detection mechanisms.

The parameters are:

- The Session ID in the header of the transport layer
- The Transaction ID in the header of the session layer

The SCPBL automatically increments its internal counter following the ACK of specific data transfer commands. The host software, following the same rules, must increment its internal counter simultaneously to remain synchronized. An unexpected sequence number indicates an error in the SCP protocol, and the SCPBL terminates the correction to prevent the communication of corrupted or compromised data.

The host and SCPBL both increment their sequence numbers after an ACK command that follows these commands:

- DATA_TRANSFER
- HELLO
- HELLO_RSP
- ECHO_RSP

The Transaction ID increments at the session level after each successful data transfer from the host to the SCPBL.

26.12 Opening an SCP Session

The host initiates a new session request by sending a CON_REQ to the SCPBL. The procedure is in *Table 26-7*. An error is generated if the host sends a CON_REQ command while a session is already in progress.

Table 26-7: Session Opening Protocol

HOST	Communication Direction	SCPBL	Session Sequence	Transaction ID
Session Closed				
Connection Request			0	N/A
	CON_REQ →		0	N/A
		Connection Accepted	0	N/A
	← CON_REP		0	N/A
Valid Command			0	N/A
	ACK →		0	N/A
Connection Confirmed			0	N/A
HELLO			0	N/A
	HELLO →		0	N/A
		Valid Command	0	N/A
	← ACK		1	N/A
		HELLO Accepted SEQ = 1	1	N/A
	← HELLO_REPLY		1	N/A
Valid Command			1	N/A
	ACK →		2	N/A
Session Open				

26.13 Transport/Session Layer Commands

Table 26-8: Transport/Session Layer Command Summary

Command	CTL	Description
CON_REQ	0x01	A connection request from the host to the SCPBL
CON_REP	0x02	A connection acceptance from the SCPBL to the host
DISC_REQ	0x03	A disconnection request from the host to the SCPBL
DISC_REP	0x04	A disconnection acceptance from SCPBL to the host
ACK	0x06	Transport command acknowledgment
HELLO	0x05/0x1	Status/information request from the host
HELLO_REPLY	0x05/0x2	Status/information reply from the SCPBL
DATA_TRANSFER Commmands	0x05/varies	Includes all application layer commands *Table 26-8*.
ACK	0x06	Command acknowledgment between the host to the SCPBL and the SCPBL to the host
ECHO_REQ	0x0B	Echo request
ECHO_REP	0x0C	Echo reply

26.14 Transport/Session Layer Command Details

26.14.1 CON_REQ

The host sends a CON_REQ to the SCPBL to initiate a connection.

The sequence number is always 0b0000 for this command.

The session ID encoded in this command becomes the session ID for all SCP transactions.

This command does not have a Data element.

Figure 26-7: CON_REQ Command Structure

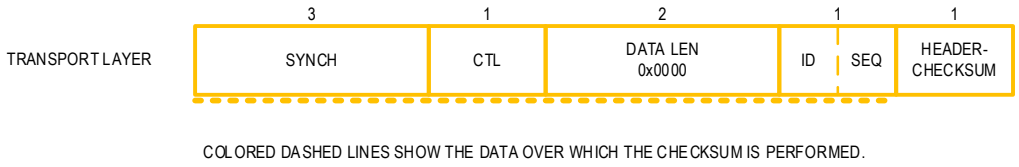


Table 26-9: CON_REQ

CON_REQ	0x01	Connection Request	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	**Synchronization Pattern** 0xBEEFED (fixed)
CTL	0x03	1	**Command Code** 0x01 (fixed)
DATA LEN	0x04	2	**Length of Data Segment** 0x0000 (fixed)
ID	0x06	4 bits	**Channel ID (upper 4 bits)** The Channel ID is user-defined. After being defined in this command, the same value is used for all commands within a session.
SEQ		4 bits	**Sequence Number** 0b000 (fixed)
HEADER CHECKSUM	0x07	1	**Header Checksum**

26.14.2 CON_REP

The SCPBL sends a confirmation to the host in response to a CON_REQ command.

The host responds with an ACK. The ACK response to this command does not increment the SEQ field.

The sequence number is 0b0000 for this command since this is always the first command used to start a session.

This command does not have a Data element.

Figure 26-8: CON_REP Command Structure

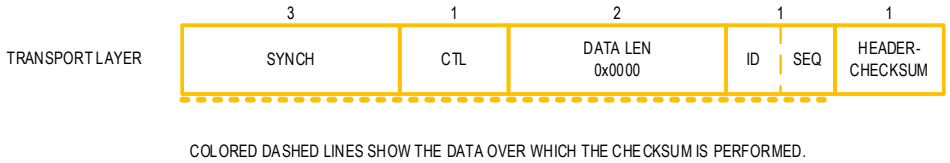


Table 26-10: CON_REP

CON_REP	0x02	Connection Reply	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	**Synchronization Pattern** 0xBEEFED (fixed)
CTL	0x03	1	**Command Code** 0x02 (fixed)
DATA LEN	0x04	2	**Length of Data Segment** 0x0000 (fixed)
ID	0x06	4 bits	**Channel ID (upper 4 bits)** Defined in the CON_REQ command
SEQ		4 bits	**Sequence Number** 0b000 (fixed)
HEADER CHECKSUM	0x07	1	**Header Checksum**

26.14.3 DISC_REQ

The host sends a DISC_REQ to the SCPBL to terminate the SCP session. The SCPBL responds with an ACK.

This command does not change the sequence number.

This command does not have a Data element.

Figure 26-9: DISC_REQ Command Structure

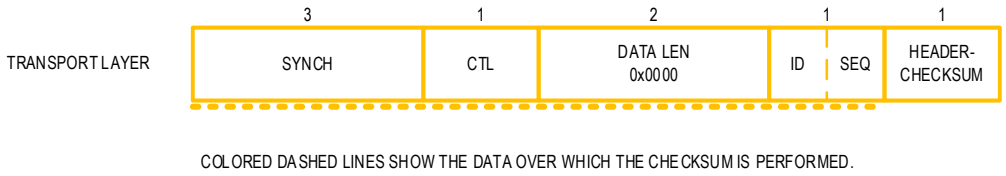


Table 26-11: DISC_REQ

DISC_REQ	0x03	Connection Request	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	**Synchronization Pattern** 0xBEEFED (fixed)
CTL	0x03	1	**Command Code** 0x03 (fixed)
DATA LEN	0x04	2	**Length of Data Segment** 0x0000 (fixed)
ID	0x06	4 bits	**Channel ID (upper 4 bits)** Defined in the CON_REQ command
SEQ		4 bits	**Sequence Number**
HEADER CHECKSUM	0x07	1	**Header Checksum**

26.14.4 DISC_REP

The SCPBL sends a confirmation response to the host in response to a DISC_REQ command. The host responds with an ACK. The bootloader session terminates, and the device performs a reset. The device reenters the SCPBL if the stimulus pin is asserted.

This command does not change the sequence number.

This command does not have a Data element.

Figure 26-10: DISC_REP Command Structure

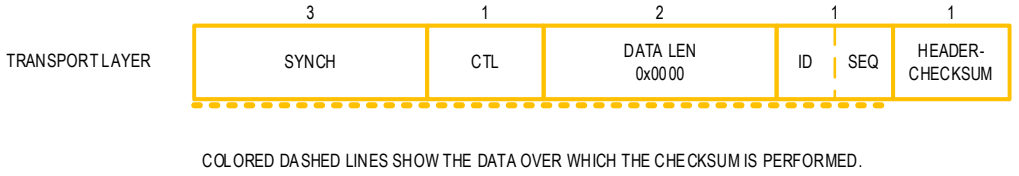


Table 26-12: DISC_REP

DISC_REQ	0x04	Connection Request	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	**Synchronization Pattern** 0xBEEFED (fixed)
CTL	0x03	1	**Command Code** 0x04 (fixed)
DATA LEN	0x04	2	**Length of Data Segment** 0x0000 (fixed)
ID	0x06	4 bits	**Channel ID (upper 4 bits)** Defined in the CON_REQ command
SEQ		4 bits	**Sequence Number** 0b000 (fixed)
HEADER CHECKSUM	0x07	1	**Header Checksum**

26.14.5 ACK

The SCPBL and host each send out an acknowledgment packet to confirm the receipt of any valid command. The ACK command has the same ID and sequence number as the command it is acknowledging.

This command does not have a Data element.

Figure 26-11: ACK Command Structure

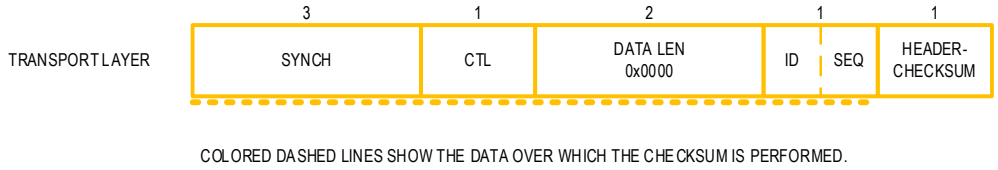


Table 26-13: ACK

ACK	0x06	Connection Request	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	**Synchronization Pattern** 0xBEEFED (fixed)
CTL	0x03	1	**Command Code** 0x06 (fixed)
DATA LEN	0x04	2	**Length of Data Segment** 0x0000 (fixed)
ID	0x06	4 bits	**Channel ID (upper 4 bits)** Defined in the CON_REQ command
SEQ		4 bits	**Sequence Number (lower 4 bits)** Same as the command that it is acknowledging.
HEADER CHECKSUM	0x07	1	**Header Checksum**

26.14.6 HELLO

The host sends this command as part of the sequence to establish a new session. The command is a unique version of the DATA_TRANSFER command with a specific payload. The SPCBL responds with ACK followed by HELLO_REPLY.

The HELLO command does not include a signature following the session layer data like other DATA_TRANSFER commands.

Figure 26-12: HELLO Command Structure

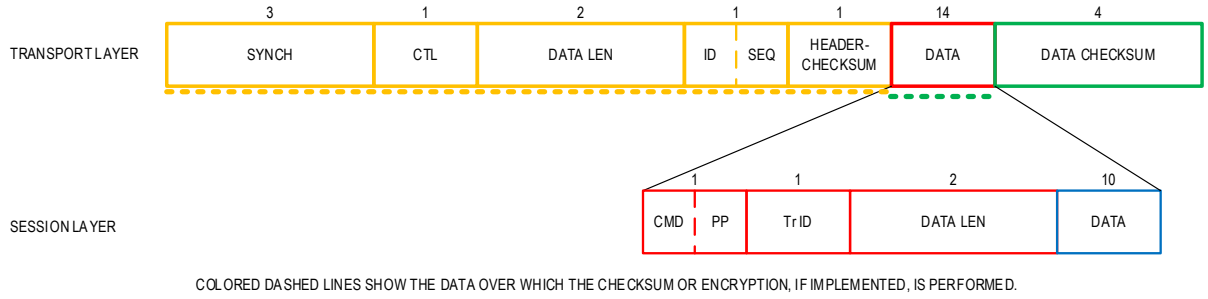


Table 26-14: HELLO Command

HELLO	0x06	Connection Request	
Command Format (Transport Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	**Synchronization Pattern** 0xBEEFED (fixed)
CTL	0x03	1	**Command Code** 0x06 (fixed)
DATA LEN	0x04	2	**Length of Data Segment** 0x0000 (fixed)
ID	0x06	4 bits	**Channel ID (upper 4 bits)** Defined in the CON_REQ command
SEQ		4 bits	**Sequence Number (lower 4 bits)** Same as the sequence number of the preceding command
HEADER CHECKSUM	0x07	1	**Header Checksum**
Command Format (Session Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
CMD	0x00	4 bits	**Data Transfer Command Code (lower 4 bits)** 0b0001 (fixed - HELLO)
PP		4 bits	**Protection Profile (upper 4 bits)** 0b0000 (fixed - none)
TrID	0x01	1	**Transaction Sequence Number** 0x00 (fixed)
DATA LEN	0x02	2	**Length of Data Segment** 0x000A (fixed)
DATA	0x04	10	**Data Segment** 0x00: 'H' 0x01: 'E' 0x02: 'L' 0x03: 'L' 0x04: 'O'

HELLO	0x06	Connection Request	
			0x05: 0x20 0x06: 'B' 0x07: 'L' 0x08: 0x03 0x09: 0x02

26.14.7 HELLO_REPLY

The SCPBL sends a HELLO_REPLY to the host in response to the HELLO command. The payload of the command provides information about the device configuration, including:

- ROM version
- Life cycle information
- JTAG status
- USN

Figure 26-13: HELLO_REPLY Structure

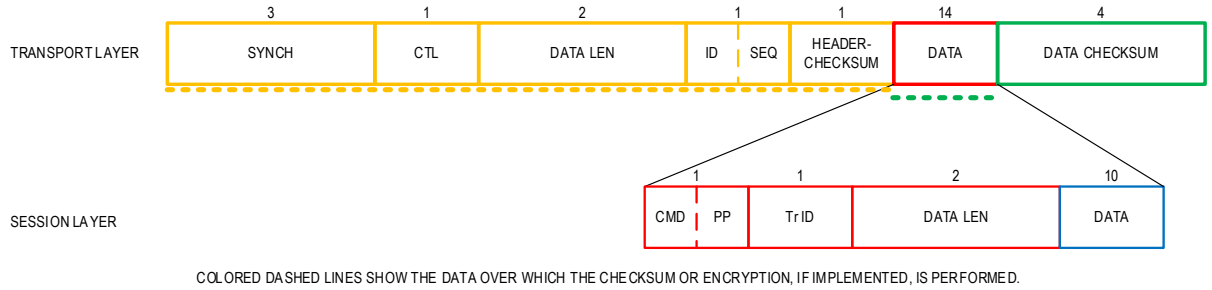


Table 26-15: HELLO_REPLY Command

HELLO_REPLY	0x06	Connection Request	
Command Format (Transport Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	**Synchronization Pattern** 0xBEEFED (fixed)
CTL	0x03	1	**Command Code** 0x06 (fixed)
DATA LEN	0x04	2	**Length of Data Segment** 0x0000 (fixed)
ID	0x06	4 bits	**Channel ID (upper 4 bits)** Defined in the CON_REQ command
SEQ		4 bits	**Sequence Number (lower 4 bits)** Same as the sequence number of the preceding command
HEADER CHECKSUM	0x07	1	**Header Checksum** See *Table 26-4* for details.
Command Format (Session Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
CMD	0x00	4 bits	**Data Transfer Command Code (upper 4 bits)** Same as the channel ID of the preceding command
PP		4 bits	**Protection Profile (lower 4 bits)** 0x8: The payload is in plaintext format with an RSA-4096 signature 0x8: The payload is in plaintext format with an RSA-2048 signature 0xA: payload is in plaintext format with an ECDSA-256 signature
TrID	0x01	1	**Transaction Sequence Number**
DATA LEN	0x02	2	**Length of Data Segment** 0x0034

HELLO_REPLY	0x06	Connection Request	
DATA	0x04	10	**Data Segment (fixed)** 0x00: 'H' 0x01: 'E' 0x02: 'L' 0x03: 'L' 0x04: 'O' 0x05: 0x20 0x06: 'H' 0x07: 'O' 0x08: 'S' 0x09: 'T' 0x0A - 0x0D: ROM Version 0x0E: Lifecycle 0x0F: 0x00 0x10: 0x00 0x11: JTAG Configuration 0x00: Activated 0x10: Deactivated 0x12 - 0x1F: USN 0x20 - 0x2C: 0x00

26.15 Data Transfer Command Details

26.15.1 WRITE_DATA

WRITE_DATA is a specific implementation of the DATA_TRANSFER command. It is the way the application layer commands (which include loading program memory) pass through the SCP.

Unlike other DATA_TRANSFER commands, WRITE_DATA implements the session-level signature to verify and authenticate the host.

Figure 26-14: WRITE_DATA Command Structure

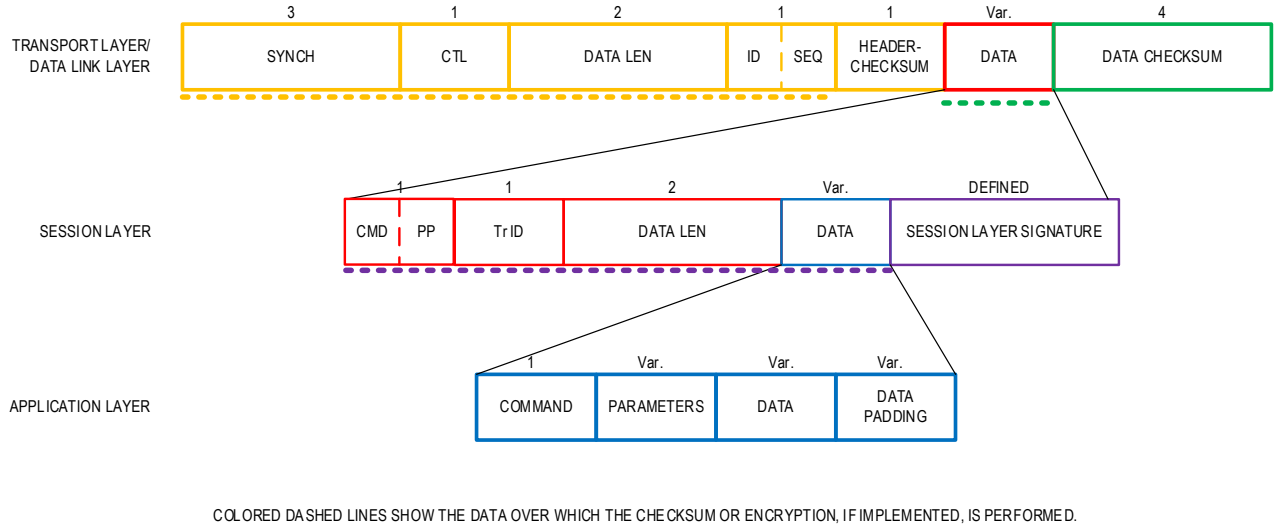


Table 26-16: WRITE_DATA

WRITE_DATA	0x06	Write Data to Memory or Configuration Commands	
Command Format (Transport Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	**Synchronization Pattern** 0xBEEFED (fixed)
CTL	0x03	1	**Command Code** 0x06 (fixed)
DATA LEN	0x04	2	**Length of Data Segment** 0x0000 (fixed)
ID	0x06	4 bits	**Channel ID (upper 4 bits)** Defined in the CON_REQ command
SEQ		4 bits	**Sequence Number (lower 4 bits)** Same as the sequence number of the preceding command
HEADER CHECKSUM	0x07	1	**Header Checksum**
Command Format (Session Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
CMD	0x00	4 bits	**Data Transfer Command Code (upper 4 bits)** 0b1001 (fixed)
PP		4 bits	**Protection Profile (lower 4 bits)** 0x8: The payload is in plaintext with RSA-4096 signature

WRITE_DATA	0x06	Write Data to Memory or Configuration Commands	
			0x8: The payload is in plaintext with RSA-2048 signature 0xA: The payload is in plaintext with ECDSA-256
TrID	0x01	1	**Transaction Sequence Number**
DATA LEN	0x02	2	**Length of Data Segment** User-defined
DATA	0x04	10	**Data Segment** User-defined
SIGNATURE	0x0E	1	**Digital Signature** Defined by device

26.15.2 COMMAND_RSP

Most application layer commands generate a 4-byte response indicating the success or failure of the command. This packet is sent from the SCPBL to the host after the SCPBL ACKs the command and after the application layer command is complete. A response value of 0x0000 indicates a successful command. Other values indicate an error and are specific to each command.

COMMAND_RSP is a DATA TRANSFER command.

Figure 26-15: COMMAND_RSP Command Structure

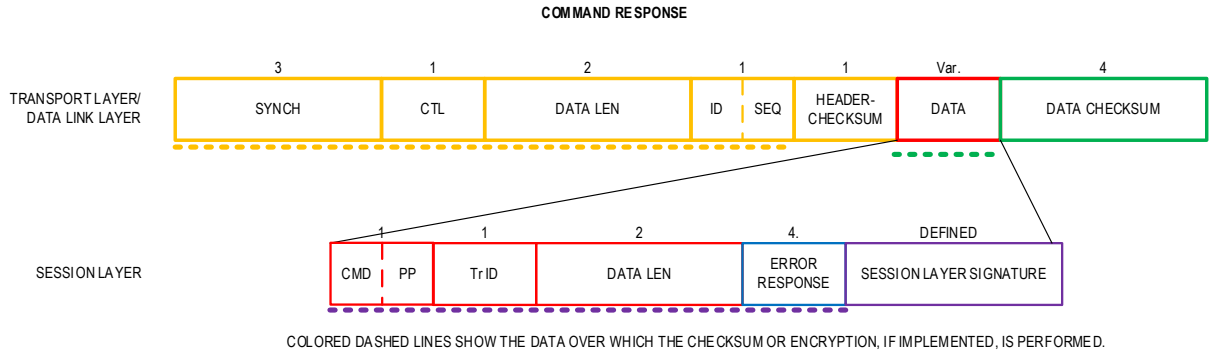


Table 26-17: COMMAND_RSP

COMMAND_RSP		0x06	Success/Failure Response from Application Layer Command	
Command Format (Transport Layer)				
Element		**Offset**	**Length (Bytes)**	**Values**
SYNCH		0x00	3	**Synchronization Pattern** 0xBEEFED (fixed)
CTL		0x03	1	**Command Code** 0x06 (fixed)
DATA LEN		0x04	2	**Length of Data Segment** 0x0000 (fixed)
ID		0x06	4 bits	**Channel ID (upper 4 bits)** Defined in the CON_REQ command
SEQ			4 bits	**Sequence Number (lower 4 bits)** Same as the sequence number of the preceding command
HEADER CHECKSUM		0x07	1	**Header Checksum**
Command Format (Session Layer)				
Element		**Offset**	**Length (Bytes)**	**Values**
CMD		0x00	4 bits	**Data Transfer Command Code (upper 4 bits)** 0b1001 (fixed)
PP			4 bits	**Protection Profile (lower 4 bits)** 0x8: The payload is in plaintext with an RSA4096 signature 0x8: The payload is in plaintext with an RSA2048 signature 0xA: The payload is in plaintext with an ECDSA-256 signature
TrID		0x01	1	**Transaction Sequence Number**
DATA LEN		0x04	2	**Length of Data Segment**

COMMAND_RSP		0x06	Success/Failure Response from Application Layer Command	
				User-defined
RESPONSE		0x04	4	**Command Response**
SIGNATURE		0x0E	1	**Digital Signature** Defined by device.

26.16 Application Layer Commands

The application layer deals with the commands used that directly modify the memory and configure various functions.

Table 26-18: MAX32666 Application Command Summary

CATEGORY	COMMAND	VALUE
Key Management	*WRITE_CRK*	0x470A
	REWRITE_CRK	0x461A
	WRITE_OTP	0x4714
Administrative	Reserved	0x4426
	Reserved	0x4538
	KILL_CHIP2	0x4539
	Reserved	0x4427
	WRITE_STIMULUS	0x4428
	WRITE_SLA_VERSION	0x470B
	Reserved	0x4429
Memory	*WRITE_DATA*	0x2402
	COMPARE_DATA	0x2403
	ERASE_DATA	0x4401
	EXECUTE_CODE	0x2101

26.16.1 Application Layer Command Details

The memory commands WRITE_DATA, ERASE_DATA, and COMPARE_DATA directly address internal flash and internal RAM based on the target address.

The execute command is used indirectly as with secondary-level applets that address external memory through WRITE_DATA, ERASE_DATA, and COMPARE_DATA. This makes the memory commands fully flexible so that these commands can operate on any external memory within the security context of the SCP framework.

26.16.1.1 WRITE_CRK

Table 26-19: WRITE_CRK

WRITE_CRK	0x4701	Write CRK to Device	
Description	This command writes the customer CRK to the nonvolatile memory of the SCPBL. This command can only be executed once per device. The REWRITE_CRK command can also only be executed once per device, allowing a maximum of two CRK values. This MRK signature is distinct from the packet's signature, which is also computed with the MRK.		
Response	ERR_NO ERR_BAD_VALUES ERR_PROHIBIT		
Command Format			
Element	**Start Address**	**Length (Bytes)**	**Values**
Command	0x0000	2	0x4701 (fixed)
Key Length	0x0002	2	0x0204 (fixed)
CRK	0x0004	64	Public key
MRK Signature	0x0104	64	MRK signature of the CRK using the public key

26.16.1.2 REWRITE_CRK

Table 26-20: REWRITE_CRK

REWRITE_CRK	0x461A	Write a Second Write CRK to Device	
Description	This command writes a second CRK to the nonvolatile memory of the SCPBL. It is the only way to change the CRK after the WRITE_CRK command. This command can only be executed once per device, so a maximum of two CRK values can be written. This command requires the previous CRK as one of the arguments to prevent accidental changing of the CRK value.		
Response	ERR_NO ERR_BAD_VALUES ERR_PROHIBIT		
Command Format			
Element	**Start Address**	**Length (Bytes)**	**Values**
Command	0x0000	2	0x461A (fixed)
Key Length	0x0002	2	0x0204
Previous CRK	0x0004	64	Previous CRK
New CRK	0x0104	64	New CRK
MRK Signature	0x0208	64	MRK signature

26.16.1.3 WRITE_OTP

Table 26-21: WRITE_OTP

WRITE_OTP	0x4714	Write Data to OTP Memory	
Description	This reserved command should only be used when specifically instructed by the factory.		
Response	ERR_NO ERR_BAD_VALUES ERR_PROHIBIT		
Command Format			
Element	**Start Address**	**Length**	**Values**
Command	0x0000	2	0x4714 (fixed)
Data Length	0x0002	2	This is the length of the data segment in bytes. The length is sent MSB first.
Offset	0x0004	2	This offset value starts after the CRK data in the real OTP (i.e., the value 0 points to the first byte at the position 0x2C8). The offset in OTP is sent LSB first.
Data	0x0006	var	This is the data to be stored in OTP. This data value contains the lock bit. The CV value is not transmitted and is computed by the ROM code itself. The data is transmitted in LSB first format.

26.16.1.4 WRITE_TIMEOUT

Table 26-22: WRITE_TIMEOUT

WRITE_TIMEOUT	0x4426	Set MAX32666 Session Timeout Interval	
Description	This command specifies the time within a hello session that the SCPBL waits before terminating the session and searching for a valid stimulus again. The details of the conditions which start/restart the timer and the action at the end of the timeout are listed above. This command can only be executed once for the interfaces listed in *Table 26-2*. The timeout period is in units of milliseconds; an interval of 0x000003E8 corresponds to 1s. Note that this timeout is different than the retransmission timeout.		
Response	ERR_NO ERR_ALREADY ERR_BAD_VALUES ERR_PROHIBIT		
Command Format			
Segment	**Start Address**	**Length (Bytes)**	**Values**
Command	0x0000	2	0x4426 (fixed)
Target	0x0002	1	See *Table 26-2* for the interfaces that are supported. 0x00: UART 0x45: Ethernet 0x53: SPI 0x55: USB 0x56: VBUS detect
Value	0x0003	2	This is the value of the timeout in milliseconds, from 0x0001 to 0xFFFE. The timeout value is sent LSB first. Some devices have a fixed timeout, as shown in *Table 26-2*.

26.16.1.5 KILL_CHIP2

Table 26-23: KILL_CHIP2

KILL_CHIP2	0x4539	Transition to Shutdown Mode	
Description	This command permanently disables the SCPBL and execution of any code in flash, as well as erasing sensitive registers. The shutdown occurs immediately when the command executes, without waiting for a session closure. The USN of the chip (as seen in the HELLO_RSP packet from the part) is part of this command.		
Response	ERR_NO All other error codes are part-specific.		
Command Format			
Segment	**Start Address**	**Length (Bytes)**	**Values**
Command	0x0000	2	0x4539 (fixed)
USN	0x0002	13	USN

26.16.1.6 WRITE_STIMULUS

Table 26-24: WRITE_STIMULUS

WRITE_STIMULUS	0x4427	Configure SCPBL Stimulus Pin (MAX32666 Only)								
Description	This command allows the SCPBL to permanently change the location and polarity of the SCPBL stimulus pin. This command can be executed three times. Each execution can designate the stimulus pin for one of the available interfaces. Not all GPIO ports or GPIO pins are implemented on all devices. Not all GPIO are available in all packages. See *Table 26-2* for valid pin configurations. Regardless of specific settings, the value of this byte must not be 0xFF.									
	Bit	**7**	**6**	**5**	**4**	**3**	**3**	**2**	**1**	**0**
	Function	GPIO Port 0x0: Port 0 0x1: Port 1 0x2: Port 2 0x3: Port 3		Active: 0: Low 1: High	GPIO Pin 0x00: GPIO Pin 0 of the selected port … 0x1F: GPIO Pin 31 of the selected port					
Response	ERR_NO All other error codes are part-specific.									

Command Format			
Segment	**Start Address**	**Length (Bytes)**	**Values**
Command	0x0000	2	0x4427 (fixed)
Program Count	0x0002	1	Each interface can be configured up to three times. The location value is the last one programmed. 0x01: First setting of interface/stimulus pin 0x02: Second setting of interface/stimulus pin 0x03: Third setting of interface/stimulus pin
Stimulus Location	0x0003	1	See *Table 26-2* for valid pin options.
Target	0x0004	1	See *Table 26-2* for the interfaces supported by each part.
Configuration	0x0005	4	UART: 0x00, 0x03, 0x08, 0x07 (115200 bps, 8 data bits, no parity, 1 stop bit) USB: 0x00, 0x00, 0x00, 0x00

26.16.1.7 WRITE_SLA_VERSION

Table 26-25: WRITE_SLA_VERSION

WRITE_SLA_VERSION	0x470B	Set Minimum Revision Value	
Description	This command sets up the minimum required revision level in the SLA header of a command. Only commands with a revision level equal to or greater than the one set by this command will be allowed to run. This command can only change the revision value six times. The HELLO_RSP command returns the last value written with this command. The HELLO_RSP command returns 0x0000000 if no value has been set yet by this command.		
Response	ERR_NO All other error codes are part-specific.		
Command Format			
Segment	**Start Address**	**Length (Bytes)**	**Values**
Command	0x0000	2	0x470B (fixed)
Revision	0x0002	4	User-defined 4-byte value. (default 0x00000000)

26.16.1.8 WRITE_DEACTIVATE

Table 26-26: WRITE_DEACTIVATE

WRITE_DEACTIVATE	0x4429	Permanently Deactivate SPCPBL Interface	
Description	This command deactivates an SCP interface. This command can be used once for each link. A deactivated link cannot be reactivated.		
Response	ERR_NO All other error codes are part-specific.		
Command Format			
Segment	**Start Address**	**Length (Bytes)**	**Values**
Command	0x0000	2	0x4429 (fixed)
Link	0x0002	1	See *Table 26-2* for the interfaces supported by each part.

26.16.1.9 WRITE_DATA

Table 26-27: WRITE_DATA

WRITE_DATA	0x2402	Write Data to Memory	
Description	The command writes data into internal flash or internal RAM beginning at the target start address.		
Response	ERR_NO ERR_BAD_VALUES ERR_PROHIBIT		
Command Format			
Segment	**Start Address**	**Length (Bytes)**	**Values**
Command	0x0000	2	0x2402 (fixed)
Start Address	0x0002	4	This is the target start address. The MSB is sent first.
Length	0x0004	4	This is the length of the data segment in bytes. The MSB is sent first.
Data	0x0006	Variable	This is the data to be written.

26.16.1.10 COMPARE_DATA

Table 26-28: COMPARE_DATA

COMPARE_DATA	0x2403	Compare Data Against Internal Memory	
Description	The command compares the supplied data against the contents of internal flash or internal RAM based on the target address.		
Response	ERR_NO ERR_BAD_VALUES ERR_PROHIBIT		
Command Format			
Segment	**Start Address**	**Length (Bytes)**	**Values**
Command	0x0000	2	0x2403 (fixed)
Start Address	0x0002	4	This is the target start address. The MSB is sent first.
Length	0x0004	4	This is the length of the data segment in bytes. The MSB is sent first.
Data	0x0006	Variable	Compare data

26.16.1.11 ERASE_DATA

Table 26-29: ERASE_DATA

ERASE_DATA	0x4401	Erase Range of Flash Memory	
Description	Erase a number of bytes of flash starting from the start address.		
Response	ERR_NO ERR_BAD_VALUES ERR_PROHIBIT		
Command Format			
Segment	**Start Address**	**Length (Bytes)**	**Values**
Command	0x0000	2	0x2403 (fixed)
Start Address	0x0002	4	This is the target start address. The MSB is sent first.
Length	0x0004	4	This is the length of the data segment in bytes. The MSB is sent first.

26.16.2 EXECUTE_CODE

Table 26-30: EXECUTE_CODE

EXECUTE_CODE	0x2101	Execute Code from Flash Memory	
Description	The execute command is used indirectly as part of secondary level applets. This makes the memory commands fully flexible so that these commands can operate on any external memory within the security context of the SCP framework.		
Response	ERR_NO		
Command Format			
Segment	**Start Address**	**Length (Bytes)**	**Values**
Command	0x0000	2	0x2401 (fixed)
Start Address	0x0002	4	This is the address to begin parsing for an application header in the target memory.

27. Debug Access Port (DAP)

The Arm debug access port (DAP) supports debugging during application development. Refer to the ordering information in the device data sheet to determine if a specific part number supports a customer-accessible DAP. *GCR_SYSST.icelock* = 0 if the device provides a customer-accessible DAP.

27.1 Instances

The DAP interface communicates through the serial wire debug (SWD), and JTAG interface signals shown in *Table 27-1*.

Table 27-1: MAX32665/MAX32666 DAP Instances

SWD SIGNAL	JTAG SIGNAL
SWCLK	TMS
SWDIO	TCK
SWDIO2	-
SWCLK2	-
	JTAG TDI
	JTAG TDO

27.2 DAP Access Control

The DAP is enabled after every POR to allow debugging during development. The interface can be disabled in software by setting the *GCR_SCON.sw_dis* field to 1. The *GCR_SCON.sw_dis* field clears to 0 again, reenabling the DAP after a POR. Parts with a customer-accessible DAP should disable the DAP in a final customer product.

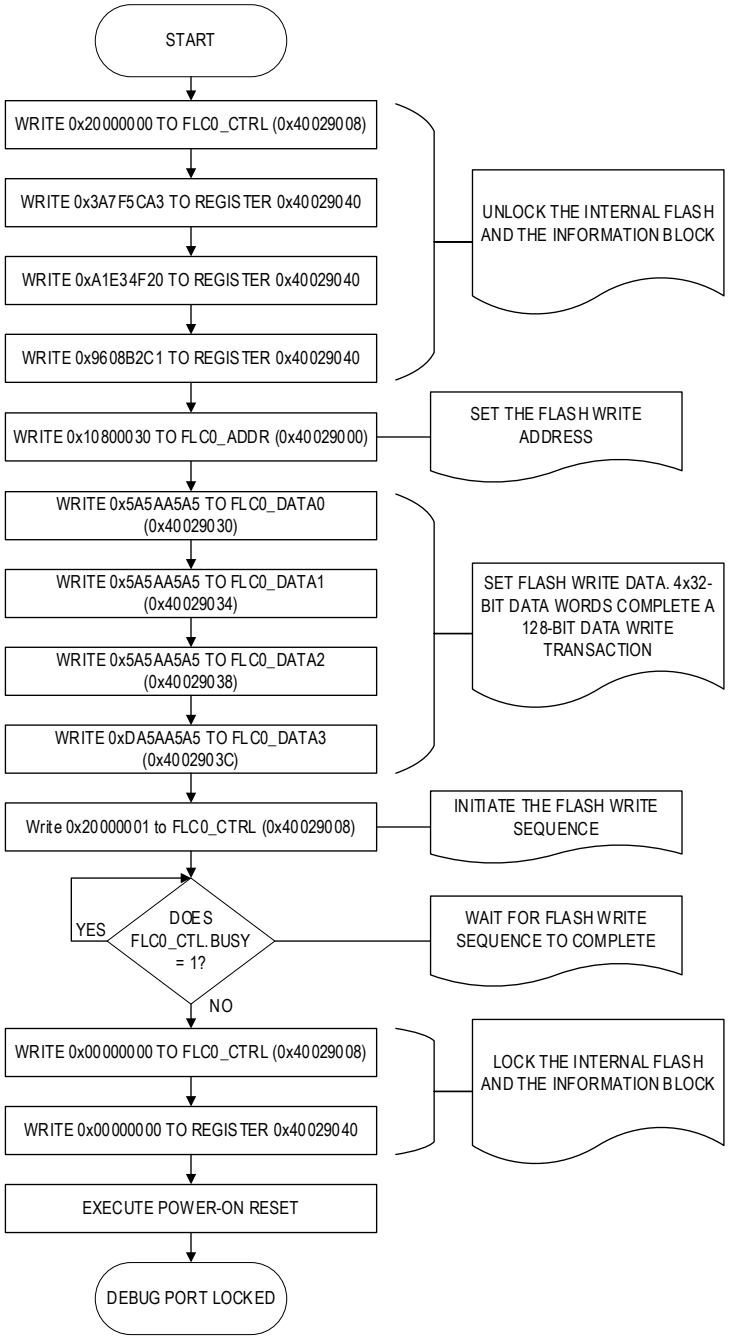
1.1.1 Locking the DAP

There are two options for locking out the debug access port. Option 1 locks the DAP and makes it available to be unlocked later. This is a one-time-only process. The DAP port cannot be re-locked. Option 2 locks the DAP permanently.

1.1.1.1 Option 1

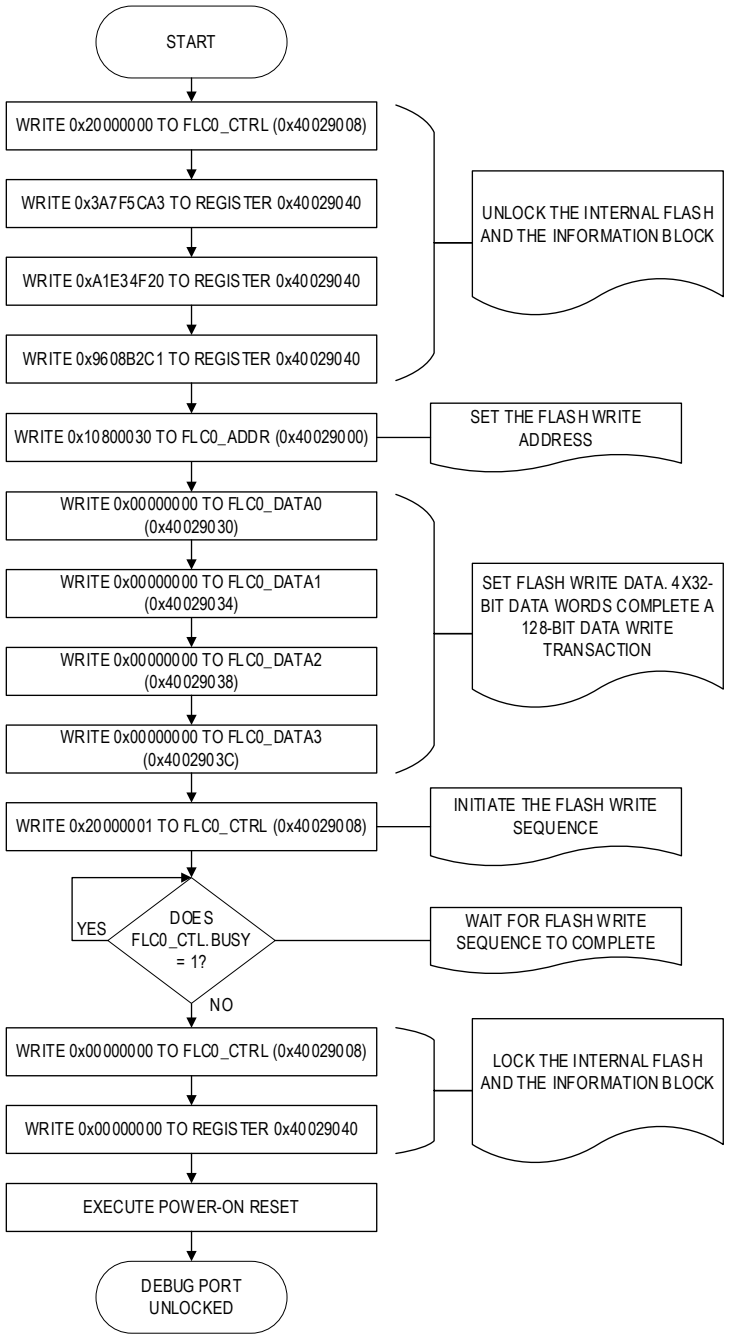
To lock the DAP and make it available to be unlocked later (one time only), follow the flow chart in *Figure 27-1*.

Figure 27-1: Locking the DAP to Make it Available for Unlock Later



To unlock the DAP after it has been locked using the flow chart of *Figure 27-1*, follow the flow chart in *Figure 27-2*.

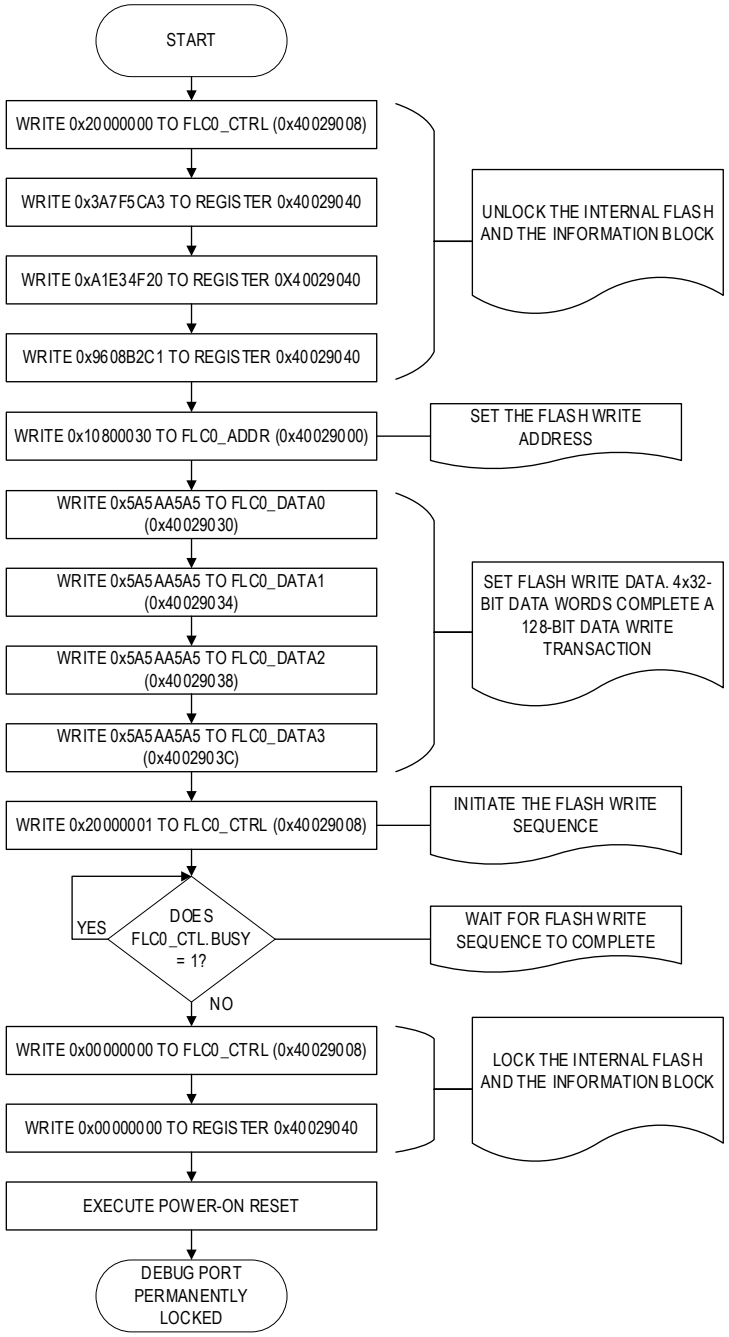
Figure 27-2: Unlocking the DAP After Being Locked as in Figure 27-1



1.1.1.2 Option 2

To lock the DAP permanently, follow the flow chart in *Figure 27-3*.

Figure 27-3: Locking the Debug Access Port Permanently



27.3 Pin Configuration

SWD or JTAG signals in the GPIO and Alternate Function matrices indicate which GPIO pins are associated with a particular signal. It is unnecessary to configure a pin to use the DAP following a POR for an alternate function.

By default, the pin associated with the bidirectional SWDIO/TMS signal is a GPIO high-impedance input after any POR. While the DAP is in use, connect a pullup resistor to the SWDIO/TMS pin. The pullup ensures the signal is in a known state when the SWDIO/TMS pin control is transferred between the host and target. Remove the pullup resistor if the associated pin is used as a GPIO to avoid unnecessary current consumption.

28. Revision History

<table>
<tr><th>REVISON NUMBER</th><th>REVISION DATE</th><th>DESCRIPTION</th></tr>
<tr><td>0</td><td>6/19</td><td>Initial Release</td></tr>
<tr><td>0.5</td><td>01/10/2020</td><td>Register field naming updates to match SDK.
Added Chapter 23 Audio Subsystem
Added Chapter 26 Debug Access Port</td></tr>
<tr><td>0.6</td><td>02/19/2020</td><td>Updated Chapter 23 Audio Subsystem to add special considerations.</td></tr>
<tr><td>1</td><td>02/21</td><td>Added Example 32.768kHz Crystal Capacitor Determination.
Added Chapter 25 Secure Communication Protocol Bootloader (SCPBL).</td></tr>
<tr><td>2</td><td>01/22</td><td>Removed all references to MAX32667; Removed all references to MAX32668.
Added bit definitions for VDD5, VDD4, and VDD3 power domains.
Added register and bit field definitions for the following:
<table>
<tr><td>USBHS_M31_PHY_PONRST</td><td>USBHS Power-On Reset Register</td></tr>
<tr><td>USBHS_M31_PHY_NONCRY_RSTB</td><td>USBHS Hi-Speed VBUS Reset Register</td></tr>
<tr><td>USBHS_M31_PHY_NONCRY_EN</td><td>USBHS Non-Clock Recovery Enable</td></tr>
<tr><td>USBHS_M31_PHY_PLL_EN</td><td>USBHS PLL Enable Register</td></tr>
<tr><td>USBHS_M32_PHY_OSCOUTEN</td><td>USBHS Oscillator Output Enable Register</td></tr>
<tr><td>USBHS_M32_PHY_CORECLKIN</td><td>USBHS Hi-Speed Core Clock Input Register</td></tr>
<tr><td>USBHS_M32_PHY_XTLSEL</td><td>USBHS Hi-Speed Clock Source Frequency Select Register</td></tr>
<tr><td>USBHS_M32_PHY_OUTCLKSEL</td><td>USBHS Hi-Speed Reference Clock Select Register</td></tr>
<tr><td>USBHS_MXM_INT</td><td>USBHS Hi-Speed VBUS Interrupt Register</td></tr>
<tr><td>USBHS_MXM_INT_EN</td><td>USBHS Hi-Speed VBUS Interrupt Enable Register</td></tr>
<tr><td>USBHS_MXM_SUSPEND</td><td>USBHS Suspend Register</td></tr>
<tr><td>USBHS_MXM_REG_A4</td><td>USBHS Hi-Speed VBUS State Register</td></tr>
</table>
Added Table 22-1: USB Instance Table
Added Chapter 20 Semaphores
Updated Table 13-1: MAX32665/MAX32666 SPI Instances
Clarified GPIO 6.2.1 Reset State
Revised the register offset addresses in the DMA register tables 9.9 DMA Registers and 9.12 DMA Channel Registers
Updated all of the the clocking scheme diagrams Figure 4-1: Clock Block Diagram, Figure 4-4: SLEEP Mode Clock Control, Figure 4-5: DEEPSLEEP Clock Control, Figure 4-6: BACKUP Mode Clock Control
Added GCR_PCKDIV.audclksel
CRC Chapter removed.
Updated Chapter 25 Trust Protection Unit (TPU) to add CRC as a subsection.
Added Cache Invalidating steps 4.5.1 Enabling ICC0/ICC1/SFCC
Updated Chapter 26 Secure Communication Protocol Bootloader (SCPBL)
Added Chapter 16 Wake-Up Timer (WUT0)</td></tr>
<tr><td>3</td><td>02/22</td><td>Updated note in 7.2.4 Flash Write</td></tr>
</table>