MAX32690 User Guide

UG7618; Rev 1; 1/2023

Abstract: This user guide provides application developers information on how to use the memory and peripherals of the MAX32690 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.

MAX32690 User Guide

Table of Contents

Table of Figures

Table of Tables

Table of Equations

1. Introduction

For ordering information, mechanical, and electrical characteristics for the MAX32690 family of devices, refer to the data sheet.

1.1 Related Documentation

The MAX32690 data sheet and errata are available from the Analog Devices website (https://www.analog.com).

1.2 Document Conventions

1.2.1 Number Notations

Notation	Description
0xNN	Hexadecimal (Base 16) numbers are preceded by the prefix 0x.
0bNN	Binary (Base 2) numbers are preceded by the prefix 0b.
NN	Decimal (Base 10) numbers are represented using no additional prefix or suffix.
V[X:Y]	Bit field representation of a register, field, or value (V) covering Bit X to Bit Y.
Bit N	Bits are numbered in little-endian format, i.e., the least significant bit of a number is referred to as Bit 0.
[0xNNNN]	An address offset from a base address is shown in bracket form.

1.2.2 Register and Field Access Definitions

All the fields 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 1-1*.

Table 1-1: Field Access Definitions

Access Type	Definition
RO	**Reserved** This access type is reserved for static fields. Reads of this field return the reset value. Writes are ignored.
DNM	**Reserved. Do Not Modify** Software must first read this field and write the same value whenever writing to this register.
R	**Read Only** Reads of this field return a value. Writes to the field do not affect device operation.
W	**Write Only** Reads of this field return indeterminate values. Writes to the field change the field's state to the value written and can affect device operation.
R/W	**Unrestricted Read/Write** Reads of this field return a value. Writes to the field change the field's state to the value written and can affect device operation.
RC	**Read to Clear** Reading this field clears the field to 0. Writes to the field do not affect device operation.
RS	**Read to Set** Reading this field sets the field to 1. Writes to the field do not affect device operation.
R/W0O	**Read/Write 0 Only** Writing 0 to this field set the field to 0. Writing 1 to the field does not affect device operation.
R/W1O	**Read/Write 1 Only** Writing 1 to this field sets the field to 1. Writing 0 to the field does not affect device operation.
R/W1C	**Read/Write 1 to Clear** Writing 1 to this field clears this field to 0. Writing 0 to the field does not affect device operation.
R/W0S	**Read/Write 0 to Set** Writing 0 to this field sets this field to 1. Writing 1 to the field does not affect device operation.

1.2.3 Register Lists

Each peripheral includes a table listing all of the peripheral's registers. The register table includes the offset, register name, and description of each register. The offset shown in the table must be added to the peripheral's base address in *Table 3-2* to get the register's absolute address.

Table 1-2: Example Registers

Offset	Register Name	Description
[0x0000]	REG_NAME0	Name 0 Register

1.2.4 Register Detail Tables

Each register in a peripheral includes a detailed register table, as shown in *Table 1-3*. The first row of the register detail table includes the register's description, name, and offset from the base peripheral address. The second row of the table is the header for the bit fields represented in the register. The third and subsequent rows of the table include the bit or bit range, the field name, bit's or field's access, reset value, and a description of the field. All registers are 32-bits unless specified otherwise. Reserved bits and fields are shown as **Reserved** in the description column. See *Table 1-1* for a list of all access types for each bit and field.

Table 1-3: Example Name 0 Register

Name 0			**REG_NAME0**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	-	**Reserved**
15:0	field_name	R/W	0	**Field name description** Description of *field_name*.

2. Overview

The MAX32690 microcontroller (MCU) is an advanced system-on-chip (SoC) featuring an Arm® Cortex®-M4F CPU, large Flash and SRAM memories, and the latest generation Bluetooth® 5 Low Energy (LE) radio. This device unites processing horsepower with the connectivity required for IoT applications.

The MCU offers many features to increase reliability: full-time error detection and correction (ECC) on processor cache and optional ECC on SRAM. The MAX32690 is qualified to operate at a temperature range of -40°C to +105°C, ideal for Industrial environments.

Bluetooth® 5 Low Energy (LE) radio supports Mesh, LE Audio, angle of arrival (AoA), and angle of departure (AoD) for direction finding, long-range (coded), and high-throughput modes. A dedicated RISC-V core handles timing-critical controller tasks, freeing the programmer from BLE interrupt latency concerns.

A Trust Protection Unit (TPU) provides advanced security features, including a Modular Arithmetic Accelerator (MAA) for fast ECDSA, an AES Engine, TRNG, SHA-256 Hash, and a Secure Boot Loader.

Internal code and data spaces can be expanded off-chip through a quad SPI execute-in-place (XIP) interface or a Hyperbus/Xccela™ interface.

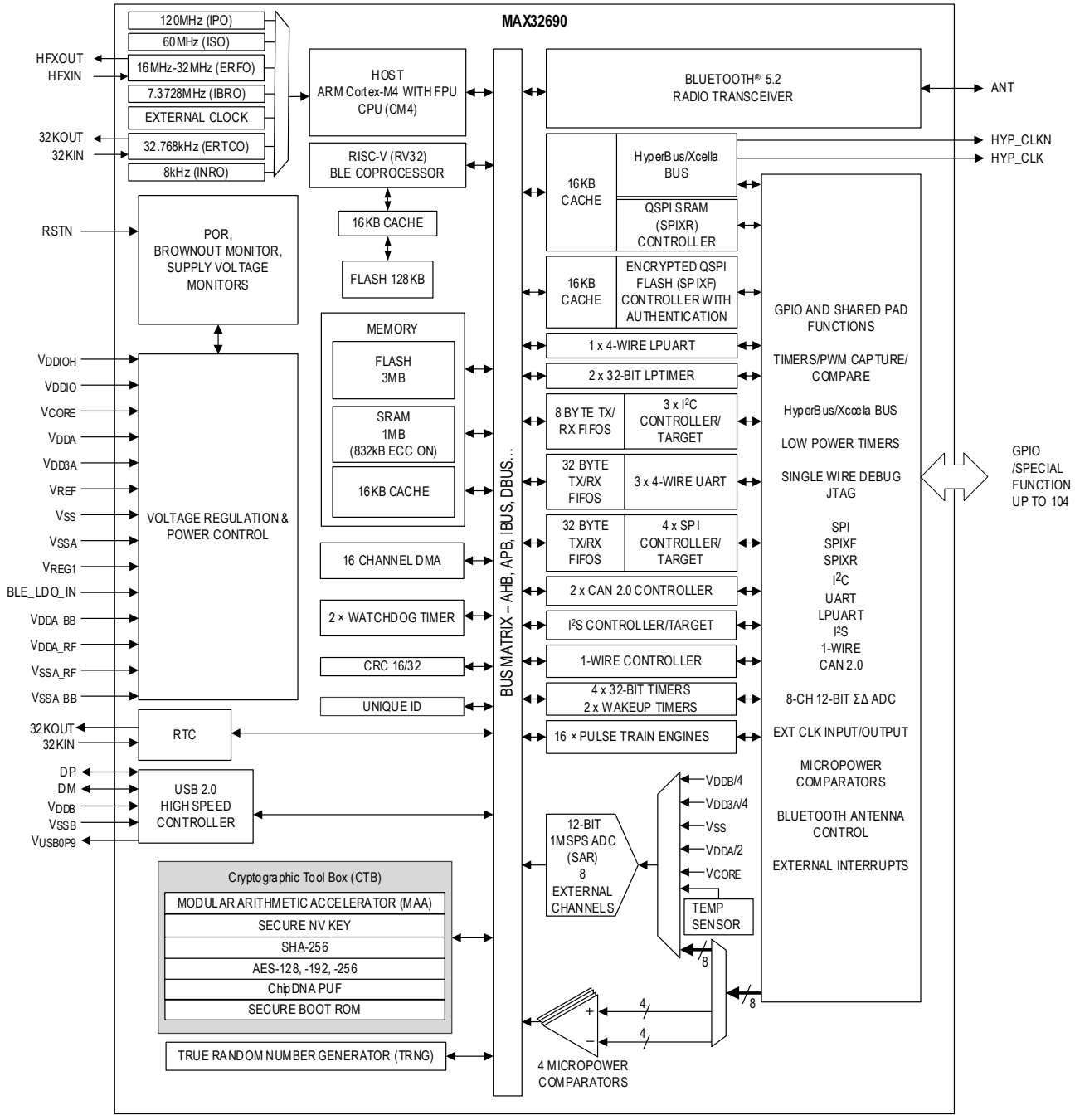
Many high-speed interfaces are supported on the device, including multiple QSPI, UART, CAN 2.0, and I^2C serial interfaces, plus one I^2S port for connecting to an audio codec. All interfaces support efficient DMA-driven transfers between peripheral and memory. A 12-input (8 external), 12-Bit SAR ADC samples analog data at up to 1MHz.

For information on the Arm Cortex-M4 with FPU core, refer to the *Arm Cortex-M4 Processor Technical Reference Manual*.

The high-level block diagram for the MAX32690 is shown in *Figure 2-1*.

2.1 Block Diagram

Figure 2-1: MAX32690 Block Diagram



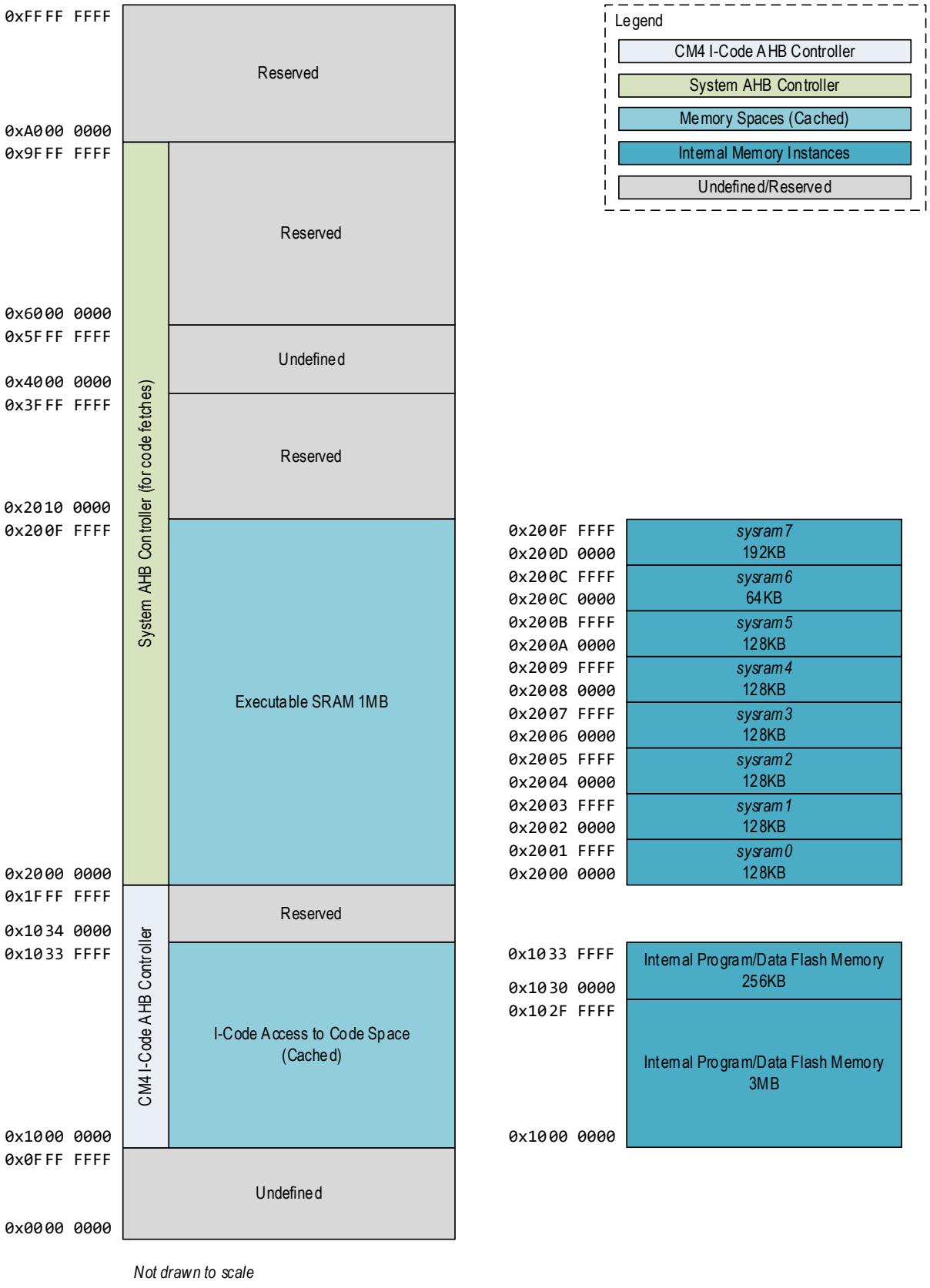
3. Memory, Register Mapping, and Access

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.

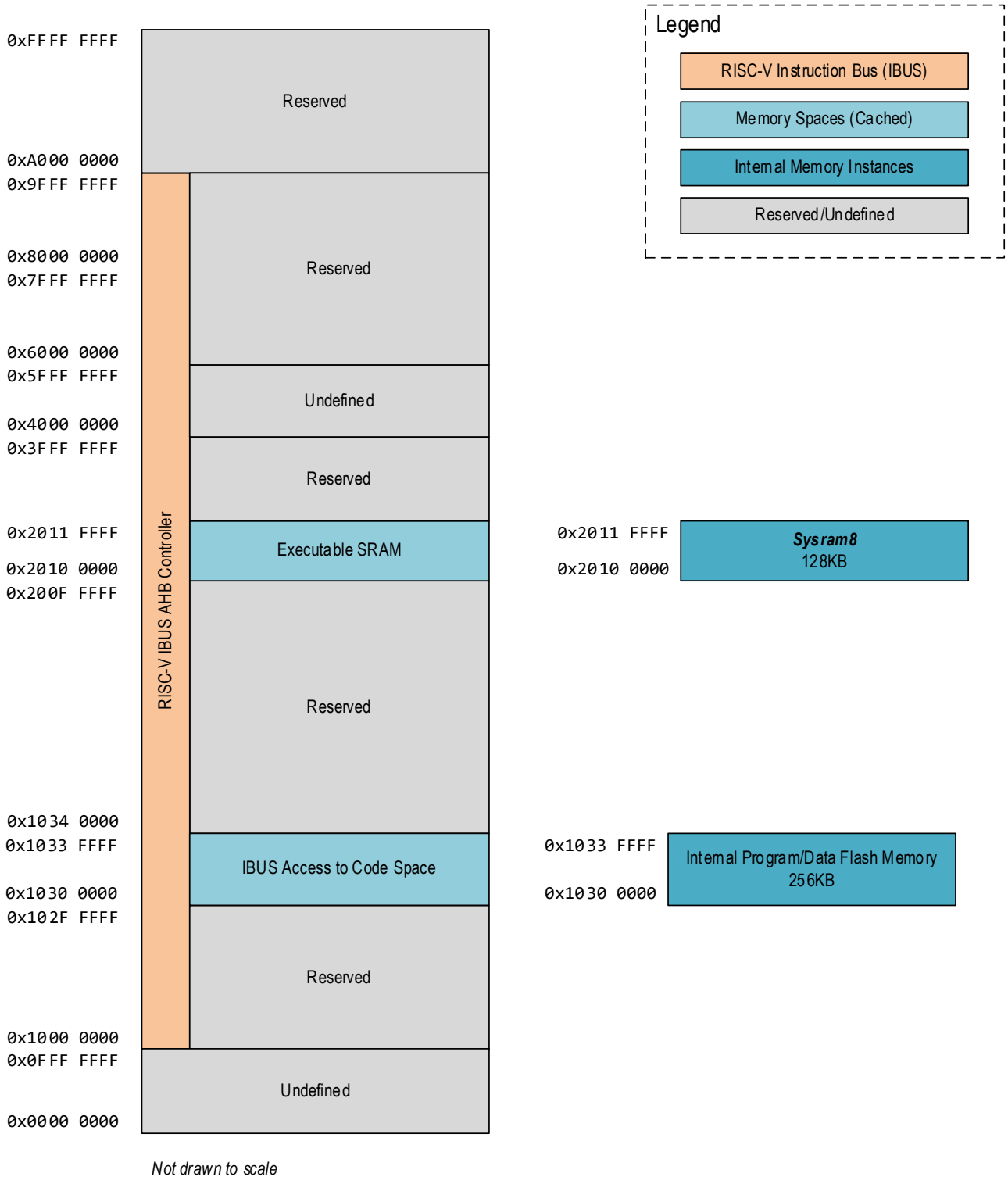
It is important to note, however, that the architectural definition does not require the entire 4GB memory range to be populated with addressable memory instances.

Figure 3-1: CM4 Code Memory Mapping



Not drawn to scale

Figure 3-2: RISC-V IBUS Code Memory Mapping



Not drawn to scale

Figure 3-3: CM4 Peripheral and Data Memory Mapping

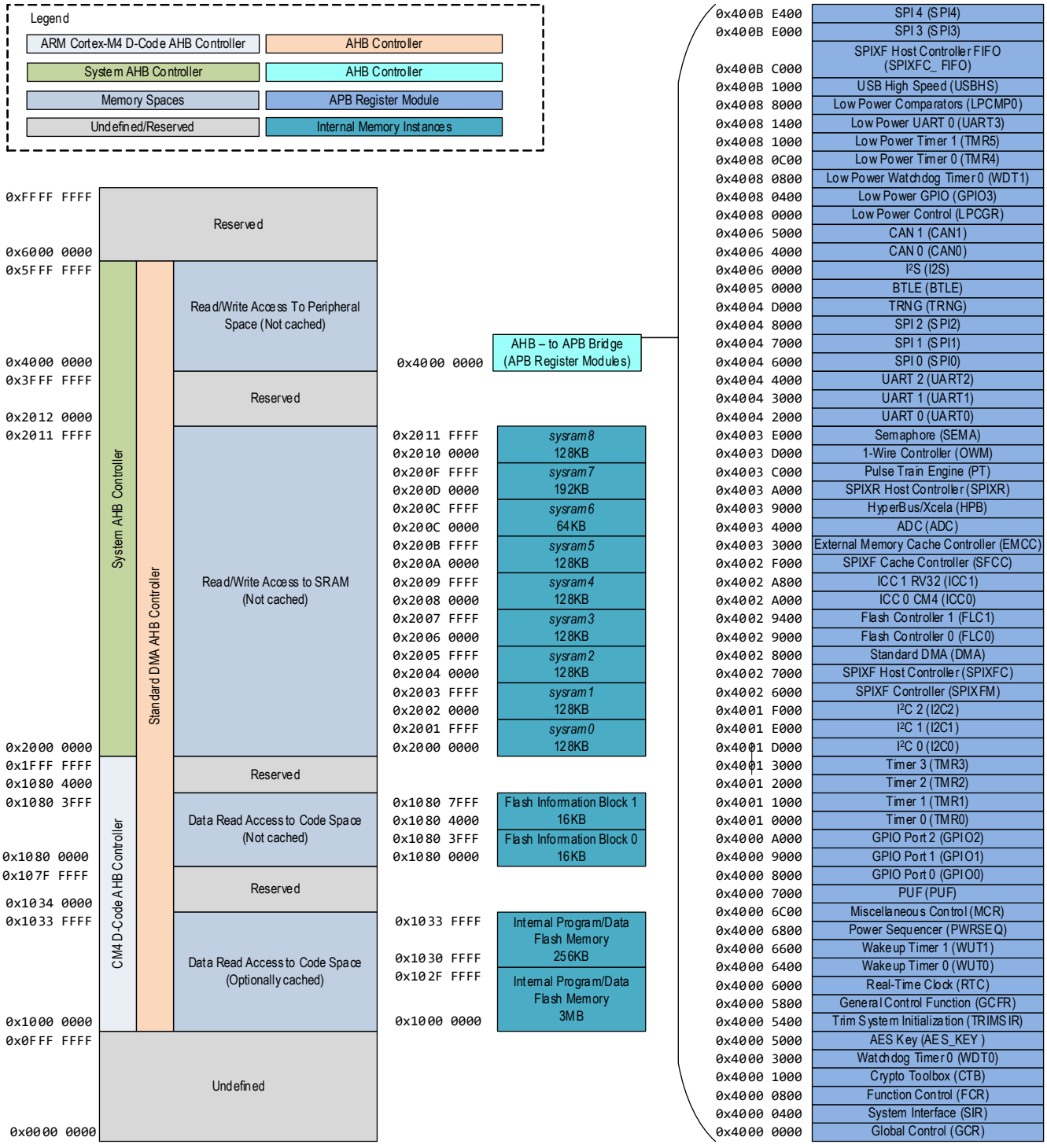
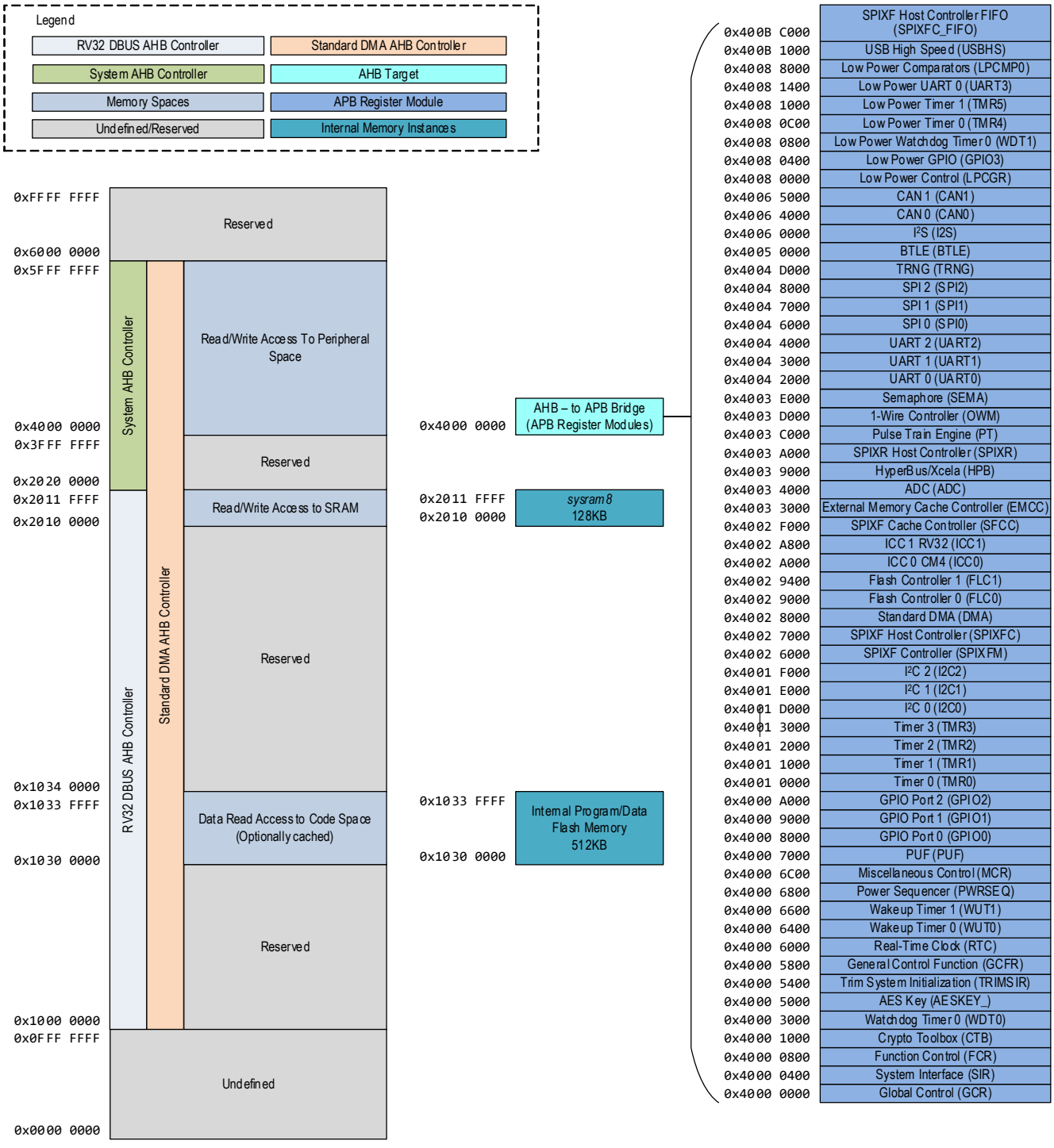


Figure 3-4: RV32 DBUS Peripheral and Data Memory Mapping



3.2 Standard Memory Regions

Several standard memory regions are defined for the Arm Cortex-M4 and RISC-V architectures; the use of many of these is optional for the system integrator. At a minimum, the MAX32690 must contain some code and data memory for application code and variable/stack use for the CM4, as well as certain components that are part of the instantiated Cortex-M4 core.

3.2.1 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 controllers are used by the Cortex-M4 core and Arm debugger to access this memory area. The I-Code AHB bus controllers is used for instruction decode fetching from code memory, while the D-Code AHB bus controller is used for data fetches from code memory. This is arranged so that data fetches avoid interfering with instruction execution. Additionally, the RV32 uses the IBUS for instruction fetches from code memory and the DBUS for data fetches from code memory.

The MAX32690 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 executed on the device. The internal flash memory is mapped into both code and data space from 0x1000 0000 to 0x1007 FFFF.

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 and contains the device ROM code that transfers execution to user code at address 0x1000 0000.

The code space memory on the MAX32690 also contains the mapping for the flash information block, from 0x1080 0000 to 0x1080 3FFF. However, this mapping is generally only present during Analog Devices production test; it is disabled once the information block is loaded with valid data and the info block lockout option is 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).

3.2.2 Instruction Cache Memory

The MAX32690 includes a dedicated unified internal cache controller with 16,384 bytes of cache memory for the CM4 core (ICC0). A dedicated unified internal cache controller for the RV32 is also provided (ICC1) which is also 16,384 bytes.

The unified internal cache memory is used to cache data and instructions fetched through the I-Code bus for the CM4 or the IBUS for the RV32 from the internal flash memory. See *Unified Internal Cache Controllers (ICC)* for detailed instructions on enabling the unified internal cache controllers.

3.2.3 Information Block Flash Memory

The information block is a separate area of the internal flash memory and is 16,384 Bytes. The information block is used to store trim settings (option configuration and analog trim) as well as other nonvolatile device-specific information. The information block also contains the device's unique serial number (USN). The USN is a 104-bit field.

Figure 3-5: 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	0	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	0	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
	0x10800014	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Reading the USN requires unlocking the information block. Unlocking the information block does not enable write access to the block but allows the contents of the USN to be read from the block. Unlock the information block using the following steps:

1. Write 0x3A7F 5CA3 to *FLCn_ACTRL*.
2. Write 0xA1E3 4F20 to *FLCn_ACTRL*.
3. Write 0x9608 B2C1 to *FLCn_ACTRL*.
4. Information block is now read-only 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 *FLCn_ACTRL*.

3.2.4 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. This memory can be used for general purpose variable and data storage, code execution, and the CM4 stack as well as the RV32 stack. An additional 16,384 byte cache is provided for the HyperBus/Xccela and SPI XF

The MAX32690 CM4's data memory mapping is illustrated in *Figure 3-1*. The MAX32690 RV32's data memory mapping is illustrated in *Figure 3-2*. The system SRAM configuration is defined in *Table 3-1*.

The SRAM memory area contains the main system SRAM. The size of the internal general-purpose SRAM for the CM4 is 1MB and an additional 128KB of SRAM is available for the RV32. The SRAM is divided into eight banks. The CM4 has access to bank 0 through bank 7 (*sysram0* through *sysram7*). The CM4's contiguous address range is 0x2000 0000 to 0x200F FFFF. *Sysram8* is dedicated to the RV32 and is mapped from address 0x2010 0000 to 0x2011 FFFF.

The SRAM area on the MAX32690 can be used for data storage and code execution by the CM4 for *sysram0* through *sysram7*. The RV32 is limited to use of *sysram8* for code and data storage in SRAM.

Code stored in the SRAM is accessed directly for execution (using the system bus) and is not cached. The SRAM is also where the CM4 and RV32 stack must be located, as it is the only general-purpose SRAM memory on the device capable of this function.

Table 3-1: System SRAM Configuration

System RAM Block #	Size (KB)	Start Address	End Address	CM4 Accessible	RV32 Accessible
sysram0	128	0x2000 0000	0x2001 FFFF	✓	No
sysram1	128	0x2002 0000	0x2003 FFFF	✓	No
sysram2	128	0x2004 0000	0x2005 BFFF	✓	No
sysram3	128	0x2006 C000	0x2007 FFFF	✓	No
sysram4	128	0x2008 000	0x2009 FFFF	✓	No
sysram5	128	0x200A 0000	0x200B FFFF	✓	No
sysram6	64	0x200C 0000	0x200C FFFF	✓	No
sysram7	192	0x200D 0000	0x200F FFFF	✓	No
sysram8	128	0x2010 0000	0x2011 FFFF	Configurable access	✓

The MAX32690-specific AHB bus controllers can access the SRAM to use as general storage or working space.

The entirety of the SRAM memory space on the MAX32690 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 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 128KB 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 controllers do not trigger a bit-banding operation and instead result in an AHB bus error.

3.2.5 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 MAX32690, all device-specific module registers are mapped to this memory area, as well as any local memory buffers or FIFOs 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) 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 controller accesses the peripheral bit-banding alias region, the bit-banding remapping operation does not take place. In this case, the bit-banding alias region appears to be a non-implemented memory area (causing an AHB bus error).

On the MAX32690, 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.

3.2.6 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.

3.2.7 External Memory Space

The external memory space area 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 MAX32690 supports a HyperBus/Xccela interface for code an data storage, a SPIXR interface for external data storage, and an SPI XiP (SPIXF) interface for external code storage and execution. These interfaces support up to 512MB and are mapped from address 0x6000 0000 to 0x8000 0000.

3.2.8 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 MAX32690 does not implement this memory area.*

3.2.9 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 controllers, such as the DMA 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 nested interrupt vector controller (NVIC) controller, and the flash breakpoint controller.

3.2.10 System Area (Vendor Defined)

The system area (vendor defined) memory space is reserved for vendor (system integrator) specific functions not handled by another memory area. It is defined from byte address range 0xE010 0000 to 0xFFFF FFFF. *The MAX32690 does not implement this memory region.*

3.3 AHB Interfaces

This section details memory accessibility on the AHB and the organization of AHB controller and target instances.

3.3.1 Arm Core AHB Interfaces

3.3.1.1 I-Code

This AHB controller 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 controller is used to fetch instructions from the internal flash memory. Instructions fetched by this bus controller are returned by the instruction cache, which in turn triggers a cache line fill cycle to fetch instructions from the internal flash memory when a cache miss occurs.

3.3.1.2 D-Code

This AHB controller 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 controller has access to the internal flash memory and information block.

3.3.1.3 System

This AHB controller 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 controller.

3.3.2 Standard DMA

The standard DMA AHB controller is used for accessing the system RAM, APB mapped peripherals (through the AHB-to-APB bridge), and the AHB mapped peripheral RAMs (SPI0, SPI1, and the USB endpoints).

3.3.3 Crypto Toolbox

This AHB controller enables the CTB to access to the system RAM.

3.3.4 USB

The AHB controller enables the USB to access the system RAMs.

3.3.5 AHB Targets

3.3.5.1 Standard DMA

The standard DMA AHB target 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.

3.3.5.2 SPI3/SPI4

The SPI3 and SPI4 are AHB targets that have access to all off-core memory areas accessible by the System bus. It does not have access to the Arm Private Peripheral Bus area.

3.4 Peripheral Register Map

3.4.1 APB Peripheral Base Address Map

Table 3-2 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 register's offset.

Table 3-2: 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
Crypto Toolbox	CTB_	0x4000 1000	0x4000 1FFF
Watchdog Timer 0	WDT0_	0x4000 3000	0x4000 33FF
Memory Protection Key	AESKEY_	0x4000 5000	0x4000 53FF
Trim System Initialization	TRIMSIR_	0x4000 5400	0x4000 57FF
General Control Function	GCFR_	0x4000 5800	0x4000 5BFF
Real time Clock	RTC_	0x4000 6000	0x4000 63FF
Wakeup Timer 0	WUT0_	0x4000 6400	0x4000 67FF
Wakeup Timer 1	WUT1_	0x4000 6600	0x4000 6BFF
Power Sequencer	PWRSEQ_	0x4000 6800	0x4000 6FFF
Miscellaneous Control	MCR_	0x4000 6C00	0x4000 6FFF
PUF	PUF_	0x4000 7000	0x4000 73FF
GPIO Port 0	GPIO0_	0x4000 8000	0x4000 8FFF
GPIO Port 1	GPIO1_	0x4000 9000	0x4000 9FFF
GPIO Port 2	GPIO2_	0x4000 A000	0x4000 AFFF
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
I^2C 0	I2C0_	0x4001 D000	0x4001 DFFF
I^2C 1	I2C1_	0x4001 E000	0x4001 EFFF
I^2C 2	I2C2_	0x4001 F000	0x4001 FFFF
SPIXF Controller	SPIXFM_	0x4002 6000	0x4002 6FFF
SPIXF Host Controller	SPIXFC_	0x4002 7000	0x4002 7FFF
Standard DMA	DMA_	0x4002 8000	0x4002 8FFF
Flash Controller 0	FLC0_	0x4002 9000	0x4002 93FF
Flash Controller 1	FLC1_	0x4002 9400	0x4002 97FF
Instruction-Cache Controller 0 (CM4)	ICC0_	0x4002 A000	0x4002 A3FF
Instruction Cache Controller 1 (RV32)	ICC1_	0x4002 A800	0x4002 ABFF
SPIXF Cache Controller	SFCC_	0x4002 F000	0x4002 EFFF
External Memory Cache Controller	EMCC_	0x4003 3000	0x4004 2FFF
Analog-to-Digital Converter	ADC_	0x4003 4000	0x4003 4FFF
HyperBus/Xccela	HPB_	0x4003 9000	0x4003 9FFF
SPIXR Host Controller	SPIXR_	0x4003 A000	0x4003 AFFF
Pulse Train Engine	PT_	0x4003 C000	0x4003 CFFF
1-Wire Controller	OWM_	0x4003 D000	0x4003 DFFF
Semaphore	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
SPI 0	SPI0_	0x4004 6000	0x4004 6FFF
SPI 1	SPI1_	0x4004 7000	0x4004 7FFF
SPI 2	SPI2_	0x4004 8000	0x4004 8FFF
TRNG	TRNG_	0x4004 D000	0x4004 DFFF

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
I^2S	I2S_	0x4006 0000	0x4006 0FFF
CAN 0	CAN0_	0x4006 4000	0x4006 4FFF
CAN 1	CAN1_	0x4006 5000	0x4006 5FFF
Low-Power General Control	LPGCR_	0x4008 0000	0x4008 03FF
Low-Power GPIO 3 (LPGPIO0)	GPIO3_	0x4008 0400	0x4008 05FF
Low-Power Watchdog Timer 1 (LPWDT0)	WDT1_	0x4008 0800	0x4008 0BFF
Low-Power Timer 4 (LPTMR0)	TMR4_	0x4008 0C00	0x4008 0FFF
Low-Power Timer 5 (LPTMR1)	TMR5_	0x4008 1000	0x4008 13FF
Low-Power UART 3 (LPUART0)	UART3_	0x4008 1400	0x4008 17FF
Low-Power Comparator	LPCMP0_	0x4008 8000	0x4008 83FF
USB Hi-Speed Host	USBHS_	0x400B 1000	0x400B 1FFF
SPIXF Host Controller FIFO	SPIXFC_FIFO_	0x400B C000	0x400B CFFF
SPI3	SPI3_	0x400B E000	0x400B E3FF
SPI4	SPI4_	0x400B E400	0x400B E7FF

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. Support for selectable core operating voltage is provided, and the internal primary oscillator frequency is scaled based on the specific core operating voltage range selected.

4.1 Oscillator Sources

4.1.1 120MHz Internal Primary Oscillator (IPO)

The MAX32690 includes a 120MHz internal high-speed oscillator, referred to in this document as the IPO. The IPO is the fastest oscillator and draws the most power.

The IPO can optionally be powered down in *LPM* by setting the *GCR_PM.ipo_pd* field to 1.

The IPO can be selected as SYS_OSC using the following steps:

1. Enable the IPO by setting *GCR_CLKCTRL.ipo_en* to 1.
2. Wait until the *GCR_CLKCTRL.ipo_rdy* field reads 1, indicating the IPO is operating.
3. Set *GCR_CLKCTRL.sysclk_sel* to 4.
4. Wait until the *GCR_CLKCTRL.sysclk_rdy* field reads 1. The IPO is now operating as the SYS_OSC.

4.1.1.1 IPO Calibration

The IPO can be calibrated to improve accuracy. The calibration circuitry divides down the IPO to a value close to the 32.768kHz ERTCO frequency. The calibration hardware then increments or decrements a trim value to get the divided down frequency as close to the ERTCO frequency as possible. Each trim increment or decrement is approximately 205kHz. The following steps describe how to calibrate the IPO using the ERTCO.

1. Enable the ERTCO by setting *GCR_CLKCTRL.ertco_en* to 1.
2. Wait until *GCR_CLKCTRL.ertco_rdy* reads 1. The ERTCO is now operating.
3. Set the *FCR_AUTOCAL2.acdiv* field to 3,662. See the *FCR_AUTOCAL2.acdiv* field for additional information.
4. Set the *FCR_AUTOCAL1.inittrm* field to 0x40.
5. Set the *FCR_AUTOCAL0.acrun*, *FCR_AUTOCAL0.acen*, and *FCR_AUTOCAL0.ldtrm* fields to 1 by performing a bitwise OR of the *FCR_AUTOCAL0* register with 0x7.
6. Wait 10ms for the trim to complete.
   a. The calculated trim is loaded to the *FCR_AUTOCAL0.hirc96mactmrout* field and is used by the hardware as long as *FCR_AUTOCAL0.acen* is set to 1.
7. Set the *FCR_AUTOCAL0.acrun* field to 0 to stop the calibration.

4.1.2 32MHz External RF Oscillator (ERFO)

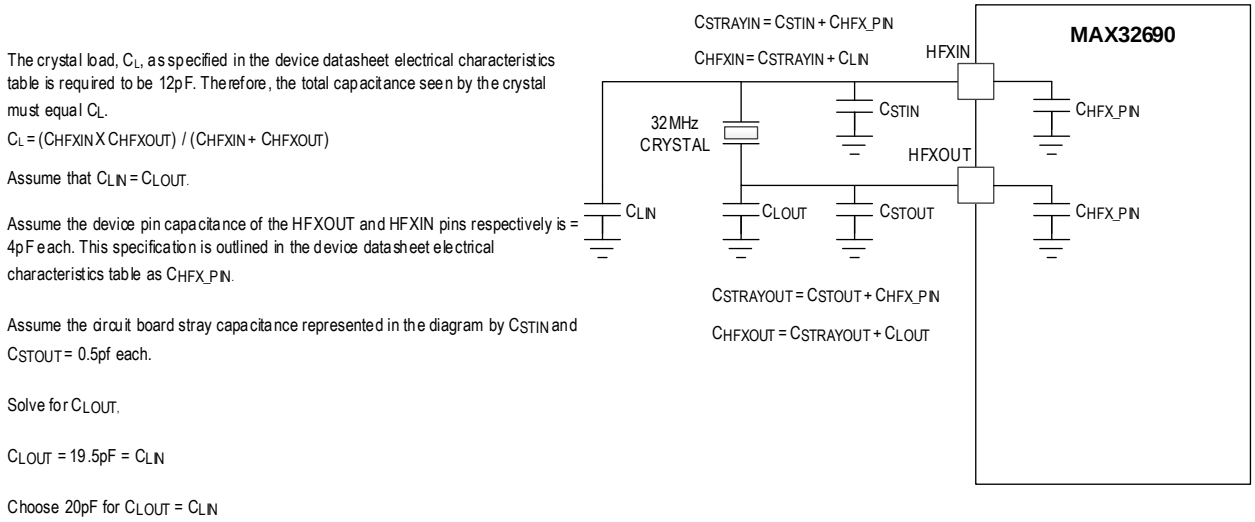
The ERFO is the oscillator that directly drives the Bluetooth radio. It can also be selected as the SYS_OSC. It is important to use the correct capacitor values on the PCB when connecting the crystal. *Figure 4-1* depicts the method to determine the capacitor values C_{LIN} and C_{LOUT}. The Bluetooth LDOs (LDORX and LDOTX) must be enabled to use the ERFO. Enable the Bluetooth LDOs by setting *GCR_BTLELDOCTRL.ldorxen* and *GCR_BTLELDOCTRL.ldotxen* to 1.

The following steps describe how to use the ERFO as the SYS_OSC:

1. Enable the internal secondary oscillator (ISO) by setting *GCR_CLKCTRL.iso_en* to 1.
2. Wait until *GCR_CLKCTRL.iso_rdy* reads 1. The ISO is now operating.
3. Enable the ERFO by setting *GCR_CLKCTRL.erfo_en* to 1.
4. Wait until *GCR_CLKCTRL.erfo_rdy* reads 1. The ERFO is now operating.
5. Set *GCR_CLKCTRL.sysclk_sel* = 2, selecting the ERFO as the SYS_OSC.
6. Wait until *GCR_CLKCTRL.sysclk_rdy* is set. The ERFO is now operating as the SYS_OSC.

Note: The ISO must remain enabled while the ERFO is operating as the SYS_OSC.

Figure 4-1: Example 32MHz Crystal Capacitor Determination



4.1.2.1 ERFO Kick-Start

The device includes an internal kick-start circuit that improves the startup time for the ERFO. This kick-start circuit enables the software to programmatically determine the optimal settings required for the fastest startup time for a given external oscillator. The ERFO startup time is crystal, temperature, and layout dependent; therefore, it is suggested that the software settings required for optimal startup time are performed on the final system-level design.

4.1.2.2 Programmatic Determination of the Optimum Kick-Start Settings

The following steps describe a methodology for determining the optimum settings for the kick-start circuit in a given application:

1. Disable the ERFO by setting *GCR_CLKCTRL.erfo_en* to 0.
2. Disable the kick-start circuit by setting the following fields to 0:
   a. *FCR_XO32MKS.clksel*
   b. *FCR_XO32MKS.driver*
   c. *FCR_XO32MKS.clk*
3. Select either the ISO or IPO to use to kick start the ERFO as follows:
   a. For the ISO set the *FCR_XO32MKS.clksel* field to 2.
   b. For the IPO set the *FCR_XO32MKS.clksel* field to 3.
4. Enable the Bluetooth LE LDOs by setting the *GCR_BTLELDOCTRL.ldorxen* and *GCR_BTLELDOCTRL.ldotxen* fields to 1.
   Note: The Bluetooth LE LDOs must be enabled to use the ERFO regardless of whether the Bluetooth LE peripheral is used.
5. Enable the oscillator selected in step 3, used to kick start the ERFO, by setting the appropriate clock enable as follows:
   a. For the IPO, set *GCR_CLKCTRL.ipo_en* to 1 and read the *GCR_CLKCTRL.ipo_rdy* until it reads 1.
   b. For the ISO, set *GCR_CLKCTRL.iso_en* to 1 and read the *GCR_CLKCTRL.iso_rdy* until it reads 1.
6. Configure a timer for counter mode using the oscillator selected in step 3 as the timer clock. See *Timers (TMR/LPTMR)* for details of timer configuration and modes.
7. The number of kick-start pulses supported is from 1 to 127. The following steps should be performed by incrementing the kick-start count by 1 on each iteration while measuring the time it takes to start the ERFO. Once the fastest startup time is determined, the settings should be saved and used when the software starts the ERFO.
   a. Set the kick-start pulse count, *FCR_XO32MKS.clk*, to the number of kick-start pulses to test. This is the iteration number from 1 to 127.
   b. Enable the kick-start circuit by setting *FCR_XO32MKS. en* to 1.
   b. Start the configured timer.
   c. Enable the ERFO by setting *GCR_CLKCTRL.erfo_en* to 1.
   d. Read the *GCR_CLKCTRL.erfo_rdy* field until it reads 1.
   e. Stop the timer and determine the elapsed time.
   f. Disable the kick-start circuit by setting *FCR_XO32MKS. en* to 0.
   g. Repeat steps *a* to *f* until all supported kick-start pulses are tested and determine the optimum kick-start pulses for the fastest ERFO startup time.

4.1.2.3 Using the ERFO Kick-Start

Perform the following steps when enabling the ERFO to use the kick-start circuity:

1. Set the kick-start pulse count by setting the *FCR_XO32MKS.clk* field.
2. Enable the kick-start circuit by setting *FCR_XO32MKS.en* to 1.
3. Enable the ERFO by setting *GCR_CLKCTRL.erfo_en* to 1.
4. Read the *GCR_CLKCTRL.erfo_rdy* field until it reads 1.
5. Disable the kick-start circuit by setting *FCR_XO32MKS.en* to 0.

4.1.3 60MHz Internal Secondary Oscillator (ISO)

The ISO 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 a POR.

4.1.4 7.3728MHz Internal Baud Rate Oscillator (IBRO)

The IBRO is an exceptionally low-power internal oscillator that can be selected as SYS_OSC. The IBRO can optionally be used as a dedicated baud rate clock for the UARTs. This is useful if the SYS_OSC selected does not allow the targeted UART baud rate.

This oscillator can optionally be automatically powered down when in *LPM* and *UPM* by setting register bit *GCR_PM.ibro_pd*.

This oscillator is enabled by default at power-up.

4.1.5 32.768kHz External Real-Time Clock Oscillator (ERTCO)

The ERTCO is an exceptionally low-power 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). Refer to the device data sheet for details on alternate function mapping.

The ERTCO is the dedicated clock for the RTC. If the RTC is enabled, the ERTCO must be enabled, independent of the selection of SYS_OSC. This oscillator is disabled at power-up.

4.1.5.1 Enabling the ERTCO

Perform the following steps to enable the ERTCO:

1. Enable the ERTCO by setting *GCR_CLKCTRL.ertco_en* to 1.
2. Wait until *GCR_CLKCTRL.ertco_rdy* field reads 1.
    a. The ERTCO is now operating.
3. If setting the ERTCO as the system oscillator, set *GCR_CLKCTRL.sysclk_sel* to 6.

Article I. 4. Wait until *GCR_CLKCTRL.sysclk_rdy* reads 1.

    a. The ERTCO is now operating as the SYS_OSC.

Enable the ERTCO to operate in all low power modes by setting *MCR_CTRL.ertco_en* to 1.

4.1.6 8kHz Internal Nano-Ring Oscillator (INRO)

The INRO is an ultra-low-power internal oscillator that can be selected as SYS_OSC. The INRO is enabled at power-up and cannot be disabled by software. The INRO is not an accurate clock source and may vary by more than ±50%.

4.2 System Oscillator (SYS_OSC)

The MAX32690 supports multiple clock sources as the SYS_OSC. Each oscillator, description, and nominal frequency are shown in *Table 4-1*. The ERTCO requires an external crystal or input clock for operation. An external clock source, CLKEXT, is supported on P0.23, Alternate Function 1.

Table 4-1: Oscillators, Descriptions, and Nominal Frequencies

Oscillator	Description	Nominal Frequency
IPO	Internal Primary Oscillator	120MHz
ISO	Internal Secondary Oscillator	60MHz
INRO	Internal Nano-Ring Oscillator	8kHz
IBRO	Internal Baud Rate Oscillator	7.3728MHz
ERTCO	External Real-Time Clock Oscillator	32.768kHz
ERFO	External RF Oscillator	32MHz
CLKEXT	External Clock	Up to 80MHz

4.2.1 System Oscillator Selection

Set the system oscillator using the *GCR_CLKCTRL.sysclk_sel* field. Before selecting an oscillator as the system oscillator, the oscillator source must first be enabled and ready. See *Oscillator Sources* for each oscillator's detailed description for the required steps to enable the oscillator and select it as a system oscillator.
When the *GCR_CLKCTRL.sysclk_sel* is modified, the hardware clears the *GCR_CLKCTRL.sysclk_rdy* field, and there is a delay until the switchover is complete. When the switchover to the selected SYS_OSC is complete, the *GCR_CLKCTRL.sysclk_rdy* field is set to 1 by the hardware. The software must verify that the switchover is complete before continuing operation.

4.2.2 System Clock (SYS_CLK)

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

CAUTION: When switching the SYS_OSC or touching the SYS_OSC prescaler (*GCR_CLKCTRL.sysclk_div*), 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.

Equation 4-1: System Clock Scaling

$$SYS\_CLK = \frac{SYS\_OSC}{2^{sysclk\_div}}$$

GCR_CLKCTRL.sysclk_div 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 core, the RV32 core, and all Advanced High-Performance Bus (AHB) masters in the system. SYS_CLK generates the following internal clocks as shown below:

- Advanced High-Performance Bus (AHB) Clock
  - $HCLK = SYS\_CLK$
- Advanced Peripheral Bus (APB) Clock,
  - $PCLK = \frac{SYS\_CLK}{2}$

The RTC uses the ERTCO for its clock source. Optionally, the RTC can run using the INRO. See *Real-Time Clock (RTC)* for details on using the INRO for the RTC.

All oscillators are reset to their reset default state during:

- POR
- System Reset

Oscillator settings are *not* reset during:

- Soft Reset
- Peripheral Reset

Table 4-2 shows each oscillator's enabled state for each type of reset source in the MAX32690. Table 4-3 details each reset source's effect on the system clock selection and system clock prescaler settings.

Table 4-2: Reset Sources and Effect on Oscillator Status

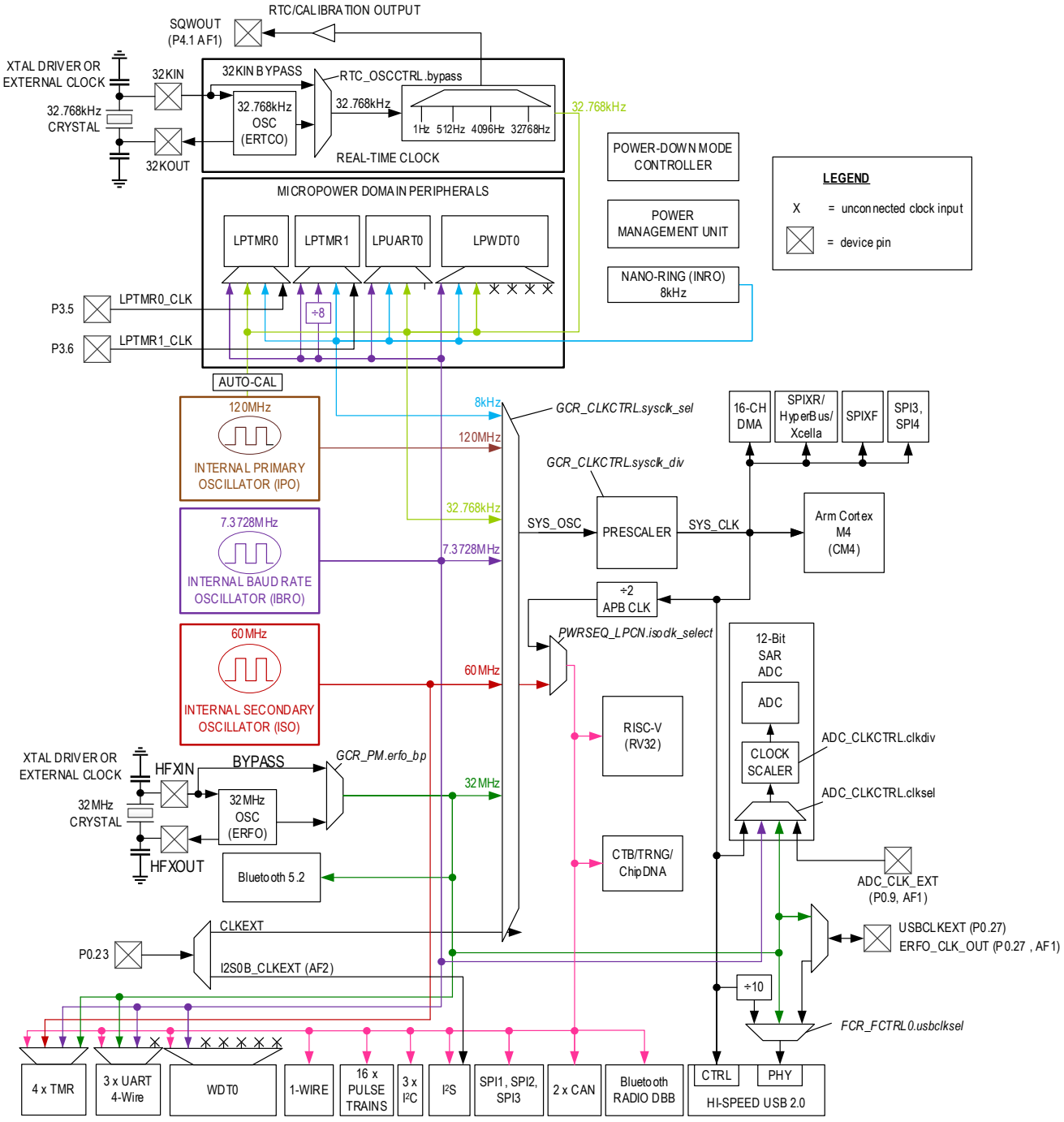
	Reset Source			
Oscillator	**POR**	**System**	**Soft**	**Peripheral**
IPO	Disabled	Disabled	Retains State	Retains State
ISO	Enabled	Enabled	Retains State	Retains State
ERFO	Disabled	Disabled	Retains State	Retains State
IBRO	Enabled	Enabled	Enabled	Enabled
INRO	Enabled	Enabled	Enabled	Enabled
ERTCO	Disabled	Retains State	Retains State	Retains State

Table 4-3: Reset Sources and Effect on System Oscillator Selection and Prescaler

	Reset Source				
Clock Field	**POR**	**System**	**Watchdog**	**Soft**	**Peripheral**
System Oscillator *GCR_CLKCTRL.sysclk_sel*	0 (ISO)	0 (ISO)	0 (ISO)	Retains State	Retains State
System Clock Prescaler *GCR_CLKCTRL.sysclk_div*	1	1	1	Retains State	Retains State

Figure 4-2 shows a high-level diagram of the MAX32690 clock tree.

Figure 4-2: MAX32690 Clock Block Diagram



4.3 Operating Modes

The MAX32690 includes multiple operating modes and the ability to fine-tune power options to optimize performance and power. The system supports the following operating modes:

- *ACTIVE*
- *SLEEP*
- *LOW-POWER Mode* (*LPM*)
- *MICROPOWER Mode* (*UPM*)
- *STANDBY*
- *BACKUP*

4.3.1 ACTIVE Mode

The CM4 and RV32 cores can execute application software in this mode, and all digital and analog peripherals are available on demand. Dynamic clocking disables peripherals not in use, providing the optimal mix of high performance and low-power consumption. The CM4 has access to all system RAM by default. The RV32 has access to *sysram8* and can configure *sysram8* for exclusive access.

4.3.2 SLEEP

SLEEP is a low-power mode that suspends the CPU with a fast wake-up time to *ACTIVE*. It is like *ACTIVE,* 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 are enabled.

The device returns to *ACTIVE* from any internal or external interrupt. The device status is as follows:

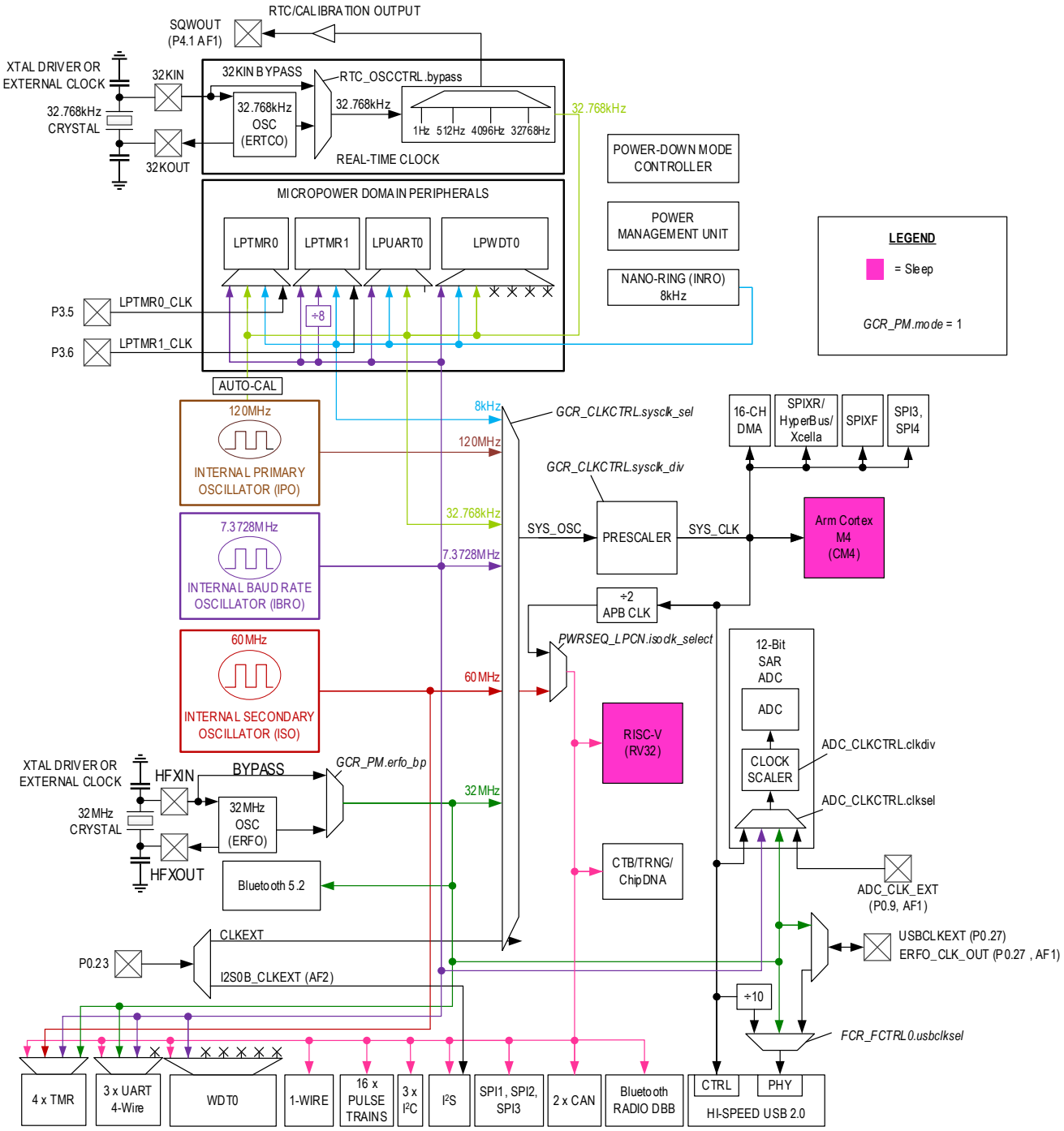
- The CM4 (CPU0) is sleeping.
- The RV32 (CPU1) is sleeping.
- Standard DMA is available for use.
- All peripherals are on unless explicitly disabled before entering *SLEEP*.

4.3.2.1 Entering SLEEP

Place the CM4 in *SLEEP* by performing the following steps.

1. Configure any desired wake-up function.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKST0*
   b. PWRSEQ_LPWKST1
   c. PWRSEQ_LPWKST2
   d. PWRSEQ_LPWKST3
   e. PWRSEQ_LPWKST4
   f. PWRSEQ_LPPWST
3. Set SCR.sleepdeep to 0.
4. Perform a wait for interrupt (WFI) or wait for event (WFE) instruction.

Figure 4-3: SLEEP Mode Clock Control



4.3.3 Low-Power Mode (LPM)

This mode is suitable for running the RV32 processor to collect and move data from enabled peripherals. The device status in *LPM* is as follows:

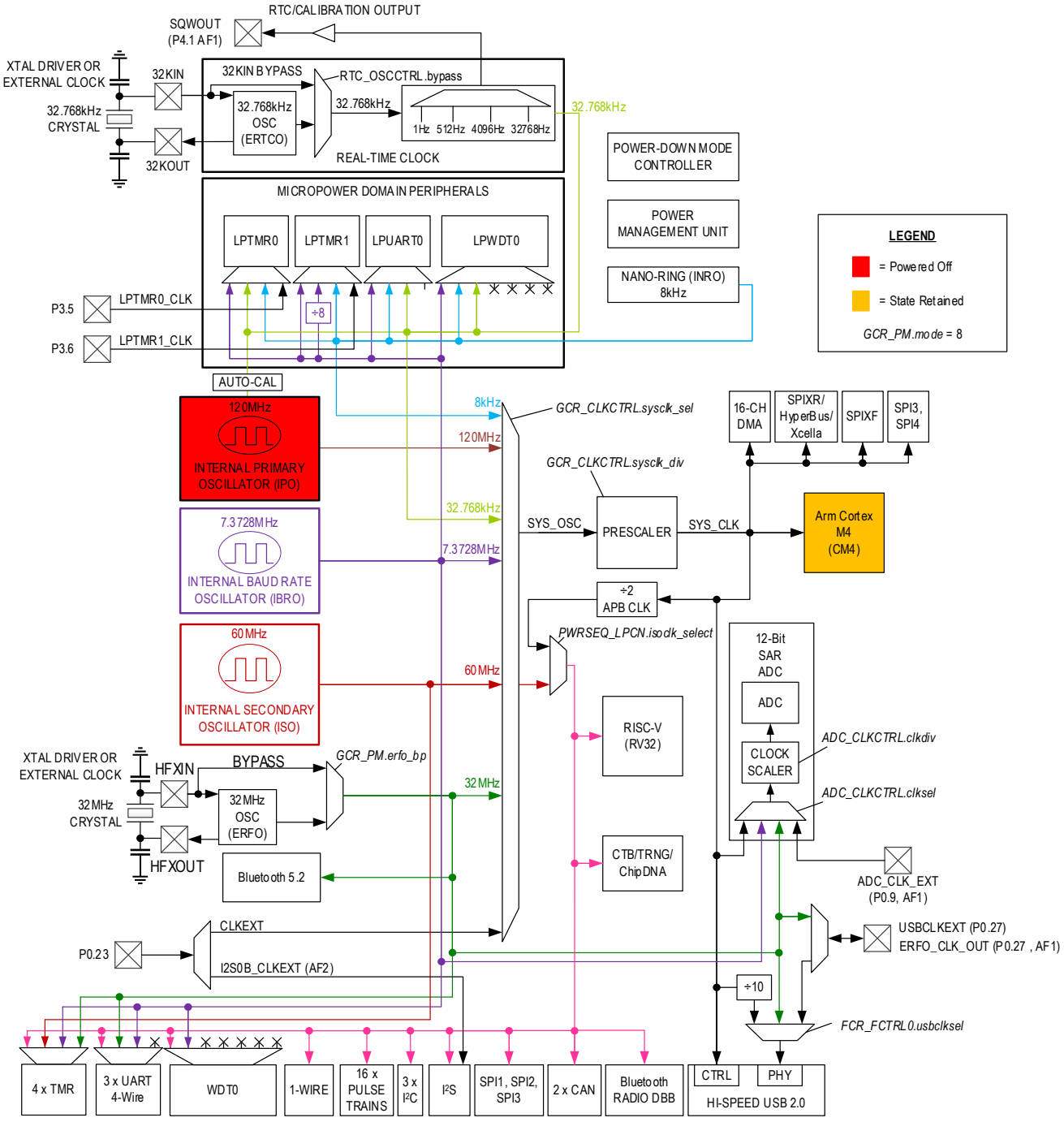
- The CM4, *sysram0 - sysram7* are in state retention.
- The RV32 can access SPI0, SPI1, SPI2, BLE, CAN0, CAN1, all UARTS, all timers, I^2C, 1-Wire®, the pulse train engine, I^2S, TRNG, the comparators, as well as *sysram8*.
- CRC, AES, MAA, and USB are powered down.
- The transition from *LPM* to *ACTIVE* is faster than from *BACKUP* to *ACTIVE* because system initialization is not required.
- The DMA is in state retention mode.
- *PWRSEQ_GP0* and *PWRSEQ_GP1* registers retain state.
- Choose the system PCLK or ISO as the clock source for the RV32 and all peripherals.
  - *PWRSEQ_LPCN.isoclk_select* defaults to use the ISO during LPM. Setting this field to 1 uses the PCLK.
  - IPO can optionally be powered down.
- The following oscillators are optionally enabled:
  - ERFO
  - IBRO
  - ERTCO
  - ISO
- INRO is enabled

4.3.3.1 Entering LPM

Entry into *LPM* should be managed between the two cores using *Multiprocessor Communications* to ensure both cores are in a known state when entering *LPM*. Place the device in *LPM* by performing the following steps.

1. Configure any desired wake-up function.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKST0*
   b. *PWRSEQ_LPWKST1*
   c. PWRSEQ_LPWKST2
   d. PWRSEQ_LPWKST3
   e. PWRSEQ_LPWKST4
   f. PWRSEQ_LPPWST
3. If required, enable the ERTCO by setting *MCR_CTRL.ertco_en* to 1.
4. Set SCR.sleepdeep to 1.
5. Set the *GCR_PM.mode* field to *LPM* (8).
6. Perform a WFI or WFE instruction.

Figure 4-4: Low-Power Mode (LPM) Clock and State Retention Diagram



4.3.4 Micropower Mode (UPM)

This mode is used for extremely low-power consumption while using a minimal set of peripherals to provide wake-up capability. The device status during *UPM* is as follows:

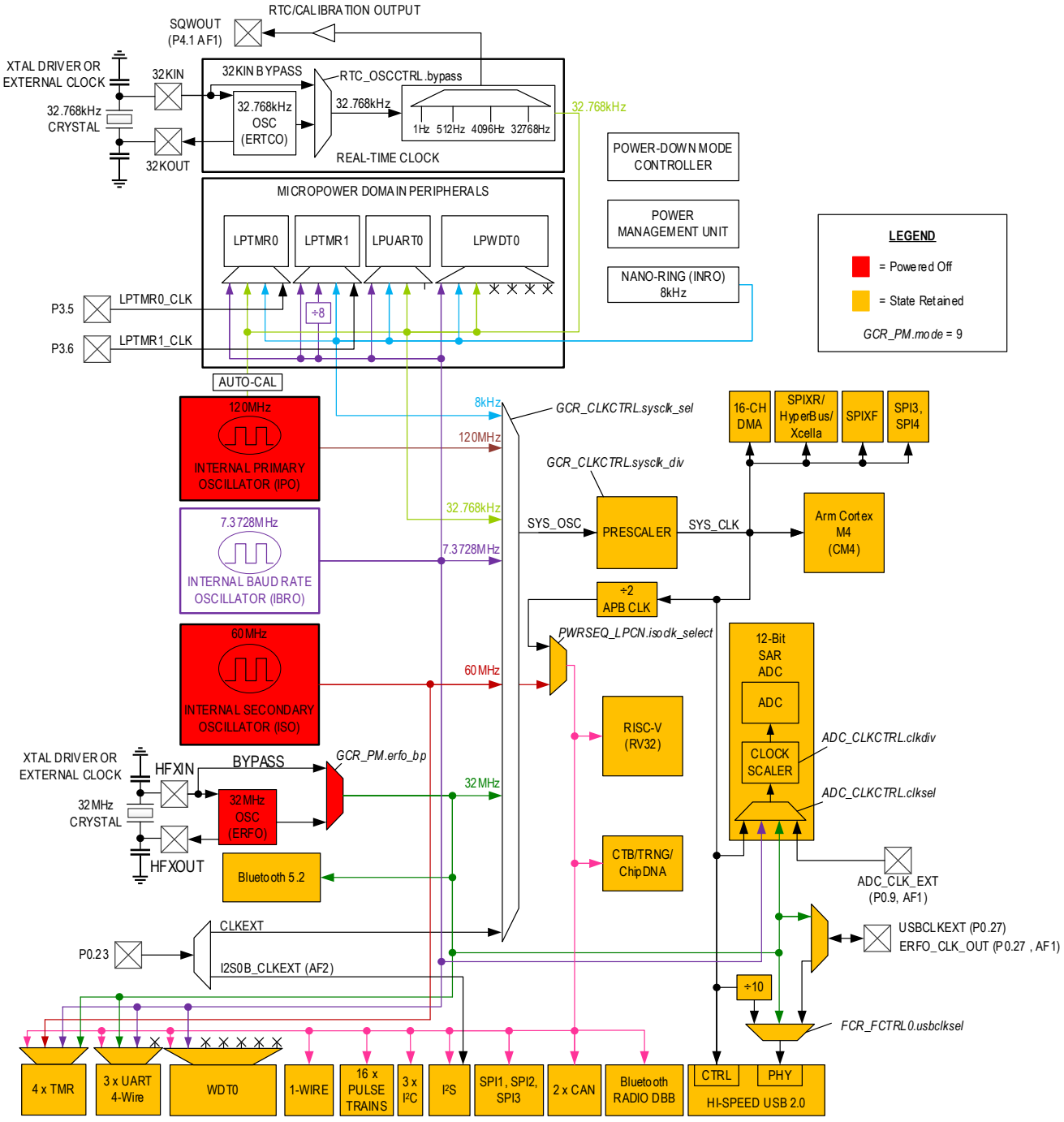
- Both the CM4 and RV32 are state retained.
- System state and all system SRAM are retained.
- The GPIO pins retain state.
- All non-micropower peripherals are state retained except:
  - USB, Bluetooth, CRC, AES, and the MAA are powered down.
- The following oscillators are powered down:
  - IPO
  - ISO
  - ERFO
- The following oscillators are enabled:
  - IBRO
  - ERTCO
  - INRO
- The following *UPM* peripherals are available for use to wake-up the device:
  - LPUART0 (UART3)
  - LPTMR0 (TMR4)
  - LPTMR1 (TMR5)
  - LPWDT0 (WDT1)
  - All comparators
  - GPIO

4.3.4.1 Entering UPM

Entering *UPM* mode requires both the CM4 and RV32 to cooperate. Synchronization is necessary for deterministic entry into *UPM*. The software should use *Multiprocessor Communications* to ensure both the CM4 and RV32 are in a known state before entering *UPM*. The following steps should be used to put the part into *UPM*.

1. Configure any desired wake-up function.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKST0*
   b. PWRSEQ_LPWKST1
   c. *PWRSEQ_LPWKST2*
   d. PWRSEQ_LPWKST3
   e. PWRSEQ_LPWKST4
   f. PWRSEQ_LPPWST
3. If required, enable the ERTCO by setting *MCR_CTRL.ertco_en* to 1.
4. Set SCR.sleepdeep to 1.
5. Set the *GCR_PM.mode* field to *UPM* (9).
6. Perform a WFI or WFE instruction.

Figure 4-5: UPM Clock and State Retention Block Diagram



4.3.5 STANDBY

This mode maintains the system operation while keeping time with the RTC. The device status in *STANDBY* is as follows:

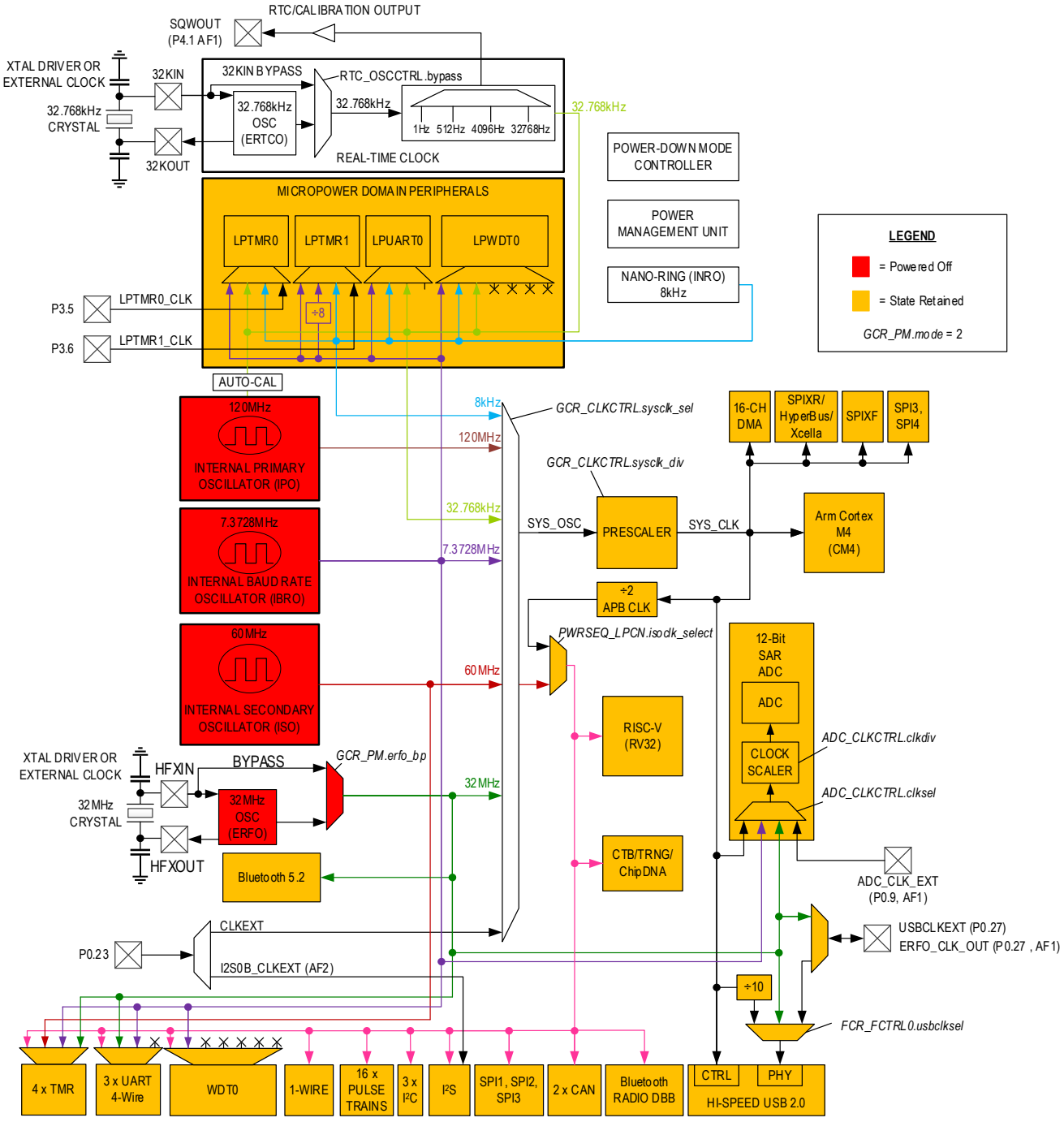
- Both the CM4 and the RV32 are state retained.
- The system state and all system RAM are retained.
- GPIO pins retain their state.
- The RTC is optionally enabled.
- The wake-up timers are optionally enabled.
- CMP0 is optionally enabled.
- All peripherals retain state except:
  - USB, Bluetooth, CRC, AES, and the MAA are powered down.
- *PWRSEQ_GP0* and *PWRSEQ_GP1* registers retain state.
- The following oscillators are powered down:
  - IPO
  - ISO
  - ERFO
  - IBRO
- INRO is on.
- ERTCO is optionally enabled.

4.3.5.1 Entering STANDBY

The software should use *Multiprocessor Communications* to ensure both the CM4 and RV32 are in a known state before entering *STANDBY*. The following steps should be used to put the part into *STANDBY*.

1. Configure any desired wake-up function.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKST0*
   b. PWRSEQ_LPWKST1
   c. PWRSEQ_LPWKST2
   d. PWRSEQ_LPWKST3
   e. PWRSEQ_LPWKST4
   f. PWRSEQ_LPPWST
3. If required, enable the ERTCO by setting *MCR_CTRL.ertco_en* to 1.
4. Set SCR.sleepdeep to 1.
5. Set the *GCR_PM.mode* field to *STANDBY* (2).
6. Perform a WFI or WFE instruction.

Figure 4-6: STANDBY Mode Clock and State Retention Block Diagram



4.3.6 BACKUP

This mode maintains the system RAM contents. The device status in *BACKUP* is as follows:

- The CM4 and RV32 are powered off.
- *sysram0* through *sysram6* , and *sysram8* can be independently configured to be state retained as shown in *Table 4-4*.
- All peripherals are powered off.
- *PWRSEQ_GP0* and *PWRSEQ_GP1* registers retain state.
- The following oscillators are powered down:
  - IPO
  - ISO
  - IBRO
  - ERFO
- INRO is on
- The following oscillators are optionally enabled:
  - ERTCO (The RTC peripheral can be turned off, but not the oscillator.)

Table 4-4: SRAM Retention By Address Range in BACKUP, System Reset, Watchdog Reset, and External Reset

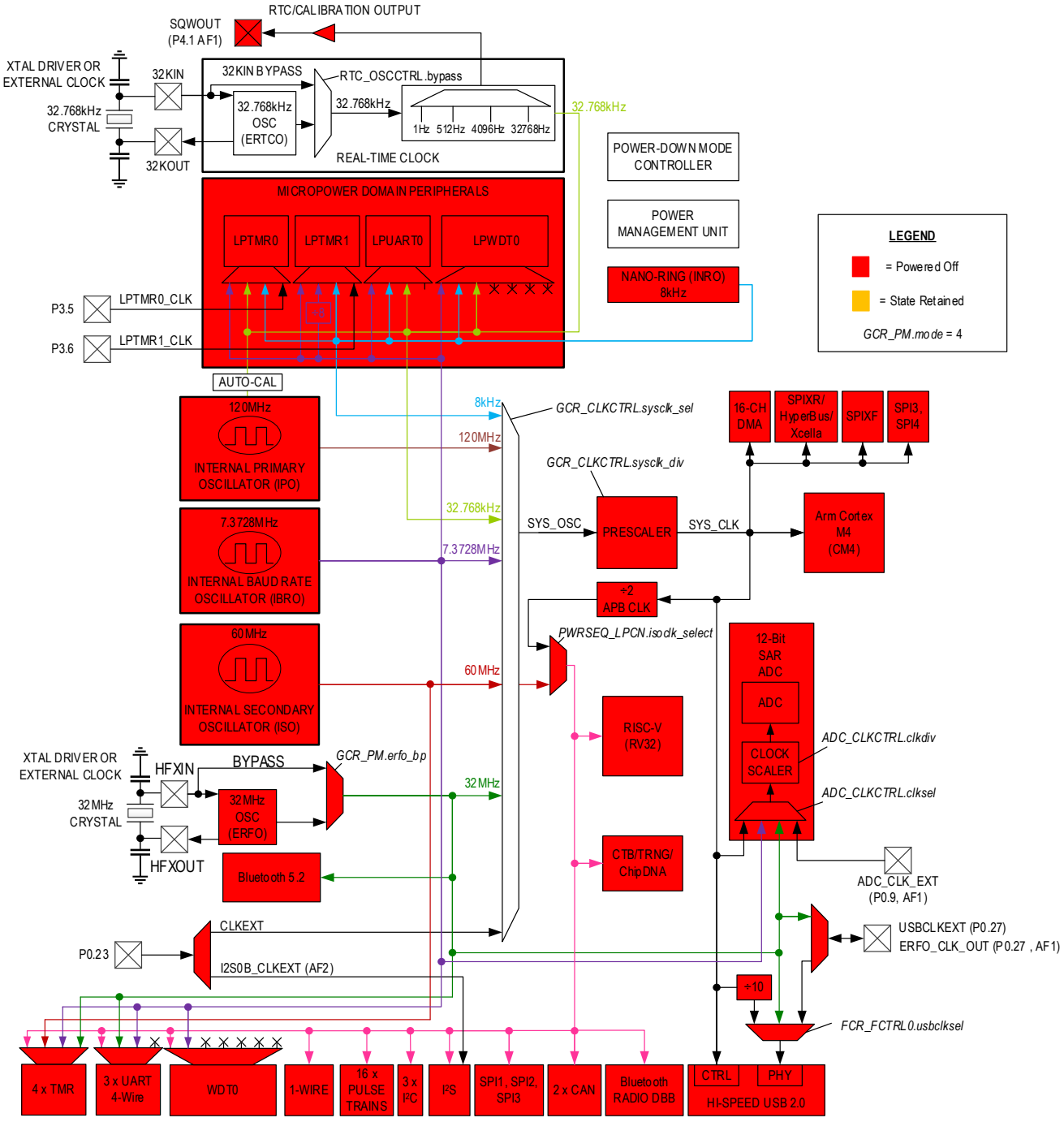
RAM Block #	State Retention Control	Address Range Retention	Size
sysram0	*PWRSEQ_LPCN.ramret0*	0x2000 A900 - 0x2000 1FFF	87.8KB
sysram1	*PWRSEQ_LPCN.ramret1*	0x2002 0000 - 0x2003 FFFF	128KB
sysram2	*PWRSEQ_LPCN.ramret2*	0x2004 0000 - 0x2005 BFFF	128KB
sysram3	*PWRSEQ_LPCN.ramret3*	0x2006 0000 - 0x2007 FFFF	128KB
sysram4	*PWRSEQ_LPCN.ramret4*	0x2008 0000 - 0x2009 FFFF	128KB
sysram5	*PWRSEQ_LPCN.ramret5*	0x200A 0000 - 0x200B FFFF	128KB
sysram6	*PWRSEQ_LPCN.ramret6*	0x200C 0000 - 0x200C FFFF	64KB
sysram7	N/A	N/A	N/A
sysram8	*PWRSEQ_LPCN.ramret8*	0x2010 0000 - 0x2011 FFFF	128KB

4.3.6.1 Entering BACKUP

The software should use *Multiprocessor Communications* to ensure both the CM4 and RV32 are in a known state before entering *BACKUP*. The following steps should be used to put the part into *BACKUP*.

1. Configure any desired wake-up function.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKST0*
   b. PWRSEQ_LPWKST1
   c. PWRSEQ_LPWKST2
   d. PWRSEQ_LPWKST3
   e. PWRSEQ_LPWKST4
   f. PWRSEQ_LPPWST
3. If required, enable the ERTCO by setting *MCR_CTRL.ertco_en* to 1.
4. Set SCR.sleepdeep to 1.
5. Set the *GCR_PM.mode* field to *BACKUP* (4).
6. Perform a WFI or WFE instruction.

Figure 4-7: BACKUP Mode Clock and State Retention Block Diagram



4.3.7 Operating Modes Wake-up Sources

In all operating modes other than *ACTIVE*, wake-up sources are required to resume *ACTIVE* operation. *Table 4-5* shows available wake-up sources for each operating mode of the MAX32690.

Note: Each wake-up source must be enabled individually except for the external reset (RSTN), which is hardware controlled.

Table 4-5: Wake-Up Sources for Each Operating Mode in the MAX32690

Operating Mode	Wake-Up Sources
SLEEP	Any enabled peripheral with interrupt capability; RSTN
LPM	SPI0, SPI1, SPI2, I²S, I2C0, I2C1, I2C2, CAN0, CAN1, UART0, UART1, UART2, UART3, BLE, WDT0, WDT1, WUT0, WUT1, TMR0, TMR1, TMR2, TMR3, TMR4, TMR5, All Comparators, RTC, RV32, GPIO, RSTN
UPM	All Comparators, UART3 (LPUART0), TMR4 (LPTMR0), TMR5 (LPTMR1), WDT1 (LPWDT0), WUT0, WUT1, RTC, GPIO, RSTN
STANDBY	CMP0, WUT0, WUT1, RTC, GPIO, RSTN
BACKUP	CMP0, WUT0, WUT1, RTC, GPIO, RSTN

4.4 Device Resets

Four device resets are available:

- Peripheral Reset
- Soft Reset
- System Reset
- Power-On Reset

On completion of any of the four reset cycles, all peripherals are reset. On completing 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.

The always-on domain (AoD) contents are only reset on power-cycling V_{CORE}, V_{DDA}, V_{DD3A}.

Each on-chip peripheral can also be reset to its POR default state using the two reset registers, *GCR_RST0* and *GCR_RST1*.

Table 4-6 shows the effects of the four reset types and five power modes.

Table 4-6: Reset and Low-Power Mode Effects

	Peripheral Reset[4]	Soft Reset[4]	System Reset[4]	POR	*ACTIVE*	*SLEEP*	*LPM*	*UPM*	*BACKUP*[3]
IPO	-	-	Off	Off	R	-	Off	Off	Off
ISO	-	-	On	Off	R	-	-	Off	-
ERTCO	-	-	Off	Off	FW	FW	FW	FW	FW
IBRO	-	-	Off	Off	R	-	Off	Off	Off
ERFO	-	-	Off	Off	R	-	Off	Off	Off
INRO	On	On	On	On	On	On	On	On	On
SYS_CLK	On	On	On[2]	On[2]	On	On	Off	Off	Off
CPU Clock	On	On	On	On	On	Off	Off	Off	Off
RTC	Reset	-	-	Reset	FW	FW	FW	FW	FW
WUT0, WUT1	Reset	-	Reset	Reset	FW	FW	FW	FW	FW
WDT0, WDT1	Reset	Reset	Reset	Reset	R	Off	Off	Off	Off
GPIO0-GPIO4	Reset	Reset	Reset	Reset	R	-	-	-	-
All Other Peripherals	Reset	Reset	Reset	Reset	R	-	Off	Off	Off
Always-On Domain	-	-	-	Reset	-	-	-	-	-
RAM Retention	-	-	-	Reset	On	On	On	On	FW

	Peripheral Reset[4]	Soft Reset[4]	System Reset[4]	POR	*ACTIVE*	*SLEEP*	*LPM*	*UPM*	*BACKUP*[3]

Table key:
FW = Controlled by software
On = Enabled by hardware (Cannot be disabled)
Off = Disabled by hardware (Cannot be enabled)
- = No Effect
R = Restored to previous *ACTIVE* setting when exiting *LPM*, restored to system reset state when exiting *BACKUP* or *STORAGE*.

1: The AoD is only reset on power-cycling V_{CORE}, V_{DDA}, V_{DD3A}
2: On a system reset or POR, the ISO is automatically set as the SYS_OSC.
3: A system reset occurs when returning from *BACKUP*.
4: Peripheral, soft, and system resets are initiated by software through the *GCR_RST0* register. System reset can also be triggered by the RSTN device pin or a watchdog reset.

4.4.1 Peripheral Reset

A peripheral reset resets all peripherals. The CPUs retain their 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_RST0*.*periph* to 1. The reset is completed immediately upon setting *GCR_RST0*.*periph* to 1.

4.4.2 Soft Reset

A soft reset is the same as a peripheral reset, except it also resets the GPIO to its POR state.

To start a soft reset, set *GCR_RST0*.*soft* = 1. The reset is completed immediately upon setting *GCR_RST0*.*soft* = 1.

4.4.3 System Reset

A system reset is the same as a 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. Set *GCR_RST0*.*sys* to 1 to start a system reset from software.

4.4.4 Power-On Reset

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

4.5 Unified Internal Cache Controllers (ICC)

The MAX32690 includes two unified internal cache controllers. ICC0 is the cache controller used for the CM4. ICC1, if enabled as a cache controller, is dedicated to the RV32 core.

When enabled, both caches, ICC0 and ICC1, include a line buffer, tag RAM, and a 16KB 2-way set associative RAM.

4.5.1 Enabling the Instruction Cache Controller

Perform the following steps to enable an ICC:

1. Write 0 to the *GCR_PCLKDIS1*.*syscache* to enable the peripheral clock to the ICC.
2. Write 1 to the *ICCn_INVALIDATE* register to ensure the cache is flushed when it is enabled.
3. Set *ICCn_CTRL*.*en* to 1.
4. Read *ICCn_CTRL*.*rdy* until it returns 1.

4.5.2 Disabling the ICC

Disable an ICC instance by setting *ICCn_CTRL*.*en* to 0.

4.5.3 Invalidating the ICC Cache and Tag RAM

The system configuration register (*GCR_SYSCTRL*) includes a field for flushing the ICC0 cache. Setting *GCR_SYSCTRL.icc0_flush* to 1 flushes the ICC0 16KB cache and the tag RAM.

Invalidate the contents of a specific ICC instance by setting the *ICCn_INVALIDATE* register to 1. Once invalidated, the system flushes the cache. Read the *ICCn_CTRL.rdy* field until it returns 1 to determine when the flush is completed.

4.5.4 Flushing the ICC

Flush ICC0 using the *GCR_SYSCTRL* register. Set *GCR_SYSCTRL.icc0_flush* to 1 to immediately flush the contents of the 16KB cache and tag RAM.

Flush ICC1 using the RV32 control register (*FCR_URVCTRL*). Set *FCR_URVCTRL.iflushen* to 1 to immediately flush the contents of the 16KB cache and tag RAM.

4.5.5 Internal Cache Control Registers (ICC)

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 4-7: Internal Cache Controller Register Summary

Offset	Register	Name
[0x0000]	*ICCn_INFO*	*Cache ID Register*
[0x0004]	*ICCn_SZ*	*Cache Memory Size Register*
[0x0100]	*ICCn_CTRL*	*Instruction Cache Control Register*
[0x0700]	*ICCn_INVALIDATE*	*Instruction Cache Controller Invalidate Register*

4.5.6 ICC0 Register Details

Table 4-8: ICC0 Cache Information Register

ICC0 Cache Information			**ICCn_INFO**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:10	id	R	-	**Cache ID** This field returns the ID for this cache instance.
9:6	partnum	R	-	**Cache Part Number** This field returns the part number indicator for this cache instance.
5:0	relnum	R	-	**Cache Release Number** This field returns the release number for this cache instance.

Table 4-9: ICC0 Memory Size Register

ICC0 Memory Size			**ICCn_SZ**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	mem	R	-	**Addressable Memory Size** This field indicates the size of addressable memory by this cache controller instance in 128KB units.
15:0	cch	R	-	**Cache Size** This field returns the size of the cache RAM in 1KB units. 16: 16KB Cache RAM

Table 4-10: ICC0 Cache Control Register

ICC0 Cache Control			ICCn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	R/W	-	**Reserved**
16	rdy	R	-	**Ready** This field is cleared by hardware anytime the cache as a whole is invalidated (including a POR). The hardware automatically sets this field to 1 when the invalidate operation is complete, and the cache is ready. 0: Cache invalidation 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**
0	en	R/W	0	**Cache Enable** Set this field to 1 to enable the cache. Setting this field to 0 invalidates the cache contents, and the line fill buffer handles reads. 0: Disabled. 1: Enabled.

Table 4-11: ICC0 Invalidate Register

ICC0 Invalidate			ICCn_INVALIDATE	[0x0700]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	invalid	W	-	**Invalidate** Writing any value to this register invalidates the cache.

4.6 RAM Memory Management

This device has many features for managing the on-chip RAM. The on-chip RAM includes data RAM, instruction and data cache for the CM4, and instruction and data cache for the RV32, and peripheral FIFOs.

4.6.1 On-Chip Cache Management

The MAX32690 includes two instruction cache controllers for code fetches from the flash memory. The caches can be enabled, disabled, and zeroized or flushed. See *Low-Power General Control Registers Details* for details.

4.6.2 RAM Zeroization

The GCR memory zeroize register, *GCR_MEMZ*, clears memory for software or security reasons. Zeroization writes all zeros to the specified memory.

The following SRAM memories can be zeroized:

- Each of the system RAMs can be individually zeroized by setting the respective *GCR_MEMZ* bit.
  - *GCR_MEMZ.ram0*
  - *GCR_MEMZ.ram1*
  - *GCR_MEMZ.ram2*
  - *GCR_MEMZ.ram3*
  - *GCR_MEMZ.ram4*
  - *GCR_MEMZ.ram5*
  - *GCR_MEMZ.ram6*
  - *GCR_MEMZ.ram7*
  - *GCR_MEMZ.ram8*
- ICC0 16KB Cache:
  - Enabling ICC0 from a disabled state zeroizes the cache RAM.
  - Setting *GCR_MEMZ.icc0* to 1 also zeroizes the cache RAM.
- ICC1 16KB Cache, if enabled:
  - Write 1 to *GCR_MEMZ.icc1*.

The system SRAM banks with corresponding bank sizes, base address, and ending addresses are shown in *Table 4-12*.

Table 4-12 RAM Block Size and Base Address

System RAM Block #	Size	Base Address	End Address
sysram0	128KB	0x2000 0000	0x2001 FFFF
sysram1	128KB	0x2002 0000	0x2003 FFFF
sysram2	128KB	0x2004 0000	0x2005 BFFF
sysram3	128KB	0x2006 0000	0x2007 FFFF
sysram4	128KB	0x2008 0000	0x2009 FFFF
sysram5	128KB	0x200A 0000	0x200B FFFF
sysram6	64KB	0x200C 0000	0x200C FFFF
sysram7	192KB	0x200D 0000	0x200F FFFF
sysram8	128KB	0x2010 0000	0x2011 FFFF

4.7 Miscellaneous Control Registers (MCR)

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 4-13: Miscellaneous Control Register Summary

Offset	Register Name	Description
[0x0000]	*MCR_ECCEN*	Error Correction Coding Enable Register
[0x0004]	*MCR_IPO_MTRIM*	IPO Manual Trim Register
[0x0008]	*MCR_OUTEN*	Miscellaneous Output Enable Register
[0x000C]	*MCR_CMP_CTRL*	Comparator 0 Control Register
[0x0010]	*MCR_CTRL*	Miscellaneous Control Register
[0x0020]	*MCR_GPIO4_CTRL*	GPIO4 Pin Control Register
[0x0040]	*MCR_CWD0*	Code Word 0 Register
[0x0044]	*MCR_CWD1*	Code Word 1 Register
[0x0050]	*MCR_ADCCFG0*	ADC Configuration 0 Register
[0x0054]	*MCR_ADCCFG1*	ADC Configuration 1 Register
[0x0058]	*MCR_ADCCFG2*	ADC Configuration 2 Register
[0x0060]	*MCR_LDOCTRL*	LDO Control Register

4.7.1 Miscellaneous Control Register Details

Table 4-14: Error Correction Coding Enable Register

Error Correction Coding Enable			MCR_ECCEN	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6	ram6	DNM	0	**Reserved, Do Not Modify**
5	ram5	DNM	0	**Reserved, Do Not Modify**
4	ram4	DNM	0	**Reserved, Do Not Modify**
3	ram3	DNM	0	**Reserved, Do Not Modify**
2	ram2	DNM	0	**Reserved, Do Not Modify**
1	ram1	DNM	0	**Reserved, Do Not Modify**
0	ram0	DNM	0	**Reserved, Do Not Modify**

Table 4-15: IPO Manual Register

IPO Manual Trim			MCR_IPO_MTRIM	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8	trim_range	R	0	**Trim Range Select**
7:0	mtrim	R	4	**Manual Trim Value**

Table 4-16: Output Enable Register

Output Enable			MCR_OUTEN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	pdown_out_en	R/W	0	**Power Down Output Enable on P4.0** Set this field to 1 to enable the power down output, P4.0 AF1 (PDOWN). PDOWN is active in *BACKUP* and *STANDBY*. 0: PDOWN output not enabled. 1: PDOWN output is enabled.
0	sqwout_en	R/W	0	**Square Wave Output Enable on P4.1 (SQWOUT)** Set this field to 1 to enable the square wave output on P4.1 AF1 (SQWOUT). 0: Square wave output not enabled. 1: Square wave output enabled.

Table 4-17: Comparator 0 Control Register

Comparator 0 Control			MCR_CMP_CTRL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	int_fl	R/W1C	0	**Comparator 0 Interrupt Flag** This field is set to 1 by hardware when the comparator output changes to the active state as set using the *MCR_CMP_CTRL.pol* field. Write 1 to clear this flag. 0: No interrupt. 1: Interrupt occurred.
14	out	RO	*	**Comparator 0 Output** This field is the comparator output state. 0: Output low. 1: Output high.
13:7	-	RO	0	**Reserved**
6	int_en	R/W	0	**Comparator 0 Interrupt Enable** Set this field to 1 to enable the interrupt for comparator 0. 0: Interrupt disabled. 1: Interrupt enabled.

Comparator 0 Control			MCR_CMP_CTRL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
5	pol	R/W	0	**Comparator 0 Interrupt Polarity Select** Set this field to select the polarity of the output change that generates a comparator 0 interrupt. 0: Interrupt occurs from a transition from low to high. 1: Interrupt occurs from a transition from high to low.
4:1	-	RO	0	**Reserved**
0	en	R/W	0	**Comparator 0 Enable** Set this field to 1 to enable the comparator. 0: Comparator disabled. 1: Comparator enable.

Table 4-18: Miscellaneous Control Register

Miscellaneous Control			MCR_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved**
10	rstnvddiohsel	R/W	0	**RSTN Rail Pull-Up Select** This field selects the I/O rail for the RSTN pull-up. 0: V_{DDIO}. 1: V_{DDIOH}.
9	rstnp1m	R/W	0	**RSTN Pull-Up Select** This field selects the pad pull-up resistor for the RSTN pin. 0: 25kΩ. 1: 1MΩ.
8:5	-	DNM	1	**Reserved, Do Not Modify**
4	ibro_en	R/W	1	**IBRO Enable for *UPM*** Set this field to 1 to enable IBRO in *UPM*. 0: IBRO off in *UPM.* 1: IBRO on in *UPM.*
3	ertco_en	R/W	0	**ERTCO Enable Low-Power Modes** Set this field to 1 to enable the ERTCO in *UPM*, *STANDBY,* and *BACKUP*. 0: ERTCO disabled in *UPM*, *STANDBY,* and *BACKUP.* 1: ERTCO enabled in *UPM*, *STANDBY,* and *BACKUP.*
2	inro_en	R/W	0	**INRO Enable for RTC** Set this field to 1 to enable the INRO as the alternate clock source for the RTC. 0: RTC uses ERTCO. 1: RTC uses INRO.
1:0	cmphyst	RO	0	**Reserved**

Table 4-19: GPIO4 Pin Control Register

GPIO4 Pin Control			MCR_GPIO4_CTRL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	p41_in	RO	See Description	**GPIO4 Pin 1 Input Status** Read this field to determine the input status of P4.1. 0: Input low. 1: Input high.
6	p41_pe	R/W	0	**GPIO4 Pin 1 Pull Enable** Set this bit to 0 to enable the pull resistor for P4.1. Use If *MCR_GPIO4_CTRL.p41_do* to set the pull direction. 0: Enabled. 1: Disabled.

GPIO4 Pin Control			MCR_GPIO4_CTRL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
5	p41_oe	R/W	0	**GPIO4 Pin 1 Output Enable** Set this bit to 1 to enable P4.1 for output mode. 0: Input mode. 1: Output mode.
4	p41_do	R/W	0	**GPIO4 Pin 1 Data Output/Pull Direction** If *MCR_GPIO4_CTRL.p41_oe* is set to 1, this field is used to control the output state of P4.1. 0: Output low if *MCR_GPIO4_CTRL.p41_oe* is 1. 1: Output high if *MCR_GPIO4_CTRL.p41_oe* is 1. If *MCR_GPIO4_CTRL.p41_pe* is set to 0, this field is used to control the resistor pull direction. 0: Pull-down resistor if *MCR_GPIO4_CTRL.p41_pe* is 0. 1: Pull-up resistor if *MCR_GPIO4_CTRL.p41_pe* is 0.
3	p40_in	RO	See Description	**GPIO4 Pin 0 Input Status** Read this field to determine the input status of P4.0. 0: Input low. 1: Input high.
2	p40_pe	R/W	0	**GPIO4 Pin 0 Pull Enable** Set this bit to 0 to enable the pull resistor for P4.0. Use If *MCR_GPIO4_CTRL.p40_do* to set the pull direction. 0: Enabled. 1: Disabled.
1	p40_oe	R/W	0	**GPIO4 Pin 0 Output Enable** Set this bit to 1 to enable P4.0 for output mode. 0: Input mode. 1: Output mode enabled.
0	p40_do	R/W	0	**GPIO4 Pin 0 Data Output/Pull Direction** If *MCR_GPIO4_CTRL.p40_oe* is set to 1, this field is used to control the output state of P4.0. 0: Output low if *MCR_GPIO4_CTRL.p40_oe* is 1. 1: Output high if *MCR_GPIO4_CTRL.p40_oe* is 1. If *MCR_GPIO4_CTRL.p40_pe* is set to 0, this field is used to control the resistor pull direction. 0: Pull-down resistor if *MCR_GPIO4_CTRL.p40_pe* is 0. 1: Pull-up resistor if *MCR_GPIO4_CTRL.p40_pe* is 0.

Table 4-20: Code Word 0 Register

Code Word 0			MCR_CWD0	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Data** This register maintains the contents written to it as long as V_{REG1} supply is valid.

Table 4-21: Code Word 1 Register

Code Word 1			MCR_CWD1	[0x0044]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Data** This register maintains the contents written to it as long as V_{REG1} supply is valid.

Table 4-22: ADC Configuration Register 0

ADC Configuration 0			MCR_ADCCFG0	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	ref_sel	R/W	0	**ADC Reference Select** This field selects either the 1.25V or 2.048V internal reference when the internal reference is selected (*MCR_ADCCFG0.ext_ref* = 0). 0: 1.25V. 1: 2.048V.
2	ext_ref	R/W	0	**ADC External Reference Select** This field selects between the internal and external references. 0: Internal reference. 1: External reference.
1:0	-	RO	0	**Reserved**

Table 4-23: ADC Configuration Register 1

ADC Configuration 1			MCR_ADCCFG1	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 4-24: ADC Configuration Register 2

ADC Configuration 2			MCR_ADCCFG2	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 4-25: LDO Control Register

LDO Control			MCR_LDOCTRL	[0x0060]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	2p5en	RO	0	**LDO 2.5V Enable**
0	0p9en	R/W	0	**LDO 0.9V Enable** This field enables the USB 0.9V LDO for USB. This field must be set to 1 to use USB.

4.8 Low-Power General Control Registers (LPGCR)

This set of general control registers provides reset and clock control for the low-power peripherals, including:

- LPUART0 (UART3)
- LPTMR0 (TMR4)
- LPTMR1 (TMR5)
- LPWDT0 (WDT1)
- LPCOMP1, LPCOMP2, and LPCOMP3
- GPIO3

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 4-26 Low-Power Control Register Summary

Offset	Register	Name
[0x0008]	*LPGCR_RST*	*Reset Control Register*
[0x000C]	*LPGCR_PCLKDIS*	*Clock Control Register*

4.8.1 Low-Power General Control Registers Details

Table 4-27: Reset Control Register

Error Correction Coding Enable			LPGCR_RST	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6	lpcomp	W1O	0	**Low-Power Comparator Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.
5	-	RO	0	**Reserved**
4	uart3	W1O	0	**UART3 (LPUART0) Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.
3	tmr5	W1O	0	**TMR5 (LPTMR1) Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.
2	tmr4	W1O	0	**TMR4 (LPTMR0) Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.
1	wdt1	W1O	0	**WDT1 (LPWDT0) Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.
0	gpio3	W1O	0	**GPIO3 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Device Resets* for additional information.

Table 4-28: Clock Disable Register

Clock Disable Register			LPGCR_PCLKDIS	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	1	**Reserved**
6	lpcomp	R/W	1	**Low-Power Comparator 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: Enabled. 1: Disabled.
5	-	RO	1	**Reserved**
4	uart3	R/W	1	**UART3 (LPUART0) 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: Enabled. 1: Disabled.
3	tmr5	R/W	1	**TMR5 (LPTMR1) 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: Enabled. 1: Disabled.
2	tmr4	R/W	1	**TMR4 (LPTMR0) 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: Enabled. 1: Disabled.

Clock Disable Register			LPGCR_PCLKDIS	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
1	wdt1	R/W	1	**WDT1 (LPWDT0) 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: Enabled. 1: Disabled.
0	gpio3	R/W	1	**GPIO3 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: Enabled. 1: Disabled.

4.9 Power Sequencer Registers (PWRSEQ)

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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.

Note: The PWRSEQ registers are reset only on a POR. System reset, soft reset, and peripheral reset do not affect the PWRSEQ register values.

Table 4-29: Power Sequencer Register Summary

Offset	Register	Name
[0x0000]	*PWRSEQ_LPCN*	*Low-Power Control Register*
[0x0004]	*PWRSEQ_LPWKST0*	*Low-Power GPIO0 Wake-up Status Flags*
[0x0008]	*PWRSEQ_LPWKEN0*	*Low-Power GPIO0 Wake-up Enable Register*
[0x000C]	*PWRSEQ_LPWKST1*	*Low-Power GPIO1 Wake-up Status Flags*
[0x0010]	*PWRSEQ_LPWKEN1*	*Low-Power GPIO1 Wake-up Enable Register*
[0x0014]	*PWRSEQ_LPWKST2*	*Low-Power GPIO2 Wake-up Status Flags*
[0x0018]	*PWRSEQ_LPWKEN2*	*Low-Power GPIO2 Wake-up Enable Register*
[0x001C]	*PWRSEQ_LPWKST3*	*Low-Power GPIO3 Wake-up Status Flags*
[0x0020]	*PWRSEQ_LPWKEN3*	*Low-Power GPIO3 Wake-up Enable Register*
[0x0024]	*PWRSEQ_LPWKST4*	*Low-Power GPIO4 Wake-up Status Flags*
[0x0028]	*PWRSEQ_LPWKEN4*	*Low-Power GPIO4 Wake-up Enable Register*
[0x0030]	*PWRSEQ_LPPWST*	*Low-Power Peripheral Wake-up Status Register*
[0x0034]	*PWRSEQ_LPPWEN*	*Low-Power Peripheral Wake-up Enable Register*
[0x0048]	*PWRSEQ_GP0*	*General Purpose Register 0*
[0x004C]	*PWRSEQ_GP1*	*General Purpose Register 1*

4.9.1 Power Sequencer Register Details

Table 4-30: Low-Power Control Register

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31	wkrst	R/W1O	0	**Low-Power Wake-up Status Register Clear** Write 1 to this field to clear the low-power wake-up status registers: • *PWRSEQ_LPWKST0* • *PWRSEQ_LPWKST1* • *PWRSEQ_LPWKST2* • *PWRSEQ_LPWKST3* • *PWRSEQ_LPPWST* 1: Write 1 to initiate a clear of all the low-power wake-up status registers. The hardware automatically clears this field when the registers are cleared.

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
30:12	-	DNM	0	**Reserved, Do Not Modify.** *Note: This field must always be set to 0 to maintain future compatibility.*
11	bgoff	R/W	1	**Band Gap Disable for *LPM* and *BACKUP*** Setting this field to 1 (default) disables the bandgap during *LPM* and *BACKUP* mode. 0: System bandgap is on in *LPM* and *BACKUP* modes 1: System bandgap is off in *LPM* and *BACKUP* modes
10	-	R/W	0	**Reserved**
9	fast_entry_dis	R/W	0	**Low-Power Mode Entry Clock Select** If the ISO is selected (default), fast *LPM* entry is enabled. Setting the clock to INRO disables fast *LPM* entry. 0: ISO used for entering *LPM* (fast mode enabled). 1: INRO used for *LPM* entry (fast mode disabled).
8	isoclk_select	R/W	1	**Low-Power Mode APB Clock Select** This field selects the clock source for the RV32 (CPU1) and other APB peripherals during *LPM*. 0: PCLK is used as the RV32 (CPU1) and APB system clock during *LPM*. 1: ISO is used as the RV32 (CPU1) and APB system clock during *LPM*.
7	ramret8	R/W	0	**System RAM 8 Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram8*. See *Table 4-4 BACKUP* retention. 0: Disabled. 1: Enabled.
6	ramret6	R/W	0	**System RAM 6 Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram6*. See *Table 4-4 BACKUP* retention. 0: Disabled. 1: Enabled.
5	ramret5	R/W	0	**System RAM 5 Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram5*. See *Table 4-4 BACKUP* retention. 0: Disabled. 1: Enabled.
4	ramret4	R/W	0	**System RAM 4 Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram4*. See *Table 4-4 BACKUP* retention. 0: Disabled. 1: Enabled.
3	ramret3	R/W	0	**System RAM 3 Data Retention Enable for *BACKUP* Mode** Set this field to 1 to enable data retention for *sysram3*. See *Table 4-4 BACKUP* retention. 0: Disabled. 1: Enabled.
2	ramret2	R/W	0	**System RAM 2 Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram2*. See *Table 4-4 BACKUP* retention. 0: Disabled. 1: Enabled.

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
1	ramret1	R/W	0	**System RAM 1 Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram1*. See *Table 4-4 BACKUP* retention. 0: Disabled. 1: Enabled.
0	ramret0	R/W	0	**System RAM 0 Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram0*. See *Table 4-4 BACKUP* retention. 0: Disabled. 1: Enabled.

Table 4-31: GPIO0 Low-Power Wake-up Status Flags

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

Table 4-32: GPIO0 Low-Power Wake-up Enable Registers

GPIO0 Low-Power Wake-up Enable			PWRSEQ_LPWKEN0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	wakeen	R/W	0	**GPIO0 Pin Wake-up Interrupt Enable** Setting a GPIO0 pin's bit in this register causes an interrupt that wakes up the device from any low-power mode to *ACTIVE*. A wake-up event sets the corresponding GPIO0 bit in the *PWRSEQ_LPWKST0* register, enabling the determination of which GPIO0 pin triggered the wake-up event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the MAX32690 to wake up from a low-power mode on a GPIO pin transition, first set the GPIO wake-up enable register bit to 1 (GCR_PM.gpio_we = 1).*

Table 4-33: GPIO1 Low-Power Wake-up Status Flags

GPIO1 Low-Power Wake-up Status Flags			PWRSEQ_LPWKST1	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	wakest	R/W1C	0	**GPIO1 Pin Wake-up Status Flag** Whenever a GPIO1 pin, in any power mode, transitions from low-to-high or high-to-low, the pin's corresponding bit in this register is set. The device transitions from a low-power mode to *ACTIVE* if the corresponding interrupt enable bit is set in *PWRSEQ_LPWKEN1*. *Note: Clear this register before entering any low-power mode.* *Note: Bits corresponding to unimplemented GPIO are ignored.*

Table 4-34: GPIO1 Low-Power Wake-up Enable Registers

GPIO1 Low-Power Wake-up Enable			PWRSEQ_LPWKEN1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	wakeen	R/W	0	**GPIO1 Pin Wake-up Interrupt Enable** Setting a GPIO1 pin's bit in this register causes an interrupt that wakes up the device from any low-power mode to *ACTIVE*. A wake-up event sets the corresponding GPIO1 bit in the *PWRSEQ_LPWKST1* register, enabling the determination of which GPIO1 pin triggered the wake-up event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the MAX32690 to wake up from a low-power mode on a GPIO pin transition, first set the GPIO wake-up enable register bit to 1 (GCR_PM.gpio_we = 1).* *Note: Bits corresponding to unimplemented GPIO are ignored.*

Table 4-35: GPIO2 Low-Power Wake-up Status Flags

GPIO2 Low-Power Wake-up Status Flags			PWRSEQ_LPWKST2	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	wakest	R/W1C	0	**GPIO2 Pin Wake-up Status Flag** Whenever a GPIO2 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 transitions from a low-power mode to ACTIVE if the corresponding interrupt enable bit is set in PWRSEQ_LPWKEN2. This register should be cleared before entering any low-power mode.* *Note: Bits corresponding to unimplemented GPIO are ignored.*

Table 4-36: GPIO2 Low-Power Wake-up Enable Registers

GPIO2 Low-Power Wake-up Enable			PWRSEQ_LPWKEN2	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	en	R/W	0	**GPIO2 Pin Wake-up Interrupt Enable** Setting a GPIO2 pin's bit in this register causes an interrupt that wakes up the device from any low-power mode to *ACTIVE*. A wake-up event sets the corresponding GPIO2 bit in the *PWRSEQ_LPWKST2* register, enabling the determination of which GPIO2 pin triggered the wake-up event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the MAX32690 to wake up from a low-power mode on a GPIO pin transition, first set the GPIO wake-up enable register bit to 1 (GCR_PM.gpio_we = 1).* *Note: Bits corresponding to unimplemented GPIO are ignored.*

Table 4-37: GPIO3 Low-Power Wake-up Status Flags

GPIO3 Low-Power Wake-up Status Flags			PWRSEQ_LPWKST3	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	wakest	R/W1C	0	**GPIO3 Pin Wake-up Status Flag** Whenever a GPIO3 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 transitions from a low-power mode to ACTIVE if the corresponding interrupt enable bit is set in PWRSEQ_LPWKEN3. This register should be cleared before entering any low-power mode.* *Note: Bits corresponding to unimplemented GPIO are ignored.*

Table 4-38: GPIO3 Low-Power Wake-up Enable Registers

GPIO3 Low-Power Wake-up Enable			PWRSEQ_LPWKEN3	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	wakeen	R/W	0	**GPIO3 Pin Wake-up Interrupt Enable** Setting a GPIO3 pin's bit in this register causes an interrupt that wakes up the device from any low-power mode to *ACTIVE*. A wake-up event sets the corresponding GPIO3 bit in the *PWRSEQ_LPWKST3* register, enabling the determination of which GPIO3 pin triggered the wake-up event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the MAX32690 to wake up from a low-power mode on a GPIO pin transition, first set the GPIO wake-up enable register bit GCR_PM.gpio_we = 1.* *Note: Bits corresponding to unimplemented GPIO are ignored.*

Table 4-39: GPIO4 Low-Power Wake-up Status Flags

GPIO4 Low-Power Wake-up Status Flags			PWRSEQ_LPWKST4	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	wakest	R/W1C	0	**GPIO4 Pin Wake-up Status Flag** Whenever a GPIO4 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 transitions from a low-power mode to ACTIVE if the corresponding interrupt enable bit is set in PWRSEQ_LPWKEN3. This register should be cleared before entering any low-power mode.* *Note: Bits corresponding to unimplemented GPIO are ignored.*

Table 4-40: GPIO4 Low-Power Wake-up Enable Registers

GPIO4 Low-Power Wake-up Enable			PWRSEQ_LPWKEN4	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	wakeen	R/W	0	**GPIO4 Pin Wake-up Interrupt Enable** Setting a GPIO4 pin's bit in this register causes an interrupt that wakes up the device from any low-power mode to *ACTIVE*. A wake-up event sets the corresponding GPIO3 bit in the *PWRSEQ_LPWKST3* register, enabling the determination of which GPIO3 pin triggered the wake-up event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the MAX32690 to wake up from a low-power mode on a GPIO pin transition, first set the "GPIO Wake-up Enable" register bit GCR_PM.gpio_we = 1.* *Note: Bits corresponding to unimplemented GPIO are ignored.*

Table 4-41: Low-Power Peripheral Wake-up Status Flags

Low-Power Peripheral Wake-up Status Flags			PWRSEQ_LPPWST	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:18		RO	0	**Reserved**
17	reset	R/W1C	0	**Reset Detected Wake-up Flag** This field is set when an external reset causes the wake-up event.
16	backup	R/W1C	0	***BACKUP* Wake-up Flag** This field is set when the device wakes up from *BACKUP*.
15:5	-	RO	0	**Reserved**
4	aincomp0	R/W1C	0	**Comparator 0 Wake-up Flag** This field is set if the wake-up event was the result of an input comparator trigger event.
3	-	RO	0	**Reserved**

Low-Power Peripheral Wake-up Status Flags			PWRSEQ_LPPWST	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
2	usbvbus	R/W1C	0	**USB V_{BUS} Detect Wake-up Flag** This field is set if the wake-up event was the result of a USB power supply state change.
1	usbls_dm	R/W1C	0	**USB D- Linestate Wake-up Flag** This field is set if the wake-up event was the result of a linestate change of the USB D- pin.
0	usbls_dp	R/W1C	0	**USB D+ Linestate Wake-up Flag** This field is set if the wake-up event was the result of a linestate change of the USB D+ pin.

Table 4-42: Low-Power Peripheral Wake-up Enable Registers

Low-Power Peripheral Wake-up Enable			PWRSEQ_LPPWEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:27	can1	R/W	0	**CAN1 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the CAN1 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
30	can0	R/W	0	**CAN0 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the CAN0 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
29	spi2	R/W	0	**SPI2 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the SPI2 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
28	spi1	R/W	0	**SPI1 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the SPI1 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
27	btle	R/W	0	**Bluetooth LE Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the Bluetooth LE interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
26	lpcmp	R/W	0	**LPCOMP Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the LPCOMP interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
25	spi0	R/W	0	**SPI0 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the SPI0 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
24	i2s	R/W	0	**I^2S Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the I^2S interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
23	i2c2	R/W	0	**I2C2 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the I2C2 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.

Low-Power Peripheral Wake-up Enable			PWRSEQ_LPPWEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
22	i2c1	R/W	0	**I2C1 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the I2C1 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
21	i2c0	R/W	0	**I2C0 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the I2C0 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
20	uart3	R/W	0	**LPUART0 (UART3) Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from LPUART0 (UART3) interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
19	uart2	R/W	0	**UART2 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the UART2 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
18	uart1	R/W	0	**UART1 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the UART1 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
17	uart0	R/W	0	**UART0 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the UART0 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
16	tmr5	R/W	0	**LPTMR1 (TMR5) Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the LPTMR1 (TMR5) interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
15	tmr4	R/W	0	**LPTMR0 (TMR4) Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the LPTMR0 (TMR4) interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
14	tmr3	R/W	0	**TMR3 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the TMR3 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
13	tmr2	R/W	0	**TMR2 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the TMR2 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
12	tmr1	R/W	0	**TMR1 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the TMR1 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
11	tmr0	R/W	0	**TMR0 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the TMR0 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.

Low-Power Peripheral Wake-up Enable			PWRSEQ_LPPWEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
10	cpu1	R/W	0	**RV32 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the RV32 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
9	wdt1	R/W	0	**WDT1 (LPWDT0) Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the WDT1 (LPWDT0) interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
8	wdt0	R/W	0	**WDT0 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from the WDT0 interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
7:5	-	RO	0	**Reserved**
4	aincomp0	R/W	0	**Comparator 0 Interrupt Wake-Up Enable** Set this field to 1 to enable wake-up events from Comparator 0's interrupt. 0: Disable wake-up on interrupt. 1: Enable wake-up on interrupt.
3	-	RO	0	**Reserved**
2	usbvbus	R/W	0	**USB VBUS Detect Wake-Up Enable** Set this field to 1 to enable wake-up from the USB power supply state change (on or off). 0: Disable wake-up 1: Enable wake-up
1:0	usbls	R/W	0	**USB Linestate Detect Wake-Up Enable** This field allows wake-up from the corresponding USB linestate signal(s) on transitions(s) from low to high or high to low when *GCR_PM.usb_we* is set to 1. usbls[1]: D+ usbls[0]: D-

Table 4-43: Low-Power General Purpose Register 0

Low-Power General Purpose Register 0			PWRSEQ_GP0	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**General Purpose Field** This register can be used as a general-purpose register by software and retains the contents during *SLEEP*, *LPM*, *UPM*, *STANDBY*, and *BACKUP* modes.

Table 4-44: Low-Power General Purpose Register 1

Low-Power General Purpose Register 1			PWRSEQ_GP1	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**General Purpose Field** This register can be used as a general-purpose register by software and retains the contents during *SLEEP*, *LPM*, *UPM*, *STANDBY*, and *BACKUP* modes.

4.10 Trim System Initialization Registers (TRIMSIR)

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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.

Note: The TRIMSIR registers are reset only on a POR. System reset, soft reset, and peripheral reset do not affect the TRIMSIR register values.

Table 4-45: Trim System Initialization Register Summary

Offset	Register Name	Description
[0x0008]	*TRIMSIR_RTC*	*RTC Trim System Initialization Register*

4.10.1 Trim System Initialization Register Details

Table 4-46: RTC Trim System Initialization Register

RTC Trim System Initialization			TRIMSIR_RTC	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31	lock	RO	0	**Lock** If this field is set to 1, this register is read-only, and the *TRIMSIR_RTC.rtcx1* and *TRIMSIR_RTC.rtcx2* fields cannot be modified.
30:26	-	RO	0	**Reserved**
25:21	rtcx1	R/W*	0	**RTC X2 Trim** The X2 trim setting for the RTC. *Note: If TRIMSIR_RTC.lock is set to 1, this field is read-only.*
20:16	rtcx2	R/W*	0	**RTC X1 Trim** The X1 trim setting for the RTC. *Note: If TRIMSIR_RTC.lock is set to 1, this field is read-only.*
15:0	-	RO	0	**Reserved**

4.11 Global Control Registers (GCR)

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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.

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 4-47: Global Control Register Summary

Offset	Register	Description
[0x0000]	*GCR_SYSCTRL*	*System Control Register*
[0x0004]	*GCR_RST0*	*Reset Register 0*
[0x0008]	*GCR_CLKCTRL*	*Clock Control Register*
[0x000C]	*GCR_PM*	*Power Management Register*
[0x0018]	*GCR_PCLKDIV*	*Peripheral Clocks Divisor*
[0x0024]	*GCR_PCLKDIS0*	*Peripheral Clocks Disable 0*
[0x0028]	*GCR_MEMCTRL*	*Memory Clock Control*
[0x002C]	*GCR_MEMZ*	*Memory Zeroize Register*
[0x0040]	*GCR_SYSST*	*System Status Flags*
[0x0044]	*GCR_RST1*	*Reset Register 1*
[0x0048]	*GCR_PCLKDIS1*	*Peripheral Clocks Disable 1*
[0x004C]	*GCR_EVENTEN*	*Event Enable Register*
[0x0050]	*GCR_REVISION*	*Revision Register*
[0x0054]	*GCR_SYSIE*	*System Status Interrupt Enable*
[0x0064]	*GCR_ECCERR*	*Error Correction Coding Error Register*
[0x0068]	*GCR_ECCCED*	*Error Correction Coding Correctable Error Detected*
[0x006C]	*GCR_ECCIE*	*Error Correction Coding Interrupt Enable Register*
[0x0070]	*GCR_ECCADDR*	*Error Correction Coding Error Address Register*
[0x0074]	*GCR_BTLELDOCTRL*	*BTLE LDO Control Register*
[0x0078]	*GCR_BTLELDODLY*	*BTLE LDO Delay Count Register*
[0x0080]	*GCR_GPR0*	*General Purpose Register 0*

4.11.1 Global Control Register Details (GCR)

Table 4-48: System Control Register

System Control			GCR_SYSCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:18	-	RO	0	**Reserved**
17:16	ovr	DNM	0b10	**Reserved, Do Not Modify** This field must be set to 0b10 for proper device operation.
15	chkres	R	0	**ROM Checksum Calculation Pass/Fail** This field is the result after setting the bit *GCR_SYSCTRL.cchk*. This field is only valid after the ROM checksum is complete and *GCR_SYSCTRL.cchk* is cleared by the hardware. 0: Pass. 1: Fail.
14	swd_dis	R/W	0	**Serial Wire Debug Disable** This bit is used to disable the serial wire debug interface. 0: SWD disabled. 1: SWD enabled. *Note: This bit is only writeable if the flash is not factory locked or the GCR_SYSST.icelock bit is 0.*
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_SYSCTRL.chkres*. Writing a 0 has no effect. 0: No operation. 1: Start ROM checksum calculation.
12	romdone	RO	0	**Reserved, Do Not Modify**
11:10	-	RO	0	**Reserved**
9	syscache_dis	R/W	1	**SPIXF (EMCC) Cache Disable** Set this field to 1 to disable the EMCC. 0: EMCC enabled. 1: EMCC disabled.
8	-	RO	0	**Reserved**
7	srcc_flush	R/W	0	**SPIXR (SRCC) Cache Disable** Set this field to 1 to flush the SRCC.
6	icc0_flush	R/W	0	**ICC0 Cache Flush** Write 1 to flush the code cache and the instruction buffer for the CM4 (CPU0). This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect and does not stop a cache flush in progress. 0: ICC0 flush complete. 1: Flush the contents of the ICC0 cache.
5	-	RO	0	**Reserved**
4	flash_page_flip	R/*	0	**Flash Page Flip Flag** This field flips the bottom and top halves of flash memory. This bit is controlled by hardware and should not be modified by software. *Note: Any change to this field flushes all caches.* 0: Physical layout matches logical layout. 1: Top and bottom halves flipped.
3:1	-	RO	0b001	**Reserved**
0	bstapen	DNM	*	**Reserved, Do Not Modify**

Table 4-49: Reset Register 0

Reset 0			GCR_RST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31	sys	R/W	0	**System Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *System Reset* for additional information.
30	periph	R/W	0	**Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *Note: Watchdog timers, GPIO ports, the AoD, RAM retention, and the GCR are unaffected. See Peripheral Reset for additional information.*
29	soft	R/W	0	**Soft Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Soft Reset* for additional information.
28	uart2	R/W	0	**UART2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
27	-	R/W	0	**Reserved**
26	adc	R/W	0	**ADC Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
25	-	RO	0	**Reserved**
24	trng	R/W	0	**TRNG Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
23	usb	R/W	0	**USB Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
22	smphr	R/W	0	**Semaphore Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
21	hpb	R/W	0	**HyperBus Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
20	can1	R/W	0	**CAN1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
19	can0	R/W	0	**CAN0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
18	crypto	R/W	0	**Crypto Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
17	rtc	R/W	0	**RTC Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
16	i2c0	R/W	0	**I2C0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
15	spi2	R/W	0	**SPI2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
14	spi1	R/W	0	**SPI1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
13	spi0	R/W	0	**SPI0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
12	uart1	R/W	0	**UART1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
11	uart0	R/W	0	**UART0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
10:9	-	RO	0	**Reserved**

Reset 0			GCR_RST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
8	tmr3	R/W	0	**TMR3 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
7	tmr2	R/W	0	**TMR2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
6	tmr1	R/W	0	**TMR1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
5	tmr0	R/W	0	**TMR0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
4	gpio2	R/W	0	**GPIO2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
3	gpio1	R/W	0	**GPIO1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
2	gpio0	R/W	0	**GPIO0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
1	wdt0	R/W	0	**Watchdog Timer 0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
0	dma	R/W	0	**DMA Access Block Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.

Table 4-50: Clock Control Register

Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	RO	0b10	**Reserved**
29	inro_rdy	RO	0	**8kHz Internal Nano-Ring Oscillator (INRO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
28	ibro_rdy	RO	0	**7.3728MHz Internal Baud Rate Oscillator (IBRO) Ready Status** 0: Not ready. 1: Oscillator ready.
27	ipo_rdy	RO	0	**120MHz Internal Primary Oscillator (IPO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
26	iso_rdy	RO	0	**60MHz Internal Secondary Oscillator (ISO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
25	ertco_rdy	RO	0	**32.768kHz External RTC Oscillator (ERTCO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
24	erfo_rdy	RO	0	**32MHz External RF Oscillator (ERFO) Ready** 0: Not ready or not enabled. 1: Oscillator ready.
23:22	-	RO	0	**Reserved**
21	ibro_vs	R/W	0	**IBRO Voltage Select** Set this field to 0 to use V_{CORE} as the IBRO. Setting this field to 1 uses a dedicated 1V regulated supply for the IBRO. 0: V_{CORE} selected as the IBRO supply. 1: IBRO supplied using a 1V regulated supply.
20	ibro_en	RO	1	**7.3728MHz IBRO Enable** The IBRO is always enabled. 1: Enabled and ready when *GCR_CLKCTRL.ibro_rdy* = 1.

Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
19	ipo_en	R/W	0	**120MHz IPO Enable** 0: Disabled. 1: Enabled and ready when *GCR_CLKCTRL.ipo_rdy* = 1.
18	iso_en	R/W	1	**60MHz ISO Enable** 0: Disabled. 1: Enabled and ready when *GCR_CLKCTRL.iso_rdy* = 1.
17	ertco_en	R/W	0	**32.768kHz ERTCO Enable** 0: Disabled. 1: Enabled and ready when *GCR_CLKCTRL.ertco_rdy* = 1.
16	erfo_en	R/W	0	**32MHz ERFO Enable** 0: Disabled. 1: Enabled and ready when *GCR_CLKCTRL.erfo_rdy* = 1.
15:14	-	RO	0	**Reserved**
13	sysclk_rdy	RO	0	**SYS_OSC Select Ready** When SYS_OSC is changed by modifying the *GCR_CLKCTRL.sysclk_sel* field, there is a delay until the switchover is complete. This field is zero until the switchover is complete. 0: Switch to new clock source not yet complete. 1: SYS_OSC is the clock source selected in *GCR_CLKCTRL.sysclk_sel*.
12	-	RO	0	**Reserved**
11:9	sysclk_sel	R/W	0	**System Oscillator Source Select** Selects the SYS_OSC source used to generate the system clock (SYS_CLK). Modifying this field clears *GCR_CLKCTRL.sysclk_rdy* immediately. 0: ISO (Reset default). 1: Reserved. 2: ERFO. 3: INRO. 4: IPO. 5: IBRO. 6: ERTCO. 7: External Clock, EXT_CLK, P0.3, AF1.
8:6	sysclk_div	R/W	0	**System Oscillator Prescaler** Sets the divider for generating SYS_CLK from the selected SYS_OSC as shown in the following equation: $SYS\_CLK = \frac{SYS\_CLK}{2^{sysclk\_div}}$ *Note: Valid values are from 0 to 7 for sysclk_div.*
5:0	-	RO	4	**Reserved**

Table 4-51: Power Management Register

Power Management			GCR_PM	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0	**Reserved**
20	erfo_bp	R/W	0	**ERFO Bypass** This bit is set to 0 on a POR is not affected by other forms of reset. Set this field to 1 to use a square wave input as the clock source for the ERFO driving the HFXIN pin. 0: The clock source is the crystal oscillator, driving the crystal connected between HFXIN and HFXOUT pins. 1: The clock source is a square wave driven into the HFXIN pin.
19:18	-	RO	0	**Reserved**

Power Management			GCR_PM	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
17	ibro_pd	R/W	1	**IBRO Power Down** Set this field to 1 to power down the IBRO during *LPM* and *UPM*. 0: IBRO enabled during *LPM* and *UPM.* 1: IBRO powered down during *LPM* and *UPM.*
16	ipo_pd	R/W	1	**IPO Power Down** Set this field to 1 to power down the IPO during *LPM*. 0: IPO enabled during *LPM.* 1: IPO powered down during *LPM.*
15	iso_pd	R/W	1	**ISO Power Down** This bit selects the power state in *LPM* for the ISO oscillator. 0: IPO enabled during *LPM.* 1: IPO powered down in *LPM.*
14:10	-	DNM	0b11100	**Reserved, Do Not Modify**
9	aincomp_we	R/W	0	**Analog Input Comparator Wake-up Enable** This bit enables the analog input comparator interrupt to wake the device from *SLEEP*, *LPM, UPM*, *STANDBY,* or *BACKUP*.
8	-	RO	0	**Reserved**
7	wut_we	R/W	0	**Wake-Up Timer Enable** Set this field to 1 to enable the wake-up timer as a wake-up source. The wake-up timer wakes the device from *SLEEP*, *LPM*, *UPM, STANDBY,* or *BACKUP*. 0: Disabled. 1: Enabled.
6	usb_we	R/W	0	**USB Wake-Up Enable** Set this field to 1 to enable the wake-up timer as a wake-up source. The USB wakes the device from *SLEEP*, *LPM*, or *BACKUP*. 0: Disabled. 1: Enabled.
5	rtc_we	R/W	0	**RTC Alarm Wake-Up Enable** Set this field to 1 to enable an RTC alarm to wake the device. The RTC alarm wakes the device from *SLEEP*, *LPM*, *UPM, STANDBY,* or *BACKUP*. 0: Disabled. 1: Enabled.
4	gpio_we	R/W	0	**GPIO Wake-Up Enable** Set this field to 1 to enable all GPIO pins as potential wake-up sources. Any GPIO configured for wake-up wakes the device from *SLEEP*, *LPM*, *UPM, STANDBY,* or *BACKUP*. 0: Disabled. 1: Enabled.
3:0	mode	R/W	0	**Operating Mode** This field controls the operating state of the device. 0b0000: *ACTIVE.* 0b0001: *SLEEP.* 0b0010: *STANDBY.* 0b0100: *BACKUP.* 0b1000: *LPM* (CM4 *sleep deep*). 0b1001: *UPM.* 0b1010: Reserved.

Table 4-52: Peripheral Clock Divisor Register

Peripheral Clocks Divisor			GCR_PCLKDIV	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 4-53: Peripheral Clock Disable Register 0

Peripheral Clocks Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31	spixipc	R/W	1	**SPI XiP Master Controller 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: Enabled. 1: Disabled.
30	spixip	R/W	1	**SPI XiP 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: Enabled. 1: Disabled.
29	pt	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: Enabled. 1: Disabled.
28	i2c1	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: Enabled. 1: Disabled.
27:24	-	RO	1	**Reserved**
23	adc	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: Enabled. 1: Disabled.
22:19	-	RO	1	**Reserved**
18	tmr3	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: Enabled. 1: Disabled.
17	tmr2	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: Enabled. 1: Disabled.
16	tmr1	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: Enabled. 1: Disabled.

Peripheral Clocks Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
15	tmr0	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: Enabled. 1: Disabled.
14	crypto	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: Enabled. 1: Disabled.
13	i2c0	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: Enabled. 1: Disabled.
12:11	-	RO	1	**Reserved**
10	uart1	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: Enabled. 1: Disabled.
9	uart0	R/W	1	**UART0 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: Enabled. 1: Disabled.
8	spi2	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: Enabled. 1: Disabled.
7	spi1	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: Enabled. 1: Disabled.
6	spi0	R/W	1	**SPI0 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: Enabled. 1: Disabled.
5	dma	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: Enabled. 1: Disabled.
4	-	RO	1	**Reserved**
3	usb	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: Enabled. 1: Disabled.

Peripheral Clocks Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
2	gpio2	R/W	1	**GPIO2 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: Enabled. 1: Disabled.
1	gpio1	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: Enabled. 1: Disabled.
0	gpio0	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: Enabled 1: Disabled.

Table 4-54: Memory Clock Control Register

Memory Clock Control			GCR_MEMCTRL	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9:8	hypclkds	RO	0	**HyperBus Drive Strength Control** Set this field to control the drive strength of the HyperBus pins. 0b00: 1mA drive. 0b01: 2mA drive. 0b10: 4mA drive. 0b11: 8mA drive.
2:0	fws	R/W	5	**Program Flash Wait States** This field is the number of wait-state cycles per flash code read access. 0: Invalid. 1–7: Number of internal flash memory access wait states. *Note: For the IPO and ISO clocks, the minimum wait state is 2.* *Note: For all other clock sources, the minimum wait state is 1.*

Table 4-55: Memory Zeroize Control Register

Memory Zeroize			GCR_MEMZ	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	-	**Reserved**
15	dcache_tag	R/W1O	0	**System Cache Tag Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
14	dcache_data	R/W1O	0	**System Cache Data Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.

Memory Zeroize			GCR_MEMZ	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
13	maaram	R/W1O	0	**MAA RAM Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
12	usbfifo	R/W1O	0	**USB FIFO Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
11	iccxip	R/W1O	0	**SPI XiP Cache Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
10	icc1	R/W1O	0	**ICC1 Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
9	icc0	R/W1O	0	**ICC0 Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
8	ram8	R/W1O	0	***sysram8* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
7	ram7	R/W1O	0	***sysram7* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
6	ram6	R/W1O	0	***sysram6* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
5	ram5	R/W1O	0	***sysram5* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
4	ram4	R/W1O	0	***sysram4* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.

Memory Zeroize			GCR_MEMZ	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
3	ram3	R/W1O	0	***sysram3* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
2	ram2	R/W1O	0	***sysram2* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
1	ram1	R/W1C	0	***sysram1* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
0	ram0	R/W1C	0	***sysram0* Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.

Table 4-56: System Status Flag Register

System Status Flag			GCR_SYSST	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	codeinterr	R	0	**SPI XiP Code Integrity Error** This field is set to 1 by hardware if a code integrity error occurs in the flash SPI XiP interface. 0: Normal operation. 1: Integrity error.
0	icelock	R	0	**Arm ICE Lock Status Flag** 0: Arm ICE is unlocked (enabled). 1: Arm ICE is locked (disabled).

Table 4-57: Reset Register 1

Reset 1			GCR_RST1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31	cpu1	RO	0	**CPU1 (RV32) Reset** Write 1 to initiate the reset operation. 0: Operation complete. 1: Operation in progress.
30:29	-	RO	0	**Reserved**
28	puf	R/W	0	**PUF Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
27:21	-	RO	0	**Reserved**
20	i2c2	R/W	0	**I²C2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
19	i2s	R/W	0	**I²S Interface Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
18	btle	R/W	0	**Bluetooth LE Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.

Reset 1			GCR_RST1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
17	-	RO	0	**Reserved**
16	smphr	R/W	0	**Semaphore Block Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
15:14	-	RO	-	**Reserved**
13	spi4	R/W	0	**SPI4 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
12	-	RO	0	**Reserved**
11	spi3	R/W	0	**SPI3 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
10:8	-	RO	0	**Reserved**
7	owm	R/W	0	**One-Wire Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
6:5	-	RO	0	**Reserved**
4	spixipm	R/W	0	**SPI XiP Master Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
3	spixip	R/W	0	**SPI XiP Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
2	-	RO	0	**Reserved**
1	pt	R/W	0	**Pulse Train Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
0	i2c1	R/W	0	**I²C1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.

Table 4-58: Peripheral Clock Disable Register 1

Peripheral Clock Disable 1			GCR_PCLKDIS1	[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.
30:28	-	RO	1	**Reserved**
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.
26:25	-	RO	1	**Reserved**
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	i2s	R/W	1	**I²S 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.
22:21	-	RO	1	**Reserved**

Peripheral Clock Disable 1			GCR_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
20	spixr	R/W	1	**SPI XiP RAM 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	can1	R/W	1	**CAN1 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.
18	-	RO	1	**Reserved**
17	spi4	R/W	1	**SPI4 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.
16	spi3	R/W	1	**SPI3 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.
15:14	-	RO	1	**Reserved**
13	owm	R/W	1	**One-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	-	RO	1	**Reserved**
11	can0	R/W	1	**CAN0 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.
10	-	RO	1	**Reserved**
9	smphr	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	-	RO	1	**Reserved**
7	syscache	R/W	1	**System 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.
6:5	-	RO	1	**Reserved**
4	hpb	R/W	1	**HyperBus 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_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
3	puf	R/W	1	**PUF 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.
2	trng	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	uart2	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	btle	R/W	1	**Bluetooth 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 4-59: Event Enable Register

Event Enable			GCR_EVENTEN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	tx	R/W	0	**CPU0 (CM4) TXEV Event Enable** When this bit is set, the TXEV event wakes the CM4 from a low-power mode entered with a WFE instruction. 0: Disabled. 1: Enabled.
1	-	RO	0	**Reserved**
0	dma	R/W	0	**CPU0 (CM4) DMA CTZ Wake-up Enable** Enables a DMA CTZ event to generate an RXEV interrupt to wake the CM4 from a low-power mode entered with a WFE instruction. 0: Disabled. 1: Enabled.

Table 4-60: Revision Register

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

Table 4-61: System Status Interrupt Enable Register

System Status Interrupt Enable			GCR_SYSIE	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**

System Status Interrupt Enable			GCR_SYSIE	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
5	syscacheerr	R/W	0	**System Cache Memory Fault Error Interrupt Enable** Set this field to generate an interrupt if a system cache memory fault error occurs. 0: Disabled. 1: Enabled.
4:2	-	RO	0	**Reserved**
1	codeinterr	R/W	0	**SPI XiP Code Integrity Error Interrupt Enable** Set this field to generate an interrupt if the *GCR_SYSST.codeinterr* field is set. 0: Disabled. 1: Enabled.
0	iceunlock	RO	0	**Reserved**

Table 4-62: Error Correction Coding Error Register

ECC Error			GCR_ECCERR	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8	icachexip	RO	0	**Reserved**
7	icache0	RO	0	**Reserved**
6	ram6	RO	0	**Reserved**
5	ram5	RO	0	**Reserved**
4	ram4	RO	0	**Reserved**
3	ram3	RO	0	**Reserved**
2	ram2	RO	0	**Reserved**
1	ram1	RO	0	**Reserved**
0	ram0	RO	0	**Reserved**

Table 4-63: Error Correction Coding Correctable Error Detected Register

ECC Correctable Error Detected			GCR_ECCCED	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8	icachexip	RO	0	**Reserved**
7	icache0	RO	0	**Reserved**
6	ram6	RO	0	**Reserved**
5	ram5	RO	0	**Reserved**
4	ram4	RO	0	**Reserved**
3	ram3	RO	0	**Reserved**
2	ram2	RO	0	**Reserved**
1	ram1	RO	0	**Reserved**
0	ram0	RO	0	**Reserved**

Table 4-64: Error Correction Coding Interrupt Enable Register

ECC Interrupt Enable			GCR_ECCIE	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8	icachexip	RO	0	**Reserved**
7	icache0	RO	0	**Reserved**
6	ram6	RO	0	**Reserved**
5	ram5	RO	0	**Reserved**
4	ram4	RO	0	**Reserved**
3	ram3	RO	0	**Reserved**
2	ram2	RO	0	**Reserved**
1	ram1	RO	0	**Reserved**
0	ram0	RO	0	**Reserved**

Table 4-65: Error Correction Coding Error Address Register

ECC Error Address			GCR_ECCADDR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**
31	te	R	0	**Reserved**
30	tb	R	0	**Reserved**
29:16	taddr	R	0	**Reserved**
15	de	R	0	**Reserved**
14	db	R	0	**Reserved**
13:0	daddr	R	0	**Reserved**

Table 4-66: Bluetooth LDO Control Register

Bluetooth LDO Control			GCR_BTLELDOCTRL	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ldorxbypenendly	R	0	**LDORX Bypass Enable Delay**
14	ldotxbypenendly	R	0	**LDOTX Bypass Enable Delay**
13	ldotxendly	R	0	**LDOTX Enable Delay**
12	ldorxendly	R	0	**LDORX Enable Delay**
11	ldorxdisch	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.
10	ldorxbyp	R/W	0	**LDORX Bypass Enable** This bit enables the LDORX bypass mode and charges the LDORX output voltage to its input voltage level.
9	ldotxdisch	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.
8	ldotxbyp	R/W	0	**LDOTX Bypass Enable** This bit enables the LDOTX bypass mode and charges the LDORX output voltage to its input voltage level.
7:6	ldotxvsel	R/W	0b01	**LDOTX Output Voltage Setting** 0b00: 0.85V. 0b01: 0.90V. 0b10: 1.0V. 0b11: 1.1V.
5	ldotxpulld	R/W	1	**LDOTX Pulldown** This bit is used to provide a weak pulldown to the LDOTX output. This bit is only valid if both *GCR_BTLELDOCTRL.ldotxen* and *GCR_BTLELDOCTRL.ldotxbyp* are set to 0. 0: Pulldown disabled. 1: Pulldown enabled.
4	ldotxen	R/W	0	**LDOTX Enable** Enable the TX LDO and charge its output voltage to the setting in *GCR_BTLELDOCTRL.ldotxvsel*. 0: Disabled. 1: Enabled.

Bluetooth LDO Control			GCR_BTLELDOCTRL	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**
3:2	ldorxvsel	R/W	0b01	**LDORX Output Voltage Setting** 0b00: 0.85V. 0b01: 0.9V. 0b10: 1.0V. 0b11: 1.1V.
1	ldorxpulld	R/W	1	**LDORX Pull Down** Setting this bit enables a weak pulldown on the output of the RX LDO. 0: Pulldown disabled. 1: Pulldown enabled.
0	ldorxen	R/W	0	**LDORX Enable** Enable the RX LDO and charge its output voltage to the setting in *GCR_BTLELDOCTRL.ldorxvsel*. 0: Disabled. 1: Enabled.

Table 4-67: Bluetooth LDO Delay Count Register

Bluetooth LDO Delay Count			GCR_BTLELDODLY	[0x0078]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	0	**Reserved**
28:20	ldotxdlycnt	R/W	0x1B	**Bluetooth LDOTX Delay Count** This field is the delay duration between *GCR_BTLELDOCTRL.ldotxen* and *GCR_BTLELDOCTRL.ldotxendly*. The delay duration should be sufficient for the LDO output to charge to the set output level (*GCR_BTLELDOCTRL.ldotxvsel*). *Note: The delay count is based on PCLK/128.*
19:17	-	RO	0	**Reserved**
16:8	ldorxdlycnt	R/W	0x1B	**Bluetooth LDORX Delay Count** This field is the delay duration between *GCR_BTLELDOCTRL.ldorxen* and *GCR_BTLELDOCTRL.ldorxendly*. The delay duration should be sufficient for the LDO output to charge to the set output level (*GCR_BTLELDOCTRL.ldorxvsel*). *Note: The delay count is based on PCLK/128.*
7:0	bypdlycnt	R/W	0x28	**Bluetooth LDO Bypass Delay Count** This field is the delay count between *GCR_BTLELDOCTRL.ldorxbyp* to *GCR_BTLELDOCTRL.ldorxbypenendly* and between *GCR_BTLELDOCTRL.ldotxbyp* and *GCR_BTLELDOCTRL.ldotxbypenendly*. *Note: The delay count is based on PCLK/128.*

Table 4-68: General Purpose 0 Register

General Purpose 0			GCR_GPR0	[0x0080]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**General Purpose Register** General-purpose register.

4.12 System Initialization Registers (SIR)

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 4-69: System Initialization Register Summary

Offset	Register Name	Description
[0x0000]	*SIR_SISTAT*	*System Initialization Status Register*

Offset	Register Name	Description
[0x0004]	SIR_SIADDR	System Initialization Address Error Register
[0x0100]	SIR_FSTAT	System initialization Function Status Register
[0x0104]	SIR_SFSTAT	System initialization Security Function Status Register

4.12.1 System Initialization Register Details

Table 4-70: System Initialization Status Register

System Initialization Status			SIR_SISTAT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	crcerr	RO	See Description	**Configuration Error Flag** This field is set by hardware during reset if an error in the device configuration is detected. 0: Configuration valid. 1: Configuration invalid. *Note: If this field reads 1, a device error has occurred. Please contact Analog Devices technical support for additional assistance providing the address contained in SIR_SIADDR.erraddr.*
0	magic	RO	See Description	**Configuration Valid Flag** This field is set to 1 by hardware during reset if the device configuration is valid. 0: Configuration invalid 1: Configuration valid *Note: If this field reads 0, the device configuration is invalid, and a device error has occurred during system initialization. Please contact Analog Devices technical support for additional assistance.*

Table 4-71: System Initialization Address Error Register

System Initialization Status			SIR_SIADDR	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	erraddr	RO	0	**Configuration Error Address** If the *SIR_SISTAT.crcerr* field is set to 1, the value in this register is the address of the configuration failure.

Table 4-72: System Initialization Function Status Register

System Initialization Function Status			SIR_FSTAT	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	btle	RO	See Description	**BTLE** This field indicates if the device includes the Bluetooth LE. 0: Peripheral is not available. 1: Peripheral is available.
8	-	RO	0	**Reserved**
7	smphr	RO	See Description	**Semaphore** This field indicates if the device includes the semaphore. 0: Peripheral is not available. 1: Peripheral is available.
6:5	-	RO	0	**Reserved**
4	hbc	RO	See Description	**HyperBus** This field indicates if the devices includes the SPI XiP peripheral. 0: Peripheral is not available. 1: Peripheral is available.

System Initialization Function Status			SIR_FSTAT	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
3	spixip	RO	See Description	**SPI XiP** This field indicates if the devices includes the SPI XiP peripheral. 0: Peripheral is not available. 1: Peripheral is available.
2	adc	RO	See Description	**ADC** This field indicates if the device includes the ADC peripheral. 0: Peripheral is not available. 1: Peripheral is available.
1	-	RO	0	**Reserved**
0	fpu	RO	See Description	**FPU** This field indicates if the device includes the FPU. 0: Block is not available. 1: Block is available.

Table 4-73: System Initialization Security Function Status Register

System Initialization Security Function Status			SIR_SFSTAT	[0x0104]
Bits	**Name**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**
5	maa	RO	See Description	**MAA** This field indicates if the device includes the AES block. 0: Block is not available. 1: Block is available.
4	sha	RO	See Description	**SHA** This field indicates if the device includes the AES block. 0: Block is not available. 1: Block is available.
3	aes	RO	See Description	**AES** This field indicates if the device includes the AES block. 0: Block is not available. 1: Block is available.
2	trng	RO	See Description	**TRNG** This field indicates if the device includes the TRNG block. 0: Block is not available. 1: Block is available.
1:0	-	RO	0	**Reserved**

4.13 Function Control Registers (FCR)

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 4-74: Function Control Register Summary

Offset	Register	Description
[0x0000]	*FCR_FCTRL0*	Function Control 0 Register
[0x0004]	*FCR_AUTOCAL0*	Automatic Calibration 0 Register
[0x0008]	*FCR_AUTOCAL1*	Automatic Calibration 1 Register
[0x000C]	*FCR_AUTOCAL2*	Automatic Calibration 2 Register
[0x0010]	*FCR_URVBOOTADDR*	RV32 Boot Address Register
[0x0014]	*FCR_URVCTRL*	RV32 Control Register
[0x0018]	*FCR_XO32MKS*	ERFO Kick-Start Register
[0x001C]	*FCR_SARBUFCN*	SAR ADC Buffer Control Register

Offset	Register	Description
[0x0020]	*FCR_TS0*	Temperature Sensor Gain Trim Register
[0x0024]	*FCR_TS1*	Temperature Sensor Offset Trim Register
[0x0028]	*FCR_ADCREFTRIM0*	ADC 1.25V Reference Trim Register
[0x002C]	*FCR_ADCREFTRIM1*	ADC 2.048V Reference Trim Register
[0x0030]	*FCR_ADCREFTRIM2*	ADC External Reference Trim Register

4.13.1 Function Control Register Details

Table 4-75: Function Control 0 Register

Function Control 0			FCR_FCTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
25	i2c2dgen1	R/W	0	**I2C2 SCL Glitch Filter Enable** 0: Disabled. 1: Enabled.
24	i2c2dgen0	R/W	0	**I2C2 SDA Glitch Filter Enable** 0: Disabled. 1: Enabled.
23	i2c1dgen1	R/W	0	**I2C1 SCL Glitch Filter Enable** 0: Disabled. 1: Enabled.
22	i2c1dgen0	R/W	0	**I2C1 SDA Glitch Filter Enable** 0: Disabled. 1: Enabled.
21	i2c0dgen1	R/W	0	**I2C0 SCL Glitch Filter Enable** 0: Disabled. 1: Enabled.
20	i2c0dgen0	R/W	0	**I2C0 SDA Glitch Filter Enable** 0: Disabled. 1: Enabled.
19:18	-	RO	0	**Reserved**
17:16	usbclksel	R/W	0	**USB Clock Select** This field selects the clock source for the USB peripheral. 0: 12MHz internal clock. 1: External clock USBCLKEXT (P0.27). 2: ERFO.
15:14	-	RO	0	**Reserved**
13:8	hypercgdly	R/W	0	**HyperBus Clock Generator Delay** This field controls the HyperBus clock generator delay. Each bit setting adds 600pS of delay to the delay for the fields as follows: 8-9: CLK0 and CLK180 delay control. 10-11: CLK90 and CLK270 delay control. 12:13: CLK0 to CLK0_VDIO delay control. For example, to add 1,800pS delay to CLK0 and CLK180, set bit 8 to 1 and bit 9 to 1.
7	-	RO	0	**Reserved**
6	rdsgcset	R/W	0	**HyperBus Read Data Strobe (RDS) Set** Setting this field to 1 enables software control of the HyperBus RDS gray code selection. 0: Internal gray code selection. 1: Use *FCR_FCTRL0.rdsgcsel* for gray code selection.
5:0	rdsgcsel	R/W	0	**HyperBus RDS Gray Code Select** This field is the gray code setting used when the *FCR_FCTRL0.rdsgcset* is set to 1. The optimum value of this field is 42.

Table 4-76: Automatic Calibration 0 Register

Automatic Calibration 0			FCR_AUTOCAL0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:23	hirc96mactmrout	RO	0	**IPO Automatic Trim Value Output** This field contains the calculated trim output from an automatic calibration run.
22:20	-	RO		**Reserved**
19:8	mu	R/W	0	**IPO Trim Pulse Count** Load this field with the desired number of trim adjustment pulses required before the trim is updated during automatic calibration operation. The recommended value for this field is 4.
7:5	-	RO	0	**Reserved**
4	atomic	R/W1O	0	**IPO Trim Atomic Start** Set this bit to start an automatic atomic calibration of the IPO. The calibration runs for *FCR_AUTOCAL2.donecnt* milliseconds. This bit is automatically cleared by hardware when the calibration is complete.
3	gaininv	R/W	0	**IPO Trim Step Invert** Set this field to invert the up/down trim steps during calibration operations. 0: Trim steps are not inverted. 1: Enable inverted trim steps for calibration.
2	ldtrm	R/*	0	**IPO Initial Trim Load** Set this bit to load the initial trim value for the IPO from *FCR_AUTOCAL1.inittrm*. This bit is cleared by hardware once the load is complete.
1	acrun	R/W	0	**IPO Automatic Calibration Run** 0: Not running. 1: Running.
0	acen	R/W	0	**IPO Automatic Calibration Enable** Selects the trim value to use for the IPO. The reset default for this field uses the factory trim for the IPO. Setting this field to 1 uses the automatic calibration trim output stored in *FCR_AUTOCAL0.ldtrm*. 0: Use default trim from the factory. 1: Use automatic calibration trim value calculated and stored in *FCR_AUTOCAL0.ldtrm*.

Table 4-77: Automatic Calibration 1 Register

Automatic Calibration 1			FCR_AUTOCAL1	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	inittrm	R/W	0	**IPO Trim Automatic Calibration Initial Trim** This field contains the initial trim setting for the IPO. Set this field to the desired initial trim value to use for an IPO automatic calibration operation. The closer this field is to the target trim value required, the faster the automatic trim operation completes. Set this field to 0x40 when performing autocalibration. *Note: Valid values for this field are 1 to 0xFF.*

Table 4-78: Automatic Calibration 2 Register

Automatic Calibration 2			FCR_AUTOCAL2	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0	**Reserved**

Automatic Calibration 2			FCR_AUTOCAL2	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
20:8	acdiv	R/W	0	**IPO Trim Automatic Calibration Divide Factor** The target frequency of the calibration is determined by dividing the IPO frequency by 32,768 before comparing it with the ERTCO frequency. For example, to achieve a target IPO frequency of 120MHz, load this field with 0xE4E (3,662). *Equation 4-2: IPO Divisor Equation* $acdiv = \frac{f_{IPO\_TARGET}}{32768}$ *Note: Setting acdiv to 0 is equivalent to setting acdiv to 1.*
7:0	donecnt	R/W	0	**IPO Trim Automatic Calibration Run Time** Set this field to the desired number of milliseconds for the atomic calibration run time for the IPO. $Automatic\ Calibration\ Run\ Time = donecnt\ (milliseconds)$

Table 4-79: RV32 Boot Address Register

RV32 Boot Address			FCR_URVBOOTADDR	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0x2001 C000	**RV32 Boot Address** Set this field to the boot address for the RV32 core. The reset value for this register is 0x2001 C000.

Table 4-80: RV32 Control Register

RV32 Boot Address			FCR_URVCTRL	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	iflushen	R/W	0	**ICC1 Cache Flush** Write 1 to flush the cache and the instruction buffer for the RV32 core. This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect and does not stop a cache flush in progress. 0: ICC1 flush complete. 1: Flush the contents of the ICC1 cache.
0	memsel	R/W	0	**RV32 Memory Select** This field determines if *sysram8* is shared between the CM4 and RV32 cores. Set this field to 1 to set the RV32 core as the exclusive master for *sysram8*. 0: *Sysram8* is shared and accessible by both the CM4 and RV32 cores. 1: *Sysram8* is accessible by the RV32 core only. *Note: The software must ensure that no accesses occur in sysram8 before setting this field to 1. See Semaphores for information on multiprocessor communications.*

Table 4-81: ERFO Kick-Start Register

ERFO Kick-Start			FCR_XO32MKS	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:14	-	R/W	0	**Reserved**
13:12	clksel	R/W	0	**Kick-Start Clock Select for ERFO** Set this field to the clock source to use for kick-starting the ERFO. 0: No kick start clock. 1: Reserved. 2: ISO. 3: IPO.

ERFO Kick-Start			FCR_XO32MKS	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
11	pulse	R/W	0	**Kick-Start ERFO Double Pulse Length** Setting this field to 1 enables double pulse length for the kick-start pulses. 0: Disabled. 1: Enabled.
10:8	driver	R/W	0	**Kick-Start ERFO Drive Strength** This field controls the drive strength of the kick-start pulses. 0: Disabled. 1 – 7: Drive strength selection.
7	en	R/W	0	**Kick-Start ERFO Enable** Set this field to 1 to start the kick start of the ERFO. 0: Disabled. 1: Enabled.
6:0	clock	R/W	0	**Kick-Start ERFO Counter** Set this value to the number of kick start counts required to improve the start time for the ERFO. See *ERFO Kick-Start* for details.

Table 4-82: SAR ADC Buffer Control Register

SAR ADC Buffer Control			FCR_SARBUFCN	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:13	-	R/W	0	**Reserved**
12:11	divsel	R/W	0	**SAR Signal Path Select** This field selects one of the signal divider paths in the analog input buffer path. The setting for this field applies to all analog input's using the analog input buffer signal path (*FCR_SARBUFCN.thru_pad_sw_en7 - FCR_SARBUFCN.thru_pad_sw_en0 = 1*). 0: Infinite resistance to GND, input divide by 1. 1: 50KΩ to GND, input divide by 1. 2: 50KΩ to GND, input divide by 2. 3: 50KΩ to GND, input divide by 3.
10	thru_rri_en	R/W	0	**SAR Buffer Rail-to-Rail Input Enable** This field enables rail-to-rail inputs (0.1V to V_{DD} - 0.1V) for the amplifier buffer path. This field applies to all inputs using the analog input buffer signal path (*FCR_SARBUFCN.thru_pad_sw_en7 - FCR_SARBUFCN.thru_pad_sw_en0 = 1*). 0: Inputs restricted to V_{GS} + V_{DSAT} to V_{DD} - 0.1V. 1: Rail-to-rail input enabled.
9	ramp_en	R/W	0	**Buffer Amplifier Enable** Set this field to 1 to enable the buffer amplifier used in the buffer path. 0: Disabled. 1: Enabled.
8	thru_en	R/W	0	**Mux Switches Enable** Set this field to enable the MUX switches (*FCR_SARBUFCN.thru_pad_sw_en7 - FCR_SARBUFCN.thru_pad_sw_en0*) using the analog input buffer path. 0: Disabled. 1: Enabled.
7	thru_pad_sw_en7	R/W	0	**Mux Pad Switch Enable for AIN7** Set this field to 1 to enable the input switch for this channel in the analog input buffer path. 0: Disabled. 1: Enabled.

SAR ADC Buffer Control			FCR_SARBUFCN	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
6	thru_pad_sw_en6	R/W	0	**Mux Pad Switch Enable for AIN6** Set this field to 1 to enable the input switch for this channel in the analog input buffer path. 0: Disabled. 1: Enabled.
5	thru_pad_sw_en5	R/W	0	**Mux Pad Switch Enable for AIN5** Each analog input (AIN0 through AIN7) includes a mux switch in its pad. Set the corresponding bit in this field to enable the mux switch for a given analog input. 0: Disabled. 1: Enabled.
4	thru_pad_sw_en4	R/W	0	**Mux Pad Switch Enable for AIN4** Set this field to 1 to enable the input switch for this channel in the analog input buffer path. 0: Disabled. 1: Enabled.
3	thru_pad_sw_en3	R/W	0	**Mux Pad Switch Enable for AIN3** Set this field to 1 to enable the input switch for this channel in the analog input buffer path. 0: Disabled. 1: Enabled.
2	thru_pad_sw_en2	R/W	0	**Mux Pad Switch Enable for AIN2** Set this field to 1 to enable the input switch for this channel in the analog input buffer path. 0: Disabled. 1: Enabled.
1	thru_pad_sw_en1	R/W	0	**Mux Pad Switch Enable for AIN1** Set this field to 1 to enable the input switch for this channel in the analog input buffer path. 0: Disabled. 1: Enabled.
0	thru_pad_sw_en0	R/W	0	**Mux Pad Switch Enable for AIN0** Set this field to 1 to enable the input switch for this channel in the analog input buffer path. 0: Disabled. 1: Enabled.

Table 4-83: Temperature Sensor Gain Register

Temperature Sensor Gain			FCR_TS0	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	R	0	**Reserved**
11:0	gain	R	*	**Reserved**

Table 4-84: Temperature Sensor Offset Register

Temperature Sensor Offset			FCR_TS1	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:14	ts_offset_sign	R	*	**Reserved**
13:0	offset	R	*	**Reserved**

Table 4-85: ADC 1.25V Reference Trim Register

ADC 1.25V Reference Trim			FCR_ADCREFTRIM0	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	R	0	**Reserved**

ADC 1.25V Reference Trim			FCR_ADCREFTRIM0	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
29:24	vx2_tune		*	**Tuning Capacitor In-Line DAC** See *1.25V Internal Reference Trim* for details on this field.
23:18	-	RO	0	**Reserved**
17:16	vcm	R	*	**Trim Code for V_{CM} Output of Reference Buffer** See *1.25V Internal Reference Trim* for details on this field.
15	-	RO	0	**Reserved**
14:8	vrefm	R	*	**Trim Code for V_{REFM} Output of Reference Buffer** See *1.25V Internal Reference Trim* for details on this field.
7	-	RO	0	**Reserved**
6:0	vrefp	R	*	**Trim Code for V_{REFP} Output of Reference Buffer** See *1.25V Internal Reference Trim* for details on this field.

Table 4-86: ADC 2.048V Reference Trim Register

ADC 2.048V Reference Trim			FCR_ADCREFTRIM1	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	R	0	**Reserved**
29:24	vx2_tune		*	**Tuning Capacitor In-Line DAC** See *2.048V Internal Reference Trim* for details on this field.
23:18	-	RO	0	**Reserved**
17:16	vcm	R	*	**Trim Code for V_{CM} Output of Reference Buffer** See *2.048V Internal Reference Trim* for details on this field.
15	-	RO	0	**Reserved**
14:8	vrefm	R	*	**Trim Code for V_{REFM} Output of Reference Buffer** See *2.048V Internal Reference Trim* for details on this field.
7	-	RO	0	**Reserved**
6:0	vrefp	R	*	**Trim Code for V_{REFP} Output of Reference Buffer** See *2.048V Internal Reference Trim* for details on this field.

Table 4-87: ADC External Reference Trim Register

ADC External Reference Trim			FCR_ADCREFTRIM2	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	RO	0	**Reserved**
29:24	vx2_tune	R	*	**Tuning Capacitor In-Line DAC** See *External Reference Trim* for details on this field.
23:18	-	RO	0	**Reserved**
17:16	vcm	R	*	**Trim Code for V_{CM} Output of Reference Buffer** See *External Reference Trim* for details on this field.
15	-	RO	0	**Reserved**
12	iboost_2p048	R	0	**Extra Drive Current Enable for 2.048V Reference** See *2.048V Internal Reference Trim* for details on this field.
11:8	idrv_2p048	R	*	**Trim Code for 2.048V Reference Buffer Drive Strength** See *2.048V Internal Reference Trim* for details on this field.
7:5	-	RO	0	**Reserved**
4	iboost_1p25	R	0	**Extra Drive Current Enable for 1.25V Reference** See *1.25V Internal Reference Trim* for details on this field.
3:0	idrv_1p25	R	*	**Trim Code for 1.25V Reference Buffer Drive Strength** See *1.25V Internal Reference Trim* for details on this field.

4.14 Global Control Function Registers (GCFR)

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a POR only.

Table 4-88: Global Control Function Register Summary

Offset	Register	Description
[0x0000]	*GCFR_REG0*	HyperBus 0 Register

Table 4-89: HyperBus 0 Register

HyperBus 0			**GCFR_REG0**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	RO	0	**Reserved**
8	rdsdll_en	R/W	0	**HyperBus RDS Delay Lock Loop (DLL) Enable** 0: Disabled. 1: Enabled. When the HyperBus is reading from an external device, the RWDS pin acts as a read data strobe.
7:4	ckndrv	R/W	0	**HyperBus CKN Drive Setting**
3:0	ckpdrv	R/W	0	**HyperBus CKP Drive Setting**

5. Interrupts and Exceptions

Interrupts and exceptions are managed by the Arm Cortex-M4 with FPU Nested Vector Interrupt Controller (NVIC) or the RV32 interrupt controller. The NVIC manages the interrupts, exceptions, priorities, and masking. *Table 5-1* and *Table 5-2* detail the MAX32690's interrupt vector tables for the CM4 and RV32 processors respectively, and describe each exception and interrupt.

5.1 CM4 Interrupt and Exception Features

- Eight programmable priority levels
- Nested exception and interrupt support
- Interrupt masking

5.2 CM4 Interrupt Vector Table

Table 5-1 lists the interrupt and exception table for the MAX32690's CM4 core. There are 112 interrupt entries for the MAX32690, including reserved interrupt placeholders. Including the 15 system exceptions for the Arm Cortex-M4 with FPU, the total number of entries is 127.

Table 5-1: MAX32690 CM4 Interrupt Vector Table

Exception (Interrupt) Number	Offset	Name	Description
1	[0x0004]	Reset_IRQn	Reset
2	[0x0008]	NonMaskableInt_IRQn	Non-Maskable Interrupt
3	[0x000C]	HardFault_IRQn	Hard Fault
4	[0x0010]	MemoryManagement_IRQn	Memory Management Fault
5	[0x0014]	BusFault_IRQn	Bus Fault
6	[0x0018]	UsageFault_IRQn	Usage Fault
7:10	[0x001C]-[0x0028]	-	Reserved
11	[0x002C]	SVCall_IRQn	Supervisor Call Exception
12	[0x0030]	DebugMonitor_IRQn	Debug Monitor Exception
13	[0x0034]	-	Reserved
14	[0x0038]	PendSV_IRQn	Request Pending for System Service
15	[0x003C]	SysTick_IRQn	System Tick Timer
16	[0x0040]	PF_IRQn	Power Fail interrupt
17	[0x0044]	WDT0_IRQn	Windowed Watchdog Timer 0 Interrupt
18	[0x0048]	USB_IRQn	USBHS Interrupt
19	[0x004C]	RTC_IRQn	Real-time Clock Interrupt
20	[0x0050]	TRNG_IRQn	True Random Number Generator Interrupt
21	[0x0054]	TMR0_IRQn	Timer 0 Interrupt
22	[0x0058]	TMR1_IRQn	Timer 1 Interrupt
23	[0x005C]	TMR2_IRQn	Timer 2 Interrupt
24	[0x0060]	TMR3_IRQn	Timer 3 Interrupt
25	[0x0064]	TMR4_IRQn	Low-Power Timer 0 (TMR4) Interrupt
26	[0x0068]	TMR5_IRQn	Low-Power Timer 1 (TMR5) Interrupt
27:28	[0x006C]:[0x0070]	-	Reserved
29	[0x0074]	I2C0_IRQn	I^2C Port 0 Interrupt
30	[0x0078]	UART0_IRQn	UART Port 0 Interrupt
31	[0x007C]	UART1_IRQn	UART Port 1 Interrupt
32	[0x0080]	SPI0_IRQn	SPI Port 0 Interrupt
33	[0x0084]	SPI1_IRQn	SPI Port 1 Interrupt
34	[0x0088]	SPI2_IRQn	SPI Port 2 Interrupt
35	[0x008C]	-	Reserved
36	[0x0090]	ADC_IRQn	ADC Interrupt

Exception (Interrupt) Number	Offset	Name	Description
37:38	[0x0094]:[0x0098]	-	Reserved
39	[0x009C]	FLC0_IRQn	Flash Controller 0 Interrupt
40	[0x00A0]	GPIO0_IRQn	GPIO Port 0 Interrupt
41	[0x00A4]	GPIO1_IRQn	GPIO Port 1 Interrupt
42	[0x00A8]	GPIO2_IRQn	GPIO Port 2 Interrupt
43	[0x00AC]	CRYPTO_IRQn	Crypto Tool Box Interrupt
44	[0x00B0]	DMA0_IRQn	DMA0 Interrupt
45	[0x00B4]	DMA1_IRQn	DMA1 Interrupt
46	[0x00B8]	DMA2_IRQn	DMA2 Interrupt
47	[0x00BC]	DMA3_IRQn	DMA3 Interrupt
48:49	[0x00C0 : 0x00C4]	-	Reserved
50	[0x00C8]	UART2_IRQn	UART Port 2 Interrupt
51	[0x00CC]	-	Reserved
52	[0x00D0]	I2C1_IRQn	I^2C Port 1 Interrupt
53	[0x00D4]	-	Reserved
54	[0x00D8]	SPIXC_IRQn	SPI-XiP Interrupt
55	[0x00DC]	BTLE_TX_DONE_IRQn	Bluetooth Transmitter Done Interrupt
56	[0x00E0]	BTLE_RX_RCVD_IRQn	Bluetooth Receive Data Interrupt
57	[0x00E4]	BTLE_RX_ENG_DET_IRQn	Bluetooth Receive Energy Detected Interrupt
58	[0x00E8]	BTLE_SFD_DET_IRQn	BTLE SFD Detected
59	[0x00EC]	BTLE_SFD_TO_IRQn	BTLE SFD Timeout
60	[0x00F0]	BTLE_GP_EVENT_IRQn	BTLE Timestamp
61	[0x00F4]	BTLE_CFO_IRQn	BTLE CFO Done
62	[0x00F8]	BTLE_SIG_DET_IRQn	BTLE Signal Detected
63	[0x00FC]	BTLE_AGC_EVENT_IRQn	BTLE AGC Event
64	[0x0100]	BTLE_RFFE_SPIM_IRQn	BTLE RFFE SPIM Done
65	[0x0104]	BTLE_TX_AES_IRQn	BTLE TX AES Done
66	[0x0108]	BTLE_RX_AES_IRQn	BTLE RX AES Done
67	[0x010C]	BTLE_INV_APB_ADDR_IRQn	BTLE Invalid APB Address
68	[0x0110]	BTLE_IQ_DATA_VALID_IRQn	BTLE IQ Data Valid
69	[0x0114]	WUT0_IRQn	Wake-up Timer 0 Interrupt
70	[0x0118]	GPIOWAKE_IRQn	GPIO Wake-up Interrupt
71	[0x011C]	-	Reserved
72	[0x0120]	SPI3_IRQn	SPI Port 3 Interrupt
73	[0x0124]	WDT1_IRQn	Low-Power Watchdog Timer 0 (WDT1) Interrupt
74	[0x0128]	GPIO3_IRQn	GPIO Port 3 Interrupt
75	[0x012C]	PT_IRQn	Pulse Train Interrupt
76	[0x0130]	-	Reserved
77	[0x0134]	HPB_IRQn	HyperBus Interrupt
78	[0x0138]	I2C2_IRQn	I^2C Port 2 Interrupt
79	[0x013C]	RISCV_IRQn	RV32 Interrupt
80:82	[0x0140]:[0x0148]	-	Reserved
83	[0x014C]	OWM_IRQn	1-Wire Controller Interrupt
84	[0x0150]	DMA4_IRQn	DMA4 Interrupt
85	[0x0154]	DMA5_IRQn	DMA5 Interrupt
86	[0x0158]	DMA6_IRQn	DMA6 Interrupt
87	[0x015C]	DMA7_IRQn	DMA7 Interrupt
88	[0x0160]	DMA8_IRQn	DMA8 Interrupt
89	[0x0164]	DMA9_IRQn	DMA9 Interrupt
90	[0x0168]	DMA10_IRQn	DMA10 Interrupt
91	[0x016C]	DMA11_IRQn	DMA11 Interrupt
92	[0x0170]	DMA12_IRQn	DMA12 Interrupt
93	[0x0174]	DMA13_IRQn	DMA13 Interrupt
94	[0x0178]	DMA14_IRQn	DMA14 Interrupt

Exception (Interrupt) Number	Offset	Name	Description
95	[0x017C]	DMA15_IRQn	DMA15 Interrupt
96	[0x0180]	USBDMA_IRQn	USB DMA Interrupt
97	[0x0184]	-	Reserved
98	[0x0188]	ECC_IRQn	Error Correction Coding Block Interrupt
99:100	[0x018C]:[0x0190]	-	Reserved
101	[0x0194]	SCA_IRQn	SCA Crypto Accelerator Interrupt
102	[0x0198]	-	Reserved
103	[0x019C]	FLC1_IRQn	Flash Controller 1 Interrupt
104	[0x01A0}	UART3_IRQn	Low-Power UART 0 (UART3) Interrupt
105:110	[0x01A4]:[0x01B8]	-	Reserved
111	[0x01BC]	PUF_IRQn	Physically Unclonable Function Interrupt
112	[0x01C0]:[0x01C8]	-	Reserved
115	[0x01CC]	I2S_IRQn	I^2S Interrupt
116:118	[0x01D0]:[0x01D8]	-	Reserved
119	[0x01DC]	LPCMP_IRQn	Low-Power Comparator Interrupt
120	[0x01E0]	-	Reserved
121	[0x01E4]	SPI4_IRQn	SPI 4 Interrupt
122	[0x01E8]	-	Reserved
123	[0x01EC]	CAN0_IRQn	CAN 0 Interrupt
124	[0x01F0]	CAN1_IRQn	CAN 1 Interrupt
125	[0x01F4]	WUT1_IRQn	Wake-up Timer 1 Interrupt
126:127	[0x01F8]:[0x1FC]	-	Reserved

5.3 RV32 Interrupt Vector Table

Table 5-2 lists the interrupt and exception table for the MAX32690's RV32 core.

Table 5-2: MAX32690 RV32 Interrupt Vector Table

Exception (Interrupt) Number	Name	Description
4	PF_IRQn	Power Fail interrupt
5	WDT0_IRQn	Windowed Watchdog Timer 0 Interrupt
6	GPIOWAKE_IRQn	GPIO Wake-up Interrupt
6	LPCMP_IRQn	Low-Power Comparator Interrupt
7	RTC_IRQn	RTC Interrupt
8	TMR0_IRQn	Timer 0 Interrupt
9	TMR1_IRQn	Timer 1 Interrupt
10	TMR2_IRQn	Timer 2 Interrupt
11	TMR3_IRQn	Timer 3 Interrupt
12	TMR4_IRQn	Low-Power Timer 0 (TMR4) Interrupt
13	TMR5_IRQn	Low-Power Timer 1 (TMR5) Interrupt
14	I2C0_IRQn	I²C Port 0 Interrupt
15	UART0_IRQn	UART Port 0 Interrupt
16	CM4_IRQn	CM4 Interrupt
17	I2C1_IRQn	I²C Port 1 Interrupt
18	UART1_IRQn	UART Port 1 Interrupt
19	UART2_IRQn	UART Port 2 Interrupt
20	I2C2_IRQn	I²C Port 2 Interrupt
21	UART3_IRQn	Low-Power UART 0 (UART3) Interrupt
22	SPI0_IRQn	SPI Port 0 Interrupt
23	WUT0_IRQn	Wake-up Timer 0 Interrupt
24	FLC1_IRQn	Flash Controller 1 Interrupt
25	GPIO0_IRQn	GPIO Port 0 Interrupt
26	GPIO1_IRQn	GPIO Port 1 Interrupt
27	GPIO3_IRQn	GPIO Port 3 Interrupt
28	DMA0_IRQn	DMA0 Interrupt
29	DMA1_IRQn	DMA1 Interrupt
30	DMA2_IRQn	DMA2 Interrupt
31	DMA3_IRQn	DMA3 Interrupt
32	BTLE_TX_DONE_IRQn	Bluetooth Transmitter Done Interrupt
33	BTLE_RX_RCVD_IRQn	Bluetooth Receive Data Interrupt
34	BTLE_RX_ENG_DET_IRQn	Bluetooth Receive Energy Detected Interrupt
35	BTLE_SFD_DET_IRQn	BTLE SFD Detected
36	BTLE_SFD_TO_IRQn	BTLE SFD Timeout
37	BTLE_GP_EVENT_IRQn	BTLE Timestamp
38	BTLE_CFO_IRQn	BTLE CFO Done
39	BTLE_SIG_DET_IRQn	BTLE Signal Detected
40	BTLE_AGC_EVENT_IRQn	BTLE AGC Event
41	BTLE_RFFE_SPIM_IRQn	BTLE RFFE SPIM Done
42	BTLE_TX_AES_IRQn	BTLE TX AES Done
43	BTLE_RX_AES_IRQn	BTLE RX AES Done
44	BTLE_INV_APB_ADDR_IRQn	BTLE Invalid APB Address
45	BTLE_IQ_DATA_VALID_IRQn	BTLE IQ Data Valid
46	DMA4_15_IRQn	DMA Channel 4 through 15 Interrupt
47	TRNG_IRQn	TRNG Interrupt
48	WDT1_IRQn	Low-Power Watchdog Timer 0 (WDT1) Interrupt
49-50	-	Reserved
51	WUT1_IRQn	Wake-up Timer 1 Interrupt
52	PT_IRQn	Pulse Train Interrupt

Exception (Interrupt) Number	Name	Description
53	ADC_IRQn	ADC Interrupt
54	OWM_IRQn	1-Wire Controller Interrupt
55	I2S_IRQn	I²S Interrupt
56-57	-	Reserved
58	CAN0_IRQn	CAN 0 Interrupt
59	-	Reserved
60	GPIO2_IRQn	GPIO Port 2 Interrupt
61	SPI1_IRQn	SPI 1 Interrupt
62	SPI2_IRQn	SPI 2 Interrupt
63	CAN1_IRQn	CAN 1 Interrupt

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

GPIO pins can be individually configured to operate in a digital I/O mode or in an alternate function (AF) mode that maps a signal associated with an enabled peripheral to that GPIO. The GPIOs support dynamic switching between I/O mode and alternate function mode. 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 *GPIOn_IN* register.

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

GPIOs are logically divided into ports of 32 pins. Package variants may not implement all pins of a specific 32-bit GPIO port.

Each pin of a port 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 the section *GPIO Interrupt Handling*.

Note: The register set used to control the GPIO are identical across multiple Analog Devices microcontrollers. However the behavior of several registers vary depending on the specific device. The behavior of the registers should not be assumed to be the same from one device to a different device. Specifically the registers GPIOn_PADCTRL0, GPIOn_PADCTRL1, GPIOn_HYSEN, GPIOn_SRSEL, GPIOn_DS0, GPIOn_DS1, and GPIOn_VSSEL are device dependent in their usage. GPIO4 is controlled differently and has different features than the other GPIO ports in the MAX32690. Details for using GPIO4 are covered in the system chapter.

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 *GPIOn_OUT* register or an enabled peripheral.
- Input data can be read from *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 for GPIO0, GPIO1, and GPIO2:
  - V_{DDIO}
  - V_{DDIOH}
  - GPIO that support HyperBus as an alternate function are tied to the V_{DDIO} supply and do not support V_{DDIOH} supply selection.
- Selectable interrupt events:
  - Level triggered low
  - Level triggered high
  - Edge-triggered rising edge
  - Edge-triggered falling edge
  - Edge-triggered rising and falling edge
- All GPIO pins default to input mode with weak pullup during power-on-reset events.

6.1 Instances

Refer to the device data sheet for the details of the 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.

Note: See Power Sequencer Registers (PWRSEQ) for details on using GPIO4.

Note: Refer to the MAX32690 device data sheet for a description of alternate functions for each GPIO port pin.

6.2 Configuration

Each device pin can be individually configured as a GPIO or an alternate function. The correct alternate function setting must be selected for each pin of a given multipin peripheral for proper operation.

6.2.1 Peripheral Clock Enable

Each GPIO port is disabled by default on a reset. Use of a GPIO port requires enabling the peripheral clock for the port. See the *GCR_PCLKDIS0* register for details.

6.2.2 Power-On-Reset Configuration

During a POR event, all I/O default to GPIO mode as high impedance inputs except the SWDIO and SWDCLK pins. The SWD is enabled by default after POR with AF1 selected by hardware. See the *Secure Communication Protocol Bootloader (SCPBL)* chapter for exceptions.

Following a POR event, all GPIO except device pins that have the SWDIO and SWDCLK function, are configured with the following default settings:

- GPIO mode enabled
  - *GPIOn_EN0*.*[pin]* = 1
  - *GPIOn_EN1*.*[pin]* = 0
  - *GPIOn_EN2*.*[pin]* = 0
- Pullup/Pulldown disabled, I/O in Hi-Z mode
  - *GPIOn_PADCTRL0*.*[pin]* = 0
  - *GPIOn_PADCTRL1*.*[pin]* = 0
- Output mode disabled
  - *GPIOn_OUTEN*.*[pin]* = 0
- Interrupt disabled
  - *GPIOn_INTEN*.*[pin]* = 0

6.2.3 Serial Wire Debug Configuration

Perform the following steps to configure the SWDIO and SWDCLK device pins for SWD mode:

1. Set the device pin P0.28 for AF1 mode:
   a. GPIO0_EN0.[28] = 0
   b. GPIO0_EN1.[28] = 0
   c. GPIO0_EN2.[28] = 0
2. Set device pin P0.29 for AF1 mode:
   a. GPIO0_EN0.[29] = 0
   b. GPIO0_EN1.[29] = 0
   c. GPIO0_EN2.[29] = 0

Note: To use the SWD pins in I/O mode, set the desired GPIO pins for SWD AF and set the SWD disable field to 1 (GCR_SYSCTRL.swd_dis = 1).

6.2.4 Pin Function Configuration

Table 6-1 depicts the bit settings for the GPIOn_EN0, GPIOn_EN1, and GPIOn_EN2 registers to configure the function of the GPIO port pins. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO0_EN0.[25]*, *GPIO0_EN1.[25]*, and *GPIO0_EN2.[25]* all represent configuration for device pin P0.25. See Table 6-4 for a detailed example of how each of these bits applies to each of the GPIO device pins.

Table 6-1: MAX32690 GPIO Pin Function Configuration

MODE	GPIOn_EN2.[pin]	GPIOn_EN1.[pin]	GPIOn_EN0.[pin]
AF1	0	0	0
AF2	0	1	0
AF3	1	0	0
I/O (transition to AF1)	0	0	1
I/O (transition to AF2)	0	1	1
I/O (transition to AF3)	1	0	1

6.2.5 Input Mode Configuration

Table 6-2 depicts the bit settings for the digital I/O input mode. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO0_PADCTRL1.[25]*, *GPIO0_PADCTRL0.[25]*, *GPIO0_PS.[25]*, and *GPIO0_VSSEL. [25]* all represent configuration for device pin P0.25. See Table 6-7 for a detailed example of how each of these bits applies to each of the GPIO device pins. Refer to the MAX32690 data sheet for details of specific electrical characteristics.

Table 6-2: MAX32690 Input Mode Configuration

Input Mode	Mode Select		Pullup/Pulldown Strength	Power Supply
	GPIOn_PADCTRL1.[pin]	GPIOn_PADCTRL0.[pin]	GPIOn_PS.[pin]	GPIOn_VSSEL.[pin]
High-impedance	0	0	N/A	N/A
Weak Pullup to V_{DDIO} (1MΩ)	0	1	0	0
Strong Pullup to V_{DDIO} (25KΩ)	0	1	1	0
Weak Pulldown to V_{DDIOH} (1MΩ)	1	0	0	1
Strong Pulldown to V_{DDIOH} (25KΩ)	1	0	1	1
Reserved	1	1	N/A	N/A
Note: GPIO that support HyperBus as an alternate function are tied to the V_{DDIO} supply and do not support V_{DDIOH} supply selection. Refer to the device data sheet's alternate function table to determine pins that support HyperBus.				

6.2.6 Output Mode Configuration

Table 6-3 shows the configuration options for digital I/O in output mode. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO2_DS0.[7], GPIO2_DS1.[7], and GPIO2_VSSEL.[7]* all

represent configuration for device pin P2.7. See *Table 6-7* for a detailed example of how each of these bits applies to each of the GPIO device pins. Refer to the MAX32690 data sheet for details of specific electrical characteristics.

Table 6-3: MAX32690 Output Mode Configuration

Input Mode	Drive Strength		Power Supply
	GPIOn_DS1.[pin]	*GPIOn_DS0.[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 *Table 6-8*.

6.3 Reference Tables

The tables in this section provide example references for register bit assignment to configure a device's GPIO pins.

Table 6-4: MAX32690 GPIO Alternate Function Configuration Reference

Device Pin	Alternate Function Configuration Bits		
P0.0	*GPIO0_EN0.[0]*	*GPIO0_EN1.[0]*	*GPIO0_EN2.[0]*
P0.1	*GPIO0_EN0.[1]*	*GPIO0_EN1.[1]*	*GPIO0_EN2.[1]*
…	…	…	…
P0.30	*GPIO0_EN0.[30]*	*GPIO0_EN1.[30]*	*GPIO0_EN2.[30]*
P0.31	*GPIO0_EN0.[31]*	*GPIO0_EN1.[31]*	*GPIO0_EN2.[31]*

Table 6-5: MAX32690 GPIO Output/Input Configuration Reference

Device Pin	GPIO Output Enable	GPIO Output Write	GPIO Input Enable	GPIO Input Read
P0.0	*GPIO0_OUTEN.[0]*	*GPIO0_OUT.[0]*	*GPIO0_INEN.[0]*	*GPIO0_IN.[0]*
P0.1	*GPIO0_OUTEN.[1]*	*GPIO0_OUT.[1]*	*GPIO0_INEN.[1]*	*GPIO0_IN.[1]*
…	…	…	…	…
P0.30	*GPIO0_OUTEN.[30]*	*GPIO0_OUT.[30]*	*GPIO0_INEN.[30]*	*GPIO0_IN.[30]*
P0.31	*GPIO0_OUTEN.[31]*	*GPIO0_OUT.[31]*	*GPIO0_INEN.[31]*	*GPIO0_IN.[31]*

Table 6-6: MAX32690 GPIO Interrupt Configuration Reference

Device Pin	Enable	Status	Dual Edge	Polarity	Trigger	Wake-up
P0.0	*GPIO0_INTEN.[0]*	*GPIO0_INTFL.[0]*	*GPIO0_DUALEDGE.[0]*	*GPIO0_INTPOL.[0]*	*GPIO0_INTMODE. 0]*	*GPIO0_WKEN.[0]*
P0.1	*GPIO0_INTEN.[1]*	*GPIO0_INTFL.[1]*	*GPIO0_DUALEDGE.[1]*	*GPIO0_INTPOL.[1]*	*GPIO0_INTMODE.[1]*	*GPIO0_WKEN.[1]*
…	…	…	…	…	…	…
P0.30	*GPIO0_INTEN.[30]*	*GPIO0_INTFL.int[30]*	*GPIO0_DUALEDGE.[30]*	*GPIO0_INTPOL.[30]*	*GPIO0_INTMODE.[30]*	*GPIO0_WKEN.[30]*
P0.31	*GPIO0_INTEN.[31]*	*GPIO0_INTFL.int[31]*	*GPIO0_DUALEDGE.[31]*	*GPIO0_INTPOL.[31]*	*GPIO0_INTMODE.[31]*	*GPIO0_WKEN.[31]*

Table 6-7: MAX32690 GPIO Pullup/Pulldown/Drive Strength/Voltage Configuration Reference

Device Pin	Pullup/Pulldown/Strength Select			Drive Strength		Voltage
P0.0	*GPIO0_PADCTRL0.[0]*	*GPIO0_PADCTRL1.[0]*	*GPIO0_PS.[0]*	*GPIO0_DS0.[0]*	*GPIO0_DS1.[0]*	*GPIO0_VSSEL.[0]*
P0.1	*GPIO0_PADCTRL0.[1]*	*GPIO0_PADCTRL1.[1]*	*GPIO0_PS.[1]*	*GPIO0_DS0.[1]*	*GPIO0_DS1.[1]*	*GPIO0_VSSEL.[1]*
…	…	…	…	…	…	…
P0.30	*GPIO0_PADCTRL0.[30]*	*GPIO0_PADCTRL1.[30]*	*GPIO0_PS.[30]*	*GPIO0_DS0.[30]*	*GPIO0_DS1.[30]*	*GPIO0_VSSEL.[30]*
P0.31	*GPIO0_PADCTRL0.[31]*	*GPIO0_PADCTRL1.[31]*	*GPIO0_PS.[31]*	*GPIO0_DS0.[31]*	*GPIO0_DS1.[31]*	*GPIO0_VSSEL.[31]*

6.4 Usage

6.4.1 Reset State

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

1. The GPIO configuration enable bits shown in *Table 6-1* are set to I/O (transition to AF1) mode.
2. Input mode is enabled (*GPIOn_INEN*.*[pin]* = 1).
3. High impedance mode enabled (*GPIOn_PADCTRL1*.*[pin]* = 0, *GPIOn_PADCTRL0*.*[pin]* = 0), pullup and pulldown disabled.
4. Output mode disabled (*GPIOn_OUTEN*.*[pin]* = 0)
5. Interrupt disabled (*GPIOn_INTEN*.*[pin]* = 0)

6.4.2 Input Mode Usage

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

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

6.4.3 Output Mode Usage

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

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

6.4.4 Alternate Function Usage

Most GPIOs support one or more alternate functions selected with the GPIO configuration enable bits shown in *Table 6-1*. The bits that select the AF must only be changed while the pin is in one of the I/O modes (*GPIOn_EN0* = 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 6-1* to the I/O mode that corresponds 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 6-2* if the assigned alternate function uses the pin as an input. See *Table 6-3* if the assigned alternate function uses the pin as an output.
3. Set the GPIO configuration enable bits shown in *Table 6-1* to the desired alternate function.

6.5 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 a peripheral alternate 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_INTFL* register. A GPIO interrupt occurs when the status bit transitions from 0 to 1 if the corresponding bit is set in the corresponding *GPIOn_INTEN* register. Note that the interrupt status bit is always set when the current interrupt configuration event occurs, but an interrupt is only 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_INTEN*.*[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_INEN* register. To maintain previously enabled interrupts, read the *GPIOn_INEN* register and save the state before setting the register to 0.
2. Clear pending interrupts by writing 1 to the *GPIOn_INTFL_CLR*.*[pin]* bit.
3. Configure the pin for the desired interrupt event
4. Set *GPIOn_INTMODE*.*[pin]* to select the desired interrupt.
5. For level-triggered interrupts, the interrupt triggers on an input high (*GPIOn_INTPOL*.*[pin]* = 0) or input low level.
6. For edge-triggered interrupts, the interrupt triggers on a transition from low to high(*GPIOn_INTPOL*.*[pin]* = 0) or high to low (*GPIOn_INTPOL*.*[pin]* = 1).
7. Optionally, set *GPIOn_DUALEDGE*.*[pin]* to 1 to trigger on both the rising and falling edges of the input signal.
   a. Set *GPIOn_INTEN*.*[pin]* to 1 to enable the interrupt for the pin.

6.5.1 GPIO Interrupt Handling

Each GPIO port is assigned its own dedicated interrupt vector as shown in *Table 6-8*.

Table 6-8: MAX32690 GPIO Port Interrupt Vector Mapping

GPIO Interrupt Source	GPIO Interrupt Status Register	CM4 Interrupt Vector Number	RV32 Interrupt Vector Number	GPIO Interrupt Vector
GPIO0	*GPIOn_INTFL*	40	25	GPIO0_IRQn
GPIO1	*GPIOn_INTFL*	41	26	GPIO1_IRQn
GPIO2	*GPIOn_INTFL*	42	60	GPIO2_IRQn
GPIO3	*GPIOn_INTFL*	74	27	GPIO3_IRQn

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

1. Read the *GPIOn_INTFL* 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_INTFL* register by writing a 1 to the *GPIOn_INTFL_CLR* bit position that triggered the interrupt. This also clears and rearms the edge detectors for edge-triggered interrupts.
4. Return from the interrupt vector handler.

6.5.2 Using GPIO for Wake-Up from Low-Power Modes

Standard GPIO interrupts wake the device from *SLEEP* and *LPM*. Additionally, wake from *LPM*, *UPM*, *STANDBY*, and *BACKUP* are supported for GPIO using the GPIOWAKE feature. GPIOWAKE allows wake from low-power modes from external 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.

6.5.3 Using GPIOWAKE for Wake-Up from DEEPSLEEP, BACKUP, and STORAGE

For wake-up interrupts on the GPIO, a single interrupt vector, GPIOWAKE_IRQn, is assigned for all the GPIO pins. When the wake-up event occurs, the application software must interrogate the *PWRSEQ_LPWKST0* register to determine which GPIO0 port pin caused the interrupt.

Enable GPIOWAKE interrupts for all power modes (*ACTIVE*, *SLEEP*, *DEEPSLEEP,* and *BACKUP*) from an external GPIO event by completing the following steps:

1. Clear pending interrupt flags by writing 0xFFFF FFFF to the *PWRSEQ_LPWKST0* , *PWRSEQ_LPWKST1* , *PWRSEQ_LPWKST2*, *PWRSEQ_LPWKST3*, and *PWRSEQ_LPWKST4* registers.
2. Set up a GPIOWAKE_IRQn interrupt handler.
3. Enable the GPIOWAKE for each desired GPIO pin by setting the corresponding GPIO port's wake enable registers.
   a. Set *PWRSEQ_LPWKEN0[pin]* to 1 for GPIO0.
   b. Set *PWRSEQ_LPWKEN1[pin]* to 1 for GPIO1.
   c. Set *PWRSEQ_LPWKEN2[pin]* to 1 for GPIO2.
   d. Set *PWRSEQ_LPWKEN3[pin]* to 1 for GPIO3.
   e. Set *PWRSEQ_LPWKEN4[pin]* to 1 for GPIO4.
4. Configure the power manager to use the GPIO as a wake-up source by writing 1 to the *GCR_PM.gpio_we* field.
5. Enter the desired low-power mode. See *Operating Modes* for details.
6. When a wake-up event occurs, if an I/O caused the wake up, the pin's corresponding bit is set in the *PWRSEQ_LPWKST0* for GPIO0, *PWRSEQ_LPWKST1* for GPIO1, for *PWRSEQ_LPWKST2* GPIO2, *PWRSEQ_LPWKST3* for GPIO3, and *PWRSEQ_LPWKST4* for GPIO4.

Table 6-9: MAX32690 GPIO Wake-up Interrupt Vector

GPIO Wake Interrupt Source	GPIO Wake Interrupt Status Register	CM4 Interrupt Vector Number	RV32 Interrupt Vector Number	GPIO Wake-up Interrupt Vector
GPIO0	*PWRSEQ_LPWKST0*	70	6	GPIOWAKE_IRQn
GPIO1	*PWRSEQ_LPWKST1*	70	6	GPIOWAKE_IRQn
GPIO2	*PWRSEQ_LPWKST2*	70	6	GPIOWAKE_IRQn
GPIO3	*PWRSEQ_LPWKST3*	70	6	GPIOWAKE_IRQn
GPIO4	*PWRSEQ_LPWKST4*	70	6	GPIOWAKE_IRQn

6.6 Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 6-10*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 1-1* 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 6-10: GPIO Register Summary

Offset	Register	Description
[0x0000]	*GPIOn_EN0*	*GPIO Port n Configuration Enable Bit 0 Register*
[0x0004]	*GPIOn_EN0_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 0 Register*
[0x0008]	*GPIOn_EN0_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 0 Register*
[0x000C]	*GPIOn_OUTEN*	*GPIO Port n Output Enable Register*
[0x0010]	*GPIOn_OUTEN_SET*	*GPIO Port n Output Enable Atomic Set Register*
[0x0014]	*GPIOn_OUTEN_CLR*	*GPIO Port n Output Enable Atomic Clear Register*
[0x0018]	*GPIOn_OUT*	*GPIO Port n Output Register*
[0x001C]	*GPIOn_OUT_SET*	*GPIO Port n Output Atomic Set Register*
[0x0020]	*GPIOn_OUT_CLR*	*GPIO Port n Output Atomic Clear Register*
[0x0024]	*GPIOn_IN*	*GPIO Port n Input Register*
[0x0028]	*GPIOn_INTMODE*	*GPIO Port n Interrupt Mode Register*
[0x002C]	*GPIOn_INTPOL*	*GPIO Port n Interrupt Polarity Register*
[0x0030]	*GPIOn_INEN*	*GPIO Port n Input Enable Register*
[0x0034]	*GPIOn_INTEN*	*GPIO Port n Interrupt Enable Register*
[0x0038]	*GPIOn_INTEN_SET*	*GPIO Port n Interrupt Enable Atomic Set Register*
[0x003C]	*GPIOn_INTEN_CLR*	*GPIO Port n Interrupt Enable Atomic Clear Register*
[0x0040]	*GPIOn_INTFL*	*GPIO Port n Interrupt Status Register*
[0x0048]	*GPIOn_INTFL_CLR*	*GPIO Port n Interrupt Clear Register*
[0x004C]	*GPIOn_WKEN*	*GPIO Port n Wake-up Enable Register*
[0x0050]	*GPIOn_WKEN_SET*	*GPIO Port n Wake-up Enable Atomic Set Register*
[0x0054]	*GPIOn_WKEN_CLR*	*GPIO Port n Wake-up Enable Atomic Clear Register*
[0x005C]	*GPIOn_DUALEDGE*	*GPIO Port n Interrupt Dual Edge Mode Register*
[0x0060]	*GPIOn_PADCTRL0*	*GPIO Port n Pad Configuration 1 Register*
[0x0064]	*GPIOn_PADCTRL1*	*GPIO Port n Pad Configuration 2 Register*
[0x0068]	*GPIOn_EN1*	*GPIO Port n Configuration Enable Bit 1 Register*
[0x006C]	*GPIOn_EN1_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 1 Register*
[0x0070]	*GPIOn_EN1_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 1 Register*
[0x0074]	*GPIOn_EN2*	*GPIO Port n Configuration Enable Bit 2 Register*
[0x0078]	*GPIOn_EN2_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 2 Register*
[0x007C]	*GPIOn_EN2_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 2 Register*
[0x00A8]	*GPIOn_HYSEN*	*GPIO Port n Hysteresis Enable Register*
[0x00AC]	*GPIOn_SRSEL*	*GPIO Port n Slew Rate Select Register*
[0x00B0]	*GPIOn_DS0*	*GPIO Port n Output Drive Strength Bit 0 Register*
[0x00B4]	*GPIOn_DS1*	*GPIO Port n Output Drive Strength Bit 1 Register*
[0x00B8]	*GPIOn_PS*	*GPIO Port n Pulldown/Pullup Strength Select Register*
[0x00C0]	*GPIOn_VSSEL*	*GPIO Port n Voltage Select Register*

6.6.1 GPIO Register Details

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

GPIO Port n Configuration Enable Bit 0			GPIOn_EN0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	1	**GPIO Configuration Enable Bit 0** These bits, in conjunction with bits in *Table 6-1* configure 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_EN0_SET* or *GPIOn_EN0_CLR*. *Table 6-4* depicts a detailed example of how each of these bits applies to each of the GPIO device pins *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect input and interrupt functionality of the associated pin.*

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

GPIO Port n Configuration Enable Atomic Set Bit 0			GPIOn_EN0_SET	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Configuration Enable Atomic Set Bit 0** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_EN0* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN0* register set to 1.

Table 6-13: GPIO Port n Configuration Enable Atomic Clear Bit 0 Register

GPIO Port n Configuration Enable Atomic Clear Bit 0			GPIOn_EN0_CLR	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Configuration Enable Atomic Clear Bit 0** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_EN0* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN0* register cleared to 0.

Table 6-14: GPIO Port n Output Enable Register

GPIO Port n Output Enable			GPIOn_OUTEN	[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_OUTEN_SET* or *GPIOn_OUTEN_CLR*. 0: Pin is set to input mode; output driver disabled. 1: Pin is set to output mode.

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

GPIO Port n Output Enable Atomic Set			GPIOn_OUTEN_SET	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Output Enable Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_OUTEN* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUTEN* set to 1.

Table 6-16: GPIO Port n Output Enable Atomic Clear Register

GPIO Port n Output Enable Atomic Clear			GPIOn_OUTEN_CLR	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Output Enable Atomic Clear** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_OUTEN* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUTEN* cleared to 0.

Table 6-17: 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 6-18: GPIO Port n Output Atomic Set Register

GPIO Port n Output Atomic Set			GPIOn_OUT_SET	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Output Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_OUT* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUTEN* set to 1.

Table 6-19: 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 clears the corresponding bits in the *GPIOn_OUT* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUTEN* cleared to 0.

Table 6-20: 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_INEN* register is set. The state is not affected by the pin's configuration as an output or alternate function. 0: Input pin low (logic 0). 1: Input pin high (logic 1).

Table 6-21: GPIO Port n Interrupt Mode Register

GPIO Port n Interrupt Mode		GPIOn_INTMODE		[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_INTEN register is set.*

Table 6-22: GPIO Port n Interrupt Polarity Register

GPIO Port n Interrupt Polarity		GPIOn_INTPOL		[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_INTMODE.mode[pin]* = 0): 0: Input low (logic 0) triggers interrupt. 1: Input high (logic 1) triggers interrupt. Edge-triggered mode (*GPIOn_INTMODE.mode[pin]*= 1): 0: Falling edge triggers interrupt. 1: Rising edge triggers interrupt. *Note: This bit has no effect unless the corresponding bit in the GPIOn_INTEN register is set.*

Table 6-23: GPIO Port n Input Enable Register

GPIO Port n Input Enable		GPIOn_INEN		[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 through the GPIOn_IN register.

Table 6-24: GPIO Port n Interrupt Enable Register

GPIO Port n Interrupt Enable		GPIOn_INTEN		[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 interrupt disabled. 1: GPIO interrupt enabled. *Note: Disabling a GPIO interrupt does not clear pending interrupts for the associated pin. Use the GPIOn_INTFL_CLR register to clear pending interrupts.*

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

GPIO Port Interrupt Enable Atomic Set		GPIOn_INTEN_SET		[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Interrupt Enable Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the GPIOn_INTEN register. 0: No effect. 1: Corresponding bits in GPIOn_INTEN register set to 1.

Table 6-26: GPIO Port n Interrupt Enable Atomic Clear Register

GPIO Port Interrupt Enable Atomic Clear			GPIOn_INTEN_CLR	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Interrupt Enable Atomic Clear** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_INTEN* register. 0: No effect. 1: Corresponding bits in *GPIOn_INTEN* register cleared to 0.

Table 6-27: GPIO Port n Interrupt Status Register

GPIO Port Interrupt Status			GPIOn_INTFL	[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 interrupt pending for associated GPIO pin. *Note: Write a 1 to the corresponding bit in the GPIOn_INTFL_CLR register to clear the interrupt pending status flag.*

Table 6-28: GPIO Port n Interrupt Clear Register

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

Table 6-29: GPIO Port n Wake-up Enable Register

GPIO Port n Wake-up Enable			GPIOn_WKEN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 6-30: GPIO Port n Wake-up Enable Atomic Set Register

GPIO Port Wake-up Enable Atomic Set			GPIOn_WKEN_SET	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 6-31: GPIO Port n Wake-up Enable Atomic Clear Register

GPIO Port Wake-up Enable Atomic Clear			GPIOn_WKEN_CLR	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 6-32: GPIO Port n Interrupt Dual Edge Mode Register

GPIO Port n Interrupt Dual Edge Mode			GPIOn_DUALEDGE	[0x005C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO 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_INTMODE* bit is set to edge-triggered. The associated polarity (*GPIOn_INTPOL*) setting has no effect when this bit is set. 0: Disabled 1: Enabled

Table 6-33: GPIO Port n Pad Configuration 1 Register

GPIO Port n Pad Configuration 1			GPIOn_PADCTRL0	[0x0060]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Pad Configuration 1** Input mode configuration for the associated GPIO pin. Input mode selection and the selection of a weak or strong pullup or weak or strong pulldown resistor are described in *Table 6-2*.

Table 6-34: GPIO Port n Pad Configuration 2 Register

GPIO Port n Pad Configuration 2			GPIOn_PADCTRL1	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Pad Configuration 2** Input mode configuration for the associated GPIO pin. Input mode selection and the selection of a weak or strong pullup or weak or strong pulldown resistor are described in *Table 6-2*.

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

GPIO Port n Configuration Enable Bit 1			GPIOn_EN1	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Configuration Enable Bit 1** These bits, in conjunction with bits in *Table 6-1* configure 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_EN1_SET* or *GPIOn_EN1_CLR*. *Table 6-4* depicts a detailed example of how each of these bits applies to each of the GPIO device pins *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect input and interrupt functionality of the associated pin.*

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

GPIO Port n Configuration Enable Atomic Set Bit 1			GPIOn_EN1_SET	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Configuration Enable Atomic Set Bit 1** 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 6-37: GPIO Port n Configuration Enable Atomic Clear Bit 1 Register

GPIO Port n Configuration Enable Atomic Clear Bit 1			GPIOn_EN1_CLR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Configuration Enable Atomic Clear Bit 1** 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 6-38: GPIO Port n Configuration Enable Bit 2 Register

GPIO Port n Configuration Enable Bit 2			GPIOn_EN2	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Configuration Enable Bit 2** These bits, in conjunction with bits in *Table 6-1* configure 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_EN2_SET* or *GPIOn_EN2_CLR*. *Table 6-4* depicts a detailed example of how each of these bits applies to each of the GPIO device pins *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect input and interrupt functionality of the associated pin.*

Table 6-39: GPIO Port n Configuration Enable Atomic Set Bit 2 Register

GPIO Port n Configuration Enable Atomic Set Bit 2			GPIOn_EN2_SET	[0x0078]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Alternate Function Select Atomic Set Bit 2** 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 6-40: GPIO Port n Configuration Enable Atomic Clear Bit 2 Register

GPIO Port n Configuration Enable Atomic Clear Bit 2			GPIOn_EN2_CLR	[0x007C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Alternate Function Select Atomic Clear Bit 2** 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 6-41: GPIO Port n Hysteresis Enable Register

GPIO Port n Hysteresis Enable			GPIOn_HYSEN	[0x00A8]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 6-42: GPIO Port n Output Drive Strength Bit 0 Register

GPIO Port n Output Drive Strength Bit 0			GPIOn_SRSEL	[0x00AC]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**Reserved**

Table 6-43: GPIO Port n Output Drive Strength Bit 0 Register

GPIO Port n Output Drive Strength Bit 0			GPIOn_DS0	[0x00B0]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Output Drive Strength Selection 0** See *Table 6-3* for details on how to set the GPIO output drive strength and other electrical characteristics.

Table 6-44: GPIO Port n Output Drive Strength Bit 1 Register

GPIO Port n Output Drive Strength Bit 1			GPIOn_DS1	[0x00B4]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Output Drive Strength Selection 1** See *Table 6-3* for details on how to set the GPIO output drive strength and other electrical characteristics.

Table 6-45: GPIO Port n Pulldown/Pullup Strength Select Register

GPIO Port n 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 data sheet for specific electrical characteristics of the pulldown/pullup resistances.*

Table 6-46: GPIO Port n 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** Selects the voltage rail used for the pin. 0: V_{DDIO} 1: V_{DDIOH} *Note: GPIO that support HyperBus as an alternate function are tied to the V_{DDIO} supply and do not support V_{DDIOH} selection. Refer to the device data sheet's alternate function table to determine pins that support HyperBus.*

7. Debug Access Port (DAP)

The device provides an Arm DAP that supports debugging during application development. The DAP enables an external debugger to access the device. The DAP is a standard Arm CoreSight™ serial wire debug port and uses a two-pin serial interface (SWDCLK and SWDIO) to communicate.

7.1 Instances

The DAP interface communicates through the Serial Wire Debug (SWD) interface signals shown in *Table 7-1*.

Table 7-1: MAX32690 DAP Instances

Instance	Pin	Alternate Function	SWD Signal
0	P0.28	AF1	SWDIO
	P0.29	AF1	SWDCLK

7.2 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_SYSCTRL*.*swd_dis* field to 1. The *GCR_SYSCTRL*.*swd_dis* field clears to 0 again, re-enabling the DAP after a POR. Parts with a customer-accessible DAP should disable the DAP in a final customer product.

7.2.1 Locking the DAP

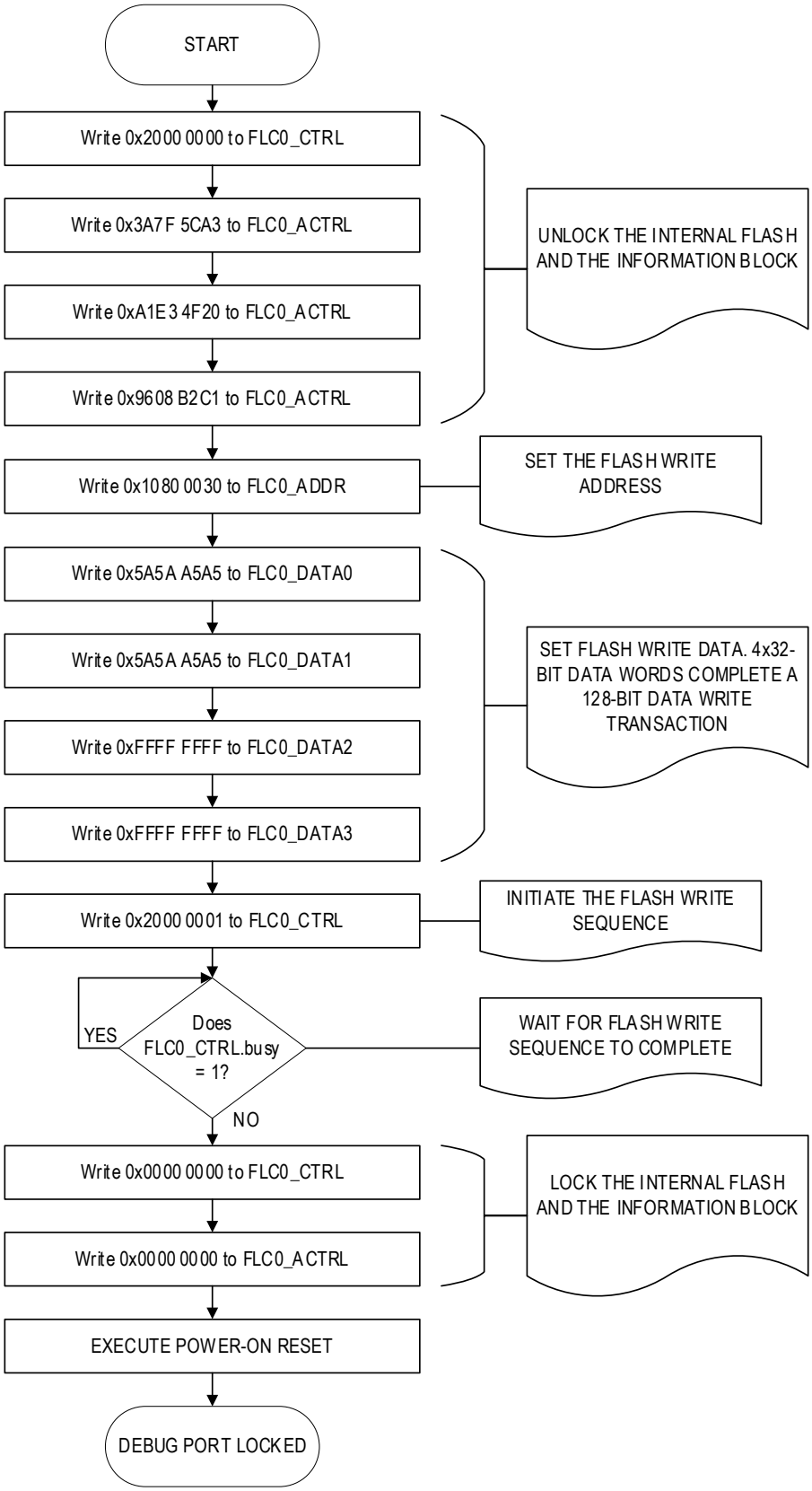
There are two options for locking out the debug access port. Option 1 locks the DAP and makes it available to be unlocked later. This is a one-time-only process. The DAP port cannot be relocked. Option 2 locks the DAP permanently.

7.2.1.1 Option 1

To lock the DAP and make it available to be unlocked later (one time only), follow the flow chart in *Figure 7-1*.

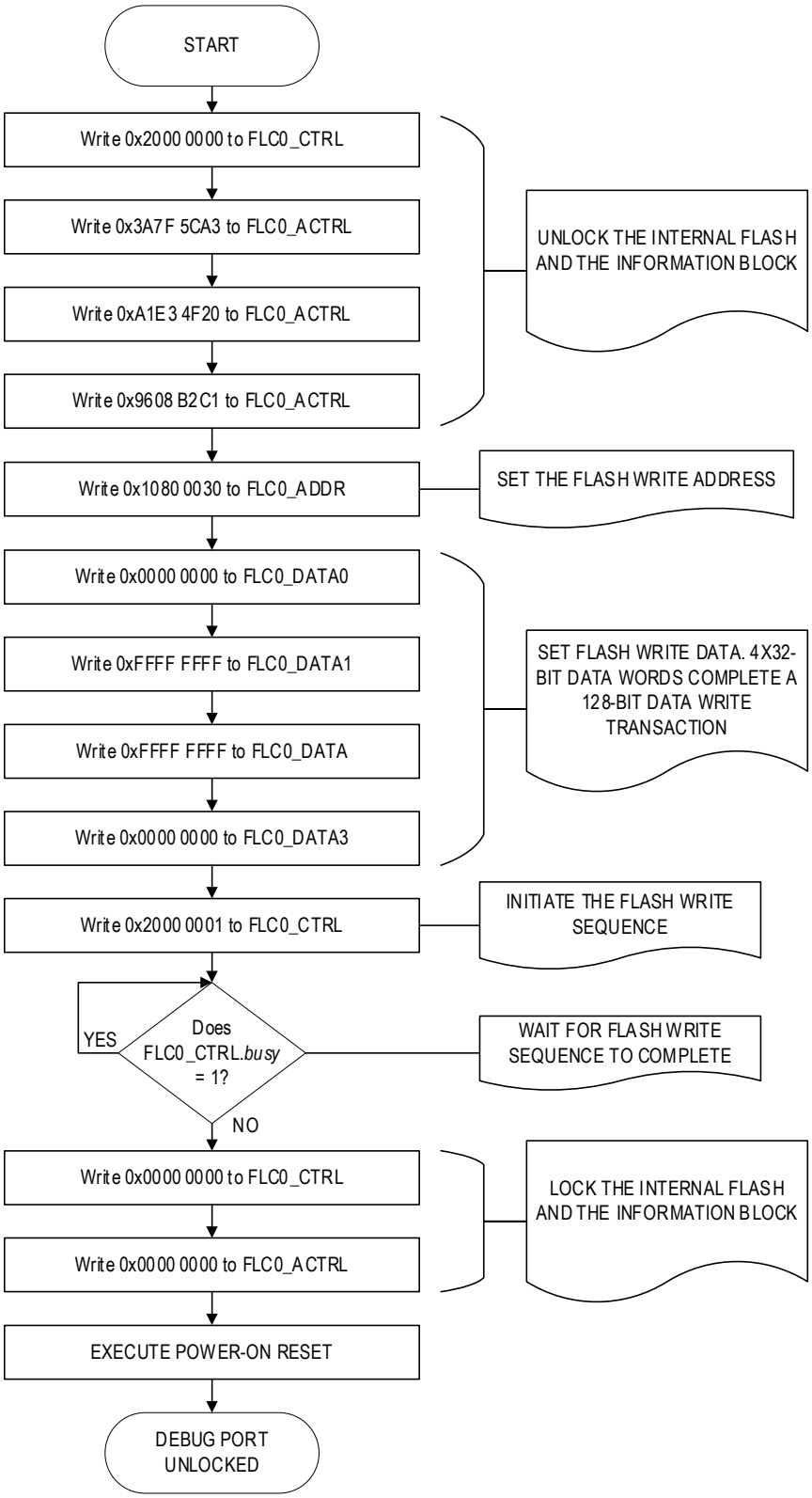
CoreSight is a registered trademark or Arm Limited.

Figure 7-1: 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 7-1*, follow the flow chart in *Figure 7-2*.

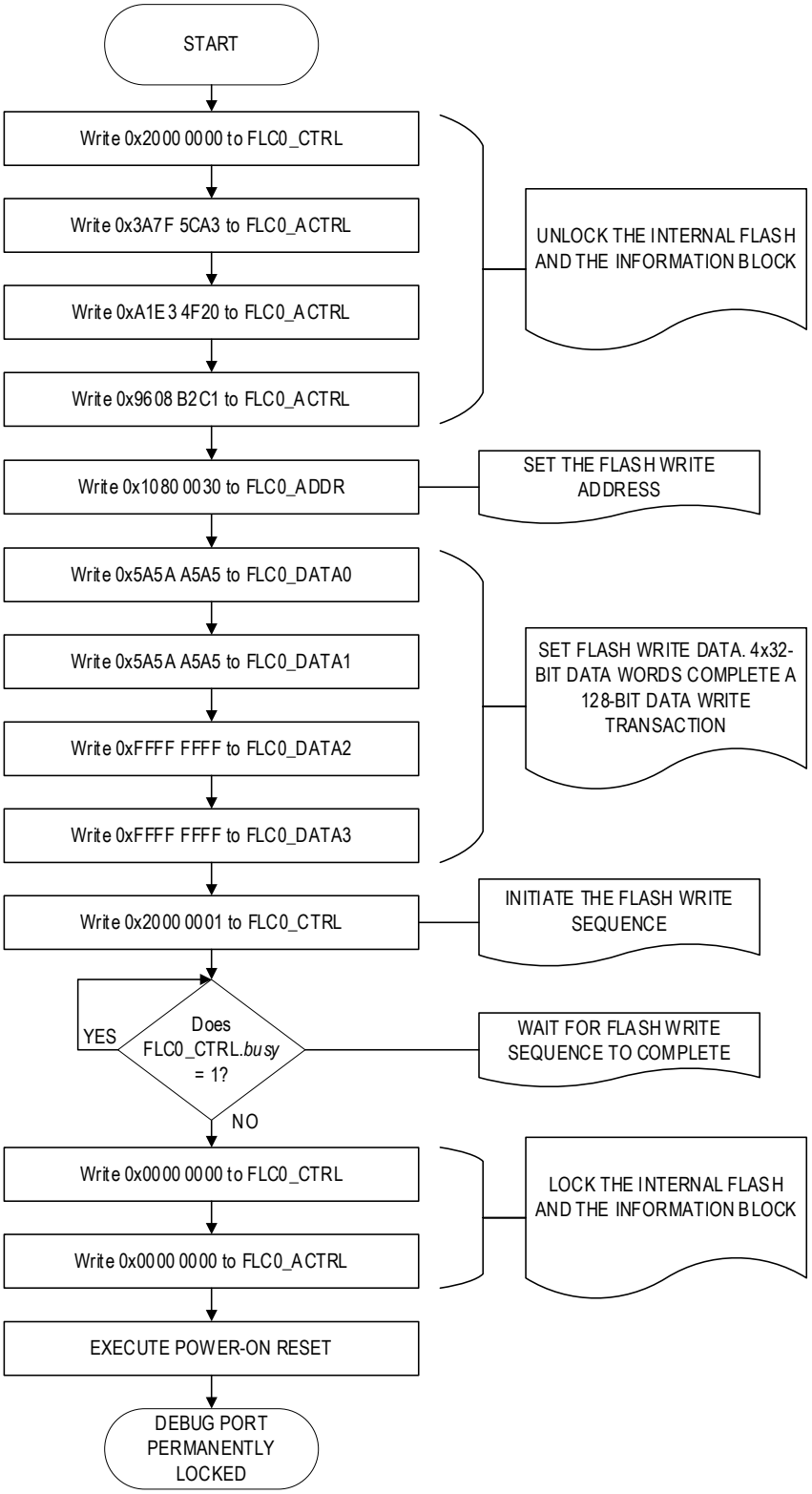
Figure 7-2: Unlocking the DAP After Being Locked as in Figure 7-1



7.2.1.2 Option 2

To lock the DAP permanently, follow the flow chart in *Figure 7-3*.

Figure 7-3: Locking the Debug Access Port Permanently



7.3 Pin Configuration

Instances of SWD or JTAG signals in the GPIO and Alternate Function matrices determine which GPIO pins are associated with a signal. It is unnecessary to configure a pin for an alternate function to use the DAP following a POR.

By default, the pin associated with the bidirectional SWDIO/TMS signal is configured as a GPIO with high-impedance input after a POR. While the DAP is in use, a pullup resistor should be connected to the SWDIO/TMS pin as shown in *Table 7-1*. The pullup ensures the signal is in a known state when control of the SWDIO/TMS pin is transferred between the host and target. The pullup resistor should be removed if the associated pin is used as a GPIO to avoid unnecessary current consumption.

8. Flash Controller (FLC)

The MAX32690 flash controllers manage read, write, and erase access to the internal flash memories and provide the following features:

- Up to 3MB internal flash memory for the CM4
  - 192 pages
  - 16,384 bytes per page
  - 4,096 words by 128 bits per page
- Up to 256KB internal flash memory for the RV32
  - 64 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

8.1 Instances

The device includes two instances of the flash controller. The 3MB of flash bank 0 and the 256KB of flash bank 1 are programmable through serial wire debug interface (in-system) or directly with software (in-application).

Table 8-1 shows the page start address and page end address of each of the internal flash memories.

Table 8-1: MAX32690 Internal Flash Memory Organization

Instance Number	Page Number	Size (per page)	Start Address	End Address
FLC0	1	16,384 Bytes	0x1000 0000	0x1000 3FFF
	2	16,384 Bytes	0x1000 4000	0x1000 7FFF
	3	16,384 Bytes	0x1000 8000	0x1000 BFFF
	4	16,384 Bytes	0x1000 C000	0x1000 FFFF
	…	…	…	…
	191	16,384 Bytes	0x102F 8000	0x102F BFFF
	192	16,384 Bytes	0x102F C000	0x102F FFFF
FLC1	1	8,192 Bytes	0x1030 0000	0x1030 1FFF
	2	8,192 Bytes	0x1030 2000	0x1030 3FFF
	3	8,192 Bytes	0x1030 4000	0x1030 5FFF
	4	8,192 Bytes	0x1030 6000	0x1030 7FFF
	…	…	…	…
	31	8,192 Bytes	0x1033 C000	0x1033 DFFF
	32	8,192 Bytes	0x1033 E000	0x1033 FFFF

8.2 Usage

The 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.

8.2.1 Clock Configuration

The flash controller requires a 1MHz internal clock. See *Oscillator Sources* for details. Use the flash controller clock divisor to generate $f_{FLCn\_CLK} = 1\text{MHz}$, as shown in *Equation 8-1*. If using the IPO as the system clock, the *FLCn_CLKDIV*.*clkdiv* should be set to 120 (0x78).

Equation 8-1: Flash Controller Clock Frequency

$$f_{FLCn\_CLK} = \frac{f_{SYS\_CLK}}{FLCn\_CLKDIV.clkdiv} = 1MHz$$

8.2.2 Lock Protection

A locking mechanism prevents accidental memory writes, and erases. All writes and erase operations require the *FLCn_CTRL.unlock* field be set to 2 before starting the operation. Writing any other value to this field, *FLCn_CTRL.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 is not set to 2, the flash controller hardware sets the access fail flag, FLCn_INTR.af, to indicate an access violation occurred.

8.2.3 Flash Write Width

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

8.2.4 Flash Write

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

1. If desired, enable the flash controller interrupts by setting the *FLCn_INTR.afie* and *FLCn_INTR.doneie* bits.
2. Read the *FLCn_CTRL.pend* bit until it returns 0.
3. Configure the *FLCn_CLKDIV.clkdiv* field to achieve a 1MHz frequency based on the selected SYS_CLK frequency.
4. Set the *FLCn_ADDR* register to a valid target address.
5. Set *FLCn_DATA[3]*, *FLCn_DATA[2]*, *FLCn_DATA[1]*, and *FLCn_DATA[0]* to the data to write.
   a. *FLCn_DATA[3]* is the most significant word and *FLCn_DATA[0]* is the least significant word.
      i. 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 the *FLCn_CTRL.unlock* field to 2 to unlock the flash.
7. Set the *FLCn_CTRL.wr* field to 1.
   a. The hardware automatically clears this field when the write operation is complete.
8. The *FLCn_INTR.done* field is set to 1 by the hardware when the write completes and an interrupt is generated if the *FLCn_INTR.doneie* is set to 1.
   a. If an error occurred, the *FLCn_INTR.af* field is set to 1 by the hardware and an interrupt is generated if the *FLCn_INTR.afie* field is set to 1.
9. Set the *FLCn_CTRL.unlock* field to any value other than 2 to relock the flash.

Note: Code execution can occur within the same flash instance as targeted programming.

Note: If the ICC is enabled, either disable it before writing to the flash or flush it after writing to the flash.

8.2.5 Page Erase

CAUTION: Care must be taken to not erase the page from which the software 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.afie* and *FLCn_INTR.doneie bits*.
2. Read the *FLCn_CTRL.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[14:0]* is ignored for FLC0 and *FLCn_ADDR[12:0]* is ignored for FLC1 to ensure the address is page aligned.
5. Set *FLCn_CTRL.unlock* to 2 to unlock the flash instance.
6. Set *FLCn_CTRL.erase_code* to 0x55 for page erase.
7. Set *FLCn_CTRL.pge* to 1 to start the page erase operation.
8. The *FLCn_CTRL.pend* bit is set by the flash controller while the page erase is in progress, and the *FLCn_CTRL.pge* and *FLCn_CTRL*.pend are cleared by the flash controller when the page erase is complete.
9. *FLCn_INTR.done* is set by the hardware when the page erase completes and if an error occurred, the *FLCn_INTR.af* flag is set. These bits generate a flash interrupt if the interrupt enable bits are set.
10. Set *FLCn_CTRL.unlock* to any value other than 2 to relock the flash instance.

8.2.6 Mass Erase

CAUTION: Care must be taken to not erase the flash from which the software 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_CTRL.pend* bit until it returns 0.
2. Configure *FLCn_CLKDIV.clkdiv* to match the SYS_CLK frequency.
3. Set *FLCn_CTRL.unlock* to 2 to unlock the internal flash.
4. Set *FLCn_CTRL.erase_code* to 0xAA for mass erase.
5. Set *FLCn_CTRL.me* to 1 to start the mass erase operation.
6. The *FLCn_CTRL.pend* bit is set by the flash controller while the mass erase is in progress, and the *FLCn_CTRL.me* and *FLCn_CTRL.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 interrupt if the interrupt enable bits are set.
8. Set *FLCn_CTRL.unlock* to any value other than 2 to relock the flash instance.

8.3 Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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.

Note: The FLC registers are reset only on a POR. System reset, soft reset, and peripheral reset do not affect the FLC register values.

Table 8-2: Flash Controller Register Summary

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_CTRL*	R/W	Flash Controller Control Register
[0x0024]	*FLCn_INTR*	R/W	Flash Controller Interrupt Register
[0x0030]	*FLCn_DATA[0]*	R/W	Flash Controller Data Register 0
[0x0034]	*FLCn_DATA[1]*	R/W	Flash Controller Data Register 1

Offset	Register Name	Access	Description
[0x0038]	*FLCn_DATA[2]*	R/W	Flash Controller Data Register 2
[0x003C]	*FLCn_DATA[3]*	R/W	Flash Controller Data Register 3
[0x0040]	*FLCn_ACTRL*	R/W	Flash Controller Access Control Register
[0x0080]	*FLCn_WELR0*	R/W	Flash Write/Erase Lock 0 Register
[0x0084]	*FLCn_RLR0*	R/W	Flash Read Lock 0 Register
[0x0088]	*FLCn_WELR1*	R/W	Flash Write/Erase Lock 1 Register
[0x008C]	*FLCn_RLR1*	R/W	Flash Read Lock 1 Register
[0x0090]	*FLCn_WELR2*	R/W	Flash Write/Erase Lock 2 Register (FLC0 only)
[0x0094]	*FLCn_RLR2*	R/W	Flash Read Lock 2 Register (FLC0 only)
[0x0098]	*FLCn_WELR3*	R/W	Flash Write/Erase Lock 3 Register (FLC0 only)
[0x009C]	*FLCn_RLR3*	R/W	Flash Read Lock 3 Register (FLC0 only)
[0x00A0]	*FLCn_WELR4*	R/W	Flash Write/Erase Lock 4 Register (FLC0 only)
[0x00A4]	*FLCn_RLR4*	R/W	Flash Read Lock 4 Register (FLC0 only)
[0x00A8]	*FLCn_WELR5*	R/W	Flash Write/Erase Lock 5 Register (FLC0 only)
[0x00AC]	*FLCn_RLR5*	R/W	Flash Read Lock 5 Register (FLC0 only)

8.3.1 Register Details

Table 8-3: Flash Controller Address Pointer Register

Flash Controller Address Pointer			FLCn_ADDR	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0x1000 0000	**Flash Address** This field contains the target address for a write operation. A valid internal flash memory address is required for all write operations.

Table 8-4: Flash Controller Clock Divisor Register

Flash Controller Clock Divisor			FLCn_CLKDIV	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	clkdiv	R/W	0x78	**Flash Controller Clock Divisor** The APB clock is divided by the value in this field to generate the flash controller peripheral clock, f_{FLC_CLK}. The flash controller peripheral clock must equal 1MHz. The default on POR, system reset, and watchdog reset is 120, resulting in f_{FLC_CLK} = 1MHz when IPO is the system oscillator. The flash controller peripheral clock is only used during erase and program functions, and not during read functions. See *Clock Configuration* for additional details.

Table 8-5: Flash Controller Control Register

Flash Controller Control			FLCn_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:28	unlock	R/W	0	**Flash Unlock** Write the unlock code, 2, before any flash write or erase operation to unlock the flash. Writing any other value to this field locks the internal flash. 2: Flash unlock code.
27:26	-	RO	-	**Reserved**
25	lve	R/W	0	**Low Voltage Enable** Set this field to 1 to enable low voltage operation for the flash memory. 0: Low voltage operation disabled. 1: Low voltage operation enabled.

Flash Controller Control			FLCn_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
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. This bit is cleared by hardware once the flash is accessible. *Note: If the flash controller is busy (FLCn_CTRL.pend = 1), reads, writes, and erase operations are not allowed and result in an access failure (FLCn_INTR.af = 1).* 0: Flash idle. 1: Flash busy.
23:16	-	RO	0	**Reserved**
15:8	erase_code	R/W	0	**Erase Code** Before 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 before 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.
7:5	-	RO	0	**Reserved**
4	width	R/W	0	**Width** Do not modify this field.
3	-	RO	0	**Reserved**
2	pge	R/W1	0	**Page Erase** Write a 1 to this field to initiate a page erase at the address in *FLCn_ADDR.addr*. Unlock the flash before attempting a page erase. See *FLCn_CTRL.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.
1	me	R/W1	0	**Mass Erase** Write a 1 to this field to initiate a mass erase of the internal flash memory. Unlock the flash before attempting a mass erase. See *FLCn_CTRL.unlock* for details. The flash controller hardware clears this bit when the mass erase operation completes. 0: No operation. 1: Initiate mass erase.
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 writes 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 8-6: Flash Controller Interrupt Register

Flash Controller Interrupt			FLCn_INTR	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
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

Flash Controller Interrupt			FLCn_INTR	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
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	0	**Reserved**
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 the 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 occurred. 1: Access failure occurred.
0	done	R/W0C	0	**Flash Operation Complete Interrupt Flag** This flag is automatically set by the hardware after a flash write or erase operation completes. 0: Operation not complete or not in process. 1: Flash operation complete.

Table 8-7: Flash Controller Data 0 Register

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

Table 8-8: Flash Controller Data Register 1

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

Table 8-9: Flash Controller Data Register 2

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

Table 8-10: Flash Controller Data Register 3

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

Table 8-11: Flash Controller Access Control Register

Flash Controller Access Control			FLCn_ACTRL	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	actrl	R/W	0	**Access Control** Access the information block by writing the access control sequence. See *Information Block Flash Memory* for details.

Table 8-12: Flash Write/Lock 0 Register

Flash Write/Lock 0			FLCn_WELR0	[0x0080]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	welr0	R/W1C	0xFFFF FFFF	**Flash Write/Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 0 of the flash and bit 31 maps to page 31. Write a 1 to a bit position in this register and the corresponding page of flash is immediately locked. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding page of flash is write protected. 1: The corresponding page of flash is *not* write protected.

Table 8-13: Flash Read Lock 0 Register

Flash Read Lock 0			FLCn_RLR0	[0x0084]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	rlr0	R/W1C	0xFFFF FFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 0 of the flash and bit 31 maps to page 31. Write a 1 to a bit position in this register and the corresponding page of flash is immediately read protected. The page's read protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is read protected. 1: The corresponding flash page is *not* read protected.

Table 8-14: Flash Write/Lock 1 Register

Flash Write/Lock 1			FLCn_WELR1	[0x0088]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	welr1	R/W1C	0xFFFF FFFF	**Flash Write/Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 32 of the flash and bit 31 maps to page 63. Write a 1 to a bit position in this register and the corresponding page of flash is immediately locked. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is write protected. 1: The corresponding flash page is *not* write protected.

Table 8-15: Flash Read Lock 1 Register

Flash Read Lock 1			FLCn_RLR1	[0x008C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	rlr1	R/W1C	0xFFFF FFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 32 of the flash and bit 31 maps to page 63. Write a 1 to a bit position in this register and the corresponding page of flash is immediately read protected. The page's read protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is read protected. 1: The corresponding flash page is *not* read protected.

Table 8-16: Flash Write/Lock 2 Register

Flash Write/Lock 2			FLCn_WELR2	[0x0090]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	welr2	R/W1C	0xFFFF FFFF	**Flash Write/Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 64 of the flash and bit 31 maps to page 95. Write a 1 to a bit position in this register and the corresponding page of flash is immediately locked. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is write protected. 1: The corresponding flash page is *not* write protected. *Note: This register is only available for FLC0.*

Table 8-17: Flash Read Lock 2 Register

Flash Read Lock 2			FLCn_RLR2	[0x0094]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	rlr2	R/W1C	0xFFFF FFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 64 of the flash and bit 31 maps to page 95. Write a 1 to a bit position in this register and the corresponding page of flash is immediately read protected. The page's read protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is read protected. 1: The corresponding flash page is *not* read protected. *Note: This register is only available for FLC0.*

Table 8-18: Flash Write/Lock 3 Register

Flash Write/Lock 3			FLCn_WELR3	[0x0098]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	welr3	R/W1C	0xFFFF FFFF	**Flash Write/Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 96 of the flash and bit 31 maps to page 127. Write a 1 to a bit position in this register and the corresponding page of flash is immediately locked. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is write protected. 1: The corresponding flash page is *not* write protected. *Note: This register is only available for FLC0.*

Table 8-19: Flash Read Lock 3 Register

Flash Read Lock 3			FLCn_RLR3	[0x009C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	rlr3	R/W1C	0xFFFF FFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 96 of the flash and bit 31 maps to page 127. Write a 1 to a bit position in this register and the corresponding page of flash is immediately read protected. The page's read protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is read protected. 1: The corresponding flash page is *not* read protected. *Note: This register is only available for FLC0.*

Table 8-20: Flash Write/Lock 4 Register

Flash Write/Lock 4			FLCn_WELR4	[0x00A0]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	welr4	R/W1C	0xFFFF FFFF	**Flash Write/Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 128 of the flash and bit 31 maps to page 159. Write a 1 to a bit position in this register and the corresponding page of flash is immediately locked. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is write protected. 1: The corresponding flash page is *not* write protected. *Note: This register is only available for FLC0.*

Table 8-21: Flash Read Lock 4 Register

Flash Read Lock 4			FLCn_RLR4	[0x00A4]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	rlr4	R/W1C	0xFFFF FFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 128 of the flash and bit 31 maps to page 159. Write a 1 to a bit position in this register and the corresponding page of flash is immediately read protected. The page's read protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is read protected. 1: The corresponding flash page is *not* read protected. *Note: This register is only available for FLC0.*

Table 8-22: Flash Write/Lock 5 Register

Flash Write/Lock 5			FLCn_WELR5	[0x00A8]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	welr5	R/W1C	0xFFFF FFFF	**Flash Write/Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 160 of the flash and bit 31 maps to page 191. Write a 1 to a bit position in this register and the corresponding page of flash is immediately locked. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is write protected. 1: The corresponding flash page is *not* write protected. *Note: This register is only available for FLC0.*

Table 8-23: Flash Read Lock 5 Register

Flash Read Lock 5			FLCn_RLR5	[0x00AC]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	rlr5	R/W1C	0xFFFF FFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 160 of the flash and bit 31 maps to page 191. Write a 1 to a bit position in this register and the corresponding page of flash is immediately read protected. The page's read protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is read protected. 1: The corresponding flash page is *not* read protected. *Note: This register is only available for FLC0.*

9. 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, automatically sets the status to 1. The semaphore status register reflects the state of each of the semaphore register's status. The status register enables checking each of the semaphore's states, but is read only, it is not guaranteed that the semaphore status fields does not 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.

The semaphore peripheral includes two general purpose mailbox registers that enable communication between the RV32 and CM4 cores. Additionally, either core can generate a semaphore interrupt for either the CM4 or the RV32, providing immediate notification of communication through the mailbox registers.

9.1 Instances

There is one instance of the semaphore peripheral, shown in *Table 9-1*.

Table 9-1: MAX32690 Semaphore Instances

Instance	Number of Semaphores
SEMA	8

9.2 Peripheral Clock Enable

The semaphore peripheral is disabled by default on a reset. Use of the semaphore peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS1.smphr* to 0 to enable the peripheral clock to the semaphore before configuring the semaphore for use.

9.3 Multiprocessor Communications

The semaphore includes support for multicore communications through two mailbox registers and provides the ability to generate an RV32 semaphore interrupt and a CM4 semaphore interrupt.

The mailbox registers, *SEMA_MAIL0* and *SEMA_MAIL1*, are general purpose 32-bit registers. The CM4 and RV32 have read and write access to both registers. Application firmware should manage how these registers are used to prevent collisions if both cores attempt to modify the registers at the same time.

9.3.1 Reset

Globally reset the semaphore peripheral by setting *GCR_RST1.smphr* to 1.

9.3.2 CM4 Semaphore Interrupt Generation

The *SEMA_IRQ0* register can generate a CM4 semaphore interrupt. Setting the *SEMA_IRQ0.cm4_irq* bit to 1 and then setting the *SEMA_IRQ0.en* bit to 1 generates a CM4 semaphore interrupt. The CM4 interrupt handler should write the *SEMA_IRQ0.en and/or the SEMA_IRQ0.cm4_irq* field(s) to 0 to clear the interrupt condition.

9.3.3 RV32 Semaphore Interrupt Generation

The *SEMA_IRQ1* register can generate a RV32 semaphore interrupt. Setting the *SEMA_IRQ1.rv32_irq* bit to 1 and then setting the *SEMA_IRQ1.en* bit to 1 generates a RV32 semaphore interrupt. The RV32 interrupt handler should write the *SEMA_IRQ1.en and/or the SEMA_IRQ1.rv32_irq* field(s) to 0 to clear the interrupt condition.

9.4 Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 9-2: Semaphore Register Summary

Offset	Register	Name
[0x0000]	*SEMA_SEMAPHORES[0]*	Semaphore 0 Register
[0x0004]	*SEMA_SEMAPHORES[1]*	Semaphore 1 Register
[0x0008]	*SEMA_SEMAPHORES[2]*	Semaphore 2 Register
[0x000C]	*SEMA_SEMAPHORES[3]*	Semaphore 3 Register
[0x0010]	*SEMA_SEMAPHORES[4]*	Semaphore 4 Register
[0x0014]	*SEMA_SEMAPHORES[5]*	Semaphore 5 Register
[0x0018]	*SEMA_SEMAPHORES[6]*	Semaphore 6 Register
[0x0020]	*SEMA_SEMAPHORES[7]*	Semaphore 7 Register
[0x0040]	*SEMA_IRQ0*	Semaphore Interrupt 0 Register
[0x0044]	*SEMA_MAIL0*	Semaphore Mailbox 0 Register
[0x0048]	*SEMA_IRQ1*	Semaphore Interrupt 1 Register
[0x004C]	*SEMA_MAIL1*	Semaphore Mailbox 1 Register
[0x0100]	*SEMA_STATUS*	Semaphore Status Register

9.4.1 Register Details

Table 9-3: Semaphore 0 Register

Semaphore 0			**SEMA_SEMAPHORES[0]**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	sema	*	0	**Semaphore Status** Reading this field returns its current value and if 0, 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 9-4: Semaphore 1 Register

Semaphore 1			**SEMA_SEMAPHORES[1]**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	sema	*	0	**Semaphore Status** Reading this field returns its current value and if 0, 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 9-5: Semaphore 2 Register

Semaphore 2			**SEMA_SEMAPHORES[2]**	**[0x0008]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**

Semaphore 2			SEMA_SEMAPHORES[2]	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
0	sema	*	0	**Semaphore Status** Reading this field returns its current value and if 0, 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 9-6: Semaphore 3 Register

Semaphore 3			SEMA_SEMAPHORES[3]	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	sema	*	0	**Semaphore Status** Reading this field returns its current value and if 0, 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 9-7: Semaphore 4 Register

Semaphore 4			SEMA_SEMAPHORES[4]	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	sema	*	0	**Semaphore Status** Reading this field returns its current value and if 0, 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 9-8: Semaphore 5 Register

Semaphore 5			SEMA_SEMAPHORES[5]	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	sema	*	0	**Semaphore Status** Reading this field returns its current value and if 0, 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 9-9: Semaphore 6 Register

Semaphore 6			SEMA_SEMAPHORES[6]	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	sema	*	0	**Semaphore Status** Reading this field returns its current value and if 0, 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 9-10: Semaphore 7 Register

Semaphore 7			SEMA_SEMAPHORES[7]	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	sema	*	0	**Semaphore Status** Reading this field returns its current value and if 0, 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 9-11: Semaphore Interrupt 0 Register

Semaphore Interrupt 0			SEMA_IRQ0	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	cm4_irq	R/W	0	**CM4 Interrupt** The RV32 can use this bit to communicate with the CM4 through the semaphore interrupt. The RV32 generates a semaphore interrupt for the CM4 by setting this field to 1 and also setting the *SEMA_IRQ0.en* bit to 1.
15:1	-	RO	0	**Reserved**
0	en	R/W	0	**Interrupt Enable** Set this field to enable interrupt generation on semaphore events. 0: Disabled. 1: Enabled.

Table 9-12: Semaphore Mailbox 0 Register

Semaphore Mailbox 0			SEMA_MAIL0	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Data** This register is readable and writable by both the CM4 and RV32 cores, allowing communication between the two cores. In conjunction with the *SEMA_IRQ0* register, the RV32 can write data to this register and then notify the CM4 by generating a semaphore interrupt. Alternately, the CM4 can write to this register and then notify the RV32 using the *SEMA_IRQ1* register to generate an RV32 semaphore interrupt event. *Note: The management of the SEMA_MAIL0 and SEMA_MAIL1 registers is left to the software. It is recommended that one mailbox is used for communication from the CM4 to the RV32 and the other mailbox register is used for communication from the RV32 to the CM4. However, there are no hardware read/write restrictions on the mailbox registers.*

Table 9-13: Semaphore Interrupt 1 Register

Semaphore Interrupt 1			SEMA_IRQ1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	rv32_irq	R/W	0	**RV32 Interrupt** The CM4 can use this bit to communicate with the RV32 through the semaphore interrupt. The CM4 generates a semaphore interrupt for the RV32 by setting this field to 1 and also setting the *SEMA_IRQ1.en* bit to 1. 0: RV32 interrupt event not active or received by RV32. 1: RV32 interrupt event is generated when the *SEMA_IRQ1.en* bit is also set to 1.
15:1	-	RO	0	**Reserved**

Semaphore Interrupt 1			SEMA_IRQ1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
0	en	R/W	0	**Interrupt Enable** Set this field to generate a RV32 semaphore interrupt when the *SEMA_IRQ1.rv32_irq* is also set to 1. The RV32 should write this bit to 0 when a semaphore interrupt is generated to prevent repeat interrupt generation. 0: Disabled. 1: Enabled.

Table 9-14: Semaphore Mailbox 1 Register

Semaphore Mailbox 1			SEMA_MAIL1	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Data** This register is readable and writable by both the CM4 and RV32 cores, allowing communication between the two cores. In conjunction with the *SEMA_IRQ0* register, the RV32 can write data to this register and then notify the CM4 by generating a semaphore interrupt. Alternately, the CM4 can write to this register and then notify the RV32 using the *SEMA_IRQ1* register to generate an RV32 semaphore interrupt event. *Note: The management of the SEMA_MAIL0 and SEMA_MAIL1 registers is left to the software. It is recommended that one mailbox is used for communication from the CM4 to the RV32 and the other mailbox register is used for communication from the RV32 to the CM4. However, there are no hardware read/write restrictions on the mailbox registers.*

Table 9-15: Semaphore Status Register

Semaphore Status			SEMA_STATUS	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
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_SEMAPHORES[7].sema* is 0. 1: *SEMA_SEMAPHORES[7].sema* 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_SEMAPHORES[6].sema* is 0. 1: *SEMA_SEMAPHORES[6].sema* 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_SEMAPHORES[5].sema* is 0. 1: *SEMA_SEMAPHORES[5].sema* 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_SEMAPHORES[4].sema* is 0. 1: *SEMA_SEMAPHORES[4].sema* 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_SEMAPHORES[3].sema* is 0. 1: *SEMA_SEMAPHORES[3].sema* is 1.

Semaphore Status			SEMA_STATUS	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
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_SEMAPHORES[2].sema* is 0. 1: *SEMA_SEMAPHORES[2].sema* 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_SEMAPHORES[1].sema* is 0. 1: *SEMA_SEMAPHORES[1].sema* 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_SEMAPHORES[0].sema* is 0. 1: *SEMA_SEMAPHORES[0].sema* is 1.

10. Standard DMA (DMA)

The DMA is a peripheral that provides the ability to perform high-speed, block memory transfers of data independent of a CPU. All DMA transactions consist of a 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 SRAM address.
- From an SRAM address to a transmit FIFO.
- From a source SRAM address to a destination SRAM address.

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

- Complete 32-bit source and destination address 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

10.1 Instances

There is one instance of the DMA, referred to as DMA. The DMA provides 16 channels, generically referred to as DMA_CHn. The DMA includes a set of interrupt registers common to all of its channels and a set of registers unique to each channel instance.

Table 10-1: MAX32690 DMA and Channel Instances

DMA Instance	DMA_CH*n* Channel Instance
DMA	DMA_CH0
	DMA_CH1
	DMA_CH2
	DMA_CH3
	DMA_CH4
	DMA_CH5
	DMA_CH6
	DMA_CH7
	DMA_CH8
	DMA_CH9
	DMA_CH10
	DMA_CH11
	DMA_CH12
	DMA_CH13
	DMA_CH14
	DMA_CH15

10.2 Peripheral Clock Enable

The DMA is disabled by default on a reset. Use of the DMA peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS0.dma* to 0 to enable the peripheral clock to the DMA before configuring the DMA for use.

10.3 DMA Channel Operation (DMA_CH)

10.3.1 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_CTRL.pri* 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_CTRL.en* bit.

When disabling a channel, poll the *DMA_CHn_STATUS.status* bit to determine if the channel is disabled. In general, *DMA_CHn_STATUS.status* follows the setting of the *DMA_CHn_CTRL.en* bit. However, the *DMA_CHn_STATUS.status* bit is automatically cleared under the following conditions:

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

Whenever *DMA_CHn_STATUS.status* transitions from 1 to 0, the corresponding *DMA_CHn_CTRL.en* bit is also cleared. If an active channel is disabled during an AHB read/write burst, the current burst continues until complete.

Only an error condition can interrupt an ongoing data transfer.

10.3.2 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_CTRL.request* field dictates the source and destination for a channel's DMA transfer, as shown in *Table 10-2*. 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_CTRL.srcinc* field is ignored when the DMA source is a peripheral memory, and the *DMA_CHn_CTRL.dstinc* field is ignored when the DMA destination is a peripheral memory.

Table 10-2: MAX32690 DMA Source and Destination by Peripheral

DMA_CHn_CTRL.request	Peripheral	DMA Source	DMA Destination
0x00	Memory-to-Memory	*DMA_CHn_SRC*	*DMA_CHn_DST*
0x01	SPI0	SPI0 Receive FIFO	*DMA_CHn_DST*
0x02	SPI1	SPI1 Receive FIFO	*DMA_CHn_DST*
0x03	SPI2	SPI2 Receive FIFO	*DMA_CHn_DST*
0x04	UART0	UART0 Receive FIFO	*DMA_CHn_DST*
0x05	UART1	UART1 Receive FIFO	*DMA_CHn_DST*
0x06	CAN0	CAN0 Receive FIFO	*DMA_CHn_DST*
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*
0x0B:0x0D	Reserved		
0x0E	UART2	UART2 Receive FIFO	*DMA_CHn_DST*
0x0F	SPI3	SPI3 Receive FIFO	*DMA_CHn_DST*
0x10	SPI4	SPI4 Receive FIFO	*DMA_CHn_DST*
0x11	USB1	USB IN Endpoint 1	*DMA_CHn_DST*
0x12	USB2	USB IN Endpoint 2	*DMA_CHn_DST*
0x13	USB3	USB IN Endpoint 3	*DMA_CHn_DST*
0x14	USB4	USB IN Endpoint 4	*DMA_CHn_DST*
0x15	USB5	USB IN Endpoint 5	*DMA_CHn_DST*
0x16	USB6	USB IN Endpoint 6	*DMA_CHn_DST*
0x17	USB7	USB IN Endpoint 7	*DMA_CHn_DST*
0x18	USB8	USB IN Endpoint 8	*DMA_CHn_DST*
0x19	USB9	USB IN Endpoint 9	*DMA_CHn_DST*
0x1A	USB10	USB IN Endpoint 10	*DMA_CHn_DST*
0x1B	USB11	USB IN Endpoint 11	*DMA_CHn_DST*
0x1C	UART3 (LPUART0)	UART3 Receive FIFO	*DMA_CHn_DST*
0x1D	Reserved	-	-
0x1E	I^2S	I^2S Receive FIFO	*DMA_CHn_DST*
0x1F	CAN1	CAN1 Receive FIFO	*DMA_CHn_DST*
0x20	Reserved	-	-
0x21	SPI0	*DMA_CHn_SRC*	SPI0 Transmit FIFO
0x22	SPI1	*DMA_CHn_SRC*	SPI1 Transmit FIFO
0x23	SPI2	*DMA_CHn_SRC*	SPI2 Transmit FIFO
0x24	UART0	*DMA_CHn_SRC*	UART0 Transmit FIFO
0x25	UART1	*DMA_CHn_SRC*	UART1 Transmit FIFO
0x26	CAN0	*DMA_CHn_SRC*	CAN0 Transmit FIFO
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:2D	Reserved	-	-
0x2E	UART2	*DMA_CHn_SRC*	UART2 Transmit FIFO
0x2F	SPI3	*DMA_CHn_SRC*	SPI3 Transmit FIFO
0x30	SPI4	*DMA_CHn_SRC*	SPI4 Transmit FIFO
0x31	USB1	*DMA_CHn_SRC*	USB OUT Endpoint 1
0x32	USB2	*DMA_CHn_SRC*	USB OUT Endpoint 2
0x33	USB3	*DMA_CHn_SRC*	USB OUT Endpoint 3
0x34	USB4	*DMA_CHn_SRC*	USB OUT Endpoint 4
0x35	USB5	*DMA_CHn_SRC*	USB OUT Endpoint 5
0x36	USB6	*DMA_CHn_SRC*	USB OUT Endpoint 6
0x37	USB7	*DMA_CHn_SRC*	USB OUT Endpoint 7
0x38	USB8	*DMA_CHn_SRC*	USB OUT Endpoint 8
0x39	USB9	*DMA_CHn_SRC*	USB OUT Endpoint 9

DMA_CHn_CTRL.request	Peripheral	DMA Source	DMA Destination
0x3A	USB10	*DMA_CHn_SRC*	USB OUT Endpoint 10
0x3B	USB11	*DMA_CHn_SRC*	USB OUT Endpoint 11
0x3C	UART3 (LPUART0)	*DMA_CHn_SRC*	LPUART Transmit FIFO
0x3D	Reserved		
0x3E	I²S	*DMA_CHn_SRC*	I²S Transmit FIFO
0x3F	CAN1	*DMA_CHn_SRC*	CAN1 Transmit FIFO

10.3.3 Data Movement from Source to DMA

Table 10-3 shows the fields that control the burst movement of data into the DMA FIFO. The source is a peripheral or memory.

Table 10-3: 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_CTRL.burst_size	Burst size (1-32)	This maximum number of bytes moved during the burst read.
DMA_CHn_CTRL.srcwd	Source width	This field 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_CTRL.srcinc	Source increment enable	Increments *DMA_CHn_SRC*. This field is ignored when the DMA source is a peripheral.

10.3.4 Data Movement from DMA to Destination

Table 10-4 shows the fields that control the burst movement of data out of the DMA FIFO. The destination is a peripheral or memory.

Table 10-4: 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_CTRL.burst_size	Burst size (1-32)	The maximum number of bytes moved during a single AHB read/write burst.
DMA_CHn_CTRL.dstwd	Destination width	This field determines the maximum data width used during each write of the AHB burst (one byte, two bytes, or four bytes).
DMA_CHn_CTRL.dstinc	Destination increment enable	Increments *DMA_CHn_DST*. This field is ignored when the DMA destination is a peripheral.

10.4 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_CTRL.en, DMA_CHn_CTRL.rlden* = 0, and *DMA_CHn_STATUS.ctz_if* = 0.
2. If using memory for the DMA transfer destination, configure the *DMA_CHn_DST* register to the destination memory's starting address.
3. If using memory for the DMA transfer source, configure the *DMA_CHn_SRC* register 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_CTRL* register fields in one or more instructions. Do not set *DMA_CHn_CTRL.en* to 1 or *DMA_CHn_CTRL.rlden* to 1 in this step:
   a. Configure *DMA_CHn_CTRL.request* to select the transfer operation associated with the DMA channel.
   b. Configure *DMA_CHn_CTRL.burst_size* for the desired burst size.
   c. Configure *DMA_CHn_CTRL.pri* to set the channel priority relative to other DMA channels.
   d. Configure *DMA_CHn_CTRL.dstwd* to dictate the number of bytes written in each transaction.
   e. If desired, set *DMA_CHn_CTRL.dstinc* to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.
   f. Configure *DMA_CHn_CTRL.srcwd* to dictate the number of bytes read in each transaction.
   g. If desired, set *DMA_CHn_CTRL.srcinc* to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.
   h. If desired, set *DMA_CHn_CTRL.dis_ie* = 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_CTRL.ctz_ie* 1 to generate an interrupt when the *DMA_CHn_CNT* register is decremented to zero.
   j. If using the reload feature, configure the reload registers to set the destination, source, and count for the following DMA transaction.
      1) Load the *DMA_CHn_SRCRLD* register with the source address reload value.
      2) Load the *DMA_CHn_DSTRLD* register with the destination address reload value.
      3) Load the *DMA_CHn_CNTRLD* register with the count reload value.
   k. If desired, enable the channel timeout feature described in *Channel Timeout Detect*. Clear *DMA_CHn_CTRL.to_clkdiv* to 0 to disable the channel timeout feature.
6. Set *DMA_CHn_CTRL.rlden* to 1 to enable the reload feature.
7. Set *DMA_CHn_CTRL.en* to 1 to start the DMA transfer immediately.
8. Wait for the interrupt flag to become 1 to indicate the completion of the DMA transfer.

10.5 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, two possible responses are possible depending on the value of the *DMA_CHn_CTRL.rlden* field:

- If *DMA_CHn_CTRL.rlden* = 1
  - 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.
- If *DMA_CHn_CTRL.rlden* = 0
  - The channel is disabled, and *DMA_CHn_STATUS.status* is cleared.

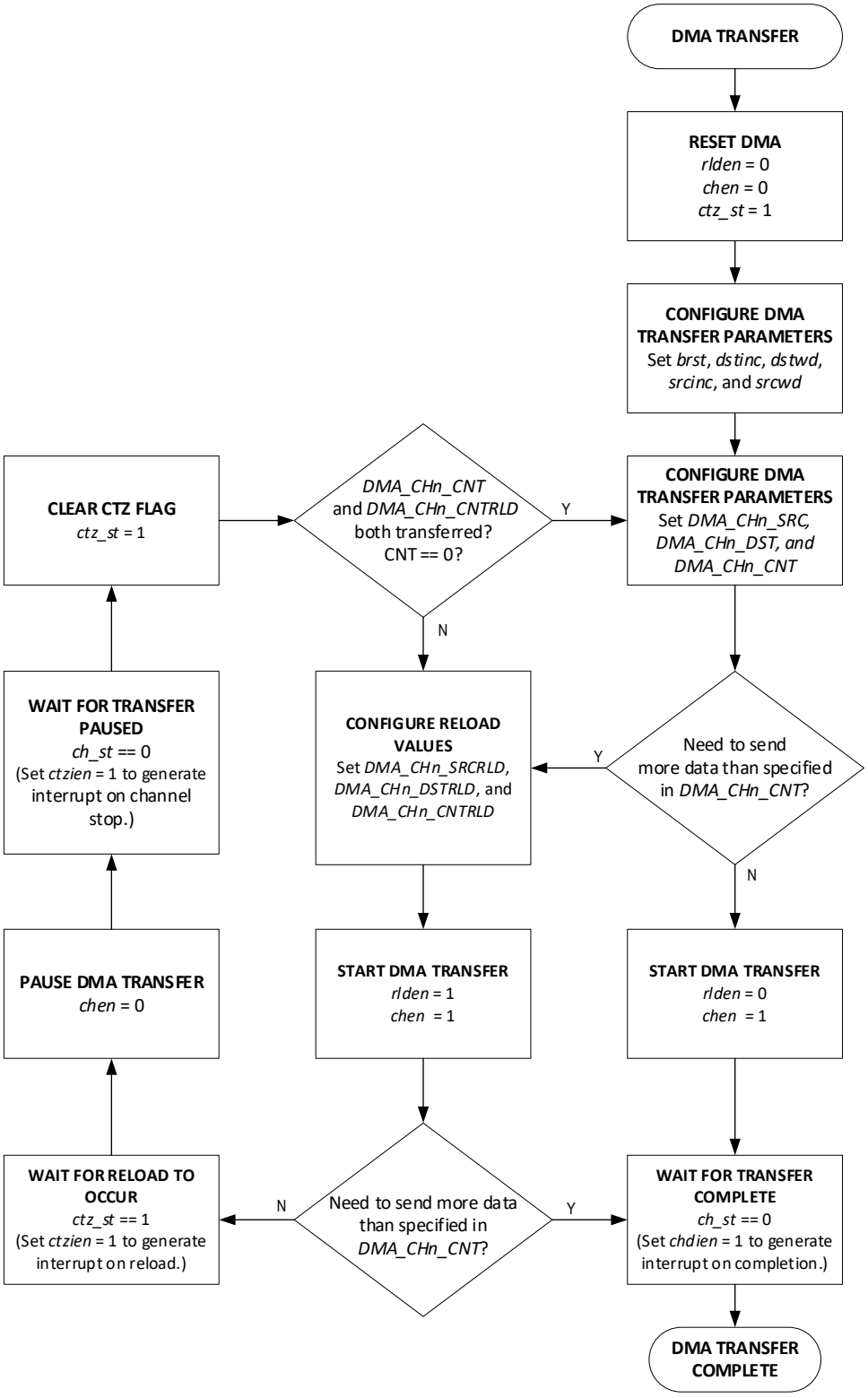
10.6 Chaining Buffers

Chaining buffers reduces the DMA interrupt response time and allows the DMA to service requests without intermediate processing from the CPU. *Figure 10-1* 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_CTRL*
  - *DMA_CHn_SRC*
  - *DMA_CHn_DST*
  - *DMA_CHn_CNT*
  - *DMA_CHn_SRCRLD*
  - *DMA_CHn_DSTRLD*
  - *DMA_CHn_CNTRLD*

Writing to any register while a channel is disabled is supported, but there are certain restrictions when a channel is enabled. The *DMA_CHn_STATUS.status* bit indicates whether the channel is enabled or not. Because an active channel might be in the middle of an AHB read or 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_STATUS.status* = 1). To disable any DMA channel, clear the *DMA_INTEN.ch<n>* bit. Then, poll the *DMA_CHn_STATUS.status* bit to verify that the channel is disabled.

Figure 10-1: DMA Block-Chaining Flowchart



10.7 DMA Interrupts

Enable interrupts for each channel by setting *DMA_INTEN.ch<n>*. When an interrupt for a channel is pending, the corresponding *DMA_INTFL.ch<n>* = 1. Set the corresponding enable bit to cause an interrupt when the flag is set.

A channel interrupt (*DMA_CHn_STATUS.ipend* = 1) is caused by:

- *DMA_CHn_CTRL.ctz_ie* = 1
  - If enabled, all CTZ occurrences set the *DMA_CHn_STATUS.ipend* bit.
- *DMA_CHn_CTRL.dis_ie* = 1
  - If enabled, any clearing of the *DMA_CHn_STATUS.status* bit sets the *DMA_CHn_STATUS.ipend* bit. Examine the *DMA_CHn_STATUS* register to determine which reasons caused the disable. The *DMA_CHn_CTRL.dis_ie* bit also enables the *DMA_CHn_STATUS.to_if* bit. The *DMA_CHn_STATUS.to_if* bit does not clear the *DMA_CHn_STATUS.status* bit.

To clear the channel interrupt, write 1 to the cause of the interrupt (the *DMA_CHn_STATUS.ctz_if*, *DMA_CHn_STATUS.rld_if*, *DMA_CHn_STATUS.bus_err*, or *DMA_CHn_STATUS.to_if* bits).

When running in normal mode without buffer chaining (*DMA_CHn_CTRL.rlden* = 0), set the *DMA_CHn_CTRL.dis_ie* 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_CTRL.rlden* = 1), set both the *DMA_CHn_CTRL.dis_ie* and *DMA_CHn_CTRL.ctz_ie* bits. The CTZ interrupts occur on completion of each DMA (count reaches zero, and reload occurs). The setting of *DMA_CHn_CTRL.dis_ie* ensures that an error condition generates an interrupt. If *DMA_CHn_CTRL.ctz_ie* = 0, then the only interrupt occurs when the DMA completes and *DMA_CHn_CTRL.rlden* = 0 (final DMA).

10.8 Channel Timeout Detect

Each channel can optionally generate an interrupt when the associated peripheral does not request a transfer in a user-configurable period. When the timeout start conditions are met, an internal 10-bit counter begins incrementing at a frequency determined by the AHB clock, *DMA_CHn_CTRL.to_clkdiv*, and *DMA_CHn_CTRL.to_per* shown in *Table 10-5*. 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 10-5: DMA Channel Timeout Configuration

***DMA_CHn_CTRL*.to_clkdiv**	**Timeout Period (µs)**
0x0	Channel timeout disabled
0x1	$\frac{2^8 * [Value\ from\ DMA\_CHn\_CTRL.to\_per]}{f_{HCLK}}$
0x2	$\frac{2^{16} * [Value\ from\ DMA\_CHn\_CTRL.to\_per]}{f_{HCLK}}$
0x3	$\frac{2^{24} * [Value\ from\ DMA\_CHn\_CTRL.to\_per]}{f_{HCLK}}$

The start of the timeout period is controlled by the *DMA_CHn_CTRL.to_wait* field as follows:

- If *DMA_CHn_CTRL.to_wait* = 0, the timer begins counting immediately after the *DMA_CHn_CTRL.to_clkdiv* field is configured to a value other than 0.
- If *DMA_CHn_CTRL.to_wait* = 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_STATUS.status* = 0).

If the timeout timer period expires, the hardware sets *DMA_CHn_STATUS.to_if* = 1 to indicate a channel timeout event has occurred. A channel timeout does not disable the DMA channel.

10.9 Memory-to-Memory DMA

Memory-to-memory transfers are processed as if the request is permanently active. 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.

10.10 DMA Registers

See *Table 3-2* for this peripheral/module's base address. See *Table 1-1* 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 10-6: DMA Register Summary

Offset	Register	Description
[0x0000]	*DMA_INTEN*	DMA Interrupt Enable register
[0x0004]	*DMA_INTFL*	DMA Interrupt Flag register

10.10.1 Register Details

Table 10-7: DMA Interrupt Enable Register

DMA Interrupt Enable			**DMA_INTEN**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	*ch<n>*	R/W	0	**DMA Channel *n* Interrupt Enable** Each bit in this field enables the corresponding channel interrupt *n* in *DMA_INTFL*. Register bits associated with unimplemented channels should not be changed from their default reset value. 0: Disabled. 1: Enabled.

Table 10-8: DMA Interrupt Flag Register

DMA Interrupt Flag			**DMA_INTFL**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	ch<n>	RO	0	**DMA Channel *n* 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_STATUS* register. An interrupt bit in this field is only set if the corresponding interrupt enable field is set in the *DMA_INTEN* register. Register bits associated with unimplemented channels should be ignored. 0: Normal operation. 1: Interrupt pending.

10.11 DMA Channel Register Summary

Table 10-9: Standard DMA Channel 0 to Channel 15 Register Summary

Offset	DMA Channel	Description
[0x0100]	DMA_CH0	DMA Channel 0
[0x0120]	DMA_CH1	DMA Channel 1
[0x0140]	DMA_CH2	DMA Channel 2
[0x0160]	DMA_CH3	DMA Channel 3
[0x0180]	DMA_CH4	DMA Channel 4
[0x01C0]	DMA_CH5	DMA Channel 5
[0x01E0]	DMA_CH6	DMA Channel 6

Offset	DMA Channel	Description
[0x0200]	DMA_CH7	DMA Channel 7
[0x0220]	DMA_CH8	DMA Channel 8
[0x0240]	DMA_CH9	DMA Channel 9
[0x0260]	DMA_CH10	DMA Channel 10
[0x0280]	DMA_CH11	DMA Channel 11
[0x02C0]	DMA_CH12	DMA Channel 12
[0x02E0]	DMA_CH13	DMA Channel 13
[0x0300]	DMA_CH14	DMA Channel 14
[0x0320]	DMA_CH15	DMA Channel 15

10.12 DMA Channel Registers

See *Table 3-2* for this peripheral/module's base address. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 10-10*. 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 1-1* 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 10-10: DMA Channel Registers Summary

Offset	Register	Description
[0x0000]	*DMA_CHn_CTRL*	DMA Channel *n* Control Register
[0x0004]	*DMA_CHn_STATUS*	DMA Channel *n* Status Register
[0x0008]	*DMA_CHn_SRC*	DMA Channel *n* Source Register
[0x000C]	*DMA_CHn_DST*	DMA Channel *n* Destination Register
[0x0010]	*DMA_CHn_CNT*	DMA Channel *n* Count Register
[0x0014]	*DMA_CHn_SRCRLD*	DMA Channel *n* Source Reload Register
[0x0018]	*DMA_CHn_DSTRLD*	DMA Channel *n* Destination Reload Register
[0x001C]	*DMA_CHn_CNTRLD*	DMA Channel *n* Count Reload Register

10.12.1 Register Details

Table 10-11: DMA Channel n Control Register

DMA Channel *n* Control			**DMA_CHn_CTRL**	**[0x0100]**
Bits	**Field**	**Access**	**Reset**	**Description**
31	ctz_ie	R/W	0	**CTZ Interrupt Enable** 0: Disabled. 1: Enabled. *DMA_INTFL.ch<n>_ipend* is set to 1 whenever a CTZ event occurs.
30	dis_ie	R/W	0	**Channel Disable Interrupt Enable** 0: Disabled. 1: Enabled. *DMA_INTFL.ch<n>_ipend* bit is set to 1 whenever *DMA_CHn_STATUS.status* changes from 1 to 0.
29	-	RO	0	**Reserved**
28:24	burst_size	R/W	0	**Burst Size** The number of bytes transferred into and out of the DMA FIFO in a single burst. 0: 1 byte. 1: 2 bytes. 2: 3 bytes. ... 31: 32 bytes.
23	-	RO	0	**Reserved**

DMA Channel *n* Control			DMA_CHn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
22	dstinc	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** This field selects the width of each AHB transaction to the destination peripheral or memory. The actual width can be less than this field's setting if fewer bytes are in the DMA FIFO than this field's selection. 0: 1 byte. 1: 2 bytes. 2: 4 bytes. 3: 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** This field selects the width of each AHB transaction from the source peripheral or memory. The actual width can be less than this field's setting if the *DMA_CHn_CNT* register indicates a smaller value than the width setting. 0: 1 byte. 1: 2 bytes. 2: 4 bytes. 3: Reserved.
15:14	to_clkdiv	R/W	0	**Timeout Timer Clock Pre-Scale Select** This field selects the pre-scale divider for the timeout clock input. 0: Timeout timer disabled. 1: $\frac{f_{HCLK}}{2^8}$ 2: $\frac{f_{HCLK}}{2^{16}}$f 3: $\frac{f_{HCLK}}{2^{24}}$
13:11	to_per	R/W	0	**Timeout Period Select** This field 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 to 4. 1: 7 to 8. 2: 15 to 16. 3: 31 to 32. 4: 63 to 64. 5: 127 to 128. 6: 255 to 256. 7: 511 to 512.
10	to_wait	R/W	0	**Request DMA Timeout Timer Wait Enable** 0: Start timer immediately when enabled. 1: Delay the timer's start until after the first DMA transaction occurs.
9:4	request	R/W	0	**Request Select** Selects the source and destination for the transfer as shown in *Table 10-2*.

DMA Channel *n* Control			DMA_CHn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
3:2	pri	R/W	0	**Channel Priority** This field sets the priority of the channel relative to other DMA channels. Channels set to the same priority are serviced in a round-robin fashion. 0: Highest priority. 1: … 2: … 3: 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 DMA_CHn_CNTRLD register.*
0	en	R/W	0	**Channel Enable** This bit is automatically cleared when *DMA_CHn_STATUS.status* changes from 1 to 0. 0: Disabled. 1: Enabled.

Table 10-12: DMA Status Register

DMA Channel *n* Status			DMA_CHn_STATUS	[0x0104]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	DNM	0	**Reserved, Do Not Modify**
6	to_if	R/W1C	0	**Timeout Interrupt Flag** Timeout. Write 1 to clear. 0: No time out. 1: A channel time out 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 is disabled by hardware. Write 1 to clear. 0: No error found. 1: An AHB bus error occurred.
3	rld_if	R/W1C	0	**Reload Interrupt Flag** Reload. Write 1 to clear. 0: Reload has not occurred. 1: Reload occurred.
2	ctz_if	R/W1C	0	**CTZ Interrupt Flag** 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
0	status	RO	0	**Channel Status** This bit indicates when it is safe to change the channel's configuration, address, and count registers. Whenever this bit is cleared by hardware, the *DMA_CHn_CTRL.en* bit is also cleared. 0: Disabled. 1: Enabled.

Table 10-13: DMA Channel n Source Register

DMA Channel *n* Source			DMA_CHn_SRC	[0x0108]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**Source Address** This field is the source RAM address for memory-to-peripheral and memory-to-memory transfers. This field is ignored for peripheral-to-memory transfers. If *DMA_CHn_CTRL.srcinc* = 1, then this register is incremented on each AHB transfer cycle by one, two, or four bytes depending on the data width selected using *DMA_CHn_CTRL.srcwd*. If *DMA_CHn_CTRL.srcinc* = 0, this register remains constant. If a CTZ condition occurs while *DMA_CHn_CTRL.rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_SRCRLD* register.

Table 10-14: 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** This field is the destination RAM address for peripheral-to-memory and memory-to-memory transfers. This field is ignored for memory-to-peripheral transfers. If *DMA_CHn_CTRL.dstinc* = 1, then this field is incremented on every AHB transfer cycle by one, two, or four bytes depending on the data width selected using *DMA_CHn_CTRL.dstwd*. If a CTZ condition occurs while *DMA_CHn_CTRL.rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_DSTRLD* register.

Table 10-15: 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_CTRL.rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_CNTRLD* register.

Table 10-16: DMA Channel n Source Reload Register

DMA Channel *n* Source Reload			DMA_CHn_SRCRLD	[0x0114]
Bits	**Field**	**Access**	**Reset**	**Description**
31	-	RO	0	**Reserved**
30:0	addr	R/W	0	**Source Address Reload Value** If *DMA_CHn_CTRL.rlden* = 1, then this register's value is loaded into *DMA_CHn_SRC* upon a CTZ condition.

Table 10-17: DMA Channel n Destination Reload Register

DMA Channel *n* Destination Reload			DMA_CHn_DSTRLD	[0x0118]
Bits	**Field**	**Access**	**Reset**	**Description**
31	-	RO	0	**Reserved**
30:0	addr	R/W	0	**Destination Address Reload Value** If *DMA_CHn_CTRL.rlden* = 1, then this register's value is loaded into *DMA_CHn_DST* upon a CTZ condition.

Table 10-18: DMA Channel n Count Reload Register

DMA Channel *n* Count Reload			**DMA_CHn_CNTRLD**	**[0x011C]**
Bits	**Field**	**Access**	**Reset**	**Description**
31	ren	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_CTRL register.* 0: Reload disabled. 1: Reload enabled.
30:24	-	RO	0	**Reserved**
23:0	cnt	R/W	0	**Count Reload Value.** If *DMA_CHn_CNTRLD.en* = 1, then this register's value is loaded into *DMA_CHn_CNT* upon a CTZ condition.

11. ADC

The 12-bit successive approximation (SAR) ADC includes a single-ended input multiplexer and an integrated reference generator. It can measure up to eight single- external analog inputs and internal power supplies.

The device samples any or all of the inputs in a user-defined conversion sequence that can execute once or run continuously. The conversion sequence can immediately begin when enabled by software or a specific hardware event such as a timer interrupt or transition on an external GPIO pin. A user-programmable delay can be inserted between conversions in continuous mode.

- 12-bit successive approximation ADC

Conversion speed up to 1MSPS without input buffer

- Conversion speed up to 25KSPS with input buffer

Internal reference without external capacitor

  - 1.25V or 2.048V
- Support for external reference from 2.048V to V_{DDA}

Capacitor calibration

11.1 Operation

Measurements are performed in a series of user-defined channel measurements called conversion sequences. Conversion sequences can be set up as single or continuous. Software triggered and hardware triggered conversion sequences are supported.

Conversion sequences can measure single or multiple channels. The specific channels for a conversion sequence are set using the ADC channel select registers (*ADC_CHSEL4*:*ADC_CHSEL0*) and the *slot0_id* through *slot19_id* fields. A conversion sequence begins with the channel configured for slot 0 and continues sequentially through the software configured number of slots (*ADC_CTRL1*.*num_slots*) up to slot 19.

Each measurement is pushed onto the 16-word FIFO to be read by software. A FIFO threshold interrupt can alert the software when the FIFO must be read to avoid overwriting previous measurements. Several data formats are available to the user.

11.1.1 Peripheral Clock Enable

The ADC is disabled by default on a reset. Use of the ADC peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS0*.*adc* to 0 to enable the peripheral clock to the ADC before configuring the ADC for use.

11.2 Input Channels

Each input channels is shown in *Table 11-1*.

Table 11-1: MAX32690 Channel Assignments

Channel ID	Source	Mode	Alternate Function Name[1]
0	AIN0	Single-ended	AIN0
1	AIN1	Single-ended	AIN1
2	AIN2	Single-ended	AIN2
3	AIN3	Single-ended	AIN3
4	AIN4	Single-ended	AIN4
5	AIN5	Single-ended	AIN5
6	AIN6	Single-ended	AIN6
7	AIN7	Single-ended	AIN7
8-11	Reserved	-	-
12	$V_{DDA}/2$	Single-ended	N/A
13	-	Single-ended	-
14	V_{COREA}	Single-ended	N/A
15	V_{SS}	Single-ended	N/A
16-17	Reserved	-	-
18	$V_{DD3A}/4$	Single-ended	N/A
19	$V_{DDB}/4$	Single-ended	N/A
20	V_{SSA}	Single-ended	N/A

1. Refer to the device data sheet's pin description table for pin numbers and alternate function assignments.

11.3 Analog Input Buffer

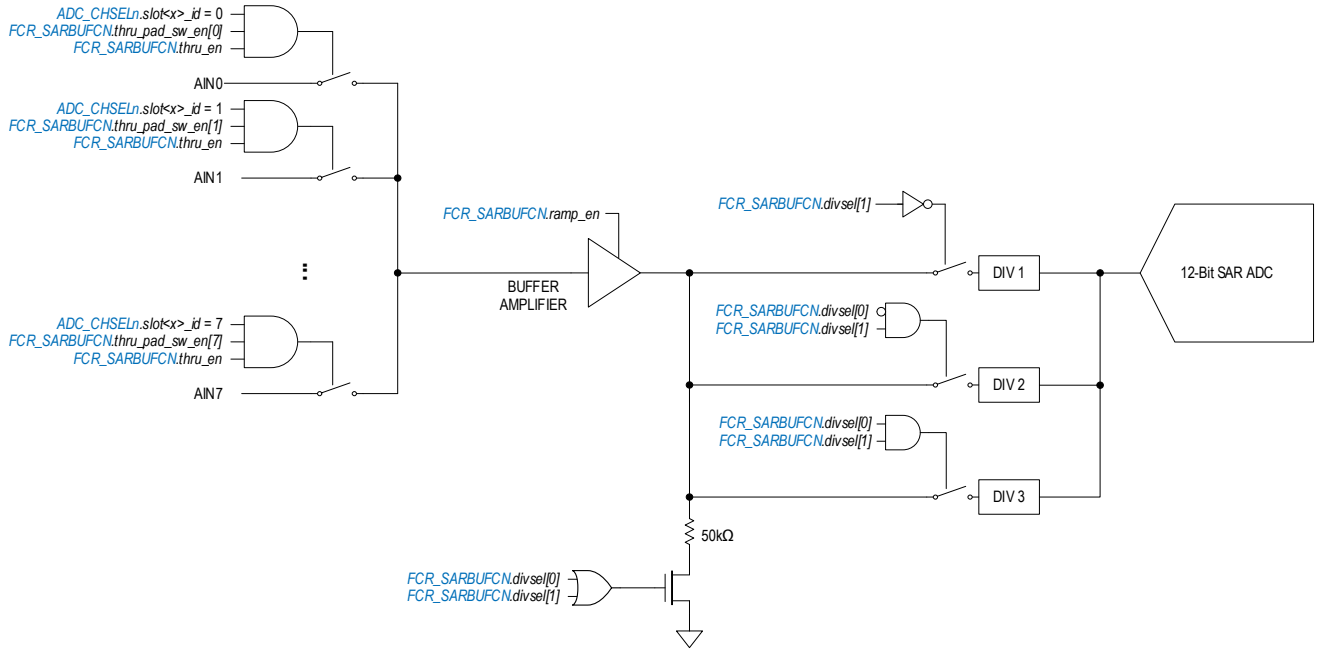
The MAX32690 includes an analog input buffer, optionally selectable for each of the external analog input signals. *Figure 11-1* shows the analog input buffer path. When using the analog input buffer path, the ADC is limited to a maximum of 25KSPS. See *Clocks and Timing* for details on configuring the sample rate of the ADC. Enable rail-to-rail support for the buffer by setting *FCR_SARBUFCN.thru_rri_en* to 1.

Note: Each individual external analog input can be selected for use with the analog input buffer path. The input signal divider (FCR_SARBUFCN.divsel) applies to all inputs using the analog input buffer path.

The following steps show how to enable the analog input buffer path for AIN2 with a divide by 3 path.

1. Configure P3.2 for Alternate Function 1. See *Alternate Function Usage* for details on configuring an I/O pin for alternate functionality.
1. 2. Set the *FCR_SARBUFCN.thru_pad_sw_en2* field to enable analog input 2 to connect to the analog input buffer path.
    a. Each analog input can be enabled for the buffer path by setting the corresponding through path switch enable field in the *FCR_SARBUFCN* register.
3. Set *FCR_SARBUFCN.thru_en* to 1 to enable the analog input buffer path.
2. 4. Select the input signal divided by 3 option by setting *FCR_SARBUFCN.divsel* field to 3.
5. Set *FCR_SARBUFCN.ramp_en* to 1 to enable the analog input buffer.
3. 6. Configure the ADC for operation and set the desired slot id field for AIN2.
    a. When the input buffer is enabled, the ADC is limited to a maximum of 25KSPS.

Figure 11-1: MAX32690 Analog Input Buffer Signal Path



11.4 Clocks and Timing

Clock and timing configurations are calculated based on the application-specific sampling rate requirements. Several parameters can be adjusted to optimize the ADC power consumption, accuracy, and start-up time. *Table 11-2* shows the ADC clock sources available for the device. The maximum sample rate of the ADC is 1MSPS. If the analog input buffer is enabled, the maximum sample rate of the ADC is 25KSPS.

Table 11-2: MAX32690 ADC Clock Sources

ADC_CLKCTRL.clksel	Source (f_{ADC_SRC})
0	SYS_OSC
1	ADC_CLK_EXT (P0.9 / AF1)
2	IBRO
3	ERFO

The ADC clock frequency (f_{SAR_CLK}) is derived from a selectable clock source (f_{ADC_SRC}) and divided by a selectable clock divider, as shown in *Equation 11-1*. *Table 11-2* lists the available sources for f_{ADC_SRC}. The clock divider is selected using the *ADC_CLKCTRL.clkdiv* field.

Equation 11-1: ADC Clock Generation

$$For\ ADC\_CLKCTRL.clkdiv \leq 3 \quad f_{SAR\_CLK} = \frac{f_{ADC\_SRC}}{2^{(ADC\_CLKCTRL.clkdiv+1)}}$$

$$For\ ADC\_CLKCTRL.clkdiv > 3 \quad f_{SAR\_CLK} = f_{ADC\_SRC}$$

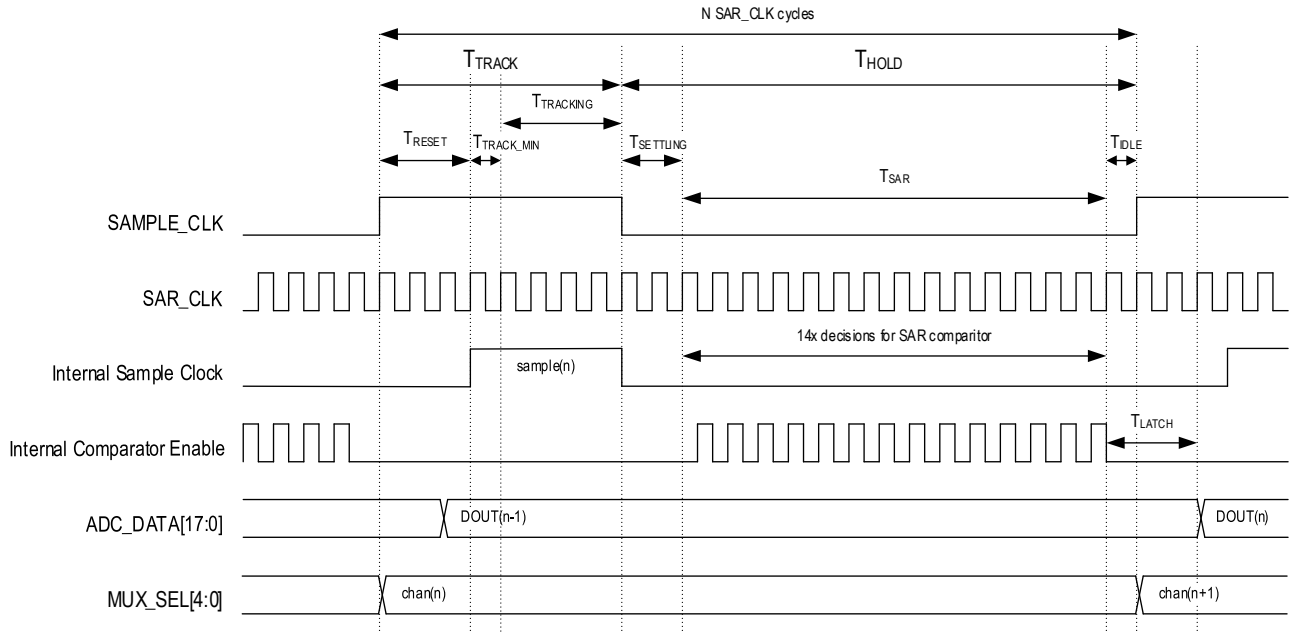
$$f_{SAR\_CLK} \leq 25MHz$$

The SAMPLE_CLK frequency determines the sampling rate, conversion time, and the delay between conversions. It is defined by a high track time and a low hold time of SAR_CLK periods. The sum of the track and hold defines the SAMPLE_CLK frequency (sample rate). *Figure 11-2* shows the SAMPLE_CLK and its relationship to the track and hold values.

Equation 11-2: Sample Clock Frequency Calculation

$$t_{SAMPLE\_CLK} = (TRACK + HOLD) \times t_{SAR\_CLK}$$

Figure 11-2: ADC Sample Clock



NOTE: T_{RESET} = 3
T_{TRACK_MIN} = 1
$T_{TRACKING}$ = *ADC_SAMPCLKCTRL.track_cnt*
$T_{SETTLING}$ = 2
T_{SAR} = 14
T_{IDLE} = 1 + *ADC_SAMPCLKCTRL.idle_cnt*

Equation 11-3: T_{TRACK} Calculation

$$T_{TRACK} = T_{RESET} + T_{TRACK\_MIN} + T_{TRACKING}$$
$$T_{TRACK} = 4 + ADC\_SAMPCLKCTRL.track\_cnt$$
$$T_{TRACK} \geq 8$$

Equation 11-4: T_{HOLD} Calculation

$$T_{HOLD} = T_{SETTLING} + T_{SAR} + T_{IDLE}$$
$$T_{HOLD} = 17 + ADC\_SAMPCLKCTRL.idle\_cnt$$
$$T_{HOLD} \geq 17$$

The ADC requires a minimum T_{TRACK} of 8 SAR_CLK cycles and a minimum T_{HOLD} of 17 SAR_CLK cycles. The *ADC_SAMPCLKCTRL.track_cnt* and *ADC_SAMPCLKCTRL.idle_cnt* fields add SAR_CLK cycles to the track and hold, as shown in *Equation 11-3* and *Equation 11-4*.

As an example, the following steps show the settings required to achieve a 250KSPS rate for the ADC using the IPO as the clock source.

1. Select the ADC_SRC clock as the IPO.
   a. Set *ADC_CLKCTRL*.*clksel* to 0.
2. Select the clock divider to achieve a valid SAR_CLK frequency using *Equation 11-1*.
   a. Set *ADC_CLKCTRL*.*clkdiv* to 3 (divide by 16, $f_{SAR\_CLK} \leq 25MHz$)
   b. $t_{SAR\_CLK} = 133ns$
3. Determine the SAMPLE_CLK for 250KSPS
   a. $T_{TRACK} + T_{HOLD} = \frac{7.5MHz}{250KHz} = 30$
4. Determine the *ADC_SAMPCLKCTRL*.*track_cnt* setting using *Equation 11-3*.
   a. *ADC_SAMPCLKCTRL*.*track_cnt* = 4 ($T_{TRACK} \geq 8$)
5. Determine the *ADC_SAMPCLKCTRL*.*idle_cnt* setting using *Equation 11-4*.
   a. $T_{HOLD} = 30 - T_{TRACK} = 30 - 8 = 22$ ($T_{HOLD} \geq 17$)
   b. *ADC_SAMPCLKCTRL*.*idle_cnt* = 22 - 17 = 5

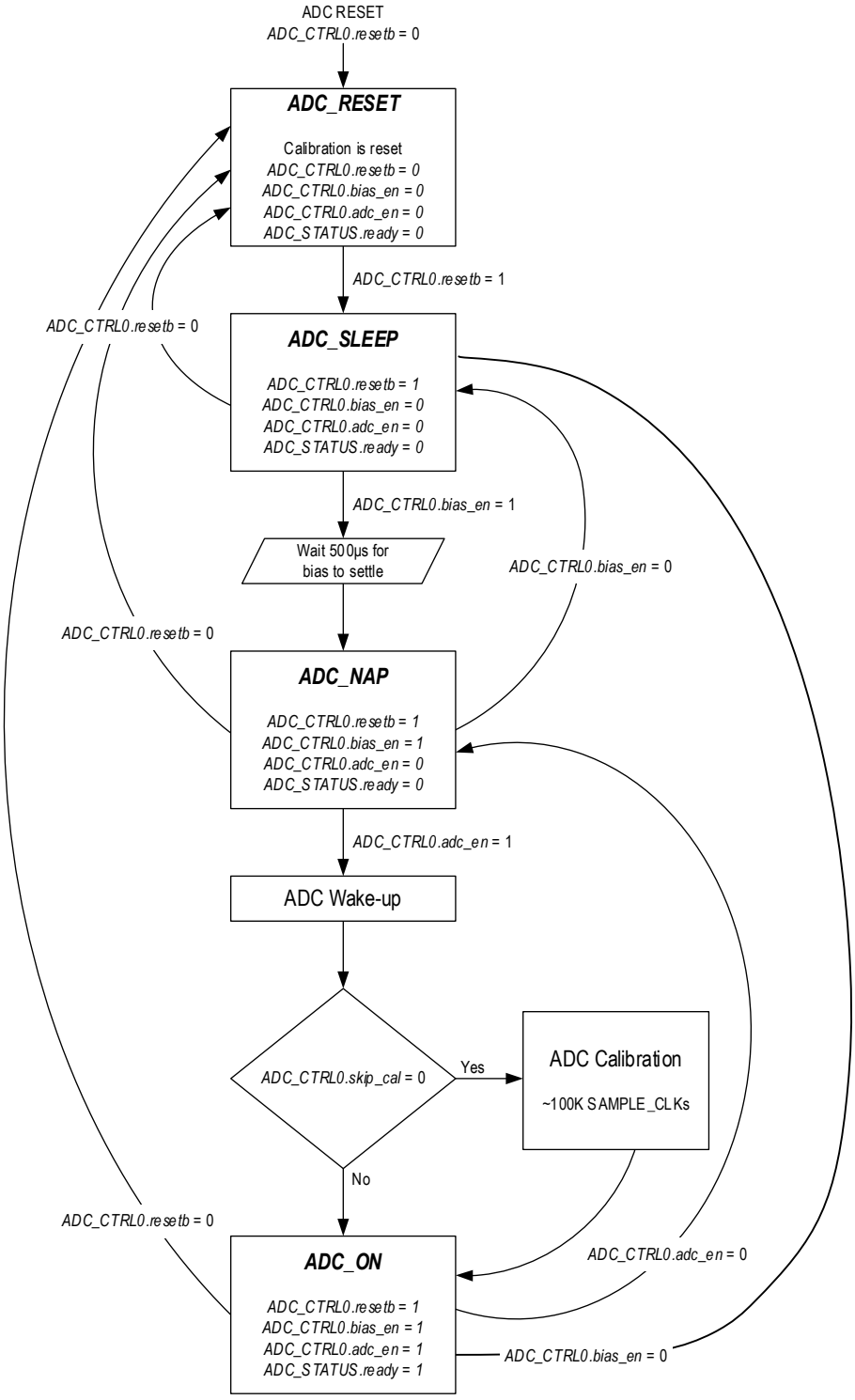
11.5 Operating Modes

Four operating modes allow the ADC to minimize power consumption based on the current needs of the peripheral. After a POR, system reset, peripheral reset (*GCR_RST0*.*adc* = 1), or software directly resetting the ADC (*ADC_CTRL0*.*resetb* = 0), the ADC bias regulator is disabled, and the ADC calibration values are reset to 0. The bias regulator must be enabled before performing a measurement, and a capacitor calibration can optionally be performed. Enabling the bias regulator requires 500µs before performing a conversion. Section *11.5.1* describes initializing the ADC from *ADC_RESET*. Section *11.5.1.2* describes entering *ADC_NAP* state. Section *11.5.1.3* describes the steps required to enter the *ADC_ON* state and perform an ADC capacitor calibration, and section *11.5.1.4* describes the steps to enter the *ADC_ON* state without performing a calibration. ADC calibration is only necessary after an ADC reset, changing the reference, or changing environmental conditions such as temperature. For example, a device moving from an indoor environment to an outdoor environment might require a recalibration depending on application requirements. *Figure 11-3* shows the ADC operating modes state diagram. *Table 11-3* shows the configuration bits' state and the status bit's state for each operating mode.

Table 11-3: ADC Operating States

Instance	*ADC_CTRL0*.*resetb*	*ADC_CTRL0*.*bias_en*	*ADC_CTRL0*.*adc_en*	*ADC_STATUS*.*ready* (Status)
ADC_ON	1	1	1	1
ADC_NAP	1	1	0	0
ADC_SLEEP	1	0	0	0
ADC_RESET	0	0	0	0

Figure 11-3: ADC Operating Modes State Diagram



The ADC remains in the *ADC_RESET* state while the *ADC_CTRL0.resetb* field is 0. The ADC enters *ADC_SLEEP* when the *ADC_CTRL0.resetb* field is set to 1. *ADC_SLEEP* is a low-power mode with the bias regulator disabled.

Enabling the bias regulator transitions the ADC to *ADC_NAP* state. Setting *ADC_CTRL0.bias_en* to 1 turns on the bias regulator required for ADC conversions. The bias regulator requires approximately 500µs to warm up. There is no dedicated

status bit indicating the transition is complete, so the software must measure the required time. The ADC's sample rate should be configured with the ADC in the *ADC_NAP* state.

The peripheral enters the *ADC_ON* state when the *ADC_CTRL0.adc_en* field is set to 1. If the *ADC_CTRL0.skip_cal* field is 1, the device performs the ADC auto-calibration, which takes approximately 100ms to complete. After the auto-calibration is complete, or immediately if it is not performed, the peripheral enters the *ADC_ON* state. An ADC-ready event occurs, indicating conversions can begin. The *ADC_STATUS.ready* field remains 1 while in the *ADC_ON* state.

11.5.1 ADC Initialization

11.5.1.1 Entering ADC_SLEEP State

The ADC must be initialized before use. These steps are performed once and are not needed before every conversion. Analog inputs are usually dedicated and not dynamically switched with digital functions.

To initialize the ADC and enter *ADC_SLEEP*:

1. Clear *GCR_PCLKDIS0.adc* to 0 to enable the ADC peripheral clock.
2. Clear *ADC_CTRL0.resetb* to 0 to enter reset.
3. Select the ADC_SRC clock from *Table 11-2* and set it to the selected clock using the *ADC_CLKCTRL.clksel* field.
4. Select the ADC reference source:
   - External: *MCR_ADCCFG0.ext_ref* to 1.
   - Internal, 1.25V: Clear *MCR_ADCCFG0.ext_ref* to 0 and clear *MCR_ADCCFG0.ref_sel* to 0.
   - Internal, 2.048: Clear *MCR_ADCCFG0.ext_ref* to 0 and set *MCR_ADCCFG0.ref_sel* to 1.
5. If desired, enable the analog input buffer and input divider for the desired external channels, as shown in *Analog Input Buffer*. The analog input buffer can be enabled individually for each external analog input channel.
6. Configure the GPIO associated with the desired external channels as inputs in high impedance mode. Configure the alternate function mode, as indicated in *Table 11-1*.
7. Set the *ADC_CTRL0.resetb* to 1 to enter the *ADC_SLEEP* state.

11.5.1.2 Entering ADC_NAP State

After the ADC is in *ADC_SLEEP*, enter *ADC_NAP* state as follows:

1. Enable the ADC bias regulator by setting *ADC_CTRL0.bias_en* to 1.
2. Wait 500µs for the bias regulator to settle.
3. The ADC is now in the *ADC_NAP* state.

11.5.1.3 Entering ADC_ON State Using Calibration

Autocalibration can only be performed when the ADC is in *ADC_NAP*. The autocalibration settings remain loaded as long as the ADC does not enter *ADC_RESET*. Perform the following steps for calibration when the ADC is in *ADC_NAP*:

1. Clear the *ADC_CTRL0.skip_cal* bit to 0.
2. Configure the ADC SAMPLE_CLK using the *ADC_SAMPCLKCTRL.track_cnt* and *ADC_SAMPCLKCTRL.idle_cnt* fields, as described in *Clocks and Timing*.
3. Clear the ADC interrupt flags register by writing 0xFFFF FFFF to the *ADC_INTFL* register.
4. Load the reference trim values for the desired reference. See *ADC SFR Interface* for details.
5. Set the *ADC_CTRL0.adc_en* field to 1.
6. The calibration is complete, and the ADC enters the *ADC_ON* state when the *ADC_INTFL.ready* field reads 1.

Once the ADC is in the *ADC_ON* state, conversions can be started.

11.5.1.4 Entering ADC_ON State Skipping Calibration

If calibration is already performed, the calibration step can be skipped. Enter *ADC_ON* state without calibration by performing the following steps:

1. Set the *ADC_CTRL0.skip_cal* bit to 1.
2. Configure the ADC SAMPLE_CLK using the *ADC_SAMPCLKCTRL.track_cnt* and *ADC_SAMPCLKCTRL.idle_cnt* fields, as described in *Clocks and Timing*.
3. Clear the ADC interrupt flags register by writing 0xFFFF FFFF to the *ADC_INTFL* register.
4. Set the *ADC_CTRL0.adc_en* field to 1.
5. The ADC enters the *ADC_ON* state when the *ADC_INTFL.ready* field reads 1.

Once the ADC is in the *ADC_ON* state, conversions can be started.

11.6 ADC SFR Interface

The ADC supports loading of several configuration and trim values. Each reference includes specific trim values stored in the *FCR_ADCREFTRIM0*, *FCR_ADCREFTRIM1*, and *FCR_ADCREFTRIM2* registers. Additionally, the bias counter and wake-up counter are configurable to achieve optimum performance.

11.6.1 Determination of Bias and Wake-up Counter Settings

The ADC bias and wake-up are configurable to achieve optimum performance based on the sample rate of the ADC. The bias must be at least 500µs and the ADC wake-up timer must be at least 30µs to ensure optimum reference buffer settling requirements. The settings for the bias counter and wake-up counter are dependent on the configured ADC sample rate. *Table 11-4* shows the settings for the bias and wake-up counters and the resulting number of clock cycles each setting achieves. The following steps show how to determine the settings for the bias counter and wake-up counter for a sample rate of 250KSPS.

1. The bias counter must be 500µs, which results in $250KHz \times 500\mu s = 125$ cycles.
   a. Referring to *Table 11-4*, the closest bias counter setting to achieve at least 125 cycles is 5 (128 clock cycles).
2. The wake-up counter must be 30µs, which results in $250KHz \times 30\mu s = 7.5$ cycles.
   a. Referring to *Table 11-4*, the closest wake-up counter setting to achieve at least 7.5 cycles is 3 (8 clock cycles).

Table 11-4: Bias and Wake-up Clock Cycle Selection

Config Counter Setting	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bias Counter Clock Cycles	4	8	16	32	64	128	256	512	1024	1536	2048	2560	3072	3584	4094	4608
Wake-up Clock Cycles	2	4	6	8	10	12	14	16	20	24	28	32	36	40	44	48

11.6.2 Using the ADC SFR Interface to Load the Reference Trim, and Bias/Wake-up Counter Settings

Note: The loading of the reference trim values must be performed while the ADC is in the ADC NAP state and are only applied when the ADC enters the ON state using calibration. See Entering ADC_ON State Using Calibration.

11.6.3 1.25V Internal Reference Trim

Perform the following steps to load the trim values for the 1.25V internal reference.

1. Write address 0x0B to *ADC_SFRADDR*.
4. 2. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
3. Mask off the upper two bits of the data read by performing a bit-wise AND of the value read with 0xC0.
5. 4. Perform a bit-wise OR of the result of step 3 with *FCR_ADCREFTRIM0.vx2_tune*.
5. Write the byte from step 4 to the *ADC_SFRWRDATA* register.
6. Write address 0x0C to *ADC_SFRADDR*.
6. 7. Write the following byte to the *ADC_SFRWRDATA* register.
   a. Perform a bitwise OR of:
      1) *FCR_ADCREFTRIM2.iboost_1p25* shift left 7 and
      7. 2) *FCR_ADCREFTRIM0.vrefp*
8. Write address 0x0D to *ADC_SFRADDR*.
8. 9. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
10. Mask off the upper bit of the data read by performing a bit-wise AND of the value read with 0x80.
9. 11. Perform a bit-wise OR of the result of step 10 with *FCR_ADCREFTRIM0.vrefm.*
12. Write the byte from step 11 to the *ADC_SFRWRDATA* register.
13. Write address 0x0E to *ADC_SFRADDR*.
14. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
10. 15. Mask off bits 2 and 3 of the data read by performing a bit-wise AND of the value read with 0x0C.
16. Write the following byte to the *ADC_SFRWRDATA* register.
   a. Perform a bitwise OR of the data from step 15 and:
      1) *FCR_ADCREFTRIM2.idrv_1p25* shift left 4
      2) *FCR_ADCREFTRIM0.vcm*
17. Write address 0x05 to *ADC_SFRADDR*.
11. 18. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
19. Mask off bits 4 - 7 of the data read by performing a bit-wise AND of the value read with 0xF0.
12. 20. Write the following byte to the *ADC_SFRWRDATA* register.
   a. Perform a bitwise OR of the data from step 19 and the bias counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
21. Write address 0x06 to *ADC_SFRADDR*.
13. 22. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
23. Mask off bits 4 - 7 of the data read by performing a bit-wise AND of the value read with 0xF0.
24. Write the following byte to the *ADC_SFRWRDATA* register.
   a. Perform a bitwise OR of the data from step 23 and the calculated wake-up counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
25. Move the ADC into the ADC_ON state using calibration.

11.6.4 2.048V Internal Reference Trim

Perform the following steps to load the trim values for the 2.048V internal reference.

1. Write address 0x0B to *ADC_SFRADDR*.
2. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
3. Mask off the upper two bits of the data read by performing a bit-wise AND of the value read with 0xC0.
4. Perform a bit-wise OR of the result of step 3 with *FCR_ADCREFTRIM1*.*vx2_tune*.
5. Write the byte from step 4 to the *ADC_SFRWRDATA* register.
6. Write address 0x0C to *ADC_SFRADDR*.
7. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of:
        1) *FCR_ADCREFTRIM2*.*iboost_2p048* shift left 7, and
        2) *FCR_ADCREFTRIM1*.*vrefp.*
8. Write address 0x0D to *ADC_SFRADDR*.
9. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
10. Mask off the upper bit of the data read by performing a bit-wise AND of the value read with 0x80.
11. Perform a bit-wise OR of the result of step 10 with *FCR_ADCREFTRIM1*.*vrefm.*
12. Write the byte from step 11 to the *ADC_SFRWRDATA* register.
13. Write address 0x0E to *ADC_SFRADDR*.
14. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
15. Mask off bits 2 and 3 of the data read by performing a bit-wise AND of the value read with 0x0C.
16. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of the data from step 15 and:
        1) *FCR_ADCREFTRIM2*.*idrv_2p048* shift left 4
        2) *FCR_ADCREFTRIM1*.*vcm*
17. Write address 0x05 to *ADC_SFRADDR*.
18. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
19. Mask off bits 4 - 7 of the data read by performing a bit-wise AND of the value read with 0xF0.
20. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of the data from step 19 and the bias counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
21. Write address 0x06 to *ADC_SFRADDR*.
22. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
23. Mask off bits 4 - 7 of the data read by performing a bit-wise AND of the value read with 0xF0.
24. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of the data from step 23 and the calculated wake-up counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
25. Move the ADC into the ADC_ON state using calibration.

11.6.5 External Reference Trim

Perform the following steps to load the trim values for the external reference.

1. Write address 0x0B to *ADC_SFRADDR*.
2. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
3. Mask off the upper bit of the data read by performing a bit-wise AND of the value read with 0x80.
4. Perform a bit-wise OR of the result of step 3 with *FCR_ADCREFTRIM2*.*vx2_tune*.
5. Write the byte from step 4 to the SFR by writing it to the *ADC_SFRWRDATA* register.
6. Write address 0x0E to *ADC_SFRADDR*.
7. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
8. Write the following byte to the *ADC_SFRWRDATA* register.
   a. Perform a bitwise OR of the data from step 7 and *FCR_ADCREFTRIM2*.*vcm*
9. Write address 0x05 to *ADC_SFRADDR*.
10. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
11. Mask off bits 4 - 7 of the data read by performing a bit-wise AND of the value read with 0xF0.
12. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of the data from step 11 and the bias counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
13. Write address 0x06 to *ADC_SFRADDR*.
14. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
15. Mask off bits 4 - 7 of the data read by performing a bit-wise AND of the value read with 0xF0.
16. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of the data from step 16 and the calculated wake-up counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
17. Move the ADC into the ADC_ON state using calibration.

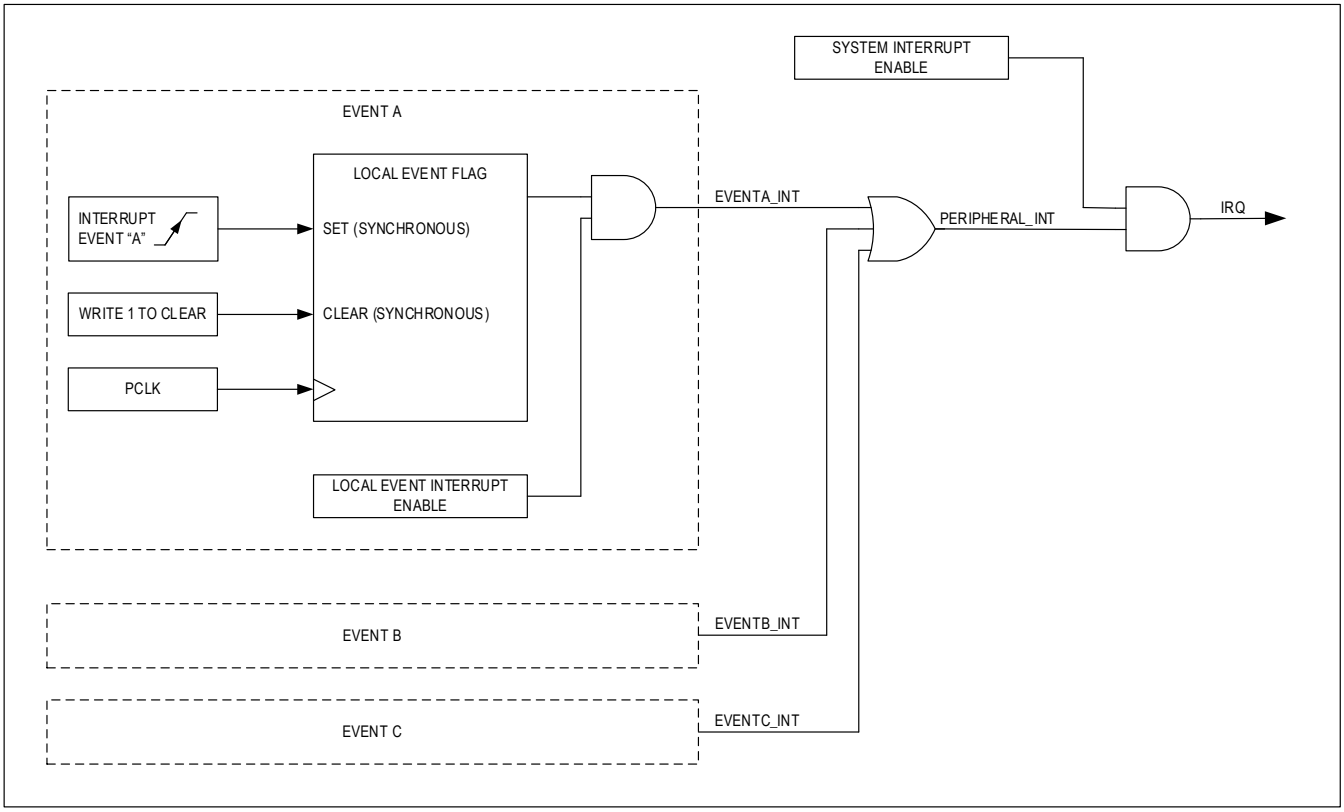
11.7 Interrupts

Multiple interrupt events are supported. Each event has a flag and interrupt enable field in the peripheral's register set unless specified otherwise. The event flag is "edge-triggered" and set when the event occurs. Further occurrences of the event do not cause any additional effect if the flag is set to 1. The interrupt signal from the event is active whenever the flag and enable fields are both set to 1. An event flag should always be cleared by writing 1 to the flag's bit position before setting its interrupt enable field.

All the interrupt signals from local events in a peripheral are OR'd together to create a peripheral interrupt for the NVIC. Some peripherals can further qualify the generation of the interrupt with one or more higher-level system interrupt enables.

Figure 11-4 is a functional diagram showing this relationship.

Figure 11-4: Interrupt Event Signal Generation



Clear a local event flag by writing a 1 to the flag. Always clear a local event flag before setting the corresponding interrupt enable field.

The interrupt events supported are listed in *Table 11-5*.

Table 11-5: MAX32690 Interrupt Events

Event	Description	Interrupt Flag	Interrupt Enable
Receive FIFO Threshold	*ADC_STATUS*. *fifo_level* > *ADC_FIFODMACTRL*.*thresh*.	*ADC_INTFL*.*fifo_lvl*	*ADC_INTEN*.*fifo_lvl*
Receive FIFO Overflow	Hardware FIFO write when *ADC_STATUS*.*fifo_level*= 0b111.	*ADC_INTFL*.*fifo_ofl*	*ADC_INTEN*.*fifo_ofl*
Receive FIFO Underflow	Read from FIFO when *ADC_STATUS*.*fifo_level* = 0	*ADC_INTFL*.*fifo_ufl*	*ADC_INTEN*.*fifo_ufl*
Data Clipped	An ADC measurement is clipped.	*ADC_INTFL*.*clipped*	*ADC_INTEN*.*clipped*
Conversion Sequence Done	A continuous conversion sequence completed while *ADC_CTRL1*.*start* is 1 or a single conversion sequence completed.	*ADC_INTFL*.*conv_done*	*ADC_INTEN*.*conv_done*
Conversion Sequence Complete	A continuous or single conversion sequence is finished.	*ADC_INTFL*.*seq_done*	*ADC_INTEN*.*seq_done*
Conversion Sequence Started	A continuous or single conversion sequence has started. This field can be used to tell when a hardware trigger occurred.	*ADC_INTFL*.*seq_started*	*ADC_INTEN*.*seq_started*
Start Bit Set	*ADC_CTRL1*.*start* transitioned from 0 to 1.	*ADC_INTFL*.*start_det*	*ADC_INTEN*.*start_det*
Conversion Sequence Abort	*ADC_CTRL1*.*start* transitioned from 1 to 0 before a conversion started.	*ADC_INTFL*.*abort*	*ADC_INTEN*.*abort*
ADC Ready	ADC transitioned to the *ADC_ON* state.	*ADC_INTFL*.*ready*	*ADC_INTEN*.*ready*

11.8 FIFO Operation

Measurement results are pushed onto the 16-word FIFO. Access the FIFO by reading the *ADC_DATA* register. Software must be sure to read the FIFO often enough to prevent data from being lost.

The current level of the FIFO is read from *ADC_STATUS*.*fifo_level.* The same register also contains empty and full status flags for the FIFO. Multiple FIFO events are supported.

- A FIFO threshold event occurs when *ADC_STATUS*.*fifo_level* exceeds the value *ADC_FIFODMACTRL*.*thresh*. This event is an indication that data should be read from the FIFO soon or can be lost.
- A FIFO overflow event occurs when a measurement is loaded into the FIFO when *ADC_STATUS*.*fifo_level* equals 7. Previous data in the FIFO is overwritten. It is possible to use the channel ID field in the *ADC_DATA* register to determine which measurement results are overwritten. The FIFO should be flushed and the current conversion sequence restarted.
- A FIFO underflow event occurs when the FIFO is read, and no data is in the FIFO.

11.9 Averaging

The ADC can take multiple measurements of a channel and average the results. Averaging is enabled when the *ADC_CTRL1*.*avg* field is set to a non-zero value. Each slot in the conversion is sampled 2^N times, where N is the *ADC_CTRL1*.*avg* value. The results are then averaged and reported in the slot. The averaging setting applies to all measured channels. A clipped measurement of any of the samples sets the clipped status field for the averaged result.

Note: The averaging is applied equally to all slots. Setting a large averaging value can result in a long conversion time for a sequence to complete if multiple channels are enabled.

11.10 Conversion Results

The results and status information are read from the *ADC_DATA* register. The selection of the format of the *ADC_DATA* register is determined using the *ADC_FIFODMACTRL*.*data_format* field. Each of the format options is shown in *Table 11-6*. A visual representation of the corresponding format of the *ADC_DATA* register all input channels are shown in *Figure 11-5*.

In processed modes, the status information includes:

- A channel identifier (*ADC_DATA*.*chan*) to assist in identifying the channel associated with the data. Although the channel is known when reading the slot, this helps identify data later if saved to memory.
- A clipped status (*ADC_DATA*.*clipped*) field indicating if the result is beyond the ADC limits (either positive or negative). The result is marked clipped for an averaged measurement if any of the samples are clipped.
- The validity of the data (*ADC_DATA*.*invalid*). Data is marked as invalid if clipped, has an invalid channel assignment, or the channel is not ready.

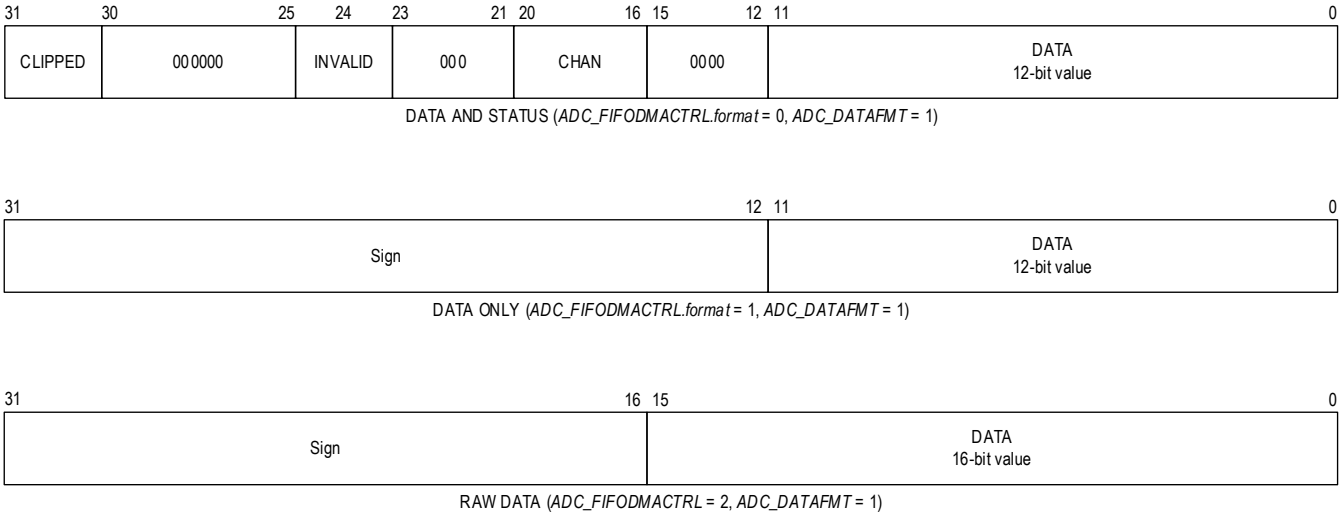
The result formatting options are shown in *Table 11-6*.

Table 11-6: ADC_DATA Register Result Formatting

Mode	*ADC_FIFODMACTRL*.*data_format*	Format code	Channel ID	Clipped	Invalid Flag	Data Format
Single-ended	0	Data and Status	Yes	Yes	Yes	12-bit unsigned
	1	Data Only	No	*ADC_CHSTATUS*	No	12-bit unsigned
	2	Raw Data Only	-	*ADC_CHSTATUS*	Yes	16-bit signed 2's complement bit 16 is the sign bit

The information structure depends on the channel mode (single-ended or differential) and the selected data format, as shown in *Figure 11-4*.

Figure 11-5: ADC Result Formats (Single-Ended)



11.11 Conversions

11.11.1 Conversion Sequence Triggers

A conversion sequence is initiated by either a software or hardware trigger. This flexibility allows manual or on-demand measurements using a timer peripheral or an external GPIO. *Table 11-7* lists the hardware triggers available to start a conversion sequence.

Table 11-7: MAX32690 Hardware Conversion Triggers

ADC_CTRL1.trig_sel	Source
0	TMR0 output
1	TMR1 output
2	TMR2 output
3	TMR3 output
4	Reserved
5	AIN_TRIG_B[1]
6	AIN_TRIG_D[1]
7	Reserved
1. Refer to the device data sheet's pin description table for alternate function assignments. Not all alternate functions are available on all packages.	

A software-triggered conversion sequence can run once and stop (single conversion sequence) or continuously (continuous conversion sequence). A conversion sequence begins when the software changes the *ADC_CTRL1.start* field from 0 to 1. Conversion sequences run until all slots are completed for both a continuous or single sequence.

A software-triggered continuous sequence converts all slots and then repeats the process, with a programmable delay between sequences, as long as the *ADC_CTRL1.start* field is set to 1. Setting *ADC_CTRL1.start* to 0 during an active continuous conversion sequence stops the sequence at the completion of the active sequence.

A hardware-triggered conversion sequence starts when the selected trigger becomes active. Only one of the hardware triggers, shown in *Table 11-7*, can be selected for the conversion sequence.

The hardware trigger is armed when the software changes *ADC_CTRL1.start* from a 0 to a 1. The device waits until the trigger event occurs, performs one conversion sequence, and then idles until the trigger event occurs again or software clears the *ADC_CTRL1.start* field to 0.

The hardware trigger source must be running and properly configured to generate the trigger signal. Trigger sources are edge-triggered; the event is only recognized if a GPIO pin transitions from low to high or a timer output signal transitions

from inactive to active. As a result, software must clear the GPIO or timer output each time before the hardware trigger event occurs for the trigger to be recognized. See *Alternate Function* in the GPIO chapter for details on configuring port pin alternate functions.

Table 11-8: Conversion Sequence Configurations

Conversion Sequence Type	Sequence Start	*ADC_CTRL1.cnv_mode*	*ADC_CTRL1.trig_mode*
Software-Triggered, continuous	*ADC_CTRL1.start* 0 → 1 Or *ADC_CTRL1.start* 0 → 1 and *ADC_INTFL.seq_done* 0 → 1 after delay	1	0
Software-Triggered, single conversion sequence	*ADC_CTRL1*.start 0 → 1	0	0
Hardware-Triggered, continuous	*ADC_CTRL1.start* 0 → 1 (armed) Trigger event Or *ADC_CTRL1.start* 0 → 1 and *ADC_INTFL.seq_done* 0 → 1 after delay	1	1
Hardware-Triggered, single conversion sequence	*ADC_CTRL1.start* 0 → 1 (armed) Trigger event	0	1

11.11.2 Single Conversion Sequences

11.11.2.1 Software Triggered

To perform a software-triggered single conversion sequence:

1. Configure the ADC and enter the *ADC_ON* state, as described in *Operating Modes*.
2. Clear the *ADC_CTRL1.trig_mode* field to 0 to select software triggering.
3. Set *ADC_CTRL1.cnv_mode* to 0 to select a single conversion sequence.
4. Select the number of channels to convert by setting the *ADC_CTRL1.num_slots* to the number of channels minus 1.
   a. As an example, set *ADC_CTRL1.num_slots* to 4 to perform a single conversion on 5 channels.
5. Set the desired channels for the conversion using the *ADC_CHSEL4*:*ADC_CHSEL0* registers slot fields.
   a. As an example, to perform a single conversion sequence on channels 6, 3, 8, 4, and 2 (*ADC_CTRL1.num_slots* = 4), set the channel select fields as follows:
      i.) *ADC_CHSEL0.slot0_id* = 6
      ii.) *ADC_CHSEL0.slot1_id* = 3
      iii.) *ADC_CHSEL0.slot2_id* = 8
      iv.) *ADC_CHSEL0.slot3_id* = 4
      v.) *ADC_CHSEL1.slot4_id* = 2
6. Configure *ADC_CTRL1.avg* to the desired number of samples to average.
   a. Set this field to 0 for a single conversion per channel. See *Averaging* for details on sample averaging.
7. Set the data format for the conversion results using the *ADC_FIFODMACTRL.data_format* field. See *Conversion Results* for details.
8. Clear the interrupt flags by writing 0xFFFF FFFF to the *ADC_INTFL* register.
9. Set *ADC_CTRL1.start* to 1 to start the conversion sequence. The conversion sequence starts immediately.

At the end of a software-triggered single conversion sequence:

- Hardware sets *ADC_INTFL.conv_done* to 1.

Hardware sets *ADC_INTFL.seq_done* to 1, indicating a sequence done event occurred.

- Software should set *ADC_CTRL1.start* to 0 to prevent additional conversions.

The converted data is available in the *ADC_DATA* register. See *FIFO Operation* for details on the FIFO.

11.11.2.2 Hardware-Triggered

Before performing a hardware-triggered conversion, a single software-triggered conversion on any channel should be performed. If a software-triggered conversion is not performed first, the initial hardware-triggered conversion occurs if the external trigger is high. Subsequent hardware-triggered conversions occur on the rising edge of the selected trigger.

Perform a hardware-triggered single conversion sequence using the following steps:

1. Configure the ADC and enter the *ADC_ON* state, as described in *Operating Modes*.
2. Clear the *ADC_CTRL1.start* field to 0.
3. Set *ADC_CTRL1.trig_mode* to 1 to select hardware triggering.
4. Configure *ADC_CTRL1.trig_sel* for the desired hardware trigger. See *Table 11-7* for details of available hardware triggers.
5. Set *ADC_CTRL1.cnv_mode* to 0 to select a single conversion sequence.
6. Configure the selected hardware trigger.
7. Select the number of channels to convert by setting the *ADC_CTRL1.num_slots* to the number of channels minus 1.
   a. As an example, set *ADC_CTRL1.num_slots* to 1 to perform a single conversion on 2 channels.
8. Set the desired channels for the conversion using the *ADC_CHSEL4*:*ADC_CHSEL0* registers slot fields.
   a. As an example, to perform a single conversion sequence on channels 11 and 12 (*ADC_CTRL1.num_slots* = 1), set the channel select fields as follows:
      i.) *ADC_CHSEL0.slot0_id* = 11
      ii.) *ADC_CHSEL0.slot1_id* = 12
9. Configure *ADC_CTRL1.avg* to the desired number of samples to average.
   a. Set this field to 0 for a single conversion per channel. See *Averaging* for details on sample averaging.
10. Set the data format for the conversion results using the *ADC_FIFODMACTRL.data_format* field. See *Conversion Results* for details.
14. 11. Clear the interrupt flags by writing 0xFFFF FFFF to the *ADC_INTFL* register.
12. Set *ADC_CTRL1.start* to 1 to arm the conversion sequence. The conversion sequence begins when the hardware trigger is activated.
13. When the sequence is triggered, hardware sets the *ADC_INTFL.seq_started* field to 1.

At the end of a hardware-triggered single conversion sequence:

- Hardware sets the *ADC_INTFL.seq_done* field to 1, indicating a sequence done event occurred.
- Hardware sets the *ADC_INTFL.conv_done* field to 1 and does not perform another conversion sequence.
- Software should set *ADC_CTRL1.start* to 0 to prevent additional conversions.
- The converted data is available in the *ADC_DATA* register. See *FIFO Operation* for details on the FIFO.

11.11.3 Continuous Conversion Sequences

11.11.3.1 Software-Triggered, Continuous Conversion Sequence

To configure the ADC for a software-triggered continuous conversion sequence:

1. Configure the ADC and enter the *ADC_ON* state, as described in *Operating Modes*.
2. Clear the *ADC_CTRL1*.*trig_mode* field to 0 to select software triggering.
3. Set *ADC_CTRL1*.*cnv_mode* to 1 to select continuous conversion mode.
4. Select the number of channels to convert by setting the *ADC_CTRL1*.*num_slots* to the number of channels minus 1.
5. Set the desired channels for the conversion using the *ADC_CHSEL4*:*ADC_CHSEL0* registers slot fields.
6. Configure *ADC_CTRL1*.*avg* to the desired number of samples to average.
   a. Set this field to 0 for a single conversion per channel. See *Averaging* for details on sample averaging.
7. Set the data format for the conversion results using the *ADC_FIFODMACTRL*.*data_format* field. See *Conversion Results* for details.
8. Clear the interrupt flags by writing 0xFFFF FFFF to the *ADC_INTFL* register.
9. If a delay between continuous conversion sequences is desired, set the number of SAMPLE_CLKs to delay using the *ADC_RESTART*.*cnt* field.
10. Set *ADC_CTRL1*.*start* to 1 to activate the conversion sequence. The conversion sequence starts immediately.
11. Software should clear *ADC_CTRL1*.*start* to stop a continuous conversion sequence, when desired.

At the end of each continuous conversion sequence:

- Hardware sets the *ADC_INTFL*.*seq_done* field to 1, indicating a sequence completed.
  If *ADC_CTRL1*.*start* remains set to 1, the device idles for the number of sample periods specified in *ADC_RESTART*.*cnt* before repeating the conversion sequence.
- If *ADC_CTRL1*.*start* is set to 0, hardware sets *ADC_INTFL*.*conv_done* to 1 and does not perform another conversion sequence.

11.11.3.2 Hardware-Triggered, Continuous Conversion Sequence

To perform a hardware-triggered continuous conversion sequence:

1. Configure the ADC and enter the *ADC_ON* state, as described in *Operating Modes*.
2. Set *ADC_CTRL1*.*cnv_mode* to 1 to select continuous conversion mode.
3. Set *ADC_CTRL1*.*trig_mode* to 1 to select hardware triggering.
4. Configure the *ADC_CTRL1*.*trig_sel* field for the desired hardware trigger. See *Table 11-7* for a list of hardware triggers.
5. Configure the selected hardware trigger. See *Timers (TMR/LPTMR)* for timer configuration and *Alternate Function* in the GPIO chapter for details on configuring alternate functions.
6. Select the number of channels to convert by setting the *ADC_CTRL1*.*num_slots* to the number of channels minus 1.
7. Set the desired channels for the conversion using the *ADC_CHSEL4*:*ADC_CHSEL0* registers slot fields.
8. Configure *ADC_CTRL1*.*avg* to the desired sample averaging.
    a. Setting this field to 0 disables averaging. Set this field to 1 for a single conversion per channel. If this field is set to greater than 1, each channel selected is converted 2^{avg} number of times and averaged before moving to the next channel.
9. Set the data format for the conversion results using the *ADC_FIFODMACTRL*.*data_format* field. See *Conversion Results* for details.
10. Clear the interrupt flags by writing 0xFFFF FFFF to the *ADC_INTFL* register.
11. If a delay between continuous conversion sequences is desired, set the number of SAMPLE_CLKs to delay using the *ADC_RESTART*.*cnt* field.
12. Set *ADC_CTRL1*.*start* to 1 to arm the conversion sequence.
13. When the sequence is triggered, the hardware sets *ADC_INTFL*.*seq_started* to 1. Continuous sequences can be stopped by clearing the *ADC_CTRL1*.*start* field to 0.

At the end of a continuous conversion sequence:

- Hardware sets *ADC_INTFL*.*seq_done* to 1, indicating a sequence done event occurred.

If *ADC_CTRL1*.*start* is set to 1:

- The device idles for the number of sample periods specified in the *ADC_RESTART*.*cnt* field and then starts another conversion sequence.

- If *ADC_CTRL1*.*start* is set to 0:
    - The hardware sets the *ADC_INTFL*.*conv_done* field to 1 and does not perform another conversion sequence.

11.12 Low-Power Analog Wake-Up Comparators

Two unique, differential analog comparators can be used as wake-up sources for the device. These are simple op-amps, which generate an internal digital signal whenever the positive input is above the negative input.

Each analog comparator supports multiple analog inputs for the positive and negative inputs. See *Table 11-9* and *Table 11-10* for each comparator's input options and selection. Refer to the device data sheet's pin description table for alternate function assignment.

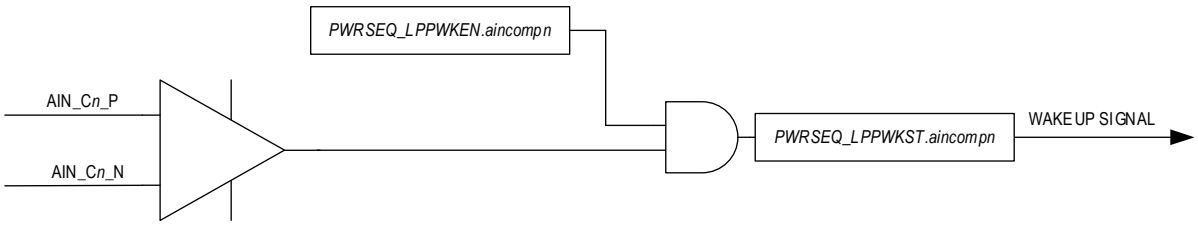
Table 11-9: MAX32690 Analog Comparator 0 Input Selection

Input	Input Signal	Selection
Positive	None	*MCR_CMP_CTRL.psel_comp0* = 0
	AIN4	*MCR_CMP_CTRL.psel_comp0* = 1
	AIN5	*MCR_CMP_CTRL.psel_comp0* = 2
	AIN6	*MCR_CMP_CTRL.psel_comp0* = 4
	AIN7	*MCR_CMP_CTRL.psel_comp0* = 8
Negative	None	*MCR_CMP_CTRL.nsel_comp0* = 0
	AIN0	*MCR_CMP_CTRL.nsel_comp0* = 1
	AIN1	*MCR_CMP_CTRL.nsel_comp0* = 2
	AIN2	*MCR_CMP_CTRL.nsel_comp0* = 4
	AIN3	*MCR_CMP_CTRL.nsel_comp0* = 8

Table 11-10: MAX32690 Analog Comparator 1 Input Selection

Input	Input Signal	Selection
Positive	None	*MCR_CMP_CTRL.psel_comp1* = 0
	AIN4	*MCR_CMP_CTRL.psel_comp1* = 1
	AIN5	*MCR_CMP_CTRL.psel_comp1* = 2
	AIN6	*MCR_CMP_CTRL.psel_comp1* = 4
	AIN7	*MCR_CMP_CTRL.psel_comp1* = 8
Negative	None	*MCR_CMP_CTRL.nsel_comp1* = 0
	AIN0	*MCR_CMP_CTRL.nsel_comp1* = 1
	AIN1	*MCR_CMP_CTRL.nsel_comp1* = 2
	AIN2	*MCR_CMP_CTRL.nsel_comp1* = 4
	AIN3	*MCR_CMP_CTRL.nsel_comp1* = 8

Figure 11-6: Analog Wake-up Comparators



The comparator status field dynamically shows the comparator output when both the corresponding positive and GPIO are configured for the appropriate alternate function. When enabled, the transition of the digital signal from 0 to 1 generates a wake-up event. The wake-up comparators function independently from the ADC converter circuitry and are not affected by the ADC operating states, settings, or enable status.

The control and status fields are listed in *Table 11-11*.

Table 11-11: MAX32690 Analog Wake-Up Comparator Fields

Comparator	Wake-up Flag	Wake-up Enable	Comparator Output
AINCOMP0	*PWRSEQ_LPPWST.aincomp0*	*PWRSEQ_LPPWEN.aincomp0*	*PWRSEQ_LPPWST.aincomp0_out*

Configure the comparators as follows:

1. Select the comparator's inputs. See *Table 11-9* and *Table 11-10* for details on selecting each comparator's inputs.
2. Enable the selected input's alternate function for the GPIO. See *Alternate Function* in the GPIO chapter for details. Refer to the device data sheet alternate function table for pin assignments.

11.13 Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* for an explanation of each field's read and write access. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific reset.

Table 11-12: ADC Register Summary

Offset	Register	Description
[0x0000]	*ADC_CTRL0*	ADC Control 0 Register
[0x0004]	*ADC_CTRL1*	ADC Control 1 Register
[0x0008]	*ADC_CLKCTRL*	ADC Clock Control Register
[0x000C]	*ADC_SAMPCLKCTRL*	ADC Sample Clock Control Register
[0x0010]	*ADC_CHSEL0*	ADC Channel Select 0 Register
[0X0014]	*ADC_CHSEL1*	ADC Channel Select 1 Register
[0x0018]	*ADC_CHSEL2*	ADC Channel Select 2 Register
[0x001C]	*ADC_CHSEL3*	ADC Channel Select 3 Register
[0x0020]	*ADC_CHSEL4*	ADC Channel Select 4 Register
[0x0030]	*ADC_RESTART*	ADC Conversion Restart Delay
[0x003C]	*ADC_DATAFMT*	ADC Data Format Register
[0x0040]	*ADC_FIFODMACTRL*	ADC FIFO and DMA Control Register
[0x0044]	*ADC_DATA*	ADC FIFO Register
[0x0048]	*ADC_STATUS*	ADC Status Register
[0x004C]	*ADC_CHSTATUS*	ADC Channel Status Register
[0x0050]	*ADC_INTEN*	ADC Interrupt Enable Register
[0x0054]	*ADC_INTFL*	ADC Interrupt Flags Register
[0x0060]	*ADC_SFRADDROFFSET*	ADC Address Offset Register
[0x0064]	*ADC_SFRADDR*	ADC SFR Address Register
[0x0068]	*ADC_SFRWRDATA*	ADC SFR Write Data Register
[0x006C]	*ADC_SFRRDDATA*	ADC SFR Read Data Register
[0x0070]	*ADC_SFRSTATUS*	ADC SFR Status Register

11.13.1 Register Details

Table 11-13: ADC Control 0 Register

ADC Control 0				ADC_CTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:5	-	RO	0	**Reserved**	
4	resetb	R/W	0	**Reset ADC** 0: ADC is in *ADC_RESET* 1: Not in *ADC_RESET*	
3	chop_force	R/W	0	**Input Chopping** 0: Disabled 1: Enabled	
2	skip_cal	R/W	0	**Skip Calibration** Set this field to 1 to skip automatic calibration before starting a conversion sequence. 0: Perform automatic calibration 1: Skip automatic calibration	
1	bias_en	R/W	0	**Bias Enable** 0: Disabled 1: Enabled	

ADC Control 0			ADC_CTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
0	adc_en	R/W	0	**ADC Enable** 0: Disabled 1: Enabled

Table 11-14: ADC Control 1 Register

ADC Control 1			ADC_CTRL1	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0	**Reserved**
20:16	num_slots	R/W	0	**Number of Slots Enabled per Conversion Sequence** 0: 1 slot 1: 2 slots 2: 3 slots … : … 19: 20 slots 20 - 31: Reserved
15:11	-	RO	0	**Reserved**
10:8	avg	R/W	0	**Sample Averaging** 0: No averaging All other values: Average 2^{avg} samples on each channel before reporting the results.
7	-	RO	0	**Reserved**
6:4	trig_sel	R/W	0	**Hardware Trigger Source** See *Table 11-7* for field settings.
3	samp_ck_off	R/W	0	**Sample Clock Control** 0: Continuous sample clock 1: Sample clock runs only while a channel is being measured
2	cnv_mode	R/W	0	**Conversion Mode** 0: Single conversion sequence 1: Continuous conversion sequence
1	trig_mode	R/W	0	**Trigger Mode Control** 0: Software trigger 1: Hardware trigger
0	start	R/W	0	**Conversion Start** In software-triggered mode (*ADC_CTRL1.trig_mode* = 0), a conversion sequence starts immediately when this field is set to 1. After a sequence (*ADC_INTFL.seq_done* = 1) or when a conversion is complete (*ADC_INTFL.conv_done* = 1), software should set this field to 0. In hardware-triggered mode (*ADC_CTRL1.trig_mode* = 1), a conversion sequence is armed when this field is set to 1. A conversion sequence starts when the selected hardware trigger becomes active. Any time after the hardware-triggered conversion is started (*ADC_INTFL.seq_started* = 1), software should set this field to 0 to prevent subsequent conversion sequences from starting if desired. See *Conversions* for details. 0: Conversion complete 1: Start a conversion sequence or arm the ADC to start a hardware trigger.

Table 11-15: ADC Clock Control Register

ADC Clock Control			ADC_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**

ADC Clock Control			ADC_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
6:4	clkdiv	R/W	3	**Clock Divider** See *Analog Input Buffer* for details on determining the required setting for this field. The maximum SAR_CLK frequency is 25MHz. 0: Divide by 2 1: Divide by 4 2: Divide by 8 3: Divide by 16 4-7: Divide by 1
3:2	-	RO	0	**Reserved**
1:0	clksel	R/W1C	0	**Clock Source** This field selects the ADC peripheral clock. See *Table 11-2* for available clock sources.

Table 11-16: ADC Sample Clock Control Register

Sample Clock Control Register			ADC_SAMPCLKCTRL	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	idle_cnt	R/W	0	**Sample Clock Hold Time** The number of cycles to add to the minimum hold time. See *Analog Input Buffer* for details on determining the required setting for this field to achieve the desired sample rate.
15:10	-	RO	0	**Reserved**
9:0	track_cnt	R/W	0	**Sample Clock Track Time** The number of cycles to add to the minimum track time. See *Analog Input Buffer* for details on determining the required setting for this field to achieve the desired sample rate.

Table 11-17: ADC Channel Select 0 Register

ADC Channel Select 0			ADC_CHSEL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	0	**Reserved**
28:24	slot3_id	R/W	0	**Slot 3 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
23:21	-	RO	0	**Reserved**
20:16	slot2_id	R/W	0	**Slot 2 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
15:13	-	RO	0	**Reserved**
12:8	slot1_id	R/W	0	**Slot 1 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
7:5	-	RO	0	**Reserved**
4:0	slot0_id	R/W	0	**Slot 0 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.

Table 11-18: ADC Channel Select 1 Register

ADC Channel Select 1			ADC_CHSEL1	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	0	**Reserved**
28:24	slot7_id	R/W	0	**Slot 7 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
23:21	-	RO	0	**Reserved**
20:16	slot6_id	R/W	0	**Slot 6 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
15:13	-	RO	0	**Reserved**

ADC Channel Select 1			ADC_CHSEL1	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
12:8	slot5_id	R/W	0	**Slot 5 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
7:5	-	RO	0	**Reserved**
4:0	slot4_id	R/W	0	**Slot 4 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.

Table 11-19: ADC Channel Select 2 Register

ADC Channel Select 2			ADC_CHSEL2	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	0	**Reserved**
28:24	slot11_id	R/W	0	**Slot 11 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
23:21	-	RO	0	**Reserved**
20:16	slot10_id	R/W	0	**Slot 10 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
15:13	-	RO	0	**Reserved**
12:8	slot9_id	R/W	0	**Slot 9 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
7:5	-	RO	0	**Reserved**
4:0	slot8_id	R/W	0	**Slot 8 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.

Table 11-20: ADC Channel Select 3 Register

ADC Channel Select 3			ADC_CHSEL3	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	0	**Reserved**
28:24	slot15_id	R/W	0	**Slot 15 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
23:21	-	RO	0	**Reserved**
20:16	slot14_id	R/W	0	**Slot 14 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
15:13	-	RO	0	**Reserved**
12:8	slot13_id	R/W	0	**Slot 13 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
7:5	-	RO	0	**Reserved**
4:0	slot12_id	R/W	0	**Slot 12 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.

Table 11-21: ADC Channel Select 4 Register

ADC Channel Select 4			ADC_CHSEL4	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	0	**Reserved**
28:24	slot19_id	R/W	0	**Slot 19 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
23:21	-	RO	0	**Reserved**
20:16	slot18_id	R/W	0	**Slot 18 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
15:13	-	RO	0	**Reserved**

ADC Channel Select 4			ADC_CHSEL4	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
12:8	slot17_id	R/W	0	**Slot 17 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
7:5	-	RO	0	**Reserved**
4:0	slot16_id	R/W	0	**Slot 16 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.

Table 11-22: ADC Restart Count Register

ADC Restart Count			ADC_RESTART	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	cnt	R/W	0	**Sample Delay Before Continuous Conversion Restart** The number of SAMPLE_CLK periods to delay before restarting a continuous mode conversion sequence.

Table 11-23: ADC Data Format Register

ADC Data Format			ADC_DATAFMT	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	mode	DNM	0x0000 DFFF	**Channel Format** This field defines the data format of each channel. Each bit position corresponds to a specific channel number, i.e., *ADC_DATAFMT.mode[0]* is the data format for channel 0, *ADC_DATAFMT.mode[1]* is the data format for channel 1. Do not change this register from its default value. All channels operate in single-ended mode. Bit positions corresponding to unimplemented channels are ignored. 1: Single-ended mode

Table 11-24: ADC FIFO and DMA Control Register

ADC FIFO and DMA Control			ADC_FIFODMACTRL	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:8	thresh	R/W	0	**FIFO and DMA Threshold** When the number of words in the FIFO exceeds the threshold, a DMA request is triggered and, the *ADC_INTFL.fifo_lvl* interrupt flag is set. Valid settings for this field are 0 to 16.
7:4	-	RO	0	**Reserved**
3:2	format	R/W	0	**FIFO Data Format** 0b00: Data and status (12 bits processed plus status fields) 0b01: Data only (12 bits processed) 0b10: Raw Data Only (18-bit raw data) 0b11: Reserved
1	flush	W1	0	**FIFO Flush** Write 1 to flush the FIFO. This bit always reads 0. 0: N/A 1: Flush FIFO.
0	dma_en	R/W	0	**DMA Enable** 0: Disabled 1: Enabled

Table 11-25: ADC Data Register

ADC Data			ADC_DATA	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31	clipped	R	1	**Clipped** This field is set if the ADC sample or samples is clipped.
30:25	-	RO	0	**Reserved**
24	invalid	R	1	**Invalid Flag** This field is set if the data is invalid, e.g., reading from an empty FIFO or reading an invalid channel.
23:21	0	RO	0	**Reserved**
20:16	chan	R	0x1F	**Channel Identifier** This field is the channel identifier associated with the *ADC_DATA*.*data* field.
15:0	data	R/W	0xFFFF	**Data** The format of this data is configurable. See *Conversion Results* for details.

Table 11-26: ADC Status Register

ADC Status			ADC_STATUS	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:8	fifo_level	R	0	**FIFO Level** This field returns the number of words available to read from the FIFO. *Note: Valid values of this field are 0 to 16.*
7:3	-	RO	0	**Reserved**
2	full	R	0	**FIFO Full** 0: FIFO not full. 1: FIFO full.
1	empty	R	1	**FIFO Empty** 0: FIFO not empty. 1: FIFO empty.
0	ready	R	0	**ADC Ready** 0: ADC is not in *ADC_ON* state. 1: ADC is in *ADC_ON* state.

Table 11-27: ADC Channel Status Register

ADC Channel Status			ADC_CHSTATUS	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	clipped	W1C	0	**Clipped Data** This field identifies channels that the conversion is clipped. Each bit position corresponds to a specific channel number, i.e., clipped[0] is the clipped status for channel 0, clipped[1] is the clipped status for channel 1, clipped[14] is the clipped status for channel 14. Once a clipped conversion occurs, the bit remains set until cleared by software. 0: Not clipped. 1: Clipped.

Table 11-28: ADC Interrupt Enable Register

ADC Interrupt Enable			ADC_INTEN	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved**
10	fifo_ofl	W1C	0	**FIFO Overflow Event Interrupt Enable** 0: Disabled. 1: Enabled.

ADC Interrupt Enable			ADC_INTEN	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
9	fifo_ufl	W1C	0	**FIFO Underflow Event Interrupt Enable** 0: Disabled. 1: Enabled.
8	fifo_lvl	W1C	0	**FIFO Level Event Interrupt Enable** 0: Disabled. 1: Enabled.
7	clipped	W1C	0	**Data Clipped Event Interrupt Enable** 0: Disabled. 1: Enabled.
6	conv_done	W1C	0	**Conversion Done Event Interrupt Enable** 0: Disabled. 1: Enabled.
5	seq_done	W1C	0	**Sequence Done Event Interrupt Enable** 0: Disabled. 1: Enabled.
4	seq_started	W1C	0	**Sequence Started Event Interrupt Enable** 0: Disabled. 1: Enabled.
3	start_det	W1C	0	**Command Start Event Interrupt Enable** 0: Disabled. 1: Enabled.
2	abort	W1C	0	**Command Aborted Event Interrupt Enable** 0: Disabled. 1: Enabled.
1	-	RO	0	**Reserved**
0	ready	W1C	0	**ADC Ready Event Interrupt Enable** 0: Disabled. 1: Enabled.

Table 11-29: ADC Interrupt Flags Register

ADC Interrupt Flags			ADC_INTFL	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved**
10	fifo_ofl	R/W	0	**FIFO Overflow Event** 0: Normal operation. 1: Event occurred.
9	fifo_ufl	R/W	0	**FIFO Underflow Event** 0: Normal operation. 1: Event occurred.
8	fifo_lvl	R/W	0	**FIFO Level Event** 0: Normal operation. 1: Event occurred.
7	clipped	R/W	0	**Data Clipped Event** 0: Normal operation. 1: Event occurred.
6	conv_done	R/W	0	**Conversion Done Event** 0: Normal operation. 1: Event occurred.
5	seq_done	R/W	0	**Sequence Done Event** 0: Normal operation. 1: Event occurred.
4	seq_started	R/W	0	**Sequence Started Event** 0: Normal operation. 1: Event occurred.
3	start_det	R/W	0	**Command Start Event** The conversion command started. 0: Normal operation. 1: Event occurred.
2	abort	R/W	0	**Command Aborted Event** The conversion command aborted before conversions are complete. 0: Normal operation. 1: Event occurred.
1	-	RO	0	**Reserved**
0	ready	R/W	0	**ADC Ready Event** 0: Normal operation. 1: Event occurred.

Table 11-30: ADC SFR Address Offset Register

ADC SFR Address Offset			ADC_SFRADDROFFSET	[0x0060]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	offset	R/W	0	**Base Address Offset for SFR Registers** See *ADC SFR Interface* for details.

Table 11-31: ADC SFR Address Register

ADC SFR Address			ADC_SFRADDR	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	addr	R/W	0	**SFR Configuration Address** See *ADC SFR Interface* for details.

Table 11-32: ADC SFR Write Data Register

ADC SFR Write Data			ADC_SFRWRDATA	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	data	R/W	0	**SRF Write Data** See *ADC SFR Interface* for details.

Table 11-33: ADC SFR Read Data Register

ADC SFR Read Data			ADC_SFRRDDATA	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0xFF	**Reserved**
7:0	data	R/W	0	**SFR Read Data** See *ADC SFR Interface* for details.

Table 11-34: ADC SFR Status Register

ADC SFR Status			ADC_SFRSTATUS	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	nack	R	0	**Last SRF Transaction Status** This field indicates if an error occurred during the last SFR transaction. It is cleared at the start of a write to *ADC_SFRADDR*, ADC_SFRWRDATA, or a read from the *ADC_SFRRDDATA* register.

12. HyperBus/Xccela External Memory Interface (HPB)

The HyperBus and Xccela Memory Controller interface is a high-speed, low-pin count interface for connecting the MAX32690 to one or more compatible external memory devices. The external HyperBus or Xccela Bus memory device is mapped into the MAX32690 memory space enabling direct code execution, data storage, or both.

The HyperBus and Xccela Bus interface is a double data rate (DDR) interface, using both the rising and falling edges of the clock to transfer data. The HPB is connected to the APB and the peripheral clock is set using the PCLK with a maximum data transfer rate of 120Mbps in DDR mode.

Data is transferred over a high-speed 8-bit data bus. Two chip select lines are supported by the HPB with only one chip select line active at a time. Data is latched for reads and writes using a read/write data strobe signal. HyperBus transfers are clocked using a differential clock while Xccela Bus transfers use a single clock.

Features of the HyperBus/Xccela Memory Controller Interface include:

- Master/slave system
- 120 MB/sec maximum data transfer rate
- DDR – two data transfers per clock cycle
- Transparent bus operation to the processor
- 16KB write-through cache
- Two chip selects for two memory ports
  - Can address two external memories, one at a time
- Each port supports memories up to 512MB
- Interfaces to HyperFlash, HyperRAM, and Xccela PSRAM
- Zero wait state burst-mode operation
- Low-power, half-sleep mode
  - Puts the external memory device into low-power mode while retaining memory contents
- Configurable timing parameters

All bus transactions are either a read or a write. Bus transactions, as well as read and write destinations and any required latency, are all slave device dependent. Read and write destinations are either in the memory space or the configuration register space of the target memory device. Refer to the slave device data sheet for specific timing diagrams, latency, and read/write memory addresses.

HyperBus memory devices are 16-bit word addressed, so all addresses must be on 16-bit boundaries. Xccela memory devices can be byte-addressed. For both protocols, all 16-bit and 32-bit transfers must be on aligned boundaries.

12.1 HyperBus/Xccela Signal Descriptions

Table 12-1: HyperBus, Xccela Bus Pin Mapping and Signal Descriptions

Function	HyperBus Pin Name	Xccela Pin Name	MAX32690 AF Name[1]	Direction	Description
Chip Select	CS0#	CE0#	HYP_CS0N	Output	Each external device requires a dedicated chip select line. The MAX32690 initiates a bus transaction by transitioning a chip select line from a high state (deselected) to a low state (selected). When the MAX32690 completes the bus transaction, the HPB drives the chip select line to the deselected state (high) indicating to the external device the end of the transaction. Only one chip select can be active at any given time. ***Warning: CS1# requires an external pullup to the supply powering the HyperBus/Xccela Bus if it is used as a chip select line.***
	CS1#	CE1#	HYP_CS1N		
Clock	CK	CK	HYP_CLKP	Output	All information on the bus is transferred with respect to the edges of the clock. CK# is not used for Xccela Bus or 3V HyperBus devices.
	CK#	not used	HYP_CLKN		
Data I/O	DQ0	DQ0	HYP_D0	Input/Output	Data Bus Inputs/Outputs
	DQ1	DQ1	HYP_D1		
	DQ2	DQ2	HYP_D2		
	DQ3	DQ3	HYP_D3		
	DQ4	DQ4	HYP_D4		
	DQ5	DQ5	HYP_D5		
	DQ6	DQ6	HYP_D6		
	DQ7	DQ7	HYP_D7		
R/W Data Strobe	RWDS	DQS	HYP_RWDS	Input/Output	RWDS is controlled by the master or slave device reading data. Data being read is edge aligned with this signal. This signal is held high during any read/write latency periods.

1. Refer to the device data sheet for alternate function pin assignments.

Note: GPIO that support HyperBus as an alternate function are tied to V_{DDIO} and do not support V_{DDIOH} supply selection.

12.2 Peripheral Clock Enable

The HPB is disabled by default on a reset. Use of the HPB peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS1.hpb* to 0 to enable the peripheral clock to the HPB before configuring the HPB for use.

12.3 Related Specifications

Refer to the HyperBus Specification for details of the HyperBus protocol including timing diagrams, AC and DC characteristics, and electrical specifications. Refer to the HyperRAM or HyperFlash data sheet of the specific device being used for timing and electrical specifications. Refer to the Xccela PSRAM data sheet of the device being used for information on the Xccela protocol including timing diagrams, AC and DC characteristics, and electrical specifications.

Most compatible memory devices have on-chip registers for configuring the device. HyperBus refers to these registers as configuration registers, while Xccela devices refer to these as mode registers. This HyperBus/Xccela interface uses the HyperBus terminology for registers and pin names.

12.4 Reading and Writing to a Slave Device from Firmware

Reading and writing with external memory devices is done through two memory-mapped regions of this microcontroller's memory. Configure the memory-mapped addresses using the Memory Base Address registers, where *HPB_MEMBADDR[0]*

configures the memory mapped region for port 0 and *HPB_MEMBADDR[1]* configures the memory mapped region for port 1. The allocated memory region for the HPB on this microcontroller is between 0x6000 0000 and 0x7FFF FFFF. The recommended values are as follows:

```
HBMC_MBR0 = 0x60000000
HBMC_MBR1 = 0x70000000
```

The address on the target memory device is the offset from the above base addresses, so if the software writes a value to RAM location 0x6000 0008, that value is written to address 0x0008 in the target memory device on Port 0. If the software reads memory location 0x7000 2000, then the value stored in address 0x2000 in the target memory device on Port 1 is read back.

These memory-mapped regions are treated as device RAM and executable code space.

Reads and writes to both the external device's memory and configuration registers are all done with these memory-mapped regions. Data transfer operations are as easy as reading and writing to microcontroller RAM. This makes the behavior of the HPB bus interface transparent to firmware. Bit banding is not supported.

Once the memory base addresses are configured, reads and writes to the memory mapped regions are device dependent. Refer to the memory device data sheet to determine how to use the address space.

From firmware, for both HyperBus and Xccela memory devices, all software read and write operations can be 16-bits or 32-bits and aligned on 16-bit or 32-bit boundaries, respectively. Xccela devices also support software single byte reads and writes on byte boundaries.

On the external bus interface, all HyperBus reads and writes are 16-bit bus transactions. All Xccela read bus transactions are 16-bit while all write bus transactions are 32-bit. RWDS is low for the active bytes, while any dummy bytes are masked with RWDS pulled high. This behavior is transparent to firmware.

For example, if the software requests a byte write to an Xccela memory device, the external bus performs a 32-bit write on a 32-bit boundary. RWDS is low for the active byte and masks the three dummy bytes with RWDS high. This behavior is transparent to firmware.

For HyperBus devices, Configuration Register space must be accessed on 16-bit boundaries. Xccela devices can access Mode Registers on byte, 16-bit, or 32-bit boundaries.

12.5 External Memory Data Cache

Between the memory mapped regions and the HyperBus/Xccela external memory is a 16KB data cache. This cache is controlled using the External Memory Cache Controller (EMCC) registers. When activated, the data cache fills its lines by performing INCR8/WRAP8 32-bit read bursts to the HyperBus controller. Configure external memory devices to have a WRAP burst length of 32 bytes.

When accessing a Configuration Register (CR) in an external memory device, the data cache is invalidated and disabled. While HyperFlash memory does not support configuration registers, HyperFlash does support the status register read command. This command requires the software to invalidate the data cache and then disable the data cache before sending the status register read command.

12.5.1 Enabling the EMCC

Perform the following steps to enable the EMCC:

1. Set the *EMCC_CTRL.en* to 0, ensuring the cache is invalidated when enabled.
2. Set *EMCC_CTRL.en* to 1.
3. Read *EMCC_CTRL.rdy* until it returns 1.

12.5.2 Disabling the EMCC

Disable the EMCC by setting *EMCC_CTRL.en* to 0.

12.5.3 Invalidating the EMCC

Invalidate the contents of the EMCC by setting the *EMCC_INVALIDATE* register to 1. Once invalidated, the system flushes the cache. Read the *EMCC_CTRL.rdy* field until it returns 1 to determine when the flush is completed.

12.5.4 EMCC Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 12-2: EMCC Register Summary

Offset	Register	Name
[0x0000]	*EMCC_INFO*	*Cache ID Register*
[0x0004]	*EMCC_SZ*	*Cache Memory Size Register*
[0x0100]	*EMCC_CTRL*	*Instruction Cache Control Register*
[0x0700]	*EMCC_INVALIDATE*	*Instruction Cache Controller Invalidate Register*

12.5.5 EMCC Register Details

Table 12-3: EMCC Information Register

EMCC Information			EMCC_INFO	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:10	id	R	-	**Cache ID** This field returns the ID for this cache instance.
9:6	partnum	R	-	**Cache Part Number** This field returns the part number indicator for this cache instance.
5:0	relnum	R	-	**Cache Release Number** This field returns the release number for this cache instance.

Table 12-4: EMCC Memory Size Register

EMCC Memory Size			EMCC_SZ	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	mem	R	-	**Addressable Memory Size** This field indicates the size of addressable memory by this cache controller instance in 128KB units.
15:0	cch	R	-	**Cache Size** This field returns the size of the cache RAM in 1KB units. 16: 16KB Cache RAM

Table 12-5: EMCC Control Register

EMCC Control			EMCC_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	R/W	-	**Reserved**
16	rdy	R	-	**Ready** This field is cleared by hardware anytime the cache as a whole is invalidated (including a POR). The hardware automatically sets this field to 1 when the invalidate operation is complete, and the cache is ready. 0: Cache invalidation 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.*

EMCC Control			EMCC_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
15:3	-	R/W	-	**Reserved**
2	cwfst_dis	R/W	0	**Critical Word First (CWF) Disable** Setting this field to 1 disables CWF operation. When CWF is disabled, the cache fills the cache line before sending the data read to the CPU. When CWF is enabled, any data fetch that results in a cache miss immediately sends the data read to the CPU before filling the cache line. 0: CWF enabled 1: CWF disabled Note: This field can only be written when the EMCC is disabled (EMCC_CTRL.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 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: Disabled 1: Enabled Note: The EMCC is a write-through cache resulting in any write to the external memory performing an immediate write to the external memory.
0	en	R/W	0	**Cache Enable** Set this field to 1 to enable the cache. Setting this field to 0 invalidates the cache contents, and the line fill buffer handles reads. 0: Disable 1: Enable

Table 12-6: EMCC Invalidate Register

EMCC Invalidate			EMCC_INVALIDATE	[0x0700]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	invalid	W	-	**Invalidate** Writing any value to this register invalidates the cache.

12.6 Memory Transfers

Using either a HyperBus or Xccela Bus memory requires configuration of the HPB to communicate with the external memory device. The HPB supports a maximum of two devices connected through the external I/O pins

This HyperBus/Xccela Interface supports three memory types:

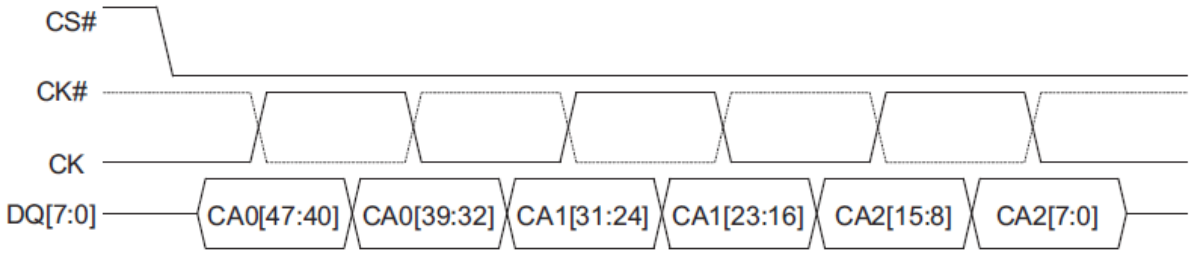
- HyperFlash
- HyperRAM
- Xccela PSRAM

Configure the bus timing for each external memory device. These parameters are in registers *HPB_MEMTIM[0]* and *HPB_MEMTIM[1]*, and should match the same timing parameters specified in the data sheet of the memory device being accessed. By default, Xccela PSRAM is accessed with a variable read latency, but for devices that support it, fixed read latency can be configured.

HyperBus and Xccela bus are similar except that HyperBus uses a differential clock (CK and CK#) while Xccela only uses one clock (CK). While there are differences in signal names, for simplicity only the HyperBus signal names are used for this interface. Both protocols use Double Data Rate (DDR) timing, so data is transferred on the 8-bit data bus on both rising and falling edges of the clock.

All bus transactions are either a read or write to the target memory. All bus transactions first start with one of the active-low chip selects (CS0# or CS1#) going low while CK is low. The first three clock cycles (six clock edges) transfer six bytes of Command/Address information. Because HyperBus is word-oriented, the data is organized as three words designated CA0, CA1, and CA2. Xccela is byte-oriented so the data is organized as one 2-byte instruction INST, and four address bytes A0, A1, A2, and A3.

Figure 12-1: HyperBus Command-Address Sequence



These three words, CA0, CA1, and CA2 define the transaction type and contain the following information:

- Transaction type
  - Read or Write
- Target
  - Slave address space
  - Configuration register space
- Burst type
  - Linear or burst.
- Target row and column as determined by the memory controller
- Global Reset (Xccela only)

Register space is used to access Device Identification (ID) and Configuration Registers (CR). These identify the characteristics of the accessed device and determine the slave-specific behavior of read and write transfers on the HyperBus interface. To write to configuration registers, first set register bit *HPB_MEMCTRL[0].crt* = 1 or *HPB_MEMCTRL[1].crt* = 1. After the configuration registers are written, set these bits back to 0 to access memory.

12.7 External Memory Reset

If the external memory is in a reset state during a read operation, the read-only status bit *HPB_STAT.rdrsterr* is set. If the external memory is in a reset state during a write operation, the read-only status bit *HPB_STAT.wrrsterr* is set. In both cases, an interrupt is generated.

12.8 Interrupts

One interrupt is supported that occurs on any bus error. The interrupt is enabled when *HPB_INTEN.err* = 1. When an AHB bus error occurs, *HPB_INTFL.err* is set to 1.

12.9 Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 12-7: HyperBus Register Names, Offsets, Access and Descriptions

Offset	Register Name	Description
[0x0000]	*HPB_STAT*	HyperBus Status Register
[0x0004]	*HPB_INTEN*	HyperBus Interrupt Enable Control
[0x0008]	*HPB_INTFL*	HyperBus Interrupt Status Flags
[0x0010]	*HPB_MEMBADDR[0]*	HyperBus CS0# Memory Base Address Register
[0x0014]	*HPB_MEMBADDR[1]*	HyperBus CS1# Memory Base Address Register
[0x0020]	*HPB_MEMCTRL[0]*	HyperBus CS0# Memory Configuration Register
[0x0024]	*HPB_MEMCTRL[1]*	HyperBus CS1# Memory Configuration Register
[0x0030]	*HPB_MEMTIM[0]*	HyperBus CS0# Memory Timing Register
[0x0034]	*HPB_MEMTIM[1]*	HyperBus CS1# Memory Timing Register

12.9.1 Register Details

Table 12-8: HPB Status Register

HPB Status Register			HPB_STAT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:27	-	RO	0	**Reserved**
26	wrrsterr	R/W	0	**Reset During Write Error** If this field is set, a reset during write error occurred. When set to 1 by hardware, the *HPB_INTFL.err* interrupt flag is also set to 1. An interrupt is generated if the HPB error interrupt enable is set to 1 (*HPB_INTEN.err* = 1). 0: Normal Operation 1: The memory controller reset during the write.
25	-	RO	0	**Reserved**
24	wraddrerr	R/W	0	**Write Address Error** If this field is set, a write address error occurred. When set to 1 by hardware, the *HPB_INTFL.err* interrupt flag is set to 1. An HPB interrupt is generated if the HPB error interrupt enable is set to 1 (*HPB_INTEN.err* = 1). 0: Normal operation. 1: The write address to the external memory is invalid.
23:17	-	RO	0	**Reserved**
16	wrtxn	R/W	0	**Write Transaction in Progress** This field is set if a write transaction is in progress. This field is cleared by hardware when the transaction completes. 0: No write in progress. 1: Write transaction in progress.
15:12	-	RO	0	**Reserved**
11	rdstall	R/W	0	**Read Data Stall** 0: Normal Operation 1: Read transaction is stalled because RDS is low (stalled).
10	rdrsterr	R/W	0	**Reset During Read Error** If this field is set a reset occurred during a read. When set to 1 by hardware, the *HPB_INTFL.err* is also set to 1. An HPB interrupt is generated if the HPB error interrupt enable flag is set (*HPB_INTEN.err* = 1). 0: Normal Operation 1: Error - Memory controller is under reset during the current read operation

HPB Status Register			HPB_STAT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
9	rdslvst	RO	0	**Reserved**
8	rdaddrerr	R/W	0	**Read Address Error** 0: Normal operation 1: Error - external read address not responding
7:1	-	RO	0	**Reserved**
0	rdtxn	R/W	0	**Read Transaction in Progress** 0: No read in progress. 1: Read transaction in progress.

Table 12-9: HPB Interrupt Enable Control Register

HPB Interrupt Enable Control			HPB_INTEN	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	err	R/W	0	**Error Interrupt Enable** 0: No interrupt 1: Generate an interrupt when an error occurs
0	-	RO	0	**Reserved**

Table 12-10: HPB Interrupt Status Flags Register

HPB Interrupt Status Flags			HPB_INTFL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	err	R/W1C	0	**Error Interrupt Status Flag** Write a 1 to clear this bit.
0	-	RO	0	**Reserved**

Table 12-11: HPB CS0# Memory Base Address Register

HPB CS0# Memory Base Address Register			HPB_MEMBADDR[0]	[0x0010]
HPB CS1# Memory Base Address Register			HPB_MEMBADDR[1]	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	addr	R/W	0	**Memory Base Address** This sets the base address of the addressable memory region where the port is mapped. Each address space is 512MB.
23:0	addr	RO	0	**Address Low** Always reads 0

This is the base address of the addressable memory region in this microcontroller's RAM. Because the addressable memory is mapped in 16MB boundaries, the lower 24 bits are fixed to 0.

The value for HBMC_MBR0 must be less than HBMC_MBR1, or CS1# becomes inactive and the Port 1 memory region is inaccessible.

The allocated memory region for the HPB on this microcontroller is between 0x6000 0000 and 0x7FFF FFFF. The recommended values are as follows:

```
HBMC_MBR0 = 0x6000 0000
HBMC_MBR1 = 0x7000 0000
```

The address on the target memory device is the offset from the base address, so with the recommended values, if 0x12345678 is written to RAM address 0x6000 0008, then 0x12345678 is written to offset address 0x0008 in the target memory device on Port0. The interface signals to Port0, including CS0# and RWDS, are all handled automatically.

Bit banding is not supported in the HyperBus/Xccela addressable memory.

Reading and writing to the memory address space is device dependent. Refer to the slave device data sheet to determine how to use the memory space address map for that device.

CS0# is the Port0 memory region chip select.

CS1# is the Port1 memory region chip select. If CS#1 is used, an external pullup resistor must be connected and tied to the supply used for the HyperBus/Xccela Bus peripheral.

Table 12-12: HPB Memory Configuration 0 Registers

HPB Memory Configuration Register 0			HPB_MEMCTRL[0]	[0x0020]
HPB Memory Configuration Register 1			HPB_MEMCTRL[1]	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31	max_en	R/W	0	**Maximum CS# Length Enable** 0: No configurable CS# low time 1: CS# low time is configured using the field maxlen.
30:27	-	RO	0	**Reserved**
26:18	maxlen	R/W	0	**Maximum Read/Write** Set this field to the CS# low time in terms of clock cycles. The number of clock cycles total: $N_{CLKS} = maxlen + 1$ Number of bytes transferred: $N_T = 2 \times (maxlen + 1)$ *Note: This field is only valid when max_en = 1.*
17:8	-	RO	0	**Reserved**

HPB Memory Configuration Register 0			HPB_MEMCTRL[0]	[0x0020]
HPB Memory Configuration Register 1			HPB_MEMCTRL[1]	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
7	hse	R/W	0	**Xccela Half Sleep Exit** When half sleep exit is enabled, the CS# line is held low for ten clock cycles. This bit is automatically cleared by hardware when a Half Sleep Exit completes. 0: Half Sleep Exit disabled 1: Half sleep exit enabled *Note: This field is only used if the memory device type is either HyperFlash or HyperRAM (devtype = 0 or devtype = 2).*
6	rdlat_en	R/W	0	**Xccela Fixed Read Latency Enable** Set this bit to enable Xccela bus Fixed Read Latency. Set this field to match the Latency Type configuration in the target PSRAM. 0: Variable read latency. 1: Fixed read latency. *Note: This field is only used if the memory device type is Xccela PSRAM (devtype = 1).*
5	crt	R/W	0	**Configuration Register Target Select** For HyperRAM and Xccela Bus devices, this field selects between read/write target being the devices memory map or configuration register space. For HyperFlash set this field to 0. 0: Access Memory space. 1: Access Configuration Register (CR) space. The data cache must be disabled/invalidated in this mode.
4:3	devtype	R/W	0	**Memory Device Type** 0: HyperFlash 1: Xccela PSRAM 2: HyperRAM 3: Reserved
2:0	-	RO	3	**Reserved**

Table 12-13: HPB Memory Timing Register 0

HPB Memory Timing Register 0			HPB_MEMTIM[0]	[0x0030]
HPB Memory Timing Register 1			HPB_MEMTIM[1]	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:28	rdcshi	R/W	0	**Read Chip Select High Between Operations (t_{RDCSHI})** This bit sets the CS# high time, in clock cycles, between read operations as shown in the following equation: $t_{RDCSHI} = t_{HPB\_CLK} \times (rcshi + 1.5)$
27:24	wrcshi	R/W	0	**Write Chip Select High Between Operations (t_{WRCSHI})** This bit sets the CS# high time, in clock cycles, between write operations. $t_{WRCSHI} = t_{HPB\_CLK} \times (wrcshi + 1.5)$ *Note: This field should be set to the same value as the rdcshi field.*
23:20	rdcsst	R/W	0	**Read Chip Select Setup Time to Next CK Rising Edge (t_{RDCSST})** This bit indicates CS# latency, in clock cycles, for read operations. It adds additional clock cycles after CS# goes low. $t_{RDCSST} = t_{HPB\_CLK} \times (rdcsst + 1.0)$
19:16	wrcsst	R/W	0	**Write Chip Select Setup Time to Next CK Rising Edge (t_{WRCSST})** This bit indicates CS# latency, in clock cycles, for write operations. The value set in this field adds additional clock cycles after the CS# line goes low. $t_{WRCSST} = t_{HPB\_CLK} \times (wrcsst + 1.5)$

HPB Memory Timing Register 0				HPB_MEMTIM[0]	[0x0030]
HPB Memory Timing Register 1				HPB_MEMTIM[1]	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**	
15:12	rdcshd	R/W	0	**Read Chip Select Hold After CK Falling Edge (t_{RDCSHD})** This field sets the CS# hold time, in clock cycles, between the completion of a read and when the CS# deassertion. $t_{RDCSHD} = t_{HPB\_CLK} \times (rcsh + 1.0)$	
11:8	wrcshd	R/W	0	**Write Chip Select Hold after CK falling edge (t_{WRCSHD})** This bit indicates the CS# hold time, in clock cycles, for the end of the write to the CS# deassertion. $t_{WRCSHD} = t_{HPB\_CLK} \times (wcsh + 1.0)$	
7:4	-	RO	0	**Reserved**	
3:0	lat	R/W	1	**RAM Latency Clock Cycles** HyperRAM: Set this field to the external HyperRAM Read Latency Configuration Register value. Xccela Bus: For Xccela Bus PSRAM see *Table 12-14*. 0: 5 clock cycles 1: 6 clock cycles 14: 3 clock cycles 15: 4 clock cycles All other values are reserved for future use. *Note: This field is ignored when using HyperFlash devices (HPB_MEMCTRL[0].devtype = 0).*	

Table 12-14: Latency Value Mapped to HyperRAM and Xccela PSRAM Latency Cycles

***HPB_MEMTIM*[0] *HPB_MEMTIM*[1] latency**	**HyperRAM R/W Latency Clock Cycles**	**Xccela PSRAM Write Latency Clock Cycles**	**Xccela PSRAM Read Latency Clock Cycles *HPB_MEMCTRL[0].rdlat_en* = 1**	**Xccela PSRAM Read Latency Clock Cycles *HPB_MEMCTRL[0].rdlat_en* = 1**
0xE	3	3	3	6
0xF	4	4	4	8
0x0	5	5	5	10
0x1	6	6	6	12

13. SPI-XiP External Memory Interface

13.1 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.

13.2 SPIXF

The SPIXF provides the following features:

- Up to 60MHz operation in mode 0 and 3.

Single target select.

- Four-wire mode for single-bit target device communication.

Dual and quad I/O supported.

- Programmable SCK frequency and duty cycle.

SS assertion and deassertion timing with respect to the leading and trailing SCK edge.

- Configurable command, address, dummy, and data fields to support a variety of SPI flash memories.

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

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

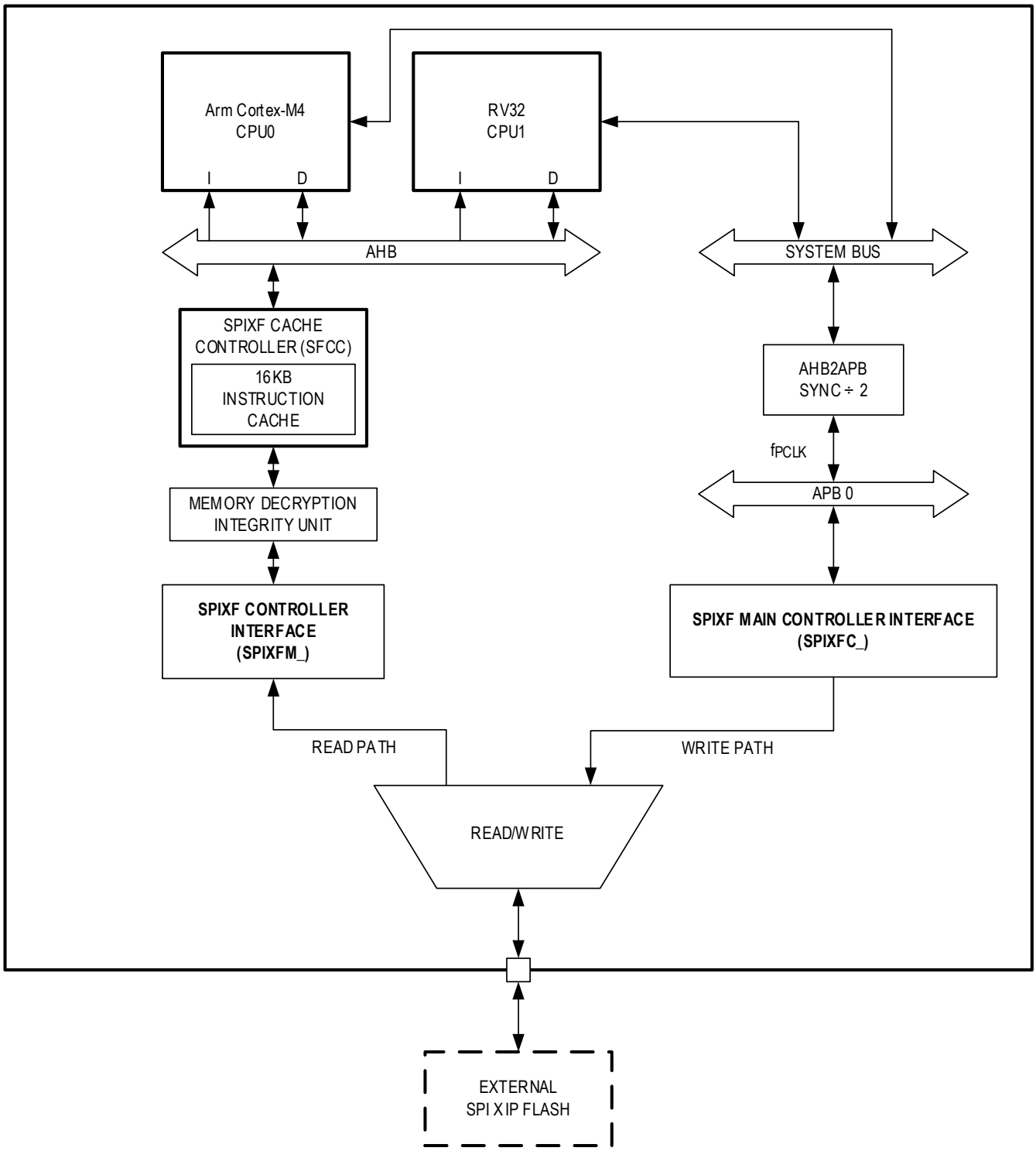
To prevent disclosure of intellectual property, code and data can optionally be stored in external flash in an encrypted form using the SPIXF. Generation of the encrypted data can be done through application software or with the cryptographic accelerator. The SPIXF can transparently decrypt this information in real-time using the memory decryption integrity unit (MDIU) with the AES-128 algorithm in ECB mode.

The SPIXF consists of the SPIXF controller and the SPIXF main controller, as shown in *Figure 13-1*. The SPIXF controller transparently reads the external SPI flash device while the SPIXF main controller is used to write data to the external SPI flash manually and to configure the SPI external flash device registers.

13.2.1.1 Peripheral Clock Enable

The SPIXFC and SPIXFM are disabled by default on a reset. Use of the SPIXFC and SPIXFM peripherals requires enabling the peripheral clocks. Set *GCR_PCLKDIS0.spixipc* and *GCR_PCLKDIS0.spixip* to 0 to enable the peripheral clock to the SPIXFC and SPIXFM before configuring the SPIXFC and SPIXFM for use.

Figure 13-1: Simplified SPIXF Block Diagram



13.2.2 SPIXF Main Controller

The SPIXF main controller block (SPIXFC) 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 the 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 multibyte (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.

13.2.2.1 SPIXFC Configuration

The SPIXF main controller shares pins with the SPIXF controller so that the SPI flash is configured for code execution or data transfer. See the *SPIXF Pin Configuration* section of the SPIXF controller for more information.

13.2.2.1.1 Configuration Modes Overview

Once the main SPIXF main 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_CTRL1* register: SCK feedback mode, enable the transmit and receive FIFOs.
   b. *SPIXFC_SSPOL* register: Target select signal polarity.
   c. *SPIXFC_CTRL0* register: Active target, SPI clock polarity and phase, clock and target select timing.
2. Dynamic configuration: Configuration required to communicate with a specific target device, which may take place while the communication port is enabled, but the target select is deasserted.
   a. *SPIXFC_CTRL0* register: SPI page transfer size if using pages. See header information in *Table 13-1*.
3. Interrupt servicing: Status and control used by an application to efficiently service the SPI data transfer.
   a. *SPIXFC_CTRL2* register: Transmit and receive FIFO monitoring levels.
   b. *SPIXFC_INTFL* register: Interrupt flag bits.
   c. *SPIXFC_INTEN* register: Interrupt enable bits.

13.2.2.1.2 SPI Main Controller Transaction

Once the SPIXFC is configured to communicate to a specific target, SPI transactions are initiated by writing to the SPI transmit FIFO *SPIXFC_FIFO_TX_32*. The FIFO is 16-bits wide and expects a 16-bit header followed by an optional payload padded out to a word boundary.

The format of the header is shown in *Table 13-1*. If the transaction generates receive data, this data is pushed into the SPI. The receive FIFO is *SPIXFC_FIFO_RX_32*.

A complete access sequence to an SPI device is made up of one or more transactions. In some cases, the target select signal remains asserted across several transactions. In other cases, the access sequence defined by the target device might require deassertion of the target 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 controller functionality by software.

Table 13-1: SPI Header Format

Name	Bits	Description	Settings
Header Type	15:14	***Reserved.*** This header field should always be set to 0b00.	0b00
Deassert SS	13	Target Select control.	0: Maintain assertion of target select after the transaction. 1: Deassert target select at the completion of the transaction.

Name	Bits	Description	Settings
RFU	12:11	***Reserved.*** This header field should always be set to 0b00.	0b00
Width	10:9	Number of SDIO pins to use for the transaction.	0b00: Single I/O mode 0b01: Dual I/O mode 0b10: Quad I/O mode 0b11: 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 transmit (i.e., *Direction* = 1 transactions).	0b00: Bits 0b01: Bytes 0b10: Pages (See the *SPIXFC_CTRL0.pgsz* field for page size definition)
Direction	1:0	Defines the direction of information transfer. For headers that do not define a transmission (i.e., direction = none or receive), no payload is required. Conversely, headers that do not define a reception (i.e., direction = none or transmit) result in no data pushing into the receive FIFO.	0b00: None 0b01: Transmit 0b10: Receive 0b11: Both

13.2.2.1.3 SPIXF Main Controller Example

The following steps describe how to set up the main controller:

1. Configure the SPIXF main controller mode, the number of bytes per page (see *SPIXFC_CTRL0.pgsz and note following*), SCK high and low values, and SS active timing and inactive timing.
   a. Example:
      i. *SPIXFC_CTRL0.mode* = 0b00 ***SPI Mode 0***
      ii. *SPIXFC_CTRL0.pgsz* = 0b00 ***Page size = four bytes***
      iii. *SPIXFC_CTRL0.loclk* = 0b01 ***SCK low time = 1 system clock period***
      iv. *SPIXFC_CTRL0.hiclk* = 0b01 ***SCK high time = 1 system clock period***
      v. *SPIXFC_CTRL0.ssact* = 0b10 ***SS Active time = 4 system clock periods***
      vi. *SPIXFC_CTRL0*.ssiact = 0b10 ***SS Inactive stretch time = 8 system clock periods***
2. Configure the almost empty and almost full levels for monitoring the FIFOs.
   a. Example:
      i. *SPIXFC_CTRL2.tx_ae_lvl* = 0x8 ***Almost Empty = 8***
      ii. *SPIXFC_CTRL2.rx_af_lvl* = 0xC ***Almost Full = 12***
3. Enable the transmit and receive FIFOs as well as the feedback clock.
   a. *SPIXFC_CTRL1.rx_fifo_en* = 1 ***Receive FIFO enabled***
   b. *SPIXFC_CTRL1.tx_fifo_en* = 1 ***Transmit FIFO enabled***
4. Write the header, then the payload data to the transmit FIFO (*SPIXFC_FIFO_TX_32*) 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 main controller interrupt flags by clearing the *SPIXFC_INTFL* register.
6. Set the interrupt enables for transmit FIFO stalled and almost empty to make sure that the transmit FIFO does not stall the AHB bus.
7. Write to the SPIXF main controller enable bit to start the transmission (*SPIXFC_CTRL1.en* = 1).
8. Monitor the receive stalled interrupt status flag to know when data is available in the receive FIFO if data is received by this command.
9. Monitor the *SPIXFC_INTFL.tx_rdy* for indication that the command is complete.
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 main controller header to configure the SPIXF main 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 main controller:

1. Configure the SPIXF main controller so it can 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 main controller to send commands to the external SPI flash.

13.2.2.1.4 Clock Phase and Polarity Control

The SPIXF main controller and the SPIXF controller support configuration of SCK phase and polarity:

- Clock polarity selects an active low/high clock and has no effect on the transfer format.

Clock phase selects one of two different transfer formats.

The controller always places data on the MOSI line a half cycle before the SCK edge for the target to latch the data.

Table 13-2 details the SCK phase and polarity combinations supported. See *SPIXFC_CTRL0.mode* field.

Table 13-2: Clock Polarity and Phase Combinations

MODE	Clock Phase	Clock Polarity	SCK Transmit Edge	SCK Receive Edge	SCK Idle State
0	0	0	Falling	Rising	Low
3	1	1	Falling	Rising	High

Note: Do not change the clock phase and polarity control while executing or reading from SPIXF space. This configuration should ideally be done before SPIXF transactions and remains unchanged while reading or executing from SPIXF space. If the clock phase and polarity need to be changed after the SPIXF target select is active, the user must not be executing from SPIXF space, and the SPIXF block should be reset by setting GCR_RST1.spixip = 1.

13.2.2.1.5 Serial Clock Configuration

The output clock speed and pulse width can be controlled with the *SPIXFC_CTRL0.hiclk* and *SPIXFC_CTRL0.loclk* register fields.

$$Target\ SPI\ Clock\ Hi\ Time = t_{PCLK}\ X\ SPIXFC\_CTRL.hiclk$$

$$Target\ SPI\ Clock\ Lo\ Time = t_{PCLK}\ X\ SPIXFC\_CTRL.loclk$$

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

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

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

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

13.2.2.1.6 Target Select Transaction Delay Configuration

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

- *SPIXFC_CTRL0.ssact*

SPIXFC_CTRL0.ssiact

- *SPIXFC_CTRL0.hiclk*

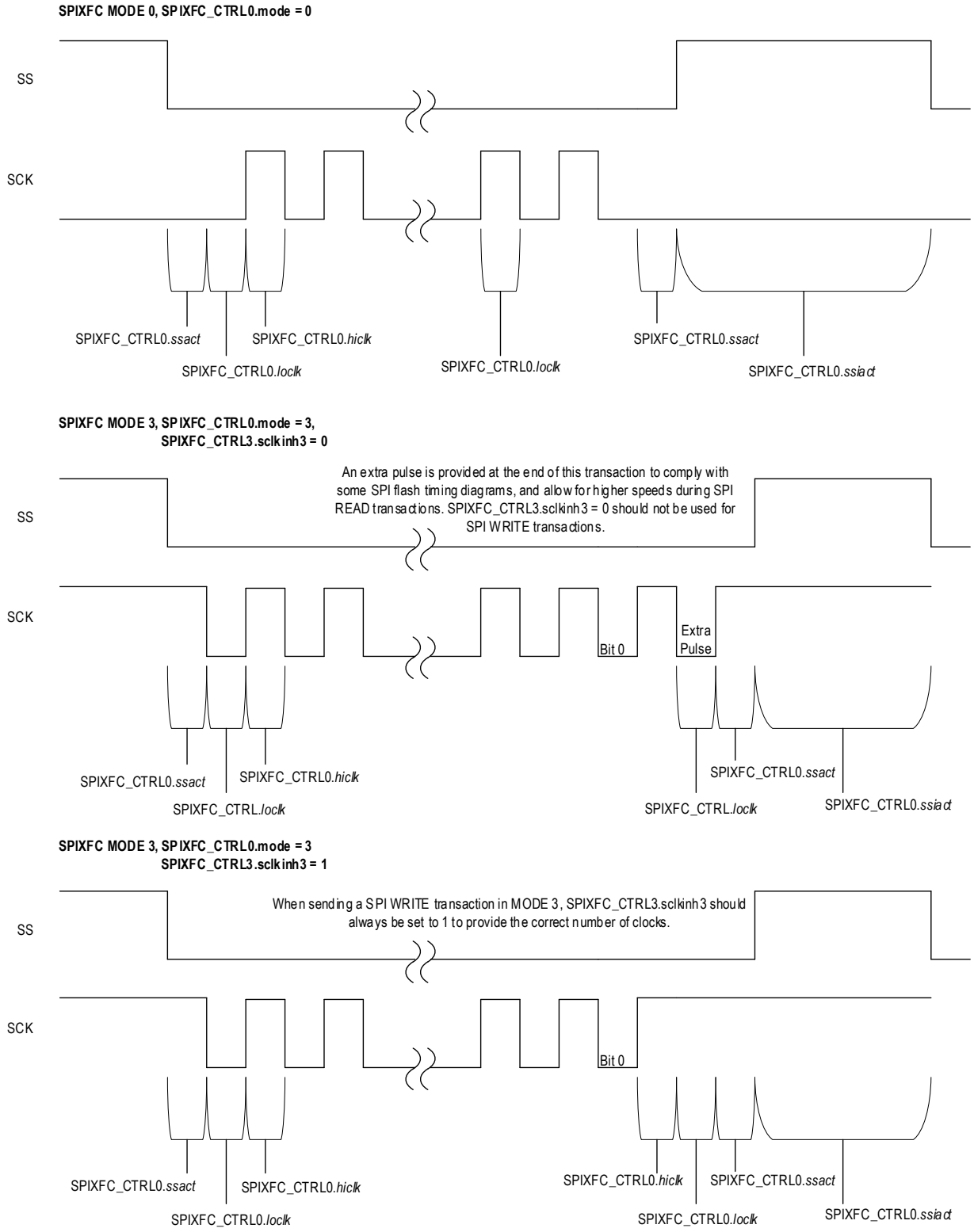
SPIXFC_CTRL0.loclk

- *SPIXFC_CTRL0.mode*

SPIXFC_CTRL3.sclkinh3 (if in mode 3 and for read transactions only)

Automatic target selection deassert for the SPIXF main controller occurs when the transaction header deassert target select field is set. The target select is automatically deasserted if the SPIXF main controller is disabled (*SPIXFC_CTRL1.en* = 0) or *GCR_RST1.spixip* = 1, resetting this peripheral.

Figure 13-2: SPIXFC Transaction Delay



13.2.2.1.7 Target Select

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

13.2.2.1.8 Interrupts

Interrupt logic is provided to allow efficient servicing of the SPIXF main controller by software. The software can group interrupts into the following two groups:

- Keeping the transmit FIFO full.

Keeping the receive FIFO empty.

Programmable levels in the *SPIXR_CTRL2* register allow interrupt 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 *SPIXR_CTRL2* register description for more information.

13.2.2.2 SPIXF Main Controller Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 13-3: SPIXF Main Controller Register Offsets, Names, Access, and Description

Offset	Register	Description
[0x0000]	*SPIXFC_CTRL0*	SPIXF Main Controller Configuration Register
[0x0004]	*SPIXFC_SSPOL*	SPIXF Main Controller Target Select Polarity Register
[0x0008]	*SPIXFC_CTRL1*	SPIXF Main Controller General Controller Register
[0x000C]	*SPIXFC_CTRL2*	SPIXF Main Controller FIFO Control and Status Register
[0x0010]	*SPIXFC_CTRL3*	SPIXF Main Controller Special Control Register
[0x0014]	*SPIXFC_INTFL*	SPIXF Main Controller Interrupt Status Register
[0x0018]	*SPIXFC_INTEN*	SPIXF Main Controller Interrupt Enable Register

13.2.2.2.1 SPIXF Main Controller Register Details

Table 13-4: SPIXF Main Controller Configuration Register

SPIXF Main Controller Configuration Register			SPIXFC_CTRL0	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
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 *hiclk* (for SPI modes 0 and 3). This value applies only in non-clock feedback mode (*SPIXFC_CTRL1*.*sclk_fb* = 0). 0b0000: No delay. 0b0001: 1 SPI clock delay. 0b1111: 15 SPI clock delay.
19:18	ssiact	R/W	0	**Target Select Inactive Stretch** This field controls the number of system clocks the bus is inactive between the end of a transaction (Target Select inactive) and the start of the next transaction (Target Select active). See *Target Select Transaction Delay Configuration* for detailed information. 0b00: 4 system clocks. 0b01: 6 system clocks. 0b10: 8 system clocks. 0b11: 12 system clocks.

SPIXF Main Controller Configuration Register			SPIXFC_CTRL0	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
17:16	ssact	R/W	0	**Target Select Holdoff** Controls the delay from assertion of target select to the start of SCK pulse and the delay from the end of SCK pulses to deassertion of target select. See *Target Select Transaction Delay Configuration* for detailed information. 0b00: 0 system clocks. 0b01: 2 system clocks. 0b10: 4 system clocks. 0b11: 8 system clocks.
15:12	loclk	R/W	0	**SCK Low Clocks** Number of system clocks that SCK is held low when SCK pulses are generated. 0: 16 system clocks 1: 1 system clock 2: 2 system clocks 3: 3 system clocks All other values: This value defines the number of system clock that SCK is held low.
11:8	hiclk	R/W	0	**SCK High Clocks** Number of system clocks that SCK is held high when SCK pulses are generated. 00: 16 system clocks. All other values: This value defines the number of system clock that SCK is held high.
7:6	pgsz	R/W	0	**Page Size** Defines the number of bytes per page for transactions that define transfers in terms of pages. 0b00: 4 bytes. 0b01: 8 bytes. 0b10: 16 bytes. 0b11: 32 bytes.
5:4	mode	R/W	0	**SPI Mode.** Set this field to the required SPI mode. 0b00: SPI Mode 0. Clock polarity = 0, clock phase = 0. 0b01: Invalid. 0b10: Invalid. 0b11: SPI Mode 3. Clock polarity = 1, clock phase = 1.
3	-	RO	0	**Reserved**
2:0	ssel	R/W	0	**Target Select.** Only target 0 is supported 0b000: Target 0 is selected. 0b001-0b111: Invalid.

Table 13-5: SPIXF Main Controller Target Select Polarity Register

SPIXF Main Controller Target Select Polarity			SPIXFC_SSPOL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	ss_pol	R/W	0	**Target Select 0 Polarity** 0: Active low. 1: Active high.

Table 13-6: SPIXF Main Controller General Control Register

SPIXF Main Controller General Control Register			SPIXFC_CTRL1	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**

SPIXF Main Controller General Control Register		SPIXFC_CTRL1		[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
25	sclk_fb_inv	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	-	RO	0	**Reserved**
22	simple	R/W	0	**Simple Mode Target Select** 0: No action. 1: Deassert target select when simple = 1.
21	simple_rx	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: Disabled. 1: Enabled.
19:16	bb_data_out_en	R/W	0	**Bit Bang SDIO Output Enable** Enable output of SDIO0 to SDIO3 in bit-bang mode. bit[3] = SDIO3. bit[2] = SDIO2. bit[1] = SDIO1. bit[0] = SDIO0.
15:12	bb_data_out	R/W	0	**SDIO Drive value in Bit-Bang mode** Defines the output state of the SDIO outputs when in bit-bang mode (*SPIXFC_CTRL1.bbmode* = 1) bit[3]: SDIO3. bit[2]: SDIO2. bit[1]: SDIO1. bit[0]: SDIO0.
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. bit[3]: SDIO3. bit[2]: SDIO2. bit[1]: SDIO1. bit[0]: SDIO0.
7	-	RO	0	**Reserved**
6	sclkdr	R/W	0	**SCK Drive and State** This bit reflects the state of the SCK. When in bit-bang mode (*SPIXFC_CTRL1.bbmode* = 1), this bit is written to control the output state of the SCK. 0: SCK is 0. 1: SCK is 1.
5	-	RO	0	**Reserved**
4	ssdr	R/W	0	**Target Select Drive and State** This bit reflects the state of the target select. This accounts for the polarity as defined in the *SPIXFC_SSPOL* register. When in bit-bang mode, this bit is written to control the output state of the target select. 0: Selected target select output is 0. 1: Selected target select output is 1.
3	bbmode	R/W	0	**Bit-Bang Mode** 0: Disable bit-bang mode. 1: Enable bit-bang mode.

SPIXF Main Controller General Control Register		SPIXFC_CTRL1		[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
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: Disabled. 1: Enabled.
0	enable	R/W	0	**SPI Controller Enable** Setting this bit to 1 enables SPI controller for processing transactions. Clearing this bit disables the SPI controller and puts the block into reset state. 0: Disable SPI controller, putting it into a reset state. 1: Enable SPI controller for processing transactions.

Table 13-7: SPIXF Main Controller FIFO Control and Status Register

SPIXF Main Controller FIFO Control and Status Register		SPIXFC_CTRL2		[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:30	-	RO	00	**Reserved**
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	-	RO	0	**Reserved**
20:16	rx_af_lvl	R/W	0	**Receive FIFO Almost Full Level** The almost full flag is asserted when the number of used FIFO entries (bytes) exceeds this value. FIFO depth is 32 bytes.
15:13	-	RO	0	**Reserved**
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	-	RO	0	**Reserved**
3:0	tx_ae_lvl	R/W	0	**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 13-8: SPIXF Main Controller Special Control Register

SPIXF Main Controller Special Control Register		SPIXFC_CTRL3		[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
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 before deassertion. 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 target select is deasserted. This is to support some SPI flash write timing diagrams. *Note: Always set this field to 0 for SPI write transactions.* 0: Allow trailing SCK low pulse before target select deassertion. 1: Inhibit trailing SCK low pulse before target select deassertion.
15:12	-	RO	0	**Reserved**

SPIXF Main Controller Special Control Register			SPIXFC_CTRL3	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
11:8	sdio_out_en	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	sdio_out	R/W	0	**SDIO Output Value Sample Mode** Defines the values for the SDIO outputs when in sample mode (*SPIXFC_CTRL3.sample* = 1). Bit 7: SDIO[3]. Bit 6: SDIO[2]. Bit 5: SDIO[1]. Bit 4: SDIO[0].
3:1	-	RO	0	**Reserved**
0	sample	R/W	0	**SDIO Sample Mode Enable** Setting this bit to a 1 enables the ability to drive SDIO outputs before the assertion of target 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 target select assertion. 0: Sample Mode disabled. 1: Sample mode enabled.

Table 13-9: SPIXF Main Controller Interrupt Status Register

SPIXF Main Controller Interrupt Status Register			SPIXFC_INTFL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**
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_CTRL2.rx_af_lvl*. 0: Normal operation. 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_CTRL2.tx_ae_lvl*. This does not depend on block enable or the target select value. 0: Normal operation. 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 target select is deasserted. 0: Normal operation. 1: Receive FIFO Not ready.
2	tx_rdy	R/W1C	0	**Transmit Ready Interrupt Status.** This flag is set by hardware when the transmit FIFO is empty, and the target select is deasserted. 0: Normal operation. 1: Transmit ready.

SPIXF Main Controller Interrupt Status Register			SPIXFC_INTFL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
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 target select is asserted. 0: Normal 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 target select is asserted. 0: Normal operation. 1: Stalled FIFO.

Table 13-10: SPIXF Main Controller Interrupt Enable Register

SPIXF Main Controller Interrupt Enable Register			SPIXFC_INTEN	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**
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: Disabled. 1: Enabled.
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: Disabled. 1: Enabled.
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: Disabled. 1: Enabled.
2	tx_rdy	R/W	0	**Transmit Ready Interrupt Enable.** Setting this bit enables interrupt generation when the *SPIXFC_INTFL.tx_rdy* flag is set. Clearing this bit means that no interrupt is generated. 0: Disabled. 1: Enabled.
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: Disabled. 1: Enabled.
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: Disabled. 1: Enabled.

13.2.2.3 SPIXF Main Controller FIFO Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 13-11: SPIXF Main Controller FIFO Register Offsets, Names, Access and Description

Offset	Register	Description
[0x0000]	*SPIXFC_FIFO_TX_32*	32-Bit Transmit FIFO Register
[0x0000]	*SPIXFC_FIFO_TX_16*	16-Bit Transmit FIFO Register
[0x0000]	*SPIXFC_FIFO_TX_8*	8-Bit Transmit FIFO Register
[0x0004]	*SPIXFC_FIFO_RX_32*	32-Bit Receive FIFO Register
[0x0004]	*SPIXFC_FIFO_RX_16*	16-bit Receive FIFO Register
[0x0004]	*SPIXFC_FIFO_RX_8*	8-Bit Receive FIFO Register

13.2.2.3.1 SPIXF Main Controller FIFO Register Details

Table 13-12: SPIXF Main Controller Transmit 32-Bit FIFO Register

SPIXF Main Controller Transmit FIFO			SPIXFC_FIFO_TX_32	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	WO	0	**32-bit Transmit FIFO**

Table 13-13: SPIXF Main Controller 16-Bit Transmit FIFO Register

SPIXF Main Controller 16-bit Transmit FIFO			SPIXFC_FIFO_TX_16	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	-	WO	0	**16-bit Transmit FIFO**

Table 13-14: SPIXF Main Controller 8-Bit Transmit FIFO Register

SPIXF Main Controller 8-bit Transmit FIFO			SPIXFC_FIFO_TX_8	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
7:0	-	WO	0	**8-bit Transmit FIFO**

Table 13-15: SPIXF Main Controller 32-Bit Receive FIFO Register

SPIXF Main Controller 32-bit Receive FIFO			SPIXFC_FIFO_RX_32	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	R	0	**32-bit Receive FIFO**

Table 13-16: SPIXF Main Controller 16-Bit Receive FIFO Register

SPIXF Main Controller 16-bit Receive FIFO			SPIXFC_FIFO_RX_16	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	-	R	0	**16-bit Receive FIFO**

Table 13-17: SPIXF Main Controller 8-Bit Receive FIFO Register

SPIXF Main Controller 8-bit Receive FIFO			SPIXFC_FIFO_RX_8	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
7:0	-	R	0	**8-bit Receive FIFO**

13.2.3 SPIXF Controller (SPIXFM)

The SPIXFM is an advanced high performance bus (AHB) target interface driven by a 16KB unified instruction and data cache. The AHB target supports either instruction execution or fetching of data 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.

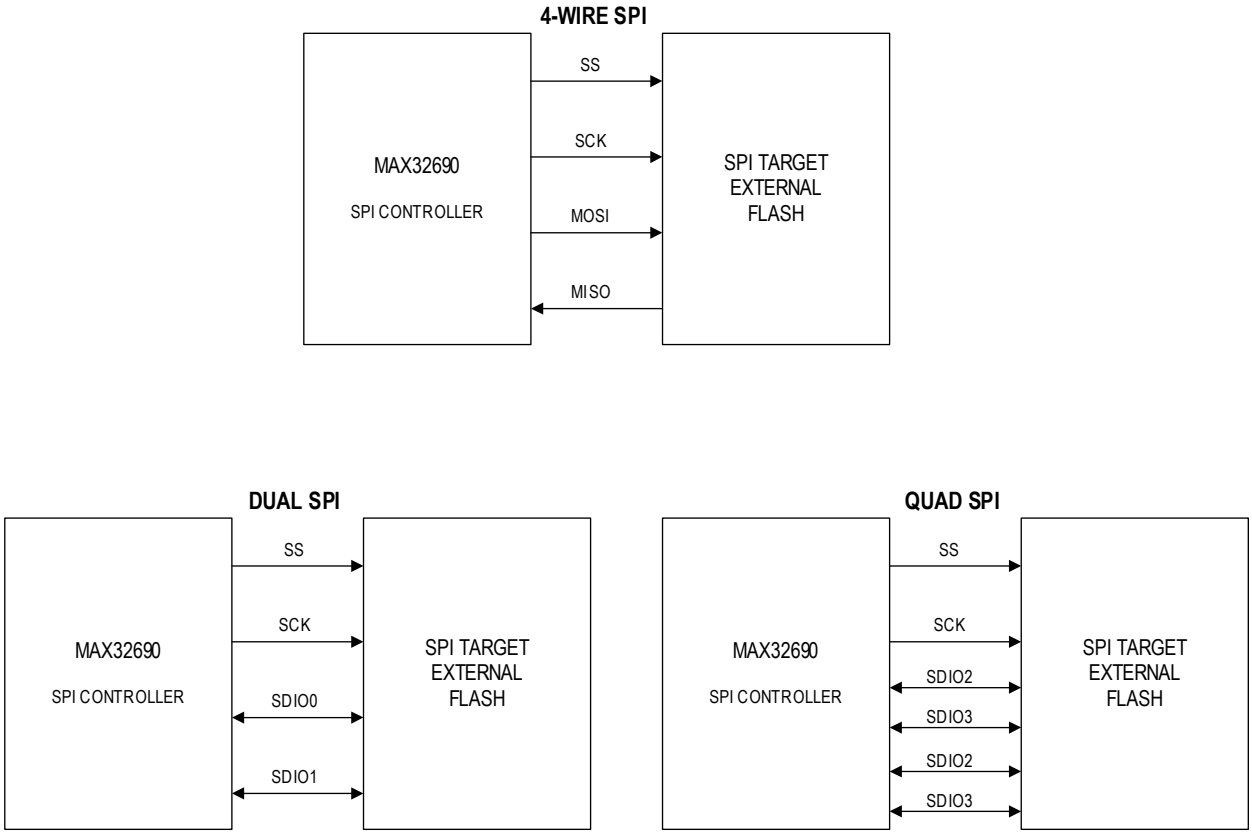
Exercise care in choosing the correct configuration and command to support the speed of data transfer. The SPIXFM provides SCK periods as fast as the AHB clock speed divided by two. The external SPI flash configuration to support data transfer rates must be performed by the SPIXFC.

13.2.3.1 SPIXF Pin Configuration

The SPIXFM and SPIXFC 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 13-3*:

- 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 13-3: Supported SPI configuration



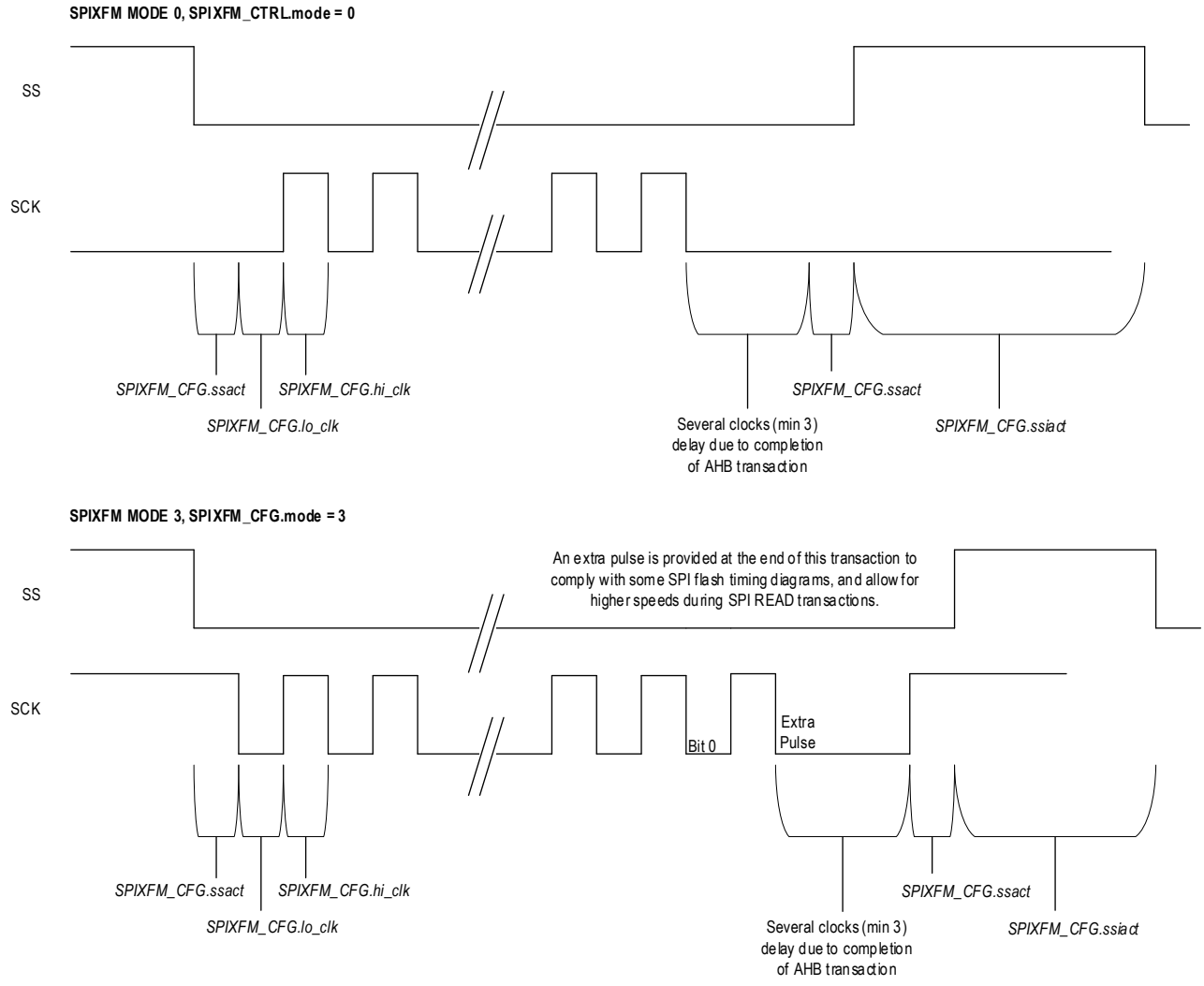
13.2.3.2 Target-Select Transaction Delay Configuration

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

- *SPIXFM_CTRL.ssact*
- *SPIXFM_CTRL.ssiact*
- *SPIXFM_CTRL.hiclk*
- *SPIXFM_CTRL.loclk*
- *SPIXFM_CTRL.mode*

Automatic target-select deassertion only occurs when the next flash address fetched is not contiguous to the current flash address being read or used for execution. The SPIXF does not automatically deassert target selection under any other circumstance including data read or execution of areas outside of the SPIXF space. For these cases, manual control of the target select is provided. Invoke manual control only when running from internal memory. Software can deassert target-select safely by setting *GCR_RST1.spixip*. This resets the SPIXF block (including turning off decryption if previously enabled) and causes the target select to deassert. The SPIXF block requires reconfiguration before subsequent access to external SPI flash space either for execution or data reads.

Figure 13-4: SPIXF Delay Configuration



13.2.3.3 SPIXF Read Sequence Configuration and Control

Assertion of SPIXF target select followed by the read command, then the read address. After the read address is sent, zero 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 deassertion of SPIXF target 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.

13.2.3.4 Sample SPIXF Controller Configuration - Execute Code

Complete the following steps to execute the SPIXF controller configuration sample:

1. Turn on the SPI XIP cache controller (*GCR_PCLKDIS0.spixipc* = 0).
2. Configure the SPIXF controller mode, target select polarity, target number, and target select timing.
    a. Example:
        i. SPI Mode 0
        ii. Target select high
        iii. Target #0
        iv. 1 SPI clock per 2 AHB clocks
        v. *SPIXFM_CTRL* = 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_FETCHCTRL* = 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_MODECTRL* = 0x0
5. Enable the SPIXF feedback clock control.
    a. Example:
        i. SPIXF feedback clock enabled using non-inverted serial feedback clock
        ii. *SPIXFM_FBCTRL* = 0x0001.
6. Jump to the start of application software in SPI flash space.
    a. Example pseudo code:
        i. jump_to_external_flash = (void) (0x0800 0001)
        ii. jump_to_external_flash()

13.2.3.5 Clock Phase and Polarity Control

The SPIXF clock phase and polarity should match the configuration set up by the SPIXF main controller. For more information about clock phase and polarity control, see *Clock Phase and Polarity Control* for the SPIXFC.

13.2.3.6 Serial Clock Feedback Mode

The SPIXF supports high-speed transfer using serial feedback clock mode (*SPIXFM_FBCTRL.fb_en* = 1). The controller output clock is routed back into the digital logic to sample incoming data from the target. This allows for automatic alignment of

the controller clock to the input target data for faster speeds. Serial feedback mode should not be changed while the SPIXF target select is low. Configuration should be done before SPIXF transactions are started and remain unchanged while reading or executing from SPIXF space. If the serial feedback mode configuration needs to change after the SPIXF target select is low, software cannot be executing from the SPIXF space, and the SPIXF block should be reset by setting *GCR_RST1.spixip* to 1.

13.2.3.7 SPIXF Controller Cache Controller

The SPIXF controller uses the same cache controller as the HyperBus interface. See *External Memory Data Cache* for details on enabling the EMCC for use with the SPIXF controller.

13.2.3.8 SPIXF Controller Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 13-18: SPIXF Controller Register Offsets, Names, Access, and Description

Offset	Register	Description
[0x0000]	*SPIXFM_CTRL*	SPIXF Controller Configuration Register
[0x0004]	*SPIXFM_FETCHCTRL*	SPIXF Controller Fetch Control Register
[0x0008]	*SPIXFM_MODECTRL*	SPIXF Controller Mode Control Register
[0x000C]	*SPIXFM_MODEDATA*	SPIXF Controller Mode Data Register
[0x0010]	*SPIXFM_FBCTRL*	SPIXF Controller SCK Feedback Control Register
[0x001C]	*SPIXFM_IOCTRL*	SPIXF Controller I/O Control Register
[0x0020]	*SPIXFM_MEMSECCTRL*	SPIXF Controller Memory Security Register
[0x0024]	*SPIXFM_BUSIDLE*	SPIXF Controller Bus Idle Detection Register
[0x0028]	*SPIXFM_AUTHOFFSET*	SPIXF Controller Authentication Register

13.2.3.9 SPIXF Controller Register Details

Table 13-19: SPIXF Controller Configuration Register

SPIXF Controller Configuration			SPIXFM_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:18	ssiact	R/W	0	**Target Select Inactive Timing** Controls delay from deassertion of target select to reassertion of target select for another SPI transaction. See *Target Select Transaction Delay Configuration* for details. 0b00: 1 system clocks. 0b01: 3 system clocks. 0b10: 5 system clocks. 0b11: 9 system clocks.
17:16	ssact	R/W	0	**Target Select Active Timing** Controls delay from assertion of target select to start of the SCK pulse and delay from the end of SCK pulses to deassertion of target select. See *Target Select Transaction Delay Configuration* for details. 0b00: 0 system clocks. 0b01: 2 system clocks. 0b10: 4 system clocks. 0b11: 8 system clocks.
15:12	hiclk	R/W	0	**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.

SPIXF Controller Configuration			SPIXFM_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
11:8	loclk	R/W	0	**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	-	RO	0	**Reserved**
6:4	ssel	R/W	0	**Target Select** This field must be set to 0.
3	-	RO	0	**Reserved**
2	sspol	R/W	1	**Target Select Polarity** This bit controls the polarity of the target select. 0: Target Select active high. 1: Target 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 13-20: SPIXF Controller Fetch Control Register

SPIXF Controller Fetch Control			SPIXFM_FETCHCTRL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	4baddr	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.
15:14	-	RO	0	**Reserved**
13:12	data_width	R/W	0	**Data Width** Set this field to the number of data I/O used to transfer data. 0b00: Single SDIO. 0b01: Dual SDIO. 0b10: Quad SDIO. 0b11: Reserved.
11:10	addr_width	R/W	0	**Address Width** Set this field to the 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	cmdwth	R/W	0	**Command Width** Set this field to the 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	0	**Command Value** Set this field to the external flash device's command to initiate fetching from the external flash.

Table 13-21: SPIXF Controller Mode Control Register

SPIXF Controller Mode Control			SPIXFM_MODECTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	exit_nocmd	R/W	0	**Mode Send** Setting this field ensures that the next SPI flash transaction sends the mode byte as defined in the *SPIXFM_MODEDATA.mddata* field. When this field is set, the next SPI flash read operation exits continuous mode cleanly. This field and the *SPIXFM_MODECTRL.nocmd* field is automatically cleared by hardware after the next SPI transaction. 0: No action. 1: Send mode byte on next transaction.
8	nocmd	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	-	RO	0	**Reserved**
3:0	mdclk	R/W	0	**Mode Clocks** Number of SPI clocks needed during the mode/dummy phase of fetch.

Table 13-22: SPIXF Controller Mode Data Register

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

Table 13-23: SPIXF Controller SCK Feedback Control Register

SPIXF Controller SCK Feedback Control			SPIXFM_FBCTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	invert	R/W	0	**SCK feedback Clock inversion.** The feedback clock can be phase selected to increase the timing margin for input data from the target 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	0	**SCK Feedback Mode Enable. Enable SCK feedback mode** 0: Disable SCK feedback mode. 1: Enable SCK feedback mode.

Table 13-24: SPIXF Controller I/O Control Register

SPIXF Controller I/O Control			SPIXFM_IOCTRL	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**

SPIXF Controller I/O Control			SPIXFM_IOCTRL	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
4:3	padctrl	R/W	1	**I/O Pullup/Pulldown Control** This field controls the pullups and pulldowns associated with all SPIXF SDIO pins. 0b00: Tristate. 0b01: Pullup. 0b10: Pulldown. 0b11: Pullup.
2	sdio_ds	R/W	1	**SDIO Drive Strength** This bit controls the drive strength of all SDIO pins. 0: Low drive strength. 1: High drive strength.
1	ss_ds	R/W	1	**Target Select Drive Strength** This bit controls the drive strength on the dedicated target select pin. Refer to the device data sheet's electrical characteristics table for details on the drive strength. 0: Low drive strength. 1: High drive strength.
0	sclk_ds	R/W	1	**SCK Drive Strength** This bit controls the drive strength on the SCK pin. Refer to the device data sheet's electrical characteristics table for details on the drive strength. 0: Low drive strength. 1: High drive strength.

Table 13-25: SPIXF Controller Memory Security Control Register

SPIXF Controller Memory Security Control			SPIXFM_MEMSECCTRL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
4	autherr_fl	R/W	0	**Authentication Error Flag** This flag is set to 1 by hardware if an AES authentication error occurs. 0: Normal operation. 1: Authentication error.
3	interl_dis	R/W	0	**Interleaving Disable** 0: Enabled. 1: Disabled.
2	cntopt_en	R/W	0	**Counter Buffer Enable** This field controls the width of the AES-GCM counter buffer used when reading or writing data to the external memory. When the counter buffer is disabled (*cntopt_en* = 0), the counter is read as a single 16 bits from the external memory and the 128-bit counter buffer is not used. Disabling the counter buffer is optimal when the external data is not stored sequentially (i.e., data storage). When the counter buffer is enabled (*cntopt_en* = 1), the AES-GCM counter is read in 128-bit reads. This mode is optimized for sequential storage such as binary software. If a read or write occurs outside the range of the 128-bit counter buffer, a full 128-bit counter read occurs. 0: Disabled. 1: Enabled.

SPIXF Controller Memory Security Control			SPIXFM_MEMSECCTRL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
1	auth_disable	R/W	0	**Authentication Enable** When this field is set to 1, on the fly authentication is performed for the AES GCM encryption and decryption. When authentication is enabled, 16 bits of authentication data is stored in memory for each 128 bits of data. When authentication is disabled, AES ECB mode is used for the external memory encryption/decryption, if enabled (*SPIXFM_MEMSECCTRL.dec_en* = 1). 0: Authentication enabled. 1: Authentication disabled (AES ECB encryption/decryption).
0	dec_en	R/W	0	**Decryption Enable** Set this field to 1 to enable decryption of the external SPIXF automatically during reads. See the *External SPI On-the-Fly Flash Decryption* section for details. 0: Disable decryption of SPIXF data. 1: Enable decryption of SPIXF data.

Table 13-26: SPIXF Controller Bus Idle Detection

SPIXF Controller Bus Idle Detection			SPIXFM_BUSIDLE	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	busidle	R/W	0	**Bus Idle Timer Limit** A 16-bit timer is triggered for each external access. The timer is restarted if another access is performed before the timer expires. When the timer expires, target select is deactivated. This register contains the limit (expiration) value for the timer. A value of 0 disables 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 *LPM*. When this number is too small, the target select pin is deactivated on every access, which may reduce current consumption, but decreases performance.

Table 13-27: SPIXF Controller Authentication Offset

SPIXF Controller Authentication Offset			SPIXFM_AUTHOFFSET	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**Authentication Offset** Set this field to the authentication offset within a page when interleaving is disabled (*SPIXFM_MEMSECCTRL.interl_dis* = 0).

13.3 External SPI Flash Encryption

The user may optionally store encrypted data or code in the external SPI flash. Encryption of the SPI flash data is achieved using the cryptographic accelerator to encrypt the data and the SPIXF main controller to write the data. There are thee modes of encryption supported for the external flash memory. The modes supported include:

- AES-128 ECB.

AES-128 GCM with encryption, authentication, and interleaving.

- AES-128 GCM with encryption, authentication, and without interleaving.

Additionally, the device includes a counter optimization buffer. This buffer allows faster access to sequentially stored external memory, such as code.

13.3.1 AES-128 ECB Mode

Data should be encrypted using AES-128 in ECB mode using keys stored in the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers.

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

- *CTB_CTRL*.*bsi* (Byte swap input).
- *CTB_CTRL*.*bso* (Byte swap output).

Software should load an AES-128 key for encryption/decryption by writing the key to the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers. Setting *CTB_CIPHER_CTRL*.*src* = 3 selects the key stored in the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers for use in cipher operations. If desired, a random key can be generated using the *TRNG Engine*, which can then be loaded into the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers by software.

In AES-128 ECB mode, the data must be preprocessed with an address mask. Each 128 bits of plain text should be XORed with its 128-bit address to avoid patterns in the 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:

```
CTB_DIN[0] = word0
CTB_DIN[1] = word1 ^ (addr_mask+4)
CTB_DIN[2] = word2 ^ (addr_mask+8)
CTB_DIN[3] = word3 ^ (addr_mask+12)
```

Once the encrypted data is available (either using the CTB FIFO or the *CTB_DOUT[3]*:*CTB_DOUT[0]* registers), this data may be written to SPI flash using the SPIXF main controller.

The available output bytes from the cryptographic accelerator should be written to SPIXF flash space, as shown in *Table 13-28*

Table 13-28: Encrypted Data Write Order to SPIXIP Flash Memory

Least Significant Word			Most Significant Word
CTB_DOUT[0]	*CTB_DOUT[1]*	*CTB_DOUT[2]*	*CTB_DOUT[3]*

13.3.2 AES-128 GCM with Interleaving

In this mode, AES-128 GCM with encryption and authentication is used for the external memory. The authentication and optional counter data is interleaved with the encrypted data in the external memory. AES-128 GCM does not require the preprocessing of the data with the address to avoid patterns in the external memory because AES-GCM provides masking of any data patterns using the initialization vector (IV). There are 16 bits of authentication data stored for each 128 bits of data.

Data should be encrypted using AES-128 in GCM mode (*CTB_CIPHER_CTRL*.*mode* = 6) using keys stored in the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers.

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

- *CTB_CTRL*.*bsi* (Byte swap input).
- *CTB_CTRL*.*bso* (Byte swap output).

Software should load an AES-128 key for encryption/decryption by writing the key to the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers. Setting *CTB_CIPHER_CTRL*.*src* = 3 selects the key stored in the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers for use in cipher operations.

The IV is 96-bits and consists of the following data concatenated:

- Bits 15:0 are the counter.
- Bits 47:16 are the current line address offset.
- Bits 95:48 are fixed as 0.

When constructing the IV, the counter value used must also be stored interleaved in the external memory, as shown in *Figure 13-5*.

The initialization vector should be loaded into the *CTB_CIPHER_INIT[2]*:*CTB_CIPHER_INIT[0]* registers. The *CTB_CIPHER_INIT[3]* register should be set to 0.

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

```
CTB_DIN[0] = word0
CTB_DIN[1] = word1
CTB_DIN[2] = word2
CTB_DIN[3] = word3
```

Once the encrypted data is available (either using the CTB FIFO or the *CTB_DOUT[3]*:*CTB_DOUT[0]* registers), this data may be written to SPI flash using the SPIXF main controller. The available encrypted output bytes from the cryptographic accelerator should be written to SPIXF flash space, as shown in *Table 13-29* and *Figure 13-5*.

Additionally, the authentication data is 16-bits long and covers 128-bits of data. When using the cryptographic accelerator, the authentication data is available in the 16 LSBs of the *CTB_TAGMIC[0]* register after an AES-128 GCM operation is performed. Each 16-bits of authentication data is concatenated in 128-bit lines and each 16-bits of authentication data covers 128-bits of data lines and should be written to external memory, as shown in *Figure 13-5*. Additionally, if counters are used in the construction of the IV, the counter values should also be written to external memory, as shown in *Figure 13-5*. In the external memory, there is physically one authentication line and one counter line for each eight lines of data. As a result, the useful external memory space is reduced to 7/9th of the maximum memory space.

Figure 13-5: External Memory Storage with AES-GCM Enabled and Interleaving Enabled

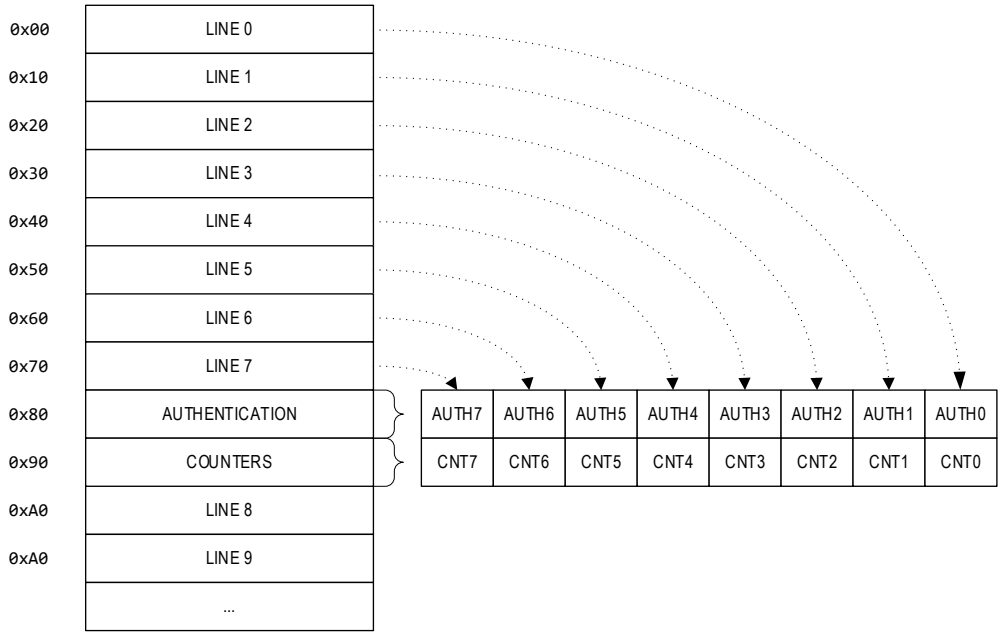


Table 13-29: AES-GCM 128-Bit Encrypted Data Write Order to SPIXIP Flash Memory

Least Significant Word			Most Significant Word
CTB_DOUT[0]	*CTB_DOUT[1]*	*CTB_DOUT[2]*	*CTB_DOUT[3]*

13.3.3 AES-128 GCM without Interleaving

In this mode, AES-128 GCM with encryption and authentication is used for the external memory. The authentication and optional counter data is not interleaved with the encrypted data in the external memory, but stored at the end of the encrypted data in the external memory at a fixed offset. AES-128 GCM does not require the preprocessing of the data with the address to avoid patterns in the external memory because AES-GCM provides masking of any data patterns using the IV. There are 16 bits of authentication data stored for each 128 bits of data.

Data should be encrypted using AES-128 in GCM mode (*CTB_CIPHER_CTRL*.*mode* = 6) using keys stored in the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers.

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

- *CTB_CTRL*.*bsi* (Byte swap input)
- *CTB_CTRL*.*bso* (Byte swap output)

Software should load an AES-128 key for encryption/decryption by writing the key to the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers. Setting *CTB_CIPHER_CTRL*.*src* = 3 selects the key stored in the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers for use in cipher operations.

The initialization vector is 96-bits and consists of the following data concatenated:

- Bits 15:0 are the counter.
- Bits 47:16 are the current line address offset.
- Bits 95:48 are fixed as 0.

When constructing the IV, the counter value used must also be stored interleaved in the external memory, as shown in *Figure 13-6*.

The initialization vector should be loaded into the *CTB_CIPHER_INIT[2]*:*CTB_CIPHER_INIT[0]* registers. The *CTB_CIPHER_INIT[3]* register should be set to 0.

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

```
CTB_DIN[0] = word0
CTB_DIN[1] = word1
CTB_DIN[2] = word2
CTB_DIN[3] = word3
```

Once the encrypted data is available (either using the CTB FIFO or the *CTB_DOUT[3]*:*CTB_DOUT[0]* registers), this data may be written to SPI flash using the SPIXF main controller. The available encrypted output bytes from the cryptographic accelerator should be written to SPIXF flash space, as shown in *Table 13-30* and *Figure 13-6*.

Additionally, the authentication data is 16-bits long and covers 128-bits of data. When using the cryptographic accelerator, the authentication data is available in the 16 LSBs of the *CTB_TAGMIC[0]* register after an AES-128 GCM operation is performed. Each 16-bits of authentication data is concatenated in 128-bit lines and each 16-bits of authentication data covers 128-bits of data lines and should be written to external memory, as shown in *Figure 13-6*. Additionally, if counters are used in the construction of the IV, the counter values should also be written to external memory, as shown in *Figure*

13-6. In the external memory, there is physically one authentication line and one counter line for each eight lines of data. As a result, the useful external memory space is reduced to 7/9th of the maximum memory space.

Figure 13-6: External Memory Storage with AES-GCM Enabled and Interleaving Disabled

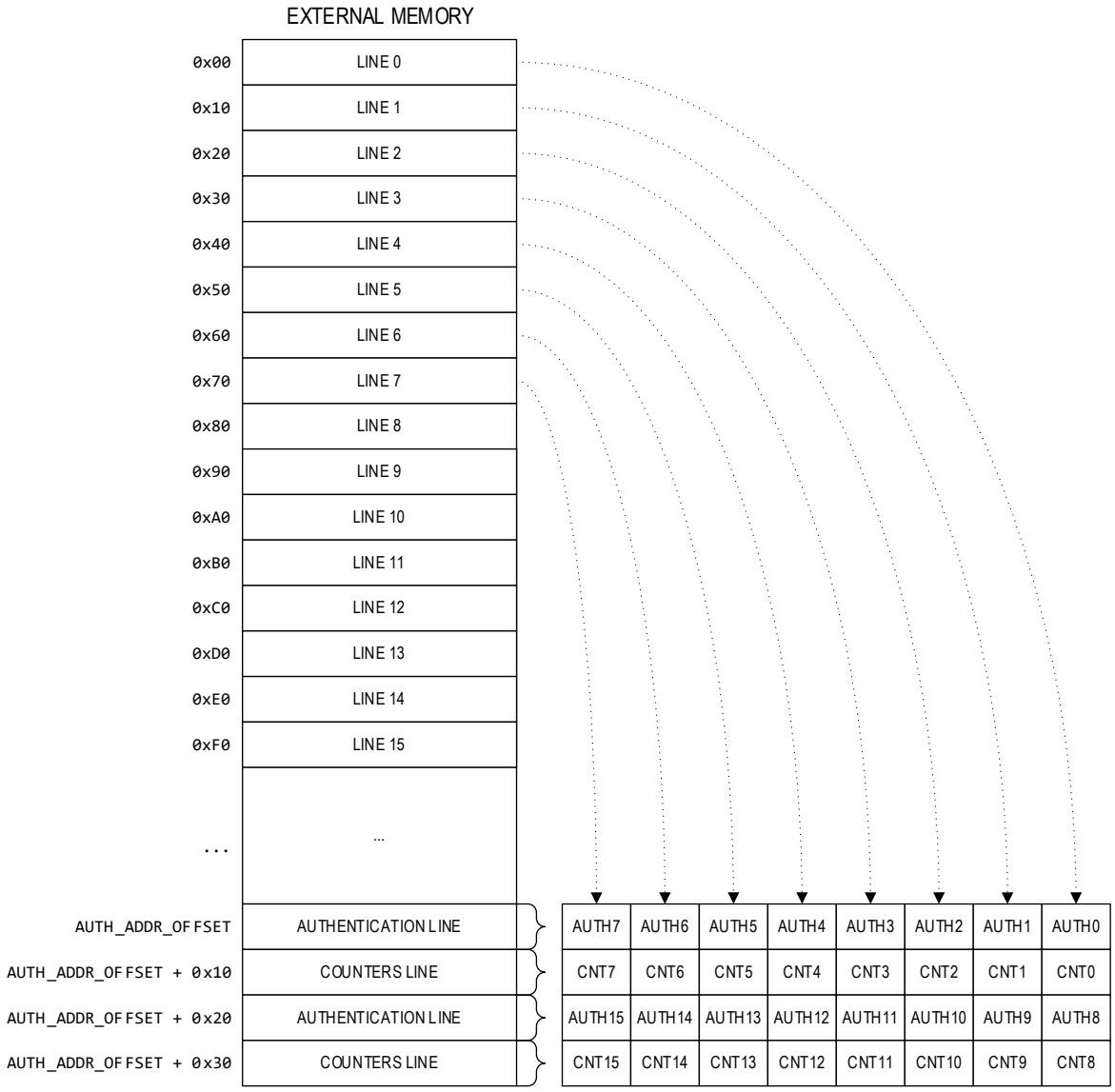


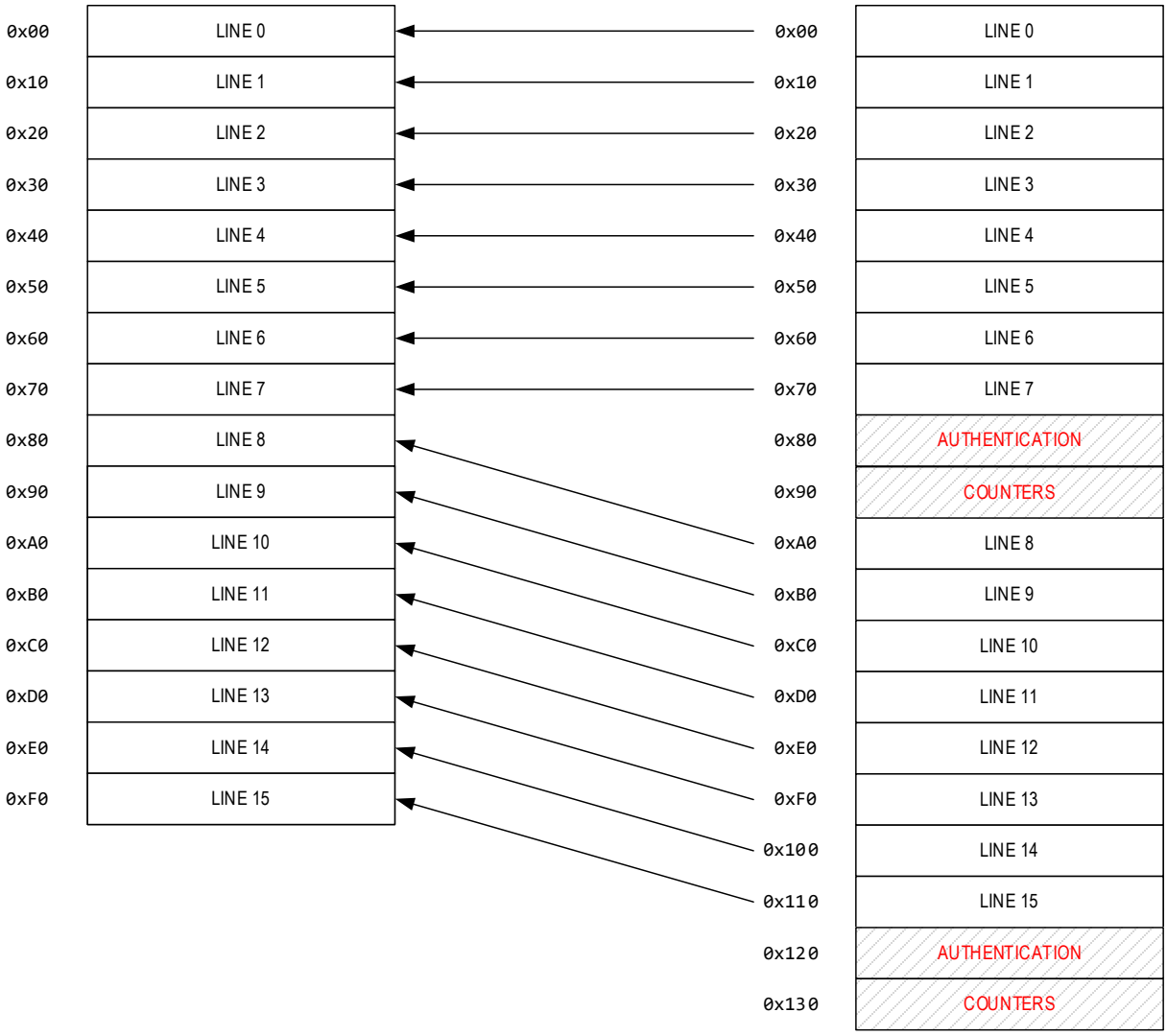
Table 13-30: AES-GCM 128-Bit Encrypted Data Write Order to SPIXIP Flash Memory

Least Significant Word			Most Significant Word
CTB_DOUT[0]	CTB_DOUT[1]	CTB_DOUT[2]	CTB_DOUT[3]

13.3.4 Address Translation

The authentication process is totally transparent for the system. The address translation, when reading external memory, is handled transparently by the SPIXFM for AES-GCM with and without counter optimization. See Figure 13-7 for an example of how the address translation is handled by the SPIXFM controller with interleaving enabled.

Figure 13-7: Address Translation with AES-GCM and Interleaving Enabled



13.4 External SPI On-the-Fly Flash Decryption

If data in the SPI flash is encrypted, as described in the *External SPI Flash Encryption* section, it can be transparently decrypted on read using either code execution or data reads. Decryption is not enabled by default.

13.4.1 AES-128 ECB Decryption Configuration

Decryption of external memory using AES-128 ECB is performed using the following configuration:

1. Load the AES-128 key into the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers.
2. Turn off authentication (*SPIXFM_MEMSECCTRL*.*auth_disable* = 1).
3. Turn off interleaving (*SPIXFM_MEMSECCTRL*.*interl_dis* = 1).
4. Turn off counter optimization (*SPIXFM_MEMSECCTRL*.*cntopt_en* = 0).
5. Enable the MDIU for decryption on the fly (*SPIXFM_MEMSECCTRL*.*dec_en* = 1).

13.4.2 AES-128 GCM Decryption With Interleaving

Decryption of external memory using AES-128 GCM with interleaving is performed using the following configuration:

1. Load the AES-128 key into the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers.
2. Turn on authentication (*SPIXFM_MEMSECCTRL*.*auth_disable* = 0).
3. Turn on interleaving (*SPIXFM_MEMSECCTRL*.*interl_dis* = 0).
4. Turn on counter optimization if accessing sequentially stored information from the external memory (*SPIXFM_MEMSECCTRL*.*cntopt_en* = 0).
    a. See the *Counters Optimization Feature* section for more information.
5. Enable the MDIU for decryption on the fly (*SPIXFM_MEMSECCTRL*.*dec_en* = 1).

13.4.3 AES-128 GCM Decryption Without Interleaving

Decryption of external memory using AES-128 GCM without interleaving is performed using the following configuration:

1. Load the AES-128 key into the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers.
2. Turn on authentication (*SPIXFM_MEMSECCTRL*.*auth_disable* = 0).
3. Turn off interleaving (*SPIXFM_MEMSECCTRL*.*interl_dis* = 1).
4. Set the *SPIXFM_AUTHOFFSET* register to the address offset within the external memory page where the authentication data is stored. See *Figure 13-6* for more information.
5. Turn on counter optimization if accessing sequentially stored information from the external memory (*SPIXFM_MEMSECCTRL*.*cntopt_en* = 1).
    a. See the *Counters Optimization Feature* section for more information.
6. Enable the MDIU for decryption on the fly (*SPIXFM_MEMSECCTRL*.*dec_en* = 1).

See *SPIXF Main Controller* for information about encrypting data for storage in the external SPI flash.

13.4.4 Counters Optimization Feature

The MAX32690 includes a counters optimization feature. This feature, when enabled, allows optimized access to sequentially stored data in external memory, such as for code access. When enabled, the counter optimization buffer is used and 128-bits of counters data is buffered.

If the external memory access is more random, as might be the case for data access, counters optimization should be disabled.

13.5 SPIXR

The SPIXR main controller is an instantiation of the SPI with the following features:

- Four SPI modes (mode 0, 1, 2, and 3)

Controller 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 SCK frequency and duty cycle

32-byte transmit FIFO and 32-byte receive FIFO

  - DMA support

  16KB data cache

The SPIXR main controller allows the CPU to transparently execute instructions stored in an external SPI SRAM device. Instructions fetched using the SPIXR main controller are cached just like instructions fetched from internal program memory. The SPIXR main controller can also be used to access large amounts of external data.

Before accessing an SPI SRAM device, the software must configure the SPIXR interface.

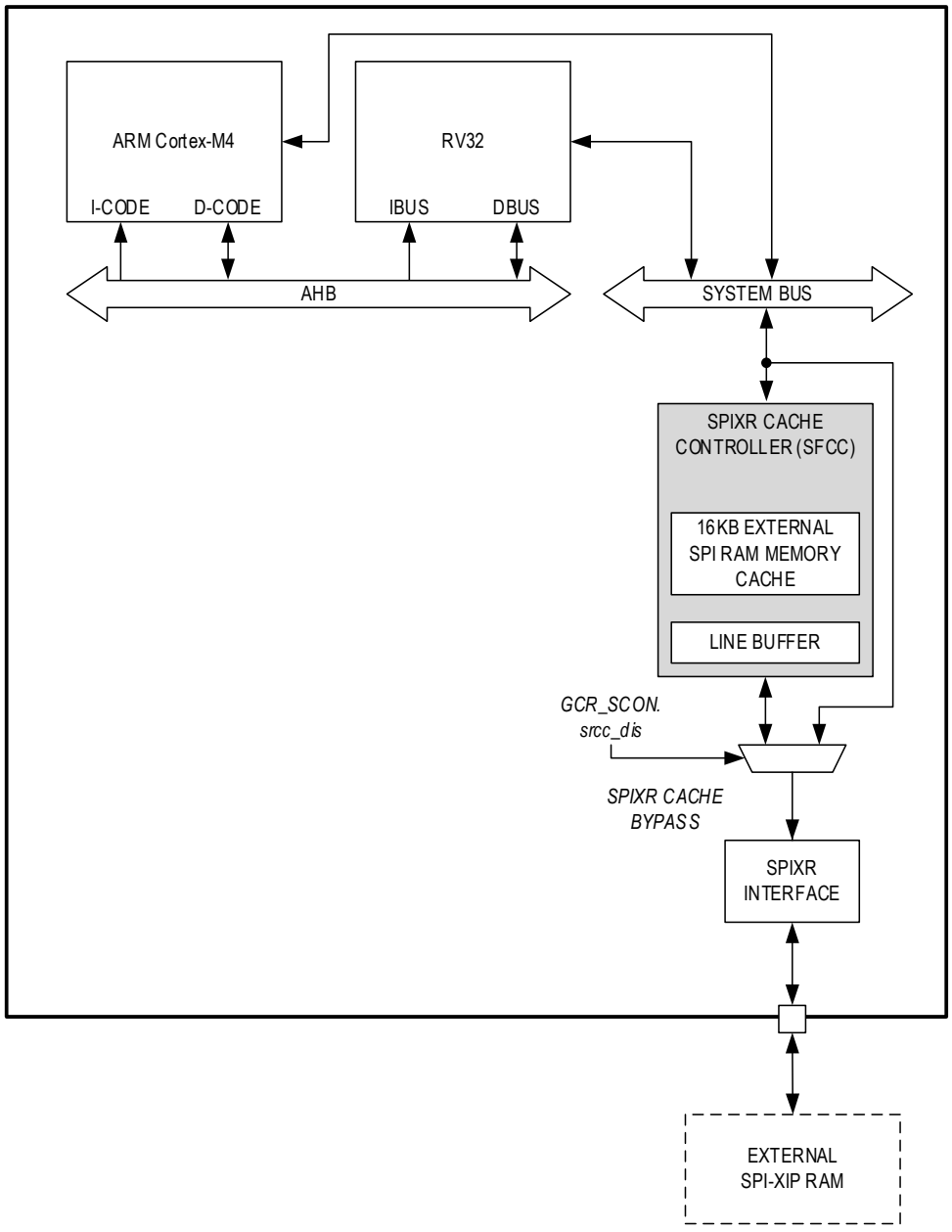
The command used to transfer SPI SRAM data is configured using software. Then, the access to SPI SRAM space (either code execution or data) may be performed by software. The AHB transaction initiated by the software provides address and other transaction critical parameters to control the data transfer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

Exercise care when choosing the correct configuration and command to support the speed of data transfer. The SPIXR main controller provides SCK periods as fast as the APB clock. The external SPI SRAM configuration to support data transfer rates must be performed by the SPIXR main controller.

13.5.1 Peripheral Clock Enable

The SPIXR is disabled by default on a reset. Use of the SPIXR peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS1*.*spixr* to 0 to enable the peripheral clock to the SPIXR before configuring the SPIXR for use.

Figure 13-8: Simplified SPIXR Block Diagram



13.5.2 SPIXR Main Controller Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 13-31: SPIXR Main Controller Register Offsets, Names, Access, and Descriptions

Offset	Register	Description
[0x0000]	*SPIXR_DATA32*	SPIXR 32-bit FIFO Data Register
[0x0000]	*SPIXR_DATA16*	SPIXR 16-bit FIFO Data Register
[0x0000]	*SPIXR_DATA8*	SPIXR 8-bit FIFO Data Register
[0x0004]	*SPIXR_CTRL0*	SPIXR Controller Signals Control Register
[0x0008]	*SPIXR_CTRL1*	SPIXR Transmit Packet Size Register

Offset	Register	Description
[0x000C]	*SPIXR_CTRL2*	SPIXR Static Configuration Register
[0x0010]	*SPIXR_CTRL3*	SPIXR Target Select Timing Register
[0x0014]	*SPIXR_BRGCTRL*	SPIXR Controller Baud Rate Register
[0x001C]	*SPIXR_DMA*	SPIXR DMA Control Register
[0x0020]	*SPIXR_INTFL*	SPIXR Interrupt Status Flags Register
[0x0024]	*SPIXR_INTEN*	SPIXR Interrupt Enable Register
[0x0028]	*SPIXR_WKFL*	SPIXR Wake-up Status Flags Register
[0x002C]	*SPIXR_WKEN*	SPIXR Wake-up Enable Register
[0x0030]	*SPIXR_STAT*	SPIXR Active Status Register
[0x0034]	*SPIXR_XMEMCTRL*	SPIXR XMEM Control Register

13.5.3 SPIXR Register Details

Table 13-32: SPIXR 32-bit FIFO Data Register

SPIXR 32-bit FIFO Data				SPIXR_DATA32	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:0	-	R/W	0	**SPIXR FIFO Data**	

Table 13-33: SPIXR 16-bit FIFO Data Register

SPIXR 16-bit FIFO Data Register				SPIXR_DATA16	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**	
15:0	-	R/W	0	**SPIXR 16-bit FIFO Data**	

Table 13-34: SPIXR 8-bit FIFO Data Register

SPIXR 8-bit FIFO Data Register				SPIXR_DATA8	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**	
7:0	-	R/W	0	**SPIXR 8-bit FIFO Data**	

Table 13-35: SPIXR Controller Signals Control Register

SPIXR Controller Signals Control Register				SPIXR_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:17	-	RO	0	**Reserved**	
16	ss	R/W	0	**Controller Target Select** This field selects the target select pin for the SPIXR interface. 0: The target select pin is not selected for the SPIXR. 1: The SPIXR target select pin is used for the SPIXR.	
15:9	-	RO	0	**Reserved**	
8	ss_ctrl	R/W	0	**Controller Target Select Control** Setting this field to 1 leaves the SS signal asserted at the end of the transmission. This enables multiple transmissions to occur without the SS signal being deasserted. At the completion of all transmissions with the SPIXR device, this field must be set to 0 to deassert the target select line. 0: SS is deasserted at the end of a transmission. 1: SS stays asserted at the end of a transmission.	
7:6	-	RO	0	**Reserved**	
5	tx_start	R/W1O	0	**Controller Start Data Transmission** Set this field to 1 to start the transaction with the target device. The hardware automatically clears this field after the transaction is started. 0: No SPIXR data transmission is in process. 1: Controller 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 transmit FIFO before setting this bit.*	

SPIXR Controller Signals Control Register			SPIXR_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
4	ssio	R/W	0	**Controller Target Select Signal Direction** This field must be set to 0 for SPIXR operation. 0: Target Select is an output. 1: Reserved. *Note: The SPIXR only operates as a SPI controller in single controller mode. Writing 1 to this field is invalid.*
3:2	-	RO	0	**Reserved**
1	mstr_en	R/W	1	**SPIXR Controller Mode Enable** This field must be set to 1 to use the SPIXR peripheral. 0: Reserved. 1: Enabled. *Note: The SPIXR peripheral only operates in controller mode. Writing 0 to this field is invalid.*
0	en	R/W	0	**SPIXR Enable/Disable** Set this field to 1 to enable the SPIXR peripheral. 0: Disabled. 1: Enabled. *Note: Setting this field to 0 disables the SPIXR but maintains all registers and the FIFO data.*

Table 13-36: SPIXR Transmit Packet Size Register

SPIXR Transmit Packet Size Register			SPIXR_CTRL1	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	rx_num_char	R/W	0	**Number of Characters in the Receive FIFO** This field returns the number of characters in the receive FIFO. *Note: This field is only used if the SPIXR is configured for three-wire SPI operation, SPIXR_CTRL2.three_wire = 1.*
15:0	tx_num_char	R/W	0	**Number of Characters in the Transmit FIFO** This field returns the number of characters in the transmit FIFO. *Note: If the SPIXR is set to four-wire mode, SPIXR_CTRL2.three_wire = 0, this field represents both the receive and transmit FIFO character count.*

Table 13-37: SPIXR Static Configuration Register

SPIXR Static Configuration Register			SPIXR_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	sspol	R/W	0	**Target 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	-	RO	0	**Reserved**
13:12	data_width	R/W	0	**SPIXR Data Width** This field 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.

SPIXR Static Configuration Register			SPIXR_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
11:8	numbits	R/W	0	**Number of Bits per Character** This field sets the number of bits per character for an SPIXR transaction.
7:5	-	RO	0	**Reserved**
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_CTRL2.cpol* field to set the polarity of the clock for a given mode. 0: Normal SCK output. 1: Reserved.
3:2	-	RO	0	**Reserved**
1	cpol	R/W	0	**Clock Polarity** Sets the SCK clock polarity for the supported modes. 0: Normal clock (SPI Mode 0 and Mode 1). 1: Inverted clock (SPI Mode 2 and Mode 3). *Note: This field must be set depending on the SPI Mode configuration.*
0	cpha	R/W	0	**Clock Phase** Sets the SPIXR SCK clock phase. 0: Data sampled on the clock's rising edge (SPI Mode 0 and Mode 2). 1: Data sampled on the clock's falling edge (SPI Mode 1 and Mode 3). *Note: This field must be set based on the SPI Mode configuration.*

Table 13-38: SPIXR Target Select Timing Register

SPIXR Target Select Timing Register			SPIXR_CTRL3	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23:16	ssiact	R/W	0	**SS Inactive Clock Delay** This is the time SS is inactive between character transmission. This field 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	**SS 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	**SS Active Before SCK** Number of system clock cycles between the time SS is asserted until the first SCK edge.

Table 13-39: SPIXR Controller Baud Rate Generator

SPIXR Controller Baud Rate Generator Register			SPIXR_BRGCTRL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	RO		**Reserved**
19:16	scale	R/W	0	**SPIXR Clock Scale Factor** Scales the system clock by 2^{scale} to generate the internal SPIXR peripheral clock. $f_{SPIXR\_CLK} = \frac{f_{SYS\_CLK}}{2^{scale}}$

SPIXR Controller Baud Rate Generator Register			SPIXR_BRGCTRL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
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_BRGCTRL.scale = 0, SPIXR_BRGCTRL.high = 0, and SPIXR_BRGCTRL.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 low cycle time to: $SCK\_LOW = lo \times SPIXR\_CLK$ *Note: If SPIXR_BRGCTRL.scale = 0, SPIXR_BRGCTRL.high = 0, and SPIXR_BRGCTRL.low = 0, character sizes of 2 and 10 bits are not supported.*

Table 13-40: SPIXR DMA Control Register

SPIXR DMA Control Register			SPIXR_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31	dma_rx_en	R/W	0	**Receive DMA Enable** Enable or disable receive DMA. 0: Disabled. Any pending DMA requests are cleared. 1: Enabled.
30	-	RO	0	**Reserved**
29:24	rx_fifo_cnt	R	0	**Number of Bytes in the Receive FIFO** Reading this field returns the number of bytes currently in the receive FIFO.
23	rx_fifo_clr	R/W1O	0	**Receive FIFO Clear** Set this field to 1 to clear the receive FIFO and all related receive FIFO flags in the SPIXR_INTFL register. 1: Clear the receive FIFO and any pending receive FIFO flags in SPIXR_INTFL. This should be done when the receive FIFO is inactive. *Note: Writing 0 has no effect.*
22	rx_fifo_en	R/W	0	**Receive FIFO Enabled** 0: Disabled. 1: Enabled.
21	-	RO	0	**Reserved**
20:16	rx_fifo_lvl	R/W	0	**Receive FIFO Threshold Level** When the receive FIFO has more entries than this field, a DMA request is triggered, and the SPIXR_INTFL.*rx_thresh* interrupt flag is set. Valid values are 0x00 to 0x1E. 0x1F: Reserved.
15	dma_tx_en	R/W	0	**Transmit DMA Enable** 0: Disabled. Any pending DMA requests are cleared. 1: Enabled.
14	-	RO	0	**Reserved**
13:8	tx_fifo_cnt	R	0	**Number of Bytes in the Transmit FIFO** Read returns the number of bytes currently in the transmit FIFO.

SPIXR DMA Control Register			SPIXR_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
7	tx_fifo_clear	WO	0	**Clear the Transmit FIFO** Set this field to 1 to clear the transmit FIFO and all transmit FIFO related flags in the *SPIXR_INTFL* register. When the transmit FIFO is cleared, the *SPIXR_INTFL*.*tx_empty* flag is set to 1 by hardware. 1: Clear the transmit FIFO and any pending transmit FIFO flags in *SPIXR_INTFL*. *Note: Only set this field to 1 when the transmit FIFO is inactive. Writing 0 has no affect.*
6	tx_fifo_en	R/W	0	**Transmit FIFO Enabled** Set this field to 1 to enable the transmit FIFO. 0: Transmit FIFO disabled. 1: Transmit FIFO enabled.
5	-	RO	0	**Reserved**
4:0	tx_fifo_lvl	R/W	0x10	**Transmit FIFO Threshold Level** When the transmit FIFO has fewer entries than this field, a DMA request is triggered and the *SPIXR_INTFL*.*tx_thresh* interrupt flag is set.

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

Table 13-41: SPIXR Interrupt Status Flag Register

SPIXR Interrupt Status Flag Register			SPIXR_INTFL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	rx_und	R/W1C	0	**Receive FIFO Underrun Flag** This field is set when a read is attempted from an empty receive FIFO.
14	rx_ovr	R/W1C	0	**Receive FIFO Overrun Flag** This field is set if the SPI is in target mode and a write to a full receive FIFO is attempted. If the SPI is in controller 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** This field is set if the SPI is in target mode and a read from an empty transmit FIFO is attempted. If the SPI is in controller 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 a full transmit FIFO.
11	m_done	R/W1C	0	**Controller Data Transmission Done Flag** This field is set if the SPI is in controller mode and all transactions have completed.
10	-	RO	0	**Reserved**
9	abort	R/W1C	0	**Target Mode Transaction Abort Detected Flag** This field is set if the SPI is in target mode and the SS pin is deasserted before a complete character is received.
8	fault	R/W1C	0	**Multi-Controller Fault Flag** This field is set if the SPI is in controller mode, multicontroller mode is enabled, and a target select input is asserted. A collision also sets this flag.
7:6	-	RO	0	**Reserved**
5	ssd	R/W1C	0	**Target Select Deasserted Flag**
4	ssa	R/W1C	0	**Target Select Asserted Flag**
3	rx_full	R/W1C	0	**Receive FIFO Full Flag** This field is set when the receive FIFO is full.

SPIXR Interrupt Status Flag Register			SPIXR_INTFL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
2	rx_thresh	R/W1C	0	**Receive FIFO Threshold Level Crossed Flag** This field is set when the receive FIFO exceeds the value in *SPIXR_DMA.rx_fifo_lvl*.
1	tx_empty	R/W1C	1	**Transmit FIFO Empty Flag** This field is set when the transmit FIFO is empty.
0	tx_thresh	R/W1C	0	**Transmit FIFO Threshold Level Crossed Flag** This field is set when the transmit FIFO is less than the value in *SPIXR_DMA.tx_fifo_lvl*.

Table 13-42: SPIXR Interrupt Enable Register

SPIXR Interrupt Enable Register			SPIXR_INTEN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	rx_und	R/W	0	**Receive FIFO Underrun Interrupt Enable** 0: Disabled. 1: Enabled.
14	rx_ovr	R/W	0	**Receive FIFO Overrun Interrupt Enable** 0: Disabled. 1: Enabled.
13	tx_und	R/W	0	**Transmit FIFO Underrun Interrupt Enable** 0: Disabled. 1: Enabled.
12	tx_ovr	R/W	0	**Transmit FIFO Overrun Interrupt Enable** 1: Interrupt enabled. 0: Interrupt disabled.
11	m_done	R/W	0	**Controller Data Transmission Done Interrupt Enable** 0: Disabled. 1: Enabled.
10	-	RO	0	**Reserved**
9	abort	R/W	0	**Target Mode Transaction Abort Detected Interrupt Enable** 0: Disabled. 1: Enabled.
8	fault	R/W	0	**Multi-Controller Fault Interrupt Enable** 0: Disabled. 1: Enabled.
7:6	-	RO	0	**Reserved**
5	ssd	R/W	0	**Target Select Deasserted Interrupt Enable** 0: Disabled. 1: Enabled.
4	ssa	R/W	0	**Target Select Asserted Interrupt Enable** 0: Disabled. 1: Enabled.
3	rx_full	R/W	0	**Receive FIFO Full Interrupt Enable** 0: Disabled. 1: Enabled.
2	rx_thresh	R/W	0	**Receive FIFO Threshold Level Crossed Interrupt Enable** 0: Disabled. 1: Enabled.
1	tx_empty	R/W	1	**Transmit FIFO Empty Interrupt Enable** 0: Disabled. 1: Enabled.
0	tx_thresh	R/W	0	**Transmit FIFO Threshold Level Crossed Interrupt Enable** 0: Disabled. 1: Enabled.

Table 13-43: SPIXR Wake-up Flag Register

SPIXR Wake-up Flag Register			SPIXR_WKFL	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	rx_full	R/W1C	0	**Wake on Receive FIFO Full Flag** If this field is set, the receive FIFO full condition caused the wake event.
2	rx_thresh	R/W1C	0	**Wake on Receive FIFO Threshold Level Crossed Flag** If this field is set, the receive FIFO threshold level crossed condition caused the wake event.
1	tx_empty	R/W1C	0	**Wake on Transmit FIFO Empty Flag** If this field is set, the transmit FIFO empty condition caused the wake event.
0	tx_thresh	R/W1C	0	**Wake on Transmit FIFO Threshold Level Crossed Flag** If this field is set, the transmit FIFO threshold level crossed caused the wake event.

Table 13-44: SPIXR Wake-up Enable Register

SPIXR Wake-up Enable Register			SPIXR_WKEN	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	rx_full	R/W	0	**Wake on Receive FIFO Full Enable** Set to 1 to wake up the device when the receive FIFO is full. 0: Wake-up disabled for this condition. 1: Wake-up enabled for this condition.
2	rx_thresh	R/W	0	**Wake on Receive FIFO Threshold Level Crossed Enable** Set to 1 to wake up the device when the receive FIFO threshold is crossed. 0: Wake-up disabled for this condition. 1: Wake-up enabled for this condition.
1	tx_empty	R/W	0	**Wake on Transmit FIFO Empty Enable** Set to 1 to wake up the device when the transmit FIFO is full. 0: Wake-up disabled for this condition. 1: Wake-up enabled for this condition.
0	tx_thresh	R/W	0	**Wake on Transmit FIFO Threshold Level Crossed Enable** Set to 1 to wake up the device when the transmit FIFO threshold is crossed. 0: Wake-up disabled for this condition. 1: Wake-up enabled for this condition.

Table 13-45: SPIXR Active Status Register

SPIXR Active Status Register			SPIXR_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
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. 1: SPI is active.

Table 13-46: SPIXR External Memory Control Register

SPIXR External Memory Control Register			SPIXR_XMEMCTRL	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31	xmem_en	R/W	0	**Enable External Memory** 0: Disabled 1: Enabled
30:24	-	RO	0	**Reserved**

SPIXR External Memory Control Register			SPIXR_XMEMCTRL	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
23:16	dummy_clk	R/W	0	**External Memory Dummy Characters** Set this field to the number of dummy characters between the address phase and the read data phase from the external memory. 0: No delay between address and read data. 1 - 255: The number of delay characters.
15:8	wr_cmd	R/W	0	**Write Command** Set this field to the SPIXR memory's write command. This is a vendor-specific value.
7:0	rd_cmd	R/W	0	**Read Command** Set this field to the SPIXR memory's read command. This is a vendor-specific value.

13.5.4 SPIXF Cache Controller (SFCC)

The SFCC is an AHB block with multiple interfaces. The address and data interfaces are connected to the AHB, and the SFCC register interface is connected through the advanced peripheral bus (APB).

The SFCC is a 16KB 2-way set-associative cache. It operates with the least recently used (LRU) replacement policy and has a write-through implementation used for caching instructions and data from an external SPI-XiP RAM device. The SFCC includes tag RAM, cache RAM, and a line fill buffer. Write allocate and critical word first are options controlled by the software. Each cache line is 256-bits wide, with the lower 5-bits of the address used as the cache line index. The SFCC 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 SFCC to access up to 512MB of external memory. The SFCC is mapped to the address range of 0x8000 0000 to 0x9FFF FFFF for a maximum of 512MB external.

13.5.4.1 Features

- 2-way set associative, LRU replacement policy.

Write-no-allocate with the option to write-allocate.

- Write-through.

Read critical word first and streaming.

- 512MB addressable range.

16KB size.

13.5.4.2 Enabling the SFCC

Enable the SFCC using the following steps:

1. Set the *GCR_SYSCTRL.sfcc_dis* field to 0.
2. Write any value to the *SFCC_INVALIDATE* register to force a cache flush.
2. Set the *SFCC_CTRL.en* field to 1.

Once enabled and flushed, 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 POR event, the cache tag RAM is cleared by hardware, ensuring that no corrupted data is accessed from the initial cache read.

13.5.4.3 Disabling the SFCC

Disabling the SFCC invalidates the cache contents. When the SFCC is disabled, all accesses to the external memory are performed using the line buffer. Disable the SFCC by setting *SFCC_CTRL.en* to 0.

The SFCC's cache memory and line buffer can both be bypassed by setting *GCR_SYSCTRL.sfcc_dis* to 1. Bypassing the SFCC enables direct access to the external memory from software.

13.5.4.4 SFCC Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* for a list of all access types for each bit and field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific reset.

Table 13-47: External Memory Cache Controller Register Addresses and Descriptions

Offset	Register Name	Description
[0x0000]	*SFCC_INFO*	Cache ID Register
[0x0004]	*SFCC_SZ*	Cache Memory Size Register
[0x0100]	*SFCC_CTRL*	Cache Control Register
[0x0700]	*SFCC_INVALIDATE*	Invalidate Register

13.5.4.5 SFCC Register Details

Table 13-48: SFCC Cache ID Register

SFCC Cache ID Register			SFCC_INFO	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	-	**Reserved**
15:10	id	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 13-49: SFCC Memory Size Register

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

Table 13-50: SFCC Cache Control Register

SFCC Cache Control Register			SFCC_CTRL	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
31:17	-	RO	-	**Reserved**
16	rdy	R	-	**Ready** This field is cleared by hardware anytime the cache as a whole is invalidated (including a POR 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	-	RO	-	**Reserved**

SFCC Cache Control Register			SFCC_CTRL	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
2	cwfst_dis	R/W	0	**Critical Word First (CWF) Disable** Set this field to 1 to disable CWF operation. When CWF is disabled, the cache fills the cache line before sending the data to the CPU core. When CWF is enabled, any data fetch that results in a cache miss immediately sends the data read to the CPU core before filling the cache line. 0: Enabled. 1: Disabled. *Note: This field is only writable when the SFCC is disabled (SFCC_CTRL.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. 1: Write allocate enabled. *Note: The SFCC is a write-through cache resulting in any write to the external memory performing an immediate write to the external device.*
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: Disabled. 1: Enabled.

Table 13-51: SFCC Invalidate Register

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

14. 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, as well as for communication between various smart home and Internet of Things (IoT) devices.

Devices operate in the unlicensed 2.4GHz ISM (Industrial, Scientific, Medical) band. A frequency-hopping transceiver is used to combat interference and fading. The system operates in the 2.4GHz ISM band at 2400MHz to 2483.5MHz. It uses 40 RF channels. These RF channels have center frequencies of 2402 + k × 2MHz, where k = 0, …, 39.

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
- Hardware on-the-fly encryption and decryption for lower power consumption
- Supports mesh networking
- Supports high-quality audio streaming (isochronous)
- Low-power proprietary mode that supports 20kbps, 40kbps, 500kbps-MSK/GFSK, 1Mpbs-GFSK

14.1 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

14.2 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 overall power consumption of the Bluetooth system.

14.3 Packetcraft Software Stack

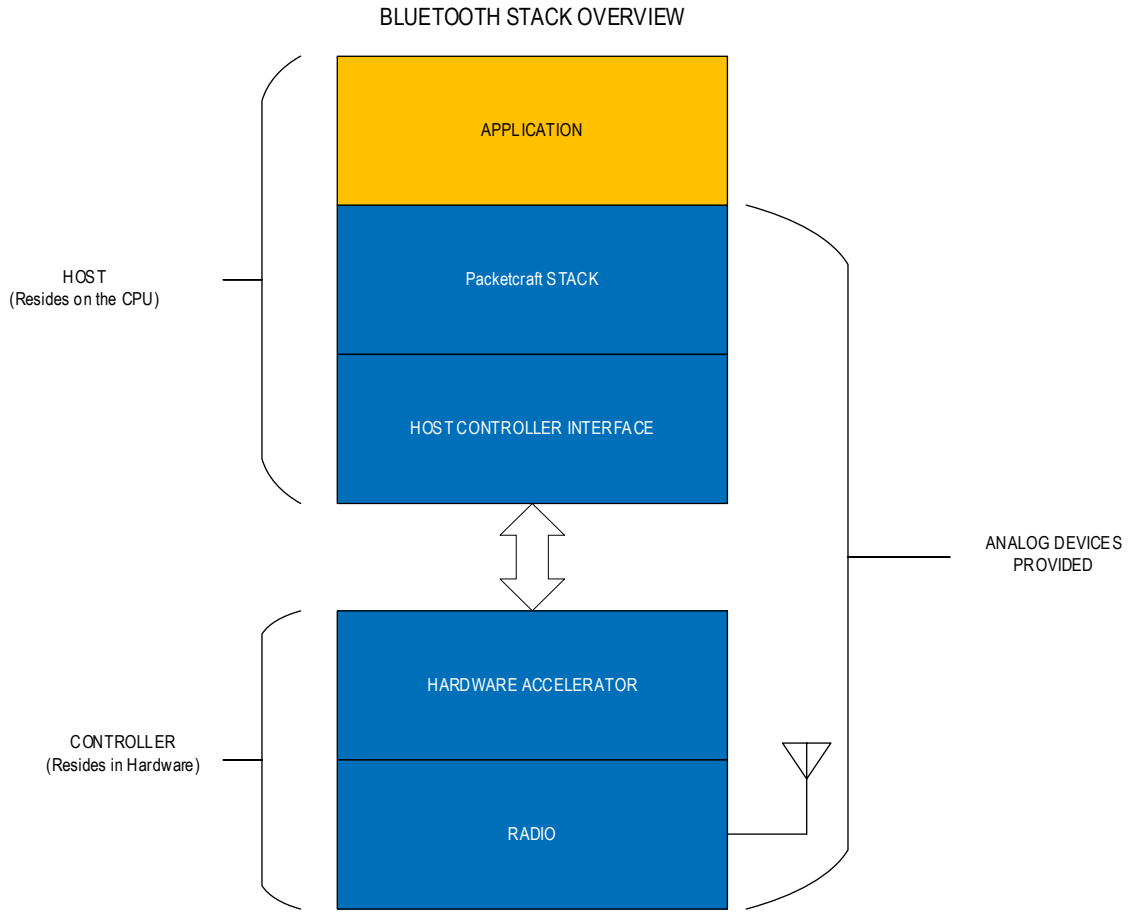
Analog Devices provides the Packetcraft Host and Controller in library form. This provides application developers access to Bluetooth without validation and development of a software stack.

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 Packetcraft software framework.

Figure 14-1: MAX32690 Bluetooth Stack Overview



14.4 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.

14.5 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_BTLELDOCTRL.ldotxen* and set to *GCR_BTLELDOCTRL.ldorxen* 1 to enable the internal Bluetooth LDO.
2. Set *GCR_CLKCTRL.erfo_en* to 1.
3. Wait for *GCR_CLKCTRL.erfo_rdy* to be set to 1. Do not access any Bluetooth registers beforehand.

14.5.1 Kick Starting the ERFO

14.6 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

15. Controller Area Network (CAN)

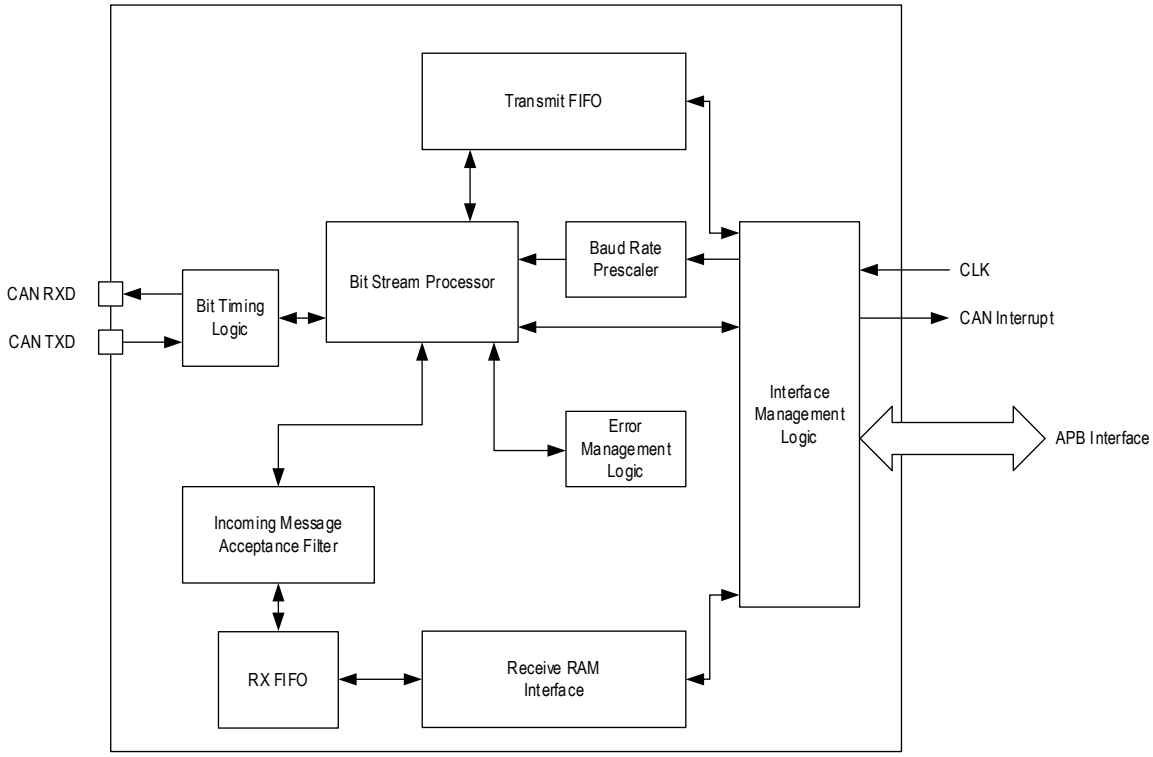
Key features:

- Supports CAN 2.0b frames.
- Selectable ID type:
  - 11-bit standard ID.
  - 11-bit Standard ID + 18-bit extended ID.
- Selectable frame type:
  - Data frame (remote transmission request (RTR) = 0).
  - Remote frame (RTR = 1) for CAN frames.
- Flexible data transfer rate:
  - Up to 1Mbps.
- Hardware message filtering (dual/single filters).
- DMA support for transmit and receive.
- 128-byte transmit FIFO and 256-byte receive FIFO.
- Overload frame is generated on a FIFO overflow.
- Protocol Exception Event detection.
- Normal and Listen Only modes supported.
- Transmitter Delay Compensation up to three data bits long.
- Single Shot transmission.
- Readable error counters.
- Last error code.
- Sleep mode and wake-up unit.

Figure 15-1 shows the block diagram of the peripheral. See *Table 15-1* for the peripheral-specific bus assignment and bit rate generator clock source.

Note: The CAN peripheral supports CAN FD. However, the device does not support CAN FD frequencies. This peripheral should only be used in CAN 2.0b networks.

Figure 15-1: CAN Block Diagram



15.1 Instances

Table 15-1: MAX32690 CAN Instances

Instance	CAN Clock Options
CAN0	PCLK
CAN1	PCLK

15.2 Bit Timing

The bit rate (f_{BIT}) of a system characterizes the number of data bits transmitted per time unit. *Equation 15-1* shows the bit rate of a CAN system. *Figure 15-2* shows the nominal bit time (t_{BIT}). The nominal bit time consists of three non-overlapping segments SYNCSEG, TSEG1, and TSEG2, with the corresponding time durations $t_{SYNCSEG}$, t_{TSEG1}, and t_{SEG2}. The time duration of the nominal bit time is the sum of the segment durations, as shown in *Equation 15-2*. Each of these segments is specified as an integer number of the basic unit of time in a bit period, referred to as the time quantum (t_q). The time quantum is one period of the CAN system clock, t_{SCL}, which is derived from selected peripheral clock, t_{CLK}.

The CAN system clock in standard CAN mode can be adjusted by the application software through a programable prescaler selected with the field *CANn_BUSTIM0.br_clkdiv*, as shown in *Equation 15-3*.

The TSEG1 and TSEG2 segments are controlled using the *CANn_BUSTIM1.tseg1* and *CANn_BUSTIM1.tseg2* fields. TSEG1 is adjustable from a minimum of 1 t_q to 16 t_q. TSEG2 is adjustable from 1 t_q to 8 t_q.

Another time segment of importance to the CAN bit timing calculation is the synchronization jump width (SJW). The SJW with time duration t_{SJW} is not a segment of the bit period but defines the maximum number of time quanta by which a bit period can be lengthened or shortened in the event of resynchronization. The SJW is controlled using the *CANn_BUSTIM0.sjw* field and can be set from 1 t_q to 4 t_q.

Equation 15-1: Bit Rate

$$f_{BIT} = \frac{1}{t_{BIT}}$$

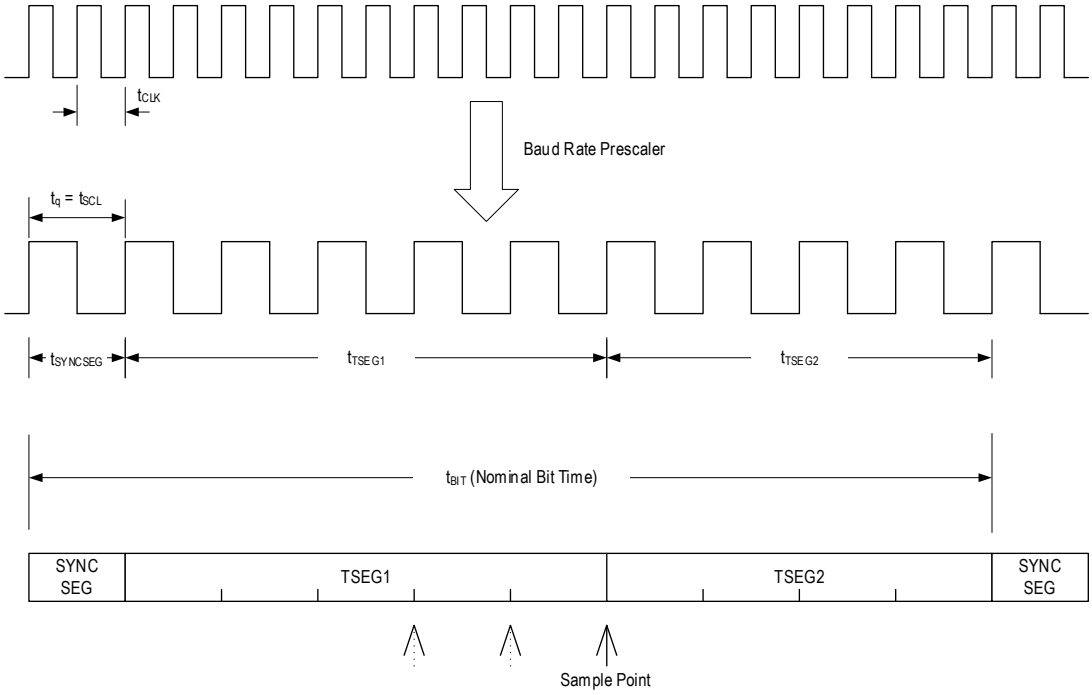
Equation 15-2: Nominal Bit Time Duration

$$t_{BIT} = t_{SYNCSEG} + t_{TSEG1} + t_{TSEG2}$$

Equation 15-3: CAN System Clock Selection for Standard CAN

$$t_{SCL} = 2 \times t_{CLK} \times CANn\_BUSTIM0.br\_clkdiv$$

Figure 15-2: CAN Clock Configuration and Nominal Bit Time



15.3 Operating Modes

The CAN controller supports multiple operating modes and test modes. After a POR, system reset, or peripheral reset, the CAN controller is in the reset state. The reset state is used for the configuration of the CAN controller by software.

15.3.1 Peripheral Clock Enable

Each CAN port is disabled by default on a reset. The use of a CAN port requires enabling the peripheral clock for the port. Enable the peripheral clock for CAN0 by setting *GCR_PCLKDIS1.can0* to 0. Enable the peripheral clock for CAN1 by setting *GCR_PCLKDIS1.can1* to 0.

15.3.2 Reset Mode

When the CAN controller is in reset mode, frame reception and transmission are disabled. Configuration of the CAN controller is performed while in reset mode. After configuration, the CAN controller can be placed into normal, sleep, or one of the supported test modes. The CAN controller is in reset mode when the *CANn_MODE.rst* field is set to 1.

15.3.3 Normal Mode

In normal mode, the CAN controller performs frame reception and transmission. Enter normal mode from reset mode by setting the *CANn_MODE.lom* to 0 and the *CANn_MODE.rst* field to 0 simultaneously.

15.3.4 Listen-Only Mode

Listen-only mode can be used to analyze traffic on the CAN bus without affecting the bus. In listen-only mode, the CAN controller gives no acknowledgment for any message, whether received successfully or for an error. The error counters are stopped at their current values. Listen-only mode can also be used for automatic bit rate detection without disturbing traffic on the network.

Enter listen-only mode by performing the following steps:

1. Place the CAN controller into the reset state by setting the *CANn_MODE.rst* field to 1.
2. Perform any other CAN controller configuration required.
3. Set the *CANn_MODE* register to 2 (*CANn_MODE.rst* = 0 and *CANn_MODE.lom* = 1) to enter listen-only mode.

Note: The CANn_MODE.rst and CANn_MODE.lom fields must be written simultaneously to enter listen-only mode.

15.3.5 Test Mode

Two test modes are supported by the CAN controller, loop back and loop back, with the transmit pin disconnected.

15.3.5.1 Loop Back Mode

The CAN controller can be put into loop back mode by setting the *CANn_TEST.lben* field to 1. This field can only be set to 1 while the CAN controller is in the reset state (*CANn_MODE.rst* = 1). Once loop back mode is enabled, set *CANn_MODE.rst* to 0 to enter normal mode.

15.3.5.2 Loop Back Mode with Transmit Disconnected

The CAN controller supports loop back mode with the transmit pin disconnected. In this mode, all messages the device transmits are looped back to the receive interface, and the CAN transmit pin is left in a recessive state. Enter loop back mode with the transmit pin disconnected by performing the following steps:

1. Place the CAN controller into the reset state by setting *CANn_MODE.rst* to 1.
2. Set the *CANn_TEST* register to 3 (*CANn_TEST.txc* = 1 and *CANn_TEST.lben* = 1) to enable both loop back mode and disconnect the transmit pin.
3. Perform any other controller configuration and then enter normal mode by setting *CANn_MODE.rst* to 0.

15.4 Arbitration Lost Capture

The CAN controller is able to identify the exact CAN bit stream position where the arbitration was lost. When arbitration is lost, an arbitration lost interrupt occurs (*CANn_STAT.al* = 1). The bit position where the arbitration was lost is stored in the *CANn_ALC.alc* field. Software must clear the CANn_INTFL.al interrupt to initiate the arbitration lost capture function for the next CAN communication. *Table 15-2* shows the mapping of the *CANn_ALC.alc* field to the identifier of where the arbitration was lost. For example, when arbitration is lost at identifier bit 20 in an extended frame, the *CANn_ALC.alc* field contains the value 8.

Table 15-2: CANn_ALC.alc Field Mapped to Arbitration Lost Identifier

***CANn_ALC.alc* bits**					**Decimal Value**	**Description**
***alc*[4]**	***alc*[3]**	***alc*[2]**	***alc*[1]**	***alc*[0]**		
0	0	0	0	0	0	Arbitration lost in ID28 / 10
0	0	0	0	1	1	Arbitration lost in ID27 / 9
0	0	0	1	0	2	Arbitration lost in ID26 / 8
0	0	0	1	1	3	Arbitration lost in ID25 / 7

CANn_ALC.alc bits					Decimal Value	Description
alc[4]	alc[3]	alc[2]	alc[1]	alc[0]		
0	0	1	0	0	4	Arbitration lost in ID24 / 6
0	0	1	0	1	5	Arbitration lost in ID23 / 5
0	0	1	1	0	6	Arbitration lost in ID22 / 4
0	0	1	1	1	7	Arbitration lost in ID21 / 3
0	1	0	0	0	8	Arbitration lost in ID20 / 2
0	1	0	0	1	9	Arbitration lost in ID19 / 1
0	1	0	1	0	10	Arbitration lost in ID18 / 0
0	1	0	1	1	11	Arbitration lost in SRTR / RTR
0	1	1	0	0	12	Arbitration lost in IDE bit
0	1	1	0	1	13	Arbitration lost in ID17[1]
0	1	1	1	0	14	Arbitration lost in ID16[1]
0	1	1	1	1	15	Arbitration lost in ID15[1]
1	0	0	0	0	16	Arbitration lost in ID14[1]
1	0	0	0	1	17	Arbitration lost in ID13[1]
1	0	0	1	0	18	Arbitration lost in ID12[1]
1	0	0	1	1	19	Arbitration lost in ID11[1]
1	0	1	0	0	20	Arbitration lost in ID10[1]
1	0	1	0	1	21	Arbitration lost in ID9[1]
1	0	1	1	0	22	Arbitration lost in ID8[1]
1	0	1	1	1	23	Arbitration lost in ID7[1]
1	1	0	0	0	24	Arbitration lost in ID6[1]
1	1	0	0	1	25	Arbitration lost in ID5[1]
1	1	0	1	0	26	Arbitration lost in ID4[1]
1	1	0	1	1	27	Arbitration lost in ID3[1]
1	1	1	0	0	28	Arbitration lost in ID2[1]
1	1	1	0	1	29	Arbitration lost in ID1[1]
1	1	1	1	0	30	Arbitration lost in ID0[1]
1	1	1	1	1	31	Arbitration lost in RTR
1: Extended frame messages only						

15.5 Acceptance Codes and Filtering

The acceptance code filter enables the CAN controller to only receive messages when the identifier bits of the incoming messages are equal to the predefined bits within the acceptance filter registers. The acceptance filters are defined using the acceptance code registers () and the acceptance mask registers (). The acceptance code registers define the bit patterns of the messages to be received, and the corresponding acceptance mask registers are used to enable or disable each of the bits in the pattern. The CAN controller supports either a single acceptance filter of 4 bytes or dual acceptance filters of 2 bytes.

15.5.1 Single Acceptance Filter

In single acceptance filter mode (*CANn_MODE.afm* = 1), a single 4 byte filter is used based on the CAN acceptance filter registers (*CANn_ACR8[0]*:*CANn_ACR8[3]*) and the CAN acceptance mask filter registers (*CANn_AMR8[0]*:*CANn_AMR8[3]*). *Figure 15-3* shows how the single acceptance filter applies to standard frame message, and *Figure 15-4* shows how the single acceptance filter applies to extended frame format message. If a standard frame format message is received, the complete identifier, including the RTR bit and the first two data bytes, if received, are used for acceptance filtering. Messages are also accepted if there is no data byte as long as the identifier matches, and messages with one data byte are accepted if they match up to the first data byte. If an extended frame format message is received in single filter mode, the complete identifier, including the RTR bit is used for acceptance filtering. All enabled single bit comparisons must match to signal acceptance for both standard frame format and extended frame format messages.

Configure the filter by setting the desired bits in the acceptance filter registers (*CANn_ACR8[0]*:*CANn_ACR8[3]*). Enable each desired bit using the corresponding acceptance mask register bits (*CANn_AMR8[0]*:*CANn_AMR8[3]*). Care must be taken

when the filter registers are accessed in 16-bit or 32-bit wide accesses to ensure the mapping of the bits are correct. See *Figure 15-3* and *Figure 15-4* for the mapping of the 8-bit registers to the CAN identifier and data bytes.

Figure 15-3: Single Acceptance Filter for Standard Frame Message

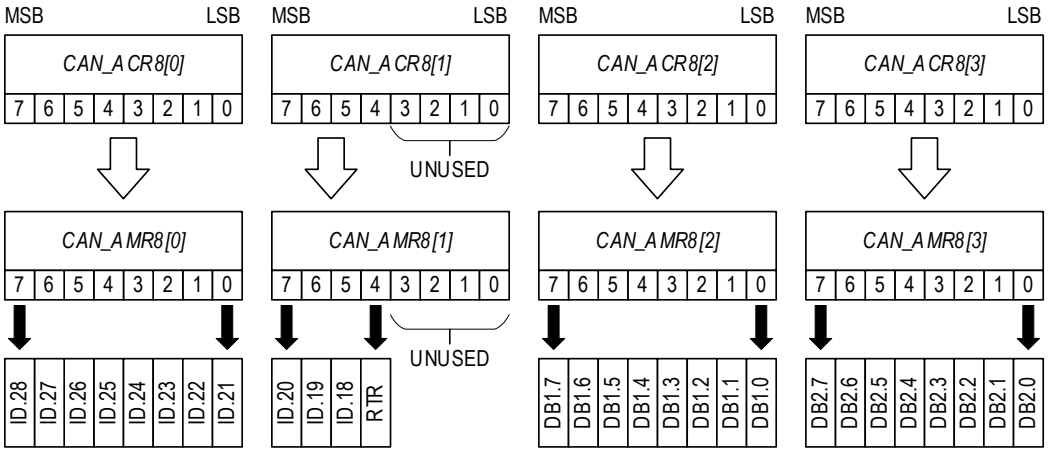
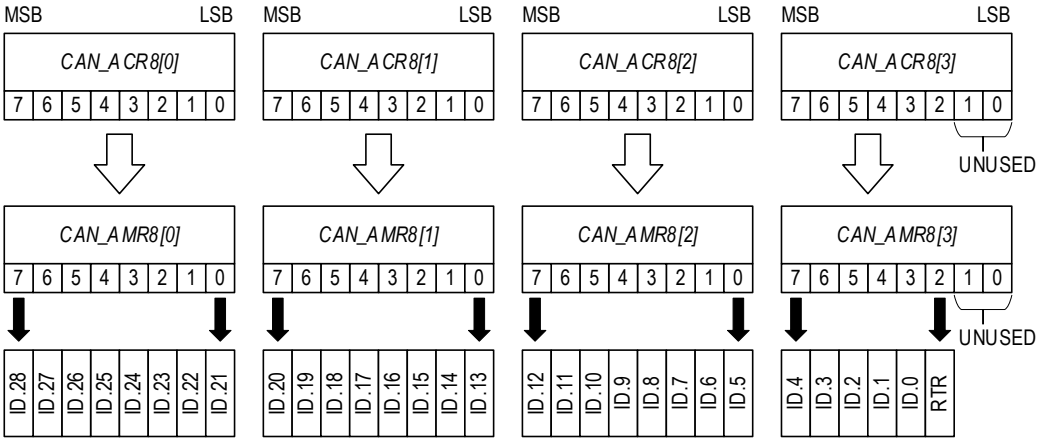


Figure 15-4: Single Acceptance Filter for Extended Frame Messages



15.5.2 Dual Acceptance Filter

In dual acceptance filter mode (*CANn_MODE.afm* = 0), two short filters can be defined. Any received message is compared with both filters to determine if the message should be received. If one or both of the message filters signal acceptance, the message is received.

For a standard frame message, the first filter compares the complete standard identifier, including the RTR bit and the first data byte of the message. The second filter compares only the complete standard identifier, including the RTR bit. See *Figure 15-5* for the acceptance code register mapping to the identifier and data bits received for a standard frame message. For successful reception, all single bit comparisons of at least one filter must match.

If an extended frame message is received in dual filter mode, both filters compare the first two bytes of the extended identifier range only. For successful reception, all single bit comparisons of at least one filter must match.

Figure 15-5: Dual Filter for Standard Frame Messages

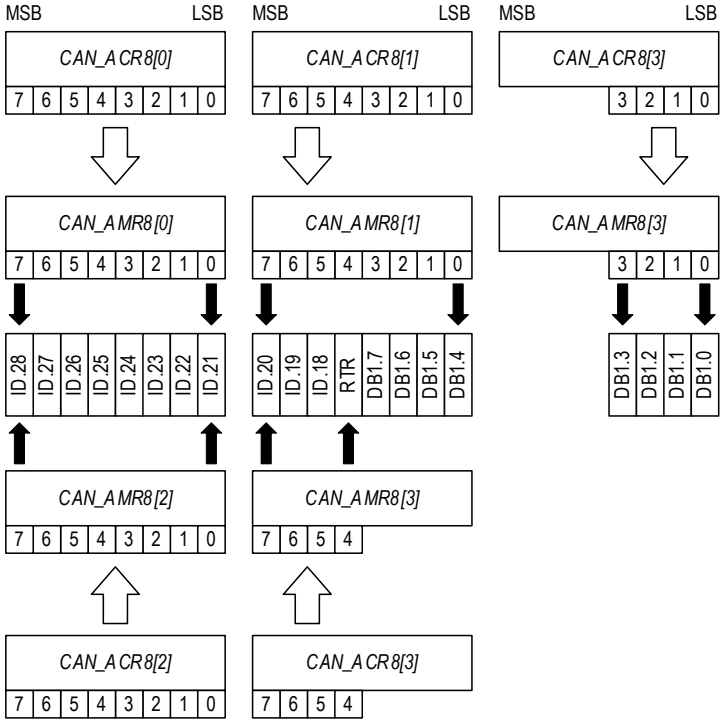
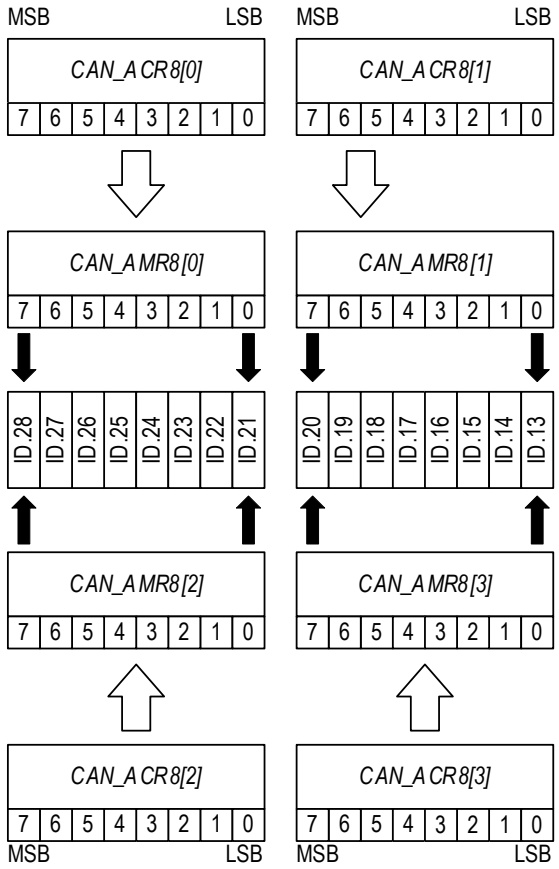


Figure 15-6: Dual Acceptance Filter for Extended Frame Messages



15.6 FIFO Interface

The CAN peripheral supports a 128 byte transmit FIFO and a 256 byte receive FIFO. The FIFOs are accessible in 8-bit, 16-bit, and 32-bit widths. Write to the transmit FIFO using the *CANn_TXFIFO32*, *CANn_TXFIFO16[1]*:*CANn_TXFIFO16[0]*, and the CANn_TXFIFO8[3]:CANn_TXFIFO8[0] registers. Read from the receive FIFO in 32-bit, 16-bit, and 8-bit width by reading from the corresponding *CANn_RXFIFO32*, *CANn_RXFIFO16[1]*:*CANn_RXFIFO16[0]*, and *CANn_RXFIFO8[3]*:*CANn_RXFIFO8[0]* registers.

Table 15-3 shows the maximum number of identical standard and extended messages that can be stored in the receive FIFO based on the data length code (DLC).

Table 15-3: Maximum Number of Identical Messages in the Receive FIFO

DLC	Number of Standard Messages	Number of Extended Messages
0	64	32
1	64	32
2	32	32
3	32	32
4	32	21
5	32	21
6	21	21
7	21	21
8	21	16
9	16	12
10	12	10
11	10	9
12	9	8
13	7	6
14	4	4
15	3	3

15.6.1 Memory Buffer FIFO Layout for Base Frames

Messages to send and receive are stored in the transmit and receive FIFO. The transmit FIFO can hold one message at a time. The receive FIFO buffer can hold many messages at a time. *Table 15-4* shows the transmit and receive FIFO buffer layout for Standard Frame messages.

Table 15-4: Transmit and Receive FIFO Buffer Layout for Standard Frames

Offset	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
0x00	DATA 1 (DLC<1 then free)								ID[2:0]			Reserved	ESI	Reserved			ID[10:3]								IDE	RRS/RTR	FDF	BRS	DLC			
0x01	DATA 5 (DLC<5 then free)								DATA 4 (DLC<4 then free)								DATA 3 (DLC<3 then free)								DATA 2 (DLC<2 then free)							
0x02	DATA 9 (DLC<9 then free)								DATA 8 (DLC<8 then free)								DATA 7 (DLC<7 then free)								DATA 6 (DLC<6 then free)							
0x03	DATA 13 (DLC<10 then free)								DATA 12 (DLC<9 then free)								DATA 11 (DLC<9 then free)								DATA 10 (DLC<9 then free)							
0x04	DATA 17 (DLC<11 then free)								DATA 16 (DLC<10 then free)								DATA 15 (DLC<10 then free)								DATA 14 (DLC<10 then free)							
0x05	DATA 21 (DLC<11 then free)								DATA 20 (DLC<11 then free)								DATA 19 (DLC<11 then free)								DATA 18 (DLC<11 then free)							
0x06	DATA 25 (DLC<12 then free)								DATA 24 (DLC<11 then free)								DATA 23 (DLC<11 then free)								DATA 22 (DLC<11 then free)							
…																																
0x0F	DATA 61 (DLC<15 then free)								DATA 60 (DLC<15 then free)								DATA 59 (DLC<15 then free)								DATA 58 (DLC<15 then free)							
0x10	free								DATA 64 (DLC<15 then free)								DATA 63 (DLC<15 then free)								DATA 62 (DLC<15 then free)							

Table 15-5: Standard Frame Fields

Field	Description
IDE	**Identifier Extension Flag** This bit specifies the type of frame and is valid in classic CAN and CAN FD frames. 0: Standard Message Identifier. 1: Extended Message Identifier.
RTR/RRS	**Remote Transmission Request** This bit specifies the frame is a standard CAN data frame or a standard CAN remote frame. For CAN FD format this bit indicates bus state at RRS position of the frame. 0: Standard CAN data frame. 1: Standard CAN remote frame.
FDF	**FD Frame Format Indicator** This field indicates a CAN FD frame. 0: Classic CAN frame format. 1: CAN FD frame format.
BRS	**Baud Rate Switch Bit** 0: Frame without baud rate switching. 1: Frame with baud rate switching. *Note: This field is only available with CAN FD mode.*
ESI	**Error State Indicator** 0: Transmitter of this message is Active Error node. 1: Transmitter of this message is Passive Error node. *Note: This field is only available with CAN FD mode.*
DLC[3:0]	**Data Length Code** Frame data length code field of classic CAN and CAN FD frame.
ID[10:0]	**Standard Message Identifier**
DATA N	**Data Bytes**

15.6.1.1 Message Identifier

The identifier consists of 11 bits (ID[10] to ID[0]). ID[10] is the most significant bit, which is transmitted first on the bus during the arbitration process. The identifier acts as the message's name. It is used in a receiver for acceptance filtering and also determining the bus access priority during the arbitration process. The lower binary value of the identifier means higher priority for the message. This is due to a larger number of leading dominant bits during arbitration.

15.6.1.2 Remote Transmission Request (RTR/RRS)

If this bit is set, a remote frame is transmitted on the CAN bus. This means that no data bytes are included within this frame. Nevertheless, it is necessary to specify the correct data length code, which depends on the corresponding data frame with the same identifier coding. If the RTR bit is not set, a data frame is sent, including the number of data bytes as specified by the data length code. In CAN FD mode, remote frames do not exist.

15.6.1.3 FD Frame (FDF)

This bit distinguishes between a classic CAN frame and a CAN FD frame. When this bit is set in the transmit data buffer in FD mode (*CANn_FDCTRL.fden* = 1), a CAN FD frame will be sent. If FD mode is disabled (*CANn_FDCTRL.fden* = 0), this field is ignored and a classic can frame is sent. If the FDF bit is set to 1 in received data, this indicates the received frame was sent in FD mode, when the CAN peripheral is in FD mode (*CANn_FDCTRL.fden* = 1). If the CAN peripheral is set to classic can (*CANn_FDCTRL.fden* = 0), this field is always set to 0 by the receive hardware.

15.6.1.4 Bit Rate Switch (BRS)

In transmit data, this bit determines that the FD frame will be sent with bit rate switching in the data phase. In receive mode, the data received where this bit is set means that the received frame included bit time switching if the CAN peripheral is in FD mode (*CANn_FDCTRL.fden* = 1).

15.6.1.5 Error State Indicator (ESI)

This field is only valid when the CAN peripheral is in FD mode (*CANn_FDCTRL.fden* = 1), and the received/transmitted frame is an FD frame. This bit indicates that the transmitter of the message was in passive error node if 1 or active error node if 0.

15.6.1.6 Data Length Code (DLC)

The number of bytes in the data field of a message is coded by the data length code. At the start of a remote frame transmission, the data length code is not considered due to the RTR bit being a logic 1 (remote). It denotes the number of transmitted/received data bytes to be logic 0. Nevertheless, the data length code must be specified correctly to avoid bus errors if two CAN controllers start a remote frame transmission with the same identifier simultaneously. The range of the data byte count is 0 to 64 bytes and is coded, as shown in *Table 15-6*.

Table 15-6: DLC Coding to Data Byte Count

Number of Bytes	DLC[3]	DLC[2]	DLC[1]	DLC[0]
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
12	1	0	0	1
16	1	0	1	0
20	1	0	1	1
24	1	1	0	0
32	1	1	0	1
48	1	1	1	0
64	1	1	1	1

In classic CAN, if the selected value is greater than 8, the maximum number of 8 bytes is transmitted in the data frame with the data length code specified in the DLC.

15.6.1.7 Data Field

The number of transferred data bytes is determined by the data length code. The first bit transmitted is the most significant bit of the data byte 1.

Note: The number of transferred data bytes is defined by the DLC. The first data bit transmitted is the MSB of data byte 1. The received data length code located in the frame information byte represents the real data length code, which may be greater than 8.

Note: For transmission in classic CAN frame format, when the DLC is specified greater than 8, only 8 bytes of data are transmitted. When the RTR bit is set to 1, no data bytes are transmitted, regardless of the value of the DLC field.

Note: ID(10) is the first bit transmitted after SOF.

15.6.2 Memory Buffer FIFO Layout for Extended Frame Messages

Messages to send and receive are stored in the transmit and receive FIFO. The transmit FIFO can hold one message at a time. The receive FIFO buffer can hold many messages at a time. *Table 15-7* shows the transmit and receive FIFO buffer layout for Extended Frame messages.

Table 15-7: Transmit and Receive FIFO Buffer Layout for Extended Frames

Offset	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
0x00	ID[12:5]								ID[20:13]								ID[28:21]								IDE	SRR	FDF	BRS	DLC			
0x01	DATA 3 (DLC<3 then free)								DATA 2 (DLC<2 then free)								DATA 1 (DLC<1 then free)								ID[4:0]					RRS\RTR	ESI	Reserved
0x02	DATA 7 (DLC<9 then free)								DATA 6 (DLC<8 then free)								DATA 5 (DLC<7 then free)								DATA 4 (DLC<6 then free)							
0x03	DATA 11 (DLC<10 then free)								DATA 10 (DLC<9 then free)								DATA 9 (DLC<9 then free)								DATA 8 (DLC<9 then free)							
0x04	DATA 15 (DLC<11 then free)								DATA 14 (DLC<10 then free)								DATA 13 (DLC<10 then free)								DATA 12 (DLC<10 then free)							
0x05	DATA 19 (DLC<11 then free)								DATA 18 (DLC<11 then free)								DATA 17 (DLC<11 then free)								DATA 16 (DLC<11 then free)							
…																																
0x0F	DATA 59 (DLC<15 then free)								DATA 58 (DLC<15 then free)								DATA 57 (DLC<15 then free)								DATA 56 (DLC<15 then free)							
0x10	DATA 64 (DLC<15 then free)								DATA 62 (DLC<15 then free)								DATA 61 (DLC<15 then free)								DATA 60 (DLC<15 then free)							
0x11	free								free								free								DATA 64 (DLC<15 then free)							

Table 15-8: Extended Frame Fields

Field	Description
IDE	**Identifier Extension Flag** This bit specifies that the frame is using Standard Identifier or Extended Identifier. Valid in both classic CAN and FD CAN frames. 0: Standard Message Identifier. 1: Extended Message Identifier.
SRR	**Substitute Remote Request** This bit is transmitted in Extended formats (classic and FD) at the position of the RTR bit in base formats. This bit is always transmitted as recessive in extended formats (classic CAN or CAN FD). 0: Dominant state on bus. 1: Recessive state on bus.
FDF	**FD Frame Format Indicator** This field indicates a CAN FD frame. 0: Classic CAN frame format. 1: CAN FD frame format.
BRS	**Baud Rate Switch Bit** 0: Frame without baud rate switching. 1: Frame with baud rate switching. *Note: This field is only available with CAN FD mode.*
ESI	**Error State Indicator** 0: Transmitter of this message is Active Error node. 1: Transmitter of this message is Passive Error node. *Note: This field is only available with CAN FD mode.*
RTR/RRS	**Remote Transmission Request** This bit specifies the frame is standard CAN data frame or standard CAN remote frame. For CAN FD format this bit indicates bus state at RRS position of the frame. 0: Standard CAN data frame. 1: Standard CAN remote frame. *Note: CAN FD format does not support remote frames. In a CAN FD frame at this position the transmitter always sends the dominant state.*

Field	Description
DLC[3:0]	**Data Length Code** Frame data length code field of classic CAN and CAN FD frame.
ID[10:0]	**Standard Message Identifier**
DATA N	**Data Bytes**

15.6.2.1 Identifier

In the extended frame, the message identifier is divided into two parts:

- Base identifier – 11 bits wide (ID[28] – ID[18]).

Extended identifier – 18 bits wide (ID[17] – ID[0])

The base identifier consists of 11 bits (ID[28] to ID[18]) and is equivalent to the standard frame identifier. It defines the extended frame's base priority.

15.6.2.2 Substitute Remote Request (SRR)

When the CAN peripheral is a transmitter it always sends recessive at this position in extended CAN frame format and extended CAN FD frame format. The receiver accepts recessive or dominant at this position.

15.6.2.3 Remote Transmission Request (RTR/RRS)

A remote frame is transmitted on the CAN bus if this bit is set. This means that no data bytes are included within this frame. Nevertheless, it is necessary to specify the correct DLC, which depends on the corresponding data frame with the same identifier coding. If the RTR bit is not set, a data frame is sent, including the number of data bytes as specified by the data length code. In CAN FD mode, remote frames are not supported.

15.6.2.4 FD Frame (FDF)

This bit distinguishes between a classic CAN frame and a CAN FD frame. When this bit is set in the transmit data buffer in FD mode (*CANn_FDCTRL.fden* = 1), a CAN FD frame is sent. If the *CANn_FDCTRL.fden* bit is cleared, the setting for FDF is ignored, and a classic CAN frame is sent. If the FDF bit is a logical one in the received data, the received frame was sent in FD mode if the CAN peripheral is in FD mode (*CANn_FDCTRL.fden* = 1). When the CAN peripheral is set to classic CAN mode, the FDF bit is always equal to 0.

15.6.2.5 Bit Rate Switch (BRS)

When transmitting, if this bit is set to 1, the FD frame is sent with bit rate switching in the data phase. When receiving data, if this bit is set to 1, the received frame had bit rate switching if the peripheral is in FD mode (*CANn_FDCTRL.fden* = 1).

15.6.2.6 Error State Indicator (ESI)

This bit is only valid if the CAN peripheral is in FD mode (*CANn_FDCTRL.fden* = 1) and the received/transmitted frame is an FD frame. If this bit is a 1, the transmitter of the message was in passive error node. If this bit is a 0, the transmitter of the message was an active error node.

15.6.2.7 Data Length Code (DLC)

The number of bytes in the data field of a message is coded by the data length code. At the start of a remote frame transmission, the data length code is not considered, due to the RTR bit being a logic 1 (remote). It denotes the number of transmitted/received data bytes to be logic 0. Nevertheless, the data length code must be specified correctly to avoid bus errors, if two CAN controllers start a remote frame transmission with the same identifier simultaneously. The range of the data byte count is 0 to 64 bytes and is coded, as shown in *Table 15-6*.

Table 15-9: DLC Coding to Data Byte Count

Number of Bytes	DLC[3]	DLC[2]	DLC[1]	DLC[0]
0	0	0	0	0
1	0	0	0	1

Number of Bytes	DLC[3]	DLC[2]	DLC[1]	DLC[0]
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
12	1	0	0	1
16	1	0	1	0
20	1	0	1	1
24	1	1	0	0
32	1	1	0	1
48	1	1	1	0
64	1	1	1	1

In classic CAN, if the selected value is greater than 8, the maximum number of 8 bytes is transmitted in the data frame with the data length code specified in the DLC.

15.6.2.8 Data Field

The DLC determines the number of transferred data bytes. The first bit transmitted is the most significant bit of data byte 1.

Note:. The DLC defines the number of transferred data bytes. The first data bit transmitted is the MSB of data byte 1. The received data length code located in the frame information byte represents the real data sent length code, which may be greater than 8.

Note: The DLC defines the number of transferred data bytes. The first data bit transmitted is the MSB of data byte 1. The received data length code located in the frame information byte represents the real sent data length code, which may be greater than 8.

Note: For transmission in classic CAN frame format, when DLC is specified as greater than 8, only 8 bytes of data are transmitted. When the RTR bit is set to 1, no data bytes are transmitted, regardless of the DLC value.

Note: ID[28] is the first bit transmitted after the SOF.

15.7 CAN Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 15-10*. 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 1-1* 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 15-10: CAN Registers

Offset	Register Name	Register Width	Description
[0x0000]	*CANn_MODE*	8-bit	Mode Register
[0x0001]	*CANn_CMD*	8-bit	Command Register
[0x0002]	*CANn_STAT*	8-bit	Status Register
[0x0003]	*CANn_INTFL*	8-bit	Interrupt Flag Register
[0x0004]	*CANn_INTEN*	8-bit	Interrupt Enable Register
[0x0005]	*CANn_RMC*	8-bit	Receive Message Counter Register r
[0x0006]	*CANn_BUSTIM0*	8-bit	Bus Timing 0 Register
[0x0007]	*CANn_BUSTIM1*	8-bit	Bus Timing 1 Register
[0x000B]:[0x0008]	*CANn_TXFIFO8[3]*:*CANn_TXFIFO8[0]*	8-bit	8-Bit Transmit FIFO 3:0 Registers
[0x000A]:[0x0008]	*CANn_TXFIFO16[1]*:*CANn_TXFIFO16[0]*	16-bit	16-Bit Transmit FIFO 1:0 Registers

Offset	Register Name	Register Width	Description
[0x0008]	*CANn_TXFIFO32*	32-bit	32-Bit Transmit FIFO Register
[0x000F]:[0x000C]	*CANn_RXFIFO8[3]*:*CANn_RXFIFO8[0]*	8-bit	8-Bit Receive FIFO 3:0 Registers
[0x000E]:[0x000C]	*CANn_RXFIFO16[1]*:*CANn_RXFIFO16[0]*	16-bit	16-Bit Receive FIFO 1:0 Registers
[0x000C]	*CANn_RXFIFO32*	32-bit	32-Bit Receive FIFO Register
[0x0013]:[0x0010]	*CANn_ACR8[3]*:*CANn_ACR8[0]*	8-bit	8-Bit Acceptance Code 3:0 Response Registers
[0x0012]:[0x0010]	*CANn_ACR16[1]*:*CANn_ACR16[0]*	16-bit	16-Bit Acceptance Code 1:0 Response Registers
[0x0010]	*CANn_ACR32*	32-bit	32-Bit Acceptance Code Response Register
[0x0017]:[0x0014]	*CANn_AMR8[3]*:*CANn_AMR8[0]*	8-bit	8-Bit Acceptance Mask 3:0 Response Registers
[0x0016]:[0x0014]	*CANn_AMR16[1]*:*CANn_AMR16[0]*	16-bit	16-Bit Acceptance Mask 1:0 Response Registers
[0x0014]	*CANn_AMR32*	32-bit	32-Bit Acceptance Mask Response Register
[0x0018]	*CANn_ECC*	8-bit	Error Code Capture Register
[0x0019]	*CANn_RXERR*	8-bit	Receive Error Counter Register
[0x001A]	*CANn_TXERR*	8-bit	Transmit Error Counter Register
[0x001B]	*CANn_ALC*	8-bit	Arbitration Lost Code Capture Register
[0x001C]	*CANn_NBT*	32-bit	Nominal Bit Time Extended Register
[0x0020]	*CANn_DBT_SSPP*	32-bit	Data Bit Time and Second Sample Point Position Register
[0x0024]	*CANn_FDCTRL*	8-bit	FD Control Register
[0x0025]	*CANn_FDSTAT*	8-bit	FD Status Register
[0x0026]	*CANn_DPERR*	8-bit	Data Phase Error Register
[0x0027]	*CANn_APERR*	8-bit	Arbitration Phase Error Counter Register
[0x0028]	*CANn_TEST*	8-bit	Test Register
[0x0029]	*CANn_WUPCLKDIV*	8-bit	Wake-up Timer Clock Divisor Register
[0x002A]	*CANn_WUPFT*	16-bit	Wake-up Filter Time Low Register
[0x002C]	*CANn_WUPET*	16-bit	Wake-up Expire Time Low Register
[0x0030]	*CANn_RXDCNT*	16-bit	Receive FIFO Data Count Register
[0x0032]	*CANn_TXSCNT*	8-bit	Transmit Buffer Space Counter Register
[0x0033]	*CANn_TXDECMP*	8-bit	Transmitter Delay Compensation Register
[0x0034]	*CANn_EINTFL*	8-bit	Extended Interrupt Flag Register
[0x0035]	*CANn_EINTEN*	8-bit	Extended Interrupt Mask Register
[0x0036]	*CANn_RXTO*	16-bit	Receive Timeout Register

15.7.1 Register Details

Table 15-11: Mode Register

Mode			CANn_MODE	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
7	slp	R/W	0	**Sleep Mode** Setting this field to 1 triggers a sleep request to the CAN controller. The CAN controller waits for the CAN bus to be inactive and then enters sleep mode. After writing a 1 to this field, hardware does not set the field to 1 until the CAN bus is inactive and the controller enters sleep. 0: CAN is in active mode. 1: CAN is in sleep mode. *Note: This field cannot be set to 1 if CANn_MODE.rst is set to 1. It is an invalid state to set both this field and CANn_MODE.rst to 1 simultaneously.*
6	dma	R/W	0	**DMA Mode Enable** 0: DMA mode disabled 1: DMA mode enabled

Mode			CANn_MODE	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
5:3	rxtrig	R/W	0	**Receive FIFO Trigger Level** Set this field to the number of 32-bit words in the receive FIFO to trigger a DMA operation. 0b000: 1 0b001: 4 0b010: 8 0b011: 16 0b100: 32 0b101: 64 0b111: Reserved *Note: This field is only used if DMA mode is enabled (CANn_MODE.dma = 1).*
2	rst	R/W	1	**Reset Mode** Set this field to 1 to enable reset mode of the CAN controller. The CAN controller must be in this mode to configure operation. In reset mode, frame reception and transmission are disabled. See *Operating Modes* for details of the operating modes. When this field is 0, the CAN controller operating mode is controlled using the *CANn_MODE.lom* field. 0: CAN controller not in reset mode. 1: CAN controller in reset mode.
1	lom	R/W	0	**Listen Only Mode Enable** Set this field to 1 to enable listen-only mode or 0 to enable normal operation. See *Operating Modes* for details. 0: Normal mode 1: Listen only mode *Note: This field is only used if CANn_MODE.rst is set to 0.*
0	afm	R/W	0	**Hardware Acceptance Filter Scheme** This field selects either dual or single filter for the hardware acceptance filter scheme. 0: Dual filter 1: Single filter *Note: This field can only be modified if CANn_MODE.rst is set to 0.*

Table 15-12: Command Register

Command			CANn_CMD	[0x0001]
Bits	**Field**	**Access**	**Reset**	**Description**
7:3	-	W	0	**Reserved**
2	txreq	W	0	**Transmit Request** Set this field to 1 to initiate a frame transmission. Setting this field to 1 simultaneously with *CANn_CMD.abort* to 1 initiates a single shot transmission. A single shot transmission does not perform a lost frame retransmission if a bus error or arbitration occur.
1	abort	W	0	**Abort Transmission** If a transmission is started by setting *CANn_CMD.txreq* to 1 and the transmission has not yet started, setting this field to 1 aborts the transmission. Simultaneously setting this field to 1 and *CANn_CMD.txreq* to 1 initiates a single shot transmission.
0	-	W	0	**Reserved**

Table 15-13: Status Register

Status			CANn_STAT	[0x0002]
Bits	**Field**	**Access**	**Reset**	**Description**
7	rxbuf	R	0	**Receive Buffer Status** This field indicates if at least one message is in the receive FIFO. When set, at least one message is in the receive FIFO, when clear, no messages are in the receive FIFO.
6	dor	R	0	**Data Overrun Status** This field indicates if a receive overrun occurred. If set, a receive overrun error occurred.
5	txbuf	R	1	**Transmit Buffer Status** This field indicates if the transmit buffer is available for software to write to it. If this field reads 0, the transmit buffer is locked and cannot be written. The transmit buffer is locked by hardware when a message is being transmitted or a transmission is pending. 0: Transmit buffer is locked. 1: Transmit buffer unlocked.
4	-	RO	0	**Reserved**
3	rx	R	0	**Receive Status** This field is set to 1 by the hardware when a message is being received.
2	tx	R	0	**Transmit Status** This field is set to 1 by hardware when the CAN controller is transmitting a message.
1	err	R	0	**Error Status** This field is set by hardware when at least one of the CAN error counters reached the warning limit of 96.
0	bus_off	R	0	**Bus Off Status** If this field is set, the CAN controller is in the bus off state and cannot transmit or receive frames. This field is set to 1 by hardware when the transmit error counter exceeds 255. When this field is set by hardware, the CAN controller automatically is placed into reset mode, and if enabled, an error warning interrupt is generated. The transmit error counter is then set to 127 and the receive error counter is cleared. The CAN controller remains in reset until the software clears the reset bit. Once the reset state is cleared, the CAN controller waits for 128 occurrences of the bus free signal (11 consecutive recessive bits) counting down the transmit error counter. When the transmit error counter reaches 0, the bus_off field is cleared and the bus is in the bus on state, the error counters are reset, and an error warning interrupt is generated if enabled. 0: Bus is in the bus on state 1: Bus in in the bus off state

Table 15-14: Interrupt Flag Register

Interrupt Flag			CANn_INTFL	[0x0003]
Bits	**Field**	**Access**	**Reset**	**Description**
7	wu	R/W1C	0	**Wake-up Interrupt** This field is set to 1 when the CAN controller is woken up from sleep by either a wake-up pattern detection or by software clearing the *CANn_MODE.slp* bit. Write 1 to clear.
6	al	R/W1C	0	**Arbitration Lost Interrupt** This field is set to 1 when the device is transmitting a message and loses arbitration, thus becoming a receiver. Read the *CANn_ALC* register to determine the location of the bit in the frame where arbitration lost occurred. Write 1 to clear.

Interrupt Flag			CANn_INTFL	[0x0003]
Bits	**Field**	**Access**	**Reset**	**Description**
5	erwarn	R/W1C	0	**Error Warning Interrupt** This field is set to 1 when there is a change of state of the error status (*CANn_STAT.err*) or bus status (*CANn_STAT.bus_off*). Write 1 to clear.
4	erpsv	R/W1C	0	**Error Passive Interrupt** This field is set to 1 when a state change to or from passive occurs (i.e., a state change of active-to-passive or passive-to-active). Write 1 to clear.
3	rx	R/W1C	0	**Receive Interrupt** This field is set to 1 when at least 1 message is in the receive FIFO. After reading a message from the receive FIFO, the software must write this field to 1 to decrement the receive message counter (RMC). The receive message counter is not automatically decremented by reading a message from the FIFO.
2	tx	R/W1C	0	**Transmission Interrupt** This field is set to 1 after a successful transmission occurs. Write 1 to clear.
1	berr	R/W1C	0	**Bus Error Interrupt** This field is set when a bus error occurs while transmitting or receiving a message.
0	dor	R/W1C	0	**Data Overrun Interrupt** This field is set when a receive FIFO overrun occurs. Write 1 to clear.

Table 15-15: Interrupt Enable Register

Interrupt Enable			CANn_INTEN	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
7	wu	R/W	0	**Wake-up Interrupt Enable** 0: Disabled 1: Enabled
6	al	R/W	0	**Arbitration Lost Interrupt Enable** 0: Disabled 1: Enabled
5	erwarn	R/W	0	**Error Warning Interrupt Enable** 0: Disabled 1: Enabled
4	erpsv	R/W	0	**Error Passive Interrupt Enable** 0: Disabled 1: Enabled
3	rx	R/W	0	**Receive Interrupt Enable** 0: Disabled 1: Enabled
2	tx	R/W	0	**Transmit Interrupt Enable** 0: Disabled 1: Enabled
1	berr	R/W	0	**Bus Error Interrupt Enable** 0: Disabled 1: Enabled
0	dor	R/W	0	**Data Overrun Interrupt Enable** 0: Disabled 1: Enabled

Table 15-16: Receive Message Counter Register

Receive Message Counter			CANn_RMC	[0x0005]
Bits	**Field**	**Access**	**Reset**	**Description**
7:5	-	RO	0	**Reserved**

Receive Message Counter			CANn_RMC	[0x0005]
Bits	**Field**	**Access**	**Reset**	**Description**
4:0	num_msgs	R	0	**Number of Frames Stored in the Receive FIFO** This field is the number of fields stored in the receive FIFO. This field is incremented by hardware on each successful frame reception. This field is decremented by clearing the receive interrupt (*CANn_INTFL.rx*) field.

Table 15-17: Bus Timing 0 Register

Bus Timing 0			CANn_BUSTIM0	[0x0006]
Bits	**Field**	**Access**	**Reset**	**Description**
7:6	sjw	R/W	0	**Synchronization Jump Width** Set this field to the maximum number of time quanta (t_q) for the SJW used during resynchronization. The following equation defines the maximum number of clock cycles a bit period may be changed by one resynchronization. $t_{SJW} = t_{SCLK} \times (CANn\_BUSTIM0.sjw + 1)$
5:0	br_clkdiv	R/W	0	**Baud Rate Prescaler** Set this field to the desired BRP to achieve the nominal bit rate. See *Bit Timing* for details on setting the nominal bit rate.

Table 15-18: Bus Timing 1 Register

Bus Timing 1			CANn_BUSTIM1	[0x0007]
Bits	**Field**	**Access**	**Reset**	**Description**
7	sam	R/W	0	**Number of Bus Level Samples** This field selects either a single bus sample or three bus samples per bit. When set to 0, the single sample occurs at the end of TSEG1. When this field is set to 1, the three samples are used in a voting scheme and occur on the last three time quanta of TSEG1. See *Figure 15-2* for more information. A single sample is recommended for high-speed CAN buses. Three samples are recommended for low/medium speed CAN buses where there is a benefit from filtering spikes. 0: Single sample 1: 3 samples
6:4	tseg2	R/W	0	**TSEG2 Time Quanta Select** Set this field to the number of time quanta for TSEG2. See *Bit Timing* for details of the nominal bit timing. 0: 1 t_q 1: 2 t_q 2: 3 t_q 3: 4 t_q 4: 5 t_q 5: 6 t_q 6: 7 t_q 7: 8 t_q

Bus Timing 1			CANn_BUSTIM1	[0x0007]
Bits	**Field**	**Access**	**Reset**	**Description**
3:0	tseg1	R/W	0	**TSEG1 Time Quanta Select** Set this field to the number of time quanta for TSEG1. See *Bit Timing* for details of the nominal bit timing. 0: 1 t_q 1: 2 t_q 2: 3 t_q …: … 14: 15 t_q 15: 16 t_q

15.7.1.1 Transmit FIFO Layout

The transmit FIFO is accessible in 8-bit, 16-bit, and 32-bit widths. The FIFO is mapped as four consecutive bytes from address offset [0x0008] through [0x000B]. When writing 32-bit words to the transmit FIFO, the write pointer is incremented after each write. For 16-bit writes, the write pointer is incremented after writing to address offset [0x000A], and when writing 8-bits, the write pointer is incremented after 4 writes. When a successful transmission occurs, the *CANn_INTFL.tx* field is set to 1 by hardware. Clearing the *CANn_INTFL.tx* field to 0 resets the transmit FIFO's internal write pointer to 0.

Table 15-19: 32-Bit Transmit FIFO Register

8-Bit Transmit FIFO 0			CANn_TXFIFO8[0]	[0x0008]
16-Bit Transmit FIFO 0			CANn_TXFIFO16[0]	[0x0008]
32-Bit Transmit FIFO			CANn_TXFIFO32	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	W	0	**8-Bit Transmit FIFO Least Significant Byte Data** Write CAN frame data to this register in 8, 16, or 32-bit widths. A 32-bit write to this address writes to *CANn_TXFIFO8[3]*:*CANn_TXFIFO8[2]*:*CANn_TXFIFO8[1]*:*CANn_TXFIFO16[0]* and automatically increments the internal write pointer. A 16-bit write to this address writes *CANn_TXFIFO8[1]*:*CANn_TXFIFO8[0]*. An 8-bit write to this address writes *CANn_TXFIFO8[0]*.

Table 15-20: 8-Bit Transmit FIFO 1 Register

8-Bit Transmit FIFO 1			CANn_TXFIFO8[1]	[0x0009]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	W	0	**8-Bit Transmit FIFO Data** Write CAN frame data to this register in 8-bit width. An 8-bit write to this register writes *CANn_TXFIFO8[1]*.

Table 15-21: 8-Bit Transmit FIFO 2 Register

8-Bit Transmit FIFO 2			CANn_TXFIFO8[2]	[0x000A]
16-Bit Transmit FIFO 1			CANn_TXFIFO16[1]	[0x000A]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	W	0	**8-Bit Transmit FIFO Data** Write CAN frame data to this register in 8, or 16-bit widths. A 16-bit write to this address writes to *CANn_TXFIFO8[3]*:*CANn_TXFIFO8[2]* and automatically increments the internal write pointer. An 8-bit write to this address writes *CANn_TXFIFO8[2]*.

Table 15-22: 8-Bit Transmit FIFO 3 Register

8-Bit Transmit FIFO 3			CANn_TXFIFO8[3]	[0x000B]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	W	0	**8-Bit Transmit FIFO Most Significant Byte Data** Write CAN frame data to this register in 8-bit width. An 8-bit write to this address writes to *CANn_TXFIFO8[3]* and automatically increments the internal write pointer.

Table 15-23: 32-Bit Receive FIFO Register

8-Bit Receive FIFO			CANn_RXFIFO8[0]	[0x000C]
16-Bit Receive FIFO			CANn_RXFIFO16[0]	[0x000C]
32-Bit Receive FIFO			CANn_RXFIFO32	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R	0	**32-Bit Receive FIFO Data**

Table 15-24: 8-Bit Receive FIFO 1 Register

8-Bit Receive FIFO 1			CANn_RXFIFO8[1]	[0x000D]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R	0	**8-Bit Receive FIFO Data**

Table 15-25: 8-Bit Receive FIFO 2 Register

8-Bit Receive FIFO 2			CANn_RXFIFO8[2]	[0x000E]
16-Bit Receive FIFO 1			CANn_RXFIFO16[1]	[0x000E]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	-	R	0	**16-Bit Receive FIFO Data**

Table 15-26: 8-Bit Receive FIFO 3 Register

8-Bit Receive FIFO 3			CANn_RXFIFO8[3]	[0x000F]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R	0	**8-Bit Receive FIFO Data**

Table 15-27: 32-Bit Acceptance Code Register

32-Bit Acceptance Code			CANn_ACR32	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**32-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-28: 16-Bit Acceptance Code 0 Register

16-Bit Acceptance Code 0			CANn_ACR16[0]	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	-	R/W	0	**16-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-29: 16-Bit Acceptance Code 1 Register

16-Bit Acceptance Code 1			CANn_ACR16[1]	[0x0012]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	-	R/W	0	**16-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-30: 8-Bit Acceptance Code 0 Register

8-Bit Acceptance Code 0			CANn_ACR8[0]	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-31: 8-Bit Acceptance Code 1 Register

8-Bit Acceptance Code 1			CANn_ACR8[1]	[0x0011]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-32: 8-Bit Acceptance Code 2 Register

8-Bit Acceptance Code 2			CANn_ACR8[2]	[0x0012]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-33: 8-Bit Acceptance Code 3 Register

8-Bit Acceptance Code 3			CANn_ACR8[3]	[0x0013]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-34: 32-Bit Acceptance Mask Register

32-Bit Acceptance Mask			CANn_AMR32	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**32-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-35: 16-Bit Acceptance Mask 0 Register

16-Bit Acceptance Mask 0			CANn_AMR16[0]	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	-	R/W	0	**16-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-36: 16-Bit Acceptance Mask 0 Register

16-Bit Acceptance Mask 1			CANn_AMR16[1]	[0x0016]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	-	R/W	0	**16-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-37: 8-Bit Acceptance Mask 0 Register

8-Bit Acceptance Mask 0			CANn_AMR8[0]	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-38: 8-Bit Acceptance Mask 1 Register

8-Bit Acceptance Mask 1			CANn_AMR8[1]	[0x0015]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-39: 8-Bit Acceptance Mask 2 Register

8-Bit Acceptance Mask 2			CANn_AMR8[2]	[0x0016]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-40: 8-Bit Acceptance Mask 3 Register

8-Bit Acceptance Mask 3			CANn_AMR8[3]	[0x0017]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 15-41: Error Code Capture Register

Error Code Capture			CANn_ECC	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
7	rxwrn	R	0	**Receive Error Counter Warning** This field is set when the receive error counter (*CANn_RXERR*) is greater than or equal to 96. *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CANn_INTFL.berr).*
6	txwrn	R	0	**Transmit Error Counter Warning** This field is set when the transmit error counter (*CANn_TXERR*) is greater than or equal to 96. *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CANn_INTFL.berr).*
5	edir	R	0	**The Direction of Transfer When Error Occurred** This field indicates the direction of the error when it occurred and applies to the error codes in bits 4:0 of this register. 0: Transmission 1: Reception *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CANn_INTFL.berr).*
4	acker	R	0	**ACK Error** 0: Normal operation 1: Error occurred *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CANn_INTFL.berr).*
3	frmer	R	0	**Form Error** 0: Normal operation 1: Error occurred *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CANn_INTFL.berr).*

Error Code Capture			CANn_ECC	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
2	crcer	R	0	**CRC Error** 0: Normal operation 1: Error occurred *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CANn_INTFL.berr).*
1	stfer	R	0	**Stuff Error**] 0: Normal operation 1: Error occurred *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CANn_INTFL.berr).*
0	ber	R	0	**Bit Error** 0: Normal operation 1: Error occurred *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CANn_INTFL.berr).*

Table 15-42: Receive Error Counter Register

Receive Error Counter			CANn_RXERR	[0x0019]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R	0	**Receive Error Counter** This field reflects the current value of the receive error counter. If a bus off event occurs (*CANn_STAT.bus_off* = 1), this field is reset to 0 by the hardware.

Table 15-43: Transmit Error Counter Register

Transmit Error Counter			CANn_TXERR	[0x001A]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R	0	**Transmit Error Counter** This field reflects the current value of the transmit error counter. If a bus off event (*CANn_STAT.bus_off* = 1) occurs, this field is set to 127 by the hardware and is decremented the protocol defined time (128 occurrences of the bus free signal). Reading this register during this time gives information about the status of the bus off recovery. *Note: The internal transmit error counter is 9-bits, whereas this register is only 8-bits, so only bits 7:0 of the transmit error counter are readable by software.*

Table 15-44: Arbitration Lost Code Capture Register

Arbitration Lost Code Capture			CANn_ALC	[0x001B]
Bits	**Field**	**Access**	**Reset**	**Description**
7:6	-	RO	0	**Reserved**
5:0	alc	R	0	**Arbitration Lost Bit Position** This field contains the bit position where arbitration was lost. See *Table 15-2* for detailed mapping of this field to the identifier of the arbitration lost. *Note: This field is not updated until software acknowledges the arbitration lost interrupt by clearing the CANn_INTFL.al field.*

Table 15-45: Arbitration Nominal Bit Time Register

Arbitration Nominal Bit Time			CANn_NBT	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:25	nsjw	R/W	0	**Synchronization Jump Width in Arbitration** This field defines the synchronization jump width in CAN FD frames when extended bit time (*CANn_FDCTRL.extbt* = 1) is selected.
24:18	nseg2	R/W	0	**Nominal Segment 2 Time in Arbitration** This field defines the time after the sample point in CAN FD frames when extended bit time (*CANn_FDCTRL.extbt* = 1) is selected.
17:10	nseg1	R/W	0	**Nominal Segment 1 Time in Arbitration** This field defines the time before the sample point in CAN FD frames when extended bit time (*CANn_FDCTRL.extbt* = 1) is selected.
9:0	nbrp	R/W	0	**Baud Rate Prescaler in Arbitration Phase** This field selects the time quantum in the arbitration phase in the range of 1 to 1024 clock periods according to the equation: $t_q = (CANn\_NBT.nbrp + 1) \times t_{clk}$ *Note: This field is only used in CAN FD frames when extended bit time (CANn_FDCTRL.extbt = 1) is selected.*

Table 15-46: Data Bit Time and Second Sample Point Position Register

Data Bit Time and Second Sample Point Position			CANn_DBT_SSPP	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31	-	RO	0	**Reserved**
30:24	sspp	R	0	**Secondary Sample Point Position** This field contains information about the secondary sample point. The secondary sample is defined as the sum of the measured transceiver loop delay and the value of the transceiver delay compensation (*CANn_TXDECMP.*). The transceiver loop delay is measured from the transmitted FDF to the r0 transition edge to the respective received one in every transmitted CAN FD frame.
23:20	dsjw	R/W	0	**Synchronization Jump Width in Data Phase** The bit time in the data phase must be less than or equal to the bit time in the arbitration phase. *Note: This field is only used when the CAN controller is in FD mode (CANn_FDCTRL.fden = 1).* *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*
19:16	dseg2	R/W	0	**DSEG2** This field is the time segment after the sample in the data phase. The bit time in the data phase must be less than or equal to the bit time in the arbitration phase. *Note: This field is only used when the CAN controller is in FD mode (CANn_FDCTRL.fden = 1).* *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*
15:10	dseg1	R/W	0	**DSEG1** This field is the time segment before the sample point in the data phase. The bit time in the data phase must be less than or equal to the bit time in the arbitration phase. *Note: This field is only used when the CAN controller is in FD mode (CANn_FDCTRL.fden = 1).* *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*

Data Bit Time and Second Sample Point Position			CANn_DBT_SSPP	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
9:0	dbrp	R/W	0	**Baud Rate Prescaler in Data Phase** This field selects the time quantum in the data phase in the range of 1 to 1024 clock periods according to the equation: $t_q = (CANn\_DBT\_SSPP.dbrp + 1) \times t_{clk}$ The bit time in the data phase must be less than or equal to the bit time in the arbitration phase. *Note: This field is only used when the CAN controller is in FD mode (CANn_FDCTRL.fden = 1).* *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*

Table 15-47: FD Control Register

FD Control			CANn_FDCTRL	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
7	-	RO	0	**Reserved**
6	ped	R/W	0	**Protocol Exception Disable** 0: A recessive state on the CAN bus at "res" position results in the CAN controller entering in bus integration state. 1: A recessive state on the CAN bus at "res" position results in a Form Error (*CANn_FDSTAT.frmerr* = 1). *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*
5	reom	R/W	0	**Restricted Operating Mode** This field selects restricted operating mode. When the CAN controller is in restricted operating mode, the node does not send error frames (active or passive) or overload frames. If an error or overload occurs, the CAN controller waits for bus idle before sending dominant bits. 0: Normal mode (error frames and overload frames are sent when they occur) 1: Restricted operating mode. *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*
4	dar	R/W	0	**Disable Auto Retransmission** Set this field to 1 to disable automatic retransmission. 0: Automatic retransmission enabled 1: Automatic retransmission disabled *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*
3	iso	R/W	0	**ISO CAN FD Format Selection** 0: Frame format according to Bosch CAN FD specification 1: Frame format according to ISO 11898-1:2015 *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*
2	extbt	R/W	0	**Bit Time Prescaler Select in Arbitration Phase** This field configures the bit time prescaler in the arbitration phase. 0: Use the bus timing registers (*CANn_BUSTIM0* and *CANn_BUSTIM1*) to configure the bit timing during the arbitration phase. 1: Use the nominal bit timing register (*CANn_NBT*) to configure the bit timing during the arbitration phase. *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*

FD Control			CANn_FDCTRL	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
1	brsen	R/W	0	**Alternative Bit Rate Select** This field selects flexible data rate during the data phase. When this field is set to 1, the data rate is set using the *CANn_DBT_SSPP* register. 0: Bit rate is not switched inside a CAN FD frame 1: Bit rate is switched from the nominal bit rate during the arbitration phase to the alternate bit rate during the data phase. *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*
0	fden	R/W	0	**CAN FD Enable** Set this field to 1 to enable CAN FD frames. 0: Classic CAN frame format 1: CAN FD frame format *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*

Table 15-48: FD Status Register

FD Status			CANn_FDSTAT	[0x0025]
Bits	**Field**	**Access**	**Reset**	**Description**
7:6	state	R	0	**CAN Operation State** This field indicates the CAN controller operating state. 0b00: Integrating – waiting for 11 recessive bit after reset or bus off. 0b01: Idle – waiting for start of frame 0b10: Receiver – node operating as receiver 0b11: Transmitter – node operating as transmitter
5	-	RO	0	**Reserved**
4	pee	R	0	**Protocol Exception Event Indicator** This field indicates that the CAN controller detects recessive state on "res" position and entry to bus integration state. This field is only updated when the CAN controller is in FD mode (*CANn_FDCTRL.fden* = 1).
3	stferr	R	0	**Stuff Error Indicator** This field indicates stuff error occurred in data phase in CAN FD frame with the BRS bit set. This field is only updated when the CAN controller is in FD mode (*CANn_FDCTRL.fden* = 1).
2	frmerr	R	0	**Form Error Indicator** This field indicates that a fixed form bit field contains at least one illegal bit in data phase of CAN FD frame with the BRS bit set. This field is only updated when the CAN controller is in FD mode (*CANn_FDCTRL.fden* = 1).
1	crcerr	R	0	**CRC Error Indicator** This field indicates that the calculated CRC is different from the received CRC in the CAN FD frame. This field is only updated when the CAN controller is in FD mode (*CANn_FDCTRL.fden* = 1).
0	biterr	R	0	**Bit Error Indicator** When this bit is set to 1, an inconsistency occurred between the transmitted bit and the received bit in the CAN FD frame. This field is only updated when the CAN controller is in FD mode (*CANn_FDCTRL.fden* = 1).

Table 15-49: Data Phase Error Register

Data Phase Error			CANn_DPERR	[0x0026]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	dperr	R	0	**Data Phase Error Counter** This field returns the error count, which is incremented each time that the transmitter and receiver detect an error during the data phase of the CAN FD frame. The counter stops counting at 255. Reset this field by placing the CAN controller into reset (*CANn_MODE.rst* = 1).

Table 15-50: Arbitration Phase Error Register

Arbitration Phase Error			CANn_APERR	[0x0027]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	aperr	R	0	**Arbitration Phase Error Counter** This field returns the error count, which is incremented each time that the transmitter and receiver detect an error during the arbitration phase of the frame. The counter stops counting at 255. Reset this field by placing the CAN controller into reset (*CANn_MODE.rst* = 1).

Table 15-51: Test Register

Test			CANn_TEST	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
7:2	-	RO	0	**Reserved**
1	txc	R/W	0	**Transmit Pin Disconnect** Set this field to 1 to leave the transmit pin disconnected while in test mode. When the transmit pin is disconnected, it remains in recessive state (transmit pin = 1). 0: Transmit pin is connected and any messages transmitted in loop back mode are visible on the transmit pin. 1: Transmit pin disconnected and held in the recessive state (transmit pin = 1). *Note: This field can only be modified when the CAN is in reset mode (CANn_MODE.rst = 1).*
0	lben	R/W	0	**Loop Back Mode Enable** Set this field to 1 to enable loop back mode. When the CAN is in loop back mode, the receive pin is disconnected from the CAN peripheral. 0: Disabled. 1: Enabled. *Note: This field can only be modified when the CAN is in reset mode (CANn_MODE.rst = 1).*

Table 15-52: Wake-up Clock Divisor Register

Wake-up Clock Divisor			CANn_WUPCLKDIV	[0x0029]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	wupdiv	R/W	0	**Wake-up Time Prescaler** This field programs the wake-up counter clock used for the wake-up filter time and wake-up expire time fields. The following equation determines the clock cycle time counter. $t = (CANn\_WUPCLKDIV + 1)/f_{CLK}$

Table 15-53: Wake-up Filter Time Register

Wake-up Filter Time			CANn_WUPFT	[0x002A]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	wupft	R/W	0	**Wake-up Filter Time** Set this field to the desired wake-up filter time. For the wake-up to work, this field must be greater than 0. When the wake-up counter is greater than or equal to this field, the measured state in the CAN bus is valid. This field can only be written when the CAN controller is in reset (*CANn_MODE.rst* = 1).

Table 15-54: Wake-up Expire Time Register

Wake-up Expire Time			CANn_WUPET	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:0	wupet	R/W	0	**Wake-up Expire Time** Set this field to the desired wake-up expire time. When the internal wake-up counter is less than this field, the detected wake-up pattern is valid. This field can only be written when the CAN controller is in reset (*CANn_MODE.rst* = 1).

Table 15-55: Receive FIFO Data Count Register

Receive FIFO Data Count			CANn_RXDCNT	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	rxdcnt	R	0	**Receive FIFO Count** This field returns the number of bytes (aligned to 4) of the data in the receive FIFO. This field is set to 0 after a hardware reset or when the CAN controller is in reset state (*CANn_MODE.rst* = 1). This field is decremented after every read from the receive FIFO (*CANn_RXFIFO32*, *CANn_RXFIFO16[0]*, *CANn_RXFIFO8[0]*).

Table 15-56: Transmit Buffer Space Count Register

Transmit Buffer Space Count			CANn_TXSCNT	[0x0032]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	txscnt	R	0x80	**Transmit FIFO Count** This field returns the number of bytes available in the transmit FIFO. When the transmit FIFO is empty (after reset or when a message is successfully sent), this field returns 0x80.

Table 15-57: Transmit Delay Compensation Register

Transmit Delay Compensation			CANn_TXDECMP	[0x0033]
Bits	**Field**	**Access**	**Reset**	**Description**
7	tdcen	R/W	0	**Transceiver Delay Compensation Enable** 0: Disabled 1: Enabled *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*
6:0	tdco	R/W	0	**Transceiver Delay Compensation Offset** This field sets the transceiver delay compensation offset that is added to the measured transceiver loop delay during frame transmission in CAN FD mode. *Note: This field can only be modified when the CAN controller is in reset (CANn_MODE.rst = 1).*

Table 15-58: Extended Interrupt Status Flag Register

Extended Interrupt Status Flag			CANn_EINTFL	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
7:2	-	RO	0	**Reserved**

Extended Interrupt Status Flag			CANn_EINTFL	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
1	rx_to	R/W1C	0	**Receive FIFO Timeout Interrupt** This field indicates if the interrupt condition occurred. Write 1 to clear this field. 0: Normal operation 1: Interrupt occurred
0	rx_thd	R/W1C	0	**Receive FIFO Threshold Interrupt** This field is set to 1 if the receive FIFO level reaches the set receive FIFO threshold (*CANn_MODE.rxtrig*). Write 1 to clear. 0: Normal operation 1: Interrupt occurred

Table 15-59: Extended Interrupt Enable Register

Extended Interrupt Enable			CANn_EINTEN	[0x0035]
Bits	**Field**	**Access**	**Reset**	**Description**
7:2	-	RO	0	**Reserved**
1	rx_to	R/W	0	**Receive FIFO Timeout Interrupt Enable** Set this field to 1 to enable the receive FIFO timeout interrupt. 0: Disabled 1: Enabled
0	rx_thd	R/W	0	**Receive FIFO Threshold Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Disabled 1: Enabled

Table 15-60: Receive FIFO Timeout Register

Receive FIFO Timeout			CANn_RXTO	[0x0036]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	rx_to	R/W	0	**Receive Timeout** This field specifies the receive timeout when at least 1 entry is in the receive FIFO. Set this field to the number of arbitration phase bit time sots to generate a timeout. When the receive timeout occurs, the *CANn_EINTFL.rx_to* field is set to 1 and an interrupt is generated if enabled (*CANn_EINTEN.rx_to* = 1). Note: This register is reset to 0 after a hardware reset and when the CAN controller is in the reset state (*CANn_MODE.rst* = 1).

16. USB 2.0 High-Speed Host Interface with PHY (USBHS)

The microcontroller includes one USB host communications peripheral with a USB physical interface (PHY). The USB host is USB 2.0 High-Speed 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 High-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.

- Crystal-free USB PHY eliminates need for external crystal.
  - Background real-time calibration during data transfers assures frequency accuracy.

Each endpoint has an associated FIFO with the following sizes:

- Endpoint 0 FIFO: 64 bytes deep

Endpoints 1 through 7 FIFOs: 512 bytes

- Endpoints 8 and 9 FIFOs: 2048 bytes

Endpoints 10 and 11 FIFOs: 4096 bytes

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 at *https://usb.org/document-library/usb-20-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.

16.1 Instances

There is one instance of the peripheral. *Table 16-1* shows the USBHS instance, the USB signals, and their package names for the MAX32690.

Note: The peripheral clock for the USBHS peripheral is disabled following a reset. Enable the USBHS peripheral clock by setting GCR_PCLKDIS0.usb to 0.

Table 16-1: MAX32690 USB Instance Table

Instance	USB Signal	Package Signal Name
USBHS	D+	DP
	D-	DM
	V_{BUS}	V_{DDB}[1]
	GND	V_{SSB}
1. V_{DDB} is used for bus connection detection. V_{DDB} cannot be connected directly to the V_{BUS} signal.		

16.2 USB HS Bus Signals

A USB cable connects a USB host and a USB device. The USB host controls the transfer and the device on a USB bus. 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+: Positive line of the differential data pair.

D-: Negative line of the differential data pair.

- V_{BUS}: Bus voltage supplied by Host.

GND: Ground

Note: See Table 16-1 for the mapping of the USB HS signals to the MAX32690 pin names. Refer to the device data sheet for the pin numbers of each of the USB HS pins.

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 software must write to the *USBHS_FADDR.addr* field.

Table 16-2 shows the USB bus states seen by the host indicated by the D+ and D- differential pair.

Table 16-2: USB Bus States Indicated by the D+ and D- Differential Pair

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 for full-speed devices. 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.

Bus State	Condition	D+	D-	Notes
Disconnect	Device wants to disconnect from host	Lo	Lo	SE0 ≥ 2.5µs
Reset	Host is initiating communication with a device	Lo	Lo	Held for ≥ 10ms

USB communication is based on the basic conditions shown in *Table 16-2*, which are used to generate the following states:

- **Data J State** – Differential "1"

Data K State – Differential "0"

- **Resume** – Data K state. Tells the device to exit Suspend state.

Start-of-Packet (SOP) – Bus switches from idle to K.

- **End-of-Packet (EOP)** – SE0 for approximately 2 bit times followed by a J for 1 bit time.

Keep-alive – EOP sent every 1 millisecond.

16.3 USB HS 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. The USBHS peripheral 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 set up 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. For example, interrupt endpoints are used by USB keyboards to transfer small data loads when keys are pressed.

- **Bulk Endpoints** – Used to send and receive high-volume data that does not require real-time processing. For example, bulk endpoints are used by USB flash drives to transfer high-volume data.

Isochronous Endpoints – Used to send or receive real-time data that requires a guaranteed bandwidth between a host and a device. A video camera performing real-time video streaming is an example of an isochronous endpoint use case.

The USBHS peripheral supports control, interrupt, bulk, and isochronous endpoints. 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 the USB device. IN endpoint 2 refers to a device endpoint holding data sent from the USB device to a USB host.

Each USB HS data endpoint supports the following features:

- Single or double buffered.

Programmable 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 handshake to the host for HS transfers to indicate it is not yet ready to receive more data.

Configurable response to status stage of control transfers.

16.4 Peripheral Clock Enable

The USBHS peripheral is disabled by default on a reset. Use of the USBHS peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS0.usb* to 0 to enable the USBHS peripheral clock.

16.5 Reset and Clock

When a Reset is detected on the bus, the USBHS peripheral 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 interrupt.

For correct operation of the USB HS interface, the system clock, f_{SYS_CLK}, must be no less than 32MHz.

16.6 Suspend and Resume States

When the USBHS peripheral sees no activity on the bus for 3ms, and suspend state is allowed (*USBHS_POWER.en_suspendm* = 1), then the USBHS peripheral goes into low-power suspend state. The suspend status flag is set (*USBHS_INTRUSB.suspend_int* = 1), and a suspend interrupt is generated, if enabled (*USBHS_INTRUSBEN.suspend_int_en* = 1).

Software can exit suspend state by sending a Resume state on the bus by setting the *USBHS_POWER.resume* bit to 1. Software must leave this bit set between 2ms and 15ms with 10ms being the optimal time after which the software must clear the resume bit (*USBHS_POWER.resume = 0*).

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 the software setting the *USBHS_POWER.resume* bit to 1.

16.7 Packet Size

The packet size is specified using the *USBHS_INMAXP* register for IN endpoints and the *USBHS_OUTMAXP* register for OUT endpoints for all transfers. These registers specify the packet size for the entire transaction.

16.8 Endpoint 0 Control Transactions

Endpoint 0 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 USB host is followed by additional data.
3. In read requests, the USB host is communicating with a USB device that is required to send data back to the host.

16.8.1 Endpoint 0 Error Handling

The USBHS peripheral can detect and generate interrupts for control transfer 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.data_end* 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 data end bit *USBHS_CSR0.data_end* 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 is 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.setup_end* 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. The software should then process the command in that packet.

16.9 Bulk Endpoints Operation and Options

16.9.1 Bulk IN Endpoints

A bulk IN endpoint is used to transfer high-volume data that does not require real-time processing. Five options are available for use with a bulk IN endpoint as shown in *Table 16-3*.

Table 16-3: 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.
Automatic Set	When the automatic set 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 can be transferred in a single USB operation. For these transfers, the USBHS supports split transactions, where large data packets that are written to bulk IN endpoints are split into multiple smaller packets. The necessary packet size information is set in the *USBHS_INCSRU* register.

Bulk IN Endpoint Option	Description
Error Handling	A packet with a STALL identifier is used to indicate that an endpoint has an error. To shut down the bulk IN endpoint transfer, set the *USBHS_INCSRL.sendstall* bit field to 1. 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.

16.9.2 Bulk OUT Endpoints

A bulk OUT endpoint is used to transfer non-periodic data from the host to the device. 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 are 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.

16.10 Interrupt Endpoints

16.10.1 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 is received from the Host.

16.10.2 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.

16.11 Isochronous Endpoints

16.11.1 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 16-4*.

Table 16-4: 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 has 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 is not 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.incomptx*, 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.

16.11.2 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 16-5*.

Table 16-5: 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 has 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.

Isochronous OUT Endpoint Option	Description
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.disnyet* is automatically set.

16.12 USBHS Device Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 16-6: USBHS Device Registers

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 (*USBHS_INDEX* = 0)
[0x0012]	*USBHS_INCSRL*	USBHS IN Endpoint Lower Control and Status Register (*USBHS_INDEX* != 0)
[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_OUTCOUNT*	USBHS Endpoint OUT FIFO Byte Count Register
[0x0018]	*USBHS_COUNT0*	USBHS Endpoint 0 IN 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 Info Register
[0x007A]	*USBHS_SOFTRESET*	USBHS Soft Reset Control Register
[0x0080]	*USBHS_CTUCH*	USBHS Hi-Speed Chirp Timeout Register

Offset	Register Name	Description
[0x0082]	*USBHS_CTHSRTN*	USBHS Hi-Speed RESUME Delay Register
[0x0400]	*USBHS_MXM_USB_REG_00*	USBHS Vendor Register
[0x0404]	*USBHS_M31_PHY_UTMI_RESET*	USBHS UTMI Reset Register
[0x0408]	*USBHS_M31_PHY_UTMI_VCONTROL*	USBHS UTMI VCONTROL Register
[0x040C]	*USBHS_M31_PHY_CLK_EN*	USBHS PHY Clock Enable Register
[0x0410]	*USBHS_M31_PHY_PONRST*	USBHS Power-On Reset Register
[0x0414]	*USBHS_M31_PHY_NONCRY_RSTB*	USBHS Hi-Speed V_{BUS} Reset Register
[0x0418]	*USBHS_M31_PHY_NONCRY_EN*	USBHS Non-Clock Recovery Enable Register
[0x0420]	*USBHS_M31_PHY_U2_COMPLIANCE_EN*	USBHS U2 Compliance Enable Register
[0x0424]	*USBHS_M31_PHY_U2_COMPLIANCE_DAC_ADJ*	USBHS U2 Compliance DAC Adjust Register
[0x0428]	*USBHS_M31_PHY_U2_COMPLIANCE_DAC_ADJ_EN*	USBHS U2 Compliance DAC Adjust Enable Register
[0x042C]	*USBHS_M31_PHY_CLK_RDY*	USBHS PHY Clock Ready Register
[0x0430]	*USBHS_M31_PHY_PLL_EN*	USBHS PLL Enable Register
[0x0434]	*USBHS_M31_PHY_BIST_OK*	USBHS PHY BIST Okay Register
[0x0438]	*USBHS_M31_PHY_DATA_OE*	USBHS PHY Data Output Enable Register
[0x043C]	*USBHS_M31_PHY_OSCOUTEN*	USBHS Oscillator Output Enable Register
[0x0440]	*USBHS_M31_PHY_LPM_ALIVE*	USBHS PHY Link Power Management Register
[0x0444]	*USBHS_M31_PHY_HS_BIST_MODE*	USBHS PHY HS BIST Mode Register
[0x0448]	*USBHS_M31_PHY_CORECLKIN*	USBHS Hi-Speed Core Clock Input Register
[0x044C]	*USBHS_M31_PHY_XTLSEL*	USBHS Hi-Speed Clock Source Frequency Select Register
[0x0450]	*USBHS_M31_PHY_LS_EN*	USBHS PHY Loopback Enable Register
[0x0454]	*USBHS_M31_PHY_DEBUG_SEL*	USBHS PHY Debug Select Register
[0x0458]	*USBHS_M31_PHY_DEBUG_OUT*	USBHS PHY Debug Output Register
[0x045C]	*USBHS_M31_PHY_OUTCLKSEL*	USBHS Hi-Speed Reference Clock Select Register
[0x0460]	*USBHS_M31_PHY_XCFGI_31_0*	USBHS Vendor Programming 0 Register
[0x0464]	*USBHS_M31_PHY_XCFGI_63_32*	USBHS Vendor Programming 1 Register
[0x0468]	*USBHS_M31_PHY_XCFGI_95_64*	USBHS Vendor Programming 2 Register
[0x046C]	*USBHS_M31_PHY_XCFGI_127_96*	USBHS Vendor Programming 3 Register
[0x0470]	*USBHS_M31_PHY_XCFGI_137_128*	USBHS Vendor Programming 4 Register
[0x0474]	*USBHS_M31_PHY_XCFGI_HS_COURSE_TUNE_NUM*	USBHS PHY Coarse Tune Decision Numbers in HS-Mode Register
[0x0478]	*USBHS_M31_PHY_XCFGI_HS_FINE_TUNE_NUM*	USBHS PHY Fine Tune Decision Numbers in HS-Mode Register
[0x047C]	*USBHS_M31_PHY_XCFGI_FS_COURSE_TUNE_NUM*	USBHS PHY Coarse Tune Decision Numbers in FS-Mode Register
[0x0480]	*USBHS_M31_PHY_XCFGI_FS_FINE_TUNE_NUM*	USBHS PHY Fine Tune Decision Numbers in FS-Mode Register
[0x0484]	*USBHS_M31_PHY_XCFGI_LOCK_RANGE_MAX*	USBHS PHY Maximum Range for Clock Tracking Register
[0x0488]	*USBHS_M31_PHY_XCFGI_LOCK_RANGE_MIN*	USBHS PHY Minimum Range for Clock Tracking Register
[0x048C]	*USBHS_M31_PHY_XCFGI_OB_RSEL*	USBHS PHY Oscillator RC Filter Resistor Select
[0x0490]	*USBHS_M31_PHY_XCFGI_OC_RSEL*	USBHS PHY Oscillator RC Filter Resistor Select Register
[0x0494]	*USBHS_M31_PHY_XCFGO*	USBHS PHY Vendor Monitor Register
[0x0498]	*USBHS_MXM_INT*	USBHS Hi-Speed V_{BUS} Interrupt Register
[0x049C]	*USBHS_MXM_INT_EN*	USBHS Hi-Speed V_{BUS} Interrupt Enable Register
[0x04A0]	*USBHS_MXM_SUSPEND*	USBHS Suspend Register
[0x04A4]	*USBHS_MXM_REG_A4*	USBHS Hi-Speed V_{BUS} State Register

16.12.1 USBHS Device Register Details

Table 16-7: USBHS Device Address Register

USBHS Device Address Register			**USBHS_FADDR**	**[0x0000]**
Bits	**Name**	**Access**	**Reset**	**Description**
7	update	R	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.

USBHS Device Address Register			USBHS_FADDR	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
6:0	addr	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 16-8: 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	R	0	**Read Hi-Speed Mode Status Flag** 0 = Full Speed mode 1 = Hi-Speed mode
3	reset	R	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	R	0	**Read SUSPEND Mode Status** 0 = Normal operation. 1 = USBHS is in SUSPEND mode. *Note: This field is automatically cleared when a SUSPEND mode interrupt occurs or if the USBHS_POWER.resume bit is set to 1.*
0	en_suspendm	R/W	0	**SUSPEND Mode Enable** 0: SUSPEND mode disabled. USB can not enter SUSPEND mode. 1: SUSPEND mode allowed. If no activity is detected on the bus for more than 3.0ms, the USB enters low-power SUSPEND mode.

Table 16-9: USBHS IN Endpoint Interrupt Flags Register

USBHS IN Endpoint Interrupt Flags			USBHS_INTRIN	[0x0002]
Bits	**Name**	**Access**	**Reset**	**Description**
15:12	-	RO	0	**Reserved**
11	ep11_in_int	R	0	**IN EP11 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
10	ep10_in_int	R	0	**IN EP10 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
9	ep9_in_int	R	0	**IN EP9 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.

USBHS IN Endpoint Interrupt Flags			USBHS_INTRIN	[0x0002]
Bits	**Name**	**Access**	**Reset**	**Description**
8	ep8_in_int	R	0	**IN EP8 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
7	ep7_in_int	R	0	**IN EP7 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
6	ep6_in_int	R	0	**IN EP6 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
5	ep5_in_int	R	0	**IN EP5 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
4	ep4_in_int	R	0	**IN EP4 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
3	ep3_in_int	R	0	**IN EP3 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
2	ep2_in_int	R	0	**IN EP2 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
1	ep1_in_int	R	0	**IN EP1 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
0	ep0_in_int	R	0	**EP0 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.

Table 16-10: USBHS OUT Endpoint Interrupt Flags Register

USBHS OUT Endpoint Interrupt Flags			USBHS_INTROUT	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
15:12	-	RO	0	**Reserved**
11	ep11_out_int	R	0	**OUT EP11 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
10	ep10_out_int	R	0	**OUT EP10 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
9	ep9_out_int	R	0	**OUT EP9 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
8	ep8_out_int	R	0	**OUT EP8 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
7	ep7_out_int	R	0	**OUT EP7 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
6	ep6_out_int	R	0	**OUT EP6 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
5	ep5_out_int	R	0	**OUT EP5 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.

USBHS OUT Endpoint Interrupt Flags			USBHS_INTROUT	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
4	ep4_out_int	R	0	**OUT EP4 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
3	ep3_out_int	R	0	**OUT EP3 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
2	ep2_out_int	R	0	**OUT EP2 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
1	ep1_out_int	R	0	**OUT EP1 Interrupt Status Flag** 0: Normal operation. 1: Interrupt occurred.
0	-	RO	0	**Reserved**

Table 16-11: USBHS IN Endpoint Interrupt Enable Register

USBHS IN Endpoint Interrupt Enable			USBHS_INTRINEN	[0x0006]
Bits	**Name**	**Access**	**Reset**	**Description**
15:12	-	RO	0	**Reserved**
11	ep11_in_int_en	R/W	0	**IN EP11 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
10	ep10_in_int_en	R/W	0	**IN EP10 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
9	ep9_in_int_en	R/W	0	**IN EP9 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
8	ep8_in_int_en	R/W	0	**IN EP8 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
7	ep7_in_int_en	R/W	0	**IN EP7 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
6	ep6_in_int_en	R/W	0	**IN EP6 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
5	ep5_in_int_en	R/W	0	**IN EP5 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
4	ep4_in_int_en	R/W	0	**IN EP4 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.

USBHS IN Endpoint Interrupt Enable			USBHS_INTRINEN	[0x0006]
Bits	**Name**	**Access**	**Reset**	**Description**
3	ep3_in_int_en	R/W	0	**IN EP3 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
2	ep2_in_int_en	R/W	0	**IN EP2 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
1	ep1_in_int_en	R/W	0	**IN EP1 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
0	ep0_int_en	R/W	0	**EP0 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.

Table 16-12: USBHS OUT Endpoint Interrupt Enable Register

USBHS OUT Endpoint Interrupt Enable			USBHS_INTROUTEN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
15:12	-	RO	0	**Reserved**
11	ep11_out_int_en	R/W	1	**OUT EP11 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
10	ep10_out_int_en	R/W	1	**OUT EP10 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
9	ep9_out_int_en	R/W	1	**OUT EP9 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
8	ep8_out_int_en	R/W	1	**OUT EP8 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
7	ep7_out_int_en	R/W	1	**OUT EP7 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
6	ep6_out_int_en	R/W	1	**OUT EP6 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
5	ep5_out_int_en	R/W	1	**OUT EP5 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.

USBHS OUT Endpoint Interrupt Enable			USBHS_INTROUTEN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
4	ep4_out_int_en	R/W	1	**OUT EP4 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
3	ep3_out_int_en	R/W	1	**OUT EP3 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
2	ep2_out_int_en	R/W	1	**OUT EP2 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
1	ep1_out_int_en	R/W	1	**OUT EP1 Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
0	-	RO	0	**Reserved**

Table 16-13: USBHS Signaling Interrupt Status Flag Register

USBHS Signaling Interrupt Status Flags			USBHS_INTRUSB	[0x000A]
Bits	**Name**	**Access**	**Reset**	**Description**
7:4	-	RO	0	**Reserved**
3	sof_int	R	0	**SOF Detected Status Flag**
2	reset_int	R	0	**RESET State Detected Status Flag**
1	resume_int	R	0	**RESUME State Detected Status Flag** Set when a RESUME state is detected while in SUSPEND mode.
0	suspend_int	R	0	**SUSPEND Mode Status Flag** Reads 1 when SUSPEND mode is active.

Table 16-14: USBHS Signaling Interrupt Enable Register

USBHS Signaling Interrupt Enable			USBHS_INTRUSBEN	[0x000B]
Bits	**Name**	**Access**	**Reset**	**Description**
7:4	-	RO	0	**Reserved**
3	sof_int_en	R/W	0	**SOF Detected Interrupt Enable** 0: Interrupt event disabled. 1: Interrupt event enabled.
2	reset_int_en	R/W	1	**RESET State Detected Interrupt Enable** 0: Interrupt event disabled. 1: Interrupt event enabled.
1	resume_int_en	R/W	1	**RESUME State Detected Interrupt Enable** 0: Interrupt event disabled. 1: Interrupt event enabled.
0	suspend_int_en	R/W	0	**SUSPEND Mode Interrupt Enable** 0: Interrupt event disabled. 1: Interrupt event enabled.

Table 16-15: USBHS Frame Number Register

USBHS Frame Number			USBHS_FRAME	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
15:11	-	RO	0	**Reserved**
10:0	framenum	R	0	**Read Last Received Frame Number** This is the 11-bit frame number received in the SOF packet.

Table 16-16: USBHS Register Index Select Register

USBHS Register Index Select			USBHS_INDEX	[0x000E]
Bits	**Name**	**Access**	**Reset**	**Description**
7:5	-	RO	0	**Reserved**
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: 0: Endpoint 0 IN/OUT status registers addressable. 1: Endpoint 1 IN/OUT status registers addressable. 2: Endpoint 2 IN/OUT status registers addressable. … 11: Endpoint 11 IN/OUT status registers addressable.

Table 16-17: USBHS Test Mode Register

USBHS Test Mode Register			USBHS_TESTMODE	[0x000F]
Bits	**Name**	**Access**	**Reset**	**Description**
7:6	-	RO	0	**Reserved**
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.
3	test_pkt	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 continues 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.

16.12.1.1 Endpoint Register Access Control

Each IN and OUT endpoint from endpoint 0x1 to 0xB uses memory mapped access to the registers in Table 16-18. Selecting a specific endpoint, using the USBHS_INDEX register, maps each of the registers in Table 16-18 to the selected endpoint.

Table 16-18: 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
USBHS_COUNT0

16.12.1.2 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 16-19: 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 USBHS_INMAXP.*maxpacketsize*. The total number of bytes transferred in the payload is: $N_{BYTES\_TRANSFERED} = maxpacketsize \times (numpackminus1 + 1)$ This must match the USBHS_INMAXP.*maxpacketsize* field 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 1 or 2. For bulk endpoints, the max multiplier is 32, so the maximum value for this register is 31 (0x1F). *Note: Only applicable for high-speed, 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 field is the maximum packet size in bytes that is transmitted for each micro-frame. 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.

16.12.1.3 USBHS Endpoint 0 Control Status Register

Table 16-20: USBHS Endpoint 0 Control Status Register

USBHS Endpoint 0 Control Status		USBHS_CSR0		[0x0012]
Bits	**Name**	**Access**	**Reset**	**Description**
7	serv_setup_end	R/W1C	0	**Clear EP0 Setup End Bit** Write a 1 to clear the *USBHS_CSR0.setup_end* bit. *Note: Automatically cleared after being set.*
6	serv_outpktrdy	R/W1C	0	**Clear EP0 Out Packet Ready Bit** Write a 1 to clear the *USBHS_CSR0.outpktrdy* bit. *Note: Automatically cleared after being set.*
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	setup_end	R	0	**Read Setup End Status** Automatically set when a Control Transaction ends before the *USBHS_CSR0.data_end* bit is set by software. An interrupt is generated when this bit is set. Write a 1 to *USBHS_CSR0.serv_setup_end* to clear.
3	data_end	R/W1O	0	**Control Transaction Data End** Write a 1 to this bit after firmware completes any of the following three transactions: 1) Set *USBHS_CSR0.inpktrdy* = 1 for the last data packet. 2) Set *USBHS_CSR0.inpktrdy* = 1 for a zero-length data packet. 3) Clear *USBHS_CSR0.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	R	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 *USBHS_CSR0.serv_outpktrdy* bit to clear after the packet is unloaded from the OUT FIFO.

16.12.1.4 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 16-21: USBHS IN Endpoint Lower Control and Status Register

USBHS IN Endpoint Lower Control and Status			USBHS_INCSRL	[0x0012]
Bits	**Name**	**Access**	**Reset**	**Description**
7	incomptx	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 are 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.*
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 USBHS_INCSRL.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 (*USBHS_CSR0.inpktrdy* = 0), an IN token was received, and a zero-length data packet was 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.*

16.12.1.5 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 16-22: 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	-	DNM	0	**Reserved, Do Not Modify**
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	-	DNM	0	**Reserved, Do Not Modify**
1	dpktbufdis	R/W	0	**Double Packet Buffering Disable** 0: Enable double packet buffering. Software must also configure the FIFO and packet size. 1: Disable double packet buffering
0	-	RO	0	**Reserved**

Table 16-23: 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 *USBHS_OUTMAXP.maxpacketsize*. The total number of bytes transferred in the payload is *USBHS_OUTMAXP.maxpacketsize* x (*numpackminus1* + 1). This must match the *maxpacketsize* field of the Standard Endpoint Descriptor for the associated endpoint. Only applicable for HS, high bandwidth isochronous endpoints, and bulk endpoints. This field is 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).
10:0	maxpacketsize	R/W	0x000	**Maximum Packet Size in a Single Transaction** This field is the maximum packet size, in bytes, that is transmitted for each micro-frame. 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 16-24: 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	R	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 (*USBHS_OUTCSRL.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	R	0	**FIFO Full Status** Set when the OUT FIFO is full. *Note: Automatically cleared when the FIFO is no longer full.*
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 16-25: 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.*

USBHS OUT Endpoint Upper Control Status Register			USBHS_OUTCSRU	[0x0017]
Bits	**Name**	**Access**	**Reset**	**Description**
6	iso	R/W	0	**Isochronous Transfer Enable** 0: Enable OUT Bulk and OUT Interrupt transfers. 1: Enable OUT Isochronous transfers.
5	-	DNM	0	**Reserved, Do Not Modify**
4	disnyet	R/W0C	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 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.*
3:2	-	DNM	0	**Reserved, Do Not Modify**
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.
0	incomprx	R	0	**Incomplete Isochronous Packet Received Error Status** High bandwidth Iisochronous 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: Endpoint 1 to 11 have a memory mapped version of this register selected using the USBHS_INDEX register.

Table 16-26: 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**
12:0	outcount	R	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 16-27: 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**
6:0	count0	R	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 16-28: 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*n*	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 16-29: USBHS Endpoint Count Info Register

USBHS Endpoint Count Info		**USBHS_EPINFO**	**[0x0078]**

Bits	Name	Access	Reset	Description
7:4	outendpoints	R	0xB	**Number of OUT Endpoints** There are 11 OUT endpoints in this USBHS peripheral. 0xB: 11 OUT endpoints.
3:0	intendpoints	R	0xB	**Number of IN Endpoints** Returns the number of IN endpoints in this USBHS peripheral. 0xB: 11 IN endpoints

Table 16-30: USBHS RAM Info Register

USBHS RAM Info		**USBHS_RAMINFO**	**[0x0079]**

Bits	Name	Access	Reset	Description
7:4	-	DNM	8	**Reserved, Do Not Modify**
3:0	rambits	R	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 16-31: USBHS Soft Reset Control Register

USBHS Soft Reset Control		**USBHS_SOFTRESET**	**[0x007A]**

Bits	Name	Access	Reset	Description
7:2	-	RO	0	**Reserved**

USBHS Soft Reset Control			USBHS_SOFTRESET	[0x007A]
Bits	**Name**	**Access**	**Reset**	**Description**
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 16-32: 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 16-33: USBHS Hi-Speed RESUME Delay Register

USBHS Hi-Speed RESUME Delay			USBHS_CTHSRTN	[0x0082]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	c_t_hstrn	R/W	0x19	**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. $t_{HI\_SPEED\_DELAY}(PHY\ clock\ cycles) = c\_t\_hstrn \times 4$ The delay value represents the number of PHY clock cycles from the end of the RESUME state to when normal USBHS operation begins.

Table 16-34: USBHS 00 Register

USBHS 00			USBHS_MXM_USB_REG_00	[0x0400]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 16-35: USBHS UTMI Reset Register

USBHS UTMI Reset			USBHS_M31_PHY_UTMI_RESET	[0x0404]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 16-36: USBHS UTMI VCONROL Register

USBHS UTMI VCONTROL Reset			USBHS_M31_PHY_UTMI_VCONTROL	[0x0408]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0x10	**Reserved**

Table 16-37: USBHS Clock Enable Register

USBHS Clock Enable Reset			USBHS_M31_PHY_CLK_EN	[0x040C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 16-38: 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 16-39: USBHS Hi-Speed V_{BUS} Reset Register

USBHS V_{BUS} Reset			USBHS_M31_PHY_NONCRY_RSTB	[0x0414]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	noncry_rstb	R/W	0	**USBHS V_{BUS} 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 V_{BUS} is on and 0 when V_{BUS} is off.

	noncry_rstb	**V_{BUS}**
Bus-Powered Device	Set to 1	Don't Care
Self-Powered Device	Set to 1	ON
	Set to 0	OFF

Table 16-40: 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 16-41: USBHS U2 Compliance Enable Register

USBHS U2 Compliance Enable			USBHS_M31_PHY_U2_COMPLIANCE_EN	[0x0420]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0x0	**Reserved**

Table 16-42: USBHS U2 Compliance DAC Adjust Register

USBHS U2 Compliance DAC Adjust			USBHS_M31_PHY_U2_COMPLIANCE_DAC_ADJ	[0x0424]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0x0	**Reserved**

Table 16-43: USBHS U2 Compliance DAC Adjust Enable Register

USBHS U2 Compliance DAC Adjust Enable			USBHS_M31_PHY_U2_COMPLIANCE_DAC_ADJ_EN	[0x0428]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0x0	**Reserved**

Table 16-44: USBHS Clock Ready Register

USBHS Clock Ready			USBHS_M31_PHY_CLK_RDY	[0x042C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0x0	**Reserved**

Table 16-45: USBHS PLL Enable Register

USBHS PLL Enable			USBHS_M31_PHY_PLL_EN	[0x0430]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
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 16-46: USBHS PHY BIST OK Register

USBHS PHY BIST OK			USBHS_M31_PHY_BIST_OK	[0x0434]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0x0	**Reserved**

Table 16-47: USBHS UTMI Data Output Enable Register

USBHS UTMI Data Output Enable			USBHS_M31_PHY_DATA_OE	[0x0438]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	1	**Reserved**

Table 16-48: USBHS Oscillator Output Enable Register

USBHS Oscillator Output Enable			USBHS_M31_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 16-49: USBHS Link Power Management (LPM) Alive Register

USBHS LPM Alive			USBHS_M31_PHY_LPM_ALIVE	[0x0440]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	1	**Reserved**

Table 16-50: USBHS BIST Mode Register

USBHS BIST Mode			USBHS_M31_PHY_HS_BIST_MODE	[0x0444
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	1	**Reserved**

Table 16-51: USBHS Hi-Speed Core Clock Input Register

USBHS Hi-Speed Core Clock Input			USBHS_M31_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 16-52: USBHS Hi-Speed Clock Source Frequency Select Register

USBHS Hi-Speed Clock Source Frequency Select			USBHS_M31_PHY_XTLSEL	[0x044C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2:0	xtlsel	R/W	2	**Clock Source Frequency Select** 0: 10Mhz. 1: 12MHz. 2: 25MHz. 3: 30MHz. 4: 19.2MHz. 5: 24MHz. 6: Reserved. 7: Reserved.

Table 16-53: USBHS Loopback Enable Register

USBHS Loopback Enable			USBHS_M31_PHY_LS_EN	[0x0450]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 16-54: USBHS Debug Select Register

USBHS Debug Select			USBHS_M31_PHY_DEBUG_SEL	[0x0454]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 16-55: USBHS Debug Out Register

USBHS Debug Out			USBHS_M31_PHY_DEBUG_OUT	[0x0458]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 16-56: USBHS Hi-Speed Reference Clock Select Register

USBHS Hi-Speed Reference Clock Select			USBHS_M31_PHY_OUTCLKSEL	[0x045C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	-	R/W	0	**Reference Clock Select** 0: Clock recovery from the data. 1: Invalid.

Table 16-57: USBHS Vendor Configuration 0 Register

USBHS Vendor Configuration 0			USBHS_M31_PHY_XCFGI_31_0	[0x0460]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 16-58: USBHS Vendor Configuration 1 Register

USBHS Vendor Configuration 1			USBHS_M31_PHY_XCFGI_63_32	[0x0464]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 16-59: USBHS Vendor Configuration 2 Register

USBHS Vendor Configuration 2			USBHS_M31_PHY_XCFGI_95_64	[0x0468]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 16-60: USBHS Vendor Configuration 3 Register

USBHS Vendor Configuration 3			USBHS_M31_PHY_XCFGI_127_96	[0x046C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 16-61: USBHS Vendor Configuration 4 Register

USBHS Vendor Configuration 4			USBHS_M31_PHY_XCFGI_137_128	[0x0470]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 16-62: USBHS HS Coarse Tune Decision Control Register

USBHS HS Coarse Tune Decision Control			USBHS_M31_PHY_XCFGI_HS_COURSE_TUNE_NUM	[0x0474]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2:0	-	RO	4	**Control Signal for the Decision of Coarse Tune Numbers in HS Mode** 0b000: Reserved 0b001: 1 time. 0b010: 2 times. 0b011: 3 times. 0b100: 4 times. 0b101: 5 times. 0b110: 6 times. 0b111: 7 times.

Table 16-63: USBHS HS Fine Tune Decision Control Register

USBHS HS Fine Tune Decision Control			USBHS_M31_PHY_XCFGI_HS_FINE_TUNE_NUM	[0x0478]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2:0	-	RO	4	**Control Signal for the Decision of Fine Tune Numbers in HS Mode** 0b000: Reserved 0b001: 1 time. 0b010: 2 times. 0b011: 3 times. 0b100: 4 times. 0b101: 5 times. 0b110: 6 times. 0b111: 7 times.

Table 16-64: USBHS FS Coarse Tune Decision Control Register

USBHS FS Coarse Tune Decision Control			USBHS_M31_PHY_XCFGI_FS_COURSE_TUNE_NUM	[0x047C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**

USBHS FS Coarse Tune Decision Control			USBHS_M31_PHY_XCFGI_FS_COURSE_TUNE_NUM	[0x047C]
Bits	**Name**	**Access**	**Reset**	**Description**
2:0	-	RO	2	**Control Signal for the Decision of Coarse Tune Numbers in FS Mode** 0b000: Reserved 0b001: 1 time. 0b010: 2 times. 0b011: 3 times. 0b100: 4 times. 0b101: 5 times. 0b110: 6 times. 0b111: 7 times.

Table 16-65: USBHS FS Fine Tune Decision Control Register

USBHS FS Fine Tune Decision Control			USBHS_M31_PHY_XCFGI_FS_FINE_TUNE_NUM	[0x0480]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2:0	-	RO	2	**Control Signal for the Decision of Fine Tune Numbers in FS Mode** 0b000: Reserved 0b001: 1 time. 0b010: 2 times. 0b011: 3 times. 0b100: 4 times. 0b101: 5 times. 0b110: 6 times. 0b111: 7 times.

Table 16-66: USBHS Lock Range Max Register

USBHS Lock Range Max			USBHS_M31_PHY_XCFGI_LOCK_RANGE_MAX	[0x0484]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3:0	-	RO	1	**Max Range for Clock Tracking**

Table 16-67: USBHS HS Lock Range Min Register

USBHS HS Lock Range Min			USBHS_M31_PHY_XCFGI_LOCK_RANGE_MIN	[0x0488]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3:0	-	RO	1	**Min Range for Clock Tracking**

Table 16-68: USBHS Oscillator RC Filer Resistor Select Register

USBHS Oscillator RC Filer Resistor Select			USBHS_M31_PHY_XCFGI_OB_RSEL	[0x048C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1:0	-	RO	1	**Oscillator RC Filter Resistor Select**

Table 16-69: USBHS Oscillator RC Filer Resistor Select Register

USBHS Oscillator RC Filer Resistor Select			USBHS_M31_PHY_XCFGI_OC_RSEL	[0x0490]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1:0	-	RO	2	**Oscillator RC Filter Resistor Select**

Table 16-70: USBHS Vendor Monitor Register

USBHS Vendor Monitor Select			USBHS_M31_PHY_XCFGO	[0x0494]
Bits	**Name**	**Access**	**Reset**	**Description**
31:27	-	RO	0	**Reserved**
26:0	-	RO	*	**Reserved**

Table 16-71: USBHS Hi-Speed V_{BUS} Interrupt Register

USBHS Hi-Speed V_{BUS} Interrupt			USBHS_MXM_INT	[0x0498]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2		RO	0	**Reserved**
1	vbus	R/W1C	*	**V_{BUS} Rising Edge Detect Flag** This flag generates an interrupt when set if *USBHS_MXM_INT_EN.vbus is set.* 0: Normal operation. 1: V_{BUS} rising edge detected.
0	novbus	R/W1C	*	**V_{BUS} Falling Edge Detect Flag** This flag generates an interrupt when set if *USBHS_MXM_INT_EN.novbus is set.* 0: Normal operation. 1: V_{BUS} falling edge detected.

Table 16-72: USBHS Hi-Speed V_{BUS} Interrupt Enable Register

USBHS Hi-Speed V_{BUS} Interrupt Enable			USBHS_MXM_INT_EN	[0x049C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2		RO	0	**Reserved**
1	vbus	R/W	0	**V_{BUS} Rising Edge Detect Enable** An interrupt is generated if *USBHS_MXM_INT.vbus* is set. 0: Disabled. 1: Enabled.
0	novbus	R/W	0	**V_{BUS} Falling Edge Detect Enable** An interrupt is generated if *USBHS_MXM_INT.novbus* is set. 0: Disabled. 1: Enabled.

Table 16-73: 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.
0	sel	R/W	1	**Suspend Control Select** The SUSPEND operation of the USBHS PHY can be controlled by the hardware or the software. This field selects the operating mode of the SUSPEND operation. 0: The USBHS controller hardware controls the PHY suspend operation. 1: The PHY SUSPEND operation is controlled by software using the *USBHS_MXM_SUSPEND.suspend* bit.

Table 16-74: USBHS Hi-Speed V_{BUS} State Register

USBHS Hi-Speed V_{BUS} State			USBHS_MXM_REG_A4	[0x04A4]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	vrst_vddb_n_a	R	0	**V_{BUS} State** Static state of V_{BUS} (V_{DDB}). 0: V_{DDB} is not powered. 1: V_{DDB} is powered.

17. Universal Asynchronous Receiver/Transmitter (UART)

The UART and the low-power universal asynchronous receiver/transmitter (LPUART) interfaces communicate with external devices using industry-standard serial communications protocols. The UARTs are full-duplex serial ports. Each UART instance is independently configurable unless using a shared external clock source.

The LPUART is a special version of the peripheral that can receive characters at up to 9600 baud while in low-power modes. The hardware loads valid received characters into the receive FIFO and wakes the device when an enabled interrupt condition occurs.

The peripheral provides the following features:

- Flexible baud rate generation for standard UART instances
- 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 frame error detection
- Separate 8-byte transmit and receive FIFOs
- Flexible interrupt conditions
- Hardware flow control (HFC) using ready-to-send (RTS) and clear-to-send (CTS) pins
- Separate DMA channels for transmit and receive
  - DMA support is available in *ACTIVE* and *SLEEP*

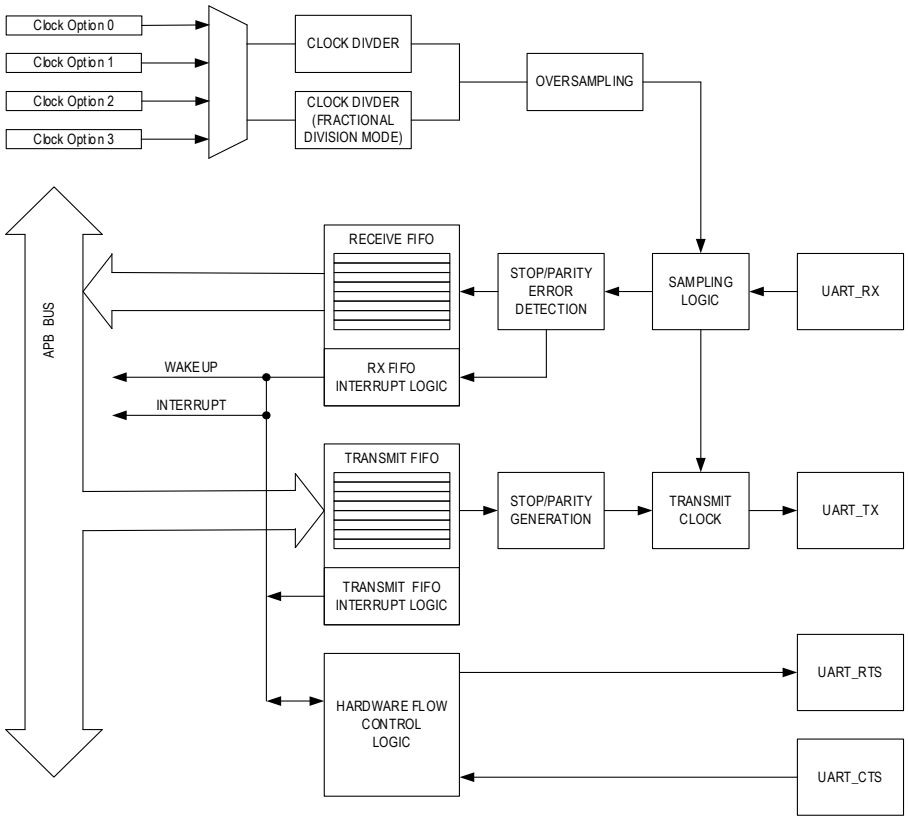
The LPUART instance provides these additional features:

- Receive characters in *SLEEP*, *LPM*, and *UPM* at up to 9600 baud

  Fractional baud rate divisor improves baud rate accuracy for 9600 and lower baud rates
- Wake up from low-power modes (*SLEEP*, *LPM*, and *UPM*) to *ACTIVE* on multiple receive FIFO conditions

Figure 17-1 shows a high-level diagram of the UART peripheral.

Figure 17-1: UART Block Diagram



Note: See Table 17-1 for the clock options supported by each UART instance.

17.1 Instances

Instances of the peripheral are shown in *Table 17-1*. The standard UARTs and the LPUARTs are functionally similar; they are referred to as UART for common functionality. The LPUART instance supports fractional division mode (FDM) and is referenced as LPUART for feature-specific options.

Table 17-1: MAX32690 UART/LPUART Instances

Instance	Register Access Name	LPUART	Power Modes	Clock Option				HFC	Transmit FIFO Depth	Receive FIFO Depth
				0	1	2	3			
UART0	UART0	No	*ACTIVE SLEEP*	PCLK	ERFO	IBRO	-	Yes	8	8
UART1	UART1									
UART2	UART2									
LPUART0	UART3	Yes	*ACTIVE SLEEP LPM UPM*	IBRO	ERTCO	-	-	No		

17.1.1 Peripheral Clock Enable

The UART ports are disabled by default on a reset. Use of a UART port requires enabling the peripheral clock for the required port. Enable the peripheral clock to a given UART port by setting the disable bit to 0. See *Table 17-2* for each UART ports peripheral clock disable bit.

Table 17-2: UART Peripheral Clock Disable Bits

UART Port	Disable Bit
UART0	*GCR_PCLKDIS0.uart0*
UART1	*GCR_PCLKDIS0.uart1*
UART2	*GCR_PCLKDIS1.uart2*
UART3 (LPUART0)	*LPGCR_PCLKDIS.uart3*

17.2 DMA

Each UART instance supports DMA for both transmit and receive; separate DMA channels can be connected to the receive and transmit FIFOs.

The UART DMA channels are configured using the UART DMA configuration register, *UARTn_DMA*. Enable the receive FIFO DMA channel by setting *UARTn_DMA.rx_en* to 1 and enable the transmit FIFO DMA channel by setting *UARTn_DMA.tx_en* to 1. DMA transfers are automatically triggered by the hardware based on the number of bytes in the receive FIFO and transmit FIFO.

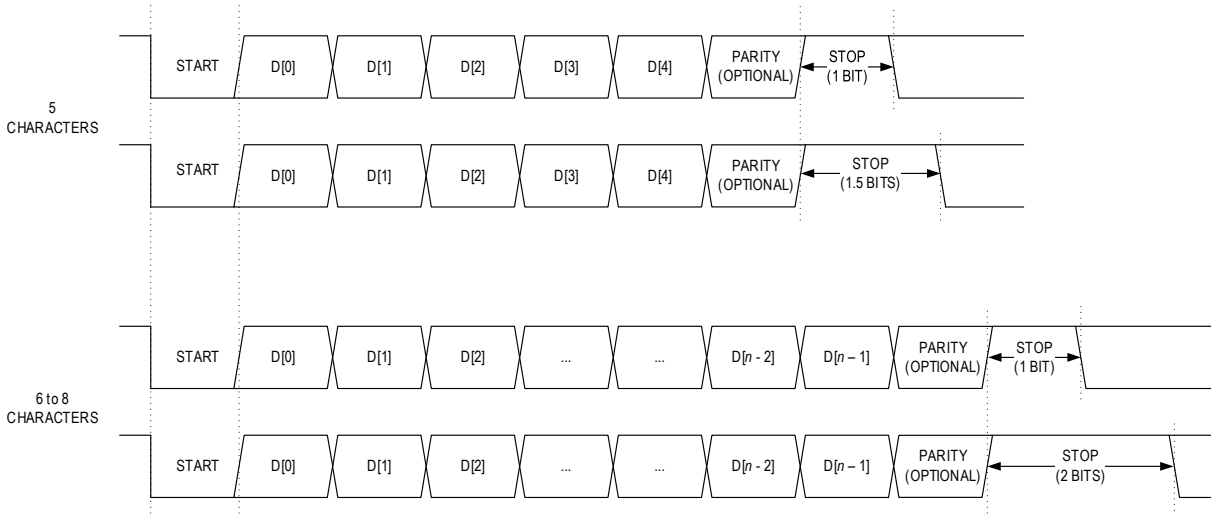
When DMA is enabled, the following describes the behavior of the DMA requests:

- A receive DMA request is asserted when the number of bytes in the receive FIFO transitions to be greater than or equal to the receive FIFO threshold.
- A transmit DMA request is asserted when the number of bytes in the transmit FIFO transitions to be less than the transmit FIFO threshold.

17.3 UART Frame

Figure 17-2 shows the UART frame structure. Character sizes of 5 to 8 bits are configurable through the *UARTn_CTRL.char_size* field. Stop bits are configurable as 1 or 1.5 bits for 5-character frames and 1 or 2 stop bits for 6, 7, or 8-character frames. Parity support includes even, odd, mark, space, and none.

Figure 17-2: UART Frame Structure



17.4 FIFOs

Separate receive and transmit FIFOs are provided. The FIFOs are both accessed through the same *UARTn_FIFO.data* field. The current level of the transmit FIFO is read from *UARTn_STATUS.tx_lvl*, and the receive FIFO current level is read from *UARTn_STATUS.rx_lvl*. Data for character sizes less than 7 bits are right justified.

17.4.1 Transmit FIFO Operation

Writing data to the *UARTn_FIFO.data* field increments the transmit FIFO pointer, *UARTn_STATUS.tx_lvl*, and loads the data into the transmit FIFO. The *UARTn_TXPEEK.data* register provides a feature that allows the software to "peek" at the current value of the write-only transmit FIFO without changing the *UARTn_STATUS.tx_lvl*. Writes to the transmit FIFO are ignored while *UARTn_STATUS.tx_lvl* = C_TX_FIFO_DEPTH.

17.4.2 Receive FIFO Operation

Reads of the *UARTn_FIFO.data* field return the character values in the receive FIFO and decrement the *UARTn_STATUS.rx_lvl*. An overrun event occurs if a valid frame, including parity, is detected while *UARTn_STATUS.rx_lvl* = C_RX_FIFO_DEPTH. When an overrun event occurs, the data is discarded by hardware.

A parity error event indicates that the value read from *UARTn_FIFO.data* contains a parity error.

17.4.3 Flushing

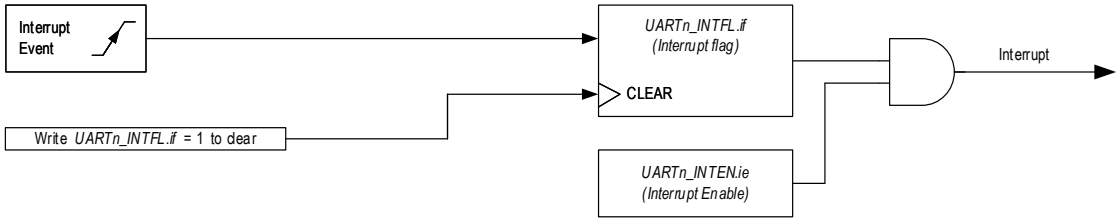
The FIFOs are flushed on the following conditions:

- Setting the *UARTn_CTRL.rx_flush* field to 1 flushes the receive FIFO by setting its pointer to 0.
- Setting the *UARTn_CTRL.tx_flush* field to 1 flushes the transmit FIFO by setting its pointer to 0.
- Flush the FIFOs by setting the respective UART's reset field (*GCR_RST0*) to 1.

17.5 Interrupt Events

The peripheral generates interrupts for the events shown in *Table 17-3*. Unless noted otherwise, each instance has its own set of interrupts and higher-level flag and enable fields, as shown in *Table 17-3*

Figure 17-3: UART Interrupt Functional Diagram



Some activity can set one or more event flags and cause more than one event. An event interrupt occurs if the corresponding interrupt enable is set. The interrupt flags, when set, must be cleared by the software by writing 1 to the corresponding interrupt flag field.

Table 17-3: MAX32690 Interrupt Events

Event	Interrupt Flag	Interrupt Enable
Frame Error	*UARTn_INT_FL.rx_ferr*	*UARTn_INT_EN.rx_ferr*
Parity Error	*UARTn_INT_FL.rx_par*	*UARTn_INT_EN.rx_par*
CTS Signal Change	*UARTn_INT_FL.cts_ev*	*UARTn_INT_EN.cts_ev*
Receive FIFO Overrun	*UARTn_INT_FL.rx_ov*	*UARTn_INT_EN.rx_ov*
Receive FIFO Threshold	*UARTn_INT_FL.rx_thd*	*UARTn_INT_EN.rx_thd*
Transmit FIFO Half-Empty	*UARTn_INT_FL.tx_he*	*UARTn_INT_EN.tx_he*
Transmit FIFO Almost Empty	*UARTn_INT_FL.tx_ae*	*UARTn_INT_EN.tx_ae*

17.5.1 Frame Error

A frame error is generated when the UART sampling circuitry detects an invalid bit. Each bit is sampled three times, as shown in *Figure 17-4*, and can generate a frame error on the start bit, stop bit, data bits, and optionally the parity bit. When a frame error occurs, the data is discarded.

The frame error criteria are different based on the following:

- Standard UART and LPUART with FDM disabled.
  - The start bit is sampled 3 times, and all samples must be 0, or a frame error is generated.
  - Each data bit is sampled, and 2 of the 3 samples must match, or a frame error is generated.
  - If parity is enabled, the parity bit is sampled 3 times, and all samples must match, or a frame error is generated.
  - The stop bit is sampled 3 times, and all samples must be 1, or a frame error is generated.
  - See *Table 17-4* for details
- LPUART with FDM enabled (*UARTn_CTRL.fdm* = 1) and data/parity edge detect enabled (*UARTn_CTRL.dpfe_en* = 1).
  - The start bit is sampled 3 times, and all samples must be 0, or a frame error is generated.
  - Each data bit is sampled 3 times, and all samples must match, or a frame error is generated.
  - If parity is enabled, the parity bit is sampled 3 times, and all samples must match, or a frame error is generated.
  - The stop bit is sampled 3 times, and all samples must be 1, or a frame error is generated.
  - See *Table 17-5* for details.

Table 17-4: Frame Error Detection for Standard UARTs and LPUART

UARTn_CTRL .par_en	UARTn_CTRL .par_md	UARTn_CTRL .par_eo	Start Samples	Data Samples	Parity Samples	Stop Samples
0	N/A	N/A	3 of 3 must be 0	2/3 must match	Not Present	3 of 3 must be 1
1	0	0			3/3 = 1 if even number "1" 3/3 = 0 if odd number "0"	
	0	1			3/3 = 1 if odd number "1" 3/3 = 0 if even number "0"	
	1	0			3/3 = 1 if even number "0" 3/3 = 0 if odd number "1"	
	1	1			3/3 = 1 if odd number "0" 3/3 = 0 if even number "1"	

Table 17-5: Frame Error Detection for LPUARTs with UARTn_CTRL.fdm = 1 and UARTn_CTRL.dpfe_en = 1

UARTn_CTRL .par_en	UARTn_CTRL .par_md	UARTn_CTRL .par_eo	Start Samples	Data Samples	Parity Samples	Stop Samples
0	N/A	N/A	3 of 3 must be 0	3 of 3 must match	Not Present	3 of 3 must be 1
1	0	0			3 of 3 = 1 if even number of 1s 3 of 3 = 0 if odd number 0s	
	0	1			3 of 3 = 1 if odd number 1s 3 of 3 = 0 if even number 0s	
	1	0			3 of 3 = 1 if even number 0s 3 of 3 = 0 if odd number 1s	
	1	1			3 of 3 = 1 if odd number 0s 3 of 3 = 0 if even number 1s	

17.5.2 Parity Error

Set UARTn_CTRL.*par_en* = 0 to enable parity checking of the received frame. If the calculated parity does not match the parity bit, then the corresponding interrupt flag is set. The data received is saved to the receive FIFO when a parity error occurs.

17.5.3 CTS Signal Change

A CTS signal change condition occurs if HFC is enabled, the UART baud clock is enabled, and the CTS pin changes state.

17.5.4 Overrun

An overrun condition occurs if a valid frame is received when the receive FIFO is full. The interrupt flag is set at the end of the stop bit, and the frame is discarded.

17.5.5 Receive FIFO Threshold

A receive FIFO threshold event occurs when a valid frame is received that causes the number of bytes to exceed the configured receive FIFO threshold UARTn_CTRL.*rx_thd_val*.

17.5.6 Transmit FIFO Half-Empty

The transmit FIFO half-empty event occurs when UARTn_STATUS.*tx_lvl* transitions from more than half-full to half-empty, as shown in Equation 17-1.

Note: When this condition occurs, verify the number of bytes in the transmit FIFO (UARTn_STATUS.tx_lvl) before refilling.

Equation 17-1: UART Transmit FIFO Half-Empty Condition

$$\left(\frac{C\_TX\_FIFO\_DEPTH}{2} + 1\right) \xrightarrow{Transistions\ from} \left(\frac{C\_TX\_FIFO\_DEPTH}{2}\right)$$

17.5.7 Transmit FIFO Almost Empty

A transmit FIFO almost empty event occurs when there is one byte remaining in the transmit FIFO.

17.6 LPUART Wake-up Events

LPUART instances can receive characters while in the low-power modes listed in *Table 17-1*. If enabled, each of the receive FIFO conditions shown in *Table 17-6* wakes the device, exits the low-power mode, and returns the device to *ACTIVE*.

Unlike interrupts, wake-up activity is based on a condition, not an event. As long as the condition is true and the wake-up enable field is set to 1, the wake-up flag remains set.

Table 17-6: MAX32690 Wake-up Events

Receive FIFO Condition	Wake-up Flag *UARTn_WKFL*	Wake-up Enable *UARTn_WKEN*	Low-Power Peripheral Wake-up Flag	Low-Power Peripheral Wake-up Enable	Low-Power Clock Disable
Threshold	*rx_thd*	*rx_thd*	*PWRSEQ_LPPWST.uart3*	*PWRSEQ_LPPWEN.uart3*	*LPGCR_PCLKDIS.uart3*
Full	*rx_full*	*rx_full*			
Not Empty	*rx_ne*	*rx_ne*			

17.6.1 Receive FIFO Threshold

This condition persists while *UARTn_STATUS.rx_lvl* ≥ *UARTn_CTRL.rx_thd_val*.

17.6.2 Receive FIFO Full

This condition persists while *UARTn_STATUS.rx_lvl* ≥ C_RX_FIFO_DEPTH.

17.6.3 Receive Not Empty

This condition persists while *UARTn_STATUS.rx_lvl* > 0.

17.7 Inactive State

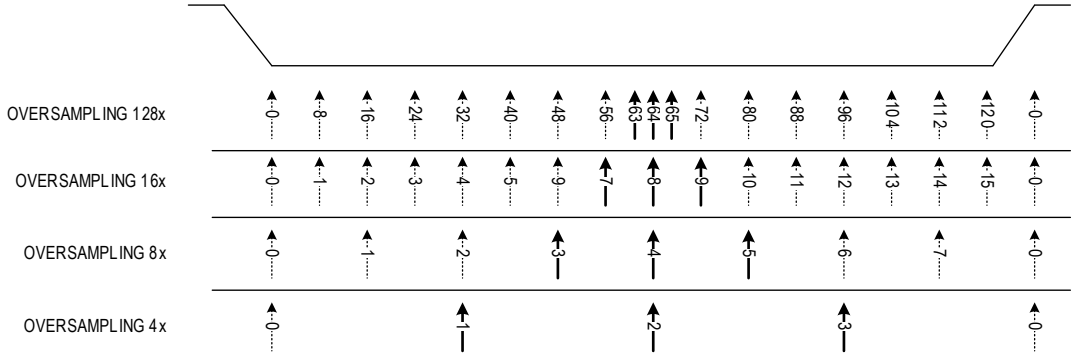
The following conditions result in the UART being inactive:

- When *UARTn_CTRL.bclken* = 0
- After setting *UARTn_CTRL.bclken* to 1 until *UARTn_CTRL.bclkrdy* = 1
- Any write to the *UARTn_CLKDIV.clkdiv field* while *UARTn_CTRL.bclken* = 1
- Any write to the *UARTn_OSR.osr* field when *UARTn_CTRL.bclken* = 1

17.8 Receive Sampling

Each bit of a frame is oversampled to improve noise immunity. The oversampling rate (OSR) is configurable with the *UARTn_OSR.osr* field. In most cases, the bit is evaluated based on three samples at the midpoint of each bit time, as shown in *Figure 17-4*.

Figure 17-4: Oversampling Example



Whenever *UARTn_CLKDIV.clkdiv* < 0x10 (i.e., division rate less than 8.0), OSR is not used, and the oversampling rate is adjusted to full sampling by the hardware. In full sampling, the receive input is sampled on every clock cycle regardless of the OSR setting.

Note: For 9600 baud low-power operation, the dual-edge sampling mode must be enabled (*UARTn_CTRL.desm* = 1).

17.9 Baud Rate Generation

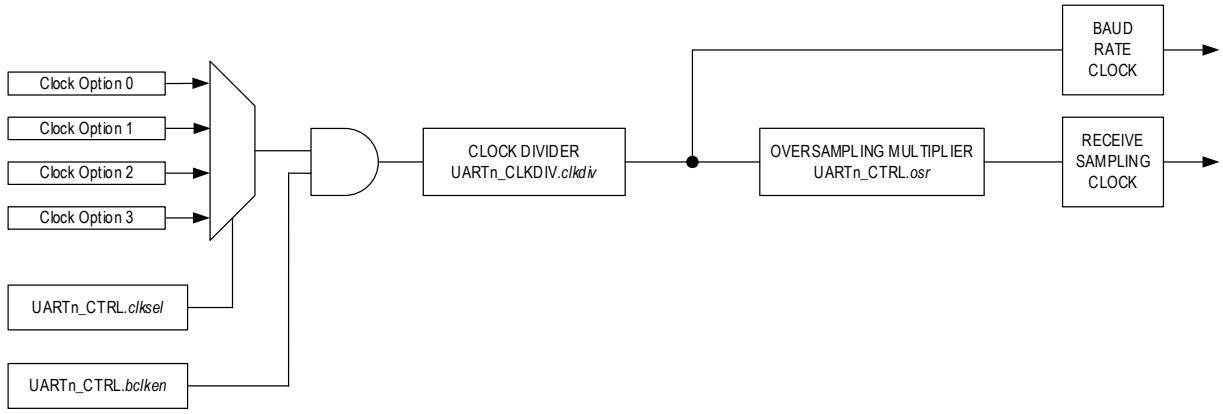
The baud rate is determined by the selected UART clock source and the value of the clock divisor. Multiple clock sources are available for each UART instance. See *Table 17-1* for available clock sources.

Note: Changing the clock source should only be done between data transfers to avoid corrupting an ongoing data transfer.

17.9.1 UART Clock Sources

Standard UART instances operate only in *ACTIVE* and *SLEEP*. Standard UART instances can only wake the device from *SLEEP*. *Figure 17-5* shows the baud rate generation path for standard UARTs.

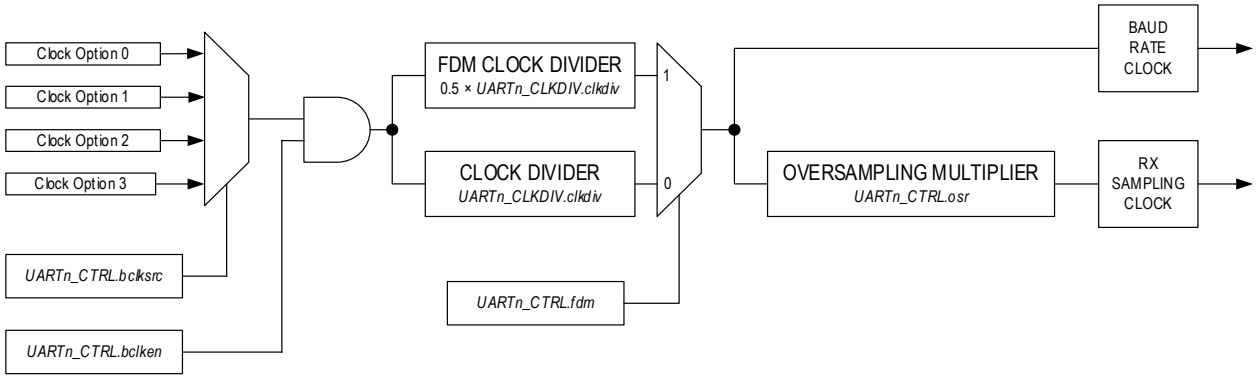
Figure 17-5: UART Baud Rate Generation



17.9.2 LPUART Clock Sources

LPUART instances support FDM and are configurable for operation at 9600 and lower baud rates for operation in *SLEEP*, *LPM*, and *UPM*. Operation in *LPM* and *UPM* require the use of the *ERTCO* as the baud rate clock source. The *ERTCO* can be configured to remain active in *LPM* and *UPM*, allowing the LPUART to receive data and serve as a wake-up source while power consumption is minimized.

Figure 17-6: LPUART Timing Generation



17.9.3 Baud Rate Calculation

The standard UART transmit and receive circuits share a common baud rate clock, which is the selected UART clock source divided by the clock divisor. Similarly, the low-power UARTs support a 0.5 fractional clock divisor when *UARTn_CTRL.fdm* is set to 1. Equation 17-2 should be used for calculating baud rates for *ACTIVE* and *SLEEP*. *Equation 17-3* is used for LPUARTs operating in *UPM* and *LPM* and requires setting *UARTn_CTRL.fdm* to 1. This allows for greater accuracy when operating at very low baud rates and finer granularity for the oversampling rate.

Equation 17-2: UART Clock Divisor Formula (UARTn_CTRL.fdm = 0)

$$UARTn\_CLKDIV.clkdiv = INT\left[\frac{f_{UART\_CLK}}{Baud\ Rate}\right]$$

$$if\ f_{UART\_CLK}\%\ Baud\ Rate > \frac{Baud\ Rate}{2}\ or\ UARTn\_CLKDIV.clkdiv = 0, then\ UARTn\_CLKDIV.clkdiv + 1$$

Equation 17-3: Low-Power UART Baud Rate Equation With FDM Enabled (UARTn_CTRL.fdm = 1)

$$UARTn\_CLKDIV.clkdiv = INT\left[\frac{f_{UART\_CLK}}{Baud\ Rate} \times 2\right]$$

$$if\ f_{UART\_CLK}\%\ Baud\ Rate > \frac{Baud\ Rate}{2}\ or\ UARTn\_CLKDIV.clkdiv = 0, then\ UARTn\_CLKDIV.clkdiv + 1$$

For example, in a case where the UART clock is PCLK (60MHz), and the target baud rate is 115,200 bps, calculate the *UARTn_CLKDIV.clkdiv* as follows:

$$UARTn\_CLKDIV.clkdiv = INT\left[\frac{60{,}000000}{115{,}200}\right] = 520$$

$$f_{UART\_CLK}\%115{,}200 = 96{,}000 > \left(\frac{115{,}200}{2}\right) \rightarrow UARTn\_CLKDIV.clkdiv = 521$$

For a low-power UART with IBRO selected as the clock source, the desired baud rate is 115,200bps, *UARTn_CTRL.fdm* = 0, calculate the *UARTn_CLKDIV.clkdiv* field as follows:

$$UARTn\_CLKDIV.clkdiv = INT\left[\frac{7{,}372{,}800}{115{,}200}\right] = 64$$

IMPORTANT: UARTn_CLKDIV.clkdiv must be greater than the OSR selected. In general, a *UARTn_OSR.osr* setting of 5 is sufficient for most applications.

17.10 Low-Power Mode Operation of LPUARTs for 9600 Baud and Below

The LPUARTs can be configured to receive up to 9600 baud in *SLEEP*, *LPM*, and *UPM*. If a valid frame is received, the receive FIFO is loaded and the receive FIFO level is incremented. If a wake-up event, shown in *Table 17-6*, is enabled, the device exits the current low-power mode and returns to *ACTIVE*. See *Baud Rate Calculation* and *Equation 17-3* for details on setting the baud rate for LPUART instances with *UARTn_CTRL.fdm* set to 1.

Table 17-7: LPUART Low Baud Rate Generation Examples (UARTn_CTRL.fdm = 1, f_{UART_CLK} = ERTCO)

Clock Source	BAUD (bits/s)	Calculated *UARTn_CLKDIV.clkdiv*	*UARTn_OSR.osr*
ERTCO	9,600	3	N/A (1×)
	7,200	5	N/A (1×)
	4,800	7	N/A (1×)
	2,400	14	0: 8× 1: 12×
	1,800	18	0: 8× 1: 12× 2: 16×
	1,200	27	0: 8× 1: 12× 2: 16× 3: 20× 4: 24×

17.10.1 Configuring an LPUART for Low-Power Modes of Operation

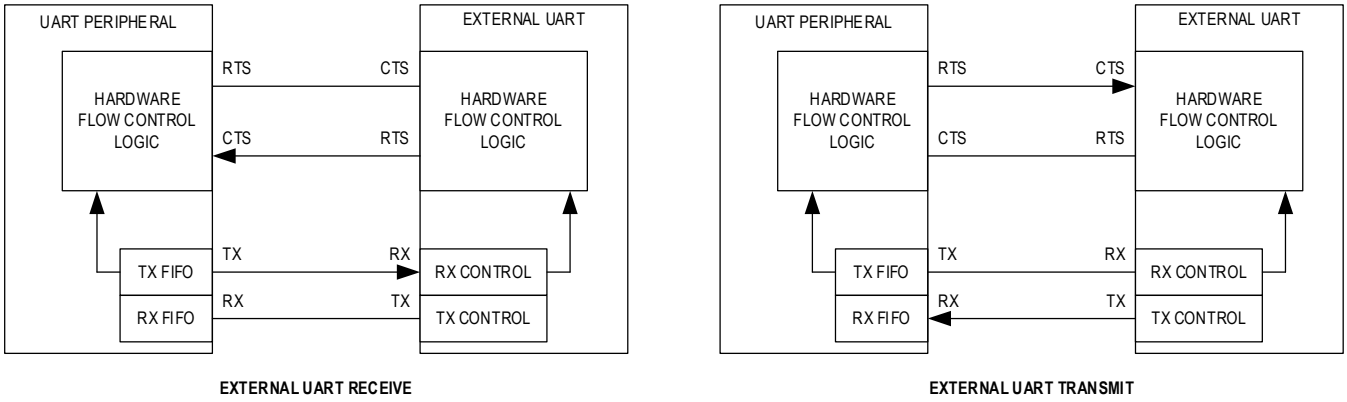
Use the following procedure to receive characters at 9600 or lower baud rates while in low-power modes:

1. Enable the LPUART for operation by setting *LPGCR_PCLKDIS.uart3* to 0.
2. Clear *UARTn_CTRL.bclken* = 0 to disable the baud clock. The hardware immediately clears *UARTn_CTRL.bclkrdy* to 0.
3. Set *MCR_CTRL.ertco_en* = 1 to ensure the 32kHz clock source remains active in *SLEEP*, *LPM,* and *UPM*.
4. Ensure *UARTn_CTRL.ucagm* = 1.
5. Configure *UARTn_CTRL.bclksrc* to select the *ERTCO*.
6. Set *UARTn_CTRL.fdm* to 1 to enable FDM.
7. Set *UARTn_CLKDIV.clkdiv* to the calculated clock divisor shown in *Table 17-7* for the required baud rate.
8. Set *UARTn_CTRL.desm* to 1 to enable dual-edge sampling receive mode.
9. Choose the desired wake-up conditions from *Table 17-6*.
    a. Clear any of the wake-up conditions chosen if currently active in the *UARTn_WKFL* register.
    b. Enable the wake-up condition; set the wake-up field to 1 in the *UARTn_WKEN* register.
10. Set the *UARTn_CTRL.bclken* field to 1 to enable the baud clock.
11. Poll the *UARTn_CTRL.bclkrdy* field until it reads 1.
12. Enter the desired low-power mode.

17.11 Hardware Flow Control

The optional HFC uses two additional pins, CTS and RTS, as a handshaking protocol to manage UART communications. For full-duplex operation, the RTS output pin on the peripheral is connected to the CTS input pin on the external UART, and the CTS input pin on the peripheral is connected to the RTS output pin on the external UART, as shown in *Figure 17-7*.

Figure 17-7: HFC Physical Connection



In HFC operation, a UART transmitter waits for the external device to assert its CTS pin. When CTS is asserted, the UART transmitter sends data to the external device. The external device keeps CTS asserted until it is unable to receive additional data, typically because the external device's receive FIFO is full. The external device then deasserts CTS until the device can receive more data. The external device then asserts CTS again, allowing additional data to be sent.

HFC can be fully automated by the peripheral hardware or by software through direct monitoring of the CTS input signal and control of the RTS output signal.

17.11.1 Automated HFC

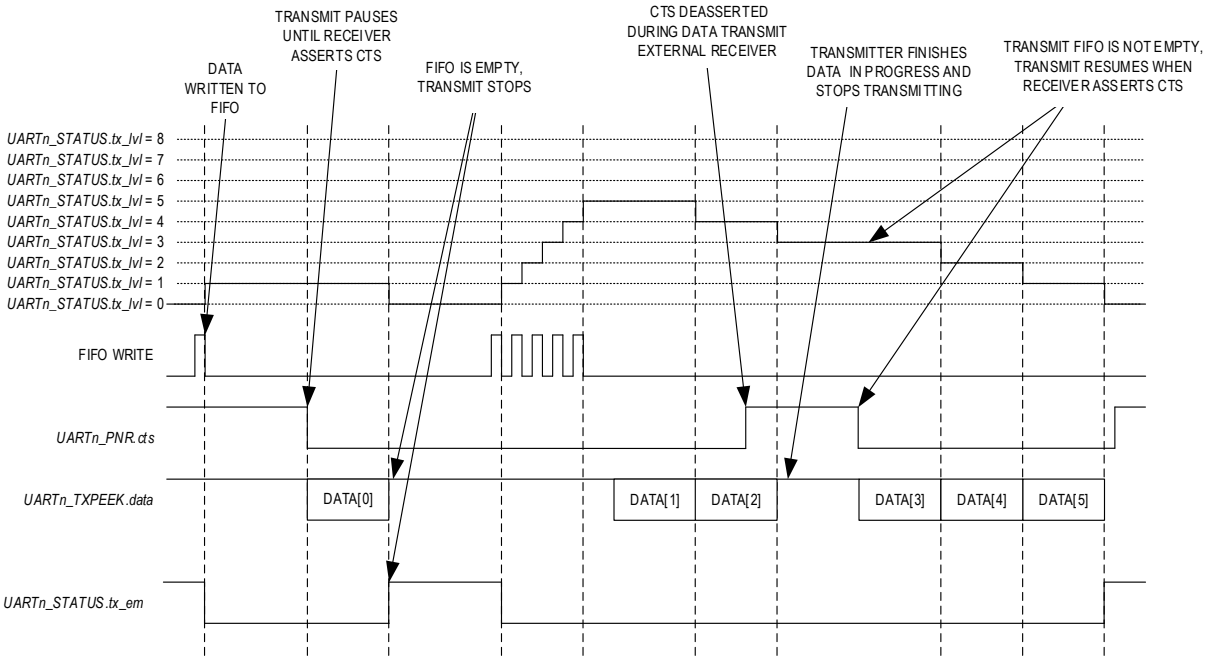
Setting *UARTn_CTRL*.*hfc_en* = 1 enables automated HFC. When automated HFC is enabled, the hardware manages the CTS and RTS signals. The deassertion of the RTS signal is configurable using the *UARTn_CTRL*.*rtsdc* field:

- *UARTn_CTRL*.*rtsdc* = 0: Deassert RTS when *UARTn_STATUS*.*rx_lvl* = C_RX_FIFO_DEPTH
- *UARTn_CTRL*.*rtsdc* = 1: Deassert RTS while *UARTn_STATUS*.*rx_lvl* ≥= *UARTn_CTRL*.*rx_thd_val*

The transmitter continues to send data as long as the CTS signal is asserted and there is data in the transmit FIFO. If the receiver deasserts the CTS pin, the transmitter finishes transmitting the current character and then waits until the CTS pin state is asserted before continuing transmission. *Figure 17-8* shows the state of the CTS pin during a transmission under automated HFC.

Automated HFC does not generate interrupt events related to the state of the transmit FIFO or the receive FIFO. The software must handle FIFO management. See *Interrupt Events* for additional information.

Figure 17-8: HFC Signaling for Transmitting to an External Receiver



17.11.2 Software-Controlled HFC

Software-controlled HFC requires the software to manually control the RTS output pin and monitor the CTS input pin. Using software-controlled HFC requires the automated HFC to be disabled by setting the *UARTn_CTRL.hfc_en* field to 1. Additionally, the software should enable CTS sampling (*UARTn_CTRL.cts_dis* = 0) if performing software-controlled HFC.

17.11.2.1 RTC/CTS Handling for Application-Controlled HFC

The software can manually monitor the CTS pin state by reading the field *UARTn_PNR.cts*. The software can manually set the state of the RTS output pin and read the current state of the RTS output pin using the field *UARTn_PNR.rts*. The software must manage the state of the RTS pin when performing software controlled HFC.

Interrupt support for CTS input signal change events is supported even when automated HFC is disabled. Software can enable the CTS interrupt event by setting the *UARTn_INT_EN.cts_ev* field to 1. The CTS signal change interrupt flag is set by the hardware any time the CTS pin state changes. The software must clear this interrupt flag manually by writing 1 to the *UARTn_INT_FL.cts_ev* field.

Note: CTS pin state monitoring is disabled any time the UART baud clock is disabled (UARTn_CTRL.bclken = 0). The software must enable CTS pin monitoring by setting the field UARTn_CTRL.cts_dis to 0 after enabling the baud clock if CTS pin state monitoring is required.

17.12 UART Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 17-8*. 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 1-1* 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.

All registers and fields apply to both UART and LPUART instances unless specified otherwise.

Table 17-8: UART/LPUART Register Summary

Offset	Register	Name
[0x0000]	*UARTn_CTRL*	UART Control Register
[0x0004]	*UARTn_STATUS*	UART Status Register
[0x0008]	*UARTn_INT_EN*	UART Interrupt Enable Register
[0x000C]	*UARTn_INT_FL*	UART Interrupt Flag Register
[0x0010]	*UARTn_CLKDIV*	UART Clock Divisor Register
[0x0014]	*UARTn_OSR*	UART Oversampling Control Register
[0x0018]	*UARTn_TXPEEK*	UART Transmit FIFO
[0x001C]	*UARTn_PNR*	UART Pin Control Register
[0x0020]	*UARTn_FIFO*	UART FIFO Data Register
[0x0030]	*UARTn_DMA*	UART DMA Control Register
[0x0034]	*UARTn_WKEN*	UART Wake-up Interrupt Enable Register
[0x0038]	*UARTn_WKFL*	UART Wake-up Interrupt Flag Register

17.12.1 Register Details

Table 17-9: UART Control Register

UART Control				UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:23	-	DNM	0	**Reserved**	
22	desm	R/W	0	**Receive Dual Edge Sampling Mode** LPUART instances only. This field is reserved in standard UART instances. 0: Sample receive input signal on clock rising edge only. 1: Sample receive input signal on both rising and falling edges.	
21	fdm	R/W	0	**Fractional Division Mode** LPUART instances only. This field is reserved in standard UART instances. 0: Baud rate divisor is an integer. 1: Baud rate divisor supports 0.5 division resolution.	
20	ucagm	R/W	0	**UART Clock Auto Gating Mode** *Note: Software must set this field to 1 for proper operation.* 0: No gating. 1: UART clock is paused during transmit and receive idle states.	
19	bclkrdy	R	0	**Baud Clock Ready** 0: Baud clock not ready. 1: Baud clock ready.	
18	dpfe_en	R/W	0	**Data/Parity Bit Frame Error Detection Enable** LPUART instances only. This field is reserved in standard UART instances. 0: Disable. Do not detect receive frame errors between the start bit and stop bit. 1: Enable. Detect frame errors when receive changes at the center of a bit time.	
17:16	bclksrc	R/W	0	**Baud Clock Source** This field selects the baud clock source. See *Table 17-1* for available clock options for each UART instance. 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.	

UART Control				UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
15	bclken	R/W	0	**Baud Clock Enable** 0: Disabled. 1: Enabled.	
14	rtsdc	R	0	**HFC RTS Deassert Condition** 0: Deassert RTS when the receive FIFO Level = C_RX_FIFO_DEPTH (FIFO full). 1: Deassert RTS while the receive FIFO Level >= *UARTn_CTRL.rx_thd_val*.	
13	hfc_en	R/W	0	**HFC Enable** 0: Disabled. 1: Enabled.	
12	stopbits	R/W	0	**Number of Stop Bits** 0: 1 stop bit. 1: 1.5 stop bits for 5-bit mode or 2 stop bits for 6/7/8-bit mode.	
11:10	char_size	R/W	0	**Character Length** 0: 5 bits. 1: 6 bits. 2: 7 bits. 3: 8 bits.	
9	rx_flush	W1	0	**Receive FIFO Flush** Write 1 to flush the receive FIFO. This bit always reads 0. 0: Normal operation. 1: Flush FIFO.	
8	tx_flush	W1	0	**Transmit FIFO Flush** Write 1 to flush the transmit FIFO. This bit always reads 0. 0: Normal operation. 1: Flush FIFO.	
7	cts_dis	R/W	1	**CTS Sampling Disable** 0: Enabled. 1: Disabled.	
6	par_md	R/W	0	**Parity Value Select** 0: Parity calculation is based on the number of 1 bits (mark). 1: Parity calculation is based on the number of 0 bits (space).	
5	par_eo	R/W	0	**Parity Odd/Even Select** 0: Even parity. 1: Odd parity.	
4	par_en	R/W	0	**Transmit Parity Generation Enable** 0: Parity transmission disabled. 1: Parity bit is calculated and transmitted after the last character bit.	
3:0	rx_thd_val	R/W	0	**Receive FIFO Threshold** Valid settings are from 1 to C_RX_FIFO_DEPTH. 0: Reserved. 1: 1 2: 2 3: 3 4: 4 5: 5 6: 6 7: 7 8: 8 9 - 15: Reserved.	

Table 17-10: UART Status Register

UART Status				UARTn_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:16	-	RO	0	**Reserved**	

UART Status				UARTn_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
15:12	tx_lvl	R	0	**Transmit FIFO Level** This field is the number of characters in the transmit FIFO. 0 - 8: Number of bytes in the transmit FIFO. 9 - 15: Reserved.	
11:8	rx_lvl	R	0	**Receive FIFO Level** This field is the number of characters in the receive FIFO. 0 - 8: Number of bytes in the receive FIFO. 9 - 15: Reserved.	
7	tx_full	R	0	**Transmit FIFO Full** 0: Not full. 1: Full.	
6	tx_em	R	1	**Transmit FIFO Empty** 0: Not empty. 1: Empty.	
5	rx_full	R	0	**Receive FIFO Full** 0: Not full. 1: Full.	
4	rx_em	R	1	**Receive FIFO Empty** 0: Not empty. 1: Empty.	
3:2	-	RO	0	**Reserved**	
1	rx_busy	R	0	**Receive Busy** 0: UART is not receiving a character. 1: UART is receiving a character.	
0	tx_busy	R	0	**Transmit Busy** 0: UART is not transmitting data. 1: UART is transmitting data.	

Table 17-11: UART Interrupt Enable Register

UART Interrupt Enable Register				UARTn_INT_EN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:7	-	RO	0	**Reserved**	
6	tx_he	R/W	0	**Transmit FIFO Half-Empty Event Interrupt Enable** 0: Disabled. 1: Enabled.	
5	tx_ae	R/W	0	**Transmit FIFO Almost Empty Interrupt Enable** 0: Disabled. 1: Enabled.	
4	rx_thd	R/W	0	**Receive FIFO Threshold Event Interrupt Enable** 0: Disabled. 1: Enabled.	
3	rx_ov	R/W	0	**Receive FIFO Overrun Event Interrupt Enable** 0: Disabled. 1: Enabled.	
2	cts_ev	R/W	0	**CTS Signal Change Event Interrupt Enable** 0: Disabled. 1: Enabled.	
1	rx_par	R/W	0	**Receive Parity Event Interrupt Enable** 0: Disabled. 1: Enabled.	
0	rx_ferr	R/W	0	**Receive Frame Error Event Interrupt Enable** 0: Disabled. 1: Enabled.	

Table 17-12: UART Interrupt Flag Register

UART Interrupt Flag			UARTn_INT_FL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6	tx_he	R/W1C	0	**Transmit FIFO Half-Empty Interrupt Flag**
5	tx_ae	R/W1C	0	**Transmit FIFO Almost Empty Flag**
4	rx_thd	R/W1C	0	**Receive FIFO Threshold Interrupt Flag**
3	rx_ov	R/W1C	0	**Receive FIFO Overrun Interrupt Flag**
2	cts_ev	R/W1C	0	**CTS Signal Change Interrupt Flag**
1	rx_par	R/W1C	0	**Receive Parity Error Interrupt Flag**
0	rx_ferr	R/W1C	0	**Receive Frame Error Interrupt Flag**

Table 17-13: UART Clock Divisor Register

UART Clock Divisor			UARTn_CLKDIV	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:0	clkdiv	R/W	0	**Baud Rate Divisor** This field sets the divisor used to generate the baud tick from the baud clock. For LPUART instances, if *UARTn_CTRL.fdm* = 1, the fractional divisors are in increments of 0.5. The over-sampling rate must be no greater than this divisor. See *Baud Rate Generation* for information on how to use this field.

Table 17-14: UART Oversampling Control Register

UART Oversampling Control			UARTn_OSR	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2:0	osr	R/W	0	**LPUART Over Sampling Rate** For LPUART instances with FDM enabled (*UARTn_CTRL.fdm* = 1): 0: 8 × 1: 12 × 2: 16 × 3: 20 × 4: 24 × 5: 28 × 6: 32 × 7: 36 × For LPUART instances with FDM disabled (*UARTn_CTRL.fdm* = 0): 0: 128 × 1: 64 × 2: 32 × 3: 16 × 4: 8 × 5: 4 × 6 - 7: Reserved

Table 17-15: UART Transmit FIFO Register

UART Transmit FIFO			UARTn_TXPEEK	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	data	RO	0	**Transmit FIFO Data** Read the transmit FIFO next data without affecting the contents of the transmit FIFO. If there are no entries in the transmit FIFO, this field reads 0. *Note: The parity bit is available from this field.*

Table 17-16: UART Pin Control Register

UART Pin Control			UARTn_PNR	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	rts	R/W	1	**RTS Pin Output State** 0: RTS signal is driven to 0. 1: RTS signal is driven to 1.
0	cts	RO	1	**CTS Pin State** This field returns the current sampled state of the GPIO associated with the CTS signal. 0: CTS state is 0. 1: CTS state is 1.

Table 17-17: UART Data Register

UART Data			UARTn_FIFO	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8	rx_par	R	0	**Receive FIFO Byte Parity** If the parity feature is disabled, this bit always reads 0. If a parity error occurred during the reception of the character at the output end of the receive FIFO (that would be returned by reading the *UARTn_FIFO.data* field), this bit reads 1, otherwise it reads 0.
7:0	data	R/W	0	**Transmit/Receive FIFO Data** Writing to this field loads the next character into the transmit FIFO if the transmit FIFO is not full. Reading from this field returns the next character from the receive FIFO if the receive FIFO is not empty. If the receive FIFO is empty, 0 is returned by the hardware. For character widths less than 8, the unused bit(s) are ignored when the transmit FIFO is loaded, and the unused high bit(s) read 0 on characters read from the receive FIFO.

Table 17-18: UART DMA Register

UART DMA			UARTn_DMA	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	rx_en	0	0	**Receive DMA Channel Enable** 0: Disabled. 1: Enabled.
8:5	rx_thd_val	0	0	**Receive FIFO Level DMA Threshold** If *UARTn_STATUS.rx_lvl* < *UARTn_DMA.rx_thd_val*, then the receive FIFO DMA interface sends a signal to the DMA indicating characters are available in the UART receive FIFO to transfer to memory.
4	tx_en	R/W	0	**Transmit DMA Channel Enable** 0: Disabled. 1: Enabled.
3:0	tx_thd_val	R/W	0	**Transmit FIFO Level DMA Threshold** If *UARTn_STATUS.tx_lvl* < *UARTn_DMA.tx_thd_val*, the transmit DMA channel sends a signal to the DMA indicating that the UART transmit FIFO is ready to receive data from memory.

Table 17-19: UART Wake-up Enable

UART Wake-up Enable			UARTn_WKEN	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**

UART Wake-up Enable			UARTn_WKEN	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
2	rx_thd	R/W	0	**Receive FIFO Threshold Wake-up Event Enable** 0: Disabled. 1: Enabled.
1	rx_full	R/W	0	**Receive FIFO Full Wake-up Event Enable** 0: Disabled. 1: Enabled.
0	rx_ne	R/W	0	**Receive FIFO Not Empty Wake-up Event Enable** 0: Disabled. 1: Enabled.

Table 17-20. UART Wake-up Flag Register

UART Wake-up Flag			UARTn_WKFL	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	rx_thd	R/W	0	**Receive FIFO Threshold Wake-up Event** 0: Disabled. 1: Enabled.
1	rx_full	R/W	0	**Receive FIFO Full Wake-up Event** 0: Disabled. 1: Enabled.
0	rx_ne	R/W	0	**Receive FIFO Not Empty Wake-up Event** 0: Disabled. 1: Enabled.

18. Serial Peripheral Interface (SPI)

The 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 or dual data lines, and one or more target select lines for communication with external SPI devices.

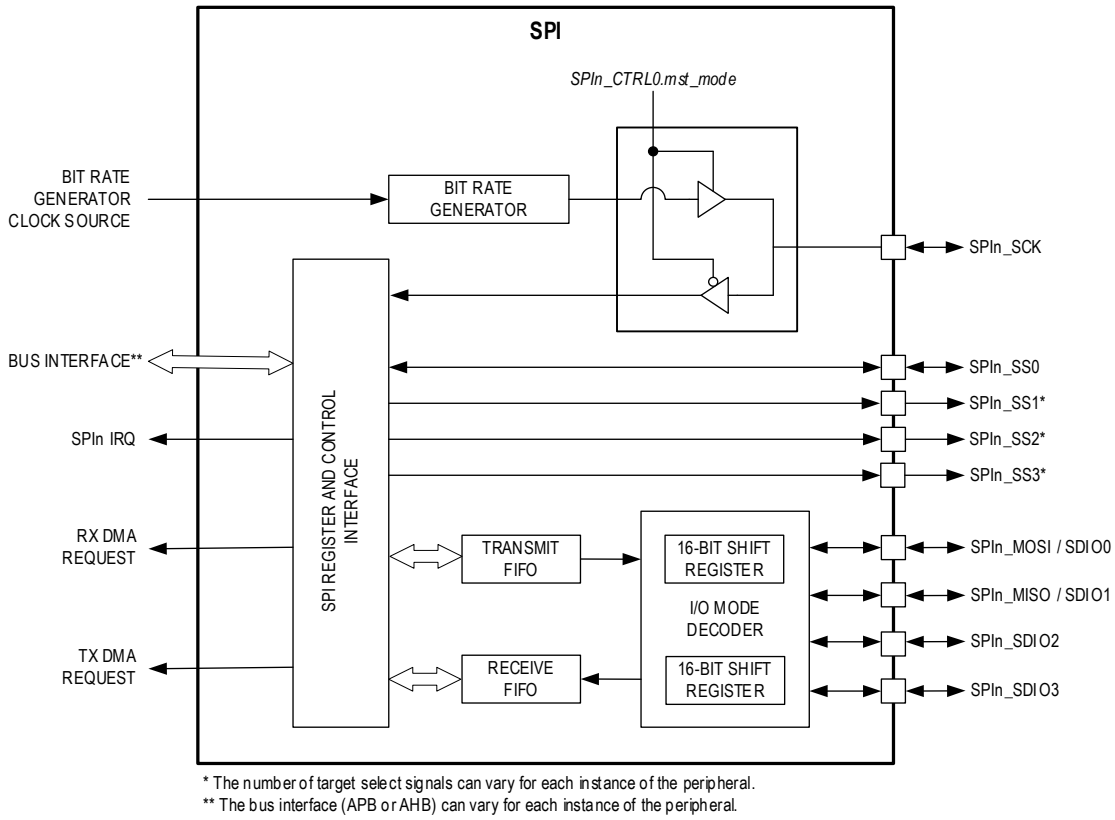
The provided SPI ports support full-duplex, bidirectional I/O, and each SPI includes a bit rate generator (BRG) for generating the clock signal when operating in controller mode. Each SPI port operates independently and requires minimal processor overhead. All instances of the SPI peripheral support both controller and target modes, and support single-controller and multi-controller networks.

Features include:

- Dedicated Bit Rate Generator for precision serial clock generation in controller mode:
  - Up to $\frac{f_{PCLK}}{2}$ for instances mapped on the APB bus.
  - Up to $\frac{f_{HCLK}}{2}$ for instances mapped on the AHB bus.
  - Programmable SCK duty cycle timing.
- Full-duplex, synchronous communication of 2- to 16-bit characters:
  - 1-bit and 9-bit characters are not supported.
  - 2-bit and 10-bit characters do not support maximum clock speed. *SPIn_CLKCTRL.clkdiv* must be > 0.
- Three-wire and four-wire SPI operation for single-bit communication.
- Single and dual 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 target select lines:
  - Programmable target select level.
- Programmable target select timing with respect to SCK starting edge and ending edge.
- Multicontroller mode fault detection.

Figure 18-1 shows the block diagram of the peripheral. See *Table 18-1* for the peripheral-specific bus assignment and bit rate generator clock source.

Figure 18-1: SPI Block Diagram



18.1 Instances

The following instances of the peripheral are provided.

Table 18-1: MAX32690 SPI Instances

Name	Formats				Bus Assignment	Bit Rate Generator Frequency
	Three-Wire	Four-Wire	Dual	Quad		
SPI0	Yes	Yes	Yes	Yes	APB	f_{PCLK}
SPI1	Yes	Yes	Yes	Yes	APB	f_{PCLK}
SPI2	Yes	Yes	Yes	Yes	APB	f_{PCLK}
SPI3	Yes	Yes	Yes	Yes	AHB	f_{PCLK}
SPI4	Yes	Yes	Yes	Yes	AHB	f_{PCLK}

Note: Refer to the device data sheet for SPI signal mapping.

18.2 SPI Formats

18.2.1 Four-Wire SPI

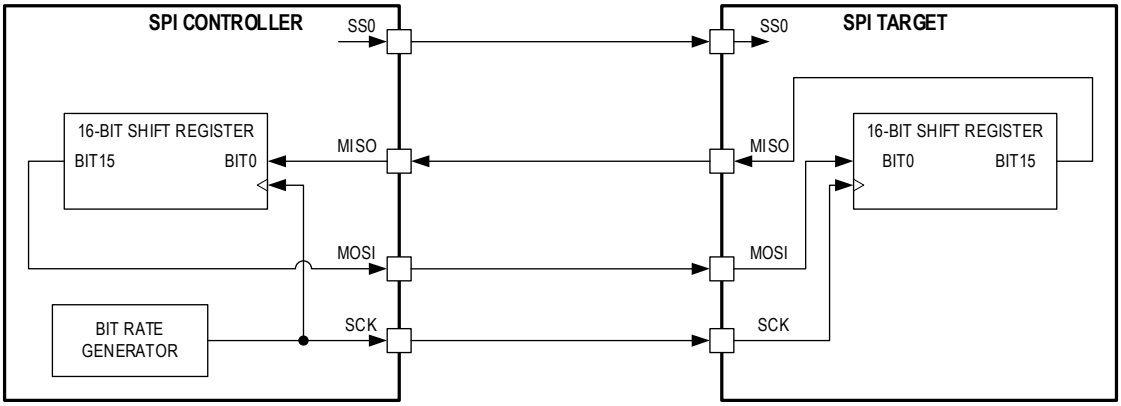
SPI devices operate as either a controller or a target device. In four-wire SPI, four signals are required for communication, as shown in *Table 18-2*.

Table 18-2: Four-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The controller generates the serial clock signal, an output from the controller, and an input to the target.
MOSI	Controller Output Target Input	In controller mode, this signal is used as an output for sending data to the target. In target mode, this is the input data from the controller.
MISO	Controller Input Target Output	In controller mode, this signal is used as an input for receiving data from the target. In target mode, this signal is an output for transmitting data to the controller.
SS	Target Select	In controller mode, this signal is an output used to select a target device before communication. Peripherals can have multiple target select outputs to communicate with one or more external target devices. In target mode, SPIn_SS0 is a dedicated input that indicates an external controller must start communication. Other target select signals into the peripheral are ignored in target mode.

In a typical SPI network, the controller device selects the target device using the target select output. The controller starts the communication by selecting the target device by asserting the target select output. The controller then starts the SPI clock through the SCK output pin. When a target device's target select pin is deasserted, the device must put the SPI pins in tri-state mode.

Figure 18-2: Four-Wire SPI Connection Diagram



18.2.2 Three-Wire SPI

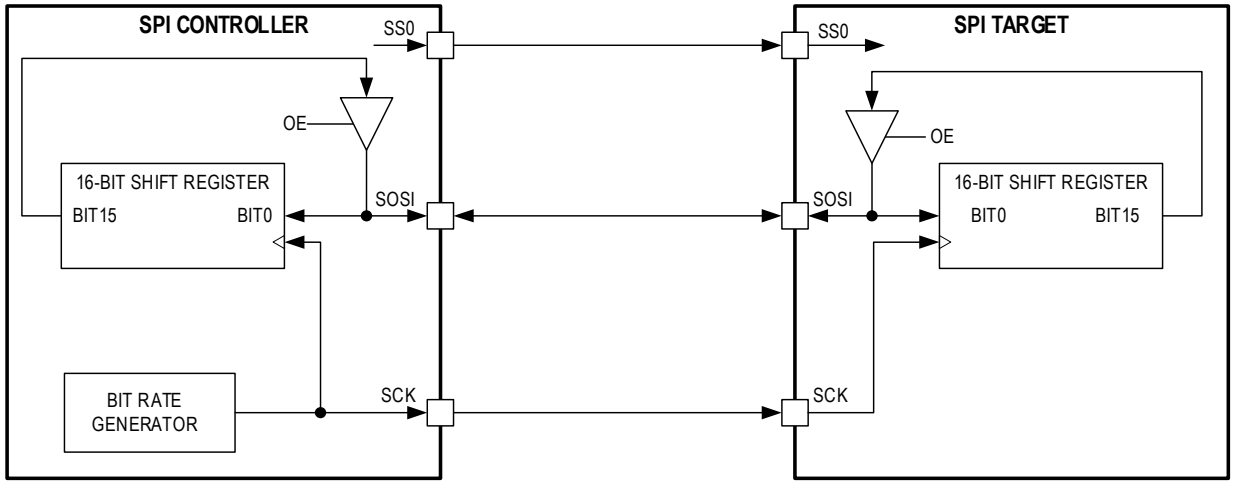
The signals in three-wire SPI operation are shown in *Table 18-3*. The MOSI signal is used as a bidirectional, half-duplex I/O referred to as target input target output (SISO). Three-wire SPI also uses a serial clock signal generated by the controller and a target select pin controlled by the controller.

Table 18-3: Three-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The controller generates the serial clock signal, an output from the controller, and an input to the target.
MOSI	Controller Output Target Input	This signal is a half-duplex, bidirectional I/O pin used for communication between the SPI controller and target. This signal is used to transmit data from the controller to the target and receive data from the target by the controller.
SS	Target Select	In controller mode, this signal is an output used to select a target device before communication. In target mode, SPIn_SS0 is a dedicated input that indicates an external controller must start communication. Other target select signals into the peripheral are ignored in target mode

A three-wire SPI network is shown in *Figure 18-3*. The controller device selects the target device using the target select output. The communication starts with the controller asserting the target select line and then starting the clock (SCK). In three-wire SPI communication, the controller and target must know the data's intended direction to prevent bus contention. For a write, the controller drives the data out of the SISO pin. The controller must release the SISO line for a read and let the target drive the SISO line. 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 18-3: Three-Wire SPI Controller to Target Connection



18.3 Pin Configuration

Before configuring the SPI peripheral, first, disable any SPI activity for the port by clearing the *SPIn_CTRL0.en* field to 0.

18.3.1 SPI Alternate Function Mapping

Pin selection and configuration are required to use the SPI port. The following information applies to SPI controller and target operation in three-wire, four-wire, and dual-mode communications. Determine the pins required for the SPI type and mode in the application, and configure the required GPIO as described in the following sections. Refer to the device data sheet for pin availability for a specific package.

When the SPI port is disabled, *SPIn_CTRL0.en* = 0, the GPIO pins enabled for SPI alternate function are placed in high-impedance input mode.

18.3.2 Four-Wire Format Configuration

Four-wire SPI uses SCK, MISO, MOSI, and one or more SS pins. Four-wire SPI can use more than one target select pin for a transaction, resulting in more than four wires total. However, the communication is referred to as four-wire for legacy reasons.

Note: 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. However, it is recommended that only one set of GPIO port pins are used for any network.

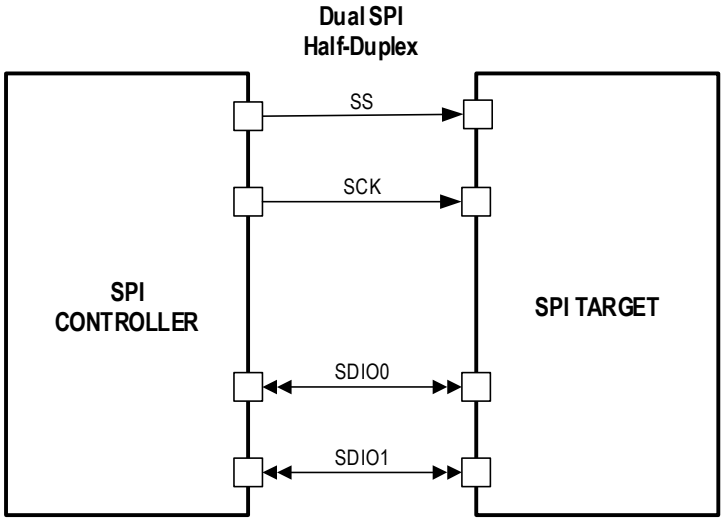
18.3.3 Three-Wire Format Configuration

Three-wire SPI uses SCK, MOSI, and one or more target select pins for an SPI transaction. Three-wire SPI configuration is identical to the four-wire configuration, except SPIn_MISO does not need to be set up for the SPI alternate function. The direction of communication in three-wire SPI mode is controlled by the SPI 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.

18.3.4 Dual Mode Format Configuration

In dual mode, SPI two I/O pins 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 controller and target for a given transaction. Dual mode SPI uses SCK, SDIO0, SDIO1, and one or more target select lines, as shown in *Figure 18-4*. The configuration of the GPIO pins for dual mode SPI is identical to four-wire SPI. The mode is controlled by setting *SPIn_CTRL2.data_width* to 1, indicating to the SPI hardware to use SDIO0 and SDIO1 for half-duplex communication rather than full-duplex communication.

Figure 18-4: Dual Mode SPI Connection Diagram

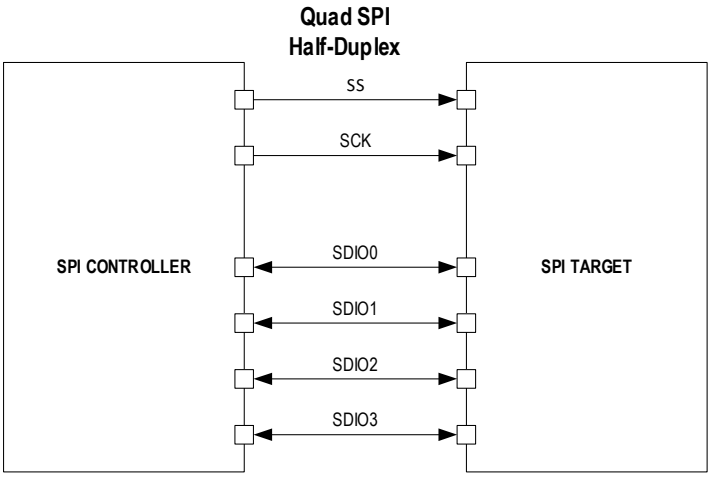


18.3.5 Quad-Mode Format Pin Configuration

Quad-mode SPI uses four I/O pins to transmit 4 bits of data per transaction. In quad-mode SPI, the communication is half-duplex, and the controller and target must know the direction of transmission for each transaction. Quad-mode SPI uses SCK, SDIO0, SDIO1, SDIO2, SDIO3, and one or more target select pins.

Quad-mode SPI transmits 4 bits per SCK cycle. Selection of quad-mode SPI is selected by setting *SPIn_CTRL2.data_width* to 2.

Figure 18-5: Quad Mode SPI Connection Diagram



18.4 SPI Clock Configuration

18.4.1 Peripheral Clock Enable

The SPI ports are disabled by default on a reset. Use of a SPI port requires enabling the peripheral clock for the required port. Enable the peripheral clock to a given SPI port by setting the disable bit to 0. See *Table 17-2* for each SPI ports peripheral clock disable bit.

Table 18-4: SPI Peripheral Clock Disable Bits

SPI Port	Disable Bit
SPI0	*GCR_PCLKDIS0.spi0*
SPI1	*GCR_PCLKDIS0.spi1*
SPI2	*GCR_PCLKDIS0.spi2*
SPI3	*GCR_PCLKDIS1.spi3*
SPI4	*GCR_PCLKDIS1.spi4*

18.4.2 Serial Clock

The SCK signal synchronizes data movement in and out of the device. The controller drives SCK as an output to the target's SCK pin. When SPI is set to controller mode, the SPI bit rate generator creates the serial clock and outputs it on the configured SPIn_SCK pin. When SPI is configured for target operation, the SPIn_SCK pin is input from the external controller, and the SPI hardware synchronizes communications using the SCK input. Operating as a target, if a SPI target select input is not asserted, the SPI ignores any signals on the serial clock and serial data lines.

In both controller and target 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.clkpha*. The SCK clock polarity field, *SPIn_CTRL2.clkpol*, controls if the SCK signal is active high or active low.

The SPI peripheral supports four combinations of SCK phase and polarity referred to as SPI modes 0, 1, 2, and 3. Clock polarity (*SPIn_CTRL2.clkpol*) selects an active low/high clock and does not affect the transfer format. Clock phase (*SPIn_CTRL2.clkpha*) selects one of two different transfer formats.

For proper data transmission, the clock phase and polarity must be identical for the SPI controller and target. The controller always places data on the MOSI line a half-cycle before the SCK edge for the target to latch the data. See section *Clock Phase and Polarity Control* for additional details.

18.4.3 SPI Peripheral Clock

The System Peripheral Clock, PCLK, drives the SPI peripheral clock. The SPI provides an internal clock, SPIn_CLK, used within the SPI peripheral for the base clock to control the module and generate the SCK clock in controller mode. Set the SPI internal clock using the field *SPIn_CLKCTRL.clkdiv* as shown in *Equation 18-1*. Valid settings for *SPIn_CLKCTRL.clkdiv* are 0 to 8, allowing a divisor of 1 to 256.

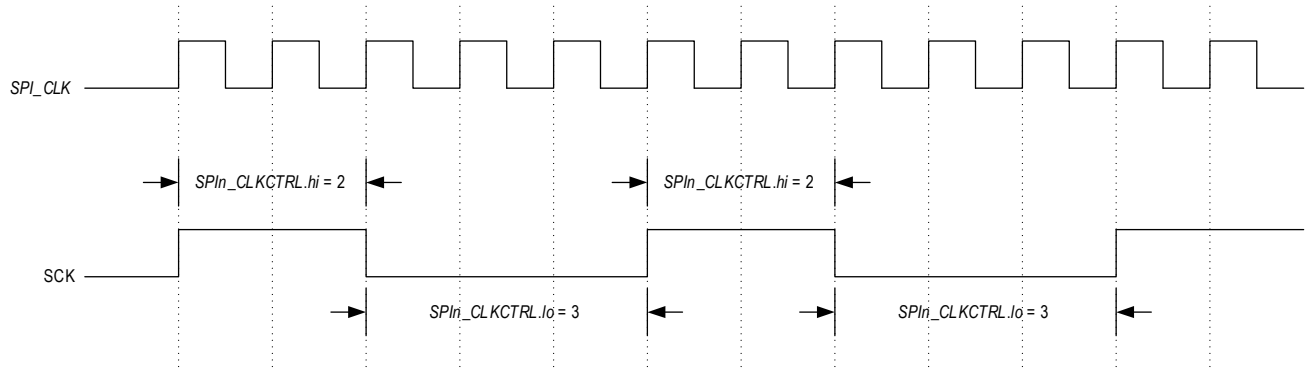
Equation 18-1: SPI Peripheral Clock

$$f_{SPI\_CLK} = \frac{f_{PCLK}}{2^{scale}}$$

18.4.4 Controller Mode Serial Clock Generation

In controller and multicontroller modes, the SCK clock is generated by the controller. The SPI provides 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 SPI peripheral clock as a base value, and the high and low values are a count of the number of f_{SPI_CLK} clocks. *Figure 18-6* visually represents the use of the *SPIn_CLKCTRL.hi* and *SPIn_CLKCTRL.lo* fields for a non-50% duty cycle serial clock generation. See *Equation 18-2* and *Equation 18-3* for calculating the SCK high and low time from the *SPIn_CLKCTRL.hi* and *SPIn_CLKCTRL.lo* field values.

Figure 18-6: SCK Clock Rate Control



Equation 18-2: SCK High Time

$$t_{SCK\_HI} = t_{SPInCLK} \times SPIn\_CLKCTRL.hi$$

Equation 18-3: SCK Low Time

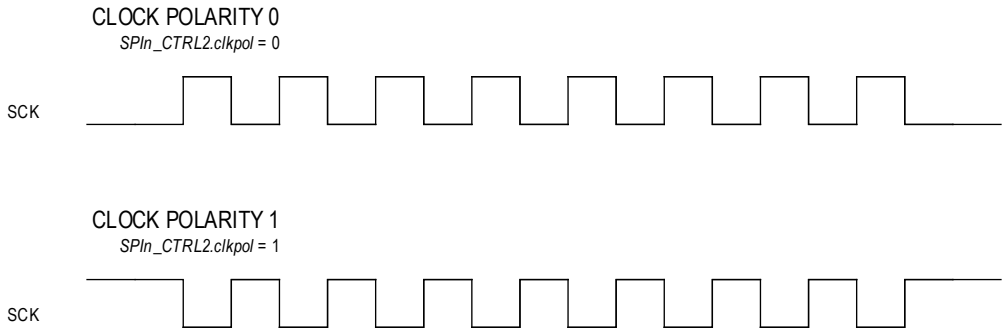
$$t_{SCK\_LOW} = t_{SPInCLK} \times SPIn\_CLKCTRL.lo$$

18.4.5 Clock Phase and Polarity Control

SPI supports four combinations of clock and phase polarity, as shown in *Table 18-5*. Clock polarity is controlled using the *SPIn_CTRL2.clkpol* bit, which determines if the clock is active high or active low, as shown in *Figure 18-7*. The clock's polarity does not affect the transfer format for SPI. The clock's phase determines when the data must be stable for sampling. Setting

the clock phase to 0, *SPIn_CTRL2.clkpha* = 0, dictates the SPI data is sampled on the initial SPI clock edge regardless of clock polarity. Phase 1, *SPIn_CTRL2.clkpha* = 1, results in the data sample occurring on the clock's second edge regardless of clock polarity.

Figure 18-7: SPI Clock Polarity



For proper data transmission, the clock phase and polarity must be identical for the SPI controller and target. The controller always places data on the MOSI line a half-cycle before the SCK edge for the target to latch the data.

Table 18-5: SPI Modes Clock Phase and Polarity Operation

SPI Mode	*SPIn_CTRL2 .clkpha*	*SPIn_CTRL2 .clkpol*	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

18.4.6 SPI 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 byte and the even byte as the least significant byte. The FIFO logic waits for the odd and even bytes to be written to this register 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, and a 32-bit read from this register dequeues four bytes from the FIFO, least significant byte first.

18.4.7 SPI Interrupts and Wake-ups

The SPI 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 software must clear the status flag 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
  - Target mode only, controller mode stalls the serial clock
- Transmit FIFO Overrun
- Receive FIFO Underrun
- Receive FIFO Overrun
  - Target mode only, controller mode stalls the serial clock

SPI supports interrupts for the internal state of the SPI as well as external signals. The following transmission interrupts are supported:

- SSn asserted or deasserted
- SPI transaction complete
  - Controller mode only
- Target mode transaction aborted
- Multicontroller fault

The SPI port can wake up the microcontroller from low-power modes when the wake event is enabled. SPI events that can wake the microcontroller are:

- Receive FIFO full
- Transmit FIFO empty
- Receive FIFO threshold
- Transmit FIFO threshold

18.5 SPI Registers

See *Table 3-2* for this peripheral/module's base address. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 18-6*. 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 1-1* 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 18-6: SPI Register Summary

Offset	Register Name	Access	Description
[0x0000]	*SPIn_FIFO32*	R/W	*SPI 32- Bit FIFO Data Register*
[0x0000]	*SPIn_FIFO16*	R/W	*SPI 16-Bit FIFO Data Register*
[0x0000]	*SPIn_FIFO8*	R/W	*SPI 8-Bit FIFO Data Register*
[0x0004]	*SPIn_CTRL0*	R/W	*SPI Controller Signals Control Register*
[0x0008]	*SPIn_CTRL1*	R/W	*SPI Transmit Packet Size Register*
[0x000C]	*SPIn_CTRL2*	R/W	*SPI Static Configuration Register*
[0x0010]	*SPIn_SSTIME*	R/W	*SPI Target Select Timing Register*
[0x0014]	*SPIn_CLKCTRL*	R/W	*SPI Controller Clock Control Register*
[0x001C]	*SPIn_DMA*	R/W	*SPI DMA Control Register*

Offset	Register Name	Access	Description
[0x0020]	*SPIn_INTFL*	R/W1C	*SPI Interrupt Flag Register*
[0x0024]	*SPIn_INTEN*	R/W	*SPI Interrupt Enable Register*
[0x0028]	*SPIn_WKFL*	R/W1C	*SPI Wake-up Flags Register*
[0x002C]	*SPIn_WKEN*	R/W	*SPI Wake-up Enable Register*
[0x0030]	*SPIn_STAT*	RO	*SPI Status Register*

18.5.1 Register Details

Table 18-7: SPI 32-Bit FIFO Data Register

SPI 32-Bit FIFO Data			SPIn_FIFO32	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**SPI 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. Reading from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 18-8: SPI 16-Bit FIFO Data Register

SPI 16-Bit FIFO Data			SPIn_FIFO16	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	data	R/W	0	**SPI 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 2-byte widths only. Reading from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 18-9: SPI 8-Bit FIFO Data Register

SPI 8-Bit FIFO Data			SPIn_FIFO8	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
8:0	data	R/W	0	**SPI 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 1-byte width only. Reading from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 18-10: SPI Control 0 Register

SPI Control 0			SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:16	ss_active	R/W	0	**Controller Target Select** The SPI includes up to four target select lines for each port. This field selects which target select pin is active when the next SPI transaction is started (*SPIn_CTRL0.start* = 1). One or more target select pins can be selected for each SPI transaction by setting the bit for each target select pin. For example, use SPIn_SS0 and SPIn_SS2 by setting this field to 0b0101 or select all target selects by setting this field to 0b1111. *Note: This field is only used when the SPI is configured for controller mode (SPIn_CTRL0.mst_mode = 1).*
15:9	-	RO	0	**Reserved**

SPI Control 0			SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
8	ss_ctrl	R/W	0	**Controller Target Select Control** This field controls the behavior of the target select pins at the completion of a transaction. The default behavior, *ss_ctrl* = 0, deasserts the target select pin at the completion of the transaction. Set this field to 1 to leave the target select pins asserted at the completion of the transaction. If the external device supports this behavior, leaving the target select pins asserted allows multiple transactions without the delay associated with the deassertion of the target select pin between transactions. 0: Target Select is deasserted at the end of a transmission. 1: Target Select stays asserted at the end of a transmission.
7:6	-	RO	0	**Reserved**
5	start	R/W1O	0	**Controller Start Data Transmission** Set this field to 1 to start an SPI controller mode transaction. 0: No controller mode transaction active. 1: Controller initiates data transmission. All pending transactions must be complete before setting this field to 1. *Note: This field is only used when the SPI is configured for controller mode (SPIn_CTRL0.mst_mode = 1).*
4	ss_io	R/W	0	**Controller Target Select Signal Direction** Set the I/O direction for the target select signal in controller mode. 0: Target Select is an output. 1: Target Select is an input. *Note: This field is only used when the SPI is configured for controller mode (SPIn_CTRL0.mst_mode = 1).*
3:2	-	RO	0	**Reserved**
1	mst_mode	R/W	0	**SPI Controller Mode Enable** This field selects between target and controller mode operation for the SPI port. Write this field to 0 to operate as an SPI target. Set this field to 1 to configure the port as an SPI controller. 0: Target mode SPI operation. 1: Controller mode SPI operation.
0	en	R/W	0	**SPI Enable/Disable** This field enables and disables the SPI port. Disable the SPI port by setting this field to 0. Disabling the SPI port does not affect the SPI FIFOs or register settings. Access to SPI registers is always available. 0: SPI port is disabled. 1: SPI port is enabled.

Table 18-11: SPI Transmit Packet Size Register

SPI Transmit Packet Size			SPIn_CTRL1	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	rx_num_char	R/W	0	**Number of Receive Characters** This field sets the number of characters to receive in the receive FIFO. *Note: If the SPI port is set to four-wire mode, this field is ignored, and the SPIn_CTRL1.tx_num_char 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** This field sets the number of characters to transmit from transmit FIFO. *Note: If the SPI port is set to four-wire mode, this field is used to receive and transmit the number of characters.*

Table 18-12: SPI Control 2 Register

SPI Control 2			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:16	ss_pol	R/W	0	**Target Select Polarity** Controls the polarity of each SS signal, where each bit position corresponds to a SS signal. SPIn_SS0 is controlled with bit position 0, and SPIn_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.*
14	-	RO	0	**Reserved**
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 SPIn_MOSI is used for half-duplex communication. ***Four-wire full-duplex SPI:*** *data_width* = 0 Set this field to 0, indicating SPIn_MOSI and SPIn_MISO are used for the SPI data output and input, respectively. ***Dual Mode SPI:*** *data_width* = 1, uses SPIn_SDIO0 and SPIn_SDIO1 for half-duplex communication. ***Quad Mode SPI:*** data_width = 2 Set this field to 2, indicating SPIn_SDIO0, SPIn_SDIO1, SPIn_SDIO2, and SPIn_SDIO3 are used 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 Mode SPI). 2: 4 bits per SCK cycle (Quad Mode 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.*
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.* *Note: 2-bit and 10-bit character lengths do not support maximum SCK speeds in controller mode. SPIn_CLKCTRL.clkdiv must be > 0.* *Note: For quad and dual mode SPI, the character size should be divisible by the number of bits per SCK cycle.*
7:2	-	RO	0	**Reserved**

SPI Control 2			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
1	clkpol	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	clkpha	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 18-13: SPI Target Select Timing Register

SPI Target Select Timing			SPIn_SSTIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
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 (target select inactive) and the start of the following transaction (target select active). 0: 256 1: 1 2: 2 … … 254: 254 255: 255 *Note: The SPIn_SSTIME register bit settings only apply when SPI is operating in controller mode (SPIn_CTRL0.mst_mode = 1).*
15:8	post	R/W	0	**Target Select Hold Post Last SCK** This field sets the number of system clock cycles that SS remains active after the last SCK edge. 0: 256 1: 1 2: 2 … … 254: 254 255: 255 *Note: The SPIn_SSTIME register bit settings only apply when SPI is operating in controller mode (SPIn_CTRL0.mst_mode = 1).*
7:0	pre	R/W	0	**Target Select Delay to First SCK** Set the number of system clock cycles the target Select is held active before the first SCK edge. 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255 *Note: The SPIn_SSTIME register bit settings only apply when SPI is operating in controller mode (SPIn_CTRL0.mst_mode = 1).*

Table 18-14: SPI Controller Clock Control Registers

SPI Controller Clock Control			SPIn_CLKCTRL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:16	clkdiv	R/W	0	**SPI Peripheral Clock Scale** Scales the SPI input clock (PCLK) by 2^{scale} to generate the SPI 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: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.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 *SPIn_CLKCTRL.clkdiv* = 0. 1 to 15: The number of SPI peripheral clocks, $f_{SPInCLK}$, that SCK is high. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*
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_CLKCTRL.hi* field. 0: Low duty cycle control disabled. Setting this field to 0 is only valid if *SPIn_CLKCTRL.clkdiv* = 0. 1 to 15: The number of SPI peripheral clocks, $f_{SPInCLK}$, that the SCK signal is low. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*

Table 18-15: SPI DMA Control Registers

SPI DMA Control			SPIn_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31	dma_rx_en	R/W	0	**Receive DMA Enable** 0: Disabled. Any pending DMA requests are cleared. 1: Enabled.
30	-	RO	0	**Reserved**
29:24	rx_lvl	R	0	**Number of Bytes in the Receive FIFO** Read returns the number of bytes currently in the receive FIFO.
23	rx_flush	W	-	**Clear the Receive FIFO** 1: Clear the receive FIFO and any pending receive FIFO flags in the *SPIn_INTFL* register. Write to this field only when the receive FIFO is inactive. *Note: Writing a 0 to this field has no effect.*
22	rx_fifo_en	R/W	0	**Receive FIFO Enabled** 0: Disabled. 1: Enabled.
21	-	RO	0	**Reserved**
20:16	rx_thd_val	R/W	0x00	**Receive FIFO Threshold Level** Set this value to the desired receive FIFO threshold level. When the receive FIFO level crosses above this setting, a DMA request is triggered if enabled by setting *SPIn_DMA.dma_rx_en*, and *SPIn_INTFL.rx_thd* becomes set. Valid values are 0 to 30. *Note: 31 is an invalid setting.*

SPI DMA Control			SPIn_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
15	dma_tx_en	R/W	0	**Transmit DMA Enable** 0: Disabled. Any pending DMA requests are cleared. 1: Transmit DMA is enabled.
14	-	RO	0	**Reserved**
13:8	tx_lvl	RO	0	**Number of Bytes in the Transmit FIFO** Read this field to determine the number of bytes currently in the transmit FIFO.
7	tx_flush	R/W	0	**Transmit FIFO Clear** Set this bit to clear the transmit FIFO and all transmit FIFO flags in the *SPIn_INTFL* 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	-	RO	0	**Reserved**
4:0	tx_thd_val	R/W	0x10	**Transmit FIFO Threshold Level** Set this value to the desired transmit FIFO threshold level. When the transmit FIFO count (*SPIn_DMA.tx_lvl*) falls below this value, a DMA request is triggered if enabled by setting *SPIn_DMA.dma_tx_en,* and *SPIn_INTFL.tx_thd* becomes set.

Table 18-16: SPI Interrupt Status Flags Registers

SPI Interrupt Status Flags			SPIn_INTFL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	rx_un	R/1	0	**Receive FIFO Underrun Flag** Set when a read is attempted from an empty receive FIFO.
14	rx_ov	R/W1C	0	**Receive FIFO Overrun Flag** Set if SPI is in target mode, and a write to a full receive FIFO is attempted. If the SPI is in controller mode, this bit is not set as the SPI stalls the clock until data is read from the receive FIFO.
13	tx_un	R/W1C	0	**Transmit FIFO Underrun Flag** Set if SPI is in target mode, and a read from empty transmit FIFO is attempted. If SPI is in controller mode, this bit is not set as the SPI stalls the clock until data is written to the empty transmit FIFO.
12	tx_ov	R/W1C	0	**Transmit FIFO Overrun Flag** Set when a write is attempted to a full transmit FIFO.
11	mst_done	R/W1C	0	**Controller Data Transmission Done Flag** This field is set if the SPI is in controller mode and all transactions are complete. *SPIn_CTRL1.tx_num_char* is reached.
10	-	RO	0	**Reserved**
9	abort	R/W1C	0	**Target Mode Transaction Abort Detected Flag** This field is set if the SPI is in target mode and the SS signal is deasserted before a complete character is received.
8	fault	R/W1C	0	**Multi-Controller Fault Flag** This field is set if the SPI is in controller mode, multicontroller mode is enabled, and a target select input is asserted. A collision also sets this flag.
7:6	-	RO	0	**Reserved**
5	ssd	R/W1C	0	**Target Select Deasserted Flag**
4	ssa	R/W1C	0	**Target Select Asserted Flag**
3	rx_full	R/W1C	0	**Receive FIFO Full Flag**

SPI Interrupt Status Flags			SPIn_INTFL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
2	rx_thd	R/W1C	0	**Receive FIFO Threshold Level Crossed Flag** Set when the receive FIFO exceeds the value in *SPIn_DMA.rx_thd_val*. Cleared once receive FIFO level drops below *SPIn_DMA.rx_thd_val.*
1	tx_em	R/W1C	1	**Transmit FIFO Empty Flag**
0	tx_thd	R/W1C	0	**Transmit FIFO Threshold Level Crossed Flag** Set when the transmit FIFO is less than the value in *SPIn_DMA.tx_thd_val*. Cleared once transmit FIFO level exceeds *SPIn_DMA.tx_thd_val.*

Table 18-17: SPI Interrupt Enable Registers

SPI Interrupt Enable			SPIn_INTEN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	rx_un	R/W	0	**Receive FIFO Underrun Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
14	rx_ov	R/W	0	**Receive FIFO Overrun Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
13	tx_un	R/W	0	**Transmit FIFO Underrun Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
12	tx_ov	R/W	0	**Transmit FIFO Overrun Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
11	mst_done	R/W	0	**Controller Data Transmission Done Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
10	-	RO	0	**Reserved**
9	abort	R/W	0	**Target Mode Abort Detected Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
8	fault	R/W	0	**Multi-Controller Fault Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
7:6	-	RO	0	**Reserved**
5	ssd	R/W	0	**Target Select Deasserted Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
4	ssa	R/W	0	**Target 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_thd	R/W	0	**Receive FIFO Threshold Level Crossed Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
1	tx_em	R/W	0	**Transmit FIFO Empty Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
0	tx_thd	R/W	0	**Transmit FIFO Threshold Level Crossed Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.

Table 18-18: SPI Wake-up Status Flags Registers

SPI Wake-up Flags			SPIn_WKFL	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	rx_full	R/W1C	0	**Wake on Receive FIFO Full Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
2	rx_thd	R/W1C	0	**Wake on Receive FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.

SPI Wake-up Flags			SPIn_WKFL	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
1	tx_em	R/W1C	0	**Wake on Transmit FIFO Empty Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
0	tx_thd	R/W1C	0	**Wake on Transmit FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.

Table 18-19: SPI Wake-up Enable Registers

SPI Wake-up Enable			SPIn_WKEN	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	rx_full	R/W	0	**Wake On Receive FIFO Full Enable** 0: Disabled. 1: Enabled.
2	rx_thd	R/W	0	**Wake On Receive FIFO Threshold Level Crossed Enable** 0: Disabled. 1: Enabled.
1	tx_em	R/W	0	**Wake On Transmit FIFO Empty Enable** 0: Disabled. 1: Enabled.
0	tx_thd	R/W	0	**Wake On Transmit FIFO Threshold Level Crossed Enable** 0: Disabled. 1: Enabled.

Table 18-20: SPI Target Select Timing Registers

SPI Status			SPIn_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	busy	R	0	**SPI Active Status** 0: SPI is not active. In controller mode, this flag is cleared when the last character is sent. In target mode, this flag is cleared when the configured target select input is deasserted. 1: SPI is active. In controller mode, this flag is set when a transaction starts. In target mode, this flag is set when a configured target select input is asserted. *Note: If this field is set, SPIn_CTRL0, SPIn_CTRL1, SPIn_CTRL2, SPIn_SSTIME, and SPIn_CLKCTRL should not be configured.*

19. I²C Controller/Target Serial Communications Peripheral

The I²C peripherals can be configured as either an I²C controller or an I²C target at standard data rates. For simplicity, I2Cn is used throughout this section to refer to any of the I²C peripherals.

For detailed information on I²C bus operation, refer to Analog Devices Application Note 4024 "SPI/I²C Bus Lines Control Multiple Peripherals" at *https://www.maximintegrated.com/en/app-notes/index.mvp/id/4024*.

19.1 I²C Controller/Target Features

Each I²C controller/target 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 controller or target 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 multicontroller systems, including support for arbitration and clock synchronization for Standard Mode, Fast Mode, and Fast Mode Plus.
- Supports 7- and 10-bit addressing.
- Supports RESTART condition.
- Supports clock stretching.
- Provides transfer status interrupts and flags.
- Provides DMA data transfer support.
- Supports I²C timing parameters fully controllable through software.
- Provides glitch filter and Schmitt trigger hysteresis on SDA and SCL.
- Provides control, status, and interrupt events for maximum flexibility.
- Provides independent 8-byte receive FIFO and 8-byte transmit FIFO.
- Provides transmit FIFO preloading.
- Provides programmable interrupt threshold levels for the transmit and receive FIFO.

19.2 Instances

The three instances of the peripheral are shown in *Table 19-1*. The table lists the alternate function names of the SDA and SCL signals for each of the I²C peripherals.

Table 19-1: MAX32690 I²C Peripheral Pins

I²C Instance	Alternate Function y = Alternate Function Number (A = AF1, B = AF2, C = AF2)*
I2C0	I2C0y_SCL
	I2C0y_SDA
I2C1	I2C1y_SCL
	I2C1y_SDA
I2C2	I2C2y_SCL
	I2C2y_SDA
** Refer to the device data sheet for alternate function and port pin mapping. Not all peripherals are available in all packages.*	

19.3 I²C Overview

19.3.1 I²C Bus Terminology

Table 19-2 contains terms and definitions used in this chapter for the I²C bus terminology.

Table 19-2: I²C Bus Terminology

Term	Definition
Transmitter	The device sending data on the bus.
Receiver	The device receiving data from the bus.
Controller	The device that initiates a transfer, generates the clock signal, and terminates a transfer.
Target	The device addressed by a controller.
Multicontroller	More than one controller can attempt to control the bus simultaneously without corrupting the message.
Arbitration	Procedure to ensure that, if more than one controller 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 target 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 controller does not have to support target clock stretching if none of the targets in the system are capable of clock stretching.

19.3.2 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 controller sends a START or repeated START condition, followed by the I²C target address of the targeted target device plus a read/write bit. The controller can transmit data to the target (a 'write' operation) or receive data from the target (a 'read' operation). Information is sent most-significant bit (MSB) first. Following the target address, the controller indicates a read or write operation and then exchanges data with the addressed target. An acknowledge bit is sent by the receiving device after each byte is transferred. When all necessary data bytes are transferred, a STOP or RESTART condition is sent by the bus controller to indicate the end of the transaction. After the STOP condition is sent, the bus is idle and ready for the next transaction. After a RESTART condition is sent, the same controller begins a new transmission. The number of bytes that can be transmitted per transfer is unrestricted.

19.3.3 START and STOP Conditions

A START condition occurs when a bus controller pulls SDA from high to low while SCL is high, and a STOP condition occurs when a bus controller 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.

19.3.4 Controller 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 target sends a NACK in response to a write operation, the I²C controller 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 controller stops the clock to allow time to read bytes from the receive FIFO or load bytes into the transmit FIFO.

19.3.5 Acknowledge and Not Acknowledge

An acknowledge bit (ACK) is generated by the receiver, whether I²C controller or target, 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 controller can then either generate a

STOP condition to abort the transfer or generate a repeated START condition (i.e., send a START condition without an intervening STOP condition) to start a new transfer.

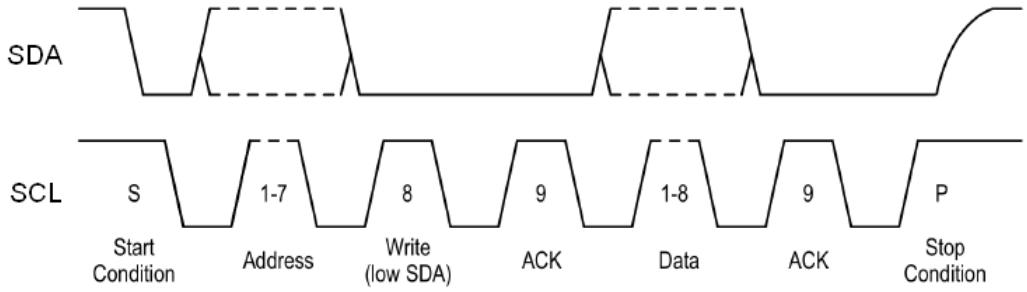
A receiver can generate a NACK after a byte transfer if any of the following conditions occur:

- 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 controller.
- 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 controller has requested data from a target, it signals the target to stop transmitting by sending a NACK following the last byte it requires.

19.3.6 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 able to be read when SCL is high, as shown in *Figure 19-1*.

Figure 19-1: I²C Write Data Transfer



An example of an I²C data transfer is as follows:

1. A bus controller indicates a data transfer to a target with a START condition.
2. The controller then transmits one byte with a 7-bit target 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 controller releases SDA. During this clock period, the addressed target responds with an ACK by pulling SDA low.
4. The controller senses the ACK condition and begins transferring data. If reading from the target, it floats SDA and allows the target to drive SDA to send data. After each byte, the controller drives SDA low to acknowledge the byte. If writing to the target, the controller 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 target to reply with the ACK indication.
5. After the last byte is transferred, the controller indicates the transfer is complete by generating a STOP condition. A STOP condition is generated when the controller pulls SDA from low to high while SCL is high.

19.4 Configuration and Usage

19.4.1 Peripheral Clock Enable

The I²C peripherals are disabled by default on a reset. Use of each of the I²C peripherals requires enabling the peripheral clock. Enable I2C0 by setting *GCR_PCLKDIS0.i2c0* to 0. Enable I2C1 by setting *GCR_PCLKDIS0.i2c1* to 0. Enable I2C2 by setting *GCR_PCLKDIS1.i2c2* to 0.

19.4.2 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 target device can hold SCL low, such as for clock stretching. Push-pull operation is enabled by setting *I2Cn_CTRL.scl* to 1. SDA, on the other hand, always operates in open-drain mode.

19.4.3 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.

19.4.4 SCL Clock Generation for Standard, Fast and Fast-Plus Modes

The controller generates the I²C clock on the SCL line. When operating as a controller, the software must configure the *I2Cn_CLKHI* and *I2Cn_CLKLO* registers for the desired I²C operating frequency.

The SCL high time is configured in the I²C Clock High Time register field *I2Cn_CLKHI.hi* using *Equation 19-2*. The SCL low time is configured in the I²C Clock Low Time register field *I2Cn_CLKLO.lo* using *Equation 19-3*. Each of these fields is 8 bits. The I²C frequency value is shown in *Equation 19-1*.

Equation 19-1: I²C Clock Frequency

$$f_{I2C\_CLK} = \frac{1}{t_{I2C\_CLK}} \; is\ either\ f_{PCLK}\ or\ f_{IBRO}$$

Equation 19-2: I²C Clock High Time Calculation

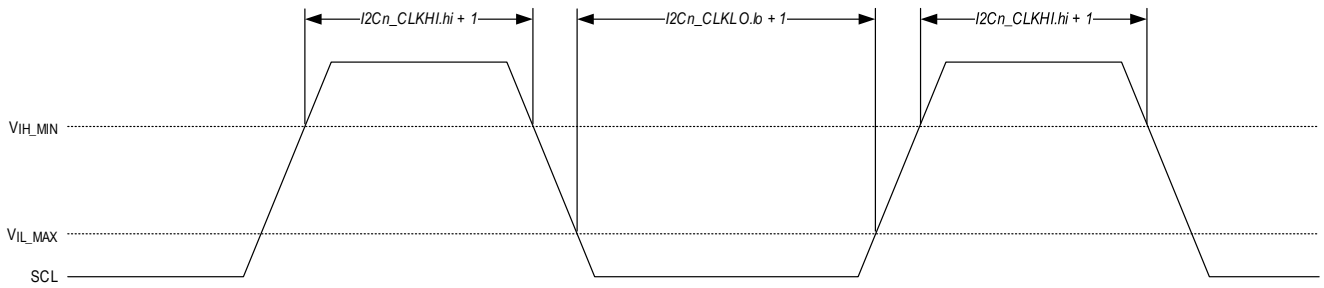
$$t_{SCL\_HI} = t_{I2C\_CLK} \times (I2Cn\_CLKHI.hi + 1)$$

Equation 19-3: I²C Clock Low Time Calculation

$$t_{SCL\_LO} = t_{I2C\_CLK} \times (I2Cn\_CLKLO.lo + 1)$$

Figure 19-2 shows the association between the SCL clock low and high times for Standard Mode, Fast Mode, and Fast Mode Plus I²C frequencies.

Figure 19-2: I²C SCL Timing for Standard, Fast and Fast-Plus Modes



During synchronization, external controllers or external targets can drive SCL simultaneously. This affects the SCL duty cycle. By monitoring SCL, the controller determines if an external controller or target 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 controller pulls SCL low before the controller has finished counting SCL high cycles, the controller starts counting SCL low cycles and releases SCL once the time period, *I2Cn_CLKLO.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, including bus loading, other devices on the bus holding SCL low, and the filter delay time of this device.

19.4.5 SCL Clock Generation for Hs-Mode

The values programmed into the *I2Cn_HSCLK.lo* register and *I2Cn_HSCLK.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 a preamble, 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.

19.4.5.1 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} = 160ns, the minimum low time for the I²C bus clock.

t_{HIGH_MIN} = 60ns, the minimum high time for the I²C bus clock.

t_{rCL_MAX} = 40ns, the maximum rise time of the I²C bus clock.

t_{fCL_MAX} = 40ns, the maximum fall time of the I²C bus clock.

19.4.5.2 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 19-4: I²C Target SCL Frequency

Desired Target Maximum I²C Frequency: $f_{SCL} = \frac{1}{t_{SCL}}$.

In Hs-Mode, the analog glitch filter within the device adds a minimum delay of t_{AF_MIN} = 10ns.

Equation 19-5: Determining the I2Cn_HSCLK.lo Register Value

$$I2Cn\_HSCLK.lo = MAX\left\{\left\lfloor\left(\frac{t_{LOW\_MIN} + t_{FCL\_MAX} + t_{I2C\_CLK} - t_{AF\_MIN}}{t_{I2C\_CLK}}\right)\right\rfloor - 1, \quad \frac{t_{SCL}}{t_{I2C\_CLK}} - 1\right\}$$

Equation 19-6: Determining the I2Cn_HSCLK.hi Register Value

$$I2Cn\_HSCLK.hi = \left\lfloor\left(\frac{t_{HIGH\_MIN} + t_{rCL\_MAX} + t_{I2C\_CLK} - t_{AF\_MIN}}{t_{I2C\_CLK}}\right)\right\rfloor - 1$$

Equation 19-7: The Calculated Frequency of the I²C Bus Clock Using the Results of Equation 19-5 *and* Equation 19-6

$$Calculated\ Frequency = \left((I2Cn\_HS\_CLK.hsclk\_hi + 1) + (I2Cn\_HS\_CLK.hsclk\_lo + 1)\right) * t_{I2C\_CLK}$$

Table 19-3 shows the I²C bus clock calculated frequencies given different f_{SYS_CLK} frequencies.

Table 19-3: Calculated I²C Bus Clock Frequencies

f_{SYS_CLK} (MHz)	*I2Cn_HSCLK.hi*	*I2Cn_HSCLK.lo*	Calculated Frequency (MHz)
100	4	9	3.3

f_{SYS_CLK} (MHz)	I2Cn_HSCLK.hi	I2Cn_HSCLK.lo	Calculated Frequency (MHz)
50	2	4	3.125
25	1	2	2.5

19.4.6 Controller Mode Addressing

After a START condition, the I²C target address byte is transmitted by the hardware. The I²C target address is composed of a target address followed by a read/write bit.

Table 19-4: I²C Target Address Format

Target 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	1*xx*	*x*	HS-mode controller code
1111	1*xx*	*x*	Reserved for future purposes
1111	0*xx*	*x*	10-bit target addressing

In 7-bit addressing mode, the controller sends one address byte. First, to address a 7-bit address target, clear the *I2Cn_MSTCTRL.ex_addr_en* field to 0, then write the address to the transmit FIFO formatted as follows, where A*n* is address A6:A0.

Controller writing to target: 7-bit address : [A6 A5 A4 A3 A2 A1 A0 0]

Controller reading from target: 7-bit address : [A6 A5 A4 A3 A2 A1 A0 1]

In 10-bit addressing mode (*I2Cn_MSTCTRL.ex_addr_en* = 1), the first byte the controller sends is the 10-bit target Addressing byte that includes 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 target. 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 target device.

19.4.7 Controller Mode Operation

The peripheral operates in controller mode when controller mode enable (*I2Cn_CTRL.mst_mode*) is set to 1. To initiate a transfer, the controller generates a START condition by setting *I2Cn_MSTCTRL.start* = 1. If the bus is busy, it does not generate a START condition until the bus is available.

A controller can communicate with multiple target devices without relinquishing the bus. Instead of generating a STOP condition after communicating with the first target, the controller generates a Repeated START condition, or RESTART, by setting *I2Cn_MSTCTRL.restart* = 1. If a transaction is in progress, the peripheral finishes the transaction before generating a RESTART. The peripheral then transmits the target address stored in the transmit FIFO. The *I2Cn_MSTCTRL.restart* bit is automatically cleared to 0 as soon as the controller begins a RESTART condition.

I2Cn_MSTCTRL.start is automatically cleared to 0 after the controller has completed a transaction and sent a STOP condition.

The controller can also generate a STOP condition by setting *I2Cn_MSTCTRL.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_MSTCTRL.stop* bit is cleared and ignored.

A target cannot generate START, RESTART, or STOP conditions. Therefore, when controller mode is disabled, the *I2Cn_MSTCTRL.start*, *I2Cn_MSTCTRL.restart*, and *I2Cn_MSTCTRL.stop* bits are all cleared to 0.

For controller mode operation, the following registers should only be configured when either:

1. The I²C peripheral is disabled,

   or

2. The I²C bus is guaranteed to be idle/free.

If this peripheral is the only controller on the bus, then changing the registers outside of a transaction (*I2Cn_MSTCTRL.start* = 0) satisfies this requirement:

- *I2Cn_CTRL.mst_mode*
- *I2Cn_CTRL.irxm_en*
- *I2Cn_CTRL.one_mst_mode*
- *I2Cn_CTRL.hs_en*
- *I2Cn_RXCTRL1.cnt*
- *I2Cn_MSTCTRL.ex_addr_en*
- *I2Cn_CLKLO.lo*
- *I2Cn_CLKHI.hi*
- *I2Cn_HSCLK.lo*
- *I2Cn_HSCLK.hi*

In contrast to the above set of register fields, the register fields below can be safely (re)programmed at any time:

- All interrupt flags and interrupt enable bits
- *I2Cn_TXCTRL0.thd_val*
- *I2Cn_RXCTRL0.thd_lvl*
- *I2Cn_TIMEOUT.scl_to_val*
- *I2Cn_DMA.rx_en*
- *I2Cn_DMA.tx_en*
- *I2Cn_FIFO.data*
- *I2Cn_MSTCTRL.start*
- *I2Cn_MSTCTRL.restart*
- *I2Cn_MSTCTRL.stop*

19.4.7.1 I²C Controller Mode Receiver Operation

When in controller mode, initiating a controller receiver operation begins with the following sequence:

1. Write the number of data bytes to receive to the I²C receive count field (*I2Cn_RXCTRL1.cnt*).
2. Write the I²C target address byte to the *I2Cn_FIFO* register with the R/W bit set to 1.
3. Send a START condition by setting *I2Cn_MSTCTRL.start* = 1.
4. The target address is transmitted by the controller from the *I2Cn_FIFO* register.
5. The I²C controller receives an ACK from the target, and the controller sets the address ACK interrupt flag (*I2Cn_INTFL0.addr_ack* = 1).
6. The I²C controller receives data from the target and automatically ACKs each byte. The software must retrieve this data by reading the *I2Cn_FIFO* register.
7. Once *I2Cn_RXCTRL1.cnt* data bytes are received, the I²C controller sends a NACK to the target and sets the Transfer Done Interrupt Status Flag (*I2Cn_INTFL0.done* = 1).
8. If *I2Cn_MSTCTRL.restart* or *I2Cn_MSTCTRL.stop* is set, then the I²C controller sends a repeated START or STOP, respectively.

19.4.7.2 I²C Controller Mode Transmitter Operation

When in controller mode, initiating a controller transmitter operation begins with the following sequence:

1. Write the I²C target 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 size of the transmit FIFO. (e.g., If the transmit FIFO size is 8 bytes, the software can write one address byte and seven data bytes before starting the transaction.)
3. Send a START condition by setting *I2Cn_MSTCTRL.start* = 1.
4. The controller transmits the target address byte written to the *I2Cn_FIFO* register.
5. The I²C controller receives an ACK from the target, and the controller sets the address ACK interrupt flag (*I2Cn_INTFL0.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 target 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_MSTCTRL.restart* or *I2Cn_MSTCTRL.stop*.
8. Once the controller sends all the remaining bytes and empties the transmit FIFO, it sets *I2Cn_INTFL0.done* and proceeds to send out either a RESTART condition if *I2Cn_MSTCTRL.restart* is set or a STOP condition if *I2Cn_MSTCTRL.stop* is set.

19.4.7.3 I²C Multicontroller Operation

The I²C protocol supports multiple controllers on the same bus. When the bus is free, two (or more) controllers might try to initiate communication simultaneously. This is a valid bus condition. If this occurs and the two controllers want to transmit different data and/or address different targets, only one controller can remain in controller mode and complete its transaction. The other controller must back off the transmission and wait until the bus is idle. This process by which the winning controller is determined is called bus arbitration.

The controller compares the data being transmitted on SDA to the value observed on SDA to determine which controller wins the arbitration for each address or data bit. If a controller attempts to transmit a 1 on SDA (i.e., the controller lets SDA float) but senses a 0 instead, then that controller loses arbitration, and the other controller that sent a zero continues with the transaction. The losing controller cedes the bus by switching off its SDA and SCL drivers.

Note: This arbitration scheme works with any number of bus controllers: if more than two controllers begin transmitting simultaneously, the arbitration continues as each controller cedes the bus until only one controller remains transmitting. Data is not corrupted because as soon as each controller realizes it has lost the arbitration, it stops transmitting on SDA, leaving the following data bits sent on SDA intact.

If the I²C controller peripheral detects it has lost the arbitration, it stops generating SCL; sets *I2Cn_INTFL0.arb_err*; sets *I2Cn_INTFL0.tx_lockout*, flushing any remaining data in the transmit FIFO; and clears *I2Cn_MSTCTRL.start*, *I2Cn_MSTCTRL.restart*, and *I2Cn_MSTCTRL.stop* to 0. As long as the peripheral is not addressed by the winning controller, the I²C peripheral stays in controller mode (*I2Cn_CTRL.mst_mode = 1)*. If, *at any time,* another controller addresses this peripheral using the address programmed in the I2Cn_SLAVE0 registers, then the I²C peripheral clears *I2Cn_CTRL.mst_mode* to 0 and begins responding as a target. 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_INTFL0.tx_lockout. Therefore, after an arbitration loss, the software needs to clear I2Cn_INTFL0.tx_lockout and reload the transmit FIFO.

Also, in a multicontroller environment, the software does *not* need to wait for the bus to become free before attempting to start a transaction (writing 1 to *I2Cn_MSTCTRL.start*). If the bus is free when *I2Cn_MSTCTRL.start* is set to 1, the transaction begins immediately. If, instead, the bus is busy, then the peripheral:

1. Waits for the other controller to complete the transaction(s) by sending a STOP,
2. Counts out the bus free time using $t_{BUF} = t_{SCL\_LO}$ (see *Equation 19-3*), *and then*
3. Sends a START condition and begins transmitting the target address byte(s) in the transmit FIFO, followed by the rest of the transfer.

The I²C controller 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.

19.4.8 Target Mode Operation

When in target mode, the I²C peripheral operates as a target device on the I²C bus and responds to an external controller's requests to transmit or receive data. To configure the I²C peripheral as a target, write the *I2Cn_CTRL.mst_mode* bit to zero. The controller drives the I²C clock on the bus, so the SCL device pin is driven by the external controller, and *I2Cn_STATUS.mst_busy* remains a zero. The desired target address must be set by writing to the *I2Cn_SLAVE0* register.

For target mode operation, the following register fields should be configured with the I²C peripheral disabled:

- *I2Cn_CTRL.mst_mode* = 0 for target operation.
- I²C target address:
  - Set up to four target addresses by programming the *I2Cn_SLAVE3.addr*, *I2Cn_SLAVE2.addr*, *I2Cn_SLAVE1.addr*, and/or the *I2Cn_SLAVE0.addr* fields must be set to the desired address for the device on the bus.
  - For extended addresses, set the corresponding *I2Cn_SLAVE3.ext_addr_en*, *I2Cn_SLAVE2.ext_addr_en*, *I2Cn_SLAVE1.ext_addr_en*, *I2Cn_SLAVE0.ext_addr_en* should be set to 1 for 10-bit addressing or 0 for 7-bit addressing.
  - Disable each I²C address register by setting the disable field to 1 (*I2Cn_SLAVE3.dis*, *I2Cn_SLAVE2.dis*, *I2Cn_SLAVE1.dis*, *I2Cn_SLAVE0.dis*).
- *I2Cn_CTRL.gc_addr_en*
- *I2Cn_CTRL.irxm_en*
  - The recommended value for this field is 0. *Note that a setting of 1 is incompatible with target mode operation with clock stretching disabled (I2Cn_CTRL.clkstr_dis = 1).*
- *I2Cn_CTRL.clkstr_dis*
- *I2Cn_CTRL.hs_en*
- *I2Cn_RXCTRL0.dnr*
  - SMBus/PMBus applications should set this to 0, while other applications should set this to 1.
- *I2Cn_TXCTRL0.nack_flush_dis*
- *I2Cn_TXCTRL0.rd_addr_flush_dis*
- *I2Cn_TXCTRL0.wr_addr_flush_dis*
- *I2Cn_TXCTRL0.gc_addr_flush_dis*
- *I2Cn_TXCTRL0.preload_mode*
  - The recommended value is 0 for applications that can tolerate target clock stretching (*I2Cn_CTRL.clkstr_dis* = 0).
  - The recommended value is 1 for applications that do not allow target clock stretching (*I2Cn_CTRL.clkstr_dis* = 1).
- *I2Cn_CLKHI.hi*
  - Applies to target mode when clock stretching is enabled (*I2Cn_CTRL.clkstr_dis* = 0)
    - This is used to satisfy $t_{SU;DAT}$ after clock stretching; program it so that the value defined by *Equation 19-2* is >= $t_{SU;DAT(min)}$.
- *I2Cn_HSCLK.hi*
  - Applies to target mode in Hs Mode when clock stretching is enabled (*I2Cn_CTRL.clkstr_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 19-6* is >= $t_{SU;DAT(min)}$.

In contrast to the above register fields, the following register fields can be safely (re)programmed at any time:

- All interrupt flags and interrupt enables.
- *I2Cn_TXCTRL0.thd_val* and *I2Cn_RXCTRL0.thd_lvl*
  - Transmit and receive FIFO threshold levels.
- *I2Cn_TXCTRL0.tx_ready_mode*
  - Transmit ready (can only be cleared by hardware).
- *I2Cn_TIMEOUT*.scl_to_val
  - Timeout control.
- *I2Cn_DMA.rx_en* and *I2Cn_DMA.tx_en*
  - Transmit and receive DMA enables.
- *I2Cn_FIFO.data*
  - FIFO access register.

19.4.8.1 Target Transmitter

The device operates as a target transmitter when the received address matches the device target address with the R/W bit set to 1. The controller is then reading from the device target. There two main modes of target transmitter operation: just-in-time mode and preload mode.

19.4.8.1.1 Just-in-Time Target Transmitter

In just-in-time mode, the software waits to write the transmit data to the transmit FIFO until after the controller addresses it for a READ transaction, just in time, to send the data to the controller. 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 I^2C 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.clkstr_dis* = 0) for just-in-time mode operation.

Program flow for target transmit operation in just-in-time mode is as follows:

1. With *I2Cn_CTRL.en* = 0, initialize all relevant registers, including:
   a. Set the *I2Cn_SLAVE3.addr*:*I2Cn_SLAVE0.addr* fields with the desired I^2C target addresses.
   b. Set the corresponding *I2Cn_SLAVE3.ext_addr_en*:*I2Cn_SLAVE0.ext_addr_en* fields for either 7-bit or 10-bit addressing.
   c. Disable any unused address registers by setting *I2Cn_SLAVE3.dis*:*I2Cn_SLAVE0.dis* fields to 1.
   d. Just-in-time mode specific settings:
      i) *I2Cn_CTRL*. *clkstr_dis* = 0
      ii) *I2Cn_TXCTRL0*[5:2] = 0x8
      iii) *I2Cn_TXCTRL0.preload_mode* = 0.
   e. Program *I2Cn_CLKHI.hi* and *I2Cn_HSCLK.hi* with appropriate values satisfying $t_{SU;DAT}$ (and HS $t_{SU;DAT}$).
2. The software sets *I2Cn_CTRL.en* = 1.
   a. The controller is now listening for its address. For either a transmit (R/W = 1) or receive (R/W = 0) operation, the peripheral responds to its address with an ACK.
   b. When the address match occurs, the hardware sets *I2Cn_INTFL0.addr_match* and *I2Cn_INTFL0.tx_lockout*.
3. The software waits for *I2Cn_INTFL0.addr_match* to read 1, either through polling the interrupt flag or setting *I2Cn_INTEN0.addr_match* to interrupt the CPU.

4. After reading *I2Cn_INTFL0.addr_match* =1, the software reads *I2Cn_CTRL.read* to determine whether the transaction is a transmit (read = 1) or receive (read = 0) operation. In this case, assume read = 1, indicating transmit.
   a. The hardware holds SCL low until the software clears *I2Cn_INTFL0.tx_lockout* and loads data into the FIFO.
5. The software clears *I2Cn_INTFL0.addr_match* and *I2Cn_INTFL0.tx_lockout*. Now that *I2Cn_INTFL0.tx_lockout* is 0, the software can begin loading the transmit data into *I2Cn_FIFO*.
6. As soon as there is data in the FIFO, the hardware releases SCL (after counting out *I2Cn_CLKHI.hi*) and sends out the data on the bus.
7. While the controller keeps requesting data and sending ACKs, *I2Cn_INTFL0.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_TXCTRL0.thd_val* and setting the *I2Cn_INTEN0.tx_thd* 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 controller ends the transaction by sending a NACK. Once this happens, the *I2Cn_INTFL0.done* interrupt flag is set, and the software can stop monitoring the transmit FIFO.
   a. If the software needs to know how many data bytes were transmitted to the controller, it should check the transmit FIFO level as soon as *I2Cn_INTFL0.done* = 1 and use it 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 the software to manually flush the transmit FIFO.
9. The transaction is complete. The software should clear the *I2Cn_INTFL0.done* interrupt flag and clear the *I2Cn_INTFL0.tx_thd* interrupt flag. Return to step 3, waiting on an address match.

19.4.8.1.2 Preload Mode Target Transmit

The other mode of operation for target 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 controller. 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 target transmit preload mode:

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers, including:
   a. Set the *I2Cn_SLAVE3*.*addr*:*I2Cn_SLAVE0*.*addr* fields with the desired I^2C target addresses.
   b. Set the corresponding *I2Cn_SLAVE3*.*ext_addr_en*:*I2Cn_SLAVE0*.*ext_addr_en* fields for either 7-bit or 10-bit addressing.
   c. Disable any unused address registers by setting *I2Cn_SLAVE3*.*dis*:*I2Cn_SLAVE0*.*dis* fields to 1.
   d. Preload mode specific settings:
      i) *I2Cn_CTRL*.*clkstr_dis* = 1
      ii) *I2Cn_TXCTRL0*[5:2] = 0xF
      iii) *I2Cn_TXCTRL0*.*preload_mode* = 1.
2. The software sets *I2Cn_CTRL*.*en* = 1.
   a. Even though the controller is enabled, it does not ACK an address match with R/W equal to 1 until the software sets the *I2Cn_TXCTRL1*.*preload_rdy* field to 1.
3. The software prepares for the transmit operation by loading data into the transmit FIFO, enabling DMA, setting *I2Cn_TXCTRL0*.*thd_val,* and setting *I2Cn_INTEN0*.*tx_thd* interrupt, etc.
   a. If clock stretching is disabled, 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_TXCTRL1*.*preload_rdy* = 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 clock stretching, which lets the transaction complete as quickly as possible.
4. Once the software has prepared for the transmit operation; it sets *I2Cn_TXCTRL1*.*preload_rdy* = 1.
   a. The controller is now fully enabled and responds with an ACK to an address match.
   b. The hardware sets *I2Cn_INTFL0*.*addr_match* when an address match occurs. *I2Cn_INTFL0*.*tx_lockout* is NOT set to 1 and remains 0.
5. The software waits for *I2Cn_INTFL0*.*addr_match* = 1, either through polling the interrupt flag or by setting *I2Cn_INTEN0*.*addr_match* to generate an interrupt when the event occurs.
6. After seeing *I2Cn_INTFL0*.*addr_match* =1, the software reads *I2Cn_CTRL*.*read* to determine if the transaction is a transmit (read = 1) or receive (read = 0) operation. In this case, assume *I2Cn_CTRL*.*read*, indicating a transmit.
   a. The hardware begins sending out the data that is preloaded into the transmit FIFO.
   b. Once the first data byte is sent, the hardware automatically clears *I2Cn_TXCTRL1*.*preload_rdy* to 0.
7. While the controller keeps requesting data and sending ACKs, *I2Cn_INTFL0*.*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_TXCTRL0*.*thd_val* and setting *I2Cn_INTEN0*.*tx_thd* interrupt.
   b. If clock stretching is disabled and the transmit FIFO empties during the transaction, the hardware sets *I2Cn_INTFL1*.*tx_un* = 1 and sends 0xFF for all following data bytes requested by the controller.
8. The controller ends the transaction by sending a NACK, causing the hardware to set the *I2Cn_INTFL0*.*done* interrupt flag.

a. If the transmit FIFO empties simultaneously that the controller indicates the transaction is complete by sending a NACK, this is not considered an underrun event *I2Cn_INTFL1*.*tx_un* flag remains 0.

b. If the software needs to know how many data bytes are transmitted to the controller, check the transmit FIFO level when the *I2Cn_INTFL0*.*done* flag is set to 1.

9. The transaction is complete, the software should "clean up," which includes clearing *I2Cn_INTFL0*.*done*. Return to step 3 and prepare for the next transaction.

a. Any data remaining in the transmit FIFO is not discarded; it is reused for the next transmit operation.

i) If this is not desired, the software can flush the transmit FIFO. Flush the transmit and receive FIFOs by writing 0 to *I2Cn_CTRL*.*en* and the writing 1 to *I2Cn_CTRL*.*en*.

Once a target starts transmitting from the *I2Cn_FIFO*, detecting an out of sequence STOP, START, or RESTART conditions terminates the current transaction. When a transaction is terminated due to an out of sequence error, *I2Cn_INTFL0*.*start_err* or *I2Cn_INTFL0*.*stop_err* is set to 1.

If the transmit FIFO is not ready (*I2Cn_TXCTRL1*.*preload_rdy* = 0) and the I^2C controller receives a data read request from the controller, the hardware automatically sends a NACK at the end of the first address byte. The setting of the do not respond field is ignored by the hardware in this case because the only opportunity to send a NACK for an I^2C read transaction is after the address byte.

19.4.8.2 Target Receivers

The device operates as a target receiver when the received address matches the device target address with the R/W bit set to 0. The external controller is writing to the target.

Program flow for a receive operation is as follows:

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers, including:
   a. Set the *I2Cn_SLAVE3*.*addr*:*I2Cn_SLAVE0*.*addr* fields with the desired I²C target addresses.
   b. Set the corresponding *I2Cn_SLAVE3*.*ext_addr_en*:*I2Cn_SLAVE0*.*ext_addr_en* fields for either 7-bit or 10-bit addressing.
   c. Disable any unused address registers by setting *I2Cn_SLAVE3*.*dis*:*I2Cn_SLAVE0*.*dis* fields to 1.
2. Set *I2Cn_CTRL*.*en* = 1.
   a. If an address match with the R/W bit equal to zero occurs, and the receive FIFO is empty, the peripheral responds with an ACK, and the *I2Cn_INTFL0*.*addr_match* flag is set.
   b. If the receive FIFO is not empty, then depending on the value of *I2Cn_RXCTRL0*.*dnr*, the peripheral NACKs either the address byte (*I2Cn_RXCTRL0*.*dnr* = 1) or the first data byte (*I2Cn_RXCTRL0*.*dnr* = 0).
3. Wait for *I2Cn_INTFL0*.*addr_match* = 1, either by polling or by enabling the *wr_addr_match* interrupt. Once a successful address match occurs, the hardware sets *I2Cn_INTFL0*.*addr_match* = 1.
4. Read *I2Cn_CTRL*.*read* to determine if the transaction is a transmit (*I2Cn_CTRL*.*read* = 1) or a receive (*I2Cn_CTRL*.*read* = 0) operation. In this case, assume *I2Cn_CTRL*.*read* = 0, indicating receive. The device begins receiving data into the receive FIFO.
5. Clear *I2Cn_INTFL0*.*addr_match*, and while the controller keeps sending data, *I2Cn_INTFL0*.*done* remains 0, and the 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_RXCTRL0*.*thd_lvl* and enabling the *I2Cn_INTFL0*.*rx_thd* interrupt.
   b. If the receive FIFO ever fills up during the transaction, then the hardware sets *I2Cn_INTFL1*.*rx_ov* and then either:
      i. If *I2Cn_CTRL*.*clkstr_dis* = 0, start clock stretching and wait until the software reads from the receive FIFO, *or*
      ii. If *I2Cn_CTRL*.*clkstr_dis* = 1, respond to the controller with a NACK, and the last byte is discarded.
6. The controller ends the transaction by sending a RESTART or STOP. Once this happens, the *I2Cn_INTFL0*.*done* interrupt flag is set, and the software can stop monitoring the receive FIFO.
7. Once a target starts receiving into its receive FIFO, detection of an out of sequence STOP, START, or RESTART condition releases the I²C bus to the Idle state, and the hardware sets the *I2Cn_INTFL0*.*start_err* field or *I2Cn_INTFL0*.*stop_err* field to 1 based on the specific condition.

If the software has not emptied the data in the receive FIFO from the previous transaction by the time a controller 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 controller, the software can control if the NACK is sent with the initial address match or sent at the end of the first data byte. Setting *I2Cn_RXCTRL0*.*dnr* to 1 chooses the former while setting *I2Cn_RXCTRL0*.*dnr* to 0 chooses the latter.

19.4.9 Interrupt Sources

The I²C controller has a very flexible interrupt generator that generates an interrupt signal to the interrupt controller on any of several events. On recognizing the I²C interrupt, the software determines the cause of the interrupt by reading the I²C interrupt flags registers *I2Cn_INTFL0* and *I2Cn_INTFL1*. Interrupts can be generated for the following events:

- Transaction Complete (controller/target).
- Address NACK received from target (controller).
- Data NACK received from target (controller).
- Lost arbitration (controller).
- Transaction timeout (controller/target).
- FIFO is empty, not empty, or full to a configurable threshold level (controller/target).
- Transmit FIFO locked out because it is being flushed (controller/target)
- Out of sequence START and STOP conditions (controller/target).
- Sent a NACK to an external controller because the transmit or receive FIFO was not ready (target).
- Address ACK or NACK received (controller).
- Incoming address match (target)
- Transmit underflow or receive overflow (target).

Interrupts for each event can be enabled or disabled by setting or clearing the corresponding bit in the *I2Cn_INTEN0* or *I2Cn_INTEN1* interrupt enable register.

Note: Disabling the interrupt does not prevent the corresponding flag from being set by the hardware but does prevent an interrupt 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, preventing a previous interrupt event from interfering with a new I²C communications session.

19.4.10 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* dequeue 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 greater 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 zeros 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 *DMA Control* for more information on configuring the DMA.

During a receive transaction (which during controller operation is a READ, and during target operation is a WRITE), received bytes are automatically written to the receive FIFO. The 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 controller mode transaction, then the hardware sets the *I2Cn_INTFL1.rx_ov* the *I2Cn_INTFL1.rx_ov* bit, and one of two things occur depending on the value of *I2Cn_CTRL.clkstr_dis*:

- If clock stretching is enabled (*I2Cn_CTRL.clkstr_dis* = 0), then the hardware stretches the clock until the software makes space available in the receive FIFO by reading *I2Cn_FIFO*. Once space is available, the hardware moves the

data byte from the shift register into the receive FIFO, the SCL device pin is released, and the controller is free to continue the transaction.

- If clock stretching is disabled (*I2Cn_CTRL.clkstr_dis* = 1), the hardware responds to the controller with a NACK, and the data byte is lost. The controller 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 controller operation is a WRITE, and during target operation is a READ), either the 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.

Both the receive FIFO and transmit FIFO are flushed when the I2Cn port is disabled by clearing *I2Cn_CTRL.en* to 0. While the peripheral is disabled, writes to the transmit FIFO have 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_TXCTRL0.flush*=1) or the receive FIFO flush bit (*I2Cn_RXCTRL0.flush*=1), respectively. In addition, under certain conditions, the transmit FIFO is automatically locked by the hardware and flushed so stale data is not unintentionally transmitted. The transmit FIFO is automatically flushed and writes locked out from the software under the following conditions:

- General Call Address Match: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.gc_addr_flush_dis*.
- Target Address Match Write: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.wr_addr_flush_dis*.
- Target Address Match Read: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.rd_addr_flush_dis*.
- During operation as a target transmitter, a NACK is received. Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.nack_flush_dis*.
- Any of the following interrupts:
  - Arbitration error, timeout error, controller mode address NACK error, controller 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 transmitted unintentionally for a new transaction. In addition to flushing the transmit FIFO, the transmit lockout flag is set (*I2Cn_INTFL0.tx_lockout* = 1) and writes to the transmit FIFO are ignored until the software acknowledges the external event by clearing *I2Cn_INTFL0.tx_lockout*.

19.4.11 Transmit FIFO Preloading

There can be situations during target mode operation where the 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 controller requesting data with an ACK and clock stretching while the software writes the data to the transmit FIFO, the hardware responds with a NACK until the software has preloaded the requested data into the transmit FIFO.

When transmit FIFO preloading is enabled, the software controls ACKs to the external controller using the transmit ready (*I2Cn_TXCTRL1.preload_rdy*) bit. When *I2Cn_TXCTRL1.preload_rdy* is set to 0, the hardware automatically NACKs all read

transactions from the controller. Setting *I2Cn_TXCTRL1.preload_rdy* to 1 sends an ACK to the controller on the next read transaction and transmits the data in the transmit FIFO. Preloading the transmit FIFO should be complete before setting the *I2Cn_TXCTRL1.preload_rdy* field to 1.

The required steps for implementing transmit FIFO preloading in software are as follows:

1. Enable the transmit FIFO preloading by setting the field *I2Cn_TXCTRL0.preload_mode* to 1. The hardware automatically clears the *I2Cn_TXCTRL1.preload_rdy* field to 0.
2. If the transmit FIFO lockout flag (*I2Cn_INTFL0.tx_lockout*) 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 controller sends the next read request.
5. Set *I2Cn_TXCTRL1.preload_rdy* to 1 to automatically let the hardware send the preloaded FIFO on the next read from a controller.
6. *I2Cn_TXCTRL1.preload_rdy* is cleared by the hardware once it finishes transmitting the first byte, and data is transmitted from the transmit FIFO. Once cleared, the software can repeat the preloading process or disable transmit FIFO preloading.

Note: To prevent the preloaded data from being cleared when the controller tries to read it, the software must at least set I2Cn_TXCTRL0.rd_addr_flush_dis to 1, disabling auto flush on READ address match. The software determines if the other auto flush disable bits should be set. For example, if a controller uses I²C WRITE transactions to determine what data the target should send in the following READ transactions, the software can clear I2Cn_TXCTRL0.wr_addr_flush_dis to 0. When a WRITE occurs, the transmit FIFO is flushed, giving the software time to load the new data. For the READ transaction, the external controller can poll the target address until the new data is loaded and I2Cn_TXCTRL1.preload_rdy is set, at which point the peripheral responds with an ACK.

19.4.12 Interactive Receive Mode (IRXM)

In some situations, the I2Cn 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.irxm_en* = 1. If IRXM is enabled, it must occur before any I²C transfer is initiated.

When IRXM is enabled, after every data byte received, the I2Cn peripheral automatically holds SCL low before the ACK bit. Additionally, after the 8th SCL falling edge, the I2Cn peripheral sets the IRXM interrupt status flag (*I2Cn_INTFL0.irxm* = 1). Software must read the data and generate a response (ACK or NACK) by setting the IRXM Acknowledge (*I2Cn_CTRL.irxm_ack*) bit accordingly. Send an ACK by clearing the *I2Cn_CTRL.irxm_ack* bit to 0. Send a NACK by setting the *I2Cn_CTRL.irxm_ack* bit to 1.

After setting the *I2Cn_CTRL.irxm_ack* bit, clear the IRXM interrupt flag. Write 1 to *I2Cn_INTFL0.irxm* to clear the interrupt flag. When the IRXM interrupt flag is cleared, the I2Cn peripheral hardware releases the SCL line and sends the *I2Cn_CTRL.irxm_ack* on the SDA line.

While the I2Cn peripheral is waiting for the software to clear the *I2Cn_INTFL0.irxm* flag, the software can disable IRXM and, if operating as a controller, 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 only applies to data bytes and does not apply to address bytes, general call address responses, or START byte responses.

Note: When enabling IRXM and operating as a target, clock stretching must remain enabled (I2Cn_CTRL.clkstr_dis = 0).

19.4.13 Clock Stretching

When the I2Cn 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 target device holding the SCL line low, this section describes situations where the I2Cn peripheral holds the SCL line low in either target *or* controller mode and refers to *both* as clock stretching.

When the I2Cn 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.irxm_en* = 1), the peripheral can also clock stretch *before* the ACK bit, allowing the software to decide if to send an ACK or NACK.

For a transmit operation (as either controller or target), 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 *I2Cn_FIFO.data* to stop clock stretching and continue the transaction. However, if operating in controller mode instead of sending more data, the software can also set either *I2Cn_MSTCTRL.stop* or *I2Cn_MSTCTRL.restart* to send a STOP or RESTART condition, respectively.

For a receive operation (as either controller or target), 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 *I2Cn_FIFO.data* to stop clock stretching and continue the transaction. If operating in controller mode and this is the final byte of the transaction, as determined by *I2Cn_RXCTRL1.cnt*, the software must also set either *I2Cn_MSTCTRL.stop* or *I2Cn_MSTCTRL.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 is moved from the receive shift register into the receive FIFO. This occurs automatically once there is space in the receive FIFO.

Note: Since some controllers do not support other devices stretching the clock, it is possible to completely disable all clock stretching during target mode by setting I2Cn_CTRL.clkstr_dis to 1 and clearing I2Cn_CTRL.irxm_en to 0. In this case, instead of clock stretching, the I2Cn peripheral sends a NACK if receiving data or sends 0xFF if transmitting data.

Note: The clock synchronization required to support other I²C controller or target devices stretching the clock is built into the peripheral and requires no intervention from the software to operate correctly.

19.4.14 Bus Timeout

The timeout field, *I2Cn_TIMEOUT.scl_to_val,* is used to detect bus errors. *Equation 19-8* and *Equation 19-9* show equations for calculating the maximum and minimum timeout values based on the value loaded into the *I2Cn_TIMEOUT.scl_to_val* field.

Equation 19-8: I²C Timeout Maximum

$$t_{TIMEOUT} \leq \left(\frac{1}{f_{I2C\_CLK}}\right) \times \left((I2Cn\_TIMEOUT.scl\_to\_val \times 32) + 3\right)$$

Due to clock synchronization, the timeout is guaranteed to meet the following minimum time calculation shown in *Equation 19-9*.

Equation 19-9: I²C Timeout Minimum

$$t_{TIMEOUT} \leq \left(\frac{1}{f_{I2C\_CLK}}\right) \times \left((I2Cn\_TIMEOUT.scl\_to\_val \times 32) + 2\right)$$

The timeout feature is disabled when *I2Cn_TIMEOUT.scl_to_val* = 0 and is enabled for any non-zero value. When the timeout is enabled, the timeout timer starts counting when the I2Cn peripheral hardware drives SCL low and is reset by the I2Cn peripheral hardware when the SCL line is released.

The timeout counter only monitors if the I2Cn peripheral hardware is driving the SCL line low. It does not monitor if an external I2Cn 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 occurred. When a timeout error occurs, the I2Cn peripheral hardware releases the SCL and SDA lines, and sets the timeout error interrupt flag to 1 (*I2Cn_INTFL0.to_err* = 1).

For applications where the device can 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.scl_to_val* = 0).

19.4.15 DMA Control

There are independent DMA channels for each transmit FIFO, and each receive FIFO. DMA activity is triggered by the transmit FIFO (*I2Cn_TXCTRL0.thd_val*) and receive FIFO (*I2Cn_RXCTRL0.thd_lvl*) threshold levels.

When the transmit FIFO byte count (*I2Cn_TXCTRL1.lvl*) is less than or equal to the transmit FIFO threshold level *I2Cn_TXCTRL0.thd_val*, then the DMA transfers data into the transmit FIFO according to the DMA configuration.

The DMA burst size should be set as shown in *Equation 19-10* to ensure the DMA does not overflow the transmit FIFO:

Equation 19-10: DMA Burst Size Calculation for I²C Transmit

$$DMA\ Burst\ Size\ \leq TX\ FIFO\ Depth - I2Cn\_TXCTRL0.thd\_val = 8 - I2Cn\_TXCTRL0.thd\_val$$

$$where\ 0\ \leq I2Cn\_TXCTRL0.thd\_val\ \leq 7$$

Software trying to avoid transmit underflow and/or clock stretching should use a smaller burst size and higher *I2Cn_TXCTRL0.thd_val* setting. This fills up the FIFO more frequently but increases internal bus traffic.

When the receive FIFO count (*I2Cn_RXCTRL1.lvl*) is greater than or equal to the receive FIFO threshold level *I2Cn_RXCTRL0.thd_lvl*, the DMA transfers data out of the receive FIFO according to the DMA configuration. The DMA burst size should be set as shown in *Equation 19-11* to ensure the DMA does not underflow the receive FIFO:

Equation 19-11: DMA Burst Size Calculation for I²C Receive

$$DMA\ Burst\ Size \leq I2Cn\_RXCTRL0.thd\_lvl$$

$$where\ 1 \leq I2Cn\_RXCTRL0.thd\_lvl \leq 8$$

Applications trying to avoid receive overflow and/or clock stretching should use a smaller burst size and lower *I2Cn_RXCTRL0.thd_lvl*. 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_RXCTRL0.thd_lvl. 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_RXCTRL0.thd_lvl = 1).

Enable the transmit DMA channel (*I2Cn_DMA.tx_en*) and/or the receive DMA channel (*I2Cn_DMA.rx_en*) to enable DMA transfers.

19.5 Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple peripheral instances are provided, each instance has its own independent set of the registers shown in *Table 16-1*. 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 1-1* 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 reset.

Table 19-5: Register Summary

Offset	Register	Description
[0x0000]	*I2Cn_CTRL*	I²C Control Register
[0x0004]	*I2Cn_STATUS*	I²C Status Register
[0x0008]	*I2Cn_INTFL0*	I²C Interrupt Flags 0 Register
[0x000C]	*I2Cn_INTEN0*	I²C Interrupt Enable 0 Register
[0x0010]	*I2Cn_INTFL1*	I²C Interrupt Flags 1 Register
[0x0014]	*I2Cn_INTEN1*	I²C Interrupt Enable 1 Register
[0x0018]	*I2Cn_FIFOLEN*	I²C FIFO Length Register
[0x001C]	*I2Cn_RXCTRL0*	I²C Receive Control 0 Register
[0x0020]	*I2Cn_RXCTRL1*	I²C Receive Control 1 Register
[0x0024]	*I2Cn_TXCTRL0*	I²C Transmit Control 0 Register
[0x0028]	*I2Cn_TXCTRL1*	I²C Transmit Control 1 Register
[0x002C]	*I2Cn_FIFO*	I²C Transmit and Receive FIFO Register
[0x0030]	*I2Cn_MSTCTRL*	I²C Controller Register
[0x0034]	*I2Cn_CLKLO*	I²C Clock Low Time Register
[0x0038]	*I2Cn_CLKHI*	I²C Clock High Time Register
(0x003C)	*I2Cn_HSCLK*	I²C Hs-Mode Clock Control Register
[0x0040]	*I2Cn_TIMEOUT*	I²C Timeout Register
[0x0048]	*I2Cn_DMA*	I²C DMA Enable Register
[0x004C]	*I2Cn_SLAVE0*	I²C Target Address 0 Register
[0x0050]	*I2Cn_SLAVE1*	I²C Target Address 1 Register
[0x0054]	*I2Cn_SLAVE2*	I²C Target Address 2 Register
[0x0058]	*I2Cn_SLAVE3*	I²C Target Address 3 Register

19.5.1 Register Details

Table 19-6: I²C Control Register

I²C Control			I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	hs_en	R/W	0	**Hs-Mode Enable** I²C high speed mode operation 0: Disabled. 1: Enabled.
14	-	RO	0	**Reserved**
13	one_mst_mode	R/W	0	**Single Controller Only** When set to 1, the device MUST ONLY be used in a single controller application with target devices that are NOT going to hold SCL low (i.e., the target devices never clock stretch).
12	clkstr_dis	R/W	0	**Target Mode Clock Stretching** 0: Enabled. 1: Disabled.
11	read	R	0	**Target Read/Write Bit Status** Returns the logic level of the R/W bit on a received address match (*I2Cn_INTFL0.addr_match* = 1) or general call match (*I2Cn_INTFL0.gc_addr_match* = 1). This bit is valid for three system clock cycles after the address match status flag is set.

I²C Control			I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
10	bb_mode	R/W	0	**Software Output Control Enabled** Setting this field to 1 enables software bit-bang control of the I2Cn Bus. 0: The I²C controller manages the SDA and SCL pins in the 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.bb_mode=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.bb_mode =1*
5	-	RO	0	**Reserved**
4	irxm_ack	R/W	0	**IRXM Acknowledge** If IRXM is enabled (*I2Cn_CTRL.irxm_en* = 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	irxm_en	R/W	0	**IRXM Enable** When receiving data, this field allows for an IRXM interrupt event after each received byte of data. The I2Cn peripheral hardware can be enabled to send either an ACK or NACK for IRXM. See the *Interactive Receive Mode* section for detailed information. 0: Disabled. 1: Enabled. *Note: Only set this field when the I²C bus is inactive.*
2	gc_addr_en	R/W	0	**General Call Address Enable** 0: Ignore general call address. 1: Acknowledge general call address.
1	mst_mode	R/W	0	**Controller Mode Enable** 0: Target mode enabled. 1: Controller mode enabled.
0	en	R/W	0	**I²C Peripheral Enable** 0: Disabled. 1: Enabled.

Table 19-7: I²C Status Register

I²C Status			I2Cn_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**

I²C Status			I2Cn_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
5	mst_busy	RO	0	**Controller Mode I²C Bus Transaction Active** The peripheral is operating in controller mode, and a valid transaction beginning with a START command is in progress on the I²C bus. This bit reads 1 until the controller ends the transaction with a STOP command. This bit continues to read 1 while a target performs clock stretching. 0: Device not actively driving SCL clock cycles. 1: Device operating as controller and actively driving SCL clock cycles.
4	tx_full	RO	0	**Transmit FIFO Full** 0: Not full. 1: Full.
3	tx_em	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_em	RO	1	**Receive FIFO Empty** 0: Not empty. 1: Empty.
0	busy	RO	0	**Controller or Target Mode I²C Busy Transaction Active** The peripheral is operating in controller or target 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 controller or an external controller ends the transaction with a STOP command. This bit continues to read 1 while a target performs clock stretching. 0: I²C bus is idle. 1: I²C bus transaction in progress.

Table 19-8: I²C Interrupt Flag 0 Register

I²C Interrupt Flag 0			I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23	wr_addr_match	R/W1C	0	**Target Write Address Match Interrupt Flag** If set, the device has been accessed for a write (i.e., receive) transaction in target mode, and the address received matches the device target address. 0: No address match. 1: Address match.
22	rd_addr_match	R/W1C	0	**Target Read Address Match Interrupt Flag** If set, the device has been accessed for a read (i.e., transmit) transaction in target mode, and the address received matches the device target address. 0: No address match. 1: Address match.
21:17	-	RO	0	**Reserved**
16	mami	R/W1C	0	**MAMI Interrupt Flag**
15	tx_lockout	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.

I²C Interrupt Flag 0			I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
14	stop_err	R/W1C	0	**Out of Sequence STOP Interrupt Flag** This flag is set if a STOP condition occurs out of the 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_err	R/W1C	0	**Out of Sequence START Interrupt Flag** This flag is set if a START condition occurs out of the 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	dnr_err	R/W1C	0	**Target Mode Do Not Respond Interrupt Flag** This occurs 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 occurred, and either the transmit or receive FIFO is not configured.
11	data_err	R/W1C	0	**Controller Mode Data NACK from External Target Interrupt Flag** The hardware sets this flag if a NACK is received from a target. This flag is only valid if the I2Cn peripheral is configured for controller mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Data NACK received from a target.
10	addr_nack_err	R/W1C	0	**Controller Mode Address NACK from Target Error Flag** The hardware sets this flag if an Address NACK is received from a target bus. This flag is only valid if the I2Cn peripheral is configured for controller mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Address NACK received from a target.
9	to_err	R/ W1C	0	**Timeout Error Interrupt Flag** This flag is set when this device holds SCL low longer than the programmed timeout value. This field's setting applies to both controller and target mode. Write 1 to clear. Write 0 has no effect. 0: Timeout error has not occurred. 1: Timeout error occurred.
8	arb_err	R/ W1C	0	**Controller Mode Arbitration Lost Interrupt Flag** Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: Condition occurred.
7	addr_ack	R/ W1C	0	**Controller Mode Address ACK from External Target Interrupt Flag** This field is set when a target address ACK is received. Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: The target device ACK for the address was received.
6	stop	R/ W1C	0	**Target 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.

I²C Interrupt Flag 0			I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
5	tx_thd	RO	1	**Transmit FIFO Threshold Level Interrupt Flag** The hardware sets this field 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 the 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 the transmit threshold level or fewer bytes.
4	rx_thd	R/W1C	1	**Receive FIFO Threshold Level Interrupt Flag** The hardware sets this field 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.
3	addr_match	R/W1C	0	**Target Mode Incoming Address Match Status Interrupt Flag** Write 1 to clear. Writing 0 has no effect. 0: Target address match has not occurred. 1: Target address match occurred.
2	gc_addr_match	R/W1C	0	**Target 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	irxm	R/W1C	0	**Interactive Receive Mode 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 controller and target mode once a transaction completes. Write 1 to clear. Writing 0 has no effect. 0: Transfer is not complete. 1: Transfer complete.

Table 19-9: I²C Interrupt Enable 0 Register

I²C Interrupt Enable 0			I2Cn_INTEN0	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23	wr_addr_match	R/W	0	**Target Write Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in target mode, and the address received matches the device target addressed for a write transaction. 0: Disabled. 1: Enabled.
22	rd_addr_match	R/W	0	**Target Read Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in target mode, and the address received matches the device target addressed for a read transaction. 0: Disabled. 1: Enabled.
21:17	-	RO	0	**Reserved**
16	mami	R/W	0	**MAMI Interrupt Enable**
15	tx_lockout	R/W	0	**Transmit FIFO Lock Out Interrupt Enable** 0: Disabled. 1: Enabled.

I²C Interrupt Enable 0			I2Cn_INTEN0	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
14	stop_err	R/W	0	**Out of Sequence STOP Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
13	start_err	R/W	0	**Out of Sequence START Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
12	dnr_err	R/W	0	**Target Mode Do Not Respond Interrupt Enable** Set this field to enable interrupts in target mode when the "Do Not Respond" condition occurs. 0: Interrupt disabled. 1: Interrupt enabled.
11	data_err	R/W	0	**Controller Mode Received Data NACK from Target Interrupt Enable** 0: Disabled. 1: Enabled.
10	addr_nack_err	R/W	0	**Controller Mode Received Address NACK from Target Interrupt Enable** 0: Disabled. 1: Enabled.
9	to_err	R/W	0	**Timeout Error Interrupt Enable** 0: Disabled. 1: Enabled.
8	arb_err	R/W	0	**Controller Mode Arbitration Lost Interrupt Enable** 0: Disabled. 1: Enabled.
7	addr_ack	R/W	0	**Received Address ACK from Target Interrupt Enable** Set this field to enable interrupts for controller mode target device address ACK events. 0: Interrupt disabled. 1: Interrupt enabled.
6	stop	R/W	0	**STOP Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
5	tx_thd	R/W	0	**Transmit FIFO Threshold Level Interrupt Enable** 0: Disabled. 1: Enabled.
4	rx_thd	R/W	0	**Receive FIFO Threshold Level Interrupt Enable** 0: Disabled. 1: Enabled.
3	addr_match	R/W	0	**Target Mode Incoming Address Match Interrupt Enable** 0: Disabled. 1: Enabled.
2	gc_addr_match	R/W	0	**Target Mode General Call Address Match Received Interrupt Enable** 0: Disabled. 1: Enabled.
1	irxm	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 19-10: I²C Interrupt Flag 1 Register

I²C Interrupt Status Flags 1			I2Cn_INTFL1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**

I²C Interrupt Status Flags 1			I2Cn_INTFL1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
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_un	R/W1C	0	**Target Mode Transmit FIFO Underflow Status Flag** In target mode operation, the hardware sets this flag automatically if the transmit FIFO is empty and the controller requests more data by sending an ACK after the previous byte is transferred. 0: Target mode transmit FIFO underflow condition has not occurred. 1: Target mode transmit FIFO underflow condition occurred.
0	rx_ov	R/W1C	0	**Target Mode Receive FIFO Overflow Status Flag** In target mode operation, the hardware sets this flag automatically when a receive FIFO overflow occurs. Write 1 to clear. Writing 0 has no effect. 0: Target mode receive FIFO overflow event has not occurred. 1: Target mode receive FIFO overflow condition occurred (data lost).

Table 19-11: I²C Interrupt Enable 1 Register

I²C Interrupt Enable 1			I2Cn_INTEN1	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	start	R/W	0	**START Condition Interrupt Enable** 0: Disabled. 1: Enabled.
1	tx_un	R/W	0	**Target Mode Transmit FIFO Underflow Interrupt Enable** 0: Disabled. 1: Enabled.
0	rx_ov	R/W	0	**Target Mode Receive FIFO Overflow Interrupt Enable** 0: Disabled. 1: Enabled.

Table 19-12: I²C FIFO Length Register

I²C FIFO Length			I2Cn_FIFOLEN	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:8	tx_depth	RO	8	**Transmit FIFO Length** This field returns the depth of the transmit FIFO. 8: 8-bytes.
7:0	rx_depth	RO	8	**Receive FIFO Length** This field returns the depth of the receive FIFO. 8: 8-bytes.

Table 19-13: I²C Receive Control 0 Register

I²C Receive Control 0			I2Cn_RXCTRL0	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**

I²C Receive Control 0			I2Cn_RXCTRL0	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
11:8	thd_lvl	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_INTFL0.rx_thd* 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	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 the hardware when the receive FIFO flush completes. Writing 0 has no effect. 0: Receive FIFO flush complete or not active. 1: Flush the receive FIFO
6:1	-	RO	0	**Reserved**
0	dnr	R/W	0	**Target Mode Do Not Respond** Target 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 always respond to data bytes with a NACK. 1: NACK the address.

Table 19-14: I²C Receive Control 1 Register

I²C Receive Control 1			I2Cn_RXCTRL1	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	lvl	R	0	**Receive FIFO Byte Count Status** This field returns the number of bytes in the receive FIFO. 0: 0 bytes (No data). 1: 1 byte. 2: 2 bytes. 3: 3 bytes. 4: 4 bytes. 5: 5 bytes. 6: 6 bytes. 7: 7 bytes. 8: 8 bytes.
7:0	cnt	R/W	1	**Receive FIFO Transaction Byte Count Configuration** In controller 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.irxm_en = 1. To receive more than 256 bytes, use I2Cn_CTRL.irxm_en = 1*

Table 19-15: I²C Transmit Control 0 Register

I²C Transmit Control 0			I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	thd_val	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_INTFL0.tx_thd* 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	flush	R/W1O	0	**Transmit FIFO Flush** A transmit FIFO flush clears all remaining data from the transmit FIFO. 0: Transmit FIFO flush is complete or not active. 1: Flush the transmit FIFO. *Note: The hardware automatically clears this bit to 0 after it is written to 1 when the flush is completed.* If *I2Cn_INTFL0.tx_lockout* = 1, then *I2Cn_TXCTRL0.flush* = 1.
6	-	RO	0	**Reserved**
5	nack_flush_dis	R/W	0	**Transmit FIFO received NACK Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Received NACK at the end of a target transmit operation enabled. 1: Received NACK at the end of a target transmit operation disabled. *Note: Upon entering transmit preload mode, the hardware automatically sets this bit to 0. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 0).*
4	rd_addr_flush_dis	R/W	0	**Transmit FIFO Target Address Match Read Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled. 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bitfield to 1).*
3	wr_addr_flush_dis	R/W	0	**Transmit FIFO Target Address Match Write Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 1).*

I²C Transmit Control 0			I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
2	gc_addr_flush_dis	R/W	0	**Transmit FIFO General Call Address Match Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled. 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 1).*
1	tx_ready_mode	R/W	0	**Transmit FIFO Ready Manual Mode** 0: The hardware controls *I2Cn_TXCTRL1.preload_rdy*. 1: Software control of *I2Cn_TXCTRL1.preload_rdy*.
0	preload_mode	R/W	0	**Transmit FIFO Preload Mode Enable** 0: Normal operation. An address match in target mode, or a general call address match, flushes and locks the transmit FIFO so it cannot be written and sets *I2Cn_INTFL0.tx_lockout*. 1: Transmit FIFO preload mode. An address match in target mode, or a general call address match, does not lock the transmit FIFO and does not set *I2Cn_INTFL0.tx_lockout*. This allows the software to preload data into the transmit FIFO. The status of the I²C is controllable at *I2Cn_TXCTRL1.preload_rdy*.

Table 19-16: I²C Transmit Control 1 Register

I²C Transmit Control Register 1			I2Cn_TXCTRL1	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	lvl	R	0	**Transmit FIFO Byte Count Status** 0: 0 bytes (No data). 1: 1 byte. 2: 2 bytes. 3: 3 bytes. 4: 4 bytes. 5: 5 bytes. 6: 6 bytes. 7: 7 bytes. 8: 8 bytes (max value).
7:1	-	RO	0	**Reserved**
0	preload_rdy	R/W1O	1	**Transmit FIFO Preload Ready Status** When transmit FIFO preload mode is enabled, *I2Cn_TXCTRL0.preload_mode* = 1, this bit is automatically cleared to 0. While this bit is 0, if the I2Cn hardware receives a target address match, a NACK is sent. Once the I2Cn hardware is ready (the software has preloaded the transmit FIFO, configured the DMA, etc.), the software must set this bit to 1, so the I2Cn hardware sends an ACK on a target address match. When transmit FIFO preload mode is disabled, *I2Cn_TXCTRL0.preload_mode* = 1, this bit is forced to 1, and the I2Cn hardware behaves normally.

Table 19-17: I²C Data Register

I²C Data			I2Cn_FIFO	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	data	R/W	0xFF	**FIFO Data** Reads from this register pop data off the receive FIFO. Writes to this register push data onto the transmit FIFO. Reading from an empty receive FIFO returns 0xFF. Writes to a full transmit FIFO are ignored.

Table 19-18: I²C Controller Control Register

I²C Controller Control			I2Cn_MSTCTRL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved**
10:8	-	RO	0	**Reserved**
7	ex_addr_en	R/W	0	**Target Extended Addressing Enable** 0: Send a 7-bit address to the target. 1: Send a 10-bit address to the target.
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 the hardware when the STOP condition begins.*
1	restart	R/W1O	0	**Send Repeated START Condition** After sending data to a target, the controller can send another START to retain control of the bus. 1: Send a repeated START condition to the target instead of sending a STOP condition at the end of the current transaction. *Note: This bit is automatically cleared by the hardware when the repeated START condition begins.*
0	start	R/W1O	0	**Start Controller Mode Transfer** 1: Start controller mode transfer. *Note: This bit is automatically cleared by the hardware when the transfer is completed or aborted.*

Table 19-19: I²C SCL Low Control Register

I²C Clock Low Control			I2Cn_CLKLO	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	lo	R/W	1	**Clock Low Time** In controller mode, this configures the SCL low time. $t_{SCL\_LO} = f_{I2C\_CLK} \times (lo + 1)$ *Note: 0 is not a valid setting for this field.*

Table 19-20: I²C SCL High Control Register

I²C Clock High Control			I2Cn_CLKHI	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	hi	R/W	1	**Clock High Time** In controller mode, this configures the SCL high time. $t_{SCL\_HI} = {}^{1}/_{f_{I2C\_CLK}} \times (hi + 1)$ In both controller and target 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_INTFL0.irxm* during IRXM. *Note: 0 is not a valid setting for this field.*

Table 19-21: I²C Hs-Mode Clock Control Register

I²C Hs-Mode Clock Control			I2Cn_HSCLK	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**

I²C Hs-Mode Clock Control			I2Cn_HSCLK	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
15:8	hi	R/W	0	**Hs-Mode Clock High Time** This field sets the Hs-Mode clock high count. In target mode, this is the time SCL is held high after data is output on SDA. *Note: See SCL Clock Generation for Hs-Mode for details on the requirements for the Hs-Mode clock high and low times.*
7:0	lo	R/W	0	**Hs-Mode Clock Low Time** This field sets the Hs-Mode clock low count. In target mode, this is the time SCL is held low after data is output on SDA. *Note: See SCL Clock Generation for Hs-Mode for details on the requirements for the Hs-Mode clock high and low times.*

Table 19-22: I²C Timeout Register

I²C Timeout			I2Cn_TIMEOUT	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	scl_to_val	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_INTFL0.to_err* = 1), and the peripheral releases the SCL and SDA lines. 0: Timeout disabled. All other values result in a timeout calculation of: $t_{BUS\_TIMEOUT} = {}^{1}/_{f_{I2C\_CLK}} \times scl\_to\_val$ *Note: The timeout counter monitors the I2Cn peripheral's driving of the SCL pin, not an external I²C device driving the SCL pin.*

Table 19-23: I²C DMA Register

I²C DMA			I2Cn_DMA	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	rx_en	R/W	0	**Receive DMA Channel Enable** 0: Disabled. 1: Enabled.
0	tx_en	R/W	0	**Transmit DMA Channel Enable** 0: Disabled. 1: Enabled.

Table 19-24: I²C Target Address 0 Register

I²C Target Address			I2Cn_SLAVE0	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ext_addr_en	R/W	0	**Target Mode Extended Address Length Select** 0: 7-bit addressing. 1: 10-bit addressing.
14:11	-	RO	0	**Reserved**
10	dis	R/W	0	**Target Address 0 Disable** Setting this field disables this register's target address.

I²C Target Address			I2Cn_SLAVE0	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
9:0	addr	R/W	0	**Target Mode Target Address** In target mode operation, (*I2Cn_CTRL.mst_mode* = 0), set this field to the target address for the I²C 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 bits, and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE0.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

Table 19-25: I²C Target Address 1 Register

I²C Target Address 1			I2Cn_SLAVE1	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ext_addr_en	R/W	0	**Target Mode Extended Address Length Select** 0: 7-bit addressing. 1: 10-bit addressing.
14:11	-	RO	0	**Reserved**
10	dis	R/W	0	**Target Address Disable** Setting this field to 1 disables this register's target address.
9:0	addr	R/W	0	**Target Mode Target Address** In target mode operation, (*I2Cn_CTRL.mst_mode* = 0), set this field to the target address for the I²C 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 bits, and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE0.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

Table 19-26: I²C Target Address 2 Register

I²C Target Address 2			I2Cn_SLAVE2	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ext_addr_en	R/W	0	**Target Mode Extended Address Length Select** 0: 7-bit addressing. 1: 10-bit addressing.
14:11	-	RO	0	**Reserved**
10	dis	R/W	0	**Target Address Disable** Setting this field to 1 disables this register's target address.
9:0	addr	R/W	0	**Target Mode Target Address** In target mode operation, (*I2Cn_CTRL.mst_mode* = 0), set this field to the target address for the I²C 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 bits, and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE2.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

Table 19-27: I²C Target Address 3 Register

I²C Target Address 3			I2Cn_SLAVE3	[0x0058]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ext_addr_en	R/W	0	**Target Mode Extended Address Length Select** 0: 7-bit addressing. 1: 10-bit addressing.
14:11	-	RO	0	**Reserved**
10	dis	R/W	0	**Target Address Disable** Setting this field to 1 disables this registers target address.

I²C Target Address 3			I2Cn_SLAVE3	[0x0058]
Bits	**Field**	**Access**	**Reset**	**Description**
9:0	addr	R/W	0	**Target Mode Target Address** In target mode operation, (*I2Cn_CTRL.mst_mode* = 0), set this field to the target address for the I²C 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 bits, and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE3.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

20. Inter-Integrated Sound Interface (I^2S)

I^2S is a serial audio interface for communicating pulse-code modulation (PCM) encoded streams between devices. The peripheral supports both controller and target modes.

Key features:

- Stereo (2 channel) and mono (left or right channel option) formats.
- Separate DMA channels for transmit and receive.
- Flexible timing
  - Configurable sampling rate from $1/65536$ to 1 of the I^2S input clock.
- Flexible data format
  - The number of bits per data word can be selected from 1 to 32, typically 8-, 16-, 24-, or 32-bit width.
  - Feature enhancement not in the I^2S specification:
    - Word/Channel select polarity control.
    - First bit position selection.
    - Selectable FIFO data alignment to the MSB or the LSB of the sample.
    - Sample size less than the word size with adjustment to MSB or LSB of the word.
    - Optional sign extension.
- Full-duplex serial communication with separate I^2S serial data input and serial data output pins.

20.1 Instances

Table 20-1: MAX32690 I^2S Instances

Instance	Supported Channels	I2S_CLK Clock Options		Receive FIFO Depth	Transmit FIFO Depth
I2S	Stereo	I2S0B_CLKEXT	PCLK	8 × 32 bits	8 × 32 bits

Note: I2S0B_CLKEXT (P0.23 AF2) must be enabled for controller operation; in target operation, external clocking is used for the LRCLK and BCLK input pins.

20.1.1 I^2S Bus Lines and Definitions

The I^2S peripheral includes support for the following signals:

1. Bit clock line
   - Continuous serial clock (SCK), referred to as bit clock (BCLK) in this document.
2. Word clock line
   - Word select (WS) referred to as left right clock (LRCLK) in this document.
3. Serial data input (SDI)
4. Serial data output (SDO)
5. ERFO is required for operation in controller mode and must be enabled.

Detailed pin and alternate function mapping are shown in *Table 20-2*.

Table 20-2: MAX32690 I²S Pin Mapping

I²S Signal	Pin Description	Alternate Function Name (y = A, B, or C)*	Notes
BCLK (SCK)	I²S bit clock	I2S0y_SCK	Also referred to as serial clock
LRCLK (WS)	I²S left/right clock	I2S0y_WS	Also referred to as word select
SDI	I²S serial data input	I2S0y_SDI	
SDO	I²S serial data output	I2S0y_SDO	
* Refer to the device's data sheet pin description table for alternate function mapping to pin numbers.			

20.2 Details

The I²S supports full-duplex serial communication with separate SDI and SDO pins. *Figure 20-1* shows an interconnect between a peripheral configured in controller mode, communicating with an external I²S target and an external I²S controller. In controller mode, the peripheral hardware generates the BCLK and LRCLK, and each is output to each target device.

Note: Controller operation requires the use of the P0.23 alternate function 1 (I2S0B_CLKEXT) to generate the LRCLK and BCLK signals.

Figure 20-1: I²S Controller Mode

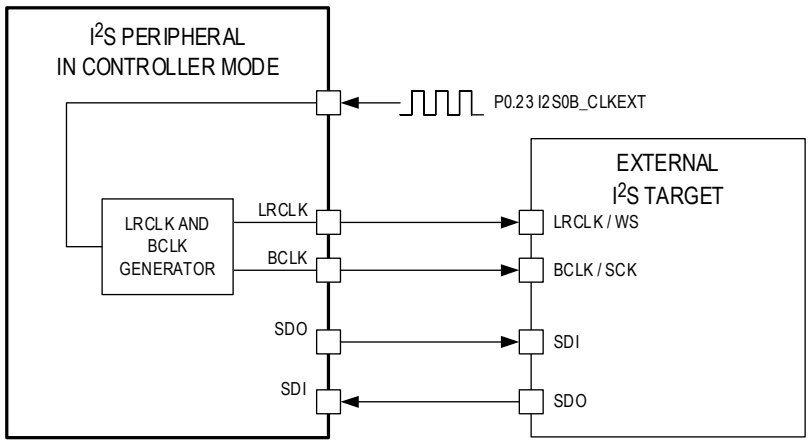
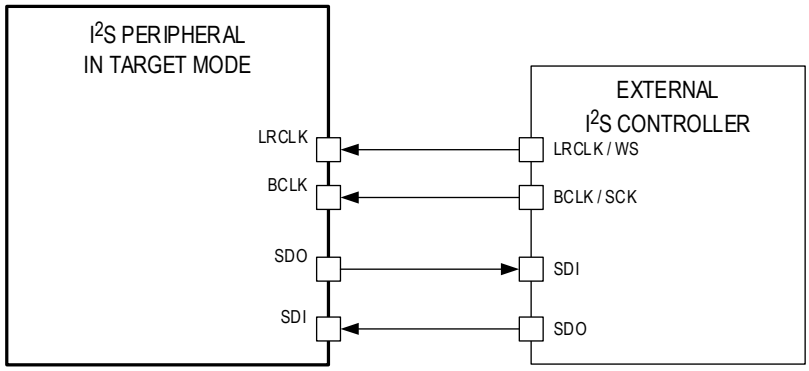


Figure 20-2 shows the I²S peripheral configured for target operation. The LRCLK and BCLK signals are generated externally by the controller and are inputs to the I²S peripheral.

Figure 20-2: I²S Target Mode



20.3 Controller and Target Mode Configuration

The device supports controller and target modes. In controller mode, the BCLK and LRCLK signals are generated internally and output on the BCLK and LRCLK pins. In target mode, the BCLK and LRCLK pins are configured as inputs, and the external controller's clock source controls the peripheral timing.

Table 20-3: I²S Mode Configuration

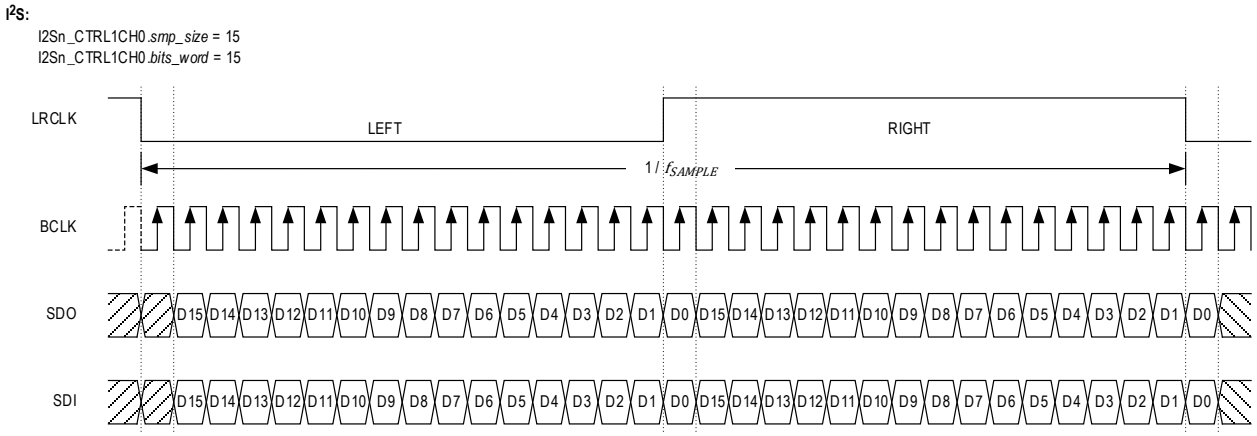
Device Mode	*I2S_CTRL0CH0.ch_mode*	LRCLK	BCLK
Controller	0	Output to target	Output to target
Target	3	Input from controller	Input from controller

20.4 Peripheral Clock Enable

The I²S is disabled by default on a reset. Use of the I²S peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS0.i2s* to 0 to enable the peripheral clock to the I²S before configuring the I²S for use.

20.5 Clocking

Figure 20-3: Audio Interface I²S Signal Diagram



I²S communication is synchronized using two signals, the LRCLK and the BCLK. When the I²S peripheral is configured as a controller, the BCLK and LRCLK signals are generated internally by the peripheral using pin P0.23 alternate function 1, I2S0B_CLKEXT. See *Table 20-2* for details of the I²S pin mapping and alternate function selection. If using the I²S peripheral in controller mode, the I2S0B_CLKEXT pin is used to generate the BCLK and LRCLK signals.

When the I²S peripheral is configured in target mode, the BCLK and LRCLK pins must be configured as inputs. An external controller generates the BCLK and LRCLK signals, which the peripheral uses to synchronize itself to the I²S bus. *Figure 20-3* shows the default I²S signals and timing for I²S communication.

The BCLK frequency is the product of the sample rate, the number of bits per channel (left and right), and the number of channels. For CD audio sampled at a frequency of 44.1kHz, with 16-bit sample width and stereo audio (left and right), the bit clock frequency, f_{BCLK}, is 1.4112MHz as shown in *Equation 20-1*.

Equation 20-1: CD Audio Bit Frequency Calculation

$$f_{BCLK} = 44.1\ kHz \times 16 \times 2 = 1.4112 MHz$$

20.5.1 BCLK Generation for Controller Mode

As indicated by *Equation 20-1*, the requirements for determining the BCLK frequency are:

1. Audio sample frequency
2. Number of bits per sample, referred to as sample width

Equation 20-2 shows the formula to calculate the bit clock frequency for a given audio file using the above requirements.

Equation 20-2: Calculating the Bit Clock Frequency for Audio

$$f_{BCLK} = f_{SAMPLE} \times Sample\ Width \times 2$$

In controller mode, the I²S external clock input is used to generate the BCLK frequency. The I²S external clock is divided by the *I2S_CTRL1CH0.clkdiv* field to achieve the target BCLK frequency, as shown in *Equation 20-3*.

Equation 20-3: Controller Mode BCLK Generation Using the I²S External Clock

$$f_{BCLK} = \frac{f_{I2S0B\_CLKEXT}}{(I2Sn\_CTRL1CH0.clkdiv + 1) \times 2}$$

Use *Equation 20-4* to determine the I²S clock divider for a target BCLK frequency.

Equation 20-4: Controller Mode Clock Divisor Calculation for a Target Bit Clock Frequency

$$I2Sn\_CTRL1CH0.clkdiv = \frac{f_{I2S0B\_CLKEXT}}{2 \times f_{BCLK}} - 1$$

20.5.2 LRCLK Period Calculation

An I²S data stream can carry mono (either left or right channel) or stereo (left and right channel) data. The LRCLK signal indicates which channel is currently being sent, either left or right channel data, as shown in *Figure 20-3*. The LRCLK is a 50% duty cycle signal and is the same frequency as the audio sampling frequency, f_{SAMPLE}.

The I²S peripheral uses the bits per word field, *I2S_CTRL1CH0.bits_word*, to define the audio's sample width, equivalent to the number of bit clocks per channel. This value should be set to the sample width of the audio minus 1. For example, the software should set the *I2S_CTRL1CH0.bits_word* field to 15 for audio sampled using a 16-bit width.

Equation 20-5: Bits Per Word Calculation

$$I2Sn\_CTRL1CH0.bits\_word = Sample\ Width - 1$$

The LRCLK frequency, or word select frequency, is automatically generated by the I²S peripheral hardware when set to operate as a controller. The LRCLK frequency calculation is shown in *Equation 20-6*.

Equation 20-6: LRCLK Frequency Calculation

$$f_{LRCLK} = f_{BCLK} \times (I2Sn\_CTRL1CH0.bits\_word + 1)$$

20.6 Data Formatting

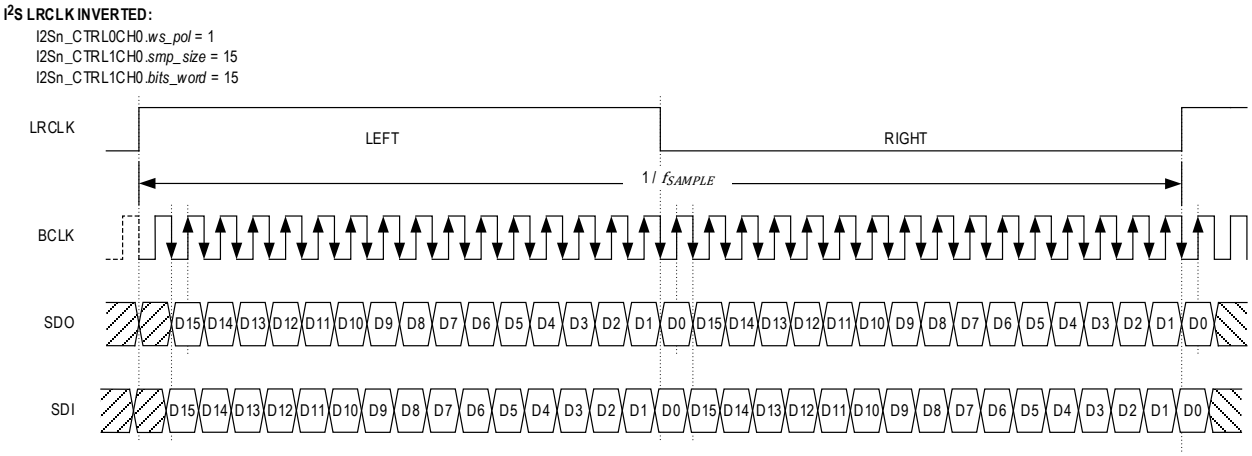
20.6.1 Sample Size

The sample size field, *I2S_CTRL1CH0.smp_word*, defines the number of desired samples within each channel, left, right or mono, for the peripheral. This field can be less than or equal to the *I2S_CTRL1CH0.bits_word* field. For example, for 16-bit sample width audio, the *I2S_CTRL1CH0.bits_word* field must be set to 15. However, the sample size field can be set from 0 to 15. Setting the sample size to 0 is equivalent to setting it to the value of the bits per word field. The sample size field determines how many of the bits per word are transmitted or saved per channel. The sample size field is a 0-based field; therefore, setting *I2S_CTRL1CH0.smp_word* to 15 collects 16 samples. See *Figure 20-6* for an example of the bits per word field's setting compared to the sample size field's setting.

20.6.2 Word Select Polarity

Left channel data, by default, is transferred when the LRCLK signal is low, and right channel data is transferred when the LRCLK signal is high. The polarity of the LRCLK is programmable, allowing left and right data to be swapped. The LRCLK polarity is controlled using the word select polarity field, *I2S_CTRL0CH0.ws_pol*. By default, LRCLK low is for the left channel, high is for the right channel as shown in *Figure 20-3*. Setting *I2S_CTRL0CH0.ws_pol* to 1 inverts the LRCLK polarity, using LRCLK high for the left channel and LRCLK low for the right channel as shown in *Figure 20-4*.

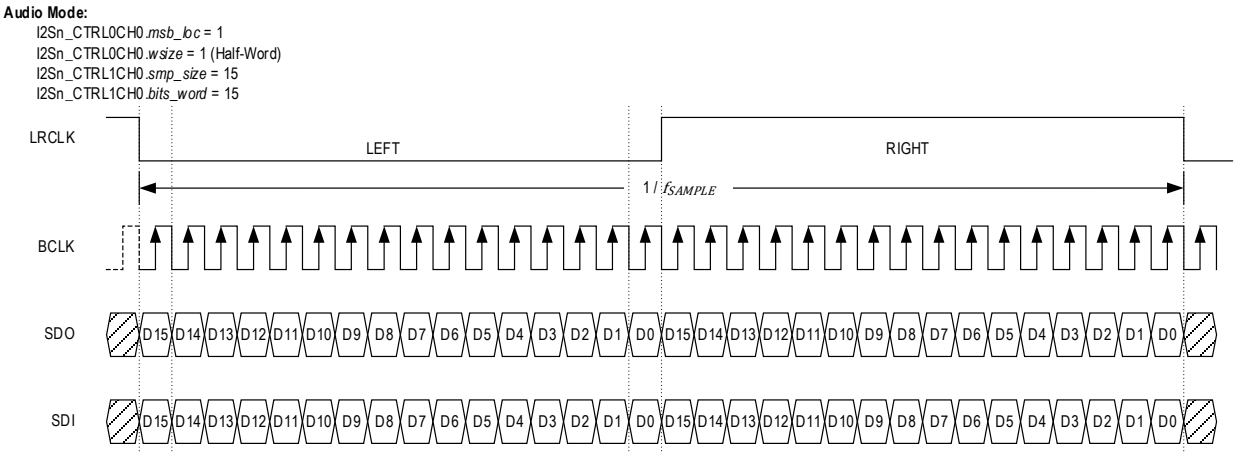
Figure 20-4: Audio Mode with Inverted Word Select Polarity



20.6.3 First Bit Location Control

The default setting is for the first bit of I²S data to be located at the second complete BCLK cycle after the LRCLK transition required by the I²S specification. See *Figure 20-3* for the standard data sampling configuration. Optionally, the first bit location can be left justified, resulting in the first bit of data being sampled on the first BCLK cycle after the LRCLK signal transitions as shown in *Figure 20-5*. Set *I2S_CTRL0CH0.msb_loc* to 1 to left justify the data with respect to the LRCLK.

Figure 20-5: Audio Controller Mode Left-Justified First Bit Location



20.6.4 Sample Adjustment

When the sample size field, *I2S_CTRL1CH0.smp_word*, is less than the bits per word field, *I2S_CTRL1CH0.bits_word*, use the *I2S_CTRL1CH0.adjst* field to set which bits are stored in the receive FIFO or transmitted from the transmit FIFO, either from the first sample of the SDI/SDO line or the last sample of the SDI/SDO line for the left and right channels. *Figure 20-6* shows an example of the default adjustment, MSB, where *I2S_CTRL1CH0.smp_word* = 7 and *I2S_CTRL1CH0.bits_word* = 15. *Figure 20-7* shows the adjustment set to the LSB of the SDI/SDO data.

Figure 20-6: MSB Adjustment when Sample Size is Less Than Bits Per Word

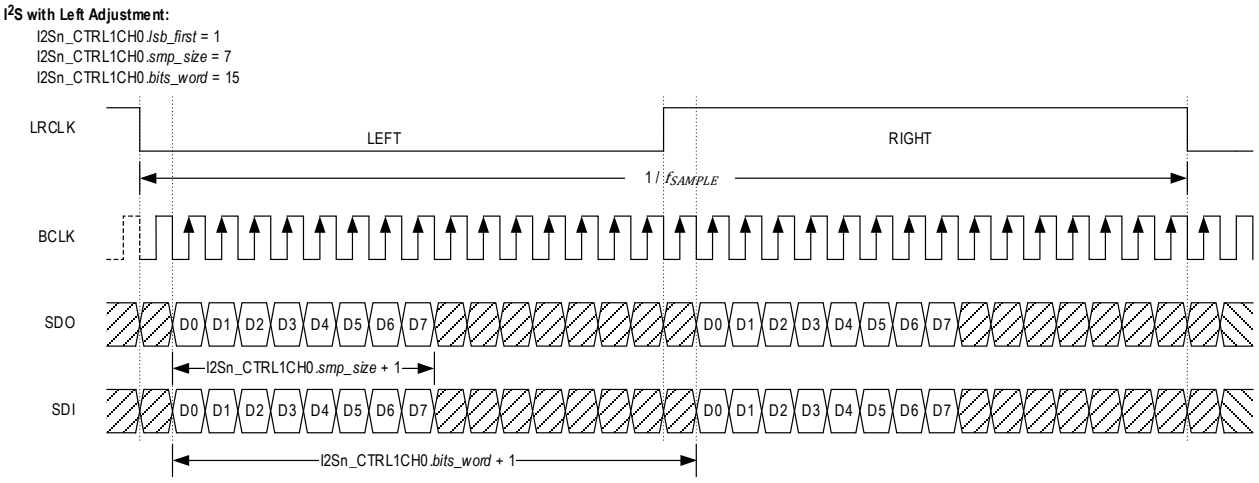
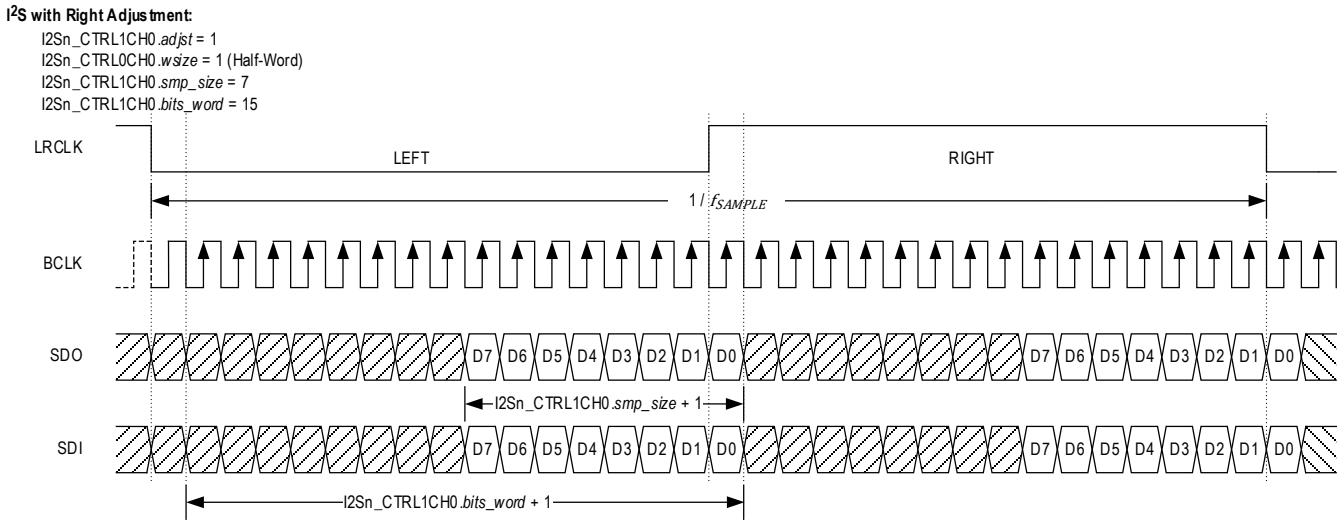


Figure 20-7: LSB Adjustment when Sample Size is Less Than Bits Per Word



20.6.5 Stereo/Mono Configuration

The I²S can transfer stereo or mono data based on the *I2S_CTRL0CH0.stereo* field. In stereo mode, both the left and right channels hold data. In mono mode, only the left or right channel contain data. For stereo mode, set *I2S_CTRL0CH0.stereo* to 0. Set the *I2S_CTRL0CH0.stereo* field to 2 for left channel mono. Set the *I2S_CTRL0CH0.stereo* field to 3 for right channel mono.

Figure 20-8: I²S Mono Left Mode

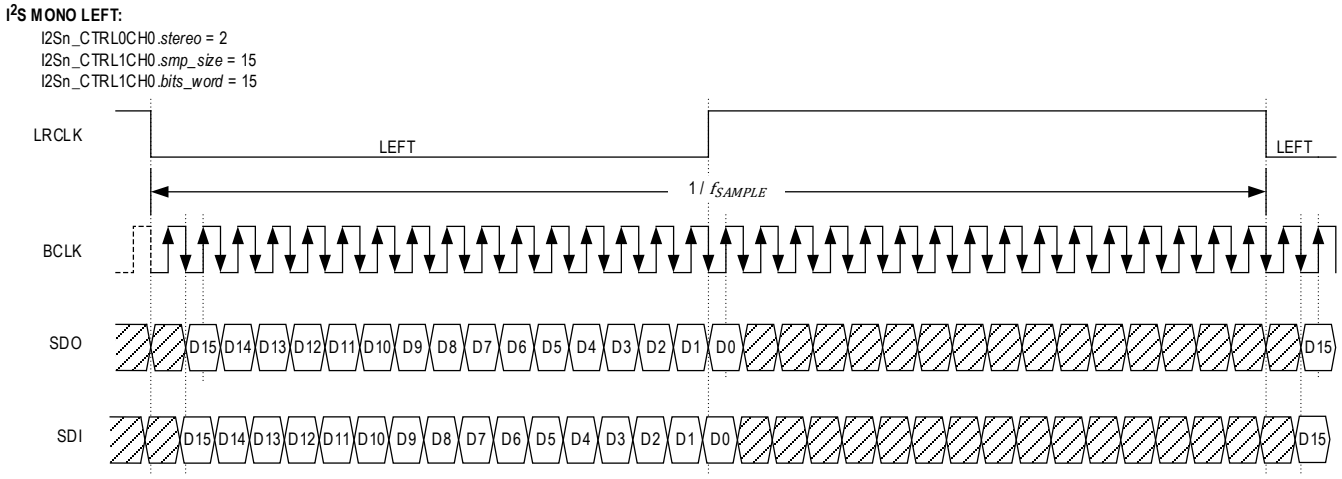
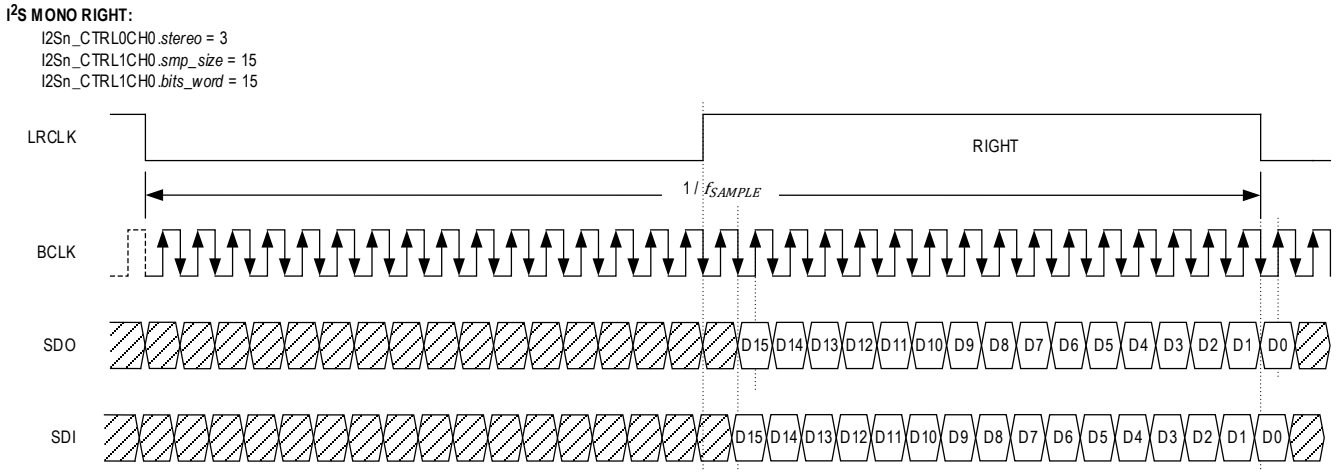


Figure 20-9: I²S Mono Right Mode



20.7 Transmit and Receive FIFOs

20.7.1 FIFO Data Width

I²S audio data is programmable from 1 to 32 bits using the *I2S_CTRL1CH0.bits_word* field. The software can set the FIFO width to either 8 bits (byte), 16 bits (half-word), or 32 bits (word). Set the FIFO width using the *I2S_CTRL0CH0.wsize* field. For FIFO word sizes less than 32 bits, the data frame, comprising a complete LRCLK cycle, can still be 64 bits; the unused bits are transmitted as zero by the hardware.

20.7.2 Transmit FIFO

An I²S transaction is started by writing data to the transmit FIFO using the *I2S_FIFOCH0.data* register, either directly or using a DMA channel. The data written is automatically transmitted out by the hardware, a FIFO word, as defined using the I2S_CTRL0CH0.*wsize* field, at a time, in the order it is written to the transmit FIFO. Use the I²S interrupt flags to monitor the transmit FIFO status and determine when the transfer cycle(s) are complete.

If the transmit FIFO becomes empty, an error condition occurs and results in undefined behavior.

20.7.3 Receive FIFO

The received data is loaded into the receive FIFO, and it can then be unloaded by reading from the *I2S_FIFOCH0.data* register. An overrun event occurs if the receive FIFO is full and another word is shifted into the FIFO.

20.7.4 FIFO Word Control

The data width of the transmit and receive FIFOs can be configured using the *I2S_CTRL0CH0.wsize* field. The following tables describe the data ordering based on the *I2S_CTRL0CH0.wsize* setting.

The transmit and receive FIFOs must be flushed, and the peripheral reset by the software before reconfiguration. The software resets the peripheral by setting the *I2S_CTRL0CH0.rst* field to 1.

Table 20-4: Data Ordering for Byte Data Size (Stereo Mode)

Byte Data Width (*I2S_CTRL0CH0*.*wsize* = 0)				
FIFO Entry	**MSByte**			**LSByte**
FIFO 0	Right Channel Byte 1	Left Channel Byte 1	Right Channel Byte 0	Left Channel Byte 0
FIFO 1	Right Channel Byte 3	Left Channel Byte 3	Right Channel Byte 2	Left Channel Byte 2
…	…	…	…	…
FIFO *7*	Right Channel Byte 14	Left Channel Byte 14	Right Channel Byte 13	Left Channel Byte 13

Table 20-5: Data Ordering for Half-Word Data Size (Stereo Mode)

Half-Word Data Width (*I2S_CTRL0CH0*.*wsize* = 1)		
FIFO Entry	**MS Half-Word**	**LS Half-Word**
FIFO 0	Right Channel Half-Word 0	Left Channel Half-Word 0
FIFO 1	Right Channel Half-Word 1	Left Channel Half-Word 1
…	…	…
FIFO *7*	Right Channel Half Word *7*	Left Channel Half-Word *7*

Table 20-6: Data Ordering for Word Data Size (Stereo Mode)

Word Data Width (*I2S_CTRL0CH0*.*wsize* = 2 or 3)	
FIFO Entry	**Word**
FIFO 0	Left Channel Word 0
FIFO 1	Right Channel Word 0
FIFO 2	Left Channel Word 1
FIFO 3	Right Channel Word 1
…	…
FIFO 6	Left Channel Word 3
FIFO 7	Right Channel Word 3

20.7.5 FIFO Data Alignment

The I^{2}S data can be left aligned or right aligned using the *I2S_CTRL0CH0*.*align* field. The following conditions apply to each setting:

Left aligned: *I2S_CTRL0CH0*.*align* = 0

- If the number of bits per word is greater than the FIFO data width:
  - Receive: All bits after the LSB of the FIFO data width are discarded.
  - Transmit: All bits after the LSB of the FIFO data width are sent as 0.
- If the number of bits per word is less than the FIFO data width:
  - Receive: The data received is stored starting at the MSB of the FIFO entry up to the number of bits per word plus one bit.
  - Transmit: The transmit FIFO data is sent from the LSB to the number of bits plus 1.

Right aligned: *I2S_CTRL0CH0.align* = 1

- If the number of bits per word is greater than the FIFO data width:
  - Receive: The data received is stored in the receive FIFO starting with the LSB up to the FIFO data width, and any additional bits are discarded.
  - Transmit: 0 bits are transmitted for all bits greater than the FIFO data width. For example, if the bits per word field is set to 12 and the FIFO data width is 8, the first 4 bits are transmitted as 0, the 8 bits of data in the FIFO are transmitted.
- If the number of bits per word is less than the FIFO data width:
  - Receive: The data received is sign extended and saved to the receive FIFO.
  - Transmit: The transmit FIFO data is sent from the LSB to the number of bits plus 1.

20.7.6 Typical Audio Configurations

Table 20-7 shows the relationship between the bits per word field and the sample size field. Equation 20-7 shows the required relationship between the sample size field and the bits per word field.

Equation 20-7: Sample Size Relationship Bits per Word

$$I2Sn\_CTRL1CH0.smp\_size \leq I2Sn\_CTRL1CH0.bits\_word$$

The *I2S_CTRL1CH0.bits_word* column in Table 20-7 is set using the equation $\frac{\#\ BCLK}{Channel} - 1$. The *I2S_CTRL1CH0.smp_size* column is the number of samples per word captured from the I²S bus and is calculated by the equation $\frac{\#\ Samples}{Channel} - 1$. Channel refers to the left and right channels of audio.

Table 20-7: Configuration for Typical Audio Width and Samples per WS Clock Cycle

Audio Sample Width/ Samples per WS Cycle	$\frac{\#\ BCLK}{Channel}$	$\frac{\#\ Samples}{Channel}$	I2S_CTRL1CH0			Sign extension (align = 1)†
			bits_word	smp_size	wsize	
8 bits / 16	8	8	7	7	0	
16 bits / 32	16	16	15	15	1	
20 bits / 40	20	20	19	19	2	sign
24 bits / 48	24	24	23	23	2	sign
24 bits / 64	32	24	31	23	2	sign
32 bits / 64	32	32	31	31	2	

† *Sign Extension only applies when I2S_CTRL0CH0.align is set to 1 and I2S_CTRL0CH0.smp_size is less than the FIFO width size setting.*

20.8 Interrupt Events

The I²S peripheral generates interrupts for the events shown in Table 20-8. An interrupt is generated if the corresponding interrupt enable field is set. The interrupt flags stay set until cleared by the software by writing 1 to the interrupt flag field.

Table 20-8: I²S Interrupt Events

Event	Interrupt Flag	Interrupt Enable
Receive FIFO overrun	*I2S_INTFL.rx_ov_ch0*	*I2S_INTEN.rx_ov_ch0*
Receive threshold	*I2S_INTFL.rx_thd_ch0*	*I2S_INTEN.rx_thd_ch0*
Transmit FIFO half-empty	*I2S_INTFL.tx_he_ch0*	*I2S_INTEN.tx_he_ch0*
Transmit FIFO one byte remaining	*I2S_INTFL.tx_ob_ch0*	*I2S_INTEN.tx_ob_ch0*

20.8.1 Receive FIFO Overrun

A receive FIFO overrun event occurs if the number of data words in the receive FIFO, *I2S_DMACH0.rx_lvl* is equal to the RX_FIFO_DEPTH, and another word is shifted into the FIFO. The hardware automatically sets the *I2S_INTFL.rx_ov_ch0* field to 1 when this event occurs.

20.8.2 Receive FIFO Threshold

A receive FIFO threshold event occurs when a word is shifted in and the number of words in the receive FIFO, *I2S_DMACH0.rx_lvl*, exceeds the *I2S_CTRL0CH0.rx_thd_val*. The event does not occur if the opposite transition occurs. When this event occurs, hardware automatically sets the *I2S_INTFL.rx_thd_ch0* field to 1.

20.8.3 Transmit FIFO Half-Empty

A transmit FIFO half-empty event occurs when the number of words in the transmit FIFO, *I2S_DMACH0.tx_lvl*, is less than ½ of the TX_FIFO_DEPTH as shown in *Equation 20-8*. When this event occurs, the *I2S_INTFL.tx_he_ch0* flag is set to 1 by hardware.

Note: The transmit FIFO half-empty interrupt flag is set by the hardware one BCLK cycle before the actual condition occurring. If the BCLK is much slower than the I²S peripheral clock, the software can receive the interrupt while the actual transmit FIFO level is still equal to ½ of the TX_FIFO_DEPTH. The software should always read the transmit FIFO level before filling it to determine the correct number of words to write to the transmit FIFO. Read the level of the transmit FIFO using the I2S_DMACH0.tx_lvl field.

Equation 20-8: Transmit FIFO Half-Empty Condition

$$I2S\_DMACH0.tx\_lvl < \left(\frac{TX\ FIFO\ DEPTH}{2}\right)$$

20.8.4 Transmit FIFO One Entry Remaining

A transmit FIFO one entry remaining event occurs when the number of entries in the transmit FIFO is 1, *I2S_DMACH0.tx_lvl* = 1. When this event occurs, the *I2S_INTFL.tx_ob_ch0* flag is set to 1 by the hardware.

Note: The transmit FIFO one entry remaining interrupt flag is set by the hardware one BCLK cycle before the actual condition occurring. If the BCLK is much slower than the I²S peripheral clock, the software can receive the interrupt while the actual transmit FIFO level is still equal to 2. The software should always read the transmit FIFO level before filling it to determine the correct number of words to write to the transmit FIFO. Read the level of the transmit FIFO using the I2S_DMACH0.tx_lvl field.

20.9 Direct Memory Access

The I²S supports DMA for both transmit and receive; separate DMA channels can be assigned to the receive and transmit FIFOs. The following describes the behavior of the receive and transmit DMA requests.

- A receive DMA request is asserted when the number of words in the receive FIFO is greater than or equal to the receive FIFO threshold.
- A transmit DMA request is asserted when the number of valid bytes in the transmit FIFO is less than ½ of the transmit FIFO's depth.

20.10 Block Operation

After exiting a power-on reset, the IP is disabled by default. It must be enabled and configured by the software to establish the I²S serial communication. A typical software sequence is shown below.

1. Set *GCR_PCLKDIS1.i2s0* to 0 to enable the I²S peripheral clock source shown in *Table 15-1*.
2. Disable the I²S clock by setting *I2S_CTRL1CH0.en* to 0.
3. Set *I2S_CTRL0CH0.rst* to 1 to reset the I²S configuration.
4. Set *I2S_CTRL1CH0.flush* to 1 to flush the FIFO buffers.
5. Configure the *I2S_CTRL0CH0.ch_mode* to select the controller or target configuration.
   a. For controller mode, configure the baud rate by programming the *I2S_CTRL1CH0.clkdiv* field to achieve the required bit rate, set the *I2S_CTRL1CH0.smp_size* field to the desired sample size of the data, and the *I2S_CTRL1CH0.adjst* field if the Sample Size is smaller than the number of bits per word.
6. Configure the threshold of the receive FIFO by programming the *I2S_CTRL1CH0.rx_thd*. The transmit FIFO threshold is a fixed value, which is half of the transmit FIFO depth.
7. If desired, configure DMA operation. See section *Direct Memory Access* for details.
8. Enable interrupt functionality by configuring the *I2S_INTEN* register if desired.
9. Program the *clkdiv* bits in the *I2S_CTRL1CH0* register for the new bit clock frequency.
10. For controller operation, load data in the transmit FIFO for transmit.
11. Re-enable the bit clock by setting *I2S_CTRL1CH0.en* to 1.

20.11 Registers

See *Table 3-2* for the base address of this peripheral. See *Table 1-1* 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 20-9: I²S Register Summary

Offset	Register Name	Description
[0x0000]	*I2S_CTRL0CH0*	I²S Global Mode Control 0 Register
[0x0010]	*I2S_CTRL1CH0*	I²S Controller Mode Configuration Register
[0x0030]	*I2S_DMACH0*	I²S DMA Control Channel Register
[0x0040]	*I2S_FIFOCH0*	I²S FIFO Register
[0x0050]	*I2S_INTFL*	I²S Interrupt Status Register
[0x0054]	*I2S_INTEN*	I²S Interrupt Enable Register

20.11.1 Register Details

Table 20-10: I²S Control 0 Register

I²S Control 0 Register			I2S_CTRL0CH0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	rx_thd_val	R/W	0	**Receive FIFO Interrupt Threshold** This field specifies the level of the receive FIFO for the threshold interrupt generation. Values of 0 or greater than the RX_FIFO_DEPTH are ignored.
23:21	-	RO	0	**Reserved**
20	fifo_lsb	R/W	0	**FIFO Bit Field Control** Only used if the FIFO size is larger than the sample size and *I2S_CTRL0CH0.align* = 0. For transmit, the LSB part is sent from the FIFO. For receive, store the LSB part in the FIFO without sign extension. 0: Disabled. 1: Enabled.

I²S Control 0 Register			I2S_CTRL0CH0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
19	rst	R/W1O	0	**Reset** Write 1 to reset the I²S peripheral. The hardware automatically clears this field to 0 when the reset is complete. 0: Reset not in process. 1: Reset peripheral.
18	flush	R/W1O	0	**FIFO Flush** Write 1 to start a flush of the receive FIFO and the transmit FIFO. The hardware automatically clears this field when the operation is complete. 0: Flush complete or not in process. 1: Flush receive and transmit FIFOs.
17	rx_en	R/W	0	**Receive Enable** Enable receive mode for the I²S peripheral. 0: Disabled. 1: Enabled.
16	tx_en	R/W	0	**Transmit Enable** Enable transmit mode for the I²S peripheral. 0: Disabled. 1: Enabled.
15:14	wsize	R/W	0x3	**Data Size When Reading/Writing FIFO** Set this field to the desired width for data writes and reads from the FIFO. 0: Byte. 1: Half-word (16 bits). 2-3: Word (32 bits).
13:12	stereo	R/W	0	**I²S Mode** Select the mode for the I²S to stereo, mono left channel only, or mono right channel only. 0-1: Stereo. 2: Mono left channel. 3: Mono right channel.
11	-	RO	0	**Reserved**
10	align	R/W	0	**FIFO Data Alignment** Set this field to control the alignment of the data in the FIFOs. This field is only used if the FIFO data width, *I2S_CTRL0CH0.wsize*, is not equal to the bits per word field. 0: MSB. 1: LSB.
9	msb_loc	R/W	0	**First Bit Location Sampling** This field controls when the first bit is transmitted/received in relation to the LRCLK. The first bit is transmitted/received on SDO/SDI on the second complete LRCLK cycle by default. Set this field to 1 to transmit/receive the first bit of data on the first complete LRCLK cycle. 0: Second complete LRCLK cycle is the first bit of the data. 1: First complete LRCLK cycle is the first bit of the data.
8	ws_pol	R/W	0	**LRCLK Polarity Select** This field determines the polarity of the LRCLK signal associated with the left channel data. Set this field to 1 to associate the left channel with the LRCLK high state. The default setting is the standard I²S association. 0: LRCLK low for the left channel. 1: LRCLK high for the left channel.

I²S Control 0 Register			I2S_CTRL0CH0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
7:6	ch_mode	R/W	0	**Mode** Set this field to indicate controller or target I²S operation. When using controller mode, the ERFO must be used to generate the LRCLK/BCLK signals. 0: Controller mode, internal generation of LRCLK/BCLK using the ERFO. 1-2: Reserved. 3: Target mode, external generation of LRCLK/BCLK.
5:2	-	DNM	0	**Reserved, Do Not Modify**
1	lsb_first	R/W	0	**LSB First** Setting this field to 1 indicates the least significant bit of the data is transmitted/received first on the SDI/SDO pins. The default setting, 0, indicates the most significant bit of the data is received first. 0: Disabled. 1: Enabled.
0	-	RO	0	**Reserved**

Table 20-11: I²S Controller Mode Configuration Register

I²S Controller Mode Configuration			I2S_CTRL1CH0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	clkdiv	R/W	0	**I²S Frequency Divisor** Set this field to the required divisor to achieve the desired frequency for the I²S BCLK. See *BCLK Generation for Controller Mode* for detailed information. *Note: This field only applies when the I²S peripheral is set to controller mode, I2S_CTRL0CH0.ch_mode = 0.*
15	adjust	R/W	0	**Data Justification When Sample Size is Less than Bits Per Word** This field is used to determine which bits are used if the sample size is less than the bits per word. 0: Left adjustment. 1: Right adjustment.
14	-	RO	0	**Reserved**
13:9	smp_size	R/W	0	**Sample Size** This field is the desired sample size of the data received or transmitted with respect to the Bits per Word field. In most use cases, the sample size is equal to the bits per word. However, in some situations, fewer bits are required by the application, which allows flexibility. An example use case would be for 16-bit audio being received, and the application only needs 8 bits of resolution. See *Sample Size* for additional details. *Note: The sample size is equal to I2S_CTRL1CH0.bits_word when I2S_CTRL1CH0.smp_size = 0 or I2S_CTRL1CH0.smp_size > I2S_CTRL1CH0.bits_word.*
8	en	R/W	0	**I²S Enable** For controller mode operation, this field is used to start generating the I²S LRCLK and BCLK outputs. In target mode, this field enables the peripheral to begin receiving signals on the I²S interface. 0: Disabled. 1: Enabled.
7:5	-	RO	0	**Reserved**
4:0	bits_word	R/W	0	**I²S Word Length** This field is defined as the I²S data bits per left and right channel. *Example: If the bit clocks is 16 per half frame, bits_word is 15.*

Table 20-12: I²S DMA Control Register

I²S DMA Control			I2S_DMACH0	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	rx_lvl	RO	0	**Receive FIFO Level** This field is the number of data words in the receive FIFO.
23:16	tx_lvl	RO	0	**Transmit FIFO Level** This field is the number of data words in the transmit FIFO.
15	dma_rx_en	R/W	0	**DMA Receive Channel Enable** 0: Disabled. 1: Enabled.
14:8	dma_rx_thd_val	R/W	0	**DMA Receive FIFO Event Threshold** If the receive FIFO level is greater than this value, then the receive FIFO DMA interface sends a signal to the system DMA indicating the receive FIFO has characters to transfer to memory.
7	dma_tx_en	R/W	0	**DMA Transmit Channel Enable** 0: Disabled. 1: Enabled.
6:0	dma_tx_thd_val	RO	0	**DMA Transmit FIFO Event Threshold** If the transmit FIFO level is less than this value, then the transmit FIFO DMA interface sends a signal to system DMA, indicating the transmit FIFO is ready to receive data from memory.

Table 20-13: I²S FIFO Register

I²S FIFO Register			I2S_FIFOCH0	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**I²S FIFO** Writing to this field loads the next character into the transmit FIFO and increments the *I2S_DMACH0.tx_lvl*. Writes are ignored if the transmit FIFO is full. Reads of this field return the next character available from the receive FIFO and decrement the *I2S_DMACH0.rx_lvl*. The value 0 is returned if *I2S_DMACH0.rx_lvl* = 0.

Table 20-14: I²S Interrupt Flag Register

I²S Interrupt Flag			I2S_INTFL	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	DNM	0	**Reserved, Do Not Modify**
3	tx_he_ch0	W1C	0	**Transmit FIFO Half-Empty Event Interrupt Flag** If this field is set to 1, the event has occurred. Write 1 to clear. 0: No event. 1: Event occurred.
2	tx_ob_ch0	W1C	0	**Transmit FIFO One Entry Remaining Event Interrupt Flag** If this field is set to 1, the event has occurred. Write 1 to clear. 0: No event. 1: Event occurred.
1	rx_thd_ch0	W1C	0	**Receive FIFO Threshold Event Interrupt Flag** If this field is set to 1, the event has occurred. Write 1 to clear. 0: No event. 1: Event occurred.
0	rx_ov_ch0	W1C	0	**Receive FIFO Overrun Event Interrupt Flag** If this field is set to 1, the event has occurred. Write 1 to clear. 0: No event. 1: Event occurred.

Table 20-15: I²S Interrupt Enable Register

I²S Interrupt Enable			I2S_INTEN	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	DNM	0	**Reserved, Do Not Modify**
3	tx_he_ch0	R/W	0	**Transmit FIFO Half-Empty Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.
2	tx_ob_ch0	R/W	0	**Transmit FIFO One Entry Remaining Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.
1	rx_thd_ch0	R/W	0	**Receive FIFO Threshold Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.
0	rx_ov_ch0	R/W	0	**Receive FIFO Overrun Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.

21. 1-Wire Master (OWM)

The device provides a 1-Wire master that software 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 manages 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.

21.1 1-Wire Master Features

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. The OWM module clock can be pre-scaled 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 software 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*
  - *www.maximintegrated.com/AN937*
- *AN1796: Guide to 1-Wire Communication*
  - *www.maximintegrated.com/AN1796*
- *AN187: 1-Wire Search Algorithm*
  - *www.maximintegrated.com/AN187*

iButton is a registered trademark of Analog Devices, Inc.

21.2 1-Wire Pins and Configuration

The one instance of the peripheral shown in *Table 21-1* lists the locations of the OWM_IO and OWM_PE signals for the OWM peripheral per package.

Table 21-1: MAX32690 1-Wire Master Peripheral Pins

Instance	Alternate Function Name	Alternate Function Number	140-WLP	64-TQFN
OWM	OWM_IO	AF1	P0.8, P1.30	P0.8
	OWM_PE	AF1	P0.7, P1.31	P0.7

21.2.1 1-Wire I/O (OWM_IO)

The OWM_IO pin is a bidirectional I/O that is used to directly drive the external 1-Wire bus. As described in the *Book of iButton Standards*, 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.

21.2.2 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.

21.2.3 Peripheral Clock Enable

The OWM is disabled by default on a reset. Use of the OWM peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS1.owm* to 0 to enable the peripheral clock to the OWM before configuring the OWM for use.

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

$$f_{owmclk} = 1MHz = \frac{f_{PCLK}}{OWM\_CLK\_DIV\_1US.divisor}$$

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).

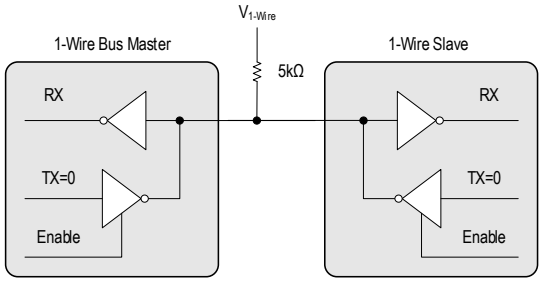
Network layers that apply to this description are the Physical, Link, Network, and Transport layers. The Transport layer consists of the software that transfers memory data other than ROM ID contents to and from the individual 1-Wire network nodes. The Presentation layer corresponds to higher-level application software functions (such as library layers) that implement communication protocols using the 1-Wire layers as a foundation. This document describes the details of the Physical, Link, and Network layers with regards to the OSI network layer model. The Transport and Presentation layers are beyond the scope of this document.

21.3.1.1 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 Link Layer

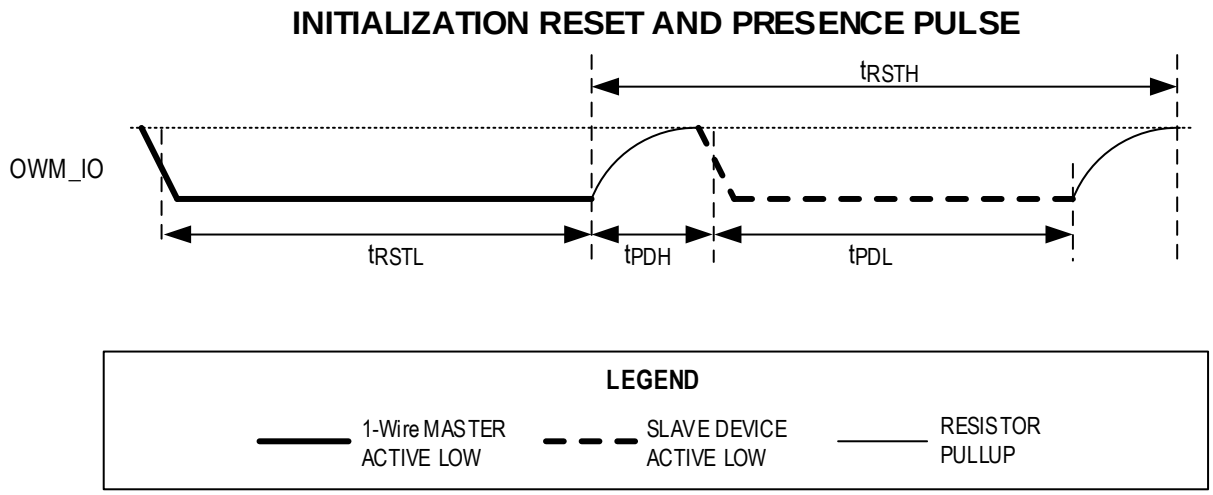
The 1-Wire bus supports a single master and one or more slave devices (multidrop). Slave devices can connect to and disconnect from the 1-Wire bus dynamically (as is typically the case with iButton devices that operate using an intermittent touch contact interface), which means that it is the master's 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, overdrive, and hyperdrive. However, only standard and overdrive are supported by the OWM peripheral in the devices.

21.3.1.2.1 OWM Reset and Presence Detect

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 not an 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.

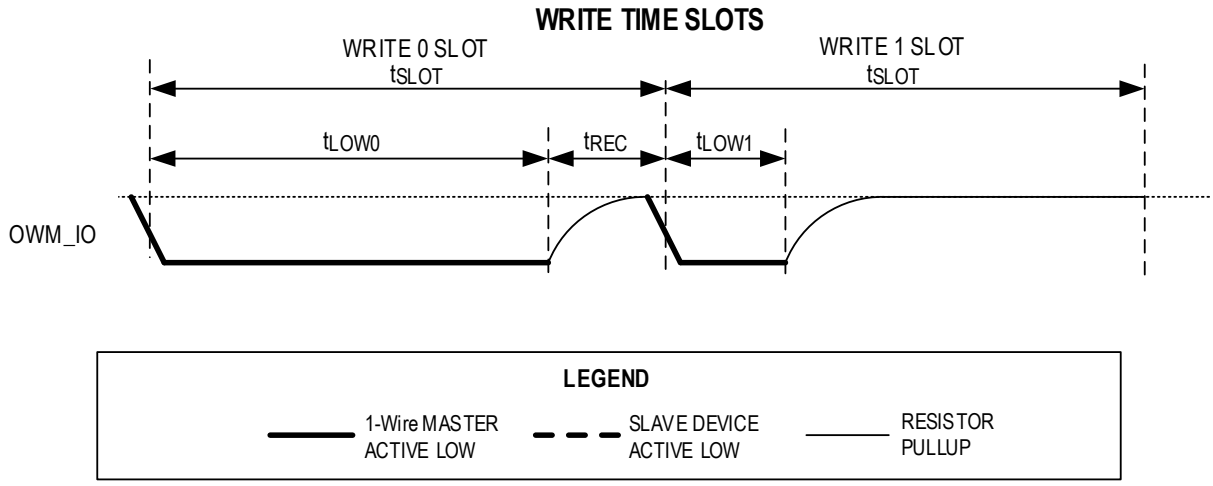
21.3.1.2.2 OWM Write Time Slot

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 if 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.

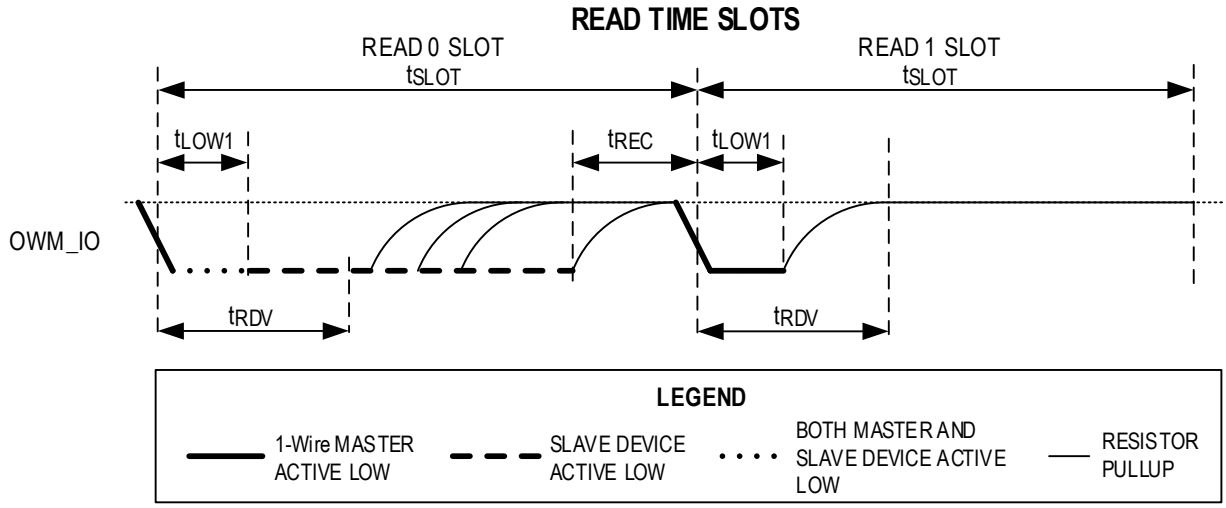
21.3.1.2.3 OWM Read Time Slot

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



21.3.1.2.4 Standard Speed and Overdrive Speed

By default, all 1-Wire communications following reset begin at the lowest rate of speed (i.e., 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 Network Layer

21.3.1.3.1 ROM Commands

Following the initial 1-Wire reset pulse on the bus, all slave 1-Wire devices are active, which means 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 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.

21.3.1.3.2 ROM ID

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

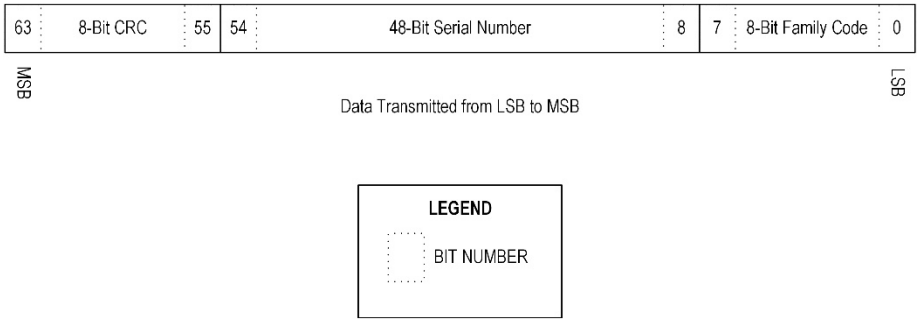


Table 21-3 provides a detailed description of each of the ROM ID fields.

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 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 following descriptions assume, however, that the master is communicating with only one slave device at a time because this is the method normally used.

As explained above, the ROM ID contents play a key 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 understood by all 1-Wire devices. After a device with overdrive capability is addressed and set into overdrive mode (i.e., 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 if 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 proceeds 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 is already determined. When transmitting this command, the master sends the command byte (i.e., 0x55 for standard speed and 0x69 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 selected by the command to shift communication speed from standard speed to overdrive speed. The Overdrive Match ROM command byte (0x69) 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 if 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 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 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 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 normally involves 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 clears 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 manages 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 the 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, 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 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_data_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 a 1 bit) or overrides by forcing the line low (to transmit a 0 bit).

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 0x0FF.
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_data_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 3-2* for the base address of this peripheral/module. See *Table 1-1* 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

21.6.1 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**
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.

OWM Configuration Register			OWM_CFG	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
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 (i.e., 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 is 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.
2	bit_bang_en	R/W	0	**Bit-Bang Mode Enable** Enable bit-bang control of the OWM_IO pin. If this bit is set to 1, *OWM_CTRL_STAT.bit_bang_oe* controls the state of the OWM_IO 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 a 1-Wire reset. 1: OWM_IO pin is driven low during presence detection portion of a 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.

Table 21-6: OWM Clock Divisor Register

OWM Clock Divisor			OWM_CLK_DIV_1US	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R	0	**Reserved**

OWM Clock Divisor			OWM_CLK_DIV_1US	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
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 *Peripheral Clock* Enable The OWM is disabled by default on a reset. Use of the OWM peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS1.owm* to 0 to enable the peripheral clock to the OWM before configuring the OWM for use. 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			OWM_CTRL_STAT	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
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**
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 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 Buffer Register

OWM Data			OWM_DATA	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	0	**Reserved**
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			OWM_INTFL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	R/W	0	**Reserved**
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			OWM_INTEN	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
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.

OWM Interrupt Enable			OWM_INTEN	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
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. Real-Time Clock (RTC)

22.1 Overview

The 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 32.768kHz 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 device data sheet for the required electrical characteristics of the external crystal.

A user-configurable, digital frequency trim is provided for applications requiring higher accuracy.

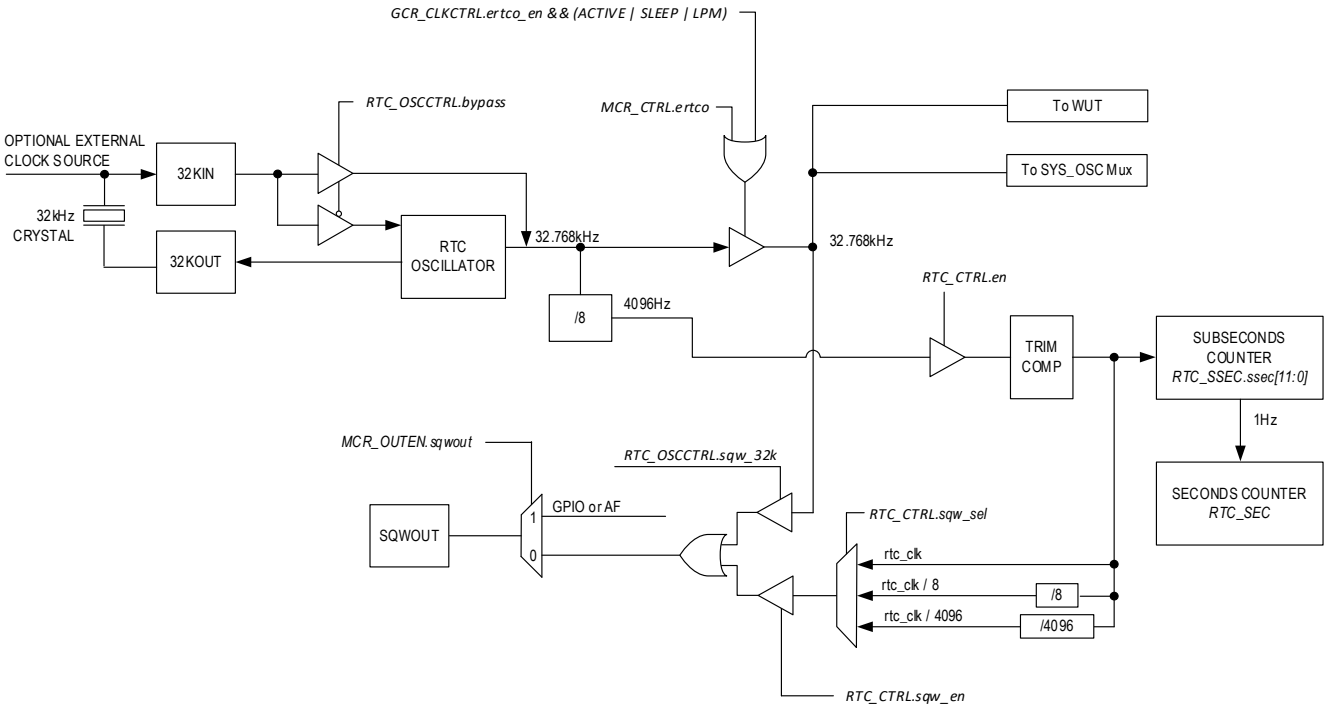
The 32-bit seconds counter register *RTC_SEC* is incremented every time there is a rollover of the *RTC_SSEC.ssec* sub-second counter field.

Two alarm functions are provided:

1. A programmable time-of-day alarm provides a single event, alarm timer using the *RTC_TODA* alarm register, *RTC_SEC* register, and *RTC_CTRL.tod_alarm_ie* field.
2. A programmable sub-second alarm provides a recurring alarm using the RTC sub-second alarm register, *RTC_SSECA*, and the *RTC_CTRL.ssec_alarm* field.

The RTC is powered in the AoD. Disabling the RTC, *RTC_CTRL.en* cleared to 0, stops incrementing the *RTC_SSEC* and *RTC_SEC*, but preserves their current values. The 32kHz oscillator is not affected by the *RTC_CTRL.en* field. While the RTC is enabled (*RTC_CTRL.en* = 1), the *RTC_TRIM.vrtc_tmr* field is also incremented every 32 seconds.

Figure 22-1: MAX32690 RTC Block Diagram



22.2 Instances

One instance of the RTC peripheral is provided. The RTC counter and alarm register details and description are shown in *Table 22-1*.

Note: See Enabling the ERTCO for details on enabling the ERTCO for use with the RTC.

Table 22-1: RTC Seconds, Sub-Seconds, Time-of-Day Alarm, and Sub-Second Alarm Register Details

Field	Width (bits)	Counter Increment	Minimum	Maximum	Description
RTC_SEC.sec	32	1 second	1 second	136 years	Seconds counter field
RTC_SSEC.ssec	12	244 µs ($\frac{1}{4096Hz}$)	244 µs	1 second	Sub-second counter field
RTC_TODA.tod_alarm	20	1 second	1 second	12 days	Time-of-day alarm field
RTC_SSECA.ssec_alarm	32	244 µs ($\frac{1}{4096Hz}$)	244 µs	12 days	Sub-second alarm field

22.3 Register Access Control

Access protection mechanisms prevent the software from accessing critical registers and fields while the hardware is updating them. Monitoring the *RTC_CTRL.busy* and *RTC_CTRL.rdy* fields allows the software to determine when it is safe to write to registers and when registers return valid results.

Table 22-2: RTC Register Access

Register	Field	Read Access	Write Access	*RTC_CTRL.busy* = 1 during writes	Description
RTC_SEC	*.sec*	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	Y	Seconds counter
RTC_SSEC	*.ssec*	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	Y	Sub-second counter
RTC_TODA	*.tod_alarm*	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.tod_alarm_ie* = 0	Y	Time-of-day alarm
RTC_SSECA	*.ssec_alarm*	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.ssec_alarm_ie* = 0	Y	Sub-second alarm
RTC_TRIM	All	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.wr_en* = 1	Y	Trim
RTC_OSCCTRL	All	Always	*RTC_CTRL.wr_en* = 1	N	Oscillator control
RTC_CTRL	*en*	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.wr_en* = 1	Y	RTC enable field
	All other fields	Always	*RTC_CTRL.busy* = 0	Y	

† See the *RTC_SEC and RTC_SSEC Read Access Control* section for details.

22.3.1 RTC_SEC and RTC_SSEC Read Access Control

The software reads of the *RTC_SEC* and *RTC_SSEC* registers return invalid results if the read operation occurs on the same cycle that the register is being updated by the hardware (*RTC_CTRL.rdy* = 0). The hardware avoids this by setting the *RTC_CTRL.rdy* field to 1 for 120µs when the *RTC_SEC* and *RTC_SSEC* registers are valid and readable by the software.

Alternately, the software can set the *RTC_CTRL.rd_en* field to 1 to allow asynchronous reads of both *RTC_SEC* and *RTC_SSEC*.

Three methods are available to ensure valid results when reading *RTC_SEC* and *RTC_SSEC*:

1. The software clears the *RTC_CTRL.rdy* field to 0.
   a. The software polls the *RTC_CTRL.rdy* field until it reads 1 before reading the registers.
   b. The software must read the *RTC_SEC* and *RTC_SSEC* registers within 120µs to ensure valid register data.
2. The software sets the *RTC_CTRL.rdy_ie* field to 1 to generate an RTC interrupt when the hardware sets the *RTC_CTRL.rdy* field to 1.
   a. The software must service the RTC interrupt and read the *RTC_SEC*, *RTC_SSEC*, or both registers while the *RTC_CTRL.rdy* field is 1 to ensure valid data, avoiding the overhead associated with polling the *RTC_CTRL.rdy* field.
3. The software sets the *RTC_CTRL.rd_en* field to 1 enabling asynchronous reads of both the *RTC_SEC* register and the *RTC_SSEC* register.
   a. The software must read consecutive identical values of each of the *RTC_SEC* register and the *RTC_SSEC* register to ensure valid data.

22.3.2 RTC Write Access Control

The read-only status field *RTC_CTRL.busy* is set to 1 by the hardware following a software instruction that writes to specific registers. The bit remains 1 while the software updates are being synchronized into the RTC. The software should not write to any of the registers until the hardware indicates the synchronization is complete by clearing *RTC_CTRL.busy* to 0.

22.4 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 alarm register's value. The sub-second interval alarm provides an auto-reload timer driven by the trimmed RTC clock source.

22.4.1 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 *RTC_SEC*[19:0] 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. Disable the time-of-day alarm (*RTC_CTRL.tod_alarm_ie* = 0) before changing the *RTC_TODA.tod_alarm* field.

When the alarm occurs, a single event sets the time-of-day alarm interrupt flag (*RTC_CTRL.tod_alarm*) to 1.

Setting the *RTC_CTRL.tod_alarm* bit to 1 in the software results in an interrupt request to the processor if the alarm time-of-day interrupt enable (*RTC_CTRL.tod_alarm_ie*) bit is set to 1, and the RTC's system interrupt enable is set.

22.4.2 Sub-Second Alarm

The *RTC_SSECA.ssec_alarm* and *RTC_CTRL.ssec_alarm_ie* fields control the sub-second alarm. Writing *RTC_SSECA.ssec_alarm* sets the starting value for the sub-second alarm counter. Writing the sub-second alarm enable (*RTC_CTRL.ssec_alarm_ie*) bit to 1 enables the sub-second alarm. Once enabled, an internal alarm counter begins incrementing from the *RTC_SSECA.ssec_alarm* field's value. When the counter rolls over from 0xFFFF FFFF to 0x0000 0000, the hardware sets the *RTC_CTRL.ssec_alarm* bit, triggering the alarm. At the same time, the hardware also reloads the counter with the value previously written to *RTC_SSECA.ssec_alarm*.

Disable the sub-second alarm, *RTC_CTRL.ssec_alarm_ie*, before changing the interval alarm value, *RTC_SSECA.ssec_alarm*.

The delay (uncertainty) associated with enabling the sub-second alarm is up to one sub-second clock period. This uncertainty is propagated to the first interval alarm. After that, 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.

22.4.3 RTC Interrupt and Wake-up Configuration

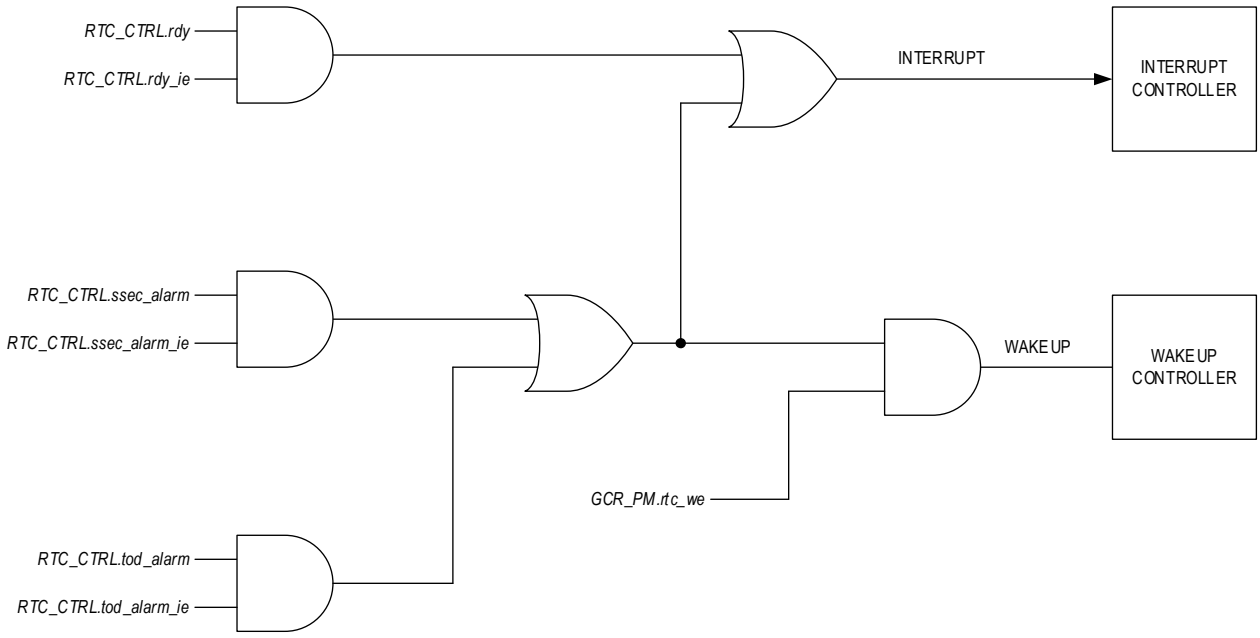
The following is a list of conditions that, when enabled, generate an RTC interrupt:

1. Time-of-day alarm
2. Sub-second alarm
3. *RTC_CTRL.rdy* field asserted high, signaling read access permitted

The RTC can be configured, so the time-of-day and sub-second alarms are a wake-up source for exiting the following low-power modes:

1. *SLEEP*
2. *LPM*
3. *UPM*
4. *STANDBY*
5. *BACKUP*

Figure 22-2: RTC Interrupt/Wake-up Diagram Wake-up Function



Use this procedure to enable the RTC as a wake-up source:

1. Configure the RTC interrupt enable bits, enabling one or more interrupt conditions to generate an RTC interrupt.
2. Create an RTC interrupt handler function and register the address of the RTC_IRQn using the NVIC.
3. Set the *GCR_PM.rtc_we* field to 1 to enable system wake-up by the RTC.
4. Enter the desired low-power mode. See *Operating Modes* for details.

22.5 Square Wave Output

The RTC can output a 50% duty cycle square wave signal derived from the 32kHz oscillator on a selected device pin. See *Table 22-3* for the device pins, frequency options, and control fields specific to this device. Frequencies noted as compensated in *Table 22-3* are used during the RTC frequency calibration procedure because they incorporate the frequency adjustments provided by the digital trim function.

Table 22-3: MAX32690 RTC Square Wave Output Configuration

Function	Option	Control Field
Output Pin	P4.0: SQWOUT	0
Enable Frequency Output	1Hz (Compensated)	*RTC_CTRL.sqw_sel* = 0 *RTC_CTRL.sqw_en* = 1 *RTC_OSCCTRL.sqw_32k* = 0
	512Hz (Compensated)	*RTC_CTRL.sqw_sel* = 1 *RTC_CTRL.sqw_en* = 1 *RTC_OSCCTRL.sqw_32k* = 0
	4kHz	*RTC_CTRL.sqw_sel* = 2 *RTC_CTRL.sqw_en* = 1 *RTC_OSCCTRL.sqw_32k* = 0
	32kHz	*RTC_OSCCTRL.sqw_32k* = 1

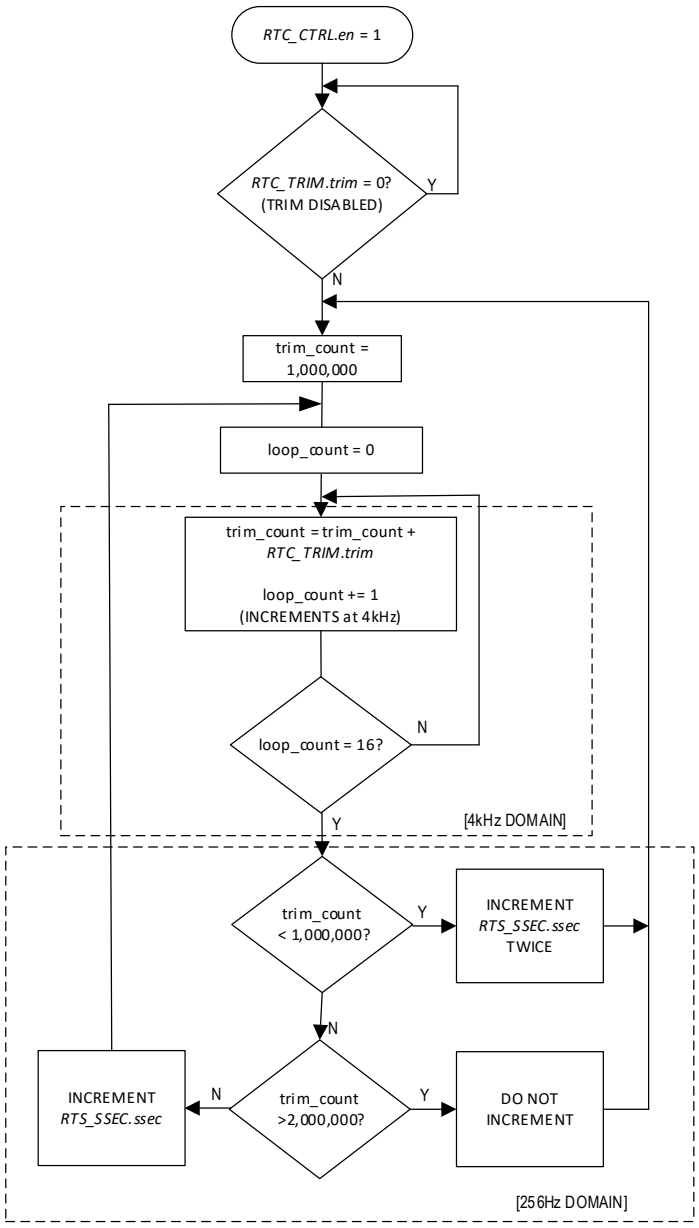
Use the following software procedure to generate and output the square wave:

1. Select the desired output frequency:
   a. Set the field *RTC_CTRL.sqw_sel* to 0 for a 1Hz compensated output frequency, or
   b. set the field *RTC_CTRL.sqw_sel* to 1 for a 512Hz compensated output frequency, or
   c. set the field *RTC_CTRL.sqw_sel* to 2 for a 4kHz output frequency, or
   d. set the field *RTC_OSCCTRL.sqw_32k* to 1 for the 32kHz frequency output.
2. Enable the system level output pin by setting the output pin shown in *Table 22-3*.
3. If the selected frequency is 1Hz, 512Hz, or 4kHz, set the *RTC_CTRL.sqw_en* field to 1 to output the selected output frequency.

22.6 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 that increments the sub-second register based on a user-supplied, 2s-complement value in the *RTC_TRIM* register as shown in *Figure 22-3*.

Figure 22-3: Internal Implementation of 4kHz Digital Trim



Complete the following steps to perform an RTC calibration:

1. The software must configure and enable 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. Clear the *RTC_CTRL*.*rdy* field to 0.
4. Wait for the *RTC_CTRL*.*rdy* to be set to 1 by the hardware:
   a. Set the *RTC_CTRL*.*rdy_ie* to 1 to generate an interrupt when the *RTC_CTRL*.*rdy* field is set to 1, or
   b. Poll the *RTC_CTRL*.*rdy* field until it reads 1.
5. Poll the *RTC_CTRL*.*busy* field until it reads 0 to allow any active operations to complete.
6. Set the *RTC_CTRL*.*wr_en* field to 1 to allow access to the *RTC_TRIM*.*trim* field.
7. Write a trim value to the *RTC_TRIM*.*trim* field to correct for any measured inaccuracy.
8. Poll the *RTC_CTRL*.*busy* field until it reads 0
9. Clear the *RTC_CTRL*.*wr_en* field to 0.
10. Repeat the process as needed until the desired accuracy is achieved.

22.7 Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 reset.

Table 22-4: 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

22.7.1 Register Details

Table 22-5: 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 22-6: RTC Sub-Second Counter Register

RTC Sub-Seconds Counter			RTC_SSEC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:0	ssec	R/W	0	**Sub-Seconds Counter** *RTC_SEC* increments when this field rolls from 0xFFF to 0x000.

Table 22-7: RTC Time-of-Day Alarm Register

RTC Time-of-Day Alarm			RTC_TODA	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:0	tod_alarm	R/W	0	**Time-of-Day Alarm** This field 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 22-8: 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 22-9: RTC Control Register

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
15	wr_en	R/W	0*	**Write Enable** This field controls access to the *RTC_TRIM* register and the RTC enable (*RTC_CTRL.en*) field. 1: Writes to the *RTC_TRIM* register and the *RTC_CTRL.en* field are allowed. 0: Writes to the *RTC_TRIM* register and the *RTC_CTRL.en* field are ignored. **Note: Reset on System Reset, Soft Reset, and GCR_RST0.rtc assertion.*
14	rd_en	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 require software interaction to ensure data accuracy.
13:11	-	RO	0	**Reserved**
10:9	sqw_sel	R/W	0*	**Frequency Output Select** This field selects the RTC-derived frequency to output on the square wave output pin. See *Table 22-3* for configuration details. 0: 1Hz (Compensated) 1: 512Hz (Compensated) 2: 4kHz 3: Reserved **Note: Reset on POR only.*
8	sqw_en	R/W	0*	**Square Wave Output Enable** This field enables the square wave output. See *Table 22-3* for configuration details. 0: Disabled. 1: Enabled. **Note: Reset on POR only.*
7	ssec_alarm	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 device. 0: No sub-second alarm pending. 1: Sub-second interrupt pending. **Note: Reset on POR only.*
6	tod_alarm	R/W	0*	**Time-of-Day Alarm Interrupt Flag** This interrupt flag is set by the 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	rdy_ie	R/W	0*	**RTC Ready Interrupt Enable** 0: Disabled. 1: Enabled. **Note: Reset on system reset, soft reset, and GCR_RST0.rtc assertion.*

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
4	rdy	R/W0	0*	**RTC Ready** This bit is set to 1 for 120µs by the hardware once a hardware update of the *RTC_SEC* and *RTC_SSEC* registers occurred. The software should read *RTC_SEC* and *RTC_SSEC* while this hardware bit is set to 1. The software can clear this bit at any time. An RTC interrupt is generated if *RTC_CTRL.rdy_ie* = 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_RST0.rtc *assertion.*
3	busy	RO	0*	**RTC Busy Flag** This field is set to 1 by the hardware while a register update is in progress. Software writes to the following registers result in this field being set to 1: • *RTC_SEC* • *RTC_SSEC* • *RTC_TRIM* The following fields cannot be written when this field is set to 1: • *RTC_CTRL.en* • *RTC_CTRL.tod_alarm_ie* • *RTC_CTRL.ssec_alarm_ie* • *RTC_CTRL.rdy_ie* • *RTC_CTRL.tod_alarm* • *RTC_CTRL.ssec_alarm* • *RTC_CTRL.sqw_en* • *RTC_CTRL.rd_en* This field is automatically cleared by the hardware when the update is complete. The software should poll this field until it reads 0 after changing the *RTC_SEC*, *RTC_SSEC*, or *RTC_TRIM* register before making any other RTC register modifications. 0: RTC not busy 1: RTC busy **Note: Reset on POR only.*
2	ssec_alarm_ie	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.*
1	tod_alarm_ie	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.*

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
0	en	R/W	0*	**Real-Time Clock Enable** The RTC write enable (*RTC_CTRL.wr_en*) 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 22-10: 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*	**V_{RTC} Time Counter** The hardware increments this field every 32 seconds while the RTC is enabled. **Note: Reset on POR only.*
7:0	trim	R/W	0*	**RTC Trim** This field specifies the 2s 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 22-11: RTC 32KHz Oscillator Control Register

RTC Oscillator Control			RTC_OSCCTRL	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	R/W	0	**Reserved**
5	sqw_32k	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 22-3* 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 drive the 32KIN pin. 0: Disable bypass. RTC time base is an external 32kHz crystal. 1: Enable bypass. RTC time base is an external square wave driven on 32KIN. **Note: Reset on POR only.*
3:0	-	DNM	9	**Reserved Do Not Modify**

23. Timers (TMR/LPTMR)

Multiple 32-bit and dual 16-bit reloadable timers are provided.

The features include:

- Operation as a single 32-bit counter or single/dual 16-bit counter(s).

Programmable clock prescaler with values from 1 to 4096.

- Non-overlapping pulse width modulated (PWM) output generation with configurable off-time.

Capture, compare, and capture/compare capability.

- Timer input and output signals available, mapped as alternate functions.

Configurable input pin for event triggering, clock gating, or capture signal.

- Timer output pin for event output and PWM signal generation.

Multiple clock source options.

Instances denoted as LPTMR, shown in *Table 23-1*, are configurable to operate in any of the low-power modes and wake the device from the low-power modes to *ACTIVE*.

Each timer supports multiple operating modes:

- One-shot: the timer counts up to terminal value then halts.

Continuous: the timer counts up to terminal value then repeats.

- Counter: the timer counts input edges received on the timer input pin.

PWM.

- Capture: the timer captures a snapshot of the current timer count when the timer's input edge transitions.

Compare: the timer pin toggles when the timer's count exceeds the terminal count.

- Gated: the timer increments only when the timer's input pin is asserted.

Capture/Compare: the timer counts when the timer input pin is asserted; the timer captures the timer's count when the input pin is deasserted.

23.1 Instances

Instances of the peripheral are listed in *Table 23-1*. Both the TMR and LPTMR are functionally similar, so for convenience all timers are referenced as TMR. The LPTMR instances can function while the device is in certain low-power modes.

Refer to the device data sheet for frequency limitations for external clock sources, if available. Refer to the data sheet for I/O signal configurations and alternate functions for each timer instance.

Table 23-1: MAX32690 TMR/LPTMR Instances

Instance	Register Access Name	Cascade 32-Bit Mode	16-Bit Mode	Operating Modes	CLK0	CLK1	CLK2	CLK3
TMR0	TMR0	Yes	Dual	*ACTIVE SLEEP LPM*	PCLK	ISO	IBRO	ERFO
TMR1	TMR1							
TMR2	TMR2							
TMR3	TMR3							
LPTMR0	TMR4	No	Single	*ACTIVE SLEEP LPM*	IBRO	ERTCO	INRO	LPTMR0_CLK P3.5 (AF1)
				UPM	N/A	N/A	ERTCO	INRO
LPTMR1	TMR5	No	Single	*ACTIVE SLEEP LPM*	IBRO	$\frac{IBRO}{8}$	INRO	LPTMR1_CLK P3.6 (AF1)
				UPM	N/A	N/A	ERTCO	INRO

Table 23-2: MAX32690 TMR/LPTMR Instances Capture Events

Instance	Capture Event 0	Capture Event 1	Capture Event 2	Capture Event 3
TMR0	Timer Input Pin	TMR0A_IOA	TMR0B_IOA	Software Event
TMR1	Timer Input Pin	TMR1A_IOA	TMR1B_IOA	Software Event
TMR2	-	-	-	-
TMR3	-	-	-	-
LPTMR0	LPTMR0B_IOA	LPCMP0 Interrupt	LPCMP1 Interrupt	-
LPTMR1	LPTMR1B_IOA	LPCMP0 Interrupt	LPCMP1 Interrupt	-

23.2 Basic Timer Operation

The timer modes operate by incrementing the *TMRn_CNT.count* register, driven by either the timer clock, an external stimulus on the timer pin, or a combination of both. The *TMRn_CNT.count* 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 the 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.count* with a new starting value, or disabling the counter. The end of a timer period always sets the corresponding interrupt bit and can generate an interrupt, if enabled.

In most modes, the timer peripheral automatically sets *TMRn_CNT.count* to 0x0000 0001 at the end of a timer period, but *TMRn_CNT.count* is set to 0x0000 0000 following a system reset. This means the first timer period following a system reset is one timer clock longer than subsequent timer periods if *TMRn_CNT.count* is not initialized to 0x0000 0001 during the timer configuration step.

23.2.1 Peripheral Clock Enable

Each timer is disabled by default on a reset. Use of a timer requires enabling the peripheral clock for the desired timer. Enable the peripheral clock to a given timer by setting the disable bit to 0. See *Table 17-2* for each timer's peripheral clock disable bit.

Table 23-3: Timer Peripheral Clock Disable Bits

Timer	Disable Bit
TMR0	*GCR_PCLKDIS0.tmr0*
TMR1	*GCR_PCLKDIS0.tmr1*
TMR2	*GCR_PCLKDIS0.tmr2*
TMR3	*GCR_PCLKDIS0.tmr3*
TMR4 (LPTMR0)	*LPGCR_PCLKDIS.tmr4*
TMR5 (LPTMR1)	*LPGCR_PCLKDIS.tmr5*

23.3 32-Bit Single / 32-Bit Cascade / Dual 16-Bit

Most instances contain two 16-bit timers, which may support combinations of single or cascaded 32-bit modes, and single or dual 16-bit modes as shown in *Table 23-1*. In most cases, the two 16-bit timers have the same functionality.

The terminology TimerA and TimerB are used to differentiate the organization of the 32-bit registers shown in *Table 23-4*. Most of the other registers have the same fields duplicated in the upper and lower 16 bits, and are differentiated with the _a and _b suffixes.

In the 32-bit modes, the fields and controls associated with TimerA are used to control the 32-bit timer functionality. In single 16-bit timer mode, the TimerA fields are used to control the single 16-bit timer and the TimerB fields are ignored. In dual 16-bit timer modes, both TimerA and TimerB fields are used to control the dual timers; TimerB fields control the upper 16-bit timer and TimerA fields control the lower 16-bit timer. In dual-16 bit timer modes, TimerB can be used as a single 16-bit timer.

Table 23-4: TimerA/TimerB 32-Bit Field Allocations

Register	Cascade 32-Bit Mode	Dual 16-Bit Mode		Single 16-Bit Mode
Timer Counter	TimerA Count = *TMRn_CNT.count[31:0]*	TimerA Compare = *TMRn_CNT.count[15:0]*	TimerB Count = *TMRn_CNT.count[31:16]*	TimerA Compare = *TMRn_CNT.count[15:0]*
Timer Compare	TimerA Compare = *TMRn_CMP.compare[31:0]*	TimerA Compare = *TMRn_CMP.compare[15:0]*	TimerB Compare = *TMRn_CMP.compare[31:16]*	TimerA Compare = *TMRn_CMP.compare[15:0]*
Timer PWM	TimerA Count = *TMRn_PWM.pwm[31:0]*	TimerA Count = *TMRn_PWM.pwm[15:0]*	TimerB Count = *TMRn_PWM.pwm[31:16]*	TimerA Count = *TMRn_PWM.pwm[15:0]*

23.4 Timer Clock Sources

Clocking of timer functions is driven by the timer clock frequency, f_{CNT_CLK}, which is a function of the selected clock source shown in *Table 23-1*. Most modes support multiple clock sources and prescaler values, which can be chosen independently for TimerA and TimerB when the peripheral is operating in dual 16-bit mode. The prescaler can be set from 1 to 4096 using the *TMRn_CNT.pres* field.

Note: The low-power timers must use the same clock selection for both TimerA and TimerB. Software must write both fields, TMRn_CTRL1.clksel_a and TMRn_CTRL1.clksel_b to the same value simultaneously to prevent a potential clock glitch.

Equation 23-1: Timer Peripheral Clock Equation

$$f_{CNT\_CLK} = \frac{f_{CLK\_SOURCE}}{prescaler}$$

The software configures and controls the timer by reading and writing to the timer's registers. External events on timer pins are asynchronous events to the timer's clock frequency. The external events are latched on the next rising edge of the timer's clock. Since it is not possible to externally synchronize to the timer's internal clock, input events may require up to 50% of the timer's internal clock cycle before the hardware recognizes the event.

The software must configure the timer's clock source by performing the following steps:

1. Disable the timer peripheral.
   a. Clear *TMRn_CTRL0.en* to 0 to disable the timer.
   b. Read the *TMRn_CTRL1.clken* field until it returns 0 confirming the timer peripheral is disabled.
2. Set *TMRn_CTRL1.clksel* to the new desired clock source.
   a. Note: In cascade 32-bit timer mode the *TMRn_CTRL1.clksel_b* field must be set to the same value as the *TMRn_CTRL1.clksel_a* field.
3. Configure the timer for the desired operating mode. See *Operating Modes* for details on mode configuration.
4. Enable the timer clock source:
   a. Set the *TMRn_CTRL0.clken* field to 1 to enable the timer's clock source.
   b. Read the *TMRn_CTRL1.clkrdy* field until it returns 1 confirming the timer clock source is enabled.
5. Enable the timer:
   a. Set *TMRn_CTRL0.en* to 1 to enable the timer.
   b. Read the *TMRn_CNT.clken* field until it returns 1 to confirm the timer is enabled.

The timer peripheral should be disabled while changing any of the registers in the peripheral.

23.5 Timer Pin Functionality

Each timer instance may have an input signal and/or output signal depending on the operating mode. Not all instances of the peripheral are available in all packages. The number of input and output signals per peripheral instance may vary as well. Refer to the device data sheet for I/O signal configurations and alternate functions for each timer instance.

The physical pin location of the timer input and/or output signals may vary between packages. The timer functionality, however, is always expressed on the same GPIO pin in the same alternate function mode.

The timer pin functionality is mapped as an alternate function that is shared with a GPIO. When the timer pin alternate function is enabled, the timer pin has the same electrical characteristics, such as pullup/pulldown strength, drive strength, etc., as the GPIO mode settings for that pin. The pin characteristics must be configured before enabling the timer. 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. Consult the GPIO section for details on how to configure the electrical characteristics for the pin.

The TimerA output controls for modes 0, 1, 3, and 5 output signals are shown in *Figure 23-1*. The TimerA input controls for modes 2, 4, 6, 7, 8, and 14 input signals are shown in *Figure 23-2*.

Figure 23-1: MAX32690 TimerA Output Functionality, Modes 0/1/3/5

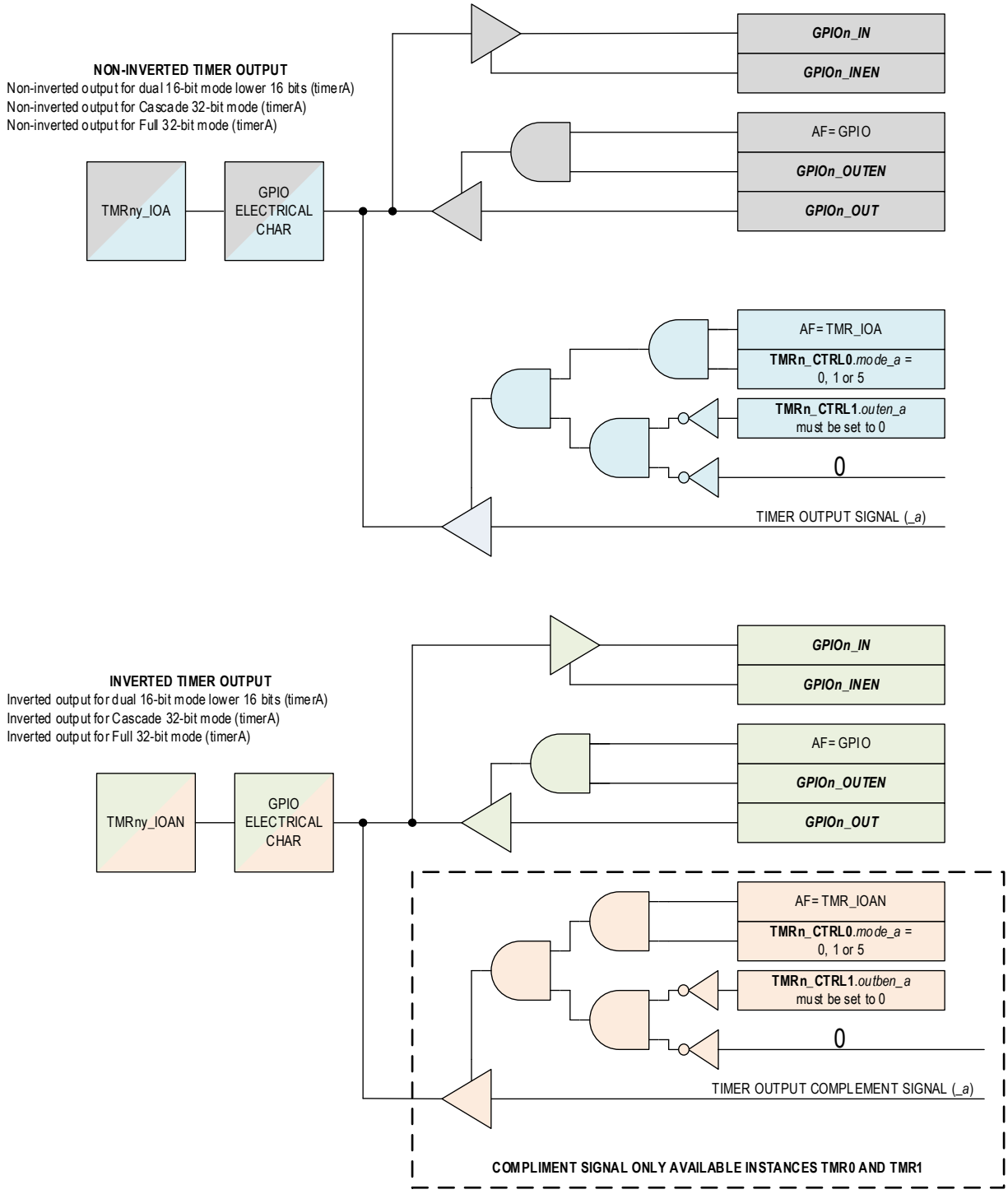
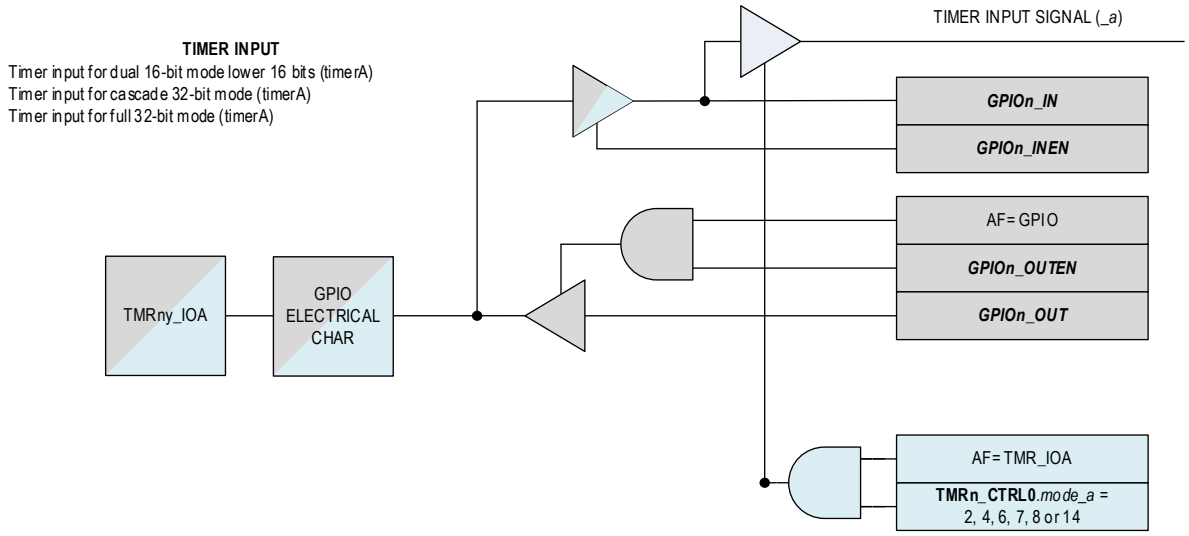


Figure 23-2: MAX32690 TimerA Input Functionality, Modes 2/4/6/7/8/14



23.6 Wake-up Events

In low-power modes, the system clock may be turned off to conserve power. LPTMR instances can continue to run from the clock sources shown in *Table 23-1*. In this case, a wake-up event can be configured to wake up the clock control logic and re-enable the system clock.

Programming sequence example:

1. Disable the timer peripheral and set the timer clock source as described in *Timer Clock Sources*.
2. Configure the timer operating mode as described in the section *Operating Modes*.
3. Enable the timer by setting *TMRn_CTRL0.en* to 1.
4. Poll *TMRn_CTRL1.clkrdy* until it reads 1.
5. Set the *TMRn_CTRL1.we* field to 1 to enable wake-up events for the timer.
6. If desired, enable the timer interrupt and provide a TMRn_IRQn for the timer.
7. Enter a low-power mode as described in the device *Operating Modes* section.
8. When the device wakes up from the low-power mode, check the *TMRn_WKFL* register to determine if the timer is the result of the wake-up event.

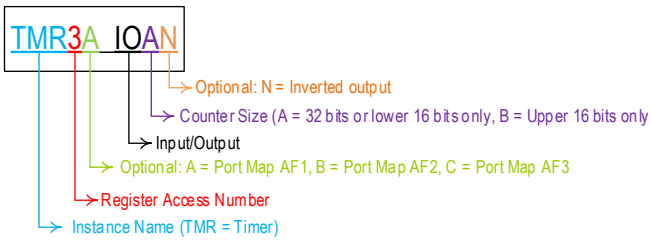
Table 23-5: MAX32690 Wake-up Events

Condition	Peripheral Wake-up Flag *TMRn_INTFL*	Peripheral Wake-up Enable	Low-Power Peripheral Wake-up Flag	Low-Power Peripheral Wake-up Enable	Power Management Wake-up Enable
Any event for LPTMR0	*irq_a*	N/A	*PWRSEQ_LPPWST .lptmr0*	*PWRSEQ_LPPWEN .lptmr0*	N/A
Any event for LPTMR1	*irq_a*	N/A	*PWRSEQ_LPPWST .lptmr1*	*PWRSEQ_LPPWEN .lptmr1*	N/A

23.7 Operating Modes

Multiple operating modes are supported. The availability of some operating modes is dependent on the device and package-specific implementation of the external input and output signals. Refer to the device data sheet for I/O signal configurations and alternate functions for each timer instance.

Figure 23-3: Timer I/O Signal Naming Conventions



In *Table 23-6*, *Table 23-7*, and *Table 23-8*, the timer's signal name is generically shown where *n* is the timer number (0, 1, 2, 3, etc.) and *y* is the port mapping alternate function. See *Figure 23-3* for details of the timer's naming convention for I/O signals.

Table 23-6: MAX32690 Operating Mode Signals for Timer 0 and Timer 1

Timer Mode	TMR0/TMR1 *TMRn_CTRL1.outen* = 0 *TMRn_CTRL1.outben* = 0	I/O Signal Name[†]	Pin Required
One-Shot Mode (0)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional
Continuous Mode (1)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional
Counter Mode (2)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
PWM Mode (3)	TimerA Output Signal	TMR*ny*_IOA	Yes
	TimerB Output Signal	TMRny_IOB	Yes
Capture Mode (4)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Compare Mode (5)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional
Gated Mode (6)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Capture/Compare Mode (7)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Dual-Edge Capture Mode (8)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Reserved (9 - 13)	-	-	-
Inactive Gated Mode (14)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes

Timer Mode	TMR0/TMR1 *TMRn_CTRL1*.*outen* = 0 *TMRn_CTRL1*.*outben* = 0	I/O Signal Name†	Pin Required
Reserved (15)	-	-	-

† *See Figure 23-3 for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.*

Table 23-7: MAX32690 Operating Mode Signals for Timer 2 and Timer 3

Timer Mode	TMR2/TMR3 *TMRn_CTRL1*.*outen_a* = 0 *TMRn_CTRL1*.*outben_a* = 0	I/O Signal Name†	Required?
One-Shot Mode (0)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
Continuous Mode (1)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
Counter Mode (2)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
PWM Mode (3)	TimerA Output Signal	TMR*ny*_IOA	Yes
	TimerB Output Signal	TMR*ny*_IOB	Yes
Capture Mode (4)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Compare Mode (5)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
Gated Mode (6)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Capture/Compare Mode (7)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Dual-Edge Capture Mode (8)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Reserved (0 - 13)	-	-	-
Interactive Gated Mode (14)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Reserved (15)	-	-	-

† *See Figure 23-3 for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.*

Table 23-8: MAX32690 Operating Mode Signals for Low-Power Timer 0 and Low-Power Timer 1

Timer mode	TMR4/TMR5 *TMRn_CTRL1*.*outen* = 0 *TMRn_CTRL1*.*outben* = 0	I/O Signal Name†	Required?
One-Shot Mode (0)	TimerA Output Signal	LPTMR*ny*_IOB	Optional
Continuous Mode (1)	TimerA Output Signal	LPTMR*ny*_IOB	Optional
Counter Mode (2)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
PWM Mode (3)	TimerA Output Signal	LPTMR*ny*_IOB	Yes
Capture Mode (4)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Compare Mode (5)	TimerA Output Signal	LPTMR*ny*_IOB	Optional
Gated Mode (6)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Capture/Compare Mode (7)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Dual-Edge Capture Mode (8)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Reserved (9 - 13)	-	-	-
Interactive Gated Mode (14)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Reserved (15)	-	-	-

† *See Figure 23-3 for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.*

23.7.1 One-Shot Mode (0)

In one-shot mode, the timer peripheral increments the timer's *TMRn_CNT.count* field until it reaches the timer's *TMRn_CMP.compare* field and the timer is then disabled. If the timer's output is enabled, the output signal is driven active for one timer clock cycle. One-shot mode provides exactly one timer period and is automatically disabled.

The timer period ends on the timer clock following *TMRn_CNT.count* = *TMRn_CMP.compare*. The timer peripheral hardware automatically performs the following actions at the end of the timer period:

- The *TMRn_CNT.count* field is set to 0x0000 0001.

  The timer is disabled (*TMRn_CTRL0.en* = 0).
- The timer output, if enabled, is driven to its active state for one timer clock period.

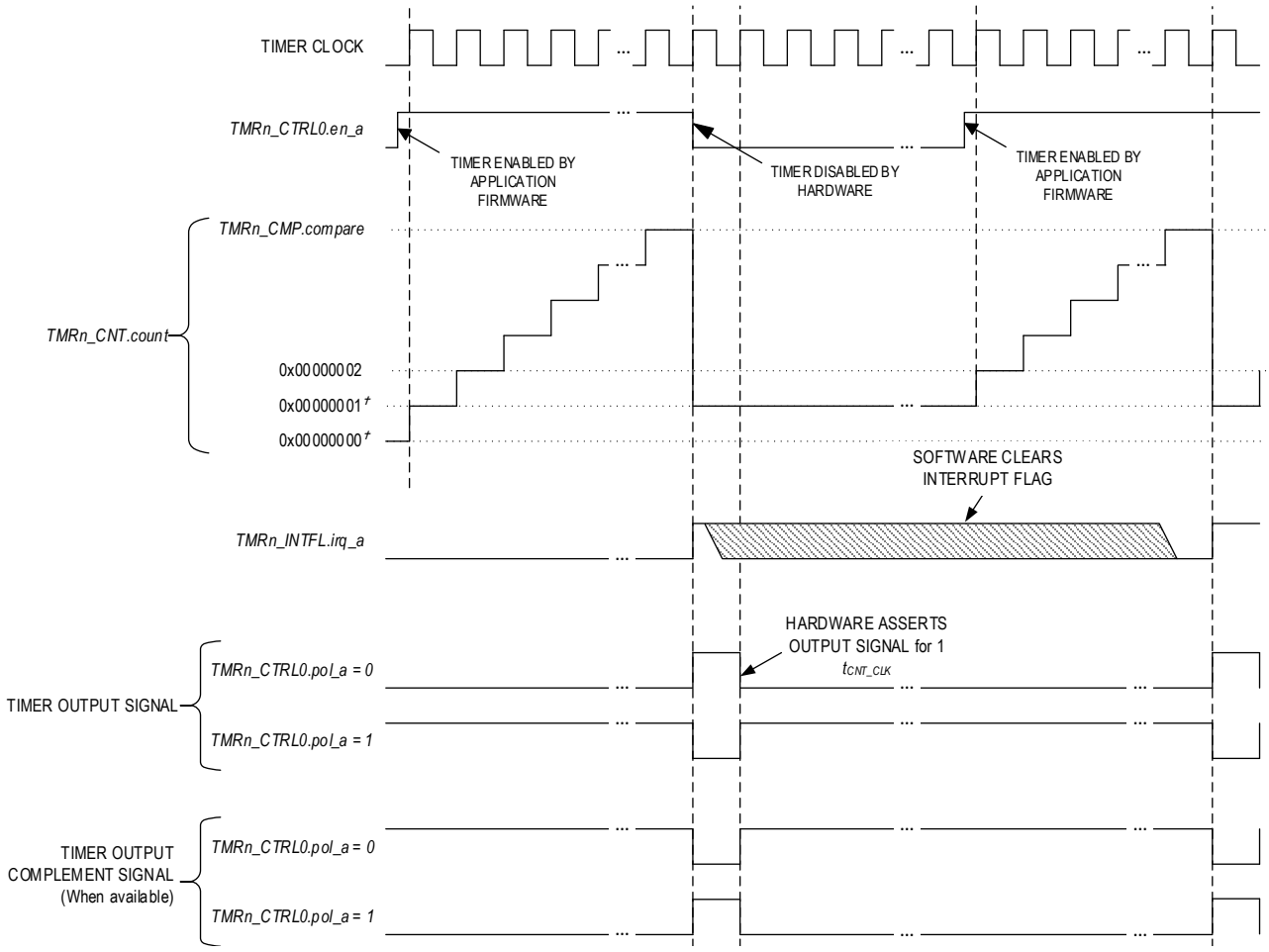
  The *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

The timer period is calculated using *Equation 23-2*.

Equation 23-2: One-Shot Mode Timer Period

$$One\text{-}shot\ mode\ timer\ period\ in\ seconds = \frac{TMRn\_CMP - TMRn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}(Hz)}$$

Figure 23-4: One-Shot Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 0 (One-shot)

† *TMRn_CNT.count* defaults to 0x00000000 on a timer reset. *TMRn_CNT.count* reloads to 0x00000001 for all following timer periods.

Configure the timer for one-shot mode by performing the following steps:

1. Disable the timer peripheral and set the timer clock source as described in *Timer Clock Sources*.
2. Set the *TMRn_CTRL0*.*mode* field to 0 to select one-shot mode.
3. Set the *TMRn_CTRL0*.*pres* field to set the prescaler for the required timer frequency.
4. If using the timer output function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
5. Or, if using the inverted timer output function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.
6. If using the timer interrupt, enable corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to the *TMRn_CMP*.*compare* field.
8. If desired, write an initial value to *TMRn_CNT*.*count* field.
   a. The initial value only affects the first period; subsequent timer periods always reset the *TMRn_CNT*.*count* field to 0x0000 0001.
9. Enable the timer peripheral as described in *Timer Clock Sources*.

23.7.2 Continuous Mode (1)

In continuous mode, the *TMRn_CNT*.*count* field increments until it matches the *TMRn_CMP*.*compare* field; the *TMRn_CNT*.*count* field is then set to 0x0000 0001 and the count continues to increment. Optionally, the software can configure continuous mode to toggle the timer output pin at the end of each timer period. A continuous mode timer period ends when the timer count field reaches the timer compare field (*TMRn_CNT*.*count* = *TMRn_CMP*.*compare*).

The timer peripheral hardware automatically performs the following actions on the timer clock cycle after the period ends:

- The *TMRn_CNT*.*count* field is set to 0x0000 0001.

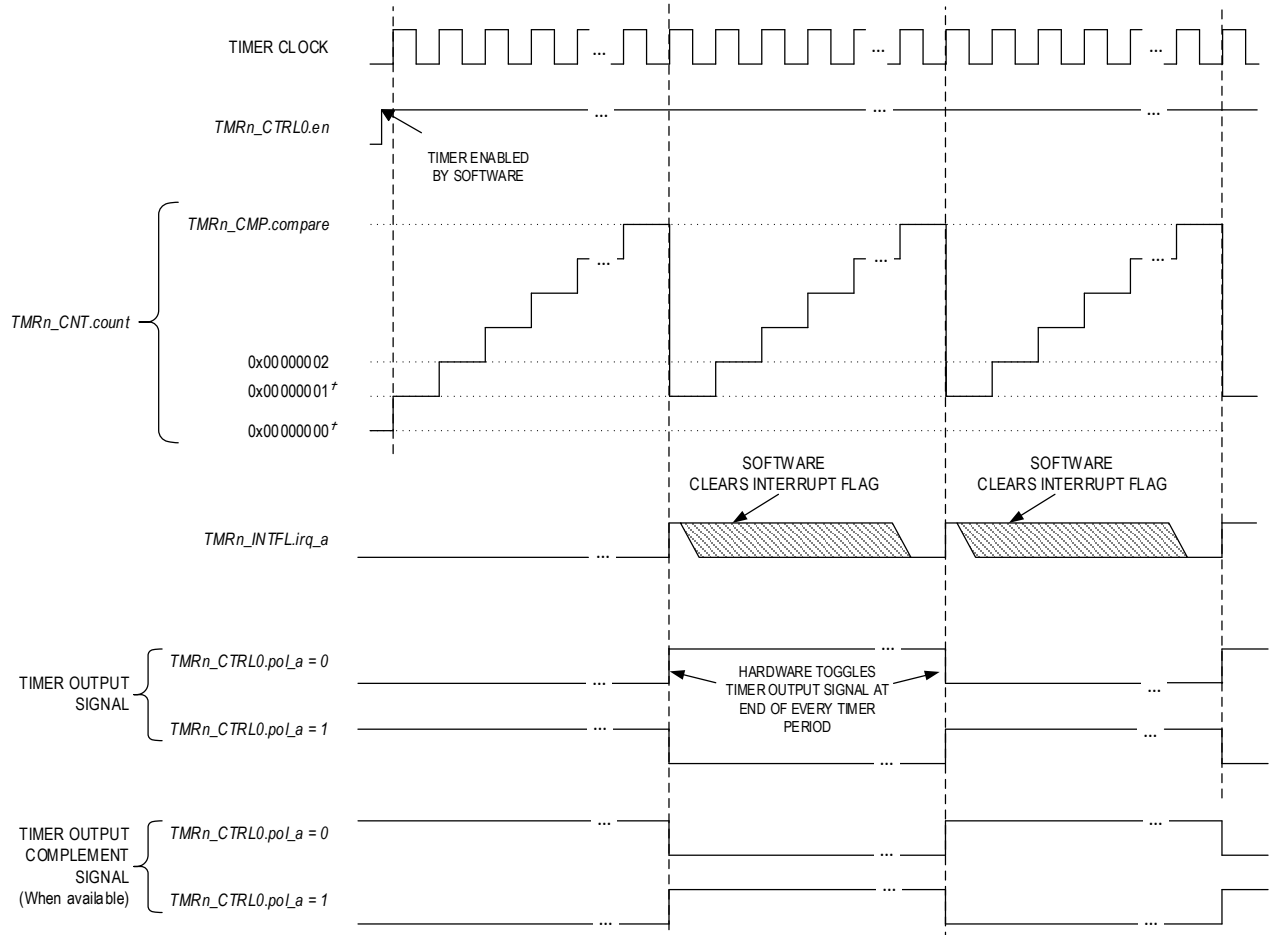
If the timer output signal is toggled, the corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

The continuous mode timer period is calculated using *Equation 23-3*.

Equation 23-3: Continuous Mode Timer Period

$$Continuous\ mode\ timer\ period\ (s) = \frac{TMRn\_CMP - TMRn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

Figure 23-5: Continuous Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 1 (Continuous)

† *TMRn_CNT.count* defaults to 0x00000000 on a timer reset. *TMRn_CNT.count* reloads to 0x00000001 for all following timer periods.

Configure the timer for continuous mode by performing the following steps:

1. Disable the timer peripheral and set the timer clock as described in *Timer Clock Sources*.
2. Set the *TMRn_CTRL0.mode* field to 1 to select continuous mode.
3. Set the *TMRn_CTRL0.pres* field to set the prescaler that determines the timer frequency.
4. If using the timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
5. Or, if using the inverted timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.
6. If using the timer interrupt, enable the corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to the *TMRn_CMP.compare* field.
8. If desired, write an initial value to the *TMRn_CNT*.count field.
   a. The initial value only affects the first period; subsequent timer periods always reset the *TMRn_CNT.count* field to 0x0000 0001.
9. Enable the timer peripheral as described in *Timer Clock Sources*.

23.7.3 Counter Mode (2)

In counter mode, the timer peripheral increments the *TMRn_CNT.count* each time a transition occurs on the timer input signal. The transition must be greater than $4 \times PCLK$ for a count to occur. When the *TMRn_CNT.count* reaches the *TMRn_CMP.compare* field, the hardware automatically sets the interrupt bit to 1 (*TMRn_INTFL.irq)*, sets the *TMRn_CNT.count* field to 0x0000 0001, and continues incrementing. The timer can be configured to increment on either the rising edge or falling edge of the timer's input signal, but not both. Use the *TMRn_CTRL0.pol_* field to select which edge is used for the timer's input signal count.

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 *Equation 23-4*.

Note: If the input signal's frequency is equal to f_{PCLK}, it is possible the transition can be missed by the timer hardware due to PCLK being an asynchronous internal clock. A minimum of 4 PCLK cycles is required for a count to occur. To guarantee a count occurs, the timer input signal should be greater than 4 PCKL cycles.

Equation 23-4: Counter Mode Maximum Clock Frequency

$$f_{CTR\_CLK} \leq \frac{f_{PCLK}\ (Hz)}{4}$$

The timer period ends on the rising edge of PCLK following *TMRn_CNT.count* = *TMRn_CMP.compare*.

The timer peripheral's hardware automatically performs the following actions at the end of the timer period:

- The *TMRn_CNT.count* field is set to 0x0000 0001.

The timer output signal is toggled if the timer output pin is enabled.

- The *TMRn_INTFL.irq* field to 1 indicating a timer interrupt event occurred.

The timer remains enabled and continues incrementing.

Note: The software must clear the interrupt flag by writing 1 to the TMRn_INTFL.irq field. If the timer period ends and the interrupt flag is already set to 1, a second interrupt does not occur.

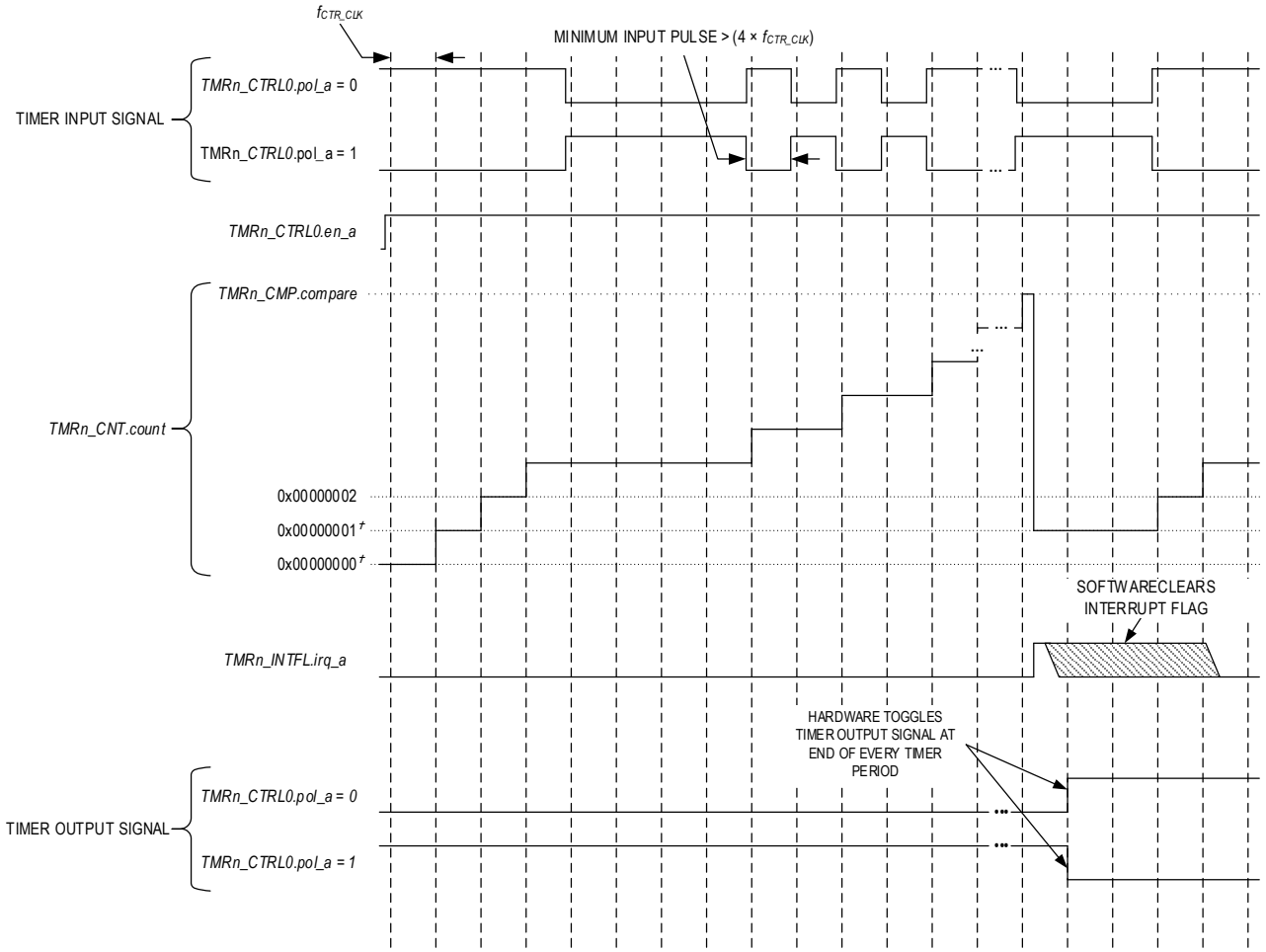
In counter mode, the number of timer input transitions that occurred during a period is equal to the *TMRn_CMP.compare* field's setting. Use *Equation 23-5* to determine the number of transitions that occurred before the end of the timer's period.

Note: Equation 23-5 *is only valid during an active timer count before the end of the timer's period.*

Equation 23-5: Counter Mode Timer Input Transitions

$$Counter\ mode\ timer\ input\ transitions = TMR\_CNT_{CURRENT\_VALUE}$$

Figure 23-6: Counter Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 2 (Counter)

† *TMRn_CNT.count* defaults to 0x00000000 on a timer reset. *TMRn_CNT.count* reloads to 0x00000001 for all following timer periods.

Configure the timer for counter mode by performing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* 0x2 to select Counter mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Set *TMRn_CTRL1*.*outen_a* and *TMRn_CTRL1*.*outben* to the values shown in the *Operating Modes* section.
   d. Select the correct alternate function mode for the timer input pin.
5. Write the compare value to *TMRn_CMP*.*compare*.
6. If desired, write an initial value to *TMRn_CNT*.*count*. This affects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0001.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

23.7.4 PWM Mode (3)

In PWM mode, the timer sends a PWM output using the timer's output signal. The timer first counts up to the match value stored in the *TMRn_PWM*.*pwm* register. At the end of the cycle where the *TMRn_CNT*.*count* value matches the *TMRn_PWM*.*pwm*, the timer output signal toggles state. The timer continues counting until it reaches the *TMRn_CMP*.*compare* value.

The timer period ends on the rising edge of f_{CNT_CLK} following *TMRn_CNT*.*count* = *TMRn_CMP*.*compare*.

The timer peripheral automatically performs the following actions at the end of the timer period:

- The *TMRn_CNT*.*count* is reset to 0x0000 0001 and the timer resumes counting.

The timer output signal is toggled.

- The corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

When *TMRn_CTRL0*.*pol* = 0, the timer output signal starts low and then transitions to high when the *TMRn_CNT*.*count* value matches the *TMRn_PWM* value. The timer output signal remains high until the *TMRn_CNT*.*count* value reaches the *TMRn_CMP*.*compare*, resulting in the timer output signal transitioning low, and the *TMRn_CNT*.*count* value resetting to 0x0000 0001.

When *TMRn_CTRL0*.*pol* = 1, the Timer output signal starts high and transitions low when the *TMRn_CNT*.*count* value matches the *TMRn_PWM* value. The timer output signal remains low until the *TMRn_CNT*.*count* value reaches *TMRn_CMP*.*compare*, resulting in the timer output signal transitioning high, and the *TMRn_CNT*.*count* value resetting to 0x0000 0001.

Complete the following steps to configure a timer for PWM mode and initiate the PWM operation:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set the *TMRn_CTRL0.mode field* to 3 to select PWM mode.
4. Set the *TMRn_CTRL0.pres* field to set the prescaler that determines the timer frequency.
5. Configure the pin as a timer input and configure the electrical characteristics as needed.
6. Set *TMRn_CTRL0.pol* to match the desired initial (inactive) state.
7. Set *TMRn_CTRL0.pol* to select the initial logic level (high or low) and PWM transition state for the timer's output.
8. Set *TMRn_CNT.count* initial value if desired.
    a. The initial *TMRn_CNT.count* value only affects the initial period in PWM mode with subsequent periods always setting *TMRn_CNT.count* to 0x0000 0001.
9. Set the *TMRn_PWM* value to the transition period count.
    a. If using the timer in dual 16-bit mode, both timers must be disabled before writing the *TMRn_PWM* register.
10. Set the *TMRn_CMP.compare* value for the PWM second transition period. Note: *TMRn_CMP.compare* must be greater than the *TMRn_PWM* value.
    a. If using the timer in dual 16-bit mode, both timers must be disabled before writing the *TMRn_CMP* register.
11. If using the timer interrupt, set the interrupt priority and enable the interrupt.
12. Enable the timer peripheral as described in *Timer Clock Sources*.

Equation 23-6 shows the formula for calculating the timer PWM period.

Equation 23-6: Timer PWM Period

$$PWM\ period\ (s) = \frac{TMRn\_CNT}{f_{CNT\_CLK}\ (Hz)}$$

If an initial starting value other than 0x0000 0001 is loaded into the *TMRn_CNT.count* register, use the one-shot mode equation, *Equation 23-2*, to determine the initial PWM period.

If *TMRn_CTRL0.pol* is 0, the ratio of the PWM output high time to the total period is calculated using *Equation 23-7*.

Equation 23-7: Timer PWM Output High Time Ratio with Polarity 0

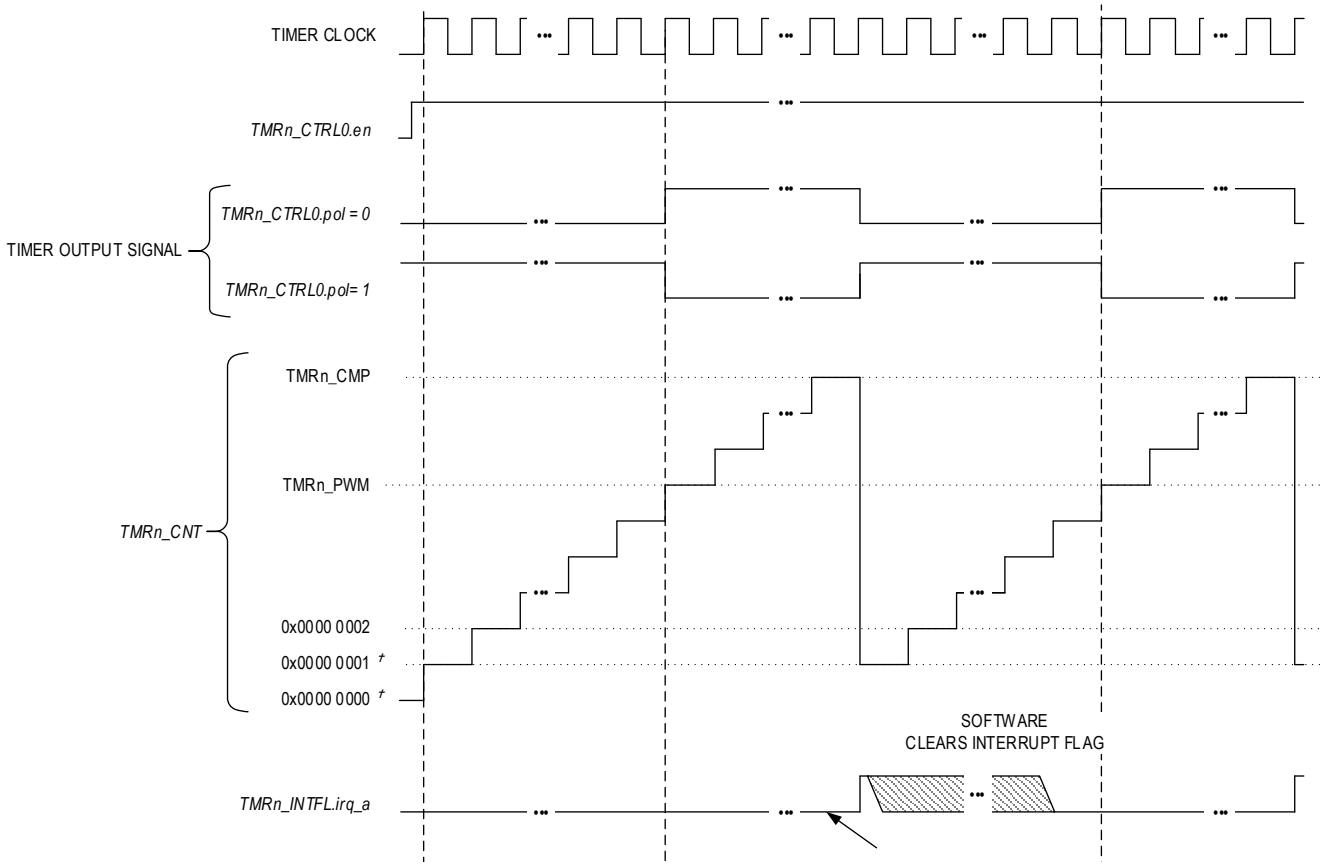
$$PWM\ output\ high\ time\ ratio\ (\%) = \frac{(TMR\_CMP - TMR\_PWM)}{TMR\_CMP} \times 100$$

If *TMRn_CTRL0.pol* is set to 1, the ratio of the PWM output high time to the total period is calculated using *Equation 23-8*.

Equation 23-8: Timer PWM Output High Time Ratio with Polarity 1

$$PWM\ output\ high\ time\ ratio\ (\%) = \frac{TMR\_PWM}{TMR\_CMP} \times 100$$

Figure 23-7: PWM Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 3 (PWM)

† *TMRn_CNT* defaults to 0x00000000 on a timer reset. *TMRn_CNT* reloads to 0x00000001 for all following timer periods.

23.7.5 Capture Mode (4)

Capture mode is used to measure the time between software-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 the hardware when the timer's input pin transitions state. *Equation 23-9* shows the formula for calculating the capture event's elapsed time.

If a capture event does not occur before the timer's count value reaching the timer's compare value (*TMRn_CNT*.*count* = *TMRn_CMP*.*compare*), a rollover event occurs. Both the capture event and the rollover event set the timer's interrupt flag, *TMRn_INTFL*.*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. The software must track the number of rollover events that occur before a capture event to determine the elapsed time of the capture event. When a capture event occurs, the 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.

23.7.5.1 Capture Event

When a capture event occurs, the timer hardware, on the next timer clock cycle, automatically performs the following actions:

- The *TMRn_CNT.count* value is copied to the *TMRn_PWM.pwm* field.

The *TMRn_INTFL.irq* field is set to 1.

- The timer remains enabled and continues counting.

The software must check the value of the TMRn_PWM.pwm field to determine the trigger of the timer interrupt.

Equation 23-9: Capture Mode Elapsed Time Calculation in Seconds

$$Capture\ elapsed\ time\ (s) = \frac{\left(TMR\_PWM - TMR\_CNT_{INITIAL\_VALUE}\right) + \left((Number\ of\ rollover\ events) \times \left(TMR\_CMP - TMR\_CNT_{INITIAL\_VALUE}\right)\right)}{f_{CNT\_CLK}}$$

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.

23.7.5.2 Rollover Event

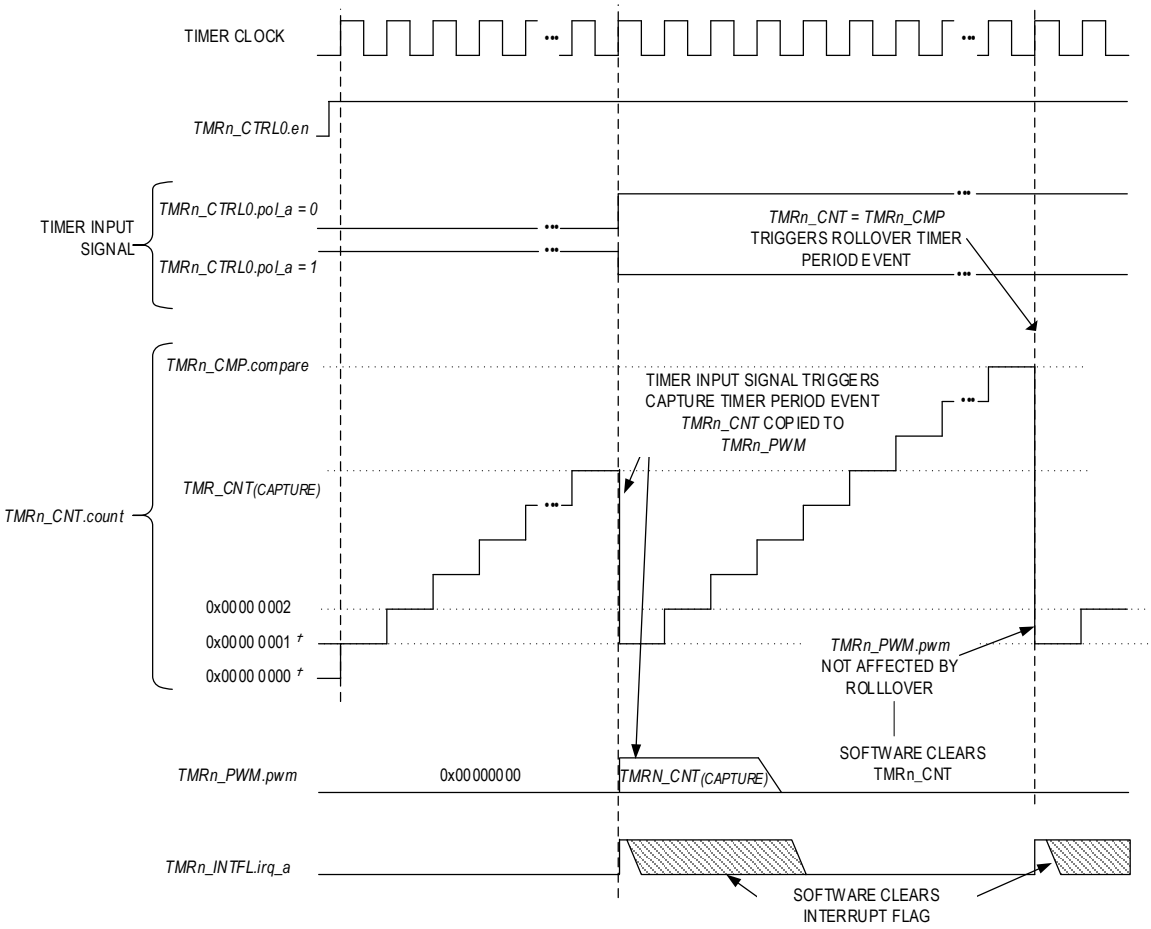
A rollover event occurs when the timer's count value reaches the timer's compare value (*TMRn_CNT.count* = *TMRn_CMP.compare*). A rollover event indicates that a capture event did not occur within the set timer period. When a rollover event occurs, the timer hardware automatically performs the following actions during the next timer clock period:

- The *TMRn_CNT.count* field is set to 0x0000 0001.

The *TMRn_INTFL.irq* field is set to 1.

- The timer remains enabled and continues counting.

Figure 23-8: Capture Mode Diagram



THIS EXAMPLES USES THE FOLLOWING CONFIGURATION IN ADDITION TO THE SETTINGS SHOWN ABOVE:
TMRn_CTRL1.cascade = 1 (32-BIT CASCADE TIMER)
TMRn_CTRL0.mode_a = 2 (COUNTER)

† *TMRn*_CNT.*count* DEFAULTS TO 0x00000000 ON A TIMER RESET. *TMRn_CNT.count* RELOADS TO 0x00000001 FOR ALL FOLLOWING TIMER PERIODS.

Configure the timer for Capture mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* to 4 to select capture mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. Write the initial value to *TMRn_CNT*.*count*, if desired.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0001.
6 Write the compare value to the *TMRn_CMP*.*compare* field.
7. Select the capture event by setting *TMRn_CTRL1*.*capeventsel*.
8. Enable the timer peripheral as described in *Timer Clock Sources*.

The timer period is calculated using the following equation:

Equation 23-10: Capture Mode Elapsed Time Calculation in Seconds

$$Capture\ elapsed\ time\ in\ seconds = \frac{TMR\_PWM - TMR\_CNT_{INITIAL\_VALUE}}{f_{CNT\_CLK}}$$

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.

23.7.6 Compare Mode (5)

In compare mode, the timer peripheral increments continually from 0x0000 0000 (after the first timer period) to the maximum value of the 32- or 16-bit mode, then rolls over to 0x0000 0000 and continues incrementing. The end of the 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 *TMRn_CNT*.*count* = *TMRn_CMP*.*compare*.

The timer peripheral automatically performs the following actions when a timer period event occurs:

- Unlike other modes, *TMRn_CNT*.*count* is reset to 0x0000 00000, not 0x0000 0001 at the end of the timer period. The timer remains enabled and continues incrementing.

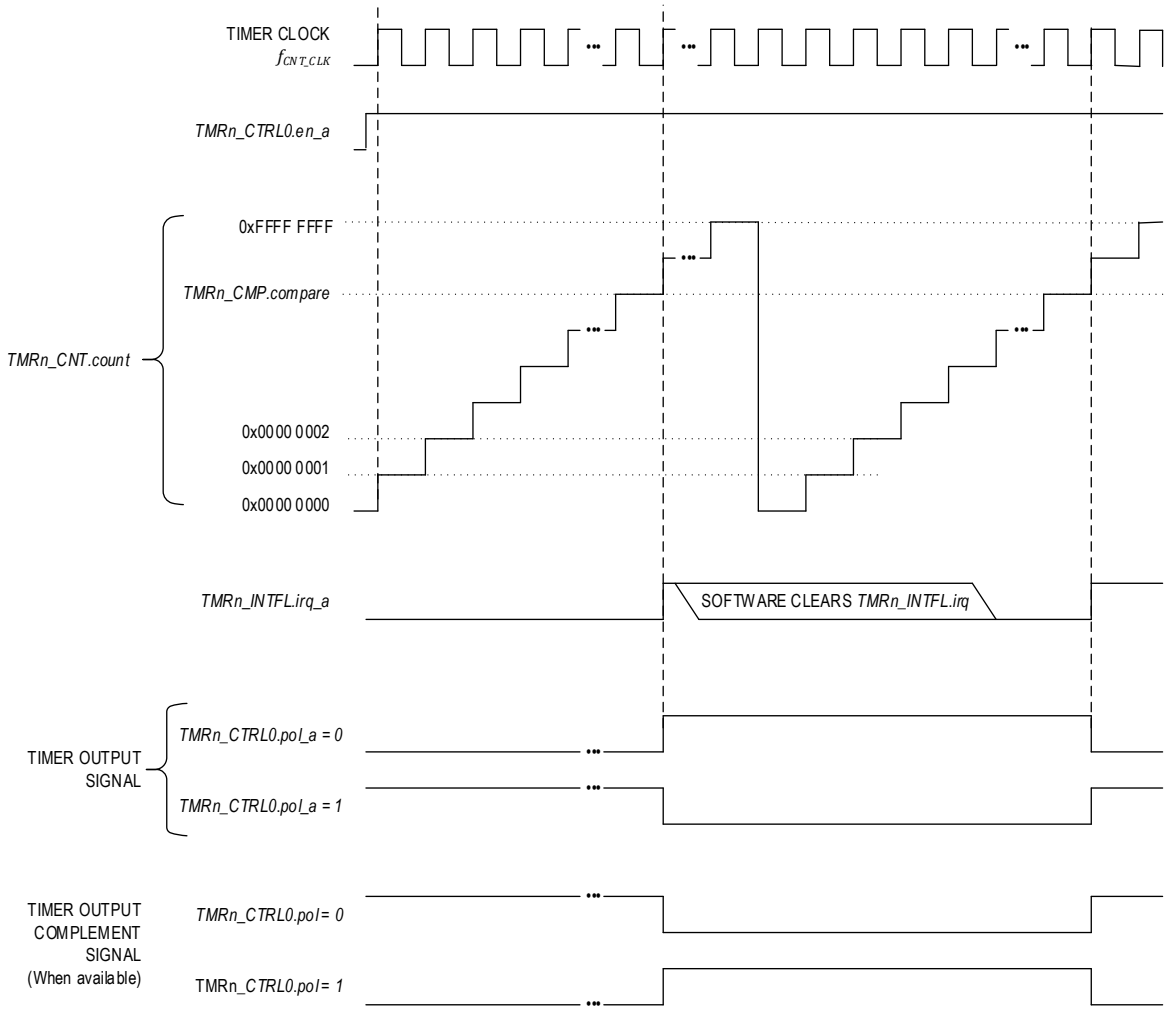
  The corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.
- The hardware toggles the state of the timer output signal. The timer output pin changes state if the timer output is enabled.

The compare mode timer period is calculated using *Equation 23-11*.

Equation 23-11: Compare Mode Timer Period

$$Compare\ mode\ timer\ period\ in\ second = \frac{(TMR\_CMP - TMR\_CNT_{INITIAL\_VALUE} + 1)}{f_{CNT\_CLK}(Hz)}$$

Figure 23-9: Compare Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 5 (Compare)

Configure the timer for compare mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* to 5 to select Compare mode.
4. Set *TMRn_CTRL0*.*pres* to set the prescaler that determines the timer frequency.
5. If using the timer output function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
6. If using the inverted timer output function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.
7. If using the timer interrupt, enable the corresponding field in the *TMRn_CTRL1* register.
8. Write the compare value to *TMRn_CMP*.*compare.*
9. If desired, write an initial value to *TMRn_CNT*.*count*.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0000.
10. Enable the timer peripheral as described in *Timer Clock Sources*.

23.7.7 Gated Mode (6)

Gated mode is similar to continuous mode, except that *TMRn_CNT*.*count* only increments when the timer input signal is in its active state.

The timer period ends on the timer clock following *TMRn_CNT*.*count* = *TMRn_CMP*.*compare*.

The timer peripheral automatically performs the following actions at the end of the timer period:

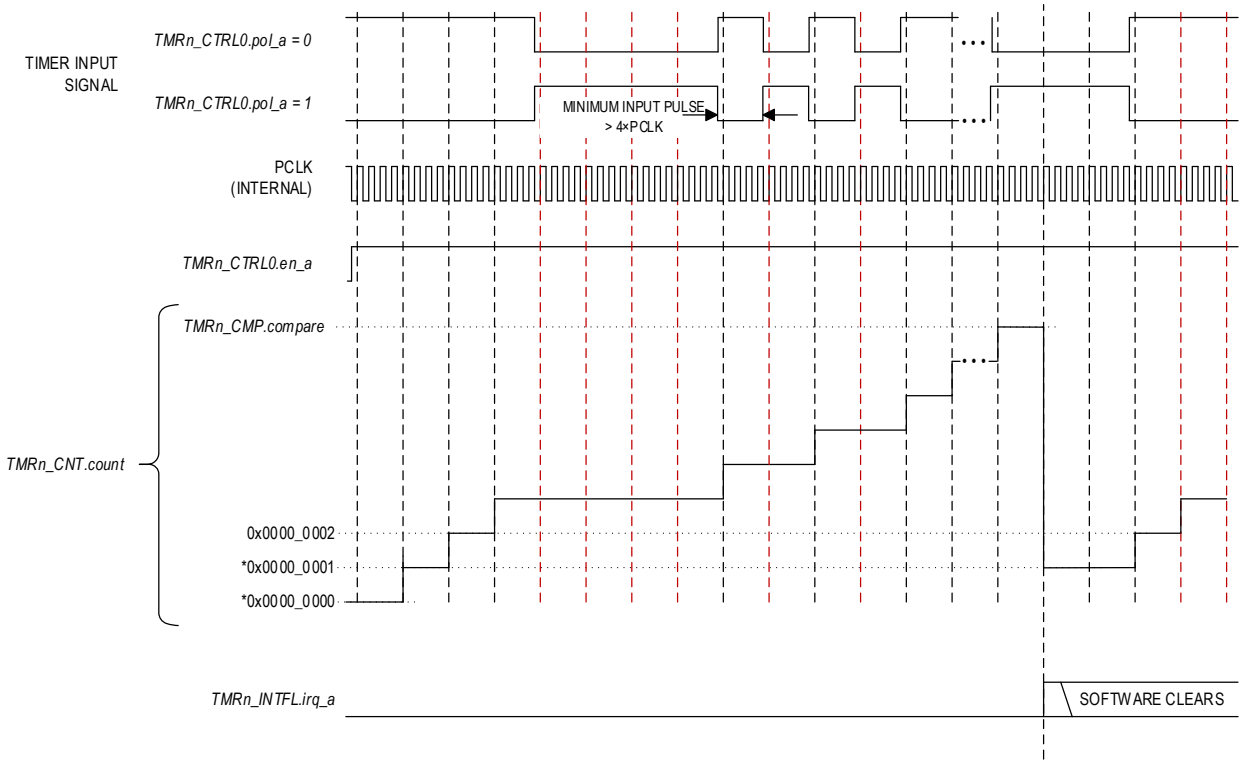
- The *TMRn_CNT*.*count* field is set to 0x0000 0001.

The timer remains enabled and continues incrementing.

- The timer output pin changes state if the timer output is enabled.

The corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

Figure 23-10: Gated Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 6 (Gated)

† *TMRn_CNT.count* defaults to 0x00000000 on a timer reset. *TMRn_CNT.count* reloads to 0x00000001 for all following timer periods.

Configure the timer for gated mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* to 6 to select gated mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. If desired, write an initial value to the *TMRn_CNT*.*count* field.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0001.
6 Write the compare value to *TMRn_CMP*.*compare*.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

23.7.8 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_CTRL0.pol* bit.

Each subsequent transition, after the first transition of the timer input signal, captures the *TMRn_CNT.count* value, writing it to the *TMRn_PWM.pwm* register (capture event). When a capture event occurs, a timer interrupt is generated, the *TMRn_CNT.count* value is reset to 0x0000_0001, and the timer resumes counting.

If no capture event occurs, the timer counts up to *TMRn_CMP.compare*. At the end of the cycle where the *TMRn_CNT.count* equals the *TMRn_CMP.compare*, a timer interrupt is generated, the *TMRn_CNT.count* value is reset to 0x0000 0001, and the timer resumes counting.

The timer period ends when the selected transition occurs on the timer pin, or on the clock cycle following *TMRn_CNT.count* = *TMRn_CMP.compare*.

The actions performed at the end of the timer period are dependent on the event that ended the timer period.

If the end of the timer period is caused by a transition on the timer pin, the hardware automatically performs the following:

- The value in *TMRn_CNT.count* field is copied to the *TMRn_PWM.pwm* field.

The *TMRn_CNT.count* field is set to 0x0000 0001.

- The timer remains enabled and continues incrementing.

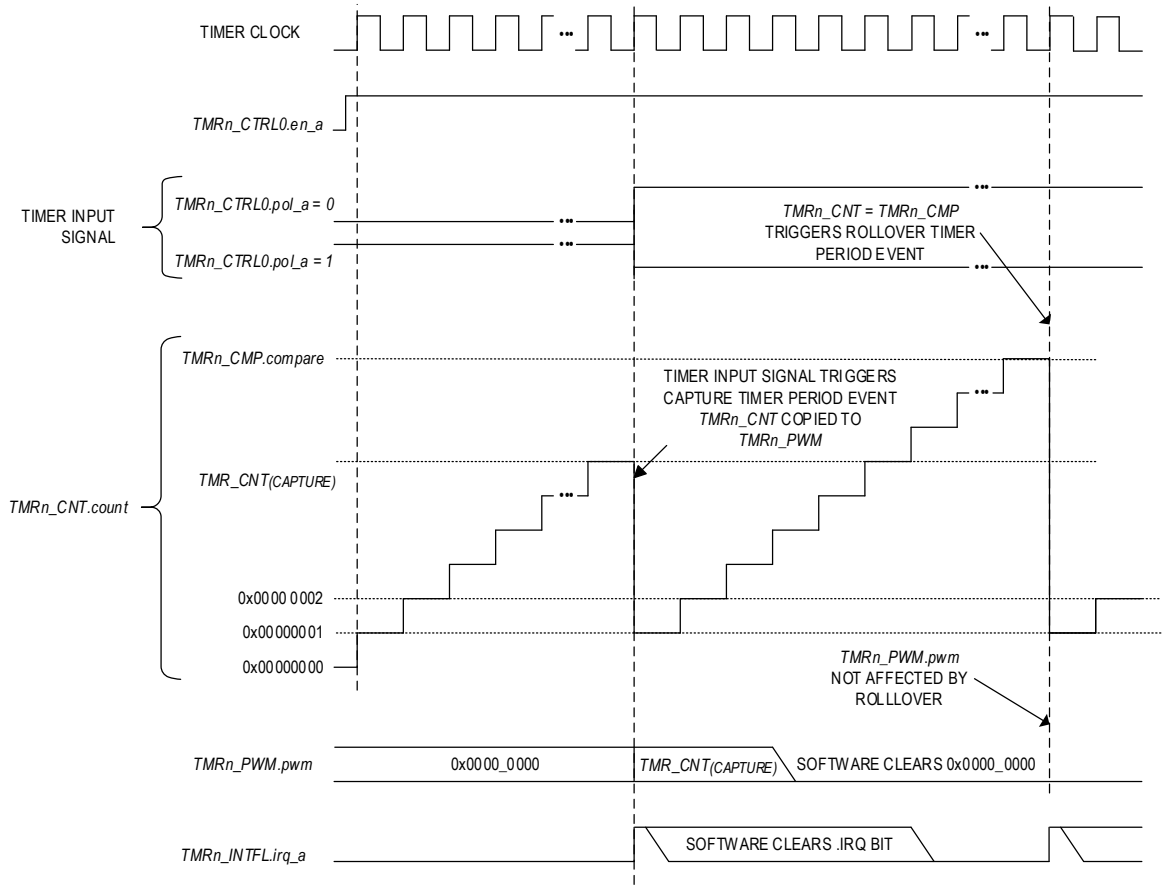
The corresponding *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

In capture/compare mode, the elapsed time from the timer start to the capture event is calculated using *Equation 23-12*.

Equation 23-12: Capture Mode Elapsed Time

$$Capture\ elapsed\ time\ (seconds) = \frac{TMRn\_PWM - TMRn\_CNT_{INITIAL\_CNT\_VALUE}}{f_{CNT\_CLK}(Hz)}$$

Figure 23-11: Capture/Compare Mode Diagram



THIS EXAMPLES USES THE FOLLOWING CONFIGURATION IN ADDITION TO THE SETTINGS SHOWN ABOVE:
TMRn_CTRL1.cascade = 1 (32-BIT CASCADE TIMER)
TMRn_CTRL0.mode_a = 7 (CAPTURE/COMPARE)

† *TMRn_CNT*.count DEFAULTS TO 0x00000000 ON A TIMER RESET. *TMRn_CNT*.count RELOADS TO 0x00000001 FOR ALL FOLLOWING TIMER PERIODS.

Configure the timer for Capture/Compare mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0.mode* to 7 to select Capture/Compare mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0.pol* to select the positive edge (*TMRn_CTRL0.pol* = 1) or negative edge (*TMRn_CTRL0.pol* = 0) transition to cause the capture event..
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. If desired, write an initial value to the *TMRn_CNT.count* field.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT.count* = 0x0000 0001.
6 Write the compare value to *TMRn_CMP.compare*.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

Note: No interrupt is generated by the first transition of the input signal.

23.7.9 Dual Edge Capture Mode (8)

Dual edge capture mode is similar to capture mode except the counter can capture on both edges of the timer input pin.

23.7.10 Inactive Gated Mode (14)

Inactive gated mode is similar to gated mode except that the interrupt is triggered when the timer input pin is in its inactive state.

23.8 Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 23-9*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 1-1* 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 23-9: Timer Register Summary

Offset	Register	Description
[0x0000]	*TMRn_CNT*	Timer Counter Register
[0x0004]	*TMRn_CMP*	Timer Compare Register
[0x0008]	*TMRn_PWM*	Timer PWM Register
[0x000C]	*TMRn_INTFL*	Timer Interrupt Register
[0x0010]	*TMRn_CTRL0*	Timer Control Register
[0x0014]	*TMRn_NOLCMP*	Timer Non-Overlapping Compare Register
[0x0018]	*TMRn_CTRL1*	Timer Configuration Register
[0x001C]	*TMRn_WKFL*	Timer Wake-up Status Register

23.8.1 Register Details

Table 23-10: Timer Count Register

Timer Count			TMRn_CNT	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	count	R/W	0	**Timer Count** This field increments at a rate dependent on the selected timer operating mode. The function of the bits in this field are dependent on the 32-bit/16-bit configuration. Reads of this register always return the current value.

Table 23-11: Timer Compare Register

Timer Compare			TMRn_CMP	[0x0004]
Bits	**Field**	**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 23-12: Timer PWM Register

Timer PWM			TMRn_PWM	[0x0008]
Bits	**Field**	**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 when *TMRn_CNT.count* = *TMRn_CMP.compare*, the PWM output transitions to the second period of the PWM cycle. The second PWM period count is stored in *TMRn_CMP.compare*. *TMRn_PWM.pwm* must be less than *TMRn_CMP.compare* for PWM mode operation. **Timer Capture Value** In capture, compare, and capture/compare modes, this field is used to store the *TMRn_CNT.count* value when a Capture, Compare, or Capture/Compare event occurs.

Table 23-13: Timer Interrupt Register

Timer Interrupt			TMRn_INTFL	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
25	wr_dis_b	R/W	0	**TimerB Write Protect in Dual Timer Mode** Set this field to 0 to write protect the TimerB fields in the *TMRn_CNT.count[31:16]* and *TMRn_PWM.pwm[31:16]*. When this field is set to 0, 32-bit writes to the *TMRn_CNT* and *TMRn_PWM* registers only modify the lower 16 bits associated with TimerA. 0: Enabled. 1: Disabled. *Note: This field always reads 0 if the timer is configured as a 32-bit cascade timer.*
24	wrdone_b	R	0	**TimerB Write Done** This field is cleared to 0 by the hardware when the software performs a write to *TMRn_CNT.count[31:16]* or *TMRn_PWM.pwm[31:16]* when in dual timer mode. Wait until the field is set to 1 before proceeding. 0: Operation in progress. 1: Operation complete.
23:17	-	RO	0	**Reserved**

Timer Interrupt			TMRn_INTFL	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
16	irq_b	R/W1C	0	**TimerB Interrupt Event** This field is set when a TimerB interrupt event occurs. Write 1 to clear. 0: No event. 1: Interrupt event occurred.
15:10	-	RO	0	**Reserved**
9	wr_dis_a	R/W	0	**TimerA Dual Timer Mode Write Protect** This field disables write access to the *TMRn_CNT.count[15:0]* and *TMRn_PWM.pwm[15:0]* fields so that only the 16 bits associated with updating TimerA are modified during writes to the *TMRn_CNT* and *TMRn_PWM* registers. 0: Enabled. 1: Disabled. *Note: This field always reads 0 if the timer is configured as a 32-bit cascade timer.*
8	wrdone_a	R	0	**TimerA Write Done** This field is cleared to 0 by the hardware when the software performs a write to *TMRn_CNT.count[15:0]* or *TMRn_PWM.pwm[15:0]* when in dual 16-bit timer mode. Wait until the field reads 1 before proceeding. 0: Operation in progress. 1: Operation complete.
7:1	-	RO	0	**Reserved**
0	irq_a	W1C	0	**TimerA Interrupt Event** This field is set when a TimerA interrupt event occurs. Write 1 to clear. 0: No event. 1: Interrupt event occurred.

Table 23-14: Timer Control 0 Register

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31	en_b	R/W	0	**TimerB Enable** 0: Disabled. 1: Enabled.
30	clken_b	R/W	0	**TimerB Clock Enable** 0: Disabled. 1: Enabled.
29	rst_b	W1	0	**TimerB Reset** 0: No action. 1: Reset TImerB.
28:24	-	RO	0	**Reserved**

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
23:20	clkdiv_b	R/W	0	**TimerB Prescaler Select** The *clkdiv_b* field selects a prescaler that divides the timer's source clock to set the timer's count clock as follows: $f_{CNT\_CLK} = {f_{CLK\_SOURCE}}/{prescaler}$ See the *Operating Modes* section for details on which timer modes use the prescaler. 0: 1 1: 2 2: 4 3: 8 4: 16 5: 32 6: 64 7: 128 8: 256 9: 512 10: 1024 11: 2048 12: 4096 13-15: Reserved.
19:16	mode_b	R/W	0	**TimerB Mode Select** Set this field to the desired mode for TimerB. 0: One-Shot. 1: Continuous. 2: Counter. 3: PWM. 4: Capture. 5: Compare. 6: Gated. 7: Capture/Compare. 8: Dual-Edge Capture. 9-11: Reserved. 12: Internally Gated. 13-15: Reserved.
15	en_a	R/W	0	**TimerA Enable** 0: Disabled. 1: Enabled.
14	clken_a	R/W	0	**TimerA Clock Enable** 0: Disabled. 1: Enabled.
13	rst_a	R/W1O	0	**TimerA Reset** 0: No action. 1: Reset TimerA.

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
12	pwmckbd_a	R/W	1	**TimerA PWM Output $\phi A'$ Disable** Set this field to 0 to enable the $\phi A'$ output signal. The $\phi A'$ output signal is disabled by default. 0: Enable the PWM $\phi A'$ output signal. 1: Disable PWM $\phi A'$ output signal.
11	nollpol_a	R/W	0	**TimerA PWM Output $\phi A'$ Polarity Bit** Set this field to 1 to invert the PWM $\phi A'$ signal. 0: Do not invert the PWM $\phi A'$ output signal. 1: Invert the PWM $\phi A'$ output signal.
10	nolhpol_a	R/W	0	**TimerA PWM Output ϕA Polarity Bit** Set this field to 1 to invert the PWM ϕA signal. 0: Do not invert the ϕA PWM output signal. 1: Invert the ϕA output signal.
9	pwmsync_a	R/W	0	**TimerA/TimerB PWM Synchronization Mode** 0: Disabled. 1: Enabled.
8	pol_a	R/W	0	**TimerA 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 timer's alternate function. This field's usage and settings are operating mode specific. See the *Operating Modes* section for details on the mode selected.
7:4	clkdiv_a	R/W	0	**TimerA Prescaler Select** The *clkdiv_a* field selects a prescaler that divides the timer's clock source to set the timer's count clock as follows: $f_{CNT\_CLK} = f_{CLK\_SOURCE} / prescaler$ See the *Operating Modes* section to determine which modes use the prescaler. 0: 1 1: 2 2: 4 3: 8 4: 16 5: 32 6: 64 7: 128 8: 256 9: 512 10: 1024 11: 2048 12: 4096 13-15: Reserved.

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
3:0	mode_a	R/W	0	**TimerA Mode Select** Set this field to the desired operating mode for TimerA. 0: One-Shot. 1: Continuous. 2: Counter. 3: PWM. 4: Capture. 5: Compare. 6: Gated. 7: Capture/Compare. 8: Dual-Edge Capture. 9-11: Reserved. 12: Internally Gated. 13-15: Reserved.

Table 23-15: Timer Non-Overlapping Compare Register

Timer Non-Overlapping Compare			TMRn_NOLCMP	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	hi_b	R/W	0	**TimerA Non-Overlapping High Compare 1** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output $\phi A'$ (phase A prime) and the next rising edge of the PWM output ϕA (phase A).
23:16	lo_b	R/W	0	**TimerA Non-Overlapping Low Compare 1** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output ϕA and the next rising edge of the PWM output $\phi A'$.
15:8	hi_a	R/W	0	**TimerA Non-Overlapping High Compare 0** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output $\phi A'$ and the next rising edge of the PWM output ϕA.
7:0	lo_a	R/W	0	**TimerA Non-Overlapping Low Compare 0** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output ϕA and the next rising edge of the PWM output $\phi A'$.

Table 23-16: Timer Control 1 Register

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31	cascade	R/W	0	**32-bit Cascade Timer Enable** This field is only supported by Timer instances with support for 32-bit cascade mode. 0: Dual 16-bit timers 1: 32-bit cascade timer
30	outben_b	R/W	0	**TimerB Output B Enable** Reserved for future use
29	outen_b	R/W	0	**TimerB Output Enable** Reserved
28	we_b	R/W	0	**TimerB Wake-up Function** 0: Disabled. 1: Enabled.

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
27	sw_capevent_b	R/W	0	**TimerB Software Event Capture** Write this field to 1 to initiate a software event capture when operating the timer in capture mode to perform a software event capture. 0: No event. 1: Reserved.
26:25	capevent_sel_b	R/W	0	**TimerB Event Capture Selection** Set this field to the desired capture event source. See *Table 23-2* for available capture event 0 and capture event 1 options. 0-3: Reserved.
24	ie_b	R/W	0	**TimerB Interrupt Enable** 0: Disabled. 1: Enabled.
23	negtrig_b	R/W	0	**TimerB Negative Edge Trigger for Event** 0: Rising-edge trigger. 1: Falling-edge trigger. *Note: External trigger events for rising-edge events must be active for a minimum of two timer clocks for detection.*
22:20	event_sel_b	R/W	0	**TimerB Event Selection** 0: Event disabled. 1-7: Reserved.
19	clkrdy_b	RO	0	**TimerB Clock Ready Status** This field indicates if the timer clock is ready. 0: Timer clock not ready or synchronization in progress. 1: Timer clock is ready.
18	clken_b	RO	0	**TimerB Clock Enable Status** This field indicates the status of the timer enable. 0: Timer not enabled or synchronization in progress. 1: Timer is enabled.
17:16	clksel_b	R/W	0	**TimerB Clock Source** See *Table 23-1* for the clock sources supported by each instance. *Note: In cascade 32-bit mode this field must be set to match the TMRn_CTRL1.clksel_a field.* 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.
15	-	RO	0	**Reserved**
14	outben_a	R/W	0	**Output B Enable** Reserved.
13	outen_a	R/W	0	**Output Enable** Reserved.
12	we_a	R/W	0	**TimerA Wake-up Function** 0: Disabled. 1: Enabled.
11	sw_capevent_a	R/W	0	**TimerA Software Event capture** 0: Normal operation. 1: Trigger software capture event.

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
10:9	capeventsel_a	R/W	0	**TimerA Event capture Selection** Set this field to the desired capture event source. See *Table 23-2* for available capture event 0 and capture event 1 options. 0: Capture event 0. 1: Capture event 1. 2: Capture event 2. 3: Capture event 3.
8	ie_a	R/W	0	**TimerA Interrupt Enable** 0: Disabled. 1: Enabled.
7	negtrig_a	R/W	0	**TimerA Edge Trigger Selection for Event** 0: Positive-edge triggered. 1: Negative-edge triggered. *Note: External trigger events for rising-edge events must be active for a minimum of two timer clocks for detection.*
6:4	event_sel_a	R/W	0	**TimerA Event Selection** 0: Event disabled. 1-7: Reserved.s
3	clkrdy_a	RO	0	**TimerA Clock Ready** This field is set to 1 after the software enables the TimerA clock by writing 1 to the *TMRn_CTRL1.clken_a* field. 0: Timer not enabled or synchronization in progress. 1: TimerA clock is ready.
2	clken_a	R/W	0	**TimerA Clock Enable** Write this field to 1 to enable the TimerA clock. 0: Timer not enabled or synchronization in progress. 1: Timer is enabled.
1:0	clksel_a	R/W	0	**Clock Source TimerA** See *Table 23-1* for the available clock options for each timer instance. *Note: In cascade 32-bit mode the TMRn_CTRL1.clksel_b field must be set to the same value as this field.* 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.

Table 23-17: Timer Wake-up Status Register

Timer Wake-up Status			TMRn_WKFL	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	b	R/W1C	1	**TimerB Wake-up Event** This flag is set when a wake-up event occurs for TimerB. Write 1 to clear. 0: No event. 1: Wake-up event occurred.
15:1	-	RO	0	**Reserved**
0	a	R/W1C	1	**TimerA Wake-up Event** This flag is set when a wake-up event occurs for TimerA. Write 1 to clear. 0: No event. 1: Wake-up event occurred.

24. Wake-Up Timer (WUT)

The WUT is a unique instance of a 32-bit timer.

- The wake-up timer uses the ERTCO for its clock source.
- Programmable prescaler with values from 1 to 4096.
- Supports three timer modes, all of which can wake the device from low-power modes:
  - One-Shot: The timer counts up to the terminal value, generates a wake-up timer event then halts.
  - Continuous: The timer counts up to the terminal value, generates a wake-up timer event then continues counting.
  - Compare: The timer counts up to the terminal value, generates a wake-up timer event, resets the count, and continues counting.
- Independent interrupt handler (WUT_IRQn).

24.1 Instances

There are two instances of the WUT peripheral in the MAX32690.

24.2 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 always sets the corresponding interrupt flag and generates a wake-up timer interrupt (WUT_IRQn), if enabled.

The timer clock frequency, f_{CNT_CLK}. is a divided version of the 32.768kHz RTC clock as shown in *Equation 24-1*.

Equation 24-1: Wake-Up Timer Clock Frequency

$$f_{CNT\_CLK} = \frac{f_{RTC\_CLK}}{prescaler}$$

The divisor (prescaler) can be set from 1 to 4096 using the concatenated fields *WUTn_CTRL.pres3*:*WUTn_CTRL.pres* as shown in *Table 24-1*.

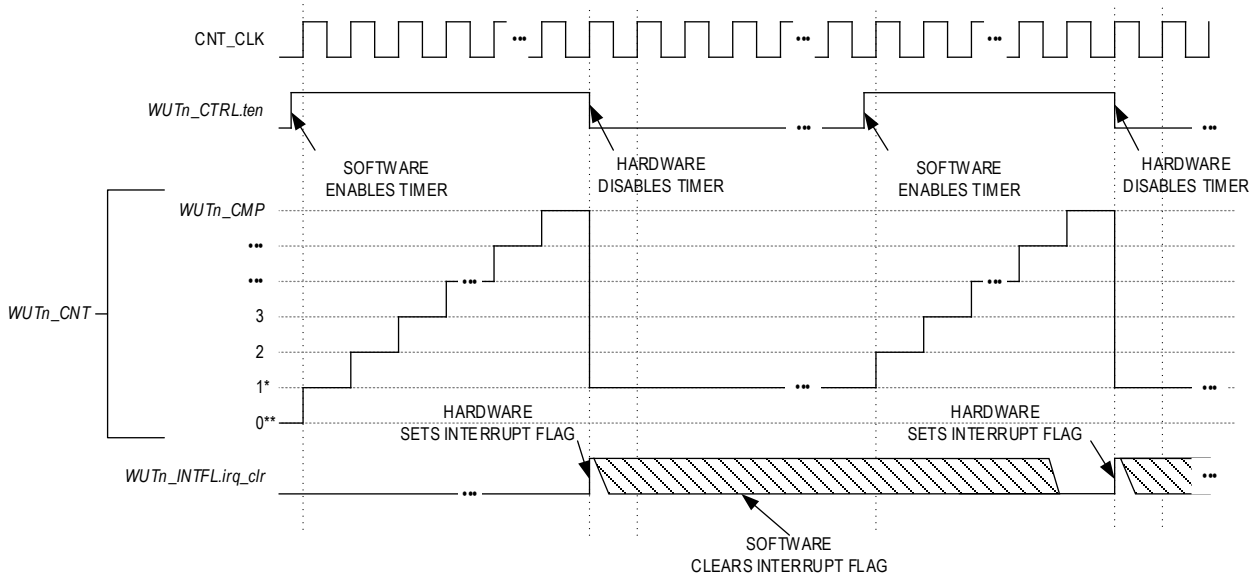
Table 24-1: MAX32690 WUT Clock Period

WUTn_CTRL.pres3	*WUTn_CTRL.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

24.3 One-Shot Mode (0)

In one-shot mode, the timer peripheral increments the *WUTn_CNT* register until it matches the *WUTn_CMP* register, generates a wake-up event, stops incrementing, and disables the timer. In this mode, the timer must be re-enabled to start another one-shot mode event.

Figure 24-1: One-Shot Mode Diagram



* *WUTn_CNT* automatically reloads with 1 at the end of the WUT period, but software can write any initial value to *WUTn_CNT* prior to enabling the timer.

** The default value of *WUTn_CNT* for the first period after a system reset is 0 unless changed by software.

24.3.1 One-Shot Mode Timer Period

The timer period ends on the timer clock when *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 1.
2. The timer is disabled by setting *WUTn_CTRL.ten* = 0.
3. The timer interrupt bit *WUTn_INTFL.irq_clr* is set and wakes up the device if the wake-up timer is enabled as a wake-up event, generating an interrupt.

24.3.2 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* to 0 to select one-shot 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*.*wut_we* to 1.
   a. If desired, register a wake-up interrupt handler (WUT_IRQn).
5. Write an initial value to the *WUTn_CNT* register, if desired. This effects only the first period; subsequent timer periods always reset the *WUTn_CNT* register to 1.
6. Write the compare value to the *WUTn_CMP* register.
7. Clear the wake-up timer interrupt flag by writing 0 to *WUTn_INTFL*.*irq_clr*.
8. Set *WUTn_CTRL*.*ten* to 1 to enable the timer.
9. Enter a low-power sleep mode. See *Operating Modes* for details.

The timer period is calculated using the following equation:

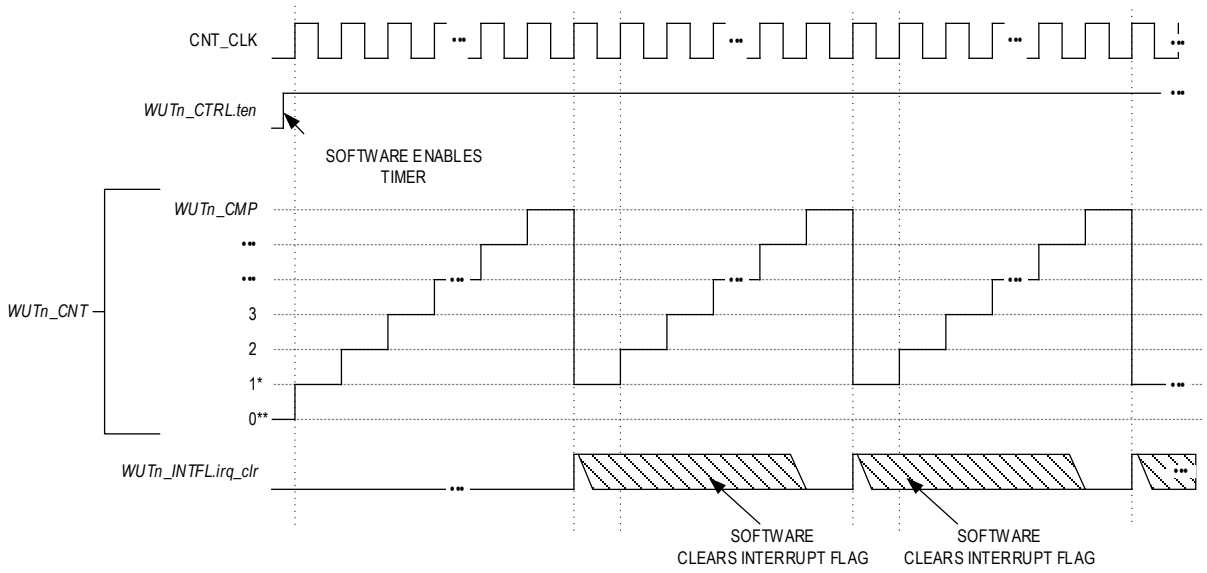
Equation 24-2: One-Shot Mode Timer Period

$$One\text{-}Shot\ mode\ timer\ period\ in\ seconds = \frac{WUTn\_CMP - WUTn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

24.4 Continuous Mode (1)

In continuous mode, the timer peripheral increments the *WUTn_CNT* register until it matches the *WUTn_CMP* register, generates a wake-up event, the hardware resets the *WUTn_CNT* register to 1, and continues incrementing.

Figure 24-2: Continuous Mode Diagram



* *WUTn_CNT* automatically reloads with 1 at the end of the wake-up timer period, but software can write any initial value to *WUTn_CNT* prior to enabling the wake-up timer.

** The value of *WUTn_CNT* for the first period after a system reset is 0 unless changed by software.

24.4.1 Continuous Mode Timer Period

The wake-up timer period ends on the timer clock following *WUTn_CNT* = *WUTn_CMP*.

The wake-up timer peripheral automatically performs the following actions at the end of the timer period:

1. *WUTn_CNT* is reset to 1. The wake-up timer remains enabled and continues incrementing.
2. The timer interrupt bit *WUTn_INTFL.irq_clr* is set. An interrupt is generated if enabled.

24.4.2 Continuous Mode Configuration

Configure the timer for continuous mode by performing the steps following:

1. Set *WUTn_CTRL*.*ten* = 0 to disable the timer.
2. Set *WUTn_CTRL*.*tmode* to 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*.*wut_we* to 1.
   a. If desired, register a wake-up interrupt handler (WUT_IRQn).
5. Write an initial value to the *WUTn_CNT* register, if desired. The initial value is only used for the first period; subsequent timer periods always reset the *WUTn_CNT* register to 1.
6. Write the compare value to the *WUTn_CMP* register.
7. Clear the wake-up timer interrupt flag by writing 0 to *WUTn_INTFL*.*irq_clr*.
8. Set *WUTn_CTRL*.*ten* to 1 to enable the timer.
9. Enter a low-power sleep mode. See *Operating Modes* for details.

The continuous mode timer period is calculated using *Equation 24-3*.

Equation 24-3: Continuous Mode Timer Period

$$Continuous\ Mode\ Timer\ Period\ in\ seconds = \frac{WUTn\_CMP - WUTn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

24.4.3 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.

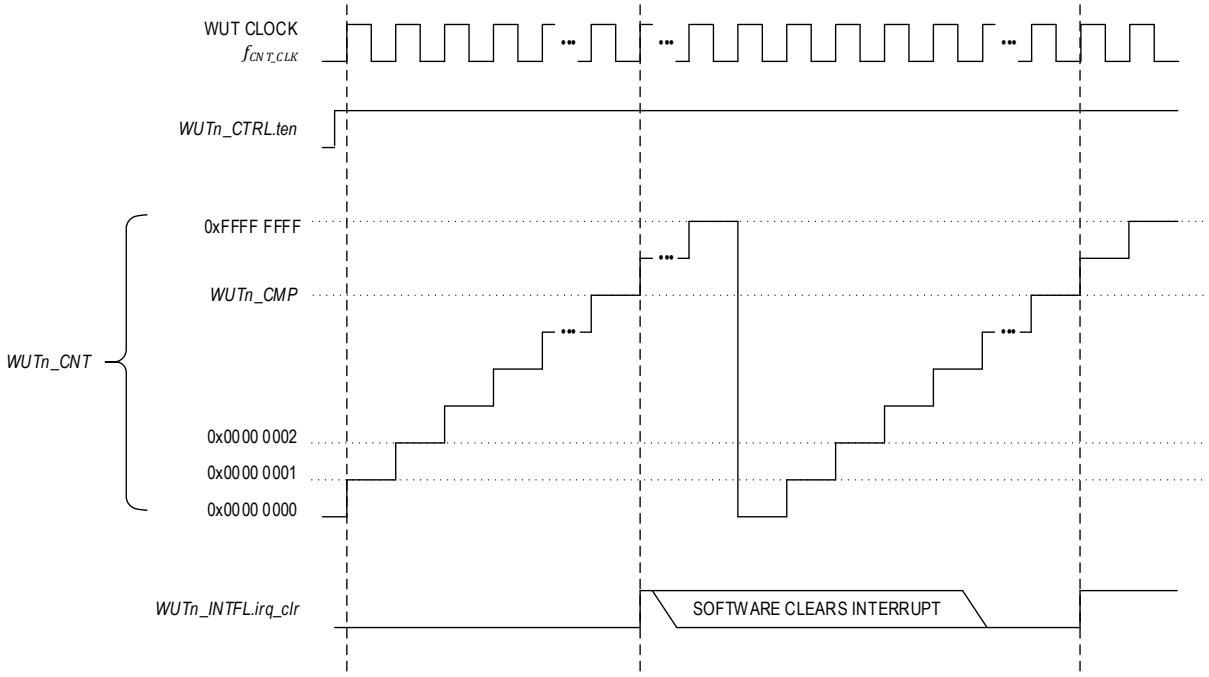
- The timer remains enabled and continues incrementing.

The initial compare mode timer period is calculated using *Equation 24-4*. Subsequent compare mode timer periods are always 0xFFFF FFFF.

Equation 24-4: Compare Mode Timer Initial Period

$$Compare\ mode\ timer\ period\ in\ seconds = \frac{(WUT\_CMP - WUT\_CNT_{INITIAL\_VALUE} + 1)}{f_{CNT\_CLK}(Hz)}$$

Figure 24-3: Compare Mode Diagram



This example uses the following configuration in addition to the settings shown above:
WUTn_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* to 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*.*wut_we* to 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_CNT* register.
7. Clear the wake-up timer interrupt flag by writing 0 to *WUTn_INTFL*.*irq_clr*.
8. Set *WUTn_CTRL*.*ten* to 1 to enable the timer.
9. Enter a low-power sleep mode. See *Operating Modes* for details.

24.5 Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 24-2*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 1-1* 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 24-2: Wake-Up Timer Register Summary

Offset	Register Name	Description
[0x0000]	*WUTn_CNT*	Wake-up Timer Counter Register
[0x0004]	*WUTn_CMP*	Wake-up Timer Compare Register
[0x0008]	*WUTn_PWM*	Wake-up Timer PWM Register
[0x000C]	*WUTn_INTFL*	Wake-up Timer Interrupt Register
[0x0010]	*WUTn_CTRL*	Wake-up Timer Control Register
[0x0014]	*WUTn_NOLCMP*	Wake-up Timer Non-Overlapping Compare Register

24.5.1 Register Details

Table 24-3: Wake-Up Timer Count Register

Wake-Up Timer Count			WUTn_CNT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	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. Before writing this field, disable the timer by clearing the bit *WUTn_CTRL.ten*.

Table 24-4: Wake-Up Timer Compare Register

Wake-Up Timer Compare			WUTn_CMP	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	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 24-5: Wake-Up Timer PWM Register

Wake-Up Timer PWM			WUTn_PWM	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 24-6: Wake-Up Timer Interrupt Register

Wake-Up Timer Interrupt			WUTn_INTFL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	irq_clr	R/W	0	**Timer Interrupt Flag** If set, this field indicates a wake-up timer interrupt condition occurred. Writing any value to this bit clears the wake-up timer's interrupt. 0: Normal operation. 1: Wake-up timer interrupt occurred.

Table 24-7: Wake-Up Timer Control Register

Wake-Up Timer Control			WUTn_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:13	-	DNM	0	**Reserved, Do Not Modify**
12	pwmckbd	DNM	0	**Reserved, Do Not Modify**
11	nollpol	DNM	0	**Reserved, Do Not Modify**
10	nolhpol	DNM	0	**Reserved, Do Not Modify**
9	pwmsync	DNM	0	**Reserved, Do Not Modify**
8	pres3	R/W	0	**Timer Prescaler Select MSB** See *WUTn_CTRL.pres* for details on this field's usage.

Wake-Up Timer Control			WUTn_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
7	ten	R/W	0	**Timer Enable** 0: Timer disable 1: Timer enabled
6	tpol	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 RTC 's 32.768KHz input clock and sets the timer's count clock as shown in *Equation 24-1*. The wake-up timer's prescaler setting is a 4-bit value with *pres3* as the most significant bit and *pres* as the three least significant bits. See *Table 24-1* for details.
2:0	tmode	R/W	0	**Timer Mode Select** Sets the timer's operating mode. 0: One-shot 1: Continuous 2 – 4: Reserved 5: Compare 6 – 7: Reserved

Table 24-8: Wake-Up Timer Non-Overlapping Compare Register

Wake-Up Timer Non-Overlapping Compare			WUTn_NOLCMP	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	DNM	0	**Reserved, Do Not Modify**

25. Watchdog Timer (WDT)

The WDT protects against corrupt or unreliable software, power faults, and other system-level problems that can place the IC into an improper operating state. The software must periodically write a unique sequence to a dedicated register to confirm the application is operating correctly. Failure to reset the watchdog timer within a user-specified time frame can first generate an interrupt allowing the application the opportunity to identify and correct the problem. If the application cannot regain normal operation, the watchdog timer can generate a system reset as a last resort.

Some instances provide a windowed timer function. These instances support an additional feature that can detect watchdog timer resets that occur too early, too late, or never. This could happen if program execution is corrupted and is accidentally forced into a tight loop of code that contains a watchdog sequence. This is not be detected with a traditional WDT because the end of the timeout periods are never reached. A new set of "watchdog timer early" fields are available to support the lower limits required for windowing. Traditional watchdog timers can only detect a loss of program control that fails to reset the watchdog timer.

Each time the application performs a reset as early as possible in the application software, the peripheral control register should be examined to determine if the reset was caused by a WDT late reset event or a WDT early reset event if the window function is enabled. If so, the software should take the desired action as part of its restart sequence.

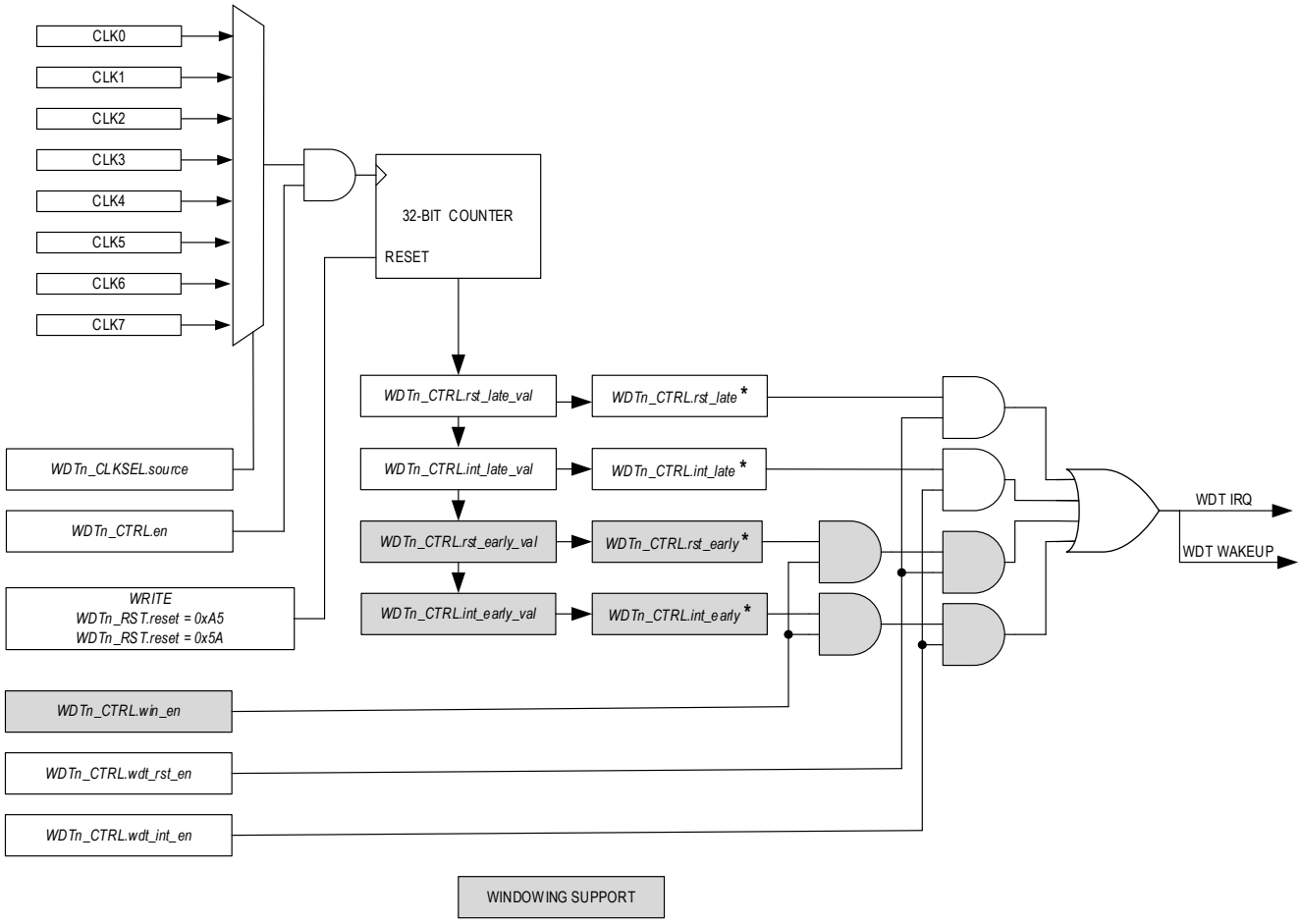
The WDT is a critical safety feature, and most fields are reset on POR or system reset events only.

Features:

- Single-ended (legacy) watchdog timeout
- Windowed mode adds lower-limit timeout settings to detect loss of control in tight code loops.
- Configurable clock source
- Configurable time-base
- Programmable upper and lower limits for reset and interrupts from 2^{16} to 2^{32} time-base ticks.

Figure 25-1 shows a high-level block diagram of the WDT.

Figure 25-1: Windowed Watchdog Timer Block Diagram



* *INTERRUPT FLAGS ARE SET REGARDLESS OF THE ENABLED STATE OF WDTn_CTRL.win_en, WDTn_CTRL.wdt_int_en and WDTn_CTRL.wdt_rst_en.*

25.1 Instances

Table 25-1 shows the peripheral instances, available clock sources, and windowed watchdog support.

Table 25-1: MAX32690 WDT Instances Summary

Instance	Register Access Name	Window Support	CLK0	CLK1	CLK2	CLK3
WDT0	WDT0	Yes	PCLK	IBRO	-	-
LPWDT0	WDT1	Yes	IBRO	INRO	ERTCO	-

25.2 Peripheral Clock Enable

The WDTs are disabled by default on a reset. Use of a WDT requires enabling the peripheral clock for the required watchdog timer. Enable the peripheral clock to a given WDT by setting the disable bit to 0. See *Table 17-2* for each WDT peripheral clock disable bit.

Table 25-2: WDT Peripheral Clock Disable Bits

WDT Port	Disable Bit
WDT0	*GCR_PCLKDIS1.wdt0*
WDT1 (LPWDT0)	*LPGCR_PCLKDIS.wdt1*

25.3 Usage

When enabled, *WDTn_CNT.count* is incremented once every t_{WDTCLK} period. The software periodically executes the feed sequence during correct operation, resetting the *WDTn_CNT.count* field to 0x0000 0000 within the target window.

The upper and lower limits of the target window are user-configurable to accommodate different applications and non-deterministic execution times within an application.

The WDT can generate interrupts and/or reset events in response to the WDT activity. Interrupts are typically configured to respond first to an event outside the target window. The approach is that a minor system event can have temporarily delayed the execution of the feed sequence, so the event can be diagnosed in an interrupt routine and control returned to the system. When the WDT feed sequence occurs much earlier than expected or not at all, a reset event can be generated that forces the system to a known good state before continuing.

Traditional WDTs only detect execution errors that fail to perform the WDT feed sequence. If the counter reaches the WDT late interrupt threshold, the device attempts to regain program control by vectoring to the dedicated WDT interrupt service routine (ISR). The ISR should reset the WDT counter, perform the desired recovery activity, and then return execution to a known good address.

If the execution error prevents the successful execution of the ISR, the WDT continues to increment until the count reaches the WDT late reset threshold. The WDT generates a late reset event that sets the WDT late reset flag and generates a system interrupt.

Instances that support the window feature (*WDTn_CTRL.win_en* = 1) can generate a WDT early interrupt event if the WDT feed sequence occurs earlier than expected. Analogously, the device attempts to regain program control by vectoring to the dedicated WDT ISR. The WDT ISR should reset the WDT counter, perform the desired recovery activity, and then return execution to a known good address.

A WDT feed sequence that occurs earlier than the WDT early reset threshold indicates the execution error is significant enough to initiate a reset of the device to correct the problem. The WDT generates an early reset event that sets the WDT late reset flag and generates a system interrupt.

The event flags are set regardless of the corresponding interrupt or reset enable and include the early interrupt and early event flags, even if the WDT is disabled (*WDTn_CTRL.win_en* = 0).

25.3.1 Using the WDT as a Long-Interval Timer

One application of the WDT is as a very long interval timer in ACTIVE mode. The timer can be configured to generate a WDT late interrupt event for as long as 2^{32} periods of the selected watchdog clock source. The WDT should not be enabled to generate WDT reset events in this application.

25.3.2 Using the WDT as a Long-Interval Wake-up Timer

The WDT can be used as a very long internal wake-up source. Another application of the WDT is as a very long interval wake-up source from *SLEEP*.

25.4 WDT Feed Sequence

The WDT feed sequence protects the system against unintentional altering of the WDT count, and unintentional enabling or disabling of the timer itself.

Two consecutive write instructions to the *WDTn_RST.reset* field are required to reset the *WDTn_CNT.count* = 0. Global interrupts should be disabled immediately before and re-enabled after writing to ensure both writes to the *WDTn_RST.reset* field complete without interruption.

The feed sequence must also be performed immediately before enabling the WDT to prevent accidental triggering of the reset or interrupt as soon as the timer is enabled. There is no timed access window for these write operations; the operations can be separated by any length of time as long as they occur in the required sequence.

1. Disable interrupts.
2. In consecutive write operations:
   a. Write *WDTn_RST.reset*: 0xA5.
   b. Write *WDTn_RST.reset*: 0x5A.
3. If desired, enable or disable the timer.
4. Re-enable interrupts.

25.5 WDT Events

Multiple events are supported, as shown in *Table 25-3*. The corresponding event flag is set when the event occurs.

Typically, the system is configured such that the late interrupt events occur before the late reset events, and early interrupts occur when the feed sequence has the least error from the target time before the early reset events.

The event flags are set even if the corresponding interrupt enable or reset enable are not enabled, and include the early interrupt flag and early event flag even if the window feature is disabled (*WDTn_CTRL.win_en* = 0).

The software must clear the event flags before enabling the WDT.

Table 25-3: WDT Event Summary

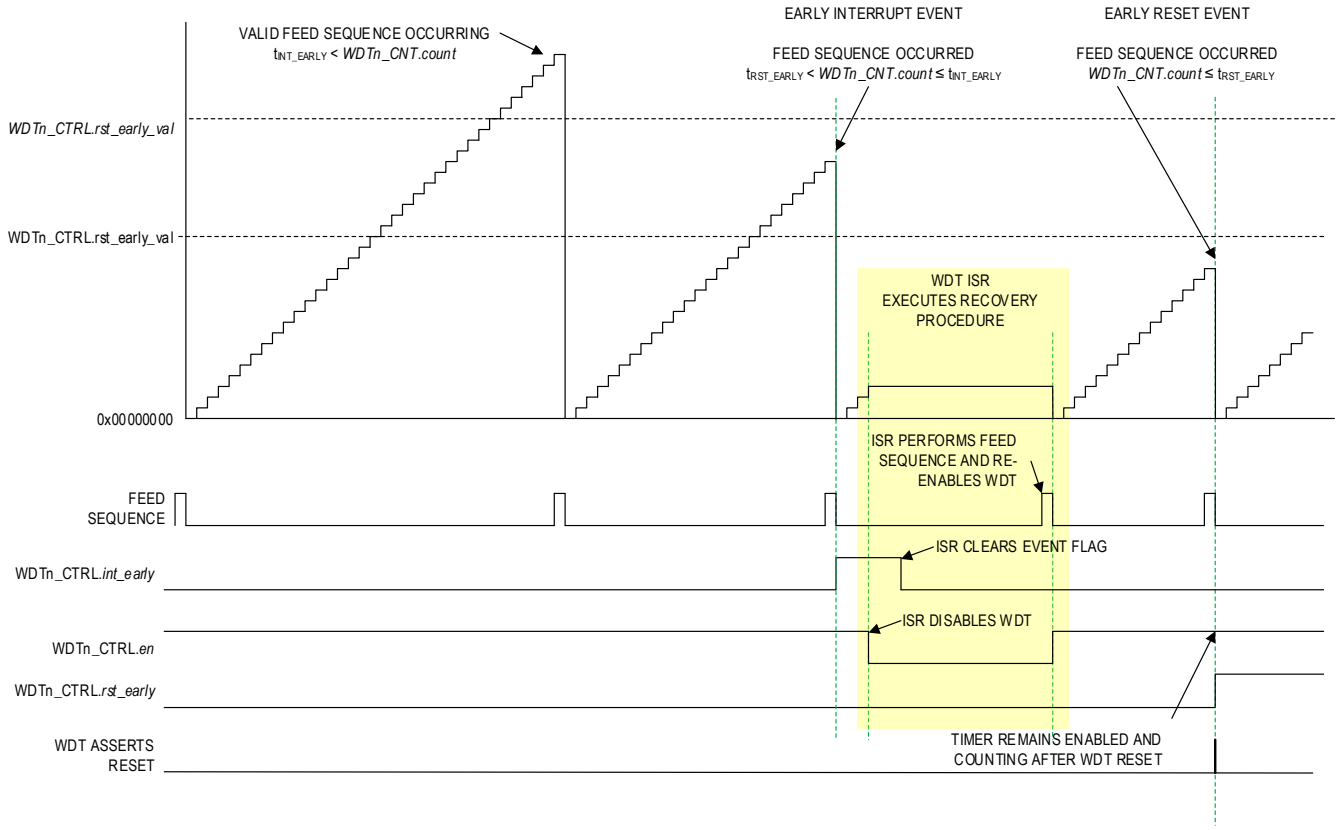
Event	Condition	Peripheral Interrupt Event Flag	Peripheral Interrupt Event Enable
Early Interrupt	Feed sequence occurs while *WDTn_CTRL.rst_early_val* ≤ *WDTn_CNT.count* < *WDTn_CTRL.int_early_val* *WDTn_CTRL.win_en = 1*	*WDTn_CTRL.int_early*	*WDTn_CTRL.wdt_int_en*
Early Reset	Feed sequence occurs while *WDTn_CNT.count* < *WDTn_CTRL.rst_early_val* *WDTn_CTRL.win_en = 1*	*WDTn_CTRL.rst_early*	*WDTn_CTRL.wdt_rst_en*
Interrupt Late	*WDTn_CNT.count* = *WDTn_CTRL.int_late_val*	*WDTn_CTRL.int_late*	*WDTn_CTRL.wdt_int_en*
Reset Late	*WDTn_CNT.count* = *WDTn_CTRL.rst_late_val*	*WDTn_CTRL.rst_late*	*WDTn_CTRL.wdt_rst_en*
Timer Enabled	*WDTn_CTRL.clkrdy* 0 → 1	No event flags are set by a timer enabled event	

25.5.1 WDT Early Reset

The early reset event occurs if the software performs the WDT feed sequence while the WDT count is less than the reset late value (*WDTn_CNT.count* < *WDTn_CTRL.rst_late_val*).

Figure 25-2 shows the sequencing details associated with an early reset event.

Figure 25-2: WDT Early Interrupt and Reset Event Sequencing Details



The following occurs when a WDT early reset event occurs:

1. The hardware sets *WDTn_CTRL.rst_early* to 1.
2. The hardware initiates a system reset.
   a. The hardware resets *WDTn_CNT.count* to 0x0000_0000 during the system reset event.
   b. The *WDTn_CTRL.en* and the *WDTn_CTRL.rst_early* fields are unaffected by a system reset.
3. After the system reset is complete, the WDT continues incrementing.

25.5.2 WDT Early Interrupt

The early interrupt event occurs if the software performs the WDT feed sequence while *WDTn_CTRL.rst_early_val* ≤ *WDTn_CNT.count* < *WDTn_CTRL.int_early_val* as shown in *Table 25-3*. *Figure 25-2* shows the sequencing details associated with an early reset event, including:

- The sequencing details associated with an early interrupt event.
- The required functions performed by the WDT interrupt handler.

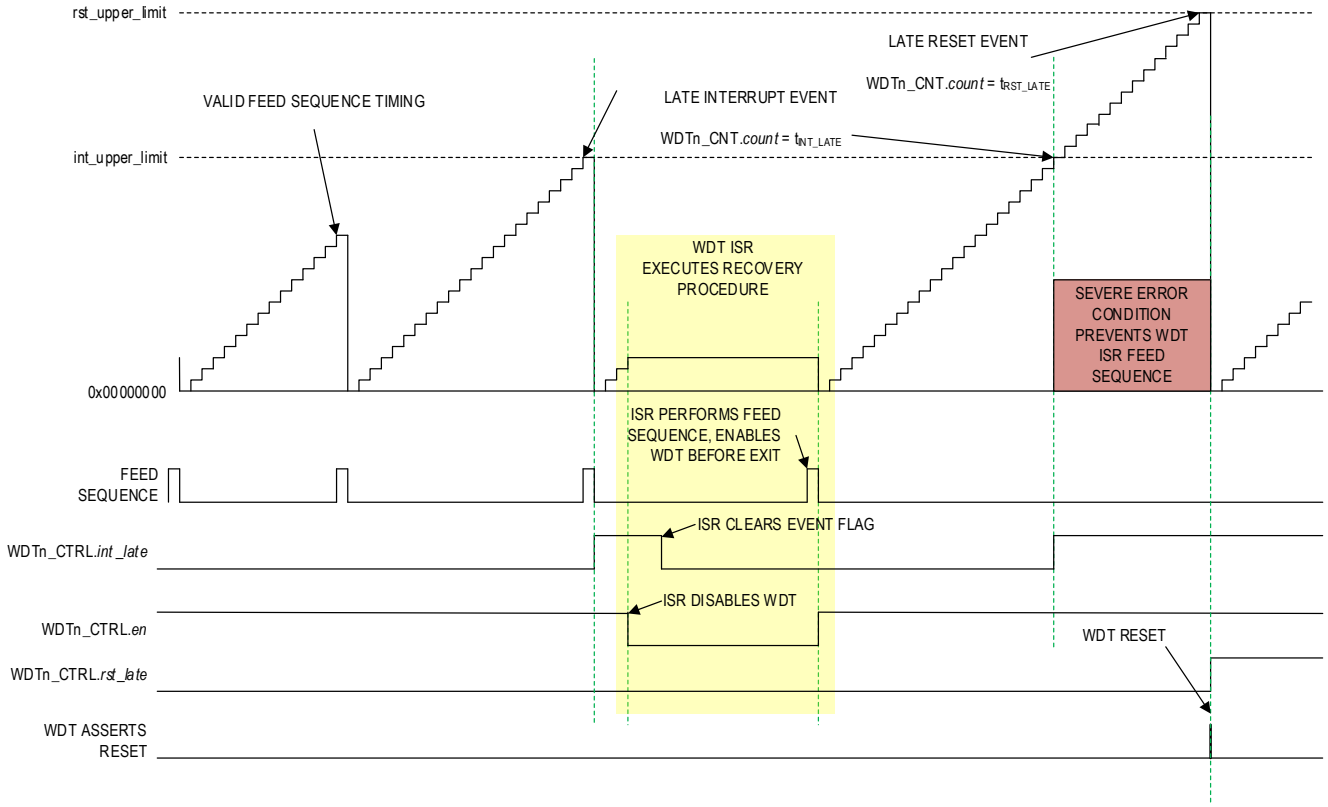
The following occurs when a WDT late interrupt event occurs:

1. The hardware sets *WDTn_CTRL.int_late* to 1.
2. The hardware initiates the WDT interrupt if enabled.

25.5.3 WDT Late Reset

The late reset event occurs if the counter increments to the point where *WDTn_CNT.count* = *WDTn_CTRL.rst_late* threshold as shown in *Table 25-3*. *Figure 25-3* shows the sequencing details associated with a late reset event.

Figure 25-3: WDT Late Interrupt and Reset Event Sequencing Details



The following occurs when a WDT late reset event occurs:

1. The hardware sets *WDTn_CTRL.rst_late* to 1.
2. The hardware initiates a system reset:
   a. The hardware resets *WDTn_CNT.count* to 0x0000 0000 during the reset event.
   b. The *WDTn_CTRL.en* and *WDTn_CTRL.rst_late* fields are unaffected by a system reset.
3. After the hardware exits the system reset, the WDT continues incrementing after the system reset completes.

25.5.4 WDT Late Interrupt

The late reset event occurs if the counter increments to the point where *WDTn_CNT.count* = *WDTn_CTRL.rst_late* threshold as shown in *Table 25-3*. *Figure 25-3* shows the sequencing details associated with a late interrupt event, including the required functions performed by the WDT interrupt handler.

The following occurs when WDT late interrupt event occurs:

1. The hardware sets *WDTn_CTRL.int_late* to 1.
2. The hardware initiates the WDT interrupt if enabled.

25.6 Initializing the WDT

The complete procedure for configuring the WDT is as follows:

1. Execute the WDT feed sequence and disable the WDT:
   a. Disable global interrupts.
   b. Write *WDTn_RST.reset* to 0xA5.
   c. Write *WDTn_RST.reset* to 0x5A.
   d. The hardware resets the WDT count (*WDTn_CNT.count* = 0x0000 0000).
   e. Set *WDTn_CTRL.en* to 0 to disable the WDT.
2. Verify the peripheral is disabled before proceeding:
   a. Poll *WDTn_CTRL.clkrdy* until it reads 1.
3. Set *WDTn_CTRL.clkrdy_ie* = 1 to generate a WDT enabled interrupt event.
4. Re-enable global interrupts.
5. Configure *WDTn_CLKSEL.source* to select the clock source.
6. Configure the standard thresholds:
   a. Configure *WDTn_CTRL.int_late* to the desired threshold for the WDT late interrupt event.
   b. Configure *WDTn_CTRL.rst_late_val* to the desired threshold for the WDT late reset event.
7. If using the optional windowed WDT feature:
   a. Set *WDTn_CTRL.win_en* = 1 to enable the windowed WDT feature.
   b. Configure *WDTn_CTRL.int_early_val* to the desired threshold for the WDT early interrupt event.
   c. Configure *WDTn_CTRL.rst_early_val* to the desired threshold for the WDT early reset event.
8. Set *WDTn_CTRL.wdt_int_en* to generate an interrupt when a WDT late interrupt event occurs. If *WDTn_CTRL.win_en* = 1, an interrupt is generated by both a WDT late interrupt event, and a WDT early interrupt event.
9. Set *WDTn_CTRL.wdt_rst_en* to generate an interrupt when a WDT late reset event occurs. If *WDTn_CTRL.win_en* = 1, an interrupt is generated by a WDT late reset event and a WDT early reset event.
10. Execute the WDT feed sequence and enable the WDT:
    a. Disable global interrupts.
    b. Write *WDTn_RST.reset* to 0xA5.
    c. Write *WDTn_RST.reset* to 0x5A. The hardware resets *WDTn_CNT.count* = 0x0000 0000.
    d. Set *WDTn_CTRL.en* to 1 to enable the WDT.
11. Verify the peripheral is enabled before proceeding:
    a. Poll *WDTn_CTRL.clkrdy* until it reads 1, or
12. Set *WDTn_CTRL.clkrdy_ie* = 1 to generate a WDT enabled event interrupt.
13. Re-enable global interrupts.

25.7 Resets

The WDT is a critical safety feature. Most of the fields are reset by a POR or system reset events only; however, the enable field (*WDTn_CTRL.en*) and the interrupt flag fields are not reset by a system reset event.

25.8 Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 25-4*. 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 1-1* 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 25-4: WDT Register Summary

Offset	Register	Name
[0x0000]	*WDTn_CTRL*	WDT Control Register
[0x0004]	*WDTn_RST*	WDT Reset Register
[0x0008]	*WDTn_CLKSEL*	WDT Clock Select Register
[0x000C]	*WDTn_CNT*	WDT Count Register

25.8.1 Register Details

Table 25-5: WDT Control Register

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31	rst_late	R/W	0	**Reset Late Event** A watchdog reset event occurred after the time specified in *WDTn_CTRL.rst_late_val*. This flag is set even if *WDTn_CTRL.win_en* = 0 or *WDTn_CTRL.wdt_rst_en* = 0. The software must clear this field to 0. 0: Watchdog did not cause a reset event. 1: Watchdog reset occurred after *WDTn_CTRL.rst_early_val*.
30	rst_early	R/W	0	**Reset Early Event** A watchdog reset event occurred before the time specified in the *WDTn_CTRL.rst_early_val* field. This flag is set even if *WDTn_CTRL.win_en* = 0 or *WDTn_CTRL.wdt_rst_en* = 0. The software must clear this field to 0. 0: Watchdog did not cause a reset event. 1: Watchdog reset occurred before the time specified in the *WDTn_CTRL.rst_early_val* field.
29	win_en	R/W	0	**Window Function Enable** 0: Disabled. The WDT recognizes interrupt late and reset late events, supporting legacy implementations. 1: Enabled.
28	clkrdy	R	0	**Clock Status** This field is cleared to 0 by the hardware when the software changes the state of the *WDTn_CTRL.en* field. The hardware sets this field to 1 when the change to the requested enable or disable is complete. 0: WDT clock is off. 1: WDT clock is on.
27	clkrdy_ie	R/W	0	**Clock Switch Ready Interrupt Enable** This interrupt prevents the software from needing to poll the *WDTn_CTRL.clkrdy* field to determine when the WDT clock is ready. When the *WDTn_CTRL.clkrdy* field transitions from 1 to 0, this interrupt signals the transition is complete. 0: Disabled. 1: Enabled.
26:24	-	RO	0	**Reserved**

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
23:20	rst_early_val	R/W	0	**Reset Early Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$ *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*
19:16	int_early_val	R/W	0	**Interrupt Early Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$ *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*
15:13	-	RO	0	**Reserved**
12	int_early	R/W	0	**Interrupt Early Flag** A feed sequence was performed earlier than the time determined by the *WDTn_CTRL.int_early* field. This flag is set even if *WDTn_CTRL.win_en* = 0. 0: No interrupt event. 1: Interrupt event occurred. *Note: A WDT interrupt is generated if the WDT interrupt is enabled (WDTn_CTRL.wdt_int_en = 1).*
11	wdt_rst_en	R/W	0	**WDT Reset Enable** 0: Disabled. 1: Enabled.
10	wdt_int_en	R/W	0	**WDT Interrupt Enable** 0: Disabled. 1: Enabled.

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
9	int_late	R/W	0	**Interrupt Late Flag** A watchdog feed sequence did not occur before the time determined by the *WDTn_CTRL.int_late_val* field. 0: No interrupt event. 1: Interrupt event occurred. *Note: A WDT interrupt is generated if the WDT interrupt is enabled (WDTn_CTRL.wdt_int_en = 1).*
8	en	R/W	0	**WDT Enable** This field enables/disables the WDT clock into the peripheral. *WDTn_CNT.count* holds its value while the WDT is disabled. The WDT feed sequence must be performed immediately before any change to this field. 0: Disabled. 1: Enabled.
7:4	rst_late_val	R/W	0	**Reset Late Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$ *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*
3:0	int_late_val	R/W	0	**Interrupt Late Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$ *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*

Table 25-6: WDT Reset Register

WDT Reset			WDTn_RST	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	reset	R/W	0†	**Reset Watchdog Timer Count** Writing the WDT feed sequence in two consecutive write instructions to this register resets the internal counter to 0x0000 0000. 1. Write *WDTn_RST.reset*: 0xA5. 2. Write *WDTn_RST.reset*: 0x5A. Writes to the *WDTn_CTRL.en* field, which enables or disables the WDT, must be the next instruction following the WDT feed sequence. *†Note: This field is set to 0 on a POR and is not affected by other resets.*

Table 25-7: WDT Clock Source Select Register

WDT Clock Source Select			WDTn_CLKSEL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2:0	source	R/W	0†	**Clock Source Select** See *Table 25-1* for the available clock options. 0: CLK0. 1: CLK1. 2: CLK2. 3: CLK3. 4: CLK4. 5: CLK5. 6: CLK6. 7: CLK7. *†Note: This field is only reset on a POR and unaffected by other resets.* *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*

Table 25-8: WDT Count Register

WDT Count			WDTn_CNT	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	count	R	0	**WDT Counter** The counter value for debugging. This register is reset by system reset, as well as the watchdog feeding sequence. When the WDT clock is off, the feeding sequence generates an asynchronous reset of 1 PCLK width. When the WDT clock is on, the feeding sequence generates a synchronous reset that is a handshake to the WDT clock domain. *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before reading this field.*

26. 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 PCLK.

26.1 Instances

The device provides 16 instances of the pulse train engine peripheral.

- PT0 to PT15.

All peripheral registers share a common register set.

26.2 Peripheral Clock Enable

The PT is disabled by default on a reset. Use of the PT peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS0*.*pt* to 0 to enable the peripheral clock to the PT before configuring the PT for use.

26.3 Features

The pulse train outputs with individually programmable modes, patterns, and output enables. The pulse train engine uses the 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.

26.4 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.

26.4.1 Pulse Train Output Modes

Each pulse train output supports the following modes:

- Pulse train
- Bit pattern length
- Square wave

26.4.1.1 Pulse Train Mode

When pulse train *n* (*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* as follows:

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* configured for pulse train mode, 32-bit pattern)

26.4.1.2 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 LSB first. If the output is configured in square wave mode, then the *PTn_TRAIN* register is ignored.

Equation 26-1: Pulse Train Mode Output Function

$$PTn\_TRAIN = [Bit\ pattern\ for\ PTn]$$

26.4.1.3 Synchronize Two or More Outputs, if Needed

The write-only register *PTG_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.

26.4.1.4 Pulse Train Loop Mode

By default, a pulse train engine runs indefinitely until it is disabled by the software. 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.

26.4.1.5 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.

26.4.1.6 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:

- 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*.

26.5 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: Before changing a pulse train output's configuration, the corresponding pulse train output should be halted to prevent unexpected behavior.

26.6 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 (*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 (*PTG_SAFE_EN*) and disable (*PTG_SAFE_DIS*) the outputs.

26.6.1 Pulse Train Atomic Enable

The global safe enable register, *PTG_SAFE_EN*, 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.

26.6.2 Pulse Train Atomic Disable

The global safe disable register, *PTG_SAFE_DIS*, 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.

26.7 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* is initialized to a non-zero value, and the loop count reaches 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.

26.8 Interrupts

Each pulse train can generate an interrupt only if it is configured in pulse train mode, and the loop counter, *PTn_LOOP*, is initialized to a non-zero number. When *PTn_LOOP* 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.

26.9 Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 26-1: Pulse Train Engine Register Summary

Offset	Register	Description
[0x0000]	*PTG_ENABLE*	PT Global Enable/Disable Control Register
[0x0004]	*PTG_RESYNC*	PT Global Resync Register
[0x0008]	*PTG_INTFL*	PT Stopped Global Status Flags Register
[0x000C]	*PTG_INTEN*	PT Global Interrupt Enable Register
[0x0010]	*PTG_SAFE_EN*	PT Global Safe Enable Register
[0x0014]	*PTG_SAFE_DIS*	PT Global Safe Disable Register
[0x0020]	*PT0_RATE_LENGTH*	PT0 Configuration Register
[0x0024]	*PT0_TRAIN*	PT0 Pulse Train Mode Bit Pattern Register
[0x0028]	*PT0_LOOP*	PT0 Loop Control Register
[0x002C]	*PT0_RESTART*	PT0 Automatic Restart Register
[0x0030]	*PT1_RATE_LENGTH*	PT1 Configuration Register
[0x0034]	*PT1_TRAIN*	PT1 Pulse Train Mode Bit Pattern Register
[0x0038]	*PT1_LOOP*	PT1 Loop Control Register
[0x003C]	*PT1_RESTART*	PT1 Automatic Restart Register
[0x0040]	*PT2_RATE_LENGTH*	PT2 Configuration Register
[0x0044]	*PT2_TRAIN*	PT2 Pulse Train Mode Bit Pattern Register
[0x0048]	*PT2_LOOP*	PT2 Loop Control Register
[0x004C]	*PT2_RESTART*	PT2 Automatic Restart Register
[0x0050]	*PT3_RATE_LENGTH*	PT3 Configuration Register
[0x0054]	*PT3_TRAIN*	PT3 Pulse Train Mode Bit Pattern Register
[0x0058]	*PT3_LOOP*	PT3 Loop Control Register
[0x005C]	*PT3_RESTART*	PT3 Automatic Restart Register
[0x0060]	*PT4_RATE_LENGTH*	PT4 Configuration Register
[0x0064]	*PT4_TRAIN*	PT4 Pulse Train Mode Bit Pattern Register
[0x0068]	*PT4_LOOP*	PT4 Loop Control Register

Offset	Register	Description
[0x006C]	*PT4_RESTART*	PT4 Automatic Restart Register
[0x0070]	*PT5_RATE_LENGTH*	PT5 Configuration Register
[0x0074]	*PT5_TRAIN*	PT5 Pulse Train Mode Bit Pattern Register
[0x0078]	*PT5_LOOP*	PT5 Loop Control Register
[0x007C]	*PT5_RESTART*	PT5 Automatic Restart Register
[0x0080]	*PT6_RATE_LENGTH*	PT6 Configuration Register
[0x0084]	*PT6_TRAIN*	PT6 Pulse Train Mode Bit Pattern Register
[0x0088]	*PT6_LOOP*	PT6 Loop Control Register
[0x008C]	*PT6_RESTART*	PT6 Automatic Restart Register
[0x0090]	*PT7_RATE_LENGTH*	PT7 Configuration Register
[0x0094]	*PT7_TRAIN*	PT7 Pulse Train Mode Bit Pattern Register
[0x0098]	*PT7_LOOP*	PT7 Loop Control Register
[0x009C]	*PT7_RESTART*	PT7 Automatic Restart Register
[0x00A0]	*PT8_RATE_LENGTH*	PT8 Configuration Register
[0x00A4]	*PT8_TRAIN*	PT8 Pulse Train Mode Bit Pattern Register
[0x00A8]	*PT8_LOOP*	PT8 Loop Control Register
[0x00AC]	*PT8_RESTART*	PT8 Automatic Restart Register
[0x00B0]	*PT9_RATE_LENGTH*	PT9 Configuration Register
[0x00B4]	*PT9_TRAIN*	PT9 Pulse Train Mode Bit Pattern Register
[0x00B8]	*PT9_LOOP*	PT9 Loop Control Register
[0x00BC]	*PT9_RESTART*	PT9 Automatic Restart Register
[0x00C0]	*PT10_RATE_LENGTH*	PT10 Configuration Register
[0x00C4]	*PT10_TRAIN*	PT10 Pulse Train Mode Bit Pattern Register
[0x00C8]	*PT10_LOOP*	PT10 Loop Control Register
[0x00CC]	*PT10_RESTART*	PT10 Automatic Restart Register
[0x00D0]	*PT11_RATE_LENGTH*	PT11 Configuration Register
[0x00D4]	*PT11_TRAIN*	PT11 Pulse Train Mode Bit Pattern Register
[0x00D8]	*PT11_LOOP*	PT11 Loop Control Register
[0x00DC]	*PT11_RESTART*	PT11 Automatic Restart Register
[0x00E0]	*PT12_RATE_LENGTH*	PT12 Configuration Register
[0x00E4]	*PT12_TRAIN*	PT12 Pulse Train Mode Bit Pattern Register
[0x00E8]	*PT12_LOOP*	PT12 Loop Control Register
[0x00EC]	*PT12_RESTART*	PT12 Automatic Restart Register
[0x00F0]	*PT13_RATE_LENGTH*	PT13 Configuration Register
[0x00F4]	*PT13_TRAIN*	PT13 Pulse Train Mode Bit Pattern Register
[0x00F8]	*PT13_LOOP*	PT13 Loop Control Register
[0x00FC]	*PT13_RESTART*	PT13 Automatic Restart Register
[0x0100]	*PT14_RATE_LENGTH*	PT14 Configuration Register
[0x0104]	*PT14_TRAIN*	PT14 Pulse Train Mode Bit Pattern Register
[0x0108]	*PT14_LOOP*	PT14 Loop Control Register
[0x010C]	*PT14_RESTART*	PT14 Automatic Restart Register
[0x0110]	*PT15_RATE_LENGTH*	PT15 Configuration Register
[0x0114]	*PT15_TRAIN*	PT15 Pulse Train Mode Bit Pattern Register
[0x0118]	*PT15_LOOP*	PT15 Loop Control Register
[0x011C]	*PT15_RESTART*	PT15 Automatic Restart Register

26.9.1 Register Details

26.9.1.1 Pulse Train Engine Global Enable/Disable Register

This register enables each of the individual pulse trains. Write a 1 to the individual pulse train enable bits to enable the corresponding pulse train. When, for a given pulse train, the *PTn_LOOP.count* loop counter is set to a non-zero number,

when the loop counter reaches zero, then the given pulse train engine stops, and the corresponding enable bit in this register is cleared by the hardware.

Table 26-2: 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**
15	pt15	R/W	0	**Enable PT15** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
14	pt14	R/W	0	**Enable PT14** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
13	pt13	R/W	0	**Enable PT13** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
12	pt12	R/W	0	**Enable PT12** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
11	pt11	R/W	0	**Enable PT11** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
10	pt10	R/W	0	**Enable PT10** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
9	pt9	R/W	0	**Enable PT9** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
8	pt8	R/W	0	**Enable PT8** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
7	pt7	R/W	0	**Enable PT7** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
6	pt6	R/W	0	**Enable PT6** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
5	pt5	R/W	0	**Enable PT5** 0: Disabled. 1: Enabled. *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**
4	pt4	R/W	0	**Enable PT4** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
3	pt3	R/W	0	**Enable PT3** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
2	pt2	R/W	0	**Enable PT2** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
1	pt1	R/W	0	**Enable PT1** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
0	pt0	R/W	0	**Enable PT0** 0: Disabled. 1: Enabled. *Note: Disabling an active pulse train halts the output and does not generate a stop event.*

Table 26-3: Pulse Train Engine Resync Register

PT Resync Register			PTG_RESYNC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
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. 0: No effect 1: Reset/restart the pulse train. *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. 0: No effect 1: Reset/restart the pulse train. *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. 0: No effect 1: Reset/restart the pulse train. *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*

PT Resync Register			PTG_RESYNC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
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. 0: No effect 1: Reset/restart the pulse train. *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. 0: No effect 1: Reset/restart the pulse train. *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
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. 0: No effect 1: Reset/restart the pulse train. *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. 0: No effect 1: Reset/restart the pulse train. *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. 0: No effect 1: Reset/restart the pulse train. *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. 0: No effect 1: Reset/restart the pulse train. *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*

PT Resync Register			PTG_RESYNC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
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. 0: No effect 1: Reset/restart the pulse train. *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
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. 0: No effect 1: Reset/restart the pulse train. *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. 0: No effect 1: Reset/restart the pulse train. *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. 0: No effect 1: Reset/restart the pulse train. *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. 0: No effect 1: Reset/restart the pulse train. *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. 0: No effect 1: Reset/restart the pulse train. *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. 0: No effect 1: Reset/restart the pulse train. *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*

Table 26-4: 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**
15	pt15	R/W1C	0	**PT15 Stopped Status Flag** This bit is set to 1 by the 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 the 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 the 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 the 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 the 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 the 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.
9	pt9	R/W1C	0	**PT9 Stopped Status Flag** This bit is set to 1 by the 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**
8	pt8	R/W1C	0	**PT8 Stopped Status Flag** This bit is set to 1 by the 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 the 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 the 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 the 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 the 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 the 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 the 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 the 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.
0	pt0	R/W1C	0	**PT0 Stopped Status Flag** This bit is set to 1 by the 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 26-5: Pulse Train Engine Interrupt Enable Register

PT Interrupt Enable Register			PTG_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**

PT Interrupt Enable Register			PTG_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
15	pt15	R/W	0	**PT15 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
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
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

PT Interrupt Enable Register			PTG_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
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

26.9.1.2 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 26-6: 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**
15	pt15	WO	-	**Safe Enable Control for PT15** Writing a 1 to this field sets the *PTG_ENABLE.pt15* field to 1. 0: No effect. 1: Enable corresponding pulse train.
14	pt14	WO	-	**Safe Enable Control for PT14** Writing a 1 to this field sets the *PTG_ENABLE.pt14* field to 1. 0: No effect. 1: Enable corresponding pulse train.
13	pt13	WO	-	**Safe Enable Control for PT13** Writing a 1 to this field sets the *PTG_ENABLE.pt13* field to 1. 0: No effect. 1: Enable corresponding pulse train.

Pulse Train Engine Safe Enable Register		PTG_SAFE_EN		[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
12	pt12	WO	-	**Safe Enable Control for PT12** Writing a 1 to this field sets the *PTG_ENABLE.pt12* field to 1. 0: No effect. 1: Enable corresponding pulse train.
11	pt11	WO	-	**Safe Enable Control for PT11** Writing a 1 to this field sets the *PTG_ENABLE.pt11* field to 1. 0: No effect. 1: Enable corresponding pulse train.
10	pt10	WO	-	**Safe Enable Control for PT10** Writing a 1 to this field sets the *PTG_ENABLE.pt10* field to 1. 0: No effect. 1: Enable corresponding pulse train.
9	pt9	WO	-	**Safe Enable Control for PT9** Writing a 1 to this field sets the *PTG_ENABLE.pt9* field to 1. 0: No effect. 1: Enable corresponding pulse train.
8	pt8	WO	-	**Safe Enable Control for PT8** Writing a 1 to this field sets the *PTG_ENABLE.pt8* field to 1. 0: No effect. 1: Enable corresponding pulse train.
7	pt7	WO	-	**Safe Enable Control for PT7** Writing a 1 to this field sets the *PTG_ENABLE.pt7* field to 1. 0: No effect. 1: Enable corresponding pulse train.
6	pt6	WO	-	**Safe Enable Control for PT6** Writing a 1 to this field sets the *PTG_ENABLE.pt6* field to 1. 0: No effect. 1: Enable corresponding pulse train.
5	pt5	WO	-	**Safe Enable Control for PT5** Writing a 1 to this field sets the *PTG_ENABLE.pt5* field to 1. 0: No effect. 1: Enable corresponding pulse train.
4	pt4	WO	-	**Safe Enable Control for PT4** Writing a 1 to this field sets the *PTG_ENABLE.pt4* field to 1. 0: No effect. 1: Enable corresponding pulse train.
3	pt3	WO	-	**Safe Enable Control for PT3** Writing a 1 to this field sets the *PTG_ENABLE.pt3* field to 1. 0: No effect. 1: Enable corresponding pulse train.
2	pt2	WO	-	**Safe Enable Control for PT2** Writing a 1 to this field sets the *PTG_ENABLE.pt2* field to 1. 0: No effect. 1: Enable corresponding pulse train.
1	pt1	WO	-	**Safe Enable Control for PT1** Writing a 1 to this field sets the *PTG_ENABLE.pt1* field to 1. 0: No effect. 1: Enable corresponding pulse train.

Pulse Train Engine Safe Enable Register		PTG_SAFE_EN		[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
0	pt0	WO	-	**Safe Enable Control for PT0** Writing a 1 to this field sets the *PTG_ENABLE.pt0* field to 1. 0: No effect. 1: Enable corresponding pulse train.

26.9.1.3 Pulse Train Engine Safe Disable 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 26-7: 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**
15	pt15	WO	-	**Safe Disable Control for PT15** Writing a 1 to this field clears the *PTG_ENABLE.pt15* field. 0: No effect. 1: Disable corresponding pulse train.
14	pt14	WO	-	**Safe Disable Control for PT14** Writing a 1 to this field clears the *PTG_ENABLE.pt14* field. 0: No effect. 1: Disable corresponding pulse train.
13	pt13	WO	-	**Safe Disable Control for PT13** Writing a 1 to this field clears the *PTG_ENABLE.pt13* field. 0: No effect. 1: Disable corresponding pulse train.
12	pt12	WO	-	**Safe Disable Control for PT12** Writing a 1 to this field clears the *PTG_ENABLE.pt12* field. 0: No effect. 1: Disable corresponding pulse train.
11	pt11	WO	-	**Safe Disable Control for PT11** Writing a 1 to this field clears the *PTG_ENABLE.pt11* field. 0: No effect. 1: Disable corresponding pulse train.
10	pt10	WO	-	**Safe Disable Control for PT10** Writing a 1 to this field clears the *PTG_ENABLE.pt10* field. 0: No effect. 1: Disable corresponding pulse train.
9	pt9	WO	-	**Safe Disable Control for PT9** Writing a 1 to this field clears the *PTG_ENABLE.pt9* field. 0: No effect. 1: Disable corresponding pulse train.
8	pt8	WO	-	**Safe Disable Control for PT8** Writing a 1 to this field clears the *PTG_ENABLE.pt8* field. 0: No effect. 1: Disable corresponding pulse train.
7	pt7	WO	-	**Safe Disable Control for PT7** Writing a 1 to this field clears the *PTG_ENABLE.pt7* field. 0: No effect. 1: Disable corresponding pulse train.

Pulse Train Engine Safe Disable Register		PTG_SAFE_DIS		[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
6	pt6	WO	-	**Safe Disable Control for PT6** Writing a 1 to this field clears the *PTG_ENABLE.pt6* field. 0: No effect. 1: Disable corresponding pulse train.
5	pt5	WO	-	**Safe Disable Control for PT5** Writing a 1 to this field clears the *PTG_ENABLE.pt5* field. 0: No effect. 1: Disable corresponding pulse train.
4	pt4	WO	-	**Safe Disable Control for PT4** Writing a 1 to this field clears the *PTG_ENABLE.pt4* field. 0: No effect. 1: Disable corresponding pulse train.
3	pt3	WO	-	**Safe Disable Control for PT3** Writing a 1 to this field clears the *PTG_ENABLE.pt3* field. 0: No effect. 1: Disable corresponding pulse train.
2	pt2	WO	-	**Safe Disable Control for PT2** Writing a 1 to this field clears the *PTG_ENABLE.pt2* field. 0: No effect. 1: Disable corresponding pulse train.
1	pt1	WO	-	**Safe Disable Control for PT1** Writing a 1 to this field clears the *PTG_ENABLE.pt1* field. 0: No effect. 1: Disable corresponding pulse train.
0	pt0	WO	-	**Safe Disable Control for PT0** Writing a 1 to this field clears the *PTG_ENABLE.pt0* field. 0: No effect. 1: Disable corresponding pulse train.

26.9.1.4 Pulse Train Registers

Table 26-8: Pulse Train Engine Configuration Register

Pulse Train *n* Configuration Register			PTn_RATE_LENGTH	*
Bits	**Field**	**Access**	**Reset**	**Description**
31:27	mode	R/W	1	**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. …: … 31: Pulse train mode, 31 bits long. *Note: If this field is set to 1, square wave mode, the PTn_TRAIN register is not used.*

Pulse Train *n* Configuration Register			PTn_RATE_LENGTH	*
Bits	**Field**	**Access**	**Reset**	**Description**
26:0	rate_control	R/W	0	**Pulse Train Enable and Rate Control** Defines the rate at which the output for PT*n* changes state by setting the divisor of the PT clock. Setting this field to 0 disables the PT*n*. For all other values, the following equation is used to calculate the rate: $f_{PTn} = \frac{f_{PTE\_CLK}}{rate\_control}$ 0: Output halted. 1: $f_{PTn} = f_{PTE\_CLK}$ 2: $f_{PTn} = {f_{PTE\_CLK}}/{2}$ 3: $f_{PTn} = {f_{PTE\_CLK}}/{3}$ …

Table 26-9: Pulse Train Mode Bit Pattern Register

Pulse Train Mode Bit Pattern			PTn_TRAIN	*
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 0000 and 0x0001 0000 are invalid values for this register.*

Table 26-10: Pulse Train n Loop Configuration Register

Pulse Train Loop Configuration			PTn_LOOP	*
Bits	**Field**	**Access**	**Reset**	**Description**
31:28	-	RO	0	**Reserved**
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 *PTn_LOOP.count* is decremented after the delay. *Note: This field is ignored if firmware writes 0 to PTn_LOOP.count field.*
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 pulse train's interrupt flag is set in the *PTG_INTFL* register. Writing this field to 0 to repeat the pulse train pattern indefinitely. *Note: This field is ignored in square wave mode.*

Table 26-11: Pulse Train n Automatic Restart Configuration Register

Pulse Train Automatic Restart Configuration			PTn_RESTART	*
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
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.
14:13	-	RO	0	**Reserved**

Pulse Train Automatic Restart Configuration			PTn_RESTART	*
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. 0: PT0. 1: PT1. 2: PT2. 3: PT3. 4: PT4. 5: PT5. 6: PT6. 7: PT7. 8: PT8. 9: PT9. 10: PT10. 11: PT11. 12: PT12. 13: PT13. 14: PT14. 15: PT15. 16- 31: 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 the pulse train from the beginning of its pattern.
6:5	-	RO	0	**Reserved**
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. 0: PT0. 1: PT1. 2: PT2. 3: PT3. 4: PT4. 5: PT5. 6: PT6. 7: PT7. 8: PT8. 9: PT9. 10: PT10. 11: PT11. 12: PT12. 13: PT13. 14: PT14. 15: PT15. 16- 31: Reserved.

27. Cryptographic Toolbox (CTB)

The cryptographic toolbox combines cryptographic engines to provide advanced cryptographic security. The dedicated hardware engines significantly improve the speed of computationally intensive cryptographic algorithms.

It is critical to note that the requirements for meeting specific security validations are frequently updated. Contact Analog Devices before starting the design of any product using the physical or cryptographic security features of this device to determine compatibility with the most recent requirements.

The CTB provides the following features:

- Dedicated cryptographic DMA (CDMA) engine which provides high-speed data transfers to and from the cryptographic engines

Symmetric block cipher engine supporting:

- AES-128, 192, and 256 (FIPS 197).

CCM/GCM

- Support for NIST-approved block modes (SP800-38).

Parallel calculation of block cipher and hash functions

- Hash function accelerator supporting functions commonly used in CMAC and HMAC:
  - SHA-1

  SHA-224
  - SHA- 256

  SHA-384
  - SHA-512 (FIPS 180-3) values used in CMAC and HMAC.
- Hamming code engine calculates hamming codes used in memory ECC algorithms.

Most functions are configurable for big- or little-endian operations.

All cryptographic operations begin by resetting the CTB. The cryptographic engine functions have their own done bit and a global done bit for the cryptographic block. The engines can generate an interrupt if enabled.

27.1 Peripheral Clock Enable

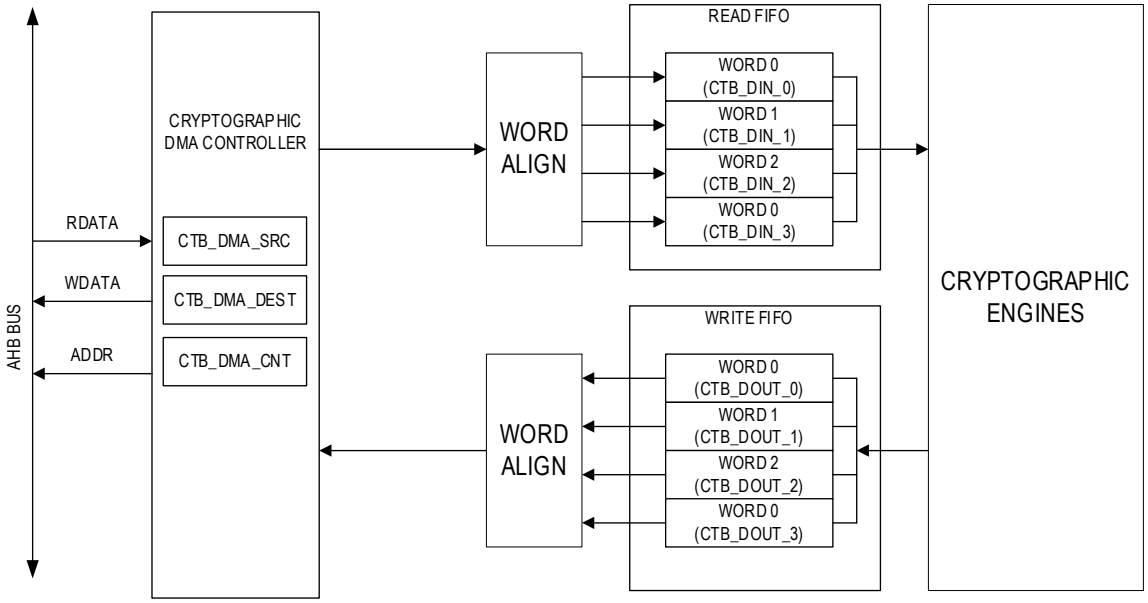
The CTB is disabled by default on a reset. Use of the CTB peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS0.crypto* to 0 to enable the peripheral clock to the CTB before configuring the CTB for use.

27.2 Cryptographic DMA (CDMA)

A dedicated DMA engine performs high-speed accesses between the CTB and memory on the AHB bus. The DMA engine significantly 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, allowing the following cipher or hash operation to immediately start the calculation on the next block. This keeps the cipher and hash generator running continuously without waiting for data to be written to or read from the bus.

Figure 27-1: CDMA Block Diagram



27.3 CTB FIFOs

The read FIFO and write FIFO have programmable sources shown in *Table 27-1* to allow flexibility in their operation.

Table 27-1: 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 uses 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 like 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. The AHB DMA is the read FIFO source in this mode, 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.

The software can use DMA to copy memory, like the memcpy() standard C function, by setting the write FIFO source to the read FIFO. If the Hamming ECC generator is enabled, software can copy flash memory pages to memory while simultaneously calculating the error correction code.

The software can fill memory with a block of data like the memset() standard C function by pointing the write FIFO source to the read FIFO and setting the read FIFO to the APB. Similarly, the software 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, software should set the write FIFO source to the read FIFO. When calculating a hash or CMAC, disable the write FIFO.

27.4 Block Cipher Engine

The block cipher engine is a dedicated hardware module that accelerates the computation of the following algorithms:

- AES-128

AES-192

- AES-256

Counter with CBC-MAC (CCM)

- Galois Counter Mode (GCM)

The symmetric block ciphers encrypt or decrypt data in blocks. The block sizes for each cipher are shown in *Table 27-2*.

Table 27-2: Symmetric Block Ciphers

Cipher	Key Size	Used Key Bits	Effective Strength (NIST SP800-57)	Block Size
AES-128	128-bits	*CTB_CIPHER_KEY[3]*:*CTB_CIPHER_KEY[0]* or *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0*	128-bits	128-bits
AES-192	192-bits	*CTB_CIPHER_KEY[5]*:*CTB_CIPHER_KEY[0]* or *AES_KEY_AES_KEY5*:*AES_KEY_AES_KEY0*	192-bits	128-bits
AES-256	256-bits	*CTB_CIPHER_KEY[7]*:*CTB_CIPHER_KEY[0]* or *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0*	256-bits	128-bits
CCM				
GCM				

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, ECB, each data block is simply encrypted or decrypted using the cipher. A side effect of this is that identical data blocks 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, plain-text 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. 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.

27.4.1 Cipher Key Storage and Initialization

Block cipher operations require a user-supplied cipher key to be loaded into *CTB_CIPHER_KEY[7]*:*CTB_CIPHER_KEY[0]* before any algorithm can be executed.

The length of the key is dependent on the specific algorithm used.

The following procedure is required load a key of 128 bits or less:

1. Clear *CTB_CTRL*.*done* = 0 and *CTB_CTRL*.*dma_done* = 0.
2. Set *CTB_CIPHER_CTRL*.*src* to select the source of the AES keys.
3. Poll until hardware sets *CTB_CTRL*.*dma_done* = 1.
4. Clear *CTB_CTRL*.*done* = 0 and *CTB_CTRL*.*dma_done* = 0.

27.4.2 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. Block cipher operations set *CTB_CTRL.cph_done* = 1 when complete.

1. Reset the engine by setting *CTB_CTRL.rst* = 1.
2. Poll until *CTB_CTRL.rdy* reads 1.
3. Select the cipher algorithm operation using *CTB_CIPHER_CTRL.cipher*.
4. Select the mode of operation using *CTB_CIPHER_CTRL.mode*.
5. Select encryption or decryption mode *CTB_CIPHER_CTRL.enc*.
6. Load *CTB_CIPHER_INIT[0]*, *CTB_CIPHER_INIT[1]*, *CTB_CIPHER_INIT[2]*, and *CTB_CIPHER_INIT[3]* with the initial vector if using CBC, CFB, OFB or counter modes.
7. Set *CTB_CTRL.rdsrc* to 1 to select the read FIFO source as DMA
8. Set *CTB_CTRL.wrsrc* to 1 to select the cipher output FIFO source as DMA
9. Load the DMA source address to *CTB_DMA_SRC*.
10. Load the DMA destination address to *CTB_DMA_DEST.addr*.
11. Load the DMA count to *CTB_DMA_CNT.count*.

At the end of the DMA count:

- *CTB_CTRL.done* = 1

CTB_CTRL.dma_done = 1

- *CTB_CTRL.cph_done* = 1

An interrupt is generated if *CTB_CTRL.intr* = 1

27.4.3 AES GCM and CCM

When using AES GCM, each time the AES key is changed, the software is responsible for computing a new H vector corresponding to the key by setting the *CTB_CIPHER_CTRL.hvc* bit. This bit is automatically cleared by hardware when the computation starts.

Before starting an AES GCM or CCM computation, the AAD length (*CTB_AAD_LENGTH[1]*:*CTB_AAD_LENGTH[0]*) and PLD length (*CTB_PLD_LENGTH[1]*:*CTB_PLD_LENGTH[0]*) registers must be set. The registers should be set to the number of bytes of the AAD or PLD. The initialization vector must be set in the *CTB_CIPHER_INIT[3]*:*CTB_CIPHER_INIT[0]* registers, and *CTB_CIPHER_CTRL.mode* must be set to GCM or CCM. For CCM mode, CCM parameters M and L must also be set in *CTB_CIPHER_CTRL.ccmm* an *CTB_CIPHER_CTRL.ccml* fields. The computation starts when data is loaded.

If using 128-bit AAD blocks, they must be loaded first (*CTB_CIPHER_CTRL.dtype* must be set to 0). Data must be left-aligned and right padded with zeros if required. When AES is ready to receive a new AAD block, *CTB_CTRL.rdy* is set.

Payload blocks must be loaded next (*CTB_CIPHER_CTRL.dtype* must be set to 1). Data must be left-aligned and right padded with zeros if required. When loading payload blocks, read *CTB_CTRL.rdy* until it reads 1, indicating that the corresponding cipher block is ready to be fetched and that a new block can be loaded (if needed).

When the computation is complete, *CTB_CTRL.cph_done* is set and the authentication tag is available in the *CTB_TAGMIC*:*CTB_TAGMIC*[0] registers.

27.5 Hash Engine

The hash engine takes an input of arbitrary length and outputs a fixed-length digest to the *CTB_HASH_DIGEST*[15]:*CTB_HASH_DIGEST*[0] registers. The supported hash algorithms are shown in *Table 27-3*. Software can use the engine to automatically seed the hash accelerator with the appropriate secure hash constants and pad the final block as required by the FIPS 180-2 standard.

Setting the *CTB_HASH_CTRL.init* bit seeds the hash engine with the secure hash constants required by security validation requirements. Contact Analog Devices for specific information about the seeding process and its comparability with the latest security requirements for NIST FIPS and other certifications.

The CDMA loads data for the hash engine, allowing large messages in RAM to be hashed with minimal CPU intervention. Calculation of the hash can occur in parallel with the block cipher as long as only one operation uses the DMA.

Table 27-3: Hash Function Digest Length and Block Sizes

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

Data is processed in 512-bit or 1024-bit blocks. The following values must be calculated before using the hash engine:

- The total length of the message in bits

  The number of bits contained in all the complete blocks in the message
  - This is the integer value of the message length divided by the block size for the algorithm.
- The number of bits associated with the final incomplete block, if any

The hash engine begins operation as soon as the *CTB_DMA_CNT.count* register is set to a non-zero value. The hashing process has two steps:

1. Software configures the CDMA to load the hash engine with all the bits of the complete blocks. The hash engine sets the *CTB_CTRL.dma_done* flag and generates an interrupt when all of the complete blocks are hashed.
2. If necessary, software sets the *CTB_HASH_CTRL.last* bit to process the final incomplete block. Software configures the CDMA to load the final bits of the incomplete block, which are automatically padded by the hash engine as required to produce the final digest. The hash engine sets the *CTB_CTRL.dma_done* flag, the *CTB_CTRL.hsh_done* flag, and generates an interrupt when the message is hashed.

Software can read the final message digest from the *CTB_HASH_DIGEST[15]*:*CTB_HASH_DIGEST[0]* registers.

27.5.1 Hash Operation

1. Reset the hash engine by setting *CTB_CTRL.rst* = 1.
2. Poll until hardware sets *CTB_CTRL.rdy* = 1.
3. Configure *CTB_HASH_CTRL.hash* to select the desired hash function.
4. Configure *CTB_HASH_MSG_SZ[0]* to the total number of bits to hash.
5. Set *CTB_CTRL.rdsrc* = 0b01 to select the read FIFO source as DMA.
6. Set *CTB_CTRL.wrsrc* to 0b01 to select the cipher output FIFO source as DMA.
7. Set *CTB_CTRL.intr* = 1 to enable an interrupt when the DMA transfer is complete.
8. Load the starting address of the input message to *CTB_DMA_SRC*.
9. Load *CTB_DMA_CNT.count* with the number of bits associated with whole blocks as shown in *Table 27-3*, starting the hashing operation.

At the completion of the DMA count the following occur:

- *CTB_CTRL.done* = 1

CTB_CTRL.dma_done = 1

- An interrupt is generated if *CTB_CTRL.intr* = 1

Software interrupt handler code should perform the following:

1. Clear *CTB_CTRL.intr* to 0.
2. Clear *CTB_CTRL.done* to 0.
3. Clear *CTB_CTRL.dma_done* to 0.
4. If *CTB_CTRL.hsh_done* is 0, process the final block:
   a. Set *CTB_HASH_CTRL.last* to 1.
      i. Hardware clear *CTB_HASH_CTRL.last* to 0 when the last block is hashed.
   b. Load the address of the last block in *CTB_DMA_SRC*.
   c. Set *CTB_CTRL.intr* to 1 to enable the interrupt when the DMA transfer is complete.
5. Return from the interrupt handler.

27.6 Hamming Code Engine

The Hamming code engine can be used to calculate an ECC on a block of data. Hamming codes are capable of detecting and correcting single-bit errors. An extra parity bit (*CTB_HAM_ECC.par*) is calculated to enable the detection of two-bit errors. This is commonly referred to as SECDED. Three errors masquerade as a correctable single-bit error.

The Hamming code generator calculates the ECC on a block of data up to 216 bits in length (8KB). Software implementations can extend this to any length. Since the Hamming code can only correct a single bit error, increasing the block size increases the likelihood of multiple uncorrectable errors. The engine generates even parity on even halves of bit groups.

27.6.1 Hamming Code Operation

1. Set CTB_CTRL.*rst* = 1 to reset the CTB.
2. Poll until hardware sets CTB_CTRL.*rdy* = 1.
3. Configure the CTB for hamming code calculation:
   a. Clear CTB_CIPHER_CTRL.*cipher* = 0.
   b. Clear CTB_HASH_CTRL.*hash* = 0.
   c. Set CTB_CRC_CTRL.*ham* = 1.
4. Set CTB_CRC_CTRL.*hrst* = 1 to reset the hamming code generator for the next calculation.
5. Set CTB_CTRL.*rdsrc* = 1 to select the DMA as the read FIFO source. The CTB_CTRL.*wrsrc* field is ignored during hamming code calculations.
6. Set CTB_CTRL.*intr* = 1 to enable the interrupt when the DMA transfer is complete.
7. Load the starting address of the block to CTB_DMA_SRC.*addr*.
8. Write the length of the block in bytes to CTB_DMA_CNT.*count* to start the calculation.

At the end of the DMA count:

- An interrupt is generated if CTB_CTRL.*intr* = 1
- CTB_CTRL.*done* = 1
- CTB_CTRL.*dma_done* = 1
- CTB_CTRL.*gls_done* = 1
- CTB_HAM_ECC.*ecc* contains the calculated hamming code.
- CTB_HAM_ECC.*par* contains the parity of the entire array.

27.7 CRC

The CRC engine performs CRC functions on data stored in RAM. The CDMA copies the data into the CRC engine so that CRC calculations on large blocks of memory are performed with minimal CPU intervention. The CRC engine cannot be used to perform a CRC of data stored in flash memory.

The CRC generator has a programmable polynomial up to 32-bits, written to the CTB_CRC_POLY register. 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 CTB_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 ($x^0 to\ x^{32}$). Table 27-4 lists frequently used polynomials and their check constants.

Table 27-4: Common CRC Polynomials

Algorithm	Polynomial Expression	Order	Polynomial	Check Constant
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	$x^{16}+ x^{15}+ x^2+ x^0$	LSB	0x8005 0000	0x800D 0000
Parity	$x^1+ x^0$	MSB	0x0000 0001	-

Because the receiving end calculates a new CRC on both the data and received CRC, software 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, the CRC is reversed bitwise, with the highest-order term of the remainder in the LSB position. Software CRC algorithms often handle this by calculating everything in reverse, with the resulting CRC being bit swapped and in the correct format.

By default, the CRC accelerator does right shifts and calculates the CRC on the LSB of the data first. To calculate the CRC MSB first, set the bit-swap control bit to 1 (CTB_CRC_CTRL.*msb* = 1) and left justify the polynomial in the CTB_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

must be set to define the length of the CRC. The initial state of the CRC must be left-justified. When the CRC calculation is complete, it is necessary to shift right the CRC to right justify it if the polynomial is less than 32 bits.

27.7.1 CRC Operation

1. Reset the CRC engine by setting *CTB_CTRL.rst* to 1.
2. Poll until hardware sets *CTB_CTRL.rdy* to 1.
3. Clear the done fields to 0:
   a. *CTB_CTRL.cph_done*
   b. *CTB_CTRL.hsh_done*
   c. *CTB_CTRL.gls_done*
   d. *CTB_CTRL.dma_done*
4. Clear *CTB_CTRL.dmadnemsk* and *CTB_CTRL.flag_mode* to 0.
5. Clear *CTB_CTRL.rdsrc* to 0 to select the input FIFO as the input source.
6. Set *CTB_CTRL.intr* to 1 to enable the interrupt when the DMA transfer completes.
7. Write the desired polynomial to the *CTB_CRC_POLY* register.
8. Set *CTB_CRC_CTRL.crc* to 1 to enable the CRC function.
9. Load *CTB_DMA_SRC.addr* with the starting address of the memory to CRC.
10. Load *CTB_.count* with the number of bytes to be processed. An interrupt is generated when the DMA operation is complete and the calculated CRC is ready.

At the end of the DMA count, hardware does the following:

- *CTB_CTRL.done* = 1
- *CTB_CTRL.dma_done* = 1
- *CTB_CTRL.gls_done* = 1
- *CTB_CRC_VAL.val* contains the calculated CRC value.
- An interrupt is generated if enabled (*CTB_CTRL.intr* = 1)

27.8 MAA

The MAA supports operation that are key parts of many cryptographic algorithms. These include IEEE Public Key Cryptography Standard P1363 for asymmetric cryptographic operations based on DSA, RSA, and ECDSA algorithms. The MAA does not use the CDMA engine.

The MAA features include:

- Supports up to a 2048-bit operand size
- Performs up to 2048-bit modular exponentiation ($a^e mod\ m$)
- Performs up to 2048-bit modular multiplication ($a \times b\ mod\ m$)
- Performs up to 2048-bit modular square ($b^2 mod\ m$)
- Performs up to 2048-bit modular square followed by modular multiply ($(b^2\ mod\ m) \times a\ mod\ m$)
- 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.

Operands and parameters are stored in a dedicated 768 x 16 data memory segment.

27.8.1 Operation

The MAA control register (*CTB_MAA_CTRL*) provides option control on arithmetic operations, data partition, and status bits for start/busy. The starting address of most parameters is configurable within the memory instance using the memory assignment fields, as shown in *Table 27-5*. The MAA uses a big-endian data convention - the most significant byte or sub-word of a multibyte word is loaded first and stored at the lowest storage location.

Table 27-5: Cryptographic Memory Segments

Segment	Memory Segment Assignment	Description
a	*CTB_MAA_CTRL.ama*	Multiplier/Operand A
b	*CTB_MAA_CTRL.bma*	Multiplicand/Operand B
e	Fixed	Exponent
m	Fixed	Modulus (only required for exponentiation operations)
t	*CTB_MAA_CTRL.tma*	Temporary storage. The data in this segment is undefined.
r	*CTB_MAA_CTRL.rma*	Result value

The modular accelerator word size field (*CTB_MAA_MAWS.size*) 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 modular operation, set *CTB_MAA_MAWS.size* to 0x03EF. Valid sizes are from 1 to 2048. The MAA does not start if *CTB_MAA_MAWS.size* is invalid.

The *CTB_CTRL.maa_done* field is set to 1 after the completion of an MAA operation or when an error occurs. Before starting an MAA operation, clear the *CTB_CTRL.maa_done* field.

27.8.1.1 MAA Memory

A dedicated, 1,535-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 segments when assigning parameter locations. The following restrictions apply when storing parameters in the MAA memory.

- Only one parameter can be located in each memory segment.
- 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 an operation that does not require an exponent is selected, memory instance 4 can hold any parameter.
- Parameters *m*, *b*, *t,* and *r* must be stored in different memory instances even when *CTB_MAA_MAWS.size* < 1024 bits. For example, assuming a size less than 1024, if *b* is stored in memory segment 0, then neither *t* nor *r* can be stored in memory segment 1. Each memory instance is 256 bytes.

Table 27-6: MAA Memory Segments and Locations (CTB_MAA_MAWS.size < 1024)

Memory Instance	Segment Number	Dedicated Function	Address Offset
0	0	-	[0x0100] – [0x017F]
	1	-	[0x0180] – [0x01FF]
1	2	-	[0x0200]– [0x027F]
	3	-	[0x0280]– [0x02FF]
2	4	-	[0x0300] – [0x037F]
	5	-	[0x0380] – [0x03FF]
3	6	-	[0x0400] – [0x047F]
	7	-	[0x0480] – [0x04FF]
4	8	Exponent (if required for operation)	[0x0500] – [0x057F]
	9	-	[0x0580] – [0x05FF]

Memory Instance	Segment Number	Dedicated Function	Address Offset
5	10	Modulus	[0x0600] – [0x06FF]

Table 27-7: MAA Memory Segments and Locations (CTB_MAA_MAWS.size ≥ 1024)

Memory Instance	Segment Number	Dedicated Function	Address Offset
0	0	-	[0x0100] –[0x01FF]
1	1	-	[0x0200]– [0x02FF]
2	2	-	[0x0300] – [0x03FF]
3	3	-	[0x0400] – [0x04FF]
4	4	Exponent (if required for operation)	[0x0500] – [0x05FF]
5	5	Modulus	[0x0600] – [0x06FF]

27.8.1.2 Memory Blinding

Memory blinding is an effective cryptographic technique that greatly 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 provide the option of shifting the starting position of the *a*, *b*, *e*, and *m* parameters within their respective memory instances. Although software can alter the starting position, the entire parameter must be stored within a single memory instance. Each parameter can be independently configured with a blinded starting position. Memory blinding for each parameter is controlled by its corresponding memory select bits (*CTB_MAA_CTRL.ams*, *CTB_MAA_CTRL.bms*, *CTB_MAA_CTRL.ems*, and *CTB_MAA_CTRL.mms*).

Table 27-8: MAA Memory Blinding Example (Memory Instance 0, CTB_MAA_MAWS.size ≥ 1024)

***CTB_MAA_CTRL.<x>ms* (<x> = *m* , *e*, *b*, *a*)**	**Shift**	**Memory Instance 0 Offset**
0	0x00	0x0100 — 0x01FF (no blinding)
1	0x40	0x0140 — 0x01FF
2	0x80	0x0180 — 0x01FF
3	0xC0	0x01C0 — 0x01FF

27.8.1.3 Optimized Calculations

The optimized calculation control bit *(CTB_MAA_CTRL.opt*) allows the software to optimize the speed of an exponentiation by skipping unnecessary multiply operations when the corresponding exponent bit is a 0. The field 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 the first 1 is detected. From this point on, multiply operations 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.

27.8.1.4 MAA Operand Sizes

MAA operand size is specified using the *CTB_MAA_MAWS.size* field. Valid values are from 1 to 2048. This value specifies the size for parameters *a*, *b*, *e*, and *m*.

Operands *a*, *b*, and *e* can have any values between 0 and $2^{CTB\_MAA\_MAWS.size-1}$ inclusive as long as the number of bits are less than those of *m*. For exponentiation operations, *b* memory must contain the value of 1 and *e* cannot be 0.

The *m* memory, which holds the modulus, can have a value from $2^{CTB\_MAA\_MAWS.size-1}$ up to $2^{CTB\_MAA\_MAWS.size} - 1$. For example, for a 16-bit size (*CTB_MAA_MAWS.size* = 0x10), the *m* memory can have a value from 32,768 bits up to 65,535 bits.

If any parameter exceeds the *CTB_MAA_MAWS.size* field, an invalid result is generated without issuing an MAA error status. It is up to the software 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,

CTB_MAA_MAWS.size = 65 and operand = 0x01 xxxx xxxx xxxx xxxx. 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 0x0000 0000 0000 0001 is stored can have any value.

For most calculations, a temporary memory *t* and a result *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 used as an additional temporary storage area during round operations.

27.9 TRNG Engine

The Analog Devices-supplied Universal Cryptographic Library (UCL) (*https://www.maximintegrated.com/en/design/technical-documents/userguides-and-manuals/7/7220.html*) provides a function to generate random numbers intended to meet the requirements of common security validation suites. The entropy from one or more internal noise sources continually feeds a TRNG, the output of which is then processed by software and hardware to generate the number returned by the UCL function. Analog Devices works directly with the customer's accredited testing laboratory to provide any information regarding the TRNG needed to support the customer's validation requirements.

The general information in this section is provided only for completeness; customers are expected to use the Analog Devices UCL to generate random numbers.

The TRNG engine can also be used to generate AES keys.

27.10 Peripheral Clock Enable

The TRNG is disabled by default on a reset. Use of the TRNG peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS1.trng* to 0 to enable the peripheral clock to the TRNG before configuring the TRNG for use.

27.11 CTB Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 27-9: Cryptographic Toolbox Register Summary

Offset	Register	Name
[0x0000]	*CTB_CTRL*	Cryptographic Control Register
[0x0004]	*CTB_CIPHER_CTRL*	Cipher Control Register
[0x0008]	*CTB_HASH_CTRL*	Hash Control Register
[0x000C]	*CTB_CRC_CTRL*	CRC Control Register
[0x0010]	*CTB_DMA_SRC*	Cryptographic DMA Source Register
[0x0014]	*CTB_DMA_DEST*	Cryptographic DMA Destination Register
[0x0018]	*CTB_DMA_CNT*	Cryptographic DMA Count Register
[0x001C]	*CTB_MAA_CTRL*	MAA Control Register
[0x0020]	*CTB_DIN[0]*	Cryptographic Data Input Register 0 [31:0]
[0x0024]	*CTB_DIN[1]*	Cryptographic Data Input Register 1 [63:32]
[0x0028]	*CTB_DIN[2]*	Cryptographic Data Input Register 2 [95:64]
[0x002C]	*CTB_DIN[3]*	Cryptographic Data Input Register 3 [127:96]
[0x0030]	*CTB_DOUT[0]*	Cryptographic Data Output Register 0 [31:0]
[0x0034]	*CTB_DOUT[1]*	Cryptographic Data Output Register 1 [63:32]
[0x0038]	*CTB_DOUT[2]*	Cryptographic Data Output Register 2 [95:64]
[0x003C]	*CTB_DOUT[3]*	Cryptographic Data Output Register 3 [127:96]
[0x0040]	*CTB_CRC_POLY*	CRC Polynomial Register
[0x0044]	*CTB_CRC_VAL*	CRC Value Register
[0x004C]	*CTB_HAM_ECC*	Hamming ECC Register
[0x0050]	*CTB_CIPHER_INIT[0]*	Cipher Initial Vector Register 0 [31:0]

Offset	Register	Name
[0x0054]	CTB_CIPHER_INIT[1]	Cipher Initial Vector Register 1 [63:32]
[0x0058]	CTB_CIPHER_INIT[2]	Cipher Initial Vector Register 2 [95:64]
[0x005C]	CTB_CIPHER_INIT[3]	Cipher Initial Vector Register 3 [127:96]
[0x0060]	CTB_CIPHER_KEY[0]	Cipher Key Register 0 [31:0]
[0x0064]	CTB_CIPHER_KEY[1]	Cipher Key Register 1 [63:32]
[0x0068]	CTB_CIPHER_KEY[2]	Cipher Key Register 2 [95:64]
[0x006C]	CTB_CIPHER_KEY[3]	Cipher Key Register 3 [127:96]
[0x0070]	CTB_CIPHER_KEY[4]	Cipher Key Register 4 [159:128]
[0x0074]	CTB_CIPHER_KEY[5]	Cipher Key Register 5 [191:160]
[0x0078]	CTB_CIPHER_KEY[6]	Cipher Key Register 6 [223:192]
[0x007C]	CTB_CIPHER_KEY[7]	Cipher Key Register 7 [255:224]
[0x0080]	CTB_HASH_DIGEST[0]	Hash Message Digest Register 0 [31:0]
[0x0084]	CTB_HASH_DIGEST[1]	Hash Message Digest Register 1 [63:32]
[0x0088]	CTB_HASH_DIGEST[2]	Hash Message Digest Register 2 [95:64]
[0x008C]	CTB_HASH_DIGEST[3]	Hash Message Digest Register 3 [127:96]
[0x0090]	CTB_HASH_DIGEST[4]	Hash Message Digest Register 4 [159:128]
[0x0094]	CTB_HASH_DIGEST[5]	Hash Message Digest Register 5 [191:160]
[0x0098]	CTB_HASH_DIGEST[6]	Hash Message Digest Register 6 [223:192]
[0x009C]	CTB_HASH_DIGEST[7]	Hash Message Digest Register 7 [255:224]
[0x00A0]	CTB_HASH_DIGEST[8]	Hash Message Digest Register 8 [287:256]
[0x00A4]	CTB_HASH_DIGEST[9]	Hash Message Digest Register 9 [319:288]
[0x00A8]	CTB_HASH_DIGEST[10]	Hash Message Digest Register 10 [351:320]
[0x00AC]	CTB_HASH_DIGEST[11]	Hash Message Digest Register 11 [383:352]
[0x00B0]	CTB_HASH_DIGEST[12]	Hash Message Digest Register 12 [415:384]
[0x00B4]	CTB_HASH_DIGEST[13]	Hash Message Digest Register 13 [447:416]
[0x00B8]	CTB_HASH_DIGEST[14]	Hash Message Digest Register 14 [479:448]
[0x00BC]	CTB_HASH_DIGEST[15]	Hash Message Digest Register 15 [511:480]
[0x00CC]	CTB_HASH_MSG_SZ[0]	Hash Message Size Register 0 [31:0]
[0x00C4]	CTB_HASH_MSG_SZ[1]	Hash Message Size Register 1 [63:32]
[0x00C8]	CTB_HASH_MSG_SZ[2]	Hash Message Size Register 2 [95:64]
[0x00CC]	CTB_HASH_MSG_SZ[3]	Hash Message Size Register 3 [127:96]
[0x00D0]	CTB_MAA_MAWS	MAA Word Size Register
[0x00D0]	CTB_AAD_LENGTH[0]	AAD Length Register 0 Register
[0x00D4]	CTB_AAD_LENGTH[1]	AAD Length Register 1 Register
[0x00D8]	CTB_PLD_LENGTH[0]	PLD Length Register 0 Register
[0x00DC]	CTB_PLD_LENGTH[1]	PLD Length Register 1 Register
[0x00E0]	CTB_TAGMIC[0]	Tag/Mic Register 0
[0x00E4]	CTB_TAGMIC[1]	Tag/Mic Register 1
[0x00E8]	CTB_TAGMIC[2]	Tag/Mic Register 2
[0x00EC]	CTB_TAGMIC	Tag/Mic Register 3

27.11.1 Register Details

Table 27-10: Cryptographic Control Register

Cryptographic Control			CTB_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 (*CTB_CTRL.cph_done*, *CTB_CTRL.hsh_done*, *CTB_CTRL.gls_done*, or *CTB_CTRL.dma_done*). This bit remains set until cleared by software. Writing 0 to one or more of the done bits (*CTB_CTRL.cph_done*, *CTB_CTRL.hsh_done*, *CTB_CTRL.gls_done*, or *CTB_CTRL.dma_done*) does not affect this bit. Setting the *CTB_CTRL.dmadnemsk* bit to 1 causes this bit to be set to 1 when hardware sets the *CTB_CTRL.dma_done* bit. 0: Normal operation. 1: One or more cryptographic operations are complete.
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 *CTB_CTRL.rst* bit. Software must poll this bit until it is set to 1 by hardware, indicating the cryptographic accelerator is again ready for use. 0: Cryptographic accelerator reset in progress. 1: Normal operation.
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: Normal operation. 1: Cryptographic accelerator DMA bus error.
28	maa_done	R/W	0	**MAA Operation Complete** Software should clear this field before starting an MAA operation. This field is read-only when an MAA operation is in progress. 0: Normal operation. 1: MAA operation complete.
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 the cryptographic accelerator has completed a hash operation. Clear this bit before starting the next hash operation. 0: Not done. 1: Hash operation done.
25	gls_done	R/W	0	**Galois Done** 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**

Cryptographic Control			CTB_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
15	dmadnemsk	R/W	0	**DMA Done Flag Mask** This field prevents the *CTB_CTRL.dma_done* flag from setting the *CTB_CTRL.done* flag. The *CTB_CTRL.dma_done* flag is still set. 0: *CTB_CTRL.dma_done* flag does not set *CTB_CTRL.done.* 1: *CTB_CTRL.dma_done* flag also sets *CTB_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 *CTB_CTRL.cph_done*, *CTB_CTRL.hsh_done*, *CTB_CTRL.gls_done*, and *CTB_CTRL.dma_done.* 1: Access to *CTB_CTRL.cph_done*, *CTB_CTRL.hsh_done*, *CTB_CTRL.gls_done*, and *CTB_CTRL.dma_done* bits is W1C.
13:12	-	RO	0	**Reserved**
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	DNM	0	**Wait Pin Polarity**
6	wait_en	DNM	0	**Wait Pin Enable**
5	bsi	R/W	0	**Byte Swap Input** Set this field to byte swap the 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** Set this field to byte swap the output. *Note: No byte swap occurs if there is not a full word.* 0: No effect. 1: Byte swap output.
3	-	RO	0	**Reserved**
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	intr	R/W	0	**Interrupt Enable** Generates an interrupt when done or an error occurs. 0: Interrupt disabled. 1: Interrupt asserted when *CTB_CTRL.done* is set.

Cryptographic Control			CTB_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
0	rst	R/W	0	**Reset Cryptographic Accelerator** Setting this bit starts an internal reset of the cryptographic accelerator. Software must poll the *CTB_CTRL.rdy* bit to determine when the reset process is complete. All internal cryptographic states and related registers are reset to their default reset values. Control registers retain their values. This bit automatically clears itself after one cycle. 0: No effect. 1: Reset cryptographic accelerator.

Table 27-11: Cipher Control Register

Cipher Control			CTB_CIPHER_CTRL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:19	-	RO	0	**Reserved**
18:16	ccml	RO	0	**CCM L Parameter** 0b000: Reserved. 0b001: 2 0b010: 3 0b011: 4 0b100: 5 0b101: 6 0b110: 7 0b111: 8
15:13	ccmm	RO	0	**CCM M Parameter** 0b000: Reserved. 0b001: 4 0b010: 6 0b011: 8 0b100: 10 0b101: 12 0b110: 14 0b111: 16
12	dtype	RO	0	**GCM/CCM data type** 0: AAD. 1: Payload.
11	hvc	RO	0	**H Vector Computation** This bit is automatically cleared by hardware when the computation is complete. 0: NOP. 1: Start an H vector computation.
10:8	mode	R/W	0	**Mode Select** Set the operating mode of the block cipher or memory operation. 0b000: ECB. 0b001: CBC. 0b010: CFB. 0b011: OFB. 0b100: CTR. 0b101: GCM. 0b110: CCM. 0b111: Reserved.
7	-	RO	0	**Reserved**

Cipher Control			CTB_CIPHER_CTRL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
6:4	cipher	R/W	0	**Block Cipher Operation Select** Select the AES block cipher operation mode. Clear this field before starting any non-AES operation of the cryptographic accelerator. 0b000: Disabled. 0b001: AES-128. 0b010: AES-192. 0b011: AES-256. 0b100: Reserved. 0b101: Reserved. 0b110: Reserved. 0b111: Reserved.
3:2	src	R/W	0	**Source of Cipher Key** This field indicates the key data source to be loaded into the registers when the *CTB_CIPHER_CTRL.key* bit is set to 1. 0b00: *CTB_CIPHER_KEY[7]*:*CTB_CIPHER_KEY[0]* registers. 0b01: Reserved. 0b10: Reserved. 0b11: *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers.
1	key	R/W1O	0	**Load Cipher Key Using DMA** Setting this bit initiates loading of the *CTB_CIPHER_KEY[7]*:*CTB_CIPHER_KEY[0]* registers. The bit must be cleared, and the *CTB_CTRL.dma_done* bit after the DMA completes loading the key. 0: NOP. 1: Initiate key loading from DMA.
0	enc	R/W	0	**Block Cipher Encryption Mode** 0: Decrypt. 1: Encrypt.

Table 27-12: Hash Control Register

Hash Control			CTB_HASH_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**
5	last	R/W	0	**Last Message Bit** This bit is set along with the *CTB_HASH_MSG_SZ[3]*:*CTB_HASH_MSG_SZ[0]* registers before hashing the last 512- or 1024-bit block of message data. It allows automatic preprocessing of the last message padding, including the trailing 1 bit followed by the respective number of zero bits for the last block size and the message length represented in bytes. The bit is automatically cleared simultaneously when the *CTB_CTRL.hsh_done* is set, designating the completion of the last message. 0: No effect. 1: Last message data.
4:2	hash	R/W	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.

Hash Control			CTB_HASH_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
1	xor	R/W	0	**XOR IV and Cipher Block** Useful when calculating HMAC to XOR the input pad and output pad. Use the feature to load the key from CDMA. This bit is automatically cleared by hardware after the DMA completes the key loading. When the DMA operation is done, it sets the appropriate CDMA done flag. 0: No XOR. 1: XOR input with IV.
0	init	R/W	0	**Initialize Hash Digest** Setting this bit to 1 initializes *CTB_HASH_DIGEST[0]* with the appropriate hash value constants stored in preprogrammed in non-volatile memory. This bit should be set before starting a new hash calculation. 0: NOP. 1: Initialize *CTB_HASH_DIGEST[0]*.

Table 27-13: CRC Control Register

CRC Control			CTB_CRC_CTRL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**
5	hrst	R/W	0	**Hamming Reset** Reset the Hamming code ECC generator for the next block. 0: NOP. 1: Reset Hamming generator.
4	ham	R/W	0	**Hamming Code Enable** Enable Hamming code calculation. 0: Disabled. 1: Enabled.
3	ent	DNM	0	**Entropy Enable** This feature is not implemented in this device. Do not modify.
2	prng	DNM	0	**PRNG Enable** This feature is not implemented in this device. Do not modify.
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** 0: CRC disabled. 1: CRC enabled.

Table 27-14: Cryptographic DMA Source Register

Cryptographic DMA Source			CTB_DMA_SRC	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**DMA Source Address** DMA source address for cryptographic operations.

Table 27-15: Cryptographic DMA Destination Register

Cryptographic DMA Destination			CTB_DMA_DEST	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**DMA Destination Address** DMA destination address for cryptographic operations.

Table 27-16: Cryptographic DMA Count Register

Cryptographic DMA Count			CTB_DMA_CNT	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0.	count	R/W	0	**DMA Byte Counter** Writing a non-zero value to this register initiates the DMA-based operations.

Table 27-17: MAA Control Register

MAA Control			CTB_MAA_CTRL	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:28	tma	R/W	0	**Temporary (*t*) Memory Assignment** This field selects the logical cryptographic RAM segment for parameter *t*. See *Table 27-6* and *Table 27-7* for details of this field and the relationship to the *CTB_MAA_MAWS.size* field.
27:24	rma	R/W	0	**Result (*r*) Memory Assignment** This field selects the logical cryptographic RAM segment for the result *r*. See *Table 27-6* and *Table 27-7* for details of this field and the relationship to the *CTB_MAA_MAWS.size* field.
23:20	bma	R/W	0	**Multiplicand/Operand *b* Memory Assignment** This field selects the logical cryptographic RAM segment for the multiplicand or operand *b*. See *Table 27-6* and *Table 27-7* for details of this field and the relationship to the *CTB_MAA_MAWS.size* field.
19:16	ama	R/W	0	**Multiplier/Operand *a* Memory Assignment** This field selects the logical cryptographic RAM segment for the multiplier or operand *a*. See *Table 27-6* and *Table 27-7* for details of this field and the relationship to the *CTB_MAA_MAWS.size* field.
15:14	mms	R/W	0	**Modulus (*m*) Memory Offset Select** This field selects the starting position of parameter *m* within memory instance 5. See *Memory Blinding* for details of this field's usage. 0: No offset. 1: Offset 0x40. 2: Offset 0x80. 3: Offset 0xC0.
13:12	ems	R/W	0	**Exponent (*e*) Memory Offset Select** This field selects the starting position of parameter *e* within memory instance 4. See *Memory Blinding* for details of this field's usage. 0: No offset. 1: Offset 0x40. 2: Offset 0x80. 3: Offset 0xC0.
11:10	bms	R/W	0	**Multiplicand/Operand *b* Memory Offset Select** This field selects the starting position of operand *b* within the memory segment specified by *CTB_MAA_CTRL.bma*. See *Memory Blinding* for details of this field's usage. 0: No offset. 1: Offset 0x40. 2: Offset 0x80. 3: Offset 0xC0.
9:8	ams	R/W	0	**Multiplier/Operand *a* Memory Offset Select** This field selects the starting position of operand *a* within the logical segment specified by *CTB_MAA_CTRL.ama*. See *Memory Blinding* for details of this field's usage. 0: No offset. 1: Offset 0x40. 2: Offset 0x80. 3: Offset 0xC0.

MAA Control			CTB_MAA_CTRL	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
7	maa_err	R/W	0	**MAA Error Flag** This field is set by hardware if software attempts to write to the *CTB_MAA_CTRL* or *CTB_MAA_MAWS* registers when an MAA operation is in progress. Software must clear this field to 0. 0: Normal operation. 1: Error.
6:5	-	RO	0	**Reserved**
4	opt	R/W	0	**Optimized Calculation Control** Setting this field to 1 skips unnecessary multiply operations after normalizing the exponent. See *Optimized Calculations* for additional details. 0: Normal operation. 1: Optimize calculation.
3:1	calc	R/W	0	**Calculation Configuration** This field selects the MAA operation. 0: Modular exponentiation. 1: Modular square. 2: Modular multiplication. 3: Modular square followed by modular multiply. 4: Modular addition. 5: Modular subtraction. 6: Reserved. 7: Reserved.
0	start	R/W	0	**Start MAA Operation** Set this field to 1 to start an MAA operation. The field remains 1 until the operation completes or encounters an error. Writing 0 to this field while the MAA is operating resets the MAA to its default state. 0: Normal operation. 1: Start MAA operation.

Table 27-18: Cryptographic Data In Registers [3:0]

Cryptographic Data In 0 [31:0]			CTB_DIN[0]	[0x0020]
Cryptographic Data In 1 [63:32]			**CTB_DIN[1]**	**[0x0024]**
Cryptographic Data In 2 [95:64]			**CTB_DIN[2]**	**[0x0028]**
Cryptographic Data In 3 [127:96]			**CTB_DIN[3]**	**[0x002C]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Cryptographic Data Input** These registers form the read FIFO for cryptographic operations. Software should not access these registers directly; instead, use CDMA to transfer data in and out of the cryptographic engines. The endian swap input control bit (*CTB_CTRL.bsi*) affects this register.

Table 27-19: Cryptographic Data Out Registers [3:0]

Cryptographic Data Out 0 [31:0]			CTB_DOUT[0]	[0x0030]
Cryptographic Data Out 1 [63:32]			**CTB_DOUT[1]**	**[0x0034]**
Cryptographic Data Out 2 [95:64]			**CTB_DOUT[2]**	**[0x0038]**
Cryptographic Data Out 3 [127:96]			**CTB_DOUT[3]**	**[0x003C]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R	0	**Cryptographic Data Output** These registers form the write FIFO for cryptographic operations. Software should not access these registers directly; instead, use the CDMA to transfer data in and out of the cryptographic engines. The endian swap output control bit (*CTB_CTRL.bso*) affects this register.

Table 27-20: CRC Polynomial Register

CRC Polynomial			CTB_CRC_POLY	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0xEDB8 8320	**CRC 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 (*CTB_CRC_CTRL.msb*).

Table 27-21: CRC Value Register

CRC Value			CTB_CRC_VAL	[0x0044]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	val	R/W	0xFFFF FFFF	**CRC Value** This register holds the result of a CRC calculation. The register is immediately set to 0x0000 0001 if the invalid value of 0x0000 0000 is detected. This register is affected by the MSB control bit (*CTB_CRC_CTRL.msb*).

Table 27-22: Hamming Engine Result Register

Hamming Engine Result			CTB_HAM_ECC	[0x004C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	par	R/W	0	**Hamming Array Parity** 0: Even. 1: Odd.
15:0	ecc	R	0	**Hamming Code Result** These bits are the even parity of their corresponding bit groups.

Table 27-23: Cipher Initial Vector Registers [3:0]

Cipher Initial Vector 0 [31:0]			CTB_CIPHER_INIT[0]	[0x0050]
Cipher Initial Vector 1 [63:32]			CTB_CIPHER_INIT[1]	[0x0054]
Cipher Initial Vector 2 [95:64]			CTB_CIPHER_INIT[2]	[0x0058]
Cipher Initial Vector 3 [127:96]			CTB_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 27-24: Cipher Key Registers [7:0]

Cipher Key 0 [31:0]			CTB_CIPHER_KEY[0]	[0x0060]
Cipher Key 1 [63:32]			CTB_CIPHER_KEY[1]	[0x0064]
Cipher Key 2 [95:64]			CTB_CIPHER_KEY[2]	[0x0068]
Cipher Key 3 [127:96]			CTB_CIPHER_KEY[3]	[0x006C]
Cipher Key 4 [159:128]			CTB_CIPHER_KEY[4]	[0x0070]
Cipher Key 5 [191:160]			CTB_CIPHER_KEY[5]	[0x0074]
Cipher Key 6 [223:192]			CTB_CIPHER_KEY[6]	[0x0078]
Cipher Key 7 [255:224]			CTB_CIPHER_KEY[7]	[0x007C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	key	W	0	**Block Cipher Key Registers** These registers hold the cipher key used for block cipher operations. The number of bits used depends on the specific operation. See *CTB_CIPHER_CTRL.key* field for information on loading this register. This register is affected by the endian swap input control bits.

Table 27-25: Hash Message Digest Registers [15:0]

Hash Message Digest 0 [31:0]	**CTB_HASH_DIGEST[0]**	**[0x0080]**
Hash Message Digest 1 [63:32]	**CTB_HASH_DIGEST[1]**	**[0x0084]**
Hash Message Digest 2 [95:64]	**CTB_HASH_DIGEST[2]**	**[0x0088]**
Hash Message Digest 3 [127:96]	**CTB_HASH_DIGEST[3]**	**[0x008C]**
Hash Message Digest 4 [159:128]	**CTB_HASH_DIGEST[4]**	**[0x0090]**
Hash Message Digest 5 [191:160]	**CTB_HASH_DIGEST[5]**	**[0x0094]**
Hash Message Digest 6 [223:192]	**CTB_HASH_DIGEST[6]**	**[0x0098]**
Hash Message Digest 7 [255:224]	**CTB_HASH_DIGEST[7]**	**[0x009C]**
Hash Message Digest 8 [287:256]	**CTB_HASH_DIGEST[8]**	**[0x00A0]**
Hash Message Digest 9 [319:288]	**CTB_HASH_DIGEST[9]**	**[0x00A4]**
Hash Message Digest 10 [351:320]	**CTB_HASH_DIGEST[10]**	**[0x00A8]**
Hash Message Digest 11 [383:352]	**CTB_HASH_DIGEST[11]**	**[0x00AC]**
Hash Message Digest 12 [415:384]	**CTB_HASH_DIGEST[12]**	**[0x00B0]**
Hash Message Digest 13 [447:416]	**CTB_HASH_DIGEST[13]**	**[0x00B4]**
Hash Message Digest 14 [479:448]	**CTB_HASH_DIGEST[14]**	**[0x00B8]**
Hash Message Digest 15 [511:480]	**CTB_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. The endian swap input control bit (*CTB_CTRL.bsi*) affects these registers.

Table 27-26: Hash Message Size Registers [3:0]

Hash Message Size 0 [31:0]	**CTB_HASH_MSG_SZ[0]**	**[0x00C0]**
Hash Message Size 1 [63:32]	**CTB_HASH_MSG_SZ[1]**	**[0x00C4]**
Hash Message Size 2 [95:64]	**CTB_HASH_MSG_SZ[2]**	**[0x00C8]**
Hash Message Size 3 [127:96]	**CTB_HASH_MSG_SZ[3]**	**[0x00CC]**

Bits	Name	Access	Reset	Description
31:0	msgsz	R/W	0	**Hash Message Size** These four registers define the length of the hash message size in bytes.

Table 27-27: MAA Word Size Register

MAA Word Size	**CTB_MAA_MAWS**	**[0x00D0]**

Bits	Name	Access	Reset	Description
31:12	-	R/W	0	**Reserved**
11:0	size	R/W	0	**MAA Word Size** Set this field to the size of the modular operation. Valid values are from 1 to 2048. Invalid values result in the MAA not starting. *Note: This field is only writable when the MAA is idle (CTB_MAA_CTRL.start = 0).*

Table 27-28: AAD Length Register 0

AAD Length 0	**CTB_AAD_LENGTH[0]**	**[0x00D0]**

Bits	Name	Access	Reset	Description
31:0	length	R/W	0	**AAD Length [31:0]** AAD length in bytes for AES GCM and CCM operations.

Table 27-29: AAD Length Register 1

AAD Length 1	**CTB_AAD_LENGTH[1]**	**[0x00D4]**

Bits	Name	Access	Reset	Description
31:0	length	R/W	0	**AAD Length [63:32]** AAD length in bytes for AES GCM and CCM operations.

Table 27-30: PLD Length Register 0

PLD Length 0			CTB_PLD_LENGTH[0]	[0x00D8]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	length	R/W	0	**PLD Length [31:0]** PLD length in bytes for AES GCM and CCM operations.

Table 27-31: PLD Length Register 1

PLD Length 1			CTB_PLD_LENGTH[1]	[0x00DC]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	length	R/W	0	**PLD Length [63:32]** PLD length in bytes for AES GCM and CCM operations.

Table 27-32: TAG/ MIC Registers [3:0]

TAG / MIC 0 [31:0]			CTB_TAGMIC[0]	[0x00E0]
TAG / MIC 1 [63:32]			CTB_TAGMIC[1]	[0x00E4]
TAG / MIC 2 [95:64]			CTB_TAGMIC[2]	[0x00E8]
TAG / MIC 3 [127:96]			CTB_TAGMIC[3]	[0x00EC]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	length	R/W	0	**TAG/MIC Output** TAG/MIC output for AES GCM and CCM operations.

27.12 User AES Key Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 27-33: User AES Key Register Summary

Offset	Name	Description
[0x0000]	*AES_KEY_AES_KEY0*	*User AES Key 0 Register*
[0x0004]	*AES_KEY_AES_KEY1*	*User AES Key 1 Register*
[0x0008]	*AES_KEY_AES_KEY2*	*User AES Key 2 Register*
[0x000C]	*AES_KEY_AES_KEY3*	*User AES Key 3 Register*
[0x0010]	*AES_KEY_AES_KEY4*	*User AES Key 4 Register*
[0x0014]	*AES_KEY_AES_KEY5*	*User AES Key 5 Register*
[0x0018]	*AES_KEY_AES_KEY6*	*User AES Key 6 Register*
[0x001C]	*AES_KEY_AES_KEY7*	*User AES Key 7 Register*

27.12.1 Register Details

Table 27-34: User AES Key 0 Register

User AES Key 0			AES_KEY_AES_KEY0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 31:0**

Table 27-35: User AES Key 1 Register

User AES Key 1			AES_KEY_AES_KEY1	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 63:32**

Table 27-36: User AES Key 2 Register

User AES Key 2			AES_KEY_AES_KEY2	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 95:64**

Table 27-37: User AES Key 3 Register

User AES Key 3			AES_KEY_AES_KEY3	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 127:96**

Table 27-38: User AES Key 4 Register

User AES Key 4			AES_KEY_AES_KEY4	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 159:128**

Table 27-39: User AES Key 5 Register

User AES Key 5			AES_KEY_AES_KEY5	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 191:160**

Table 27-40: User AES Key 6 Register

User AES Key 6			AES_KEY_AES_KEY6	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 223:192**

Table 27-41: User AES Key 7 Register

User AES Key 7			AES_KEY_AES_KEY7	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 255:224**

27.13 TRNG Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 27-42: TRNG Register Summary

Offset	Register	Name
[0x0000]	*TRNG_CTRL*	TRNG Control Register
[0x0004]	*TRNG_STATUS*	TRNG Status Register
[0x0008]	*TRNG_DATA*	TRNG Data Register

27.13.1 TRNG Register Details

Table 27-43: TRNG Control Register

Cryptographic Control			TRNG_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	keywipe	R/W	0	**AES Key Wipe** Set this field to 1 to wipe the AES key.
14:4	-	RO	0	**Reserved**
3	keygen	R/W	0	**AES Key Generation** Set this field to 1 to generate the AES key.

Cryptographic Control			TRNG_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
1	rnd_ie	R/W	0	**Random Number Interrupt Enable** This bit enables an interrupt to be generated when *TRNG_STATUS.rdy* = 1.
0	-	RO	0	**Reserved**

Table 27-44: TRNG Status Register

TRNG Status			TRNG_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	rdy	R	0	**Random Number Ready** This bit is automatically cleared to 0, and a new random number is generated when *TRNG_DATA.data* is read. 0: Random number generation in progress. The content of *TRNG_DATA.data* is invalid and reads 0. 1: *TRNG_DATA.data* contains new 32-bit random data. An interrupt is generated if *TRNG_CTRL.rnd_ie* = 1

Table 27-45: TRNG Data Register

TRNG Data			TRNG_DATA	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	RO	0	**TRNG Data** The 32-bit random number generated is available in this field when the *TRNG_STATUS.rdy* field is set to 1.

28. Physically Unclonable Function (PUF)

ChipDNA PUF security technology provides an exponential increase in protection against the invasive and reverse engineering attacks by hackers. Attempts to probe or observe ChipDNA operation, modifies the underlying circuit characteristics, preventing the discovery of the unique value used by the chip cryptographic functions. Similarly, more exhaustive reverse-engineering attempts are defeated due to the factory conditioning required to make the ChipDNA PUF circuitry operational. The per-device unique key is generated by the ChipDNA PUF circuitry only when needed for cryptographic operations and is then instantaneously deleted.

Most importantly, the ChipDNA secure key never resides statically in registers or memory, nor does it ever leave the electrical boundary of the IC. In addition to the protection benefits, ChipDNA simplifies or eliminates the need for secure IC key management.

28.1.1 Peripheral Clock Enable

The PUF is disabled by default on a reset. Use of the PUF peripheral requires enabling the peripheral clock. Set *GCR_PCLKDIS1*.*puf* to 0 to enable the peripheral clock to the PUF before configuring the PUF for use.

28.2 Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* 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 28-1: ADC Register Summary

Offset	Register	Description
[0x0000]	*PUF_CTRL*	Control Register
[0x0004]	*PUF_STAT*	Status Register

28.2.1 Register Details

Table 28-2: PUF Control Register

PUF Control			PUF_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
25	key1_dn_ie	R/W	0	**Key 1 Done Interrupt Enable** 0: Interrupt disabled. 1: Interrupt enabled.
24	key0_dn_ie	R/W	0	**Key 0 Done Interrupt Enable** 0: Interrupt disabled. 1: Interrupt enabled.
23:17	-	RO	0	**Reserved**
17	keygen_err_ie	R/W	0	**Key Generation Error Interrupt Enable** 0: Interrupt disabled. 1: Interrupt enabled.
16:10	-	RO	0	**Reserved**
9	key1_gen_en	R/W	0	**Key 1 Generation Enable** Set this field to 1 to generate PUF key 1.
8	key0_gen_en	R/W	0	**Key 0 Generation Enable** Set this field to 1 to generate PUF key 0.
7:2	-	RO	0	**Reserved**
1	key_clr_en	R/W	0	**Key Clear** Set this field to 1 to clear the PUF keys.

PUF Control			PUF_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
0	puf_en	R/W	0	**PUF Enable** 0: Disabled. 1: Enabled.

Table 28-3: PUF Status Register

PUF Status			PUF_STAT	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
25	key1_dn	R/W	0	**Key 1 Generation Complete** This field is set to 1 by hardware when PUF key 1 generation is completed. Write 1 to clear this field. This field should be cleared by software before starting a key 1 generation operation (*ADC_CTRL0.key1_gen_en* = 1). 0: Normal operation. 1: Event occurred.
24	key0_dn	R/W1C	0	**Key 0 Generation Complete** This field is set to 1 by hardware when PUF key 0 generation is completed. Write 1 to clear this field. This field should be cleared by software before starting a key 0 generation operation (*ADC_CTRL0.key0_gen_en* = 1). 0: Normal operation. 1: Event occurred.
23:18	-	RO	0	**Reserved**
17	key1_cnst_err	R/W1C	0	**Key 1 Constant Error**
16	key1_init_err	R/W1C	0	**Key 1 Initialization Error**
15:10	0	RO	0	**Reserved**
9	key0_cnst_err	R/W1C	0	**Key 0 Constant Error**
8	key0_init_err	R/W1C	0	**Key 0 Initialization Error**
7	adc_freq_fl	R/W1C	0	**ADC Frequency Flag**
6:4	-	RO	0	**Reserved**
3	keygen_err	R/W1C	0	**Key Generation Error** This field is set to 1 by hardware if an error occurs during key generation for either key 0 or 1. Write 1 to clear this field. 0: Normal operation. 1: Event occurred.
2	keygen_en_err	R/W1C	0	**Key Generation Enable Error** 0: Normal operation. 1: Event occurred.
1	magic_err	R/W1C	0	**Magic Word Error** 0: Normal operation. 1: Event occurred.
0	busy	R/W1C	0	**PUF Busy Flag**

29. 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. The SCPBL stores the signed CRK and MRK values in OTP.

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.

The device tests for a bootloader stimulus following every reset and wake up from all low-power modes except SLEEP and DEEPSLEEP. The stimulus includes receipt of the synchronization pattern and may or may not involve a physical pin, as shown in *Table 29-1*. If the stimulus is not present, the device executes the secure boot verifying the digital signature to ensure program memory is unaltered before beginning code execution.

29.1 Development Tools

The information in this chapter explains the details of the SCP protocol and the secure boot process. A software development kit (SDK) provides tools that build 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. Refer to the Secure Boot Tool User Guide for the MAX32690 for more information:
https://pdfserv.maximintegrated.com/en/an/ug7637-secure-boot-tools.pdf

The devices provided in the evaluation kit come preloaded with a test CRK (TCRK) and demonstration software.

29.2 Instances

There is only one instance of the SCPBL.

Table 29-1 shows the interface pins associated with the SCPBL. The hardware must ensure access to the default interface/stimulus pins (and a secondary interface and stimulus pin if implemented) if the application uses the SCPBL. There is no way to start an SCPBL session without setting the stimulus pin to its active state.

Table 29-1: MAX32690 Bootloader Physical Interface

Interface	Interface ID	Interface Pins Required For SCPBL	Default Stimulus Condition	Stimulus Pin Reassignment Options	Default SCPBL Activation Interval
USB	0x55	DP DM RSTN	P4.0 (active low)	All GPIO	6s
UART (115200 bps) 8-N-1 (SCPBL) 8-N-2 (Host)	0x00	UART0A_RX (P3.0) UART0A_TX (P3.1) RSTN			

Multiple methods and algorithms ensure the integrity and authority of SCPBL communication, as shown in *Table 29-2*.

Table 29-2: MAX32690 Data Security and Integrity Methods

Item	Location	Range	Option	Length (Bytes)
Header ID	Start of transport layer header	N/A	0x48,0x49,0x53,0x57,0x45,0x44,0x47,0x44	8
Application Image Digital Signature	End of the application image	Entire application image, excluding the signature itself	ECDSA-256	64
Header Checksum	End of header	The first 7 bytes of the transport header, then padded with 9 bytes of 0x00	The header checksum is AES-128 encryption with a 16-byte key of 0x00. The value is the MSB of the 16-byte digest.	1

Item	Location	Range	Option	Length (Bytes)
Data Checksum	End of transport layer	Session layer data	The data checksum is AES-128 encryption with a 16-byte key of 0x00. The value is the 4 MSB of the 16-byte digest.	4
Session Layer Protection Profile	The second nibble of the session layer	Session layer	0xA: ECDSA-256	4 bits
MRK/CRK/TCRK	Bootloader authentication keys	N/A	ECDSA-256	64

Detection of a packet without a valid signature may disable the SCPBL and prevents code execution. This condition clears when all power supply and battery inputs, including V_{RTC}, are taken to 0V and then returned to valid levels. The device then resets and operates normally. Removing the power supply and battery inputs clears all battery-backed state information and all SRAM, including volatile AES keys. Still, it preserves the nonvolatile SCPBL certificate, configuration, life cycle state, and flash program memory.

29.3 Secure Boot

Following any reset, the device executes a secure boot from the ROM to verify the integrity of the code in flash memory. This establishes the chain of trust that secure application execution relies on.

- The platform knows the software has not been accidentally or maliciously modified.
- The platform knows the software is from a trusted, authorized source.
- The software ensures it is running on a trusted platform.

A device with the secure boot feature must be initialized through the SCPBL before it executes the software:

- The application image must be created and signed with the CRK.
- The CRK must be loaded through the SCPBL.
- The application image must be loaded through the SCPBL.

Before executing any code in flash memory, the digital signature of the application image in the flash memory is calculated using the loaded CRK. If the calculated value matches the digital signature stored at the end of the application image, the device begins the execution of the software.

The device does not execute software if the calculated value does not match the digital signature.

29.4 Secure Product Life Cycle Management

29.4.1 Purpose

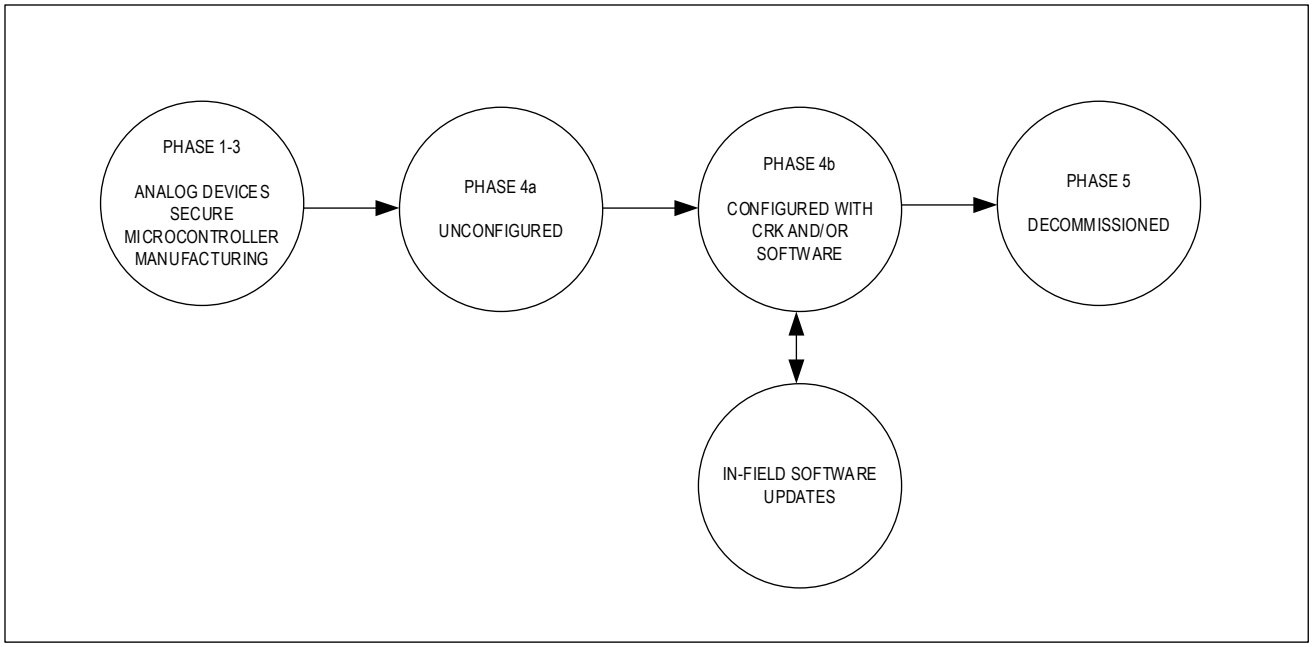
The secure product life cycle scheme implemented by the SCPBL protects the security of the product in multiple ways:

- It verifies all security measures on a device before being released to the customer.
- It prevents a device from properly operating until configured by the customer.
- It enables device programming in an unsecured environment (manufacturing location or in-field updates).
- It provides for end-of-life decommissioning to securely control the application life cycle.

29.4.2 Life cycle stages

The device supports multiple life cycle phases, as shown in *Figure 29-1*.

Figure 29-1: Life Cycle Phases



The phases are as follows:

- Phases 1 to 3 are executed in a controlled manufacturing facility to guarantee the device is in a secure, protocol-controlled state before delivery to the customer.
- Phase 4a. Devices are delivered to the customer in this state without a CRK.
- Phase 4b. The device transitions to this phase after the customer has loaded the CRK signed by Analog Devices. At this point, load a properly signed application image 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 protects against tampering and unauthorized access.
- Phase 5: Previously loaded software does not execute, and program memory cannot be loaded or verified.

The HELLO_REPLY command reports the current life cycle phase.

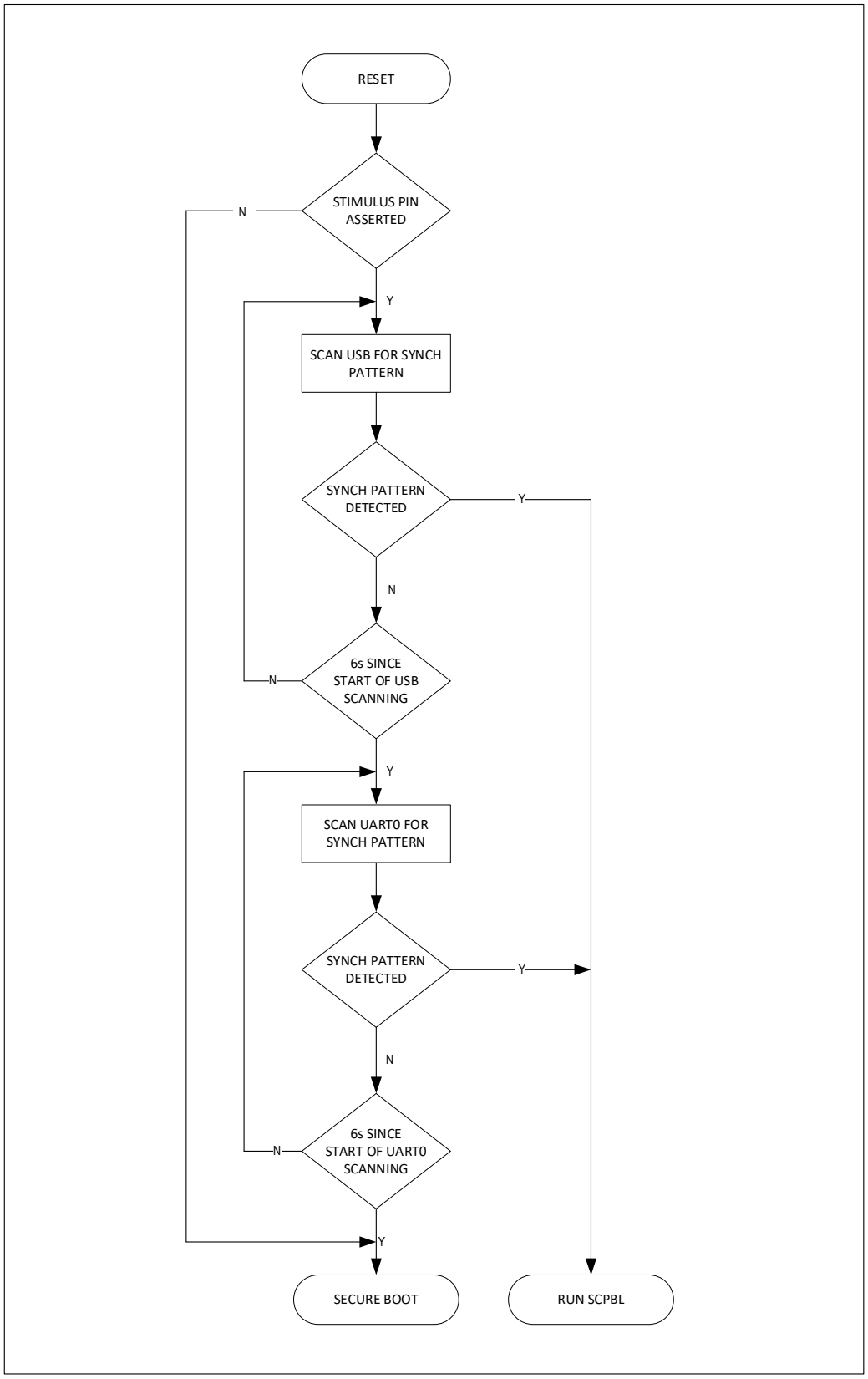
29.5 MAX32690 Bootloader Activation

Two conditions are required following a reset or wake-up from any low-power mode except SLEEP and DEEPSLEEP to activate the bootloader:

- Detection of the stimulus, as described in *Figure 29-2.*
- Detection of the SYNCH pattern on the active interface.

If a stimulus condition is present, the device searches the active interface for the synchronization pattern, as shown in *Figure 29-2*. The device executes the secure boot process if the stimulus is not active or the pattern is not seen.

Figure 29-2: MAX32690 Bootloader Flow



The device tests for the stimulus following each reset or wake-up event unless specified otherwise. It may test for the synchronization pattern multiple times during the timeout period. Once an SCPBL session begins, the stimulus pin must remain asserted until the host terminates the session.

Most devices can permanently reassign the stimulus pin to another GPIO once an SCPBL session is established. This allows access to the alternate functions of the default stimulus pin, if desired.

Following most stimulus events, some GPIO default to a high-impedance input. If a GPIO pin is the stimulus, it must be actively driven high or low by the hardware to the desired state before exiting any wake-up event or reset, including a POR.

Activation of the bootloader may overwrite some or all portions of SRAM. SRAM contents loaded before activation of the bootloader are not guaranteed to be preserved during or after SCPBL operation.

29.6 Root Key Management

The Root of Trust is a public-key infrastructure involving several key pairs.

- The manufacturer root key (MRK) pair.
- The customer root key (CRK) pair.
- The test customer rook key (TCRK) pair.

29.6.1 Manufacturer Root Key (MRK)

The factory owns the MRK pair. The public key is stored in the device and authenticates the CRK. The private key is used to sign the CRK. This certificate is used to authenticate and identify the source.

The MRK private key is stored in a restricted-access hardware security module (HSM) compliant with most security requirements.

29.6.2 Customer Root Key (CRK)

The CRK is used to sign production-ready end products. The customer generates their own CRK keypair, typically using 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.

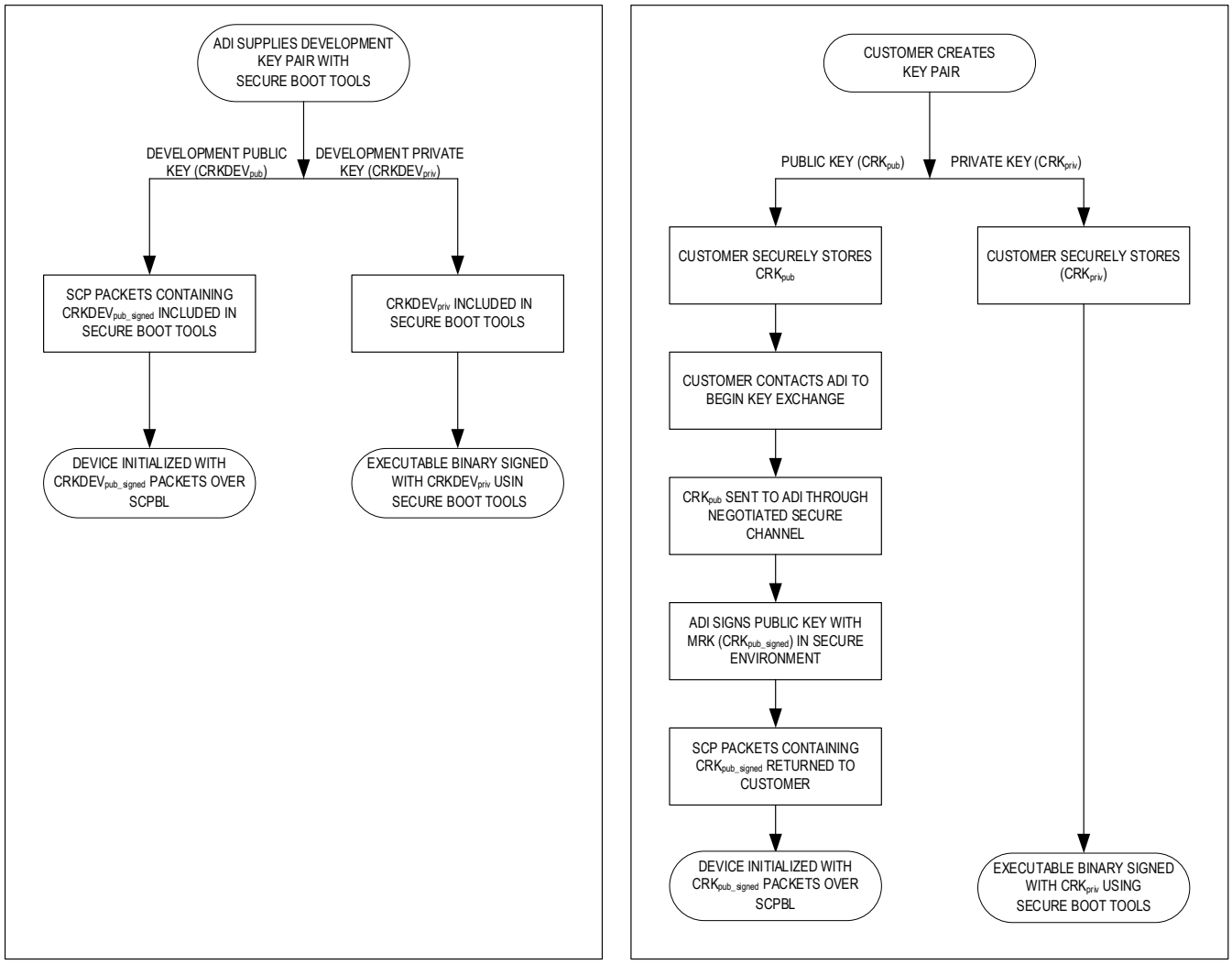
29.6.3 Test CRK (TCRK)

The SDK provides a particular version of the CRK, called the test root key, which is used during development. The use of this key eliminates the need to generate a custom CRK and have it signed by the manufacturer during the development phase. 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.

The evaluation kits for the device come preloaded with the test root key.

Figure 29-3 shows a high-level overview of the key generation and development described in the application note.

Figure 29-3: Customer Root and Development Key Generation and Usage



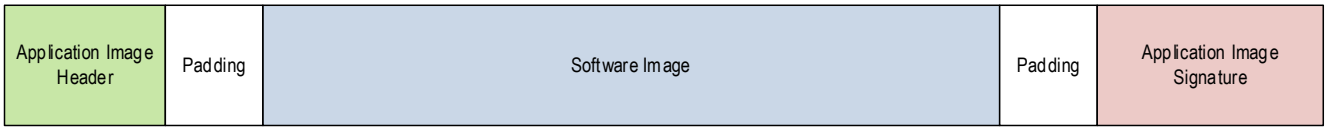
29.7 Building the Application Image

The application image is physically programmed into the program memory, regardless of the transmission protocol. The SBT provides applications to break the application image into smaller "chunks" and converts them to SCP packets.

The application image shown in *Figure 29-4* contains multiple parts:

- The application image header.
- The software image that is the compiled customer code.
- A digital signature to verify the integrity and authenticate the application image header and the software image.
- Padding as needed to ensure the start and end of the software image are aligned with the required boundaries.

Figure 29-4: Application Image Structure



The elements of the image are shown graphically in *Table 29-3*.

Table 29-3: Application Image Structure

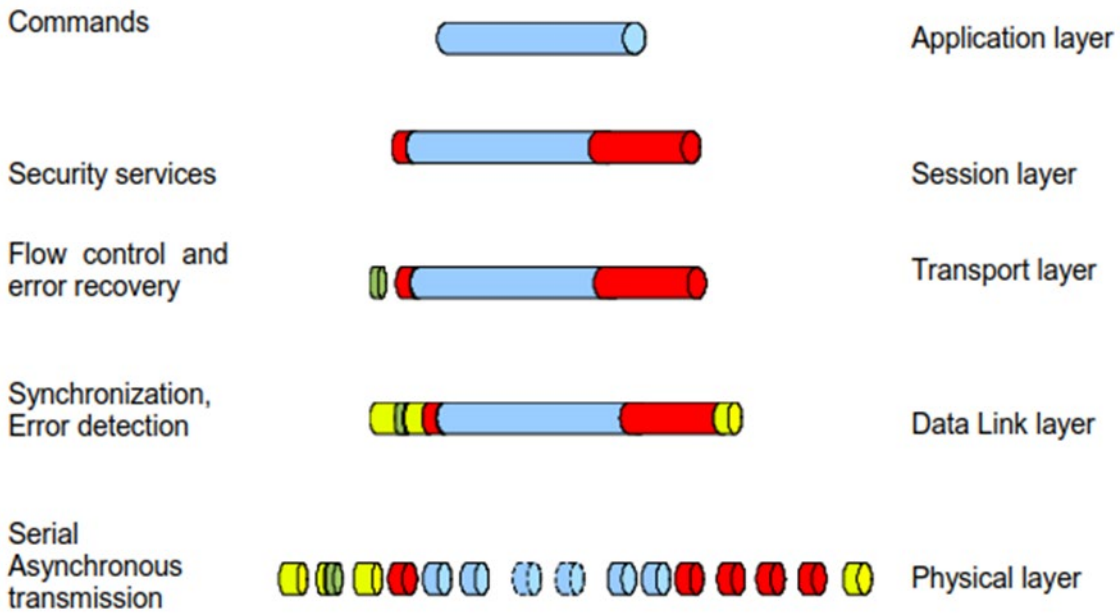
Element	Absolute Start Address Within Application Image	Length (Bytes)	Value
Header ID	0x0000 0000	8	See *Table 29-2*.
Format Version (ROM_REF)	0x0000 0008	4	0x0100 0000 (default defined by device version)
Application Image Start Address	0x0000 000C	4	0x1000 0000
Application Image Signature Offset	0x0000 0010	4	This is the start address of the digital signature located at the end of the software image plus any terminal padding (if necessary).
Executable Start Address	0x0000 0014	4	0x0000 0200 This is a pointer to the first instruction when code execution begins.
Argument Size	0x0000 0018	4	0x0000 0000
Application Version	0x0000 001C	4	User-defined (default convention used by the Secure Boot Tools is 0x0100 0000 corresponding to Version 1.0.0)
Padding	0x0000 0020	480	Padded with 0xFF out to 0x0000 01FF
Software Image	0x0000 0200	Variable	Software image
Terminal padding to 4B boundary, if necessary	End of the software image	0 1 2 3	N/A 0xFF 0xFFFF 0xFFFFFF
Application Image Signature	End of software image + terminal padding	256 bits	ECDSA-256 signature

29.8 SCP Session

The SCP is a session-based protocol that securely exchanges data packets between the host and SCPBL.

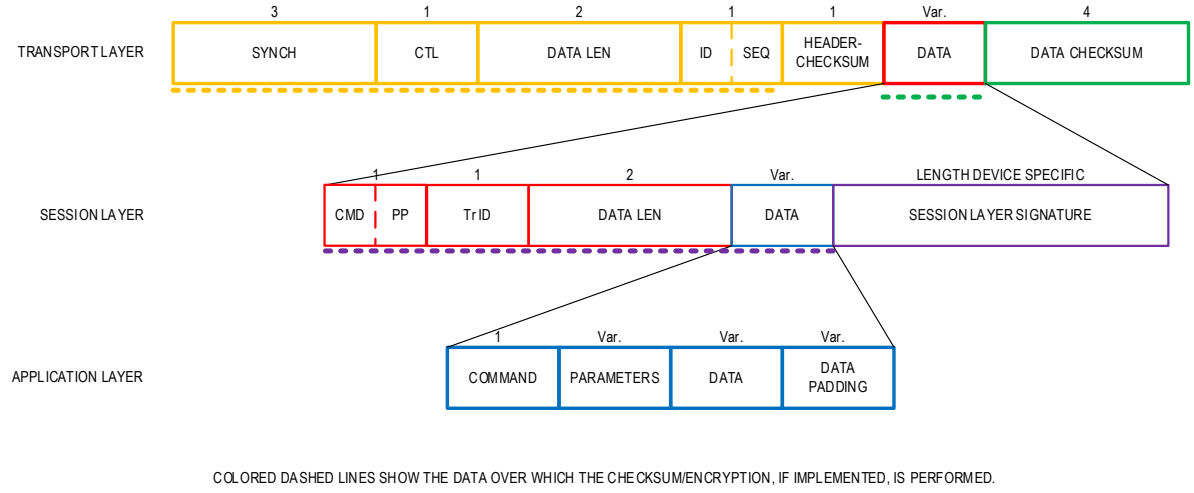
The SCP consists of a stack of layers based on the OSI model shown in *Figure 29-5*. The application layer contains basic commands to configure the SCPBL. It also includes the signed customer application and a set of WRITE data commands used for the SCPBL, including the write command that loads the code (the executable software) into the program memory.

Figure 29-5: SCPBL Implementation of OSI Model



The general structure of a packet is shown in *Figure 29-6*. Simpler commands only require a subset of these elements. Multiple integrity and authentication steps are used throughout an SCP packet.

Figure 29-6: SCP Packet Structure



29.8.1 Physical Layer

At this layer, the data between the SCPBL and the host is transmitted over the active communication interface. The stream of data is subdivided into Protocol Data Units (PDU) called bytes. When a framing error occurs, the data byte is discarded.

The UART interface is fixed at the 8-N-1 format operating at 115,200 bps. Hardware flow control is not implemented.

29.8.2 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).

Each frame contains a header identifying the type of packet and is shown in yellow in *Figure 29-6*, followed by an optional data portion.

Table 29-4: Transport/Session Layer Header Structure

Segment	Offset	Length (Bytes)	Value
Synchronization Pattern (SYNCH)	0x0000 0000	3	0xBEEFED
Segment type (CTL)	0x0000 0003	1	Transport layer packet type
Data Length (DATA LEN)	0x0000 0004	1	Data link segment length
Channel ID (ID)	0x0000 0005	Upper 4 bits	The "channel identifier" is provided by the host to identify an active connection and must be the same for SCPBL and host segments. This "channel identifier" is fixed for the whole session.
Sequence Number (SEQ)		Lower 4 bits	The lower 4 bits of this byte serve as an acknowledgment from the SCPBL that the last segment is 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 (HEADER CHECKSUM)	0x0000 0006	1	The calculation of the header checksum is described in *Table 29-2.*

29.8.3 Transport Layer

The Transport Layer provides security, data recovery, and flow control. To allow reliable communication, 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:

- A sequential numbering of the segments
- A positive acknowledgment (ACK command) of command receipt
- A response timeout limit of approximately 10 seconds between the SYNCH pattern of one packet and the next. The SCPBL session is terminated if the timeout expires.

29.8.4 Session Layer

The Session Layer manages security issues by encrypting and signing the data. At this layer, a Protocol Data Unit (PDU) is called Data.

The Data has a variable length.

29.9 Sequence and Transaction ID

The SCP incorporates two indexing schemes as an error-detection mechanism:

- The session ID in the header of the transport layer
- Transaction ID in the header of the session layer

The SCPBL automatically increments its internal counter following the ACK of certain data transfer commands. The host software, following the same rules, must increment its internal counter at the same time 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 transaction ID is incremented after each successful data transfer at the session level from the host to the SCPBL.

These incrementing of the SEQ and TrID values are summarized in *Table 29-6*.

29.10 Opening an SCP Session Example

The host initiates a new session request by sending a COM_REQ to the SCPBL. The procedure is shown in *Table 29-5*. An error is generated if the host attempts to start a session with the CON_REQ command when a session is already in progress.

Table 29-5: Session Opening Protocol

Host		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		0	N/A
			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				

29.11 SCPBL Command Summary

Table 29-6: SCPBL Command and Sequencing Summary

Command	CTL	CMD	Command Value	Command Generates an ACK Response	SEQ increments After Command ACK	TrID increments After COMMAND_RSP ACK	Description
Transport Layer Commands							
CON_REQ	0x01	N/A	N/A	No	-	-	Connection request from the host
CON_REP	0x02	N/A	N/A	Yes	No	No	Connection acceptance from the SCPBL
DISC_REQ	0x03	N/A	N/A	No	-	-	Disconnection request from the host
DISC_REP	0x04	N/A	N/A	No	-	-	Disconnection acceptance from the SCPBL
ACK	0x06	N/A	N/A	n/a	-	-	Command acknowledgment between the host and SCPBL.
Data Transfer Status Commands							
HELLO	0x05	0x1	N/A	Yes	Yes	No	Status request from the host
HELLO_REPLY	0x05	0x2	N/A	Yes	Yes	No	Status reply from the SCPBL
COMMAND_RSP	0x05	0x5*	N/A	Yes	n/a	n/a	Command success/failure response
Data Transfer Application Layer Commands							
WRITE_CRK	0x05	0x5	0x470A	Yes	Yes	Yes	Write CRK to Device
REWRITE_CRK/RENEW_CRK	0x05	0x5	0x461A	Yes	Yes	Yes	Write a Second Write CRK to the Device
WRITE_OTP	0x05	0x5	0x4714	Yes	Yes	Yes	Write to OTP
WRITE_TIMEOUT	0x05	0x5	0x4426	Yes	Yes	Yes	Set SCPBL Activation Timeout Interval
KILL_CHIP2	0x05	0x5	0x4539	Yes	Yes	Yes	Permanently disable device

Command	CTL	CMD	Command Value	Command Generates an ACK Response	SEQ increments After Command ACK	TrID increments After COMMAND_RSP ACK	Description
WRITE_PARAMS	0x05	0x5	0x4427	Yes	Yes	Yes	Write Parameters to Configure Communication Link
WRITE_STIM	0x05	0x5	0x4428	Yes	Yes	Yes	Configure SCPBL Stimulus Pin
WRITE_DEACTIVATE	0x05	0x5	0x4429	Yes	Yes	Yes	Permanently Deactivate SPCPBL Interface
WRITE_DATA	0x05	0x5	0x2402	Yes	Yes	Yes	Write Data to Memory
COMPARE_DATA	0x05	0x5	0x2403	Yes	Yes	Yes	Compare Data Against Internal Memory
ERASE_DATA	0x05	0x5	0x4401	Yes	Yes	Yes	Erase the Range of Flash Memory
EXECUTE_CODE	0x05	0x5	0x2101	Yes	Yes	Yes	Execute Code from Flash Memory
** These commands have the same CTL and CMD number but can be distinguished by context.*							

29.12 Transport Layer Command Details

29.12.1 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.

Figure 29-7: CON_REQ Command Structure

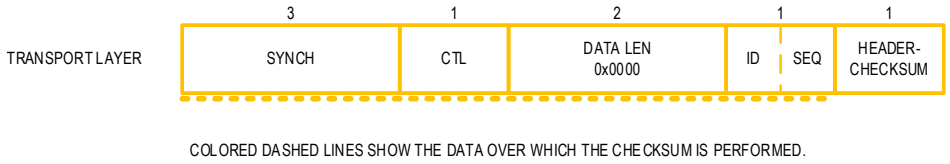


Table 29-7: CON_REQ

CON_REQ	0x01	Connection Request	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x01
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	User-defined. The channel ID value used for this command is used by all subsequent commands in the session.
SEQ		Lower 4 bits	0x0
HEADER CHECKSUM	0x07	1	Header checksum

29.12.2 CON_REP

The SCPBL sends a confirmation to the host in response to a CON_REQ command.

The host responds with an ACK. The SEQ field is not incremented by the ACK response to this command.

The sequence number is always 0b0000 for this command.

Figure 29-8: CON_REP Command Structure

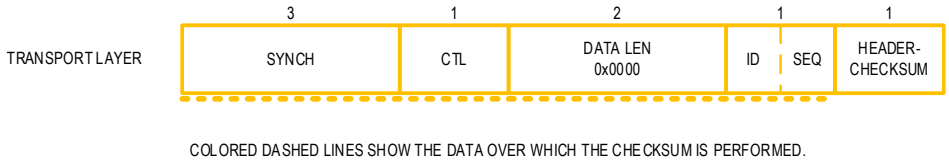


Table 29-8: CON_REP

CON_REQ	0x02	Connection Reply	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x02
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	0x0
HEADER CHECKSUM	0x07	1	Header checksum

29.12.3 DISC_REQ

The host sends a DISC_REQ to the SCPBL to terminate the SCP session. The SCPBL responds with an ACK.

The sequence number is not changed by this command.

Figure 29-9: DISC_REQ Command Structure

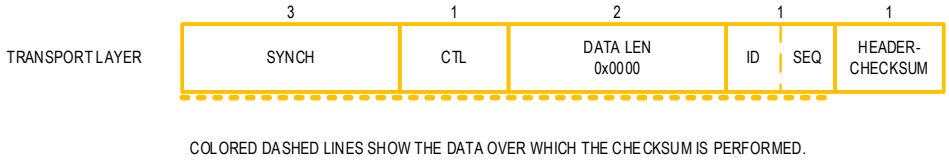


Table 29-9: DISC_REQ

DISC_REQ	0x03	Disconnection Request	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x03
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	Varies
HEADER CHECKSUM	0x07	1	Header checksum

29.12.4 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 conditions are still present.

The sequence number is not changed by this command.

Figure 29-10: DISC_REP Command Structure

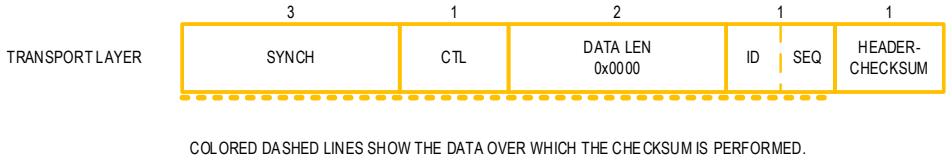


Table 29-10: DISC_REP

DISC_REQ	0x04	Disconnection Reply	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x04
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	Same as the preceding *DISC_REQ*.
HEADER CHECKSUM	0x07	1	Header checksum

29.12.5 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. The sequence number is incremented following the ACK command for the next command.

Figure 29-11: ACK Command Structure

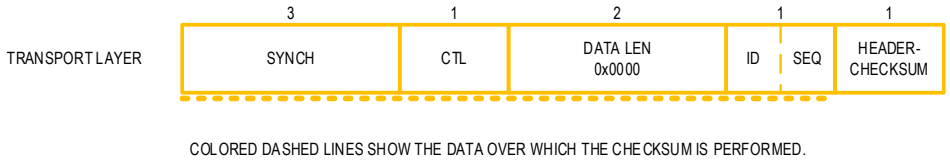


Table 29-11: ACK

ACK	0x06	Command Acknowledgement	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x06
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	Same as the sequence number of the preceding command
HEADER CHECKSUM	0x07	1	Header checksum

29.12.6 HELLO

The HELLO command is sent by the host to the SCPBL as part of the sequence to establish a new session. The command is a unique version of the DATA_TRANSFER command with a fixed payload. The SCPBL responds with an ACK followed by HELLO_REPLY.

The HELLO command does not include a signature following the session layer data like other DATA_TRANSFER commands.

Figure 29-12: HELLO Command Structure

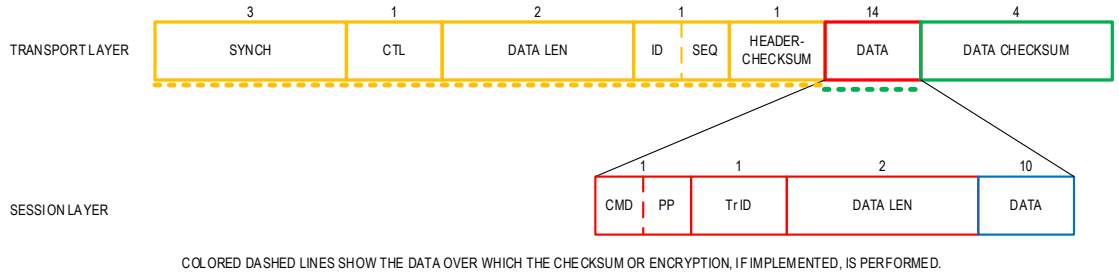


Table 29-12: HELLO Command

HELLO	0x05	Request Configuration Information	
Command Format (Transport Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x05
DATA LEN	0x04	2	0x000E
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		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**
Command	0x00	4 bits	0x1
PP		4 bits	0b0000 (there is no session layer signature for this command.)
TrID	0x01	1	Variable
DATA LEN (SESSION)	0x02	2	0x000A
DATA	0x04	10	0x00: 'H' 0x01: 'E' 0x02: 'L' 0x03: 'L' 0x04: 'O' 0x05: 0x20 0x06: 'B' 0x07: 'L' 0x08: 0x03 0x09: 0x02

29.12.7 HELLO_REPLY

A HELLO_REPLY is sent by the SCPBL to the host in response to the HELLO command. The command provides information about the device configuration, including:

- ROM version
- Life cycle information
- JTAG status
- USN

Figure 29-13: HELLO_REPLY Structure

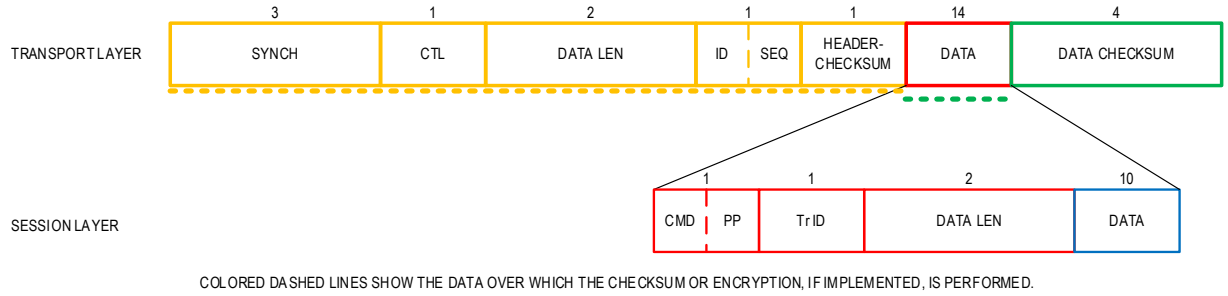


Table 29-13: HELLO_REPLY Command

HELLO_REPLY	0x05	Return Configuration Information
Command Format (Transport Layer)		
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x05
DATA LEN	0x04	2	0x0036
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	Varies
HEADER CHECKSUM	0x07	1	Header checksum
Command Format (Session Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
CMD	0x00	Upper 4 bits	0x2
PP		Lower 4 bits	0b0000 (there is no session layer signature for this command.)
TrID	0x01	1	Variable
DATA LEN (SESSION)	0x02	2	0x0032
DATA	0x04	10	0x00 – 0x9: 0x72, 0x69, 0x76, 0x76, 0x79, 0x32, 0x72, 0x79, 0x83, 0x84 0x0A - 0x0D: ROM Version 0x0E: Lifecycle 0x0F: 0x00 0x10: 0x00 0x11: Configuration 0b0xxxxxxx: JTAG Enabled 0b1xxxxxxx: JTAG Disabled 0bxxx1xxxx: No CRK loaded 0bxxx0xxxx: CRK loaded 0x12 - 0x1F: USN 0x20 - 0x31: 0x00

29.12.8 DATA TRANSFER COMMANDS

This type of command is how the application layer commands (which include the loading of program memory) are passed through the SCP.

Most of these commands implement the session-level signature to verify and authenticate the host.

Figure 29-14: DATA TRANSFER Command Structure

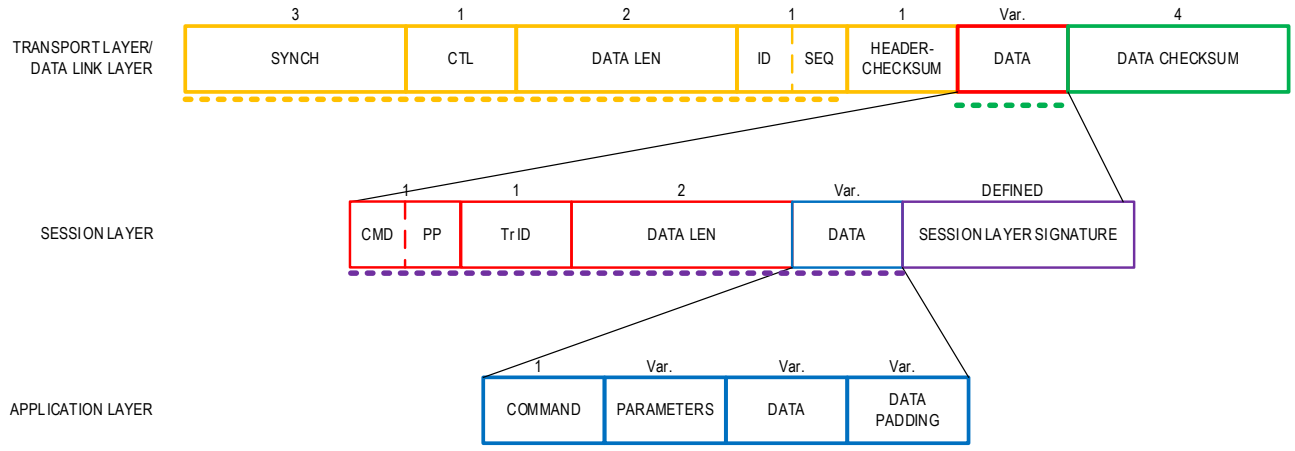


Table 29-14: DATA TRANSFER Command Example

WRITE_DATA	0x06	Write Data To Memory or Configuration Command	
Command Format (Transport Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x05
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		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	Upper 4 bits	0x5
PP		Lower 4 bits	See *Table 29-2* for the protection profile value.
TrID	0x01	1	Varies
DATA LEN (SESSION)	0x02	2	User-defined
DATA	0x04	10	User-defined
SESSION LAYER SIGNATURE	0x0E	1	Defined by the device.

29.12.9 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 the successful execution of the command. Any other value indicates an error, which may or may not be specific to the command.

This is a DATA TRANSFER command.

Figure 29-15: COMMAND_RSP Command Structure

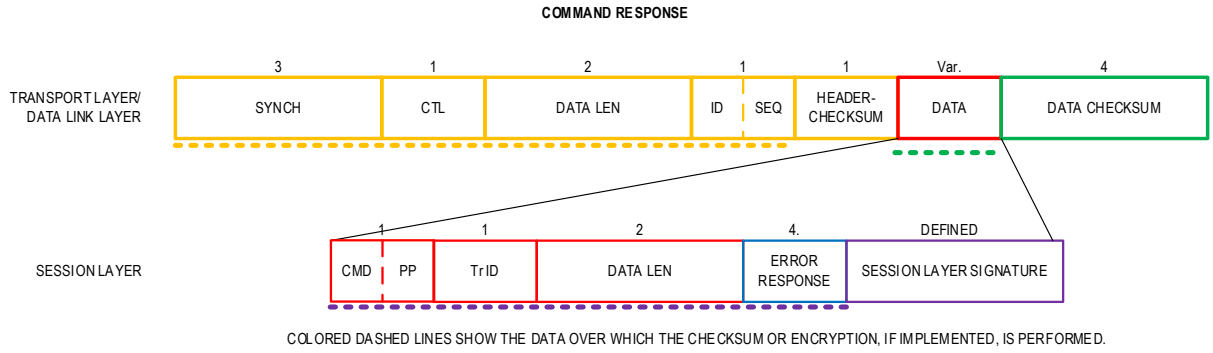


Table 29-15: COMMAND_RSP

COMMAND_RSP	0x05	Success/Failure Response from Application Layer Command	
Command Format (Transport Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x05
DATA LEN	0x04	2	0x0008
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		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	Upper 4 bits	0x5
PP		Lower 4 bits	See *Table 29-2* for the protection profile value.
TrID	0x01	1	Varies
DATA LEN (SESSION)	0x02	2	0x0004
RESPONSE	0x04	4	0x00000000: Success Any other value: Failure

29.13 Application Layer Command Details

The application layer deals with the commands used to directly modify the memory and configure various functions. The application layer is shown in *Figure 29-6* and is blue.

The memory commands WRITE_DATA, ERASE_DATA, and COMPARE_DATA directly address the internal flash and RAM based on the target address.

The execute command is used indirectly as part of the secondary-level applets that perform WRITE DATA, ERASE DATA, and COMPARE DATA commands to the external memory. 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.

Other application layer commands are associated with device configuration.

29.13.1 WRITE_CRK

Table 29-16: WRITE_CRK

WRITE_CRK	0x4701	Write CRK to the Device	
Description	This command writes the CRK to the nonvolatile memory of the SCPBL. This command can only be executed once per device. Use the REWRITE_CRK command to write a second CRK, allowing a maximum of two CRK values. The CRK value is the same as the MRK until this command is executed. The new CRK value is used when the current session terminates, and a new one is started.		
Command Format			
Element	**Start Address**	**Length**	**Values**
CMD	0x0000	2	0x4701
Key Length	0x0002	2	0x0204
CRK	0x0004	64 Bytes	Public Key (ECDSA-256)
MRK Signature	0x0044	64 Bytes	MRK Signature of the CRK using the public key (ECDSA-256)

29.13.2 REWRITE_CRK/RENEW_CRK

Table 29-17: REWRITE_CRK/RENEW_CRK

REWRITE_CRK RENEW_CRK	0x461A	Write a Second Write CRK to the 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.		
Command Format			
Element	**Start Address**	**Length**	**Values**
CMD	0x0000	2	0x461A
Key Length	0x0002	2	0x0204
Previous CRK	0x0004	64	Previous CRK (ECDSA-256)
New CRK	0x0044	64	New CRK (ECDSA-256)
MRK Signature	0x0084	64	MRK signature (ECDSA-256)

29.13.3 WRITE_OTP

Table 29-18: WRITE_OTP

WRITE_OTP	0x4714	Write Data to OTP Memory	
Description	This reserved command may only be used when specifically instructed by the factory.		
Command Format			
Element	**Start Address**	**Length**	**Values**
CMD	0x0000	2	0x4714
Arguments	0x0002	Variable	

29.13.4 WRITE_TIMEOUT

Table 29-19: WRITE_TIMEOUT

WRITE_TIMEOUT	0x4426	Set SCPBL Activation Timeout Interval	
Description	This command specifies the amount of time in milliseconds that the device waits for a valid SYNCH pattern following the detection of an active SCPBL activation pin. If no SYNCH pattern is received within the timeout interval, then the SCPBL does not, and instead performs a secure boot. If the secure boot option is not available, the device begins program execution. The device uses the default timeout value shown in *Table 29-1* until changed with this command. This command is ignored if a device does not have a default activation pin. This command can only be executed once for each interface.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
CMD	0x0000	2	0x4426
Interface ID	0x0002	1	See *Table 29-1*.
Value	0x0003	2	This is the timeout value in milliseconds, from 0x0001 to 0xFFFE. An interval of 0x0000 03E8 corresponds to 1s. The timeout value is sent LSB first.

29.13.5 KILL_CHIP2

Table 29-20: KILL_CHIP2

KILL_CHIP2	0x4539	Disable Device	
Description	This command permanently disables the SCPBL and prevents the execution of any code in the flash memory. SCPBL execution stops immediately without waiting for a session closure.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
CMD	0x0000	2	0x4539
USN	0x0002	13	USN of the device

29.13.6 WRITE_PARAMS

Table 29-21: WRITE_PARAMS

WRITE_PARAMS	0x4427	Write Parameters to Configure Communication Link	
Description	This reserved command is for factory configuration only.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x4427
Instance ID	0x0002	1	See *Table 29-1*.
Data	0x0003	Variable	The length of the field is device-specific.

29.13.7 WRITE_STIM

Table 29-22: WRITE_STIM

<table>
<tr><th>WRITE_STIM</th><th>0x4428</th><th colspan="9">Configure SCPBL Stimulus Pin (GPIO0-GPIO3)</th></tr>
<tr><td rowspan="4">Description</td><td colspan="10">This command allows the SCPBL to permanently change the location and polarity of the SCPBL stimulus pin.
Valid pin locations are shown in Table 29-1.
Regardless of specific settings, the value of this byte must not be 0xFF.
This command can only be executed once.</td></tr>
<tr><td>Bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td></td><td colspan="2">GPIO Port</td><td>Active State</td><td colspan="5">GPIO Pin of Selected Port</td></tr>
<tr><td>Function</td><td colspan="2">0b00: Port 0
0b01: Port 1
0b10: Port 2
0b11: Port 3</td><td>0: Low
1: High</td><td colspan="5">0x00: Pin 0
0x01: Pin 1
0x02: Pin 2
0x03: Pin 3
0x04: Pin 4
0x05: Pin 5
0x06: Pin 6
0x07: Pin 7
0x08: Pin 8
0x09: Pin 9
0x0A: Pin 10
0x0B: Pin 11
0x0C: Pin 12
0x0D: Pin 13
0x0E: Pin 14
0x0F: Pin 15
0x10: Pin 16
0x11: Pin 17
0x12: Pin 18
0x13: Pin 19
0x14: Pin 20
0x15: Pin 21
0x16: Pin 22
0x17: Pin 23
0x18: Pin 24
0x19: Pin 25
0x1A: Pin 26
0x1B: Pin 27
0x1C: Pin 28
0x1D: Pin 29
0x1E: Pin 30
0x1F: Pin 31</td></tr>
<tr><td colspan="10"></td></tr>
<tr><th colspan="11">Command Format</th></tr>
<tr><th>Segment</th><th>Start Address</th><th>Byte Length</th><th colspan="8">Values</th></tr>
<tr><td>Command</td><td>0x0000</td><td>2</td><td colspan="8">0x4428</td></tr>
<tr><td>Stimulus Location</td><td>0x0002</td><td>1</td><td colspan="8">See Table 29-1 for valid pin options.</td></tr>
</table>

29.13.8 WRITE_SLA_VERSION

Table 29-23: 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 is 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 is set yet by this command.	
Command Format		
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x470B
Revision	0x0002	4	User-defined 4-byte value. (default 0x0000 0000)

29.13.9 WRITE_DEACTIVATE

Table 29-24: WRITE_DEACTIVATE

WRITE_DEACTIVATE	0x4429	Permanently Deactivate SPCPBL Interface	
Description	This command deactivates an SCPBL link. This command can be used once for each link. Once deactivated, a link cannot be reactivated.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x4429
Interface ID	0x0002	1	See *Table 29-1* for the instance ID of the desired interface

29.13.10WRITE_DATA

Table 29-25: WRITE_DATA

WRITE_DATA	0x2402	Write Data to Memory	
Description	The command writes data into the internal flash or RAM based on the target address.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x2402
Start Address	0x0002	4	This is the target start address. The MSB is sent first.
Length	0x0006	4	This is the length of the data segment in bytes. The MSB is sent first.
Data	0x000A	Variable	Write Data

29.13.11COMPARE_DATA

Table 29-26: COMPARE_DATA

COMPARE_DATA	0x2403	Compare Data Against Internal Memory	
Description	This command compares the supplied data against the internal flash or RAM contents based on the target address.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x2403
Start Address	0x0002	4	This is the target start address. The MSB is sent first.
Length	0x0006	4	This is the length of the data segment in bytes. The MSB is sent first.
Data	0x000A	Variable	Compare data

29.13.12ERASE_DATA

Table 29-27: ERASE_DATA

ERASE_DATA	0x4401	Erase the Range of Flash Memory	
Description	Erases the number of bytes specified in the Length field from flash memory, starting from the Start Address field.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x4401
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.

29.13.13EXECUTE_CODE

Table 29-28: 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.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x2101
Start Address field	0x0002	4	This is the address to begin parsing for an application header in the target memory.

30. Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION
0	3/2022	Initial Release
1	1/2023	Marked *GCR_SYSCTRL.ovr* as Do Not Modify. This field must be set to 0b10. Updated *Configuring an LPUART for Low-Power Modes of Operation* to show additional steps required. Updated *Universal Asynchronous Receiver/Transmitter (UART)* supported low-power modes. Complete chapter update for *Secure Communication Protocol Bootloader (SCPBL)*. Removed support for ECC throughout. Updated *Controller Area Network (CAN)* chapter to indicate only support for CAN 2.0b, no support for CAN FD frequencies. Updated *ADC* chapter. Updated *GCR_PM* wake-up fields to show correct operating modes for wake. Marked *GPIOn_WKEN*, *GPIOn_WKEN_SET*, and *GPIOn_WKEN_CLR* as reserved. Marked flash controller registers as only reset on POR. Updated *Figure 13-2* to show correct register and field names. Complete update to *SPI-XiP External Memory Interface* chapter. Updated *General-Purpose I/O (GPIO) and Alternate Function Pins* chapter and *HyperBus/Xccela External Memory Interface (HPB)* chapter to indicate pins with HyperBus as alternate function are tied to V_{DDIO} and cannot be changed to V_{DDIOH}. Added register and bit field definitions for complete list of *USBHS Device Registers*. Updated *Operating Modes* to clearly describe entry into each mode. Added Peripheral Clock Enable section to each peripheral chapter.