MAX32672 User Guide

UG7558; Rev 2; 2/2024

Abstract: This user guide provides application developers information on how to use the memory and peripherals of the MAX32672 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.

MAX32672 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 MAX32672 family of devices, please refer to the data sheet.

1.1 Related Documentation

The MAX32672 data sheet and errata are available from the Analog Devices website at *http://www.analog.com/MAX32672*.

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; that is, 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 that are accessible by user software have distinct access capabilities. Each register table contained in this user guide has an access type defined for each field. The definition of each field access type is presented in *Table 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 sets the field to 0. Writing 1 to the field does not affect device operation.

Access Type	Definition
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, the register's name, and the register's 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, the bit's or field's access, the 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

In the DARWIN family, the MAX32672 is an ultra-low-power, cost-effective, highly integrated and highly reliable 32-bit microcontroller enabling designs with complex sensor processing without compromising battery life. It combines a flexible and versatile power management unit with the powerful Arm® Cortex®-M4 processor with floating point unit (FPU). The MAX32672 also offers legacy designs an easy and cost optimal upgrade path from 8- or 16-bit microcontrollers.

The device integrates 1MB of flash and 200KB of SRAM to accommodate application and sensor code. Error Correction Coding (ECC), capable of single error correction and double error detection (SEC-DED), is implemented on the entire flash, RAM and cache to ensure extremely reliable code execution even in the harshest of environments. Additional features such as the two windowed watchdog timers, with fully flexible and independent clocking, have been added to further enhance reliable operation. Brownout detection ensures proper operation during power-down and power-up events and unexpected supply transients. The flash is organized into two equal size physical banks in order to allow execute-while-write and facilitate "live upgrades".

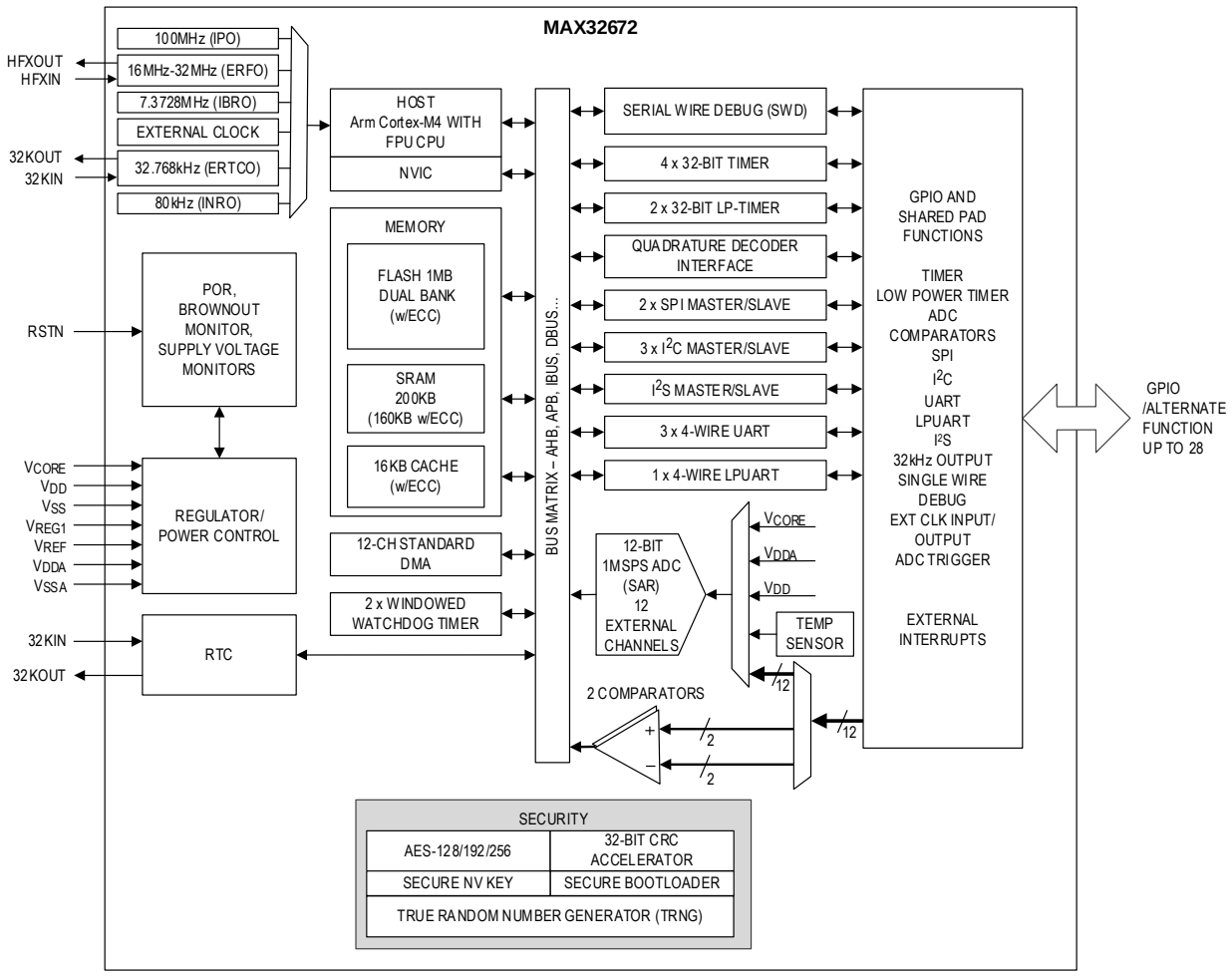
Multiple high-speed peripherals such as 3.4MHz I^2C, 50MHz SPI and UARTs are included to maximize communication bandwidth. In addition, a low power UART is available for operation in the lowest power sleep modes to facilitate wake-up on activity without any loss of data. A total of six timers with I/O capability are provided, including two low power timers to enable pulse counting, capture/compare and PWM generation even in the lowest power sleep modes. An incremental/quadrature decoder interface with multiple diagnostics is included specifically for motor control applications. A 1MSPS 12-ch 12-bit SAR ADC is integrated for digitization of analog sensor signals or other analog measurements. Two low power comparators, available in all low power modes, allow energy efficient monitoring and wakeup on external analog signals.

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

The high-level block diagram for the MAX32672 is shown in *Figure 2-1*.

2.1 Block Diagram

Figure 2-1: MAX32672 Block Diagram



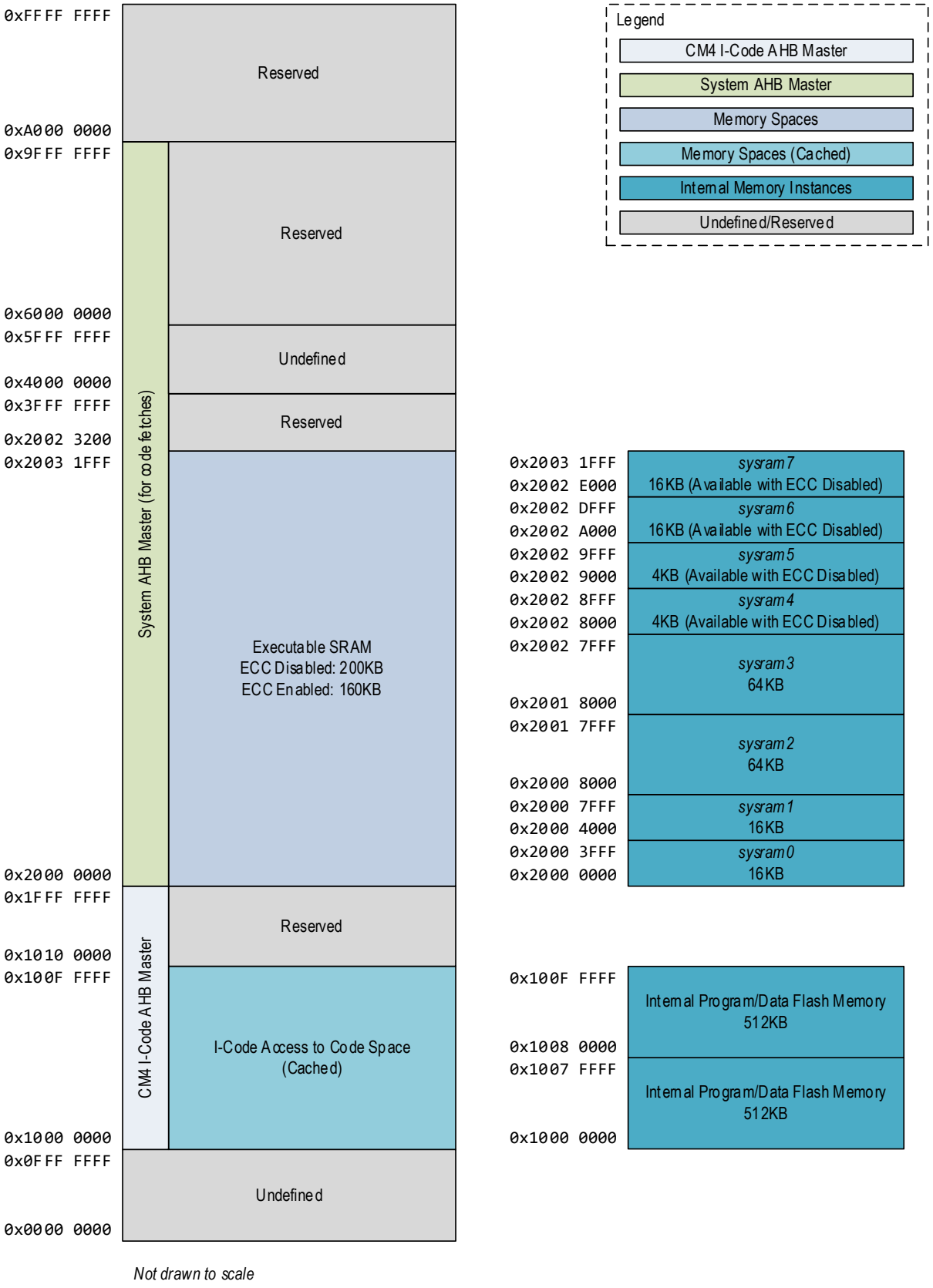
3. Memory, Register Mapping, and Access

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 single byte units but is most typically accessed in 32-bit (4 byte) units. It can 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.

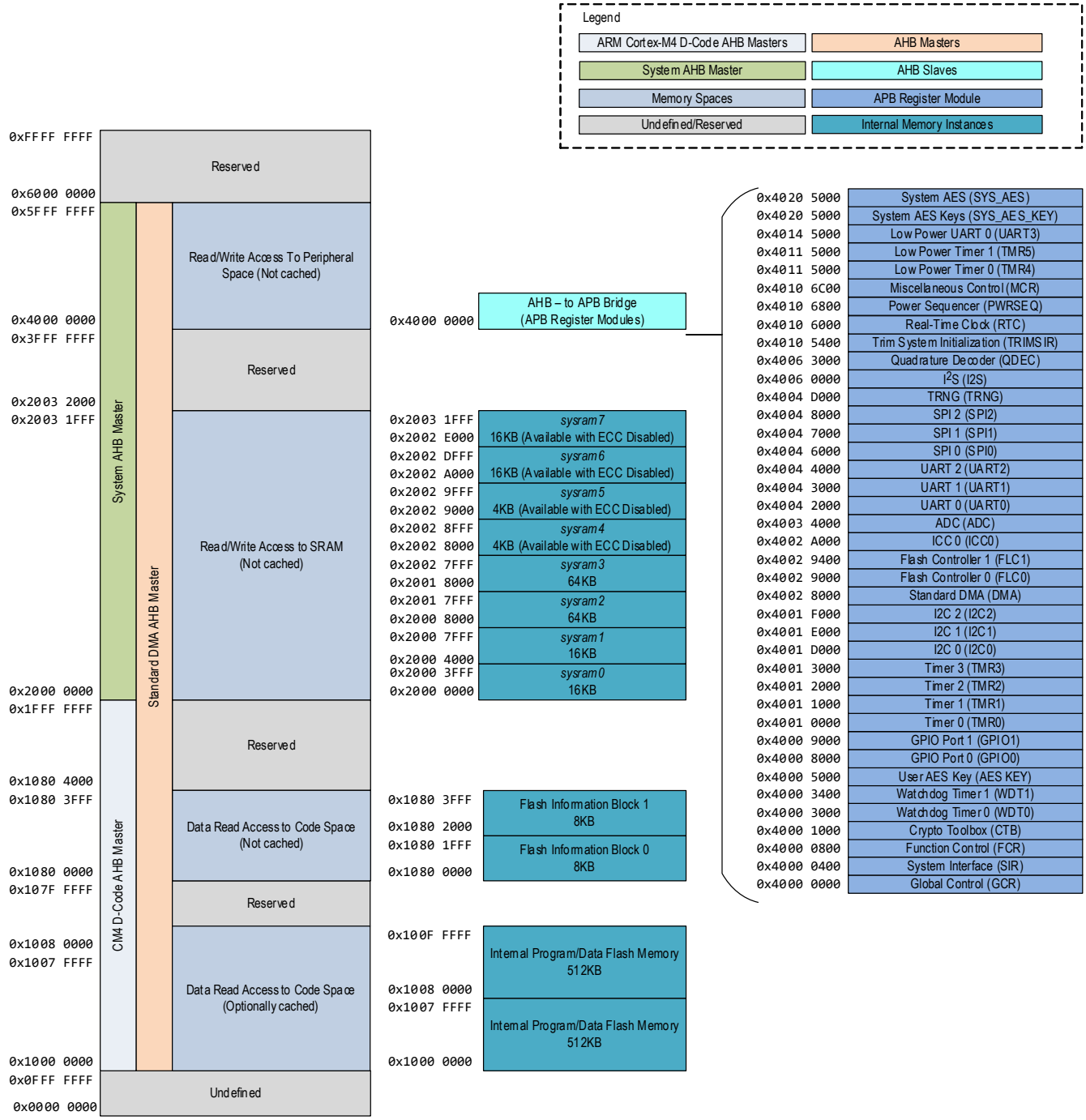
However, it is important to note that the architectural definition does not require the entire 4GB memory range to be populated with addressable memory instances.

Figure 3-1: Code Memory Mapping



Not drawn to scale

Figure 3-2: Data Memory Mapping



3.2 Standard Memory Regions

Several standard memory regions are defined for the Arm Cortex-M4 architecture. The use of many of these is optional for the system integrator. At a minimum, the MAX32672 must contain some code and data memory for the software and variable/stack use and certain components that are part of the instantiated 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). The Cortex-M4 core and Arm debugger use two different standard core bus masters to access this memory area. The I-Code AHB bus master is used for instruction decode fetching from code memory, while the D-Code AHB bus master is used for data fetches from code memory. This is arranged so that data fetches avoid interfering with instruction execution.

The MAX32672 code memory mapping is illustrated in *Figure 3-1*. The code space memory area contains the main internal flash memory, which holds most of the instruction code that is executed on the device. The internal flash memory is mapped into both code and data space from 0x1000 0000 to 0x100F FFFF. It is partitioned as two 512KB blocks of usable flash plus extra flash storage for Error Correction Coding (ECC) check bits if ECC is enabled. This additional storage is not user-accessible, even when ECC is disabled.

This program memory area must also contain the default system vector table and the initial settings for all system exception handlers and interrupt handlers. The reset vector for the device is 0x0000 0000. After execution of ROM code that is not user-accessible, execution is transferred to location 0x1000 0000.

The code space memory on the MAX32672 also contains the mapping for the flash information blocks, from 0x1080 0000 to 0x1080 3FFF. Flash information block 0 is read only and mapped from address 0x1080 0000 to 0x1080 1FFF. Flash information block 0 contains the device's unique serial number (USN) as well as trim settings for the device. Flash information block 1 is mapped from address 0x1080 2000 to 0x1080 3FFF. The first 4,096 bytes of flash information block 1 is write only and is used to store the system AES key. The second 4,096 bytes of flash information block 1 is read/write and is available as non-volatile storage. See the *Flash Information Blocks* section for details of the flash information blocks including the system AES key and the device’s USN.

3.2.2 Internal Cache Memory

The MAX32672 includes a dedicated unified internal cache controller (ICC) with 16,384 bytes of cache memory for the CM4 core (ICC0).

The unified internal cache memory is used to cache data and instructions fetched through the I-Code bus for the CM4 from the internal flash memory. See section *Unified Internal Cache Controller* for detailed instructions on enabling the unified internal cache controllers.

3.2.3 AES Key and Working Space Memory

The device's system AES keys are stored in the information block 1 memory in a write-only page. This page address range is 0x1080 2000 to 0x1080 2FFF. This memory can be erased and written but cannot be read by the device. Write the system AES keys by performing a flash write starting at address 0x1080 2000. The system AES keys, stored in information block 1, are loaded on reset into the AES key APB registers at address 0x4020 7500. The AES key peripheral registers are writable but not readable by user software. See *System AES Key Storage* for details of writing an AES key to the information block. See *System AES (AES)* for details of using the system AES keys.

The remainder of information block 1 is usable as non-volatile memory storage. This 4,096 byte page starts at address 0x1080 3000 and ends at 0x1080 3FFF.

3.2.4 SRAM Space

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

The MAX32672 data memory mapping is illustrated in *Figure 3-2*, and the SRAM configuration is defined in *Table 3-1*. This memory area contains the main system SRAM. The size of the internal SRAM is 200KB when not using ECC. Its address range is mapped from 0x2000 0000 to 0x2003 1FFF. If ECC is enabled, the SRAM size decreases to 160KB. The address range with ECC enabled is mapped from 0x2000 0000 to 0x2002 7FFF.

The entirety of the SRAM memory space on the MAX32672 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 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 double word (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 160KB 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). Thus, it is only applicable to accesses generated by the core. Reads/writes to the bit-banding alias area by other (non-Arm-core) bus masters do not trigger a bit-banding operation and instead result in an AHB bus error.

The SRAM area on the MAX32672 can be used to contain executable code. Code stored in the SRAM is accessed directly for execution (using the system bus) and is not cached. The SRAM is also where the CM4 stack must be located, as it is the only general-purpose SRAM memory on the device. A valid stack location inside the SRAM must be set by the system exception table (which is, by default, stored at the beginning of the internal flash memory).

The MAX32672 specific AHB bus masters can access the SRAM to use as general storage or working space.

Table 3-1: SRAM Configuration

System RAM Block #	Size (Words)	Start Address	End Address	ECC SRAM Complement
sysram0	4K	0x2000 0000	0x2000 3FFF	*sysram4*
sysram1	4K	0x2000 4000	0x2000 7FFF	*sysram5*
sysram2	16K	0x2000 8000	0x2001 7FFF	*sysram6*
sysram3	16K	0x2001 8000	0x2002 7FFF	*sysram7*
sysram4	1K	0x2002 8000	0x2002 9FFF	-
sysram5	1K	0x2002 9000	0x2002 9FFF	-
sysram6	4K	0x2002 A000	0x2002 DFFF	-
sysram7	4K	0x2002 E000	0x2003 1FFF	-

3.2.5 Peripheral Space

The peripheral space area of memory is intended to map control registers, internal buffers/working space, and other features needed for the software control of non-core peripherals. It is defined from byte address range 0x4000 0000 to 0x5FFF FFFF (0.5GB maximum). On the MAX32672, all device-specific module registers are mapped to this memory area and 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. Byte-aligned read/write operations in the peripheral bit-banding alias area (32MB in length, from 0x4200 0000 to 0x43FF FFFF) are translated by the core into read/mask/shift or read/modify/write operation sequences to the appropriate byte location in the bit-banding area.

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

On the MAX32672, access to the region containing most peripheral registers (0x4000 0000 to 0x400F FFFF) goes from the AHB bus through an AHB-to-APB bridge enabling peripheral modules to operate on the lower power APB bus matrix. The AHB-to-APB bridge also ensures that peripherals with slower response times do not tie up bandwidth on the AHB bus, which must have a faster response time since it handles main application instruction and data fetching.

3.2.6 System Area (Private Peripheral Bus)

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

In addition to being restricted to the core, the software can only access this area when running in the privileged execution mode (instead of 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 access this area.

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

3.2.7 System Area (Vendor Defined)

The system area (vendor defined) memory space is reserved for vendor (system integrator) specific functions that are not handled by another memory area. It is defined from byte address range 0xE010 0000 to 0xFFFF FFFF. The MAX32672 does not implement this memory region.

3.3 AHB Interfaces

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

3.3.1 Core AHB Interfaces

3.3.1.1 I-Code

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

3.3.1.2 D-Code

The Arm core uses this AHB master for data fetches from memory instances located in code space from byte addresses 0x0000 0000 to 0x1FFF FFFF. This bus master has access to the internal flash memory and the information block.

3.3.1.3 System

The Arm core uses this AHB master 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 peripherals and memory areas are also accessed using this bus master.

3.3.2 AHB Masters

3.3.2.1 Standard DMA

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

3.4 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 registers 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
Watchdog Timer 1	WDT1_	0x4000 3400	0x4000 37FF
User AES Keys	USR_AESKEYS_	0x4000 5000	0x4000 53FF
GPIO Port 0	GPIO0_	0x4000 8000	0x4000 8FFF
GPIO Port 1	GPIO1_	0x4000 9000	0x4000 9FFF
Timer 0	TMR0_	0x4001 0000	0x4001 0FFF
Timer 1	TMR1_	0x4001 1000	0x4001 1FFF
Timer 2	TMR2_	0x4001 2000	0x4001 2FFF
Timer 3	TMR3_	0x4001 3000	0x4001 3FFF
I^2C 0	I2C0_	0x4001 D000	0x4001 DFFF
I^2C 1	I2C1_	0x4001 E000	0x4001 EFFF
I^2C 2	I2C2	0x4001 F000	0x4001 FFFF
Standard DMA	DMA	0x4002 8000	0x4002 8FFF
Flash Controller 0	FLC0_	0x4002 9000	0x4002 93FF
Flash Controller 1	FLC1_	0x4002 9400	0x4002 97FF
Internal-Cache Controller	ICC_	0x4002 A000	0x4002 A3FF
ADC	ADC_	0x4003 4000	0x4003 4FFF
UART 0	UART0_	0x4004 2000	0x4004 2FFF
UART 1	UART1_	0x4004 3000	0x4004 3FFF
UART 2	UART2_	0x4004 4000	0x4004 4FFF
SPI0	SPI0_	0x4004 6000	0x4004 6FFF
SPI1	SPI1_	0x4004 7000	0x4004 7FFF
SPI2	SPI2_	0x4004 8000	0x4004 8FFF
TRNG	TRNG_	0x4004 D000	0x4004 DFFF
I^2S	I2S_	0x4006 0000	0x4006 0FFF

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
Quadrature Decoder	QDEC_	0x4006 3000	0x4006 3FFF
Trim System Initialization	TRIMSIR_	0x4010 5400	0x4010 5400
Real-Time Clock	RTC_	0x4010 6000	0x4010 63FF
Power Sequencer	PWRSEQ_	0x4010 6800	0x4010 6BFF
Miscellaneous Control	MCR_	0x4010 6C00	0x4010 6FFF
Timer 4 (Low Power Timer 0)	TMR4_	0x4011 4000	0x4011 4FFF
Timer 5 (Low Power Timer 1)	TMR5_	0x4011 5000	0x4011 5FFF
UART3 (Low Power UART 0)	UART3_	0x4014 5000	0x4014 5FFF
System AES Keys	SYS_AESKEYS_	0x4020 5000	0x4020 5FFF
System AES	AES_	0x4020 7400	0x4020 7FFF

3.5 Error Correction Coding (ECC) Module

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

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

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

3.5.1 SRAM

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

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

Table 3-3: RAM Instances Used for Check RAM for Each System RAM when ECC is Enabled

System RAM Instance	Secondary Check RAM Instance
sysram0	*sysram4*
sysram1	*sysram5*
sysram2	*sysram6*
sysram3	*sysram7*

3.5.2 FLASH

The flash implements the SEC-DED ECC by including an additional 9 check bits for every 128 data bits. These additional bits do not appear in the device's memory map making the bits inaccessible to the software. Reads from and writes to the flash memory behave the same whether ECC is enabled or not. However, it is recommended to write the flash in 128-bit blocks

when ECC is enabled. With ECC enabled, writing 32 bits to the flash sets the check bits for the full 128-bit word. Since the check bits are also stored in the flash, another 32-bit write into the same 128-bit word fails because the device cannot update the check bits.

3.5.3 Cache

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

3.5.4 Limitations

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

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 (IPO) frequency is scaled based on the specific core operating voltage range selected.

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

- 100MHz Internal Primary Oscillator (IPO)
- 7.3728MHz Internal Baud Rate Oscillator (IBRO)
- 80kHz Internal Nanoring Oscillator (INRO)
- 32.768kHz External RTC Crystal Oscillator (ERTCO)
  - Clock Source for the Real-Time Clock (RTC)
- 16MHz to 32MHz External RF Crystal Oscillator (ERFO)
- EXT_CLK1 P0.28 AF2

4.1 Core Operating Voltage Range Selection

The MAX32672 supports three selections for the core operating voltage range (OVR). In a single-supply operation, changing the OVR sets the output of the internal LDO regulator to the voltage shown in *Table 4-1*. In a dual-supply design, setting the OVR allows an external PMIC to provide the required V_{CORE} voltage dynamically. Changing the OVR also reduces the output frequency of the IPO, further reducing power consumption.

PWRSEQ_LPCN.ovr and *FLCn_CTRL.lve* do not affect the frequency of any of the oscillators other than IPO. The setting of these bit fields must correlate to any of the clock sources used as SYS_OSC, as shown in *Table 4-1*.

Changes to the OVR affect the internal flash memory access time, and the software must set the flash wait states for each OVR setting as outlined in section *Flash Wait States*. Changing the core operating voltage reduces the output frequency of the IPO immediately, as shown in *Table 4-1*. Operating the device using dual external supplies requires special considerations and must be handled carefully in software.

Table 4-1: Operating Voltage Range Selection and the Effect on V_{CORE} and SYS_OSC

PWRSEQ_LPCN.ovr	FLCn_CTRL.lve	V_{CORE} Typical (V)	SYS_OSC					
			f_{IPO} (MHz)	f_{IBRO} (MHz)	f_{ERFO} (MHz)	f_{EXT_CLK1} (MHz)	f_{INRO} (kHz)	f_{ERTCO} (kHz)
0	1	0.9	12	7.3728	32 (Max)	50 (Max)	80	32.768
1	1	1.0	50	7.3728	32 (Max)	50 (Max)	80	32.768
2	0	1.1	100	7.3728	32 (Max)	50 (Max)	80	32.768

4.1.1 Setting the Operating Voltage Range

The OVR selection is controlled using the power sequencer low-power control register *PWRSEQ_LPCN.ovr* which is only reset by a POR. These bits should be checked after every reset to determine the correct clock speed and flash wait states. Adjusting the OVR setting affects the frequency of the IPO. Before adjusting the OVR settings, it is required to set the system clock to either the INRO, IBRO, or ERTCO. The device coordinates the OVR change between the internal LDO and the IPO set frequency. When changing the OVR setting, the device must be operating from the internal LDO. In a system using an external supply for V_{CORE}, software must transition to the internal LDO before changing the OVR setting.

The following steps describe how to change the OVR for devices that use the IPO as the default SYS_OSC:

1. Set *PWRSEQ_LPCN.ldo_dis* to 0 to ensure the device is operating from the internal LDO for V_{CORE}.
   a. If using an external supply for V_{CORE}, ensure the external supply is set to the same voltage as the current OVR setting. The external supply must be equal to or greater than the set OVR voltage.
2. Set either the ERTCO or INRO as the system clock source.
   a. See the *Oscillator Sources and Clock Switching* section for details on system clock selection.
3. Set *GCR_MEMCTRL.fws* = 5 to ensure flash operation at any frequency.
4. Set *PWRSEQ_LPCN.ovr* to either 0, 1, or 2, as shown in *Table 4-2*.
5. Set *FLCn_CTRL.lve* to either 0 or 1 according to the OVR setting set in step 4.
6. If desired, set the system clock source to the IPO and update the system clock prescaler to the desired value.
   a. Set *GCR_CLKCTRL.sysclk_sel* = 0.
   1. b. Wait for the system clock ready bit, *GCR_CLKCTRL.sysclk_rdy,* to read 1.
   c. Set *GCR_CLKCTRL.sysclk_div* to the desired prescaler value.
7. Set *GCR_MEMCTRL.fws* to the minimum value shown for the selected OVR and system clock.
8. Set *GCR_RST0.periph* = 1 to perform a peripheral reset.

On each subsequent non-POR reset event:

1. Immediately after the reset event, set the flash low voltage enable bit to 1 (*FLCn_CTRL.lve*) to match the setting of the *PWRSEQ_LPCN.ovr* field since the *PWRSEQ_LPCN.ovr* field is not reset.

   Note: Set the FLCn_CTRL.lve to 1 in the reset vector code in RAM to ensure the low-voltage enable is set before accessing any code in the flash memory.
2. Set the clock prescaler, *GCR_CLKCTRL.sysclk_div*, as needed by the system.
3. Set the number of flash wait states, *GCR_MEMCTRL.fws*, as needed based on the OVR settings using *Table 4-2*.

4.1.2 Flash Wait States

The setting for the number of flash wait states affects performance, and it is critical to set it correctly based on the *PWRSEQ_LPCN.ovr* settings and the SYS_CLK frequency. Set the number of flash wait states using the field *GCR_MEMCTRL.fws* per *Table 4-2*. The *GCR_MEMCTRL.fws* field should always be set to the default POR reset value of 5 before changing the *PWRSEQ_LPCN.ovr* settings. POR, system reset, and watchdog reset all reset the flash wait state field, *GCR_MEMCTRL.fws*, to the POR default setting of 5. When changing the system clock prescaler, *GCR_CLKCTRL.sysclk_div*, moving from a slower system clock frequency to a faster system clock frequency, always set *GCR_MEMCTRL.fws* to the minimum required for the faster system clock frequency before changing the system oscillator prescaler *GCR_CLKCTRL.sysclk_div*. After a system reset or watchdog reset, the *PWRSEQ_LPCN.ovr* setting overrides the default setting of the IPO frequency to prevent system lockup. The *FLCn_CTRL.lve* setting must be restored by software after any reset.

Important: *Flash reads can fail and result in unknown instruction execution if the GCR_MEMCTRL.fws setting is lower than the minimum required for a given PWRSEQ_LPCN.ovr setting and the selected system clock frequency.*

Table 4-2: Minimum Flash Wait State Setting for Each OVR Setting (f_{SYSCLK} = f_{IPO}, GCR_CLKCTRL.ipo_div = 1)

Core Operating Voltage Range Setting		Core Voltage Range V_{CORE} (V)	f_{IPO} (MHz)	System Clock Prescaler GCR_CLKCTRL.sysclk_div	System Clock f_{SYS_CLK} (MHz)	Minimum Flash Wait State Setting GCR_MEMCTRL.fws
PWRSEQ_LPCN.ovr	FLCn_CTRL.lve					
0	1	0.9	12	0	12	0
				1	6	0
1	1	1.0	50	0	50	1
				1	25	0
2	0	1.1	100	0	100	2
				1	50	1
				2	25	0

Table 4-3: Minimum Flash Wait State Setting for Each OVR Setting (f_{SYSCLK} = f_{IBRO})

Core Operating Voltage Range Setting		Core Voltage Range V_{CORE} (V)	f_{IBRO} (MHz)	System Clock Prescaler GCR_CLKCTRL.sysclk_div	System Clock f_{SYS_CLK} (MHz)	Minimum Flash Wait State Setting GCR_MEMCTRL.fws
PWRSEQ_LPCN.ovr	FLCn_CTRL.lve					
0	1	0.9	7.3728	0	7.3728	0
				1	3.6864	0
1	1	1.0	7.3728	0	7.3728	0
				1	3.6864	0
2	0	1.1	7.3728	0	7.3728	0
				1	3.6864	0
				2	1.8432	0

Table 4-4: Minimum Flash Wait State Setting for Each OVR Setting (f_{SYSCLK} = f_{ERFO})

Core Operating Voltage Range Setting		Core Voltage Range V_{CORE} (V)	f_{ERFO} (MHz)	System Clock Prescaler GCR_CLKCTRL.sysclk_div	System Clock f_{SYS_CLK} (MHz)	Minimum Flash Wait State Setting GCR_MEMCTRL.fws
PWRSEQ_LPCN.ovr	FLCn_CTRL.lve					
0	1	0.9	16–20	0	16-20	0
				1	8-10	0
1	1	1.0	20–25	0	20-25	0
				1	10-12.5	0
2	0	1.1	25–32	0	25-32	0
				1	12.5-16	0
				2	8.33-10.66	0

Table 4-5: Minimum Flash Wait State Setting for Each OVR Setting ($f_{SYSCLK} = f_{EXT\_CLK1}$)

Core Operating Voltage Range Setting		Core Voltage Range V_{CORE} (V)	f_{EXT_CLK1} (MHz)	System Clock Prescaler GCR_CLKCTRL.sysclk_div	System Clock f_{SYS_CLK} (MHz)	Minimum Flash Wait State Setting GCR_MEMCTRL.fws
PWRSEQ_LPCN.ovr	FLCn_CTRL.lve					
0	1	0.9	1–15	0	1-15	0
				1	0.5-7.5	0
1	1	1.0	16–30	0	16-30	0
				1	8-15	0
2	0	1.1	31–45	0	31-45	0
				1	15.5-22.5	0
				2	10.33-15	0
2	0	1.1	46–50	0	46-50	1
				1	23-25	0
				2	15.33-16.66	0

Table 4-6: Minimum Flash Wait State Setting for Each OVR Setting ($f_{SYSCLK} = f_{INRO}$)

Core Operating Voltage Range Setting		Core Voltage Range V_{CORE} (V)	f_{INRO} (kHz)	System Clock Prescaler GCR_CLKCTRL.sysclk_div	System Clock f_{SYS_CLK} (kHz)	Minimum Flash Wait State Setting GCR_MEMCTRL.fws
PWRSEQ_LPCN.ovr	FLCn_CTRL.lve					
0	1	0.9	80	0	80	0
				1	40	0
1	1	1.0	80	0	80	0
				1	40	0
2	0	1.1	80	0	80	0
				1	40	0
				2	20	0

Table 4-7: Minimum Flash Wait State Setting for Each OVR Setting ($f_{SYSCLK} = f_{ERTCO}$)

Core Operating Voltage Range Setting		Core Voltage Range V_{CORE} (V)	f_{ERTCO} (kHz)	System Clock Prescaler GCR_CLKCTRL.sysclk_div	System Clock f_{SYS_CLK} (kHz)	Minimum Flash Wait State Setting GCR_MEMCTRL.fws
PWRSEQ_LPCN.ovr	FLCn_CTRL.lve					
0	1	0.9	32.768	0	32.768	0
				1	16.384	0
1	1	1.0	32.768	0	32.768	0
				1	16.384	0
2	0	1.1	32.768	0	32.768	0
				1	16.384	0
				2	8.192	0

4.2 Oscillator Sources and Clock Switching

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

Equation 4-1: System Clock Scaling (SYS_CLK)

$$SYS\_CLK = \frac{SYS\_OSC}{2^{GCR\_CLKCTRL.sysclk\_sel}}$$

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

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

Equation 4-2: AHB Clock (HCLK)

$$HCLK = SYS\_CLK$$

Equation 4-3: APB Clock (PCLK)

$$PCLK = {}^{SYS\_CLK}/_{2}$$

Equation 4-4: AoD Clock (AOD_CLK)

$$AOD\_CLK = {}^{PCLK}/_{4\ X\ 2^{GCR\_PCLKDIV.aon\_clkdiv}}$$

GCR_PCLKDIV.*aon_clkdiv* is selectable from 0 to 3 for divisors of 4, 8, 16, and 32.

CAUTION: When switching the SYS_OSC or modifying the SYS_OSC prescaler (GCR_CLKCTRL.sysclk_div), any device peripherals using SYS_CLK, APB clock, or AHB clock become unstable until the switchover is complete. The software should disable all active peripherals before switching SYS_OSC or modifying the system clock prescaler.

The RTC uses the ERTCO for its clock source. All oscillators are reset to their POR reset default state during a POR, system reset, or watchdog reset. Oscillator settings are not reset during a soft reset or peripheral reset. *Table 4-8* shows each oscillator's enabled state for each type of reset source in the MAX32672. *Table 4-9* details each reset source's effect on the system clock selection and the system clock prescaler settings.

Table 4-8: Reset Sources and Effect on Oscillator Status

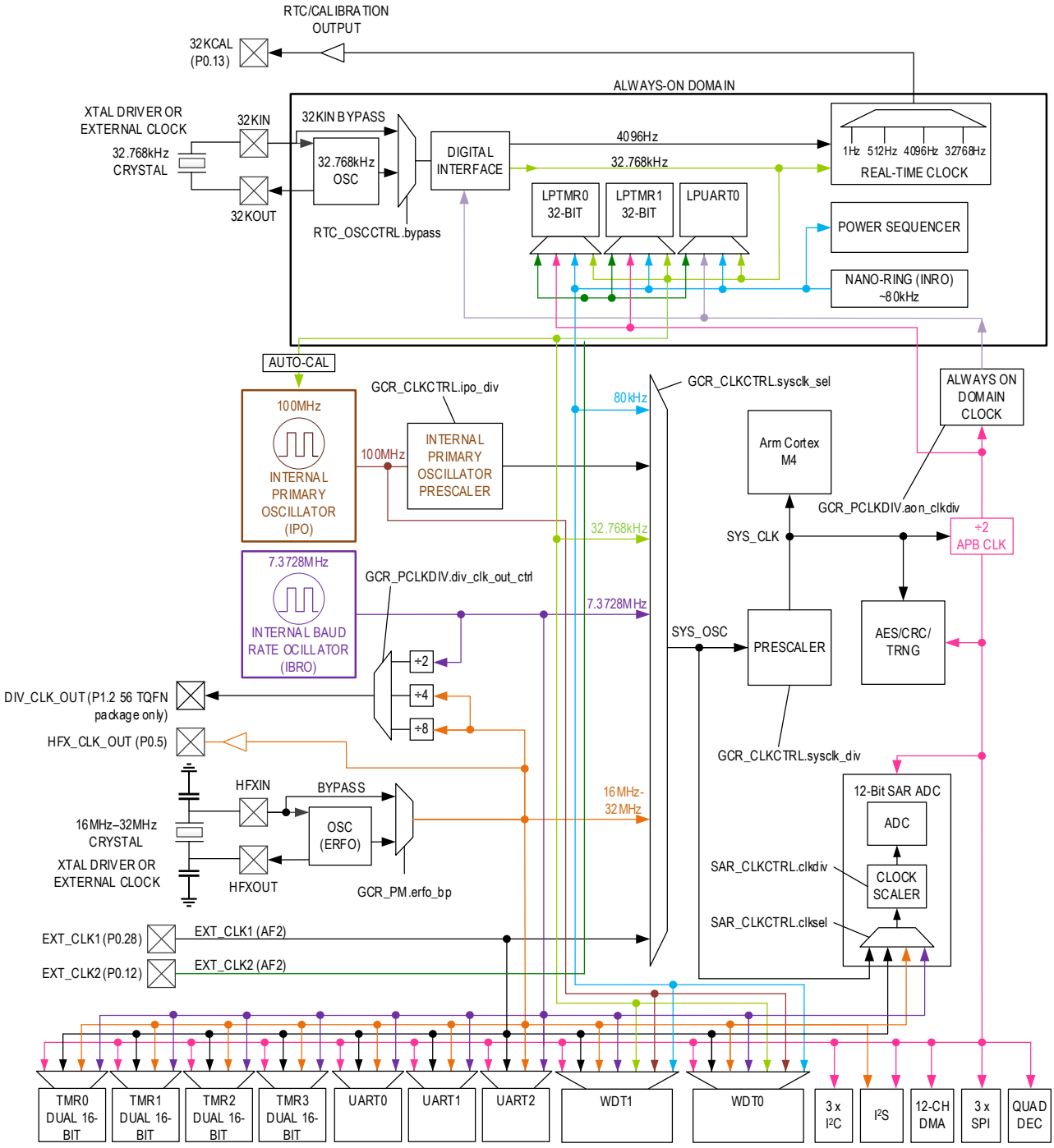
	Reset Source				
Oscillator	**POR**	**System**	**Watchdog**	**Soft**	**Peripheral**
IPO	Enabled	Enabled	Enabled	Retains State	Retains State
IBRO	Disabled	Disabled	Disabled	Retains State	Retains State
INRO	Enabled	Enabled	Enabled	Enabled	Enabled
ERTCO	Disabled	Retains State	Retains State	Retains State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

Table 4-9: 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*	IPO	IPO	IPO	Retains State	Retains State
System Clock Prescaler GCR_CLKCTRL.*sysclk_div*	1	1	1	Retains State	Retains State

Figure 4-1 shows a high-level diagram of the MAX32672 clock tree.

Figure 4-1: MAX32672 Clock Block Diagram



4.2.1 Oscillator Implementation

Following a POR or a system reset, the SYS_OSC defaults to the IPO, and the INRO is also enabled. Before using any oscillator, the desired oscillator must first be enabled by setting the oscillator's enable bit in the *GCR_CLKCTRL* register. Once an oscillator's enable bit is set, the oscillator's ready bit must read 1 before attempting to use the oscillator as a system oscillator source. The oscillator ready status flags are contained in the *GCR_CLKCTRL* register.

Once the corresponding oscillator ready bit is set, the oscillator can then be selected as SYS_OSC by configuring the clock source select field (*GCR_CLKCTRL.sysclk_sel*).

Anytime software changes SYS_OSC by changing *GCR_CLKCTRL.sysclk_sel*, the clock ready bit *GCR_CLKCTRL.sysclk_rdy* is automatically cleared to indicate that an SYS_OSC switchover is in progress. When the switchover is complete, *GCR_CLKCTRL.sysclk_rdy* is set to 1 by hardware indicating the oscillator selected is ready for use. Immediately before entering any low-power mode, the application must enable any oscillator needed during the low-power mode.

CAUTION: When switching the SYS_OSC or modifying the SYS_OSC prescaler (GCR_CLKCTRL.sysclk_div), any device peripherals using SYS_CLK, APB clock, or AHB clock become unstable until the switchover is complete. The software should disable all active peripherals before switching SYS_OSC or modifying the system clock prescaler.

4.2.2 100MHz Internal Primary Oscillator (IPO)

This oscillator can be selected as SYS_OSC. The IPO is the fastest oscillator and draws the most power.

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.

4.2.2.1 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.div* field to 3,051. See the *FCR_AUTOCAL2.div* field for additional information.
4. Set the *FCR_AUTOCAL2.runtime* field to 10.
5. Set the *FCR_AUTOCAL1.initial* field to 0x100.
6. Set the *FCR_AUTOCAL0.gain* field to 4.
7. Set the *FCR_AUTOCAL0.sel*, *FCR_AUTOCAL0.en*, and *FCR_AUTOCAL0.load* fields to 1 by performing a bitwise OR of the *FCR_AUTOCAL0* register with 0x7.
8. Wait 10ms for the trim to complete.
    a. The calculated trim is loaded to the *FCR_AUTOCAL0.gain* field and is used by the hardware as long as the *FCR_AUTOCAL0.sel* field is set to 1.
9. Set the *FCR_AUTOCAL0.en* field to 0 to stop the calibration.

4.2.3 16MHz to 32MHz External RF Oscillator (ERFO)

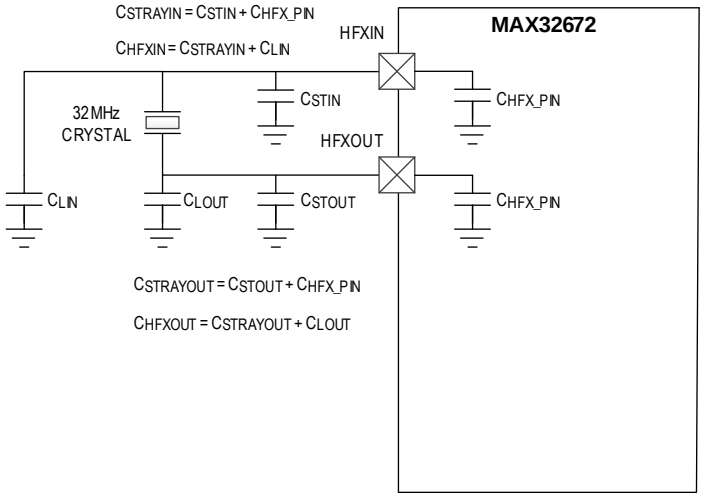
This oscillator can be selected as SYS_OSC. It is important to use the correct capacitor values on the PCB when connecting the crystal. *Figure 4-2* depicts the method to determine the capacitor values CLIN and CLOUT.

This oscillator is disabled by default at power-up.

The following steps must be followed to use this oscillator as SYS_OSC:

1. Enable the ERFO by setting *GCR_CLKCTRL.erfo_en*.
2. Wait until *GCR_CLKCTRL.erfo_rdy* is set. The ERFO is now operating.
3. Set *GCR_CLKCTRL.sysclk_sel* to 2, selecting the ERFO as the SYS_OSC.
4. Wait until *GCR_CLKCTRL.sysclk_rdy* is set. The ERFO is now operating as the SYS_OSC.

Figure 4-2: 32MHz ERFO Crystal Capacitor Determination



Equation 4-5: Determining Load Capacitance for ERFO

$$C_L = C_{HFXIN}\ X\ C_{HFXOUT}/(C_{HFXIN} + C_{HFXOUT}).$$

Calculate the values of C_{LOUT} and C_{LIN}, referring to *Figure 4-2*, using the following steps:

1. The crystal load, C_L, as specified in the device data sheet electrical characteristics table is required to be 12pF. Therefore, the total capacitance seen by the crystal must equal C_L.
2. $C_{LOUT} = C_{LIN}$
3. Assume the device pin capacitance of the HFXOUT and HFXIN pins, respectively, is = 4pF each. This specification is outlined in the electrical characteristics table of the device data sheet as C_{HFX_PIN}.
4. Assume the circuit board stray capacitance represented in the diagram by C_{STIN} and C_{STOUT} = 0.5pf each.
5. Solve for C_{LOUT}:
   $C_{LOUT} = 19.5pF = C_{LIN}$
6. Choose 20pF as the closest standard value for $C_{LOUT} = C_{LIN}$.

4.2.4 7.3728MHz Internal Baud Rate Oscillator (IBRO)

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

Software selection of the voltage that controls this oscillator is controlled by the register bit *GCR_CLKCTRL.ibro_vs*. The internal CPU 1V LDO core supply voltage is the default option. The external pin V_{CORE} can also be selected.

4.2.5 32.768kHz External Real-Time Clock Oscillator (ERTCO)

The ERTCO is a very low-power external oscillator that can be selected as SYS_OSC. This oscillator can optionally use a 32.768kHz input clock instead of an external crystal. The ERTCO is available as an output on GPIO as an alternate function (32KCAL (P0.13)).

This oscillator 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.

A POR disables the ERTCO. All other forms of reset do not change the ERTCO enable bit. See *Figure 4-3* and *Figure 4-4* for details on reset sources and the effect on the ERTCO.

4.2.5.1 Enabling the ERTCO

Perform the following steps to enable the ERTCO:

1. Power on the ERTCO by setting *MCR_CLKCTRL.ertco_pd* to 0.
2. Enable the ERTCO by setting the *MCR_CLKCTRL.ertco_en* field to 1.
3. Wait until *GCR_CLKCTRL.erfo_rdy* field reads 1.
   a. The ERFO is now operating.
4. If setting the ERTCO as the system oscillator, set *GCR_CLKCTRL.sysclk_sel* to 6 to select the ERTCO as the SYS_OSC.
   a. Wait until *GCR_CLKCTRL.sysclk_rdy* reads 1.
   b. The ERTCO is now operating as the SYS_OSC.

4.2.6 80kHz Ultra-Low-Power Internal Nanoring Oscillator (INRO)

This is a low-power internal oscillator that can be selected as SYS_OSC. This oscillator 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%.

4.3 Operating Modes

The device provides five operating modes, four of which are defined as low-power modes:

- *ACTIVE*
- *Low-Power Modes:*
  - *SLEEP*
  - *DEEPSLEEP*
  - *BACKUP*
  - *STORAGE*

A wake-up event can wake the device to *ACTIVE* from a low-power mode as shown in *Table 4-10*.

Table 4-10: Wakeup Sources

Low Power Operating Mode	Wake-Up Source
SLEEP	Interrupts (GPIO or any active peripheral), RSTN assertion
DEEPSLEEP	Interrupts (RTC and GPIO), RSTN assertion, LPUART, and LPTMR0/1
BACKUP	Interrupts (RTC and GPIO), RSTN assertion, LPUART, and LPTMR0/1
STORAGE	Interrupts (RTC and GPIO), RSTN assertion

4.3.1 ACTIVE

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

Dynamic clocking allows the software to selectively enable or disable clocks and power to individual peripherals, providing the optimal mix of high-performance and power conservation.

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 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 is sleeping.
- Standard DMA is available for use.
- All enabled peripherals are on unless explicitly disabled before entering *SLEEP*.

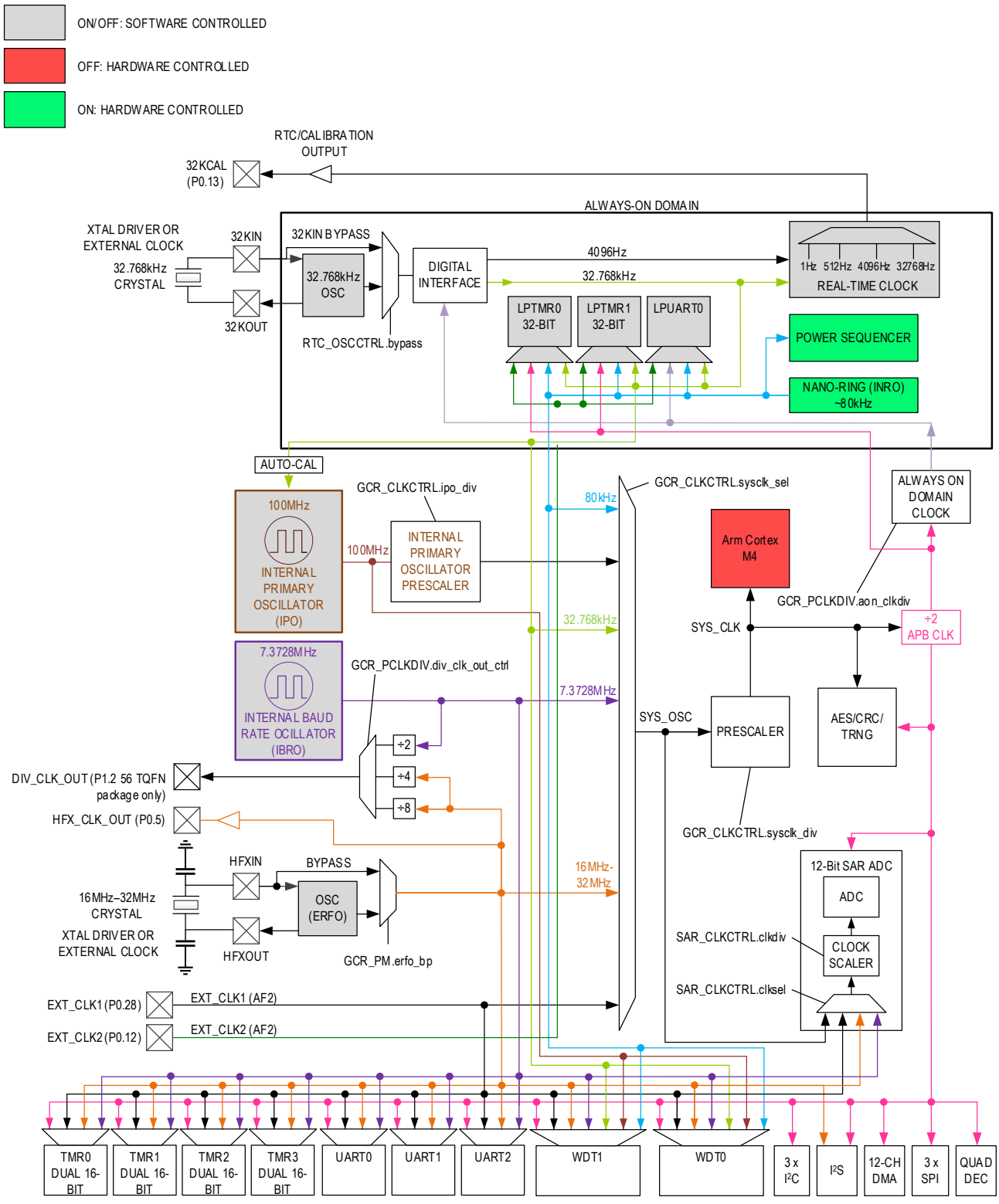
4.3.2.1 Entering SLEEP

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

1. Configure any desired wake-up functions. See *Table 4-10* for possible wake-up sources.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKST0*
   b. *PWRSEQ_LPWKST1*
3. Clear the low-power peripheral and *BACKUP* wake-up status flags by writing 0x0001 001F to the *PWRSEQ_LPPWKST* register.
4. Set the *PWRSEQ_LPCN*.*vcore_det_bypass* bit to 1.
5. Set SCR.sleepdeep to 0.
6. Perform a wait for interrupt (WFI) or wait for event (WFE) instruction.

Table 4-15 and *Table 4-17* show the effects that *SLEEP* has on the various clock sources. *Figure 4-3* shows the clocks available and blocks disabled during *SLEEP*.

Figure 4-3: MAX32672 SLEEP Clock Control



4.3.3 DEEPSLEEP

This mode places the CPU in a static, low-power state. All internal clocks, except the INRO, are gated off. SYS_OSC is gated off, so the two primary bus clocks PCLK and HCLK, are inactive. The CPU state is retained. The ERTCO can be enabled using the software.

The low-power peripherals LPUART0, LPTMR0, and LPTMR1 can be enabled to operate in this mode. The clock source for these peripherals is selectable, but because the primary bus clocks PCLK and HCLK are gated off, the clock source choice is limited. See *Table 9-1* and *Table 13-1* for the available clock sources. These three low-power peripherals are disabled/enabled before entering *DEEPSLEEP* mode by setting/clearing the associated bit in the *MCR_PCLKDIS* register, as shown in *Table 4-11*.

Table 4-11: DEEPSLEEP Low-Power Peripheral Control Truth Table

Register Settings			Configuration
MCR_PCLKDIS.lpuart0	***MCR_PCLKDIS.lptmr1***	***MCR_PCLKDIS.lptmr0***	
0	0	0	LPUART0, LPTMR1, and LPTMR0 enabled
x	x	1	LPUART0, LPTMR1, and LPTMR0 disabled
x	1	x	LPUART0, LPTMR1, and LPTMR0 disabled
1	x	x	LPUART0, LPTMR1, and LPTMR0 disabled
Note: The setting of any of the MCR_PCLKDIS register bits before entering DEEPSLEEP causes all three of these low-power peripherals to be disabled. *Note: If LPUART0, LPTMR0, and LPTMR1 are enabled, all the outputs associated with these three peripherals are no longer be available as GPIO. The GPIO pins affected are P0.6, P0.7, P0.22, P0.23, P0.24, P0.25, P0.26, and P0.27.*			

The RTC, which has an independent oscillator, can return the device to *ACTIVE*. The ERTCO remains enabled in *DEEPSLEEP* if it was enabled before entering *DEEPSLEEP*. The watchdog timers are inactive in this mode.

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

Table 4-15 and *Table 4-17* show the effects that *DEEPSLEEP* has on the various clock sources.

Figure 4-4 shows the clock control during *DEEPSLEEP*.

4.3.3.1 Entering DEEPSLEEP

Place the device in *DEEPSLEEP* by performing the following steps:

1. Configure any desired wake-up functions. See *Table 4-10* for possible wake-up sources.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKST0*
   b. *PWRSEQ_LPWKST1*
3. Clear the low-power peripheral wake-up flags and the backup wake-up status flag by writing 0x0001 001F to the *PWRSEQ_LPPWKST* register.
4. Set the *PWRSEQ_LPCN.vcore_det_bypass* bit to 1.
5. Set SCR.sleepdeep bit to 1.
6. Perform a WFI or WFE instruction.

4.3.4 BACKUP

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

- The CM4 is powered off.
- *sysram0* through *sysram3* can be independently configured to be state retained, as shown in *Table 4-12*.
- The low-power peripherals (LPUART0, LPUART1, and LPTMR0) can be configured for operation and used as a wakeup source.
- The RTC can be configured to remain on and used as a wakeup source.
- All other peripherals are powered off.
- All power sequencer registers retain state, including the *PWRSEQ_GPR0* and *PWRSEQ_GPR1* registers.
- The following oscillators are powered down:
  - IPO
  - ISO
  - IBRO
  - ERFO
- INRO is on.
- ERTCO is software controlled.

Table 4-12: SRAM Retention By Address Range in BACKUP, System Reset, Watchdog Reset, and External Reset

RAM Block #	State Retention Control	Address Range Retention	Size
sysram0 + sysram4	*PWRSEQ_LPCN.ram0ret_en*	0x2000 0700 - 0x2000 3FFF, 0x2002 8000 - 0x2002 8FFF	18KB
sysram1 + sysram5	*PWRSEQ_LPCN.ram1ret_en*	0x2000 5300 - 0x2000 7D00, 0x2002 9000 - 0x2002 9FFF	14KB
sysram2 + sysram6	*PWRSEQ_LPCN.ram2ret_en*	0x2000 8000 - 0x2001 7FFF, 0x2002 A000 - 0x2002 DFFF	80KB
sysram3 + sysram7	*PWRSEQ_LPCN.ram3ret_en*	0x2001 8000 - 0x2002 7FFF, 0x2002 E000 - 0x2003 1FFF	80KB

Note: On parts with SCPBL, invoking the SCPBL invalidates all RAM contents.

This mode places the CPU in a static, low-power state. SYS_OSC is gated off, so PCLK and HCLK are inactive. The CPU state is not maintained.

The low-power peripherals LPUART0, LPTMR0, and LPTMR1 can be enabled to operate in this mode. The clock source for these peripherals is selectable, but because the primary bus clocks PCLK and HCLK are gated off, the clock source choice is limited. See *Table 9-1* and *Table 13-1* for the clock source table. These three low-power peripherals are disabled/enabled before entering *BACKUP* by setting/clearing the associated bit in the *MCR_PCLKDIS* register, as shown in *Table 4-13*.

Table 4-13: BACKUP Low-Power Peripheral Control Truth Table

Register Settings			Configuration
MCR_PCLKDIS.lpuart0	***MCR_PCLKDIS.lptmr1***	***MCR_PCLKDIS.lptmr0***	
0	0	0	LPUART0, LPTMR1, and LPTMR0 enabled
x	x	1	LPUART0, LPTMR1, and LPTMR0 disabled
x	1	x	LPUART0, LPTMR1, and LPTMR0 disabled
1	x	x	LPUART0, LPTMR1, and LPTMR0 disabled
Note: The setting of any of the MCR_PCLKDIS register bits before entering DEEPSLEEP causes all three of these low-power peripherals to be disabled. *Note: If LPUART0, LPTMR0, and LPTMR1 are enabled, all the outputs associated with these three peripherals are no longer available as GPIO. The GPIO pins affected are P0.6, P0.7, P0.22, P0.23, P0.24, P0.25, P0.26, and P0.27.*			

The RTC has an independent oscillator and can return the device to *ACTIVE*. The ERTCO remains enabled in *BACKUP* if it was enabled before entering *BACKUP*. The watchdog timers are inactive in this mode.

The AoD and RAM retention can optionally be set to automatically disable (and clear) themselves when entering this mode. RAM can be optionally retained. The amount of RAM retained is controlled by appropriately setting the *PWRSEQ_LPCN.ram3ret_en*, *PWRSEQ_LPCN.ram2ret_en*, *PWRSEQ_LPCN.ram1ret_en*, and *PWRSEQ_LPCN.ram0ret_en* register bits.

Table 4-15 and *Table 4-17* show *BACKUP's* effects on the various clock sources.

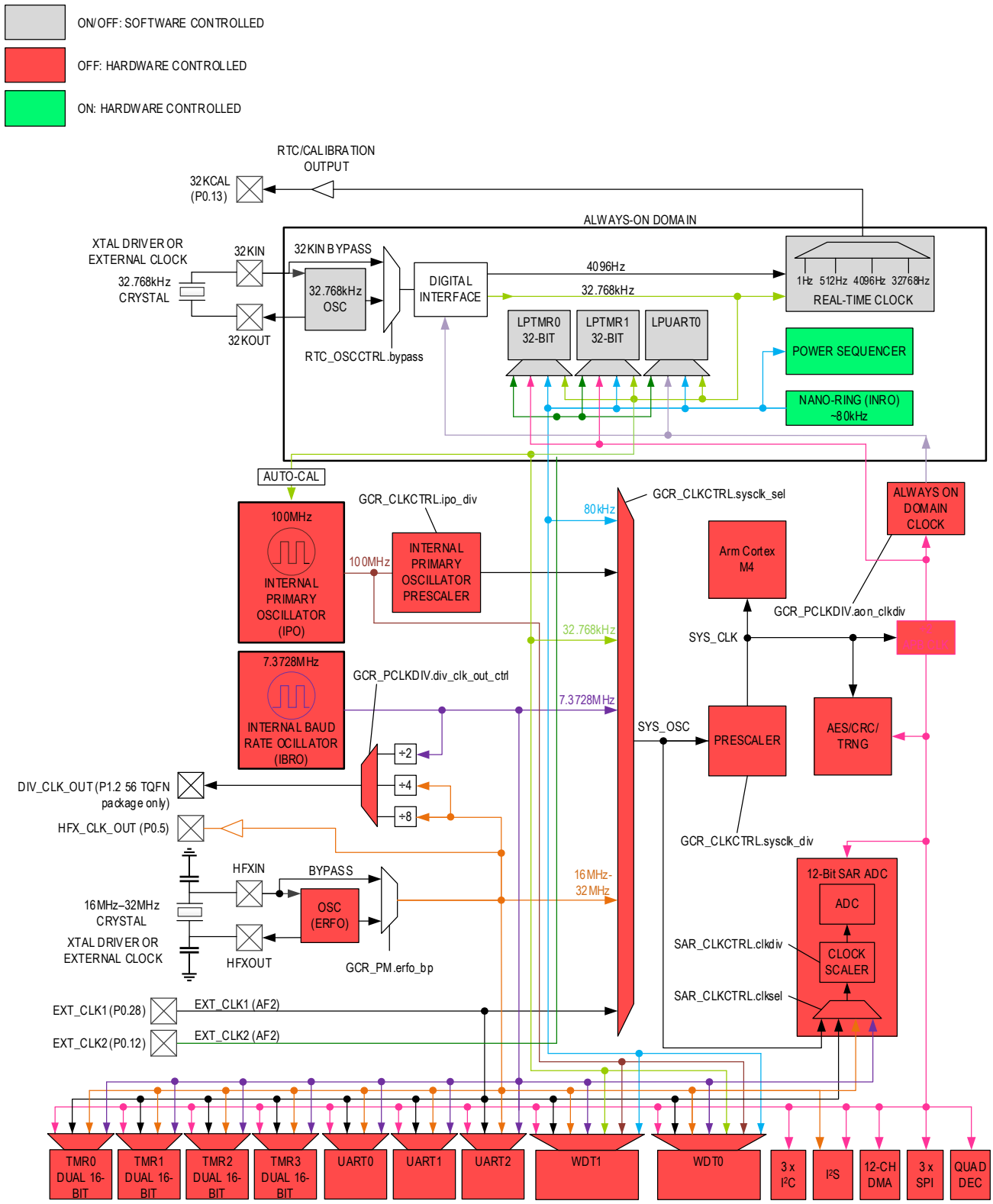
Figure 4-4 shows the clock control during *BACKUP*.

4.3.4.1 Entering BACKUP

Place the device in *BACKUP* by performing the following steps:

1. Configure any desired wake-up functions. See *Table 4-10* for possible wake-up sources.
2. Configure desired RAM retention. See *Table 4-13* for details.
3. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKST0*
   b. *PWRSEQ_LPWKST1*
4. Clear the low-power peripheral wake-up flags and the backup wake-up status flag by writing 0x0001 001F to the *PWRSEQ_LPPWKST* register.
5. Set the *PWRSEQ_LPCN.vcore_det_bypass* bit to 1.
6. Set *GCR_PM.pm* to 4 (*BACKUP*).
7. When the device wakes from *BACKUP*, it resumes operation from the reset vector.

Figure 4-4: MAX32672 DEEPSLEEP and BACKUP Clock Control



4.3.5 STORAGE

This mode is similar to *BACKUP* with the following exceptions:

- No SRAM can be retained.
- LPUART0, LPTMR0, and LPTMR1 are disabled.
- The ERTCO remains enabled in *STORAGE* if it was enabled before entering *STORAGE*.

Table 4-15 and *Table 4-17* show the effects that *STORAGE* has on the various clock sources.

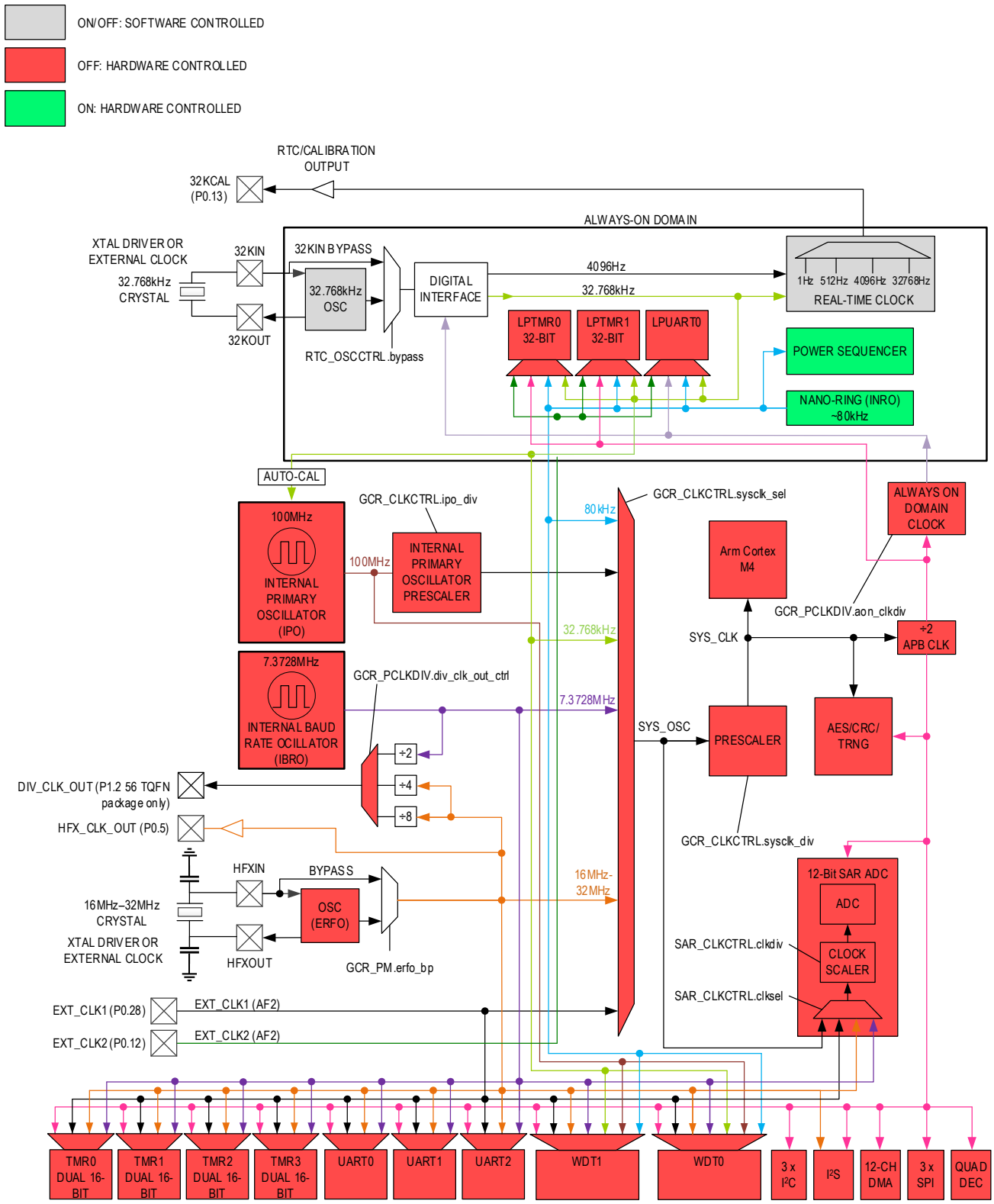
Figure 4-5 shows the clock control during *STORAGE*.

4.3.5.1 Entering STORAGE

Place the device in *STORAGE* by performing the following steps:

1. Configure any desired wake-up functions. See *Table 4-10* for possible wake-up sources.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKST0*
   b. *PWRSEQ_LPWKST1*
3. Clear the low-power peripheral wake-up flags and the backup wake-up status flag by writing 0x0001 001F to the *PWRSEQ_LPPWKST* register.
4. Set the *PWRSEQ_LPCN*.*vcore_det_bypass* bit to 1.
5. Set the *PWRSEQ_LPCN*.*storage_en* bit to 1.
6. Set *GCR_PM*.*pm* to 4 (*BACKUP*).
7. When the device wakes from *STORAGE*, it resumes operation from the reset vector.

Figure 4-5: MAX32672 STORAGE Clock Control



4.4 Shutdown State

Shutdown state is not a low-power mode. It is intended to zeroize all volatile memory in the device.

In the shutdown state, internal power, including the AoD, is gated off. There is no data or register retention. Power is removed from the RAM, effectively zeroizing the RAM contents in this mode. All wakeup sources, wakeup logic, and interrupts are disabled. The device only exits this state through a POR which reinitializes the device.

Setting the *GCR_PM.mode* field to 7 results in the device entering shutdown state immediately.

4.5 Device Resets

Four device resets are available:

- Peripheral Reset
- Soft Reset
- System Reset (includes external and watchdog resets)
- Power-On Reset (POR)

On completion of any reset cycle, HCLK and PCLK are operational, the CPU core receives clocks and power, and the device is in *ACTIVE*. Program execution begins at the reset vector address on completion of a system reset or POR. Contents of the AoD are reset only upon power cycling V_{DD} and V_{CORE}. Soft reset and peripheral reset do not reset the Arm core and do not reset the program counter.

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

Table 4-14, *Table 4-15*, *Table 4-16*, and *Table 4-17* show the effects on the system of the four reset types and the five power modes.

Table 4-14: MAX32672 Clock Source and Reset Effects

	Peripheral Reset[4]	Soft Reset[4]	System Reset[4]	POR
GCR	-	-	Reset	Reset
INRO	On	On	On	On
ERTCO	-	-	-	Off
IBRO	-	-	Off	Off
ERFO	-	-	Off	Off
IPO	-	-	On	On
SYS_CLK	On	On	On[2]	On[2]
CPU Clock	On	On	On	On

Table key:
SW = 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 *DEEPSLEEP*; restored to system reset state when exiting *BACKUP* or *STORAGE*.

1: AoD is only reset upon power cycling V_{DD} and V_{CORE}.
2: On a system reset or POR, the IPO is automatically selected as SYS_OSC.
3: A system reset occurs when exiting *BACKUP* or *STORAGE*. The system reset does not affect the low-power peripherals.
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 watchdog reset.

Table 4-15: MAX32672 Clock Source and Global Control Register Low-Power Mode Effects

	ACTIVE	*SLEEP*	*DEEPSLEEP*	*BACKUP*[3]	*STORAGE*
GCR	R	-	-	-	-
INRO	On	On	On	On	On
ERTCO	SW	SW	SW	SW	SW
IBRO	SW	-	R	R	R
ERFO	SW	-	R	R	R
IPO	SW	-	R	R	R
SYS_CLK	On	On	Off	Off	Off
CPU Clock	On	Off	Off	Off	Off

Table key:
SW = 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 *DEEPSLEEP;* restored to system reset state when exiting *BACKUP* or *STORAGE*.

1: AoD is only reset upon power cycling V_{DD} and V_{CORE}.
2: On a system reset or POR, the IPO is automatically selected as the SYS_OSC.
3: A system reset occurs when exiting *BACKUP* or *STORAGE*. The system reset does not affect the low-power peripherals.
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 watchdog reset.

Table 4-16: MAX32672 Peripheral and CPU Reset Effects

	Peripheral Reset[4]	Soft Reset[4]	System Reset[4]	POR
RTC	-	-	-	Reset
CPU	-	-	Reset	Reset
WDT0/1	-	-	Reset	Reset
GPIO	-	Reset	Reset	Reset
Low-Power Peripherals	Reset	Reset	Reset	Reset
Other Peripherals	Reset	Reset	Reset	Reset
Always-On Domain[1]	-	-	-	Reset
RAM Retention	-	-	-	Reset

Table key:
SW = 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 *DEEPSLEEP*; restored to system reset state when exiting *BACKUP* or *STORAGE*.
1: AoD is only reset upon power cycling V_{DD} and V_{CORE}.
2: On a system reset or POR, the IPO is automatically selected as the SYS_OSC.
3: A system reset occurs when exiting *BACKUP* or *STORAGE*. The system reset does not affect the low-power peripherals.
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 watchdog reset.

Table 4-17: MAX32672 Peripheral and CPU Low-Power Mode Effects

	ACTIVE	*SLEEP*	*DEEPSLEEP*	*BACKUP*[3]	*STORAGE*
RTC	SW	SW	SW	SW	SW
CPU	R	Off	Off	Off	Off
WDT0/1	R	-	Off	Off	Off
GPIO	R	-	-	-	-
Low-Power Peripherals	SW	SW	SW	SW	Off
Other Peripherals	R	-	Off	Off	Off
Always-On Domain[1]	-	-	-	-	-
RAM Retention	-	-	On	SW	Off

Table key:
SW = 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 *DEEPSLEEP*; restored to system reset state when exiting *BACKUP* or *STORAGE*.
1: AoD is only reset upon power cycling V_{DD} and V_{CORE}.
2: On a system reset or POR, the IPO is automatically selected as the SYS_OSC.
3: A system reset occurs when exiting *BACKUP* or *STORAGE*. The system reset does not affect the low-power peripherals.
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 watchdog reset.

4.5.1 Peripheral Reset

As shown in *Table 4-14* and *Table 4-16*, peripheral reset performs a reset for all peripherals. The CPU retains its state. The GPIO, watchdog timers, AoD, RAM retention, GCR, including the clock configuration, are unaffected.

To start a peripheral reset, set *GCR_RST0.periph* to 1. The reset is completed immediately.

4.5.2 Soft Reset

As shown in *Table 4-14* and *Table 4-16*, a soft reset is the same as a peripheral reset, except that it also resets the GPIO to its POR state.

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

4.5.3 System Reset

As shown in *Table 4-14* and *Table 4-16*, a system reset is the same as a soft reset, except it also resets all the 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.

An external reset and a watchdog timer reset event initiates a system reset. To perform a system reset from software, set *GCR_RST0.sys* = 1.

4.5.4 Power-On Reset (POR)

As shown in *Table 4-14* and *Table 4-16*, a POR resets everything in the device to its default state.

4.6 Unified Internal Cache Controller (ICC)

ICC is the cache controller used for the internal flash memory. ICC includes a line buffer, tag RAM, and a 16KB two-way set-associative data RAM.

4.6.1 Enabling ICC

Perform the following steps to enable ICC:

1. Write 1 to the *ICC_INVALIDATE* register to force a flush of the cache.
2. Read *ICC_CTRL.rdy* until it returns 1.
3. Set *ICC_CTRL.en* to 1.
4. Read *ICC_CTRL.rdy* until it returns 1.

4.6.2 Disabling ICC

Disable the ICC by setting *ICC_CTRL.en* to 0.

4.6.3 Invalidating ICC Cache

The system configuration register (*GCR_SYSCTRL*) includes a field for flushing the ICC. Setting the *GCR_SYSCTRL.icc0_flush* field to 1 flushes the ICC cache and the tag RAM. Setting the *ICC_INVALIDATE* register to 1 invalidates the ICC cache and forces a cache flush. Read the *ICC_CTRL.rdy* field until it returns 1 to determine when the flush is completed.

4.7 ICC 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 4-18: Internal Cache Controller Register Summary

Offset	Register	Description
[0x0000]	*ICC_INFO*	*Cache ID Register*
[0x0004]	*ICC_SZ*	*Cache Memory Size Register*
[0x0100]	*ICC_CTRL*	*Internal Cache Control Register*
[0x0700]	*ICC_INVALIDATE*	*Internal Cache Controller Invalidate Register*

4.7.1 Register Details

Table 4-19: ICC Cache Information Register

ICC Cache Information			ICC_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** Returns the release number for this cache instance.

Table 4-20: ICC Memory Size Register

ICC Memory Size			ICC_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: Cache RAM.

Table 4-21: ICC Cache Control Register

ICC Cache Control			ICC_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	R/W	-	**Reserved**
16	rdy	R	1	**Ready** This field is cleared by hardware anytime the cache as a whole is invalidated (including a POR). 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. Once set to 1, setting this field to 0 invalidates the cache contents, and the line fill buffer handles reads. After a POR, the cache should be flushed before it is enabled. 0: Disabled. 1: Enabled.

Table 4-22: ICC Invalidate Register

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

4.8 RAM Memory Management

This device has many features for managing the on-chip RAM. The on-chip RAM includes data RAM, internal cache.

4.8.1 On-Chip Cache Management

The internal cache controller fetches code from the flash memory. The cache can be enabled, disabled, and zeroized, and the cache clock can be disabled by placing it in *LIGHTSLEEP*. See the *Unified Internal Cache Controller* section for details.

4.8.2 RAM Zeroization

The GCR Memory Zeroize register, *GCR_MEMZ*, allows clearing memory for software or security reasons. Zeroization writes all zeros to the specified memory.

The following RAMs can be zeroized:

- Internal System RAM
- The system RAM can be zeroized by setting the respective field to 1 in the *GCR_MEMZ* register.
- ICC 16KB Cache
- Write 1 to *GCR_MEMZ.icc0*

4.8.3 RAM Low-Power Modes

RAM low-power modes and shutdown are controlled on a bank basis. The system RAM banks are shown with corresponding bank sizes and base addresses *Table 3-1*.

4.8.4 RAM LIGHTSLEEP

RAM can be placed in a low-power mode, referred to as *LIGHTSLEEP*, using the memory clock control register, *GCR_MEMCTRL*. *LIGHTSLEEP* gates off the clock to the RAM and makes the RAM unavailable for read/write operations while memory contents are retained, reducing power consumption. *LIGHTSLEEP* is available for the four system RAM blocks, the ECC RAM blocks, and the ICC RAM.

4.8.5 RAM Shutdown

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

4.9 Miscellaneous Control Registers (MCR)

This set of control registers provides reset and clock control for the AoD peripherals (LPUART0, LPTMR0, and LPTMR1).

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 4-23: Miscellaneous Control Register Summary

Offset	Register	Description
[0x0004]	*MCR_RST*	*Reset Control Register*
[0x0008]	*MCR_CLKCTRL*	*Clock Control Register*
[0x000C]	*MCR_AINCOMP*	*Analog Input Comparator Register*
[0x0010]	*MCR_LPPIOCTRL*	*Low-Power Peripheral I/O Control Register*
[0x0024]	*MCR_PCLKDIS*	*Clock Disable Register*
[0x0034]	*MCR_AESKEY*	*AES Key Register*
[0x0038]	*MCR_ADC_CFG0*	*ADC Config Register 0*
[0x003C]	*MCR_ADC_CFG1*	*ADC Config Register 1*
[0x0040]	*MCR_ADC_CFG2*	*ADC Config Register 2*
[0x0044]	*MCR_ADC_CFG3*	*ADC Config Register 3*

4.9.1 Registers Details

Table 4-24: Reset Control Register

Reset Control			MCR_RST	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	rtc	R/W1O	0	**RTC Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *Note: This field is not used on 0xA1 silicon revisions, for 0xA1 revision silicon use GCR_RST0.rtc instead. See the GCR_REVISION register to determine the silicon revision.*
2	lpuart0	R/W1O	0	**LPUART0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *See the Device Resets section for additional information.*
1	lptmr1	R/W1O	0	**LPTMR1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *See the Device Resets section for additional information.*
0	lptmr0	R/W1O	0	**LPTMR0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *See the Device Resets section for additional information.*

Table 4-25: Clock Control Register

Clock Control			MCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:18	-	RO	0	**Reserved**

Clock Control			MCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
17	ertco_en	R/W	0	**ERTCO Enable** This field enables the ERTCO. 0: Disabled. 1: Enabled.
16	ertco_pd	R/W	1	**ERTCO Power Down** This field powers down the ERTCO. 0: ERTCO powered on. 1: ERTCO powered down.
15:0	-	RO	0	**Reserved**

Table 4-26: Analog Comparator Register

Analog Comparator			MCR_AINCOMP	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:28	psel_comp1	R/W	0	**Comparator 1 Positive Input Select** This field selects the positive input for comparator 1. Before enabling a comparator input selection, the corresponding GPIO must be configured for the corresponding alternate function for the input. Refer to the device data sheet for the alternate function details. 0: None. 1: AIN4. 2: AIN5. 4: AIN6. 8: AIN7. All other values are reserved. *Note: None is only a valid selection if the comparator is powered down. See MCR_AINCOMP.pd for details on powering down a comparator.*
27:24	nsel_comp1	R/W	0	**Comparator 1 Negative Input Select** This field selects the negative input for comparator 1. Before enabling a comparator input selection, the corresponding GPIO must be configured for the corresponding alternate function for the input. Refer to the device data sheet for the alternate function details. 0: None. 1: AIN0. 2: AIN1. 4: AIN2. 8: AIN3. All other values are reserved. *Note: None is only a valid selection if the comparator is powered down. See MCR_AINCOMP.pd for details on powering down a comparator.*

Analog Comparator			MCR_AINCOMP	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
23:20	psel_comp0	R/W	0	**Comparator 0 Positive Input Select** This field selects the positive input for comparator 0. Before enabling a comparator input selection, the corresponding GPIO must be configured for the corresponding alternate function for the input. Refer to the device data sheet for the alternate function details. 0: None. 1: AIN4. 2: AIN5. 4: AIN6. 8: AIN7. All other values are reserved. *Note: None is only a valid selection if the comparator is powered down. See MCR_AINCOMP.pd for details on powering down a comparator.*
19:16	nsel_comp0	R/W	0	**Comparator 0 Negative Input Select** This field selects the negative input for comparator 0. Before enabling a comparator input selection, the corresponding GPIO must be configured for the corresponding alternate function for the input. Refer to the device data sheet for the alternate function details. 0: None. 1: AIN0. 2: AIN1. 4: AIN2. 8: AIN3. All other values are reserved. *Note: None is only a valid selection if the comparator is powered down. See MCR_AINCOMP.pd for details on powering down a comparator.*
15:4	-	RO	0	**Reserved**
3:2	hyst	RO	0	**Analog Comparator Hysteresis Control** Reserved.
1	pd1	R/W	1	**Analog Comparator 1 Power Down** This field enables analog comparator 1. 0: Enabled. 1: Disabled. *Note: The analog inputs must be configured before enabling a comparator. See fields MCR_AINCOMP.psel_comp1 and MCR_AINCOMP.nsel_comp1.*
0	pd0	R/W	1	**Analog Comparator 0 Power Down** This field enables analog comparator 0. 0: Enabled. 1: Disabled. *Note: The analog inputs must be configured before enabling a comparator. See fields MCR_AINCOMP.psel_comp0 and MCR_AINCOMP.nsel_comp0.*

Table 4-27: Low-Power Peripheral I/O Control Register

Low-Power Peripheral I/O Control			MCR_LPPIOCTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**

Low-Power Peripheral I/O Control			MCR_LPPIOCTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
7	lpuart0_rts	R/W	0	**LPUART0 (UART3) RTS Control Enable** This field enables peripheral control for the RTS signal for LPUART0. When this field is enabled and the LPUART0 peripheral is enabled (*MCR_PCLKDIS*.*lpuart0* = 0), the LPUART0 peripheral controls the state of the RTS signal. Otherwise, the RTS signal is controlled by GPIO. 0: Disabled. 1: LPUART0 controls the RTS signal if the LPUART0 is enabled.
6	lpuart0_cts	R/W	0	**LPUART0 (UART3) CTS Control Enable** This field enables peripheral control for the CTS signal for LPUART0. When this field is enabled and the LPUART0 peripheral is enabled (*MCR_PCLKDIS*.*lpuart0* = 0), the LPUART0 peripheral controls the state of the CTS signal. Otherwise, the CTS signal is controlled by GPIO. 0: Disabled. 1: LPUART0 controls the CTS signal if the LPUART0 is enabled.
5	lpuart0_tx	R/W	0	**LPUART0 (UART3) TX Control Enable** This field enables peripheral control for the TX signal for LPUART0. When this field is enabled and the LPUART0 peripheral is enabled (*MCR_PCLKDIS*.*lpuart0* = 0), the LPUART0 peripheral controls the state of the TX signal. Otherwise, the TX signal is controlled by GPIO. 0: Disabled. 1: LPUART0 controls the TX signal if the LPUART0 is enabled.
4	lpuart0_rx	R/W	0	**LPUART0 (UART3) RX Control Enable** This field enables peripheral control for the RX signal for LPUART0. When this field is enabled and the LPUART0 peripheral is enabled (*MCR_PCLKDIS*.*lpuart0* = 0), the LPUART0 peripheral controls the state of the RX signal. Otherwise, the RX signal is controlled by GPIO. 0: Disabled. 1: LPUART0 controls the RX signal if the LPUART0 is enabled.
3	lptmr1_o	R/W	0	**LPTMR1 (TMR5) Output Control** This field enables peripheral control for the LPTMR1 output signal. When this field is enabled and the LPTMR1 peripheral is enabled (*MCR_PCLKDIS*.*lptmr1* = 0), the LPTMR1 peripheral controls the state of the output signal. Otherwise, the output signal is controlled by GPIO. 0: Disabled. 1: LPTMR1 controls the output signal if the LPTMR1 is enabled.
2	lptmr1_i	R/W	0	**LPTMR1 (TMR5) Input Control** This field enables peripheral control for the LPTMR1 input signal. When this field is enabled and the LPTMR1 peripheral is enabled (*MCR_PCLKDIS*.*lptmr1* = 0), the LPTMR1 peripheral controls the state of the input signal. Otherwise, the input signal is controlled by GPIO. 0: Disabled. 1: LPTMR1 controls the input signal if the LPTMR1 is enabled.
1	lptmr0_o	R/W	0	**LPTMR0 (TMR4) Output Control** This field enables peripheral control for the LPTMR0 output signal. When this field is enabled and the LPTMR0 peripheral is enabled (*MCR_PCLKDIS*.*lptmr0* = 0), the LPTMR0 peripheral controls the state of the output signal. Otherwise, the output signal is controlled by GPIO. 0: Disabled. 1: LPTMR0 controls the output signal if the LPTMR0 is enabled.

Low-Power Peripheral I/O Control			MCR_LPPIOCTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
0	lptmr0_i	R/W	0	**LPTMR0 (TMR4) Input Control** This field enables peripheral control for the LPTMR0 input signal. When this field is enabled and the LPTMR0 peripheral is enabled (*MCR_PCLKDIS.lptmr0* = 0), the LPTMR0 peripheral controls the state of the input signal. Otherwise, the input signal is controlled by GPIO. 0: Disabled. 1: LPTMR0 controls the input signal if the LPTMR0 is enabled.

Table 4-28: Clock Disable Register

Clock Disable			MCR_PCLKDIS	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	lpuart0	R/W	0	**LPUART0 (UART3) Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. For *DEEPSLEEP* and *BACKUP* operation, see the *DEEPSLEEP* section and the *BACKUP* section. 0: Enabled. 1: Disabled.
1	lptmr1	R/W	0	**LPTMR1 (TMR5) Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. For *DEEPSLEEP* and *BACKUP* operation, see the *DEEPSLEEP* section and the *BACKUP* section. 0: Enabled. 1: Disabled.
0	lptmr0	R/W	0	**LPTMR0 (TMR4) Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. For *DEEPSLEEP* and *BACKUP* operation, see the *DEEPSLEEP* section and the *BACKUP* section. 0: Enabled. 1: Disabled.

Table 4-29: AES Key Pointer/Status Register

AES Key Pointer/Status			MCR_AESKEY	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	ptr	R/W	0	**AES Key Pointer/Status**

Table 4-30: ADC Configuration Register 0

ADC Configuration 0			MCR_ADC_CFG0	[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_ADC_CFG0.ext_ref* = 0). 0: 1.25V. 1: 2.048V.

ADC Configuration 0			MCR_ADC_CFG0	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
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	lp_50k_dis	R/W	0	**Divide by 2 (50KΩ) Disable in Low-Power Modes** When this field is set, if any channel is configured for divide by 2 (50kΩ) (*MCR_ADC_CFG2.ch<n>* = 1), the channel is set to pass-through mode in low-power modes. 0: Divide by 2 (50kΩ) enabled in low-power modes. 1: Divide by 2 (50kΩ) disabled in low-power modes.
0	lp_5k_dis	R/W	0	**Divide by 2 (5KΩ) Disable in Low-Power Modes** When this field is set, if any channel is configured for divide by 2 (5kΩ) (*MCR_ADC_CFG2.ch<n>* = 1), the channel is set to pass-through mode in low-power modes. 0: Divide by 2 (5kΩ) enabled in low-power modes. 1: Divide by 2 (5kΩ)disabled in low-power modes.

Table 4-31: ADC Configuration Register 1

ADC Configuration 1			MCR_ADC_CFG1	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:13	-	RO	0	**Reserved**
12	ch12_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch12* field. 0: Disabled (*MCR_ADC_CFG2.ch12* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch12* selection is only used when the channel is selected).
11	ch11_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch11* field. 0: Disabled (*MCR_ADC_CFG2.ch11* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch11* selection is only used when the channel is selected).
10	ch10_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch10* field. 0: Disabled (*MCR_ADC_CFG2.ch10* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch10* selection is only used when the channel is selected).
9	ch9_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch9* field. 0: Disabled (*MCR_ADC_CFG2.ch9* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch9* selection is only used when the channel is selected).

ADC Configuration 1			MCR_ADC_CFG1	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
8	ch8_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch8* field. 0: Disabled (*MCR_ADC_CFG2.ch8* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch8* selection is only used when the channel is selected).
7	ch7_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch7* field. 0: Disabled (*MCR_ADC_CFG2.ch7* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch7* selection is only used when the channel is selected).
6	ch6_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch6* field. 0: Disabled (*MCR_ADC_CFG2.ch6* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch6* selection is only used when the channel is selected).
5	ch5_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch5* field. 0: Disabled (*MCR_ADC_CFG2.ch5* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch5* selection is only used when the channel is selected).
4	ch4_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch4* field. 0: Disabled (*MCR_ADC_CFG2.ch4* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch4* selection is only used when the channel is selected).
3	ch3_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch3* field. 0: Disabled (*MCR_ADC_CFG2.ch3* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch3* selection is only used when the channel is selected).
2	ch2_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch2* field. 0: Disabled (*MCR_ADC_CFG2.ch2* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch2* selection is only used when the channel is selected).
1	ch1_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch1* field. 0: Disabled (*MCR_ADC_CFG2.ch1* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch1* selection is only used when the channel is selected).
0	ch0_pu_dyn	R/W	0	**ADC Dynamic Pullup Enable** This field enables the dynamic application of the divider selected in the *MCR_ADC_CFG2.ch0* field. 0: Disabled (*MCR_ADC_CFG2.ch0* selection is always used). 1: Enabled (*MCR_ADC_CFG2.ch0* selection is only used when the channel is selected).

Table 4-32: ADC Configuration Register 2

ADC Configuration 2			MCR_ADC_CFG2	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
25:24	ch12	R/W	0	**ADC Input Channel 12 Divider Option** 0: Pass through (no divider). 1: Divide by 2 (5kΩ). 2: Divide by 2 (50kΩ). 3: Reserved. *Note: This channel is always set to pass-through mode in all low-power modes.*
23:22	ch11	R/W	0	**ADC Input Channel 11 Divider Option** 0: Pass-through (no divider). 1: Divide by 2 (5kΩ). 2: Divide by 2 (50kΩ). 3: Reserved.
21:20	ch10	R/W	0	**ADC Input Channel 10 Divider Option** 0: Pass-through (no divider). 1: Divide by 2 (5kΩ) 2: Divide by 2 (50kΩ) 3: Reserved
19:18	ch9	R/W	0	**ADC Input Channel 9 Divider Option** 0: Pass-through (no divider) 1: Divide by 2 (5kΩ) 2: Divide by 2 (50kΩ) 3: Reserved
17:16	ch8	R/W	0	**ADC Input Channel 8 Divider Option** 0: Pass-through (no divider) 1: Divide by 2 (5kΩ). 2: Divide by 2 (50kΩ). 3: Reserved.
15:14	ch7	R/W	0	**ADC Input Channel 7 Divider Option** 0: Pass-through (no divider). 1: Divide by 2 (5kΩ). 2: Divide by 2 (50kΩ). 3: Reserved.
13:12	ch6	R/W	0	**ADC Input Channel 6 Divider Option** 0: Pass-through (no divider). 1: Divide by 2 (5kΩ). 2: Divide by 2 (50kΩ). 3: Reserved.
11:10	ch5	R/W	0	**ADC Input Channel 5 Divider Option** 0: Pass-through (no divider). 1: Divide by 2 (5kΩ). 2: Divide by 2 (50kΩ). 3: Reserved.
9:8	ch4	R/W	0	**ADC Input Channel 4 Divider Option** 0: Pass-through (no divider). 1: Divide by 2 (5kΩ). 2: Divide by 2 (50kΩ). 3: Reserved.

ADC Configuration 2			MCR_ADC_CFG2	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
7:6	ch3	R/W	0	**ADC Input Channel 3 Divider Option** 0: Pass-through (no divider). 1: Divide by 2 (5kΩ). 2: Divide by 2 (50kΩ). 3: Reserved.
5:4	ch2	R/W	0	**ADC Input Channel 2 Divider Option** 0: Pass-through (no divider). 1: Divide by 2 (5kΩ). 2: Divide by 2 (50kΩ). 3: Reserved.
3:2	ch1	R/W	0	**ADC Input Channel 1 Divider Option** 0: Pass-through (no divider). 1: Divide by 2 (5kΩ). 2: Divide by 2 (50kΩ). 3: Reserved.
1:0	ch0	R/W	0	**ADC Input Channel 0 Divider Option** 0: Pass-through (no divider). 1: Divide by 2 (5kΩ). 2: Divide by 2 (50kΩ). 3: Reserved.

Table 4-33: ADC Configuration Register 3

ADC Configuration 3			MCR_ADC_CFG3	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31:25	-	RO	0	**Reserved**
24	d_iboost	R/W	0	**Drive Current Boost Trim** See the *ADC SFR Interface* section for details.
23:22	atb	DNM	0	**Do Not Modify**
21:20	vcm	R/W	0	**V_{CM} Output of Reference Buffer** See the *ADC SFR Interface* section for details.
19:16	idrv	R/W	0	**Reference Buffer Drive Strength** See the *ADC SFR Interface* section for details.
15	-	RO	0	**Reserved**
14:8	vrefp	R/W	0	**V_{REFP} Output of Reference Buffer** See the *ADC SFR Interface* section for details.
7	-	RO	0	**Reserved**
6:0	vrefm	R/W	0	**V_{REFM} Output of Reference Buffer** See the *ADC SFR Interface* section for details.

4.10 Power Sequencer and Always-On Domain 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. The PWRSEQ registers are only reset on a POR.

Table 4-34: Power Sequencer and Always-On Domain Register Summary

Offset	Register	Description
[0x0000]	*PWRSEQ_LPCN*	*Low-Power Control Register*
[0x0004]	*PWRSEQ_LPWKST0*	*GPIO0 Low-Power Wakeup Status Flags*
[0x0008]	*PWRSEQ_LPWKEN0*	*GPIO0 Low-Power Wakeup Enable Register*
[0x000C]	*PWRSEQ_LPWKST1*	*GPIO1 Low-Power Wakeup Status Flags*
[0x0010]	*PWRSEQ_LPWKEN1*	*GPIO1 Low-Power Wakeup Enable Register*
[0x0030]	*PWRSEQ_LPPWKST*	*Peripheral Low-Power Wakeup Status Flags*
[0x0034]	*PWRSEQ_LPPWKEN*	*Peripheral Low-Power Wakeup Enable Register*
[0x0040]	*PWRSEQ_LPMEMSD*	*RAM Shutdown Control Register*
[0x0048]	*PWRSEQ_GPR0*	*General Purpose 0 Register*
[0x004C]	*PWRSEQ_GPR1*	*General Purpose 1 Register*

4.10.1 Register Details

Table 4-35: Low-Power Control Register

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	DNM	0	**Reserved, Do Not Modify**
29	ertco_en	R/W	0	**ERTCO Low-Power Mode Control** Set this field to enable the ERTCO in low-power modes. 0: ERTCO state controlled by power sequencer. 1: ERTCO enabled during low-power modes.
28	inro_en	R/W	0	**INRO Low-Power Mode Control** This bit allows control of the INRO for low-power modes. 0: INRO controlled by power sequencer. 1: INRO enabled in low-power modes. *Note: If PWRSEQ_LPCN.storage_en is 1 this field is ignored and INRO is powered off.*
27	vbbmon_dis	R/W	0	**ADC Digital Supply Monitor Disable** This field controls the power monitor on the ADC digital supply in all operating modes. 0: Enable if *PWRSEQ_LPCN.bg* is enabled. 1: Disabled.
26	-	RO	0	**Reserved**
25	porvddmon_dis	R/W	0	**V_{DD} GPIO Supply POR Monitor Disable** Setting this field to 1 disables the V_{DDIO} supply monitor in all operating modes. 0: Enabled. 1: Disabled.
24:23	-	RO	0	**Reserved**

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
22	vddamon_dis	R/W	0	**V_{DDA} Analog Supply Power Monitor Disable** Set this field to 1 to disable the V_{DDA} supply monitor. 0: Enabled. 1: Disabled.
21	-	RO	0	**Reserved**
20	vcoremon_dis	R/W	0	**V_{CORE} Supply Power Monitor Disable** Set this field to 1 to disable the V_{CORE} supply monitor. This field is ignored if *PWRSEQ_LPCN.bg_dis* = 1. 0: Enabled. 1: Disabled.
19:18	-	RO	0	**Reserved**
17	vcore_ext	R/W	0	**V_{CORE} Supply** Setting this bit allows the V_{CORE} device pin to be used as the internal 1V supply.
16	ldo_dis	R/W	1	**LDO Disable** This field initializes to 1 on a POR until the hardware determines if an external power source is connected to the V_{CORE} device pin. If no power supply is connected, this bit is set to 0 by the hardware. If a power supply is connected to the V_{CORE} device pin, the bit remains set to 1. Set this field to 1 to manually disable the LDO. 0: Enabled. 1: Disabled.
15:13	-	RO	0	**Reserved**
12	vcorepor_dis	R/W	1	**V_{CORE} POR Disable for *DEEPSLEEP* and *BACKUP*** Setting this bit to 1 blocks the POR signal to the core when the device is in *DEEPSLEEP* or *BACKUP* operation. Disconnecting the POR signal from the core during *DEEPSLEEP* and *BACKUP* prevents the core from detecting a POR event while the device is in *DEEPSLEEP* or *BACKUP*. 0: POR signal is connected to the core during *DEEPSLEEP* and *BACKUP*. 1: POR signal is not connected to the core during *DEEPSLEEP* and *BACKUP*.
11	bg_dis	R/W	1	**Bandgap Disable for *DEEPSLEEP* and *BACKUP*** Setting this field to 1 disables the bandgap during *DEEPSLEEP* and *BACKUP*. 0: Enabled. 1: Disabled.
10	fastwk_en	R/W	0	**Fast Wakeup Enable for *DEEPSLEEP* Mode** Set to 1 to enable fast wakeup from *DEEPSLEEP*. When enabled, the system exits *DEEPSLEEP* faster by: • Bypassing the INRO warmup • Reducing the warmup time for the IPO • Reducing the warmup time for the LDO • Code execution resumes at the next instruction after the entry to *DEEPSLEEP*. When fast wakeup is disabled code execution begins at the reset vector as if a reset occurred. 0: Disabled. 1: Enabled.

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
9	storage_en	R/W	0	***STORAGE* Mode Enable** 0: Disabled. 1: Enabled. *Note: Setting this bit causes the device to enter STORAGE when setting GCR_PM.mode to BACKUP.*
8	retreg_en	R/W	1	**RAM Retention Regulator Enable for *BACKUP*** This field selects the source used to retain the RAM contents during *BACKUP* operation. Setting this field to 0 sets the V_{DD} supply for RAM retention during *BACKUP* and disables the RAM retention regulator. 0: RAM retention regulator disabled. The V_{DD} supply is used to retain the state of the internal SRAM as configured by the *PWRSEQ_LPCN.ram0ret_en, PWRSEQ_LPCN.ram1ret_en, PWRSEQ_LPCN.ram2ret_en, and PWRSEQ_LPCN.ram3ret_en* fields. 1: RAM retention regulator enabled. RAM retention in *BACKUP* is configured with the *PWRSEQ_LPCN.ram0ret_en, PWRSEQ_LPCN.ram1ret_en, PWRSEQ_LPCN.ram2ret_en, and PWRSEQ_LPCN.ram3ret_en* fields.
7	fvdden	R/W	0	**Flash V_{DD} Enable in Low-Power Modes** Set this field to 1 to enable the flash to remain fully powered-on during low-power modes. 0: Flash powered down in low-power modes. 1: Flash powered on during low-power modes.
6	vcore_det_bypass	R/W	0	**Bypass V_{CORE} External Supply Detection** Set this field to 1 if the system runs from a single supply only and V_{CORE} is not connected to an external supply. Set this field to 1 before entering low-power modes to enable a faster wake-up time. 0: Enabled. 1: Disabled.
5:4	ovr	R/W	0b10	**Output Voltage Range for Internal Regulator** Set this field to control the output voltage of the internal regulator, allowing selection of the internal core operating voltage and the frequency of the IPO. On POR, this field defaults to 1.1V output ± 10% with f_{IPO} = 100MHz. ***Note**: If V_{CORE} is connected to an external supply voltage, this field should be modified only to set it to match the input voltage on V_{CORE}. The external supply must be equal to or greater than this field setting indication.* **Dual-Supply Operation:** 0b11: Reserved. 0b10: V_{CORE} = 1.1V, f_{IPO} = 100MHz. 0b01: V_{CORE} = 1.0V, f_{IPO} = 50MHz. 0b00: V_{CORE} = 0.9V, f_{IPO} = 12MHz. **Single-Supply Operation (V_{CORE} = GND)** 0b11: Reserved. 0b10: V_{LDO} = 1.1V, f_{IPO} = 100MHz. 0b01: V_{LDO} = 1.0V, f_{IPO} = 50MHz. 0b00: V_{LDO} = 0.9V, f_{IPO} = 12MHz.

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
3	ram3ret_en	R/W	0	***sysram3* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram3*. See *Table 3-1* for system RAM configuration. This retention setting also retains the *sysram3* ECC complement located at *sysram7*. 0: Data retention for *sysram3* address space disabled in *BACKUP*. 1: Data retention for *sysram3* address space enabled in *BACKUP.* *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.*
2	ram2ret_en	R/W	0	***sysram2* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram2*. See *Table 3-1* for system RAM configuration. This retention setting also retains the *sysram2* ECC complement located at *sysram6*. 0: Data retention for *sysram2* address space disabled in *BACKUP*. 1: Data retention for *sysram2* address space enabled in *BACKUP*. *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.*
1	ram1ret_en	R/W	0	***sysram1* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram1*. See *Table 3-1* for system RAM configuration. This retention setting also retains the *sysram1* ECC complement located at *sysram5*. 0: Data retention for *sysram1* address space disabled in *BACKUP*. 1: Data retention for *sysram1* address space enabled in *BACKUP*. *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.*
0	ram0ret_en	R/W	0	***sysram0* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram0*. See *Table 3-1* for system RAM configuration. This retention setting also retains the *sysram0* ECC complement located at *sysram4*. 0: Data retention for *sysram0* address space disabled in *BACKUP*. 1: Data retention for *sysram0* address space enabled in *BACKUP*. *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.*

Table 4-36: GPIO0 Low-Power Wakeup Status Flags

GPIO0 Low-Power Wakeup Status Flags			PWRSEQ_LPWKST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1C	0	**GPIO0 Pin Wakeup Status Flag** Whenever a GPIO0 pin, in any power mode, transitions from low-to-high or high-to-low, the corresponding bit in this register is set. The device transitions from a low-power to *ACTIVE* if the corresponding interrupt enable bit is set in *PWRSEQ_LPWKEN0*. This register should be cleared before entering any low-power mode. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 4-37: GPIO0 Low-Power Wakeup Enable Registers

GPIO0 Low-Power Wakeup Enable			PWRSEQ_LPWKEN0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO0 Pin Wakeup Interrupt Enable** Setting a bit in this register causes an interrupt to be generated and wakes up the device from any low-power mode to *ACTIVE* if the corresponding bit in the *PWRSEQ_LPWKST0* register is set. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the device to wake up from a low-power mode on a GPIO pin transition, first set the GPIO wakeup enable register bit GCR_PM.gpio_we = 1.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 4-38: GPIO1 Low-Power Wakeup Status Flags

GPIO1 Low-Power Wakeup Status Flags			PWRSEQ_LPWKST1	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO1 Pin Wakeup Status Flag** Whenever a GPIO1 pin, in any power mode, transitions from low-to-high or high-to-low, the corresponding bit in this register is set. The device transitions from a low-power mode to *ACTIVE* if the corresponding interrupt enable bit is set in *PWRSEQ_LPWKEN1*. This register should be cleared before entering any low-power mode. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 4-39: GPIO1 Low-Power Wakeup Enable Registers

GPIO1 Low-Power Wakeup Enable			PWRSEQ_LPWKEN1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO1 Pin Wakeup Interrupt Enable** Setting a bit in this register causes an interrupt to be generated and wakes up the device from any low-power mode to *ACTIVE* if the corresponding bit in the *PWRSEQ_LPWKST1* register is set. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the device to wake up from a low-power mode on a GPIO pin transition, first set the GPIO wakeup enable register bit GCR_PM.gpio_we = 1.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 4-40: Peripheral Low-Power Wakeup Status Flags

Peripheral Low-Power Wakeup Status Flags			PWRSEQ_LPPWKST	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	backup	R/W	0	***BACKUP* Status** This field is set to 1 by hardware if the wake-up event was from *BACKUP*.
15:7	-	RO	0	**Reserved**
6	aincomp1_out	R	*	**Analog Comparator 1 Output Status** This field returns the output status of the analog comparator 1.

Peripheral Low-Power Wakeup Status Flags			PWRSEQ_LPPWKST	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
5	aincomp0_out	R	*	**Analog Comparator 0 Output Status** This field returns the output status of the analog comparator 0.
4	aincomp1	R/W1C	0	**Analog Comparator 1 Wakeup Flag** This field is set when this peripheral causes the CPU to wake to *ACTIVE*. 0: Normal operation. 1: Wakeup event detected. *Note: If the corresponding bit in the PWRSEQ_LPPWKEN register is set, the event generates an interrupt to wake up the device from a low-power mode when PWRSEQ_LPCN.bg_dis is cleared to 0.*
3	aincomp0	R/W1C	0	**Analog Comparator 0 Wakeup Flag** This field is set when this peripheral causes the CPU to wake to *ACTIVE*. 0: Normal operation. 1: Wakeup event detected. *Note: If the corresponding bit in the PWRSEQ_LPPWKEN register is set, the event generates an interrupt to wake up the device from a low-power mode when PWRSEQ_LPCN.bg_dis is cleared to 0.*
2	lpuart0	R/W1C	0	**LPUART0 Wakeup Flag** This field is set when this peripheral causes the CPU to wake to *ACTIVE*. 0: Normal operation. 1: Wakeup event detected. *Note: If the corresponding bit in the PWRSEQ_LPPWKEN register is set, the event generates an interrupt to wake up the device from a low-power mode when PWRSEQ_LPCN.bg_dis is cleared to 0.*
1	lptmr1	R/W1C	0	**LPTMR1 Wakeup Flag** This field is set when this peripheral causes the CPU to wake to *ACTIVE*. 0: Normal operation. 1: Wakeup event detected. *Note: If the corresponding bit in the PWRSEQ_LPPWKEN register is set, the event generates an interrupt to wake up the device from a low-power mode when PWRSEQ_LPCN.bg_dis is cleared to 0.*
0	lptmr0	R/W1C	0	**LPTMR0 Wakeup Flag** This field is set when this peripheral causes the CPU to wake to *ACTIVE*. 0: Normal operation. 1: Wakeup event detected. *Note: If the corresponding bit in the PWRSEQ_LPPWKEN register is set, the event generates an interrupt to wake up the device from a low-power mode when PWRSEQ_LPCN.bg_dis is cleared to 0.*

Table 4-41: Peripheral Low-Power Wakeup Enable Register

Peripheral Low-Power Wakeup Enable			PWRSEQ_LPPWKEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	aincomp1	R/W	0	**Analog Comparator 1 Wakeup Enable** Setting this bit enables an interrupt and wakes up the device from any low-power mode when *PWRSEQ_LPPWKST.aincomp1* is set to 1.

Peripheral Low-Power Wakeup Enable			PWRSEQ_LPPWKEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
3	aincomp0	R/W	0	**Analog Comparator 0 Wakeup Enable** Setting this bit enables an interrupt and wakes up the device from any low-power mode when *PWRSEQ_LPPWKST.aincomp0* is set to 1.
2	lpuart0	R/W	0	**LPUART0 Wakeup Enable** Setting this bit enables an interrupt and wakes up the device from any low-power mode when *PWRSEQ_LPPWKST.lpuart0* does not equal 0. 0: Disabled. 1: Enabled.
1	lptmr1	R/W		**LPTMR1 Wakeup Enable** Setting this bit enables an interrupt and wakes up the device from any low-power mode when *PWRSEQ_LPPWKST.lptmr1* does not equal 0. 0: Disabled. 1: Enabled.
0	lptmr0	R/W	0	**LPTMR0 Wakeup Enable** Setting this bit enables an interrupt and wakes up the device from any low-power mode when *PWRSEQ_LPPWKST.lptmr0* does not equal 0. 0: Disabled. 1: Enabled.

Table 4-42: RAM Shutdown Control Register

RAM Shutdown Control			PWRSEQ_LPMEMSD	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	ram3	R/W	0	***Sysram3* and *Sysram7* Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *See Table 3-1 for system RAM configuration. Note: See GCR_MEMCTRL register for retention mode power settings.*
2	ram2	R/W	0	***Sysram2* and *Sysram6* Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *See Table 3-1 for system RAM configuration. Note: See GCR_MEMCTRL register for retention mode power settings.*
1	ram1	R/W	0	***Sysram1* and *Sysram5* Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *See Table 3-1 for system RAM configuration. Note: See GCR_MEMCTRL register for retention mode power settings.*
0	ram0	R/W	0	***Sysram0* and *Sysram4* Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. *See Table 3-1 for system RAM configuration. Note: See GCR_MEMCTRL register for retention mode power settings.*

Table 4-43: General Purpose 0 Register

General Purpose 0			PWRSEQ_GPR0	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**General Purpose** This field is usable for general purpose storage and is only reset on a POR.

Table 4-44: General Purpose 1 Register

General Purpose 1			PWRSEQ_GPR1	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**General Purpose** This field is usable for general purpose storage and is only reset on a POR.

4.11 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. The TRIMSIR is only reset on a POR.

Table 4-45: Trim System Initialization Register Summary

Offset	Register	Description
[0x0008]	*TRIMSIR_BB_SIR2*	*Configuration Register 2*
[0x0018]	*TRIMSIR_BB_SIR6*	*Configuration Register 6*

4.11.1 Register Details

Table 4-46: TRIMSIR Configuration 2 Register

TRIMSIR Configuration 2			TRIMSIR_BB_SIR2	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	trim_ibro	R/W	*	**IBRO Trim**
15:14	-	RO	0	**Reserved**
13	fl1eccen	R/W	1	**Flash Bank 1 ECC Enable** Set this field to 1 to enable ECC on the memory. 0: Disabled. 1: Enabled.
12	fl0eccen	R/W	1	**Flash Bank 0 ECC Enable** Set this field to 1 to enable ECC on the memory. 0: Disabled. 1: Enabled.
11	icc0eccen	R/W	1	**ICC ECC Enable** Set this field to 1 to enable ECC on the memory. 0: Disabled. 1: Enabled.
10	ram3eccen	R/W	1	**Sysram3 ECC Enable** Set this field to 1 to enable ECC on the memory. 0: Disabled. 1: Enabled.

TRIMSIR Configuration 2			TRIMSIR_BB_SIR2	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
9	ram2eccen	R/W	1	**Sysram2 ECC Enable** Set this field to 1 to enable ECC on the memory. 0: Disabled. 1: Enabled.
8	ram0_1eccen	R/W	1	**Sysram0/Sysram1 ECC Enable** Set this field to 1 to enable ECC on the memory. 0: Disabled. 1: Enabled.
7:0	trim_ibro_rbias	R/W	*	**IBRO Trim Bias Value**

Table 4-47: TRIMSIR Configuration 6 Register

TRIMSIR Configuration 6			TRIMSIR_BB_SIR6	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:14	-	RO	*	**Reserved**
13:9	rtcx2trim	R/W	*	**RTC X2 Trim**
8:4	rtcx1trim	R/W	*	**RTC X1 Trim**
3:0	-	RO	*	**Reserved**

4.12 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.

Note: The GCR is only reset on a system reset or POR. A soft reset or peripheral reset does not affect these registers.

Table 4-48: 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*

Offset	Register	Description
[0x006C]	GCR_ECCIE	Error Correction Coding Interrupt Enable Register
[0x0070]	GCR_ECCADDR	Error Correction Coding Error Address Register

4.12.1 Register Details

Table 4-49: System Control Register

System Control			GCR_SYSCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	chkres	R	0	**ROM Checksum Calculation Pass/Fail** This field is the result after setting bit GCR_SYSCTRL.cchk. This bit is only valid after the ROM checksum is complete and GCR_SYSCTRL.cchk is cleared. 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 GCR_SYSST.icelock word is not programmed.*
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.
12:7	-	RO	0	**Reserved**
6	icc0_flush	R/W	0	**ICC Flush** Write 1 to flush the ICC. This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect. 0: Normal operation. 1: Flush.
5	fpu_dis	R/W	0	**Floating Point Unit Disable** 0: Enabled 1: Disabled
4	flash_page_flip	DNM	0	**Reserved, Do Not Modify**
3	imbarb	RO	0	**Reserved, Do Not Modify**
2:1	sbusarb	R/W	1	**System Bus Arbitration Scheme** 0: Fixed Burst. 1: Round-Robin. 2: Reserved. 3: Reserved.
0	-	RO	0	**Reserved**

Table 4-50: 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 the Device Resets section 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 the Device Resets section 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 the Device Resets section for additional information.*
28	uart2	R/W	0	**UART2 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
27	-	RO	0	**Reserved**
26	adc	R/W	0	**ADC Peripheral 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 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
23:19	-	RO	0	**Reserved**
18	ctb	R/W	0	**Crypto Toolbox Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
17	rtc	R/W	0	**RTC Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *Note: This field only applies to 0xA1 revision silicon, use MCR_RST.rtc for all other revisions. See the GCR_REVISION register to determine the silicon revision.*
16	i2c0	R/W	0	**I²C0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
15	spi2	R/W	0	**SPI2 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
14	spi1	R/W	0	**SPI1 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
13	spi0	R/W	0	**SPI0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
12	uart1	R/W	0	**UART1 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
11	uart0	R/W	0	**UART0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
10:9	-	RO	0	**Reserved**
8	tmr3	R/W	0	**TMR3 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.

Reset 0			GCR_RST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
7	tmr2	R/W	0	**TMR2 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
6	tmr1	R/W	0	**TMR1 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
5	tmr0	R/W	0	**TMR0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
4	-	RO	-	**Reserved**
3	gpio1	R/W	0	**GPIO1 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
2	gpio0	R/W	0	**GPIO0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
1	wdt0	R/W	0	**Watchdog Timer 0 Peripheral 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 4-51: System Clock Control Register

System Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31	extclk_rdy	R	0	**External Clock Ready** This bit field is set when the signal on the P0.28 device pin, driven by an external clock, is ready. 0: Not ready or not enabled. 1: External clock is ready.
30	-	RO	0	**Reserved**
29	inro_rdy	R	0	**INRO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
28	ibro_rdy	R	0	**IBRO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
27	ipo_rdy	R	0	**IPO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
26	-	RO	0	**Reserved**
25	ertco_rdy	R	0	**ERTCO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
24	erfo_rdy	R	0	**ERFO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
23:22	-	RO	0	**Reserved**

System Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
21	ibro_vs	R/W	0	**IBRO Voltage Source Select** In *DEEPSLEEP*, the IBRO voltage is sourced by a dedicated internal 1V regulated supply. When exiting *DEEPSLEEP*, the voltage is automatically switched back to this bit's setting. 0: Dedicated internal 1V regulated supply. 1: V_{CORE} supply.
20	ibro_en	R/W	0	**IBRO Enable** 0: Disabled. 1: Enabled and ready when *GCR_CLKCTRL.ibro_rdy* = 1.
19	ipo_en	R/W	1	**IPO Enable** 0: Disabled. 1: Enabled and ready when *GCR_CLKCTRL.ipo_rdy* = 1.
18:17	-	DNM	0	**Reserved. Do Not Modify.**
16	erfo_en	R/W	0	**ERFO Enable** 0: Disabled. 1: Enabled and ready when *GCR_CLKCTRL.erfo_rdy* = 1.
15:14	ipo_div	R/W	0	**IPO Prescaler** Divides the IPO clock before it is selected as SYS_OSC. 0: Divide by 1. 1: Divide by 2. 2: Divide by 4. 3: Divide by 8.
13	sysclk_rdy	R	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 bit is cleared 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	4	**System Oscillator Source Select** Selects the system oscillator (SYS_OSC) source used to generate the system clock (SYS_CLK). Modifying this field clears *GCR_CLKCTRL.sysclk_rdy* immediately. 0: Reserved. 1: Reserved. 2: ERFO. 3: INRO. 4: IPO. 5: IBRO. 6: ERTCO. 7: External Clock P0.28.
8:6	sysclk_div	R/W	0	**System Oscillator Prescaler** Sets the divider for generating SYS_CLK from the selected SYS_OSC. See *Equation 4-1* for details.
5:0	-	RO	0	**Reserved**

Table 4-52: 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 and is not affected by other resets. 0: The clock source is a crystal oscillator, driving the crystal connected between the HFXIN and HFXOUT pins. 1: The clock source is a square wave driven into the HFXIN pin.
19:18	-	RO	0	**Reserved**
17	ibro_pd	DNM	1	**IBRO Power Down** Not Used. Must be set to 1.
16	ipo_pd	DNM	1	**IPO Power Down** Not Used. Must be set to 1.
15:13	-	RO	0	**Reserved**
12	erfo_pd	DNM	1	**ERFO Power Down** Not used. Must be set to 1.
11:10	-	RO	0	**Reserved**
9	aincomp_we	R/W	0	**AINCOMP Wakeup Enable** Set this field to 1 to enable the comparators as a wakeup source. The analog comparators can wake the device from *SLEEP*, *DEEPSLEEP*, or *BACKUP*.
8	lpuart0_we	R/W	0	**LPUART0 Wakeup Enable** Set this field to 1 to enable LPUART0 as a wakeup source. LPUART0 can wake the device from *SLEEP*, *DEEPSLEEP*, or *BACKUP.* 0: Disabled. 1: Enabled.
7	lptmr1_we	R/W	0	**LPTMR1 Wakeup Enable** Set this field to 1 to enable LPTMR1 as a wakeup source. LPTMR1 wakes the device from *SLEEP*, *DEEPSLEEP*, or *BACKUP.* 0: Disabled. 1: Enabled.
6	lptmr0_we	R/W	0	**LPTMR0 Wakeup Enable** Set this field to 1 to enable LPTMR0 as a wakeup source. LPTMR0 wakes the device from *SLEEP* or *DEEPSLEEP*, or *BACKUP*. 0: Disabled. 1: Enabled.
5	rtc_we	R/W	0	**RTC Alarm Wakeup Enable** Set this field to 1 to enable an RTC alarm to wake the device. The RTC alarm wakes the device from *SLEEP*, *DEEPSLEEP*, *BACKUP*, or *STORAGE*. 0: Disabled. 1: Enabled.
4	gpio_we	R/W	0	**GPIO Wakeup Enable** Set this field to 1 to enable all GPIO pins as potential wakeup sources. Any GPIO configured for wake-up can wake the device from *SLEEP*, *DEEPSLEEP*, *BACKUP*, or *STORAGE*. 0: Disabled. 1: Enabled.

Power Management			GCR_PM	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
3	-	RO	0	**Reserved**
2:0	mode	R/W	0	**Operating Mode** 0b000: *ACTIVE.* 0b001: *ACTIVE.* 0b010: *ACTIVE.* 0b100: *BACKUP.* 0b101: *BACKUP.* 0b110: *BACKUP.* 0b011: *SHUTDOWN.* 0b111: *SHUTDOWN.*

Table 4-53: Peripheral Clock Divisor Register

Peripheral Clocks Divisor			GCR_PCLKDIV	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	-	**Reserved**
16	div_clk_out_en	R/W	0	**DIV_CLK_OUT Enable** This field enables the DIV_CLK_OUT signal on P1.2 AF4 based on the frequency selected using the *GCR_PCLKDIV.div_clk_out_ctrl*. 0: Disabled. 1: Enabled. ***CAUTION**: This field must be set to 0 for the 40-pin TQFN package.*
15:14	div_clk_out_ctrl	R/W	0	**Clock Output Frequency Selection** This field selects the frequency of the clock output on the DIV_CLK_OUT alternate function (P1.2 , AF4). The GPIO must be configured for alternate function 4 and *GCR_PCLKDIV.div_clk_out_en* must be set to 1 to output the clock signal. 0: Disabled. 1: $\frac{IBRO}{2}$ 2: $\frac{ERFO}{4}$ 3: $\frac{ERFO}{8}$ *Note: The corresponding clock must be enabled for the output to function.* *Note: This function is only supported on the 56-pin TQFN package.*
13:2	-	RO	0	**Reserved**
1:0	aon_clkdiv	R/W	0	**AoD Clock Divider** This field configures the frequency of the AoD clock. See *Equation 4-4* for details.

Table 4-54: Peripheral Clock Disable Register 0

Peripheral Clock Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	1	**Reserved**

Peripheral Clock Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
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.
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	ctb	RO	1	**Crypto Toolbox 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**

Peripheral Clock Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
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	0	**UART0 Clock Disable** Disabling a clock peripheral 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.
4:2	-	RO	1	**Reserved**
1	gpio1	R/W	1	**GPIO1 Port and Pad Logic Clock Disable** 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 4-55: Memory Clock Control Register

Memory Clock Control			GCR_MEMCTRL	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:14	-	RO	0	**Reserved**

Memory Clock Control			GCR_MEMCTRL	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
13	romls_en	R/W	0	**ROM LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled.
12	icc0ls_en	R/W	0	**ICC LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled.
11	ram3ls_en	R/W	0	***Sysram3* and *Sysram7* LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 3-1 for base address and size information.*
10	ram2ls_en	R/W	0	***Sysram2* and *Sysram6* LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 3-1 for base address and size information.*
9	ram1ls_en	R/W	0	***Sysram1* and *Sysram5* LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 3-1 for base address and size information.*
8	ram0ls_en	R/W	0	***Sysram0* and *Sysram4* LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 3-1 for base address and size information.*
7:5	-	RO	0	**Reserved**
4	ramws_en		1	**System RAM Wait State Enable** 0: No wait state. 1: Wait state enabled.
3	-	RO	0	**Reserved**
2:0	fws	R/W	5	**Flash Wait States** This field sets the number of SYS_OSC wait-state cycles per flash code read access. See the *Flash Wait States* section for additional information. 0-7: Number of flash memory wait states.

Table 4-56: Memory Zeroization Control Register

Memory Zeroization Control			GCR_MEMZ	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	icc0	R/W1O	0	**ICC Cache Data and Tag Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
3	ramcb	R/W1O	0	**System RAM Check Bit Block Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
2	ram2	R/W1O	0	**Sysram2 Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
1	ram1	R/W1O	0	**Sysram1 Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
0	ram0	R/W1O	0	**Sysram0 Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Table 4-57: System Status Flag Register

System Status Flag			GCR_SYSST	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	icelock	R	0	**Arm ICE Lock Status Flag** 0: Arm ICE is enabled (unlocked). 1: Arm ICE is disabled (locked).

Table 4-58: Reset Register 1

Reset 1			GCR_RST1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
25	qdec	R/W1O	0	**Quadrature Decoder Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
24	-	RO	0	**Reserved**
23	i2s	R/W1O	0	**I²S Peripheral Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
22:18	-	RO	0	**Reserved**

Reset 1			GCR_RST1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
17	i2c2	R/W1O	0	**I²C 2 Peripheral Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
16:15	-	RO	0	**Reserved**
14	ac	R/W1O	0	**Auto Calibration Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
13:11	-	RO	-	**Reserved**
10	aes	R/W1O	0	**AES Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
9	-	RO	0	**Reserved**
8	wdt1	R/W1O	0	**Watchdog Timer 1 Peripheral Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
7:1	-	RO	0	**Reserved**
0	i2c1	R/W1O	0	**I²C 1 Peripheral Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Table 4-59: Peripheral Clock Disable Register 1

Peripheral Clock Disable 1			GCR_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	1	**Reserved**
25	qdec	R/W	1	**Quadrature Decoder Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
24	-	R/W	1	**Reserved**
23	i2s	R/W	1	**I²S Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
22	-	RO	1	**Reserved**

Peripheral Clock Disable 1			GCR_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
21	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.
20:16	-	RO	1	**Reserved**
15	aes	R/W	1	**AES 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.
14:12	-	RO	1	**Reserved**
11	icc0	R/W	1	**ICC 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.
10:6	-	RO	1	**Reserved**
5	wdt1	R/W	1	**Watchdog Timer 1 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.
4	wdt0	R/W	1	**Watchdog Timer 0 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.
3	-	RO	1	**Reserved**
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	-	RO	1	**Reserved**

Table 4-60: Event Enable Register

Event Enable			GCR_EVENTEN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**

Event Enable			GCR_EVENTEN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
2	tx	R/W	0	**TXEV On SEV Enable** When set, a SEV (Send Event) instruction causes a TXEV event from the CPU. 0: Disabled. 1: Enabled.
1	rx	R/W	0	**RXEV Event Enable** Set this field to 1 to enable the generation of an RXEV event to wake the CPU from a Wait for Event (WFE) sleep state. See the device data sheet for AF details. Not all alternate functions are available on all packages. 0: Disabled. 1: Enabled.
0	dma	R/W	0	**CPU DMA CTZ Wake-Up Enable** Allows a DMA CTZ event to generate an RXEV to wake the CPU from a low-power mode entered with a WFE instruction. See the device data sheet for AF details. Not all alternate functions are available on all packages. 0: Disabled. 1: Enabled.

Table 4-61: 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 a packed BCD. 0xA1: A1 revision. 0xB1: B1 revision. 0xB2: B2 revision.

Table 4-62: System Status Interrupt Enable Register

System Status Interrupt Enable			GCR_SYSIE	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	*	**Reserved**
0	iceunlock	R/W	0	**Arm ICE Unlocked Interrupt Enable** Generates an interrupt if the *GCR_SYSST.icelock* field is set. 0: Disabled. 1: Enabled.

Table 4-63: ECC Error Detected Register

ECC Correctable Error Detected			GCR_ECCERR	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**

ECC Correctable Error Detected			GCR_ECCERR	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
5	flash1	R/W1C	0	**Flash Bank 1 ECC Error Detected** This field is set to 1 if ECC is enabled and an error is detected in flash bank 1. Write 1 to clear the flag. *Note: An ECC error occurs when reading from blank/erased flash memory. Until the flash is programmed with valid data, the ECC check bits are not set.* 0: Normal operation. 1: Error detected.
4	flash0	R/W1C	0	**Flash Bank 0 ECC Error Detected** This field is set to 1 if ECC is enabled and an error is detected in flash bank 0. Write 1 to clear the flag. *Note: An ECC error occurs when reading from blank/erased flash memory. Until the flash is programmed with valid data, the ECC check bits are not set.* 0: Normal operation. 1: Error detected.
3	icc0	R/W1C	0	**Internal Cache ECC Error Detected** This field is set to 1 if ECC is enabled and an error is detected in the ICC0. Write 1 to clear the flag. 0: Normal operation. 1: Error detected.
2	ram3	R/W1C	0	**Sysram3 ECC Error Detected** This field is set to 1 if ECC is enabled and an error is detected in the RAM bank. Write 1 to clear the flag. 0: Normal operation. 1: Error detected.
1	ram2	R/W1C	0	**Sysram2 ECC Error Detected** This field is set to 1 if ECC is enabled and an error is detected in the RAM bank. Write 1 to clear the flag. 0: Normal operation. 1: Error detected.
0	ram0_1	R/W1C	0	**Sysram0/Sysram1 ECC Error Detected** This field is set to 1 if ECC is enabled and an error is detected in the RAM bank. Write 1 to clear the flag. 0: Normal operation. 1: Error detected.

Table 4-64: ECC Correctable Error Detected Register

ECC Correctable Error Detected			GCR_ECCCED	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**
5	flash1	R/W1C	0	**Flash Bank 1 Correctable ECC Error Detected** When this field is set, it indicates there is a single correctable error in flash bank 0. Write to 1 to clear the flag. 0: No error. 1: Correctable Error.

ECC Correctable Error Detected			GCR_ECCCED	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
4	flash0	R/W1C	0	**Flash Correctable ECC Error Detected** When this field is set, it indicates there is a single correctable error in flash bank 1. Write to 1 to clear the flag. 0: No error. 1: Correctable Error.
3	icc0	R/W1C	0	**ICC Correctable ECC Error Detected** When this field is set, it indicates that there is a single correctable error in the ICC. Write to 1 to clear the flag. 0: No error. 1: Correctable Error.
2	ram3	R/W1C		**Sysram3 Correctable ECC Error Detected** When this field is set, it indicates there is a single correctable error in the RAM bank. Write to 1 to clear the flag. 0: No error. 1: Correctable Error.
1	ram2	R/W1C		**Sysram2 Correctable ECC Error Detected** When this field is set, it indicates there is a single correctable error in the RAM bank. Write to 1 to clear the flag. 0: No error. 1: Correctable Error.
0	ram0_1	R/W1C	0	**Sysram0/Sysram1 Correctable ECC Error Detected** When this field is set, it indicates there is a single correctable error in the RAM bank. Write to 1 to clear the flag. 0: No error. 1: Correctable Error.

Table 4-65: ECC Interrupt Enable Register

ECC Interrupt Enable			GCR_ECCIE	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**
5	flash1		0	**Flash Bank 1 ECC Error Interrupt Enable** 0: Disabled. 1: Enabled.
4	flash0	R/W	0	**Flash ECC Error Interrupt Enable** 0: Disabled. 1: Enabled.
3	icc0	R/W	0	**ICC ECC Error Interrupt Enable** 0: Disabled. 1: Enabled.
2	ram3	R/W	0	**Sysram3 RAM ECC Error Interrupt Enable** 0: Disabled. 1: Enabled.
1	ram2	R/W	0	**Sysram2 RAM ECC Error Interrupt Enable** 0: Disabled. 1: Enabled.
0	ram0_1	R/W	0	**Sysram0/Sysram1 ECC Error Interrupt Enable** 0: Disabled. 1: Enabled.

Table 4-66: ECC Address Register

ECC Address			GCR_ECCADDR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**
31	tagramerr	R	0	**ECC Error Address/TAG RAM Error** Data depends on which block has reported the error. If *sysram* or flash, then this field represents the bit(s) of the AMBA address of the read that produced the error. If the error is from the cache, then this bit is set as shown below: 0: No error 1: Tag_Error. The error is in the TAG RAM.
30	tagrambank	R	0	**ECC Error Address/TAG RAM Error Bank** Data depends on which block has reported the error. If *sysram* or flash, then this field represents the bit(s) of the AMBA address of the read that produced the error. If the error is from the cache, then this bit is set as shown below: 0: Error is in TAG RAM Bank 0 1: Error is in TAG RAM Bank 1
29:16	tagramaddr	R	0	**ECC Error Address/TAG RAM Error Address** Data depends on which block has reported the error. If *sysram* or flash, this field represents the bit(s) of the AMBA address the read that produced the error. If the error is from the cache, then this bit is set as shown below: [TAG ADDRESS]: Represents the TAG RAM address
15	dataramerr	R	0	**ECC Error Address/DATA RAM Error Address** Data depends on which block has reported the error. If *sysram* or flash, this field represents the bit(s) of the AMBA address of the read that produced the error. If the error is from the cache, then this bit is set as shown below: 0: No error 1: Cache data RAM error. The error is in the cache's data RAM.
14	datarambank	R	0	**ECC Error Address/DATA RAM Error Bank** Data depends on which block has reported the error. If *sysram* or flash, this field represents the bit(s) of the AMBA address of the read that produced the error. If the error is from the cache, then this bit is set as shown below: 0: Error is in cache data RAM bank 0 1: Error is in cache data RAM bank 1
13:0	dataramaddr	R	0	**ECC Error Address/TAG RAM Error Address** Data depends on which block has reported the error. If *sysram* or flash, this bit(s) represents the bit(s) of the AMBA address read, which produced the error. If the error is from the cache, then this bit is set as shown below: [DATA ADDRESS]: Represents the cache's data RAM error address.

4.13 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 4-67: System Initialization Register Summary

Offset	Register	Description
[0x0000]	*SIR_STATUS*	*System Initialization Error Status Register*
[0x0004]	*SIR_ADDR*	*System Initialization Error Address Register*

4.13.1 Register Details

Table 4-68: System Initialization Error Status Register

System Initialization Error Status			SIR_STATUS	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	user_cfg_err	RO	*	**System AES Keys Valid** This field reads 1 if the system AES keys are programmed in the information block using the correct flash magic value. See *System AES Key Storage* for details.
1	cfg_err	RO	*	**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.*
0	cfg_valid	RO	*	**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.*

Table 4-69: System Initialization Error Address Register

System Initialization Error Address			SIR_ADDR	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	RO	0	**Configuration Error Address** If the *SIR_STATUS.cfg_err* field is set to 1, the value in this register is the address of the configuration failure.

Table 4-70: Function Status Register

Function Status			SIR_FSTAT	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ds_ack	R	*	**DEEPSLEEP ACK** This field reads 1 if DEEPSLEEP is supported in the device.
14	trng	R	*	**TRNG** This field reads 1 if the TRNG is supported in the device.
13:1	-	RO	0	**Reserved**
0	fpu	R	*	**FPU** This field reads 1 if the FPU is supported in the device.

4.14 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 4-71: Function Control Register Summary

Offset	Register	Description
[0x0000]	*FCR_FCTRL0*	Function Control Register 0
[0x0004]	*FCR_AUTOCAL0*	Automatic Calibration 0 Register
[0x0008]	*FCR_AUTOCAL1*	Automatic Calibration 1 Register
[0x000C]	*FCR_AUTOCAL2*	Automatic Calibration 2 Register
[0x0010]	*FCR_TS0*	Temperature Sensor Gain Trim Register
[0x0014]	*FCR_TS1*	Temperature Sensor Offset Trim Register
[0x0018]	*FCR_ADCREFTRIM0*	ADC 1.25V Reference Trim Register
[0x001C]	*FCR_ADCREFTRIM1*	ADC 2.048V Reference Trim Register
[0x0020]	*FCR_ADCREFTRIM2*	ADC External Reference Trim Register
[0x0024]	*FCR_ERFOKS*	ERFO Kick Start

4.14.1 Register Details

Table 4-72: Function Control 0 Register

Function Control 0			FCR_FCTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
25	i2c2_scl_filter_en	R/W	0	**I²C2 SCL Glitch Filter Enable** 0: Disabled. 1: Enabled.
24	i2c2_sda_filter_en	R/W	0	**I²C2 SDA Glitch Filter Enable** 0: Disabled. 1: Enabled.
23	i2c1_scl_filter_en	R/W	0	**I²C1 SCL Glitch Filter Enable** 0: Disabled. 1: Enabled.
22	i2c1_sda_filter_en	R/W	0	**I²C1 SDA Glitch Filter Enable** 0: Disabled. 1: Enabled.
21	i2c0_scl_filter_en	R/W	0	**I²C0 SCL Glitch Filter Enable** 0: Disabled. 1: Enabled.
20	i2c0_sda_filter_en	R/W	0	**I²C0 SDA Glitch Filter Enable** 0: Disabled. 1: Enabled.
19:9	-	RO	0	**Reserved**

Function Control 0			FCR_FCTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
8	keywipe_sys	R/W	0	**System AES Key Wipe** Set this field to 1 to wipe the system AES key registers (*SYS_AESKEYS_KEY7*: *SYS_AESKEYS_KEY0*). This field is automatically cleared to 0 by hardware when the operation is complete. 0: Normal operation. 1: Wipe system AES key registers.
7:3	-	RO	0	**Reserved**
2:0	erfo_range_sel	R/W	0	**ERFO Frequency Range Select** Set these bits to reflect the crystal frequency connected to the HFXOUT and HFXIN device pins. 0: < 22.5MHz 1: 22.5MHz to 24.5MHz 2: 24.5MHz to 26.3MHz 3: 26.3MHz to 28.0MHz 4: 28.0MHz to 29.6MHz 5: 29.6MHz to 31.1MHz 6: 31.1MHz to 32.6MHz 7: Reserved

Table 4-73: Automatic Calibration 0 Register

Function Control 1			FCR_AUTOCAL0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:23	trim	R	0	**IPO Trim Value**
22:20	-	RO	0	**Reserved**
19:8	gain	R/W	0	**IPO Trim Adaptation Gain**
7:5	-	RO	0	**Reserved**
4	atomic	R/W1O	0	**IPO Trim Atomic Start** Set this bit to start an atomic calibration of the IPO. The calibration runs for *FCR_AUTOCAL2.runtime* milliseconds. This bit is cleared by hardware once the calibration is complete.
3	invert	RO	0	**IPO Trim Step Invert**
2	load	R/W1O	0	**IPO Initial Trim Load** Set this bit to load the initial trim value for the IPO from *FCR_AUTOCAL1.initial*. This bit is cleared by hardware once the load is complete.
1	en	R/W	0	**IPO Automatic Calibration Continuous Mode Enable** 0: Disabled. 1: Enabled.
0	sel	R/W	0	**IPO Trim Select** 0: Use default trim. 1: Use automatic calibration trim values.

Table 4-74: Automatic Calibration 1 Register

Function Control 2			FCR_AUTOCAL1	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	initial	R/W	0	**IPO Automatic Calibration Initial Trim**

Table 4-75: Automatic Calibration 2 Register

Function Control 3			FCR_AUTOCAL2	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0	**Reserved**
20:8	div	R/W	0	**IPO Trim Automatic Calibration Divide Factor** Target trim frequency for the IPO: $f_{IPO} = \text{div} \times 32768$ *Note: Setting div to 0 is equivalent to setting div to 1.*
7:0	runtime	R/W	0	**IPO Trim Automatic Calibration Run Time** $Atomic\ Run\ Time = runtime\ milliseconds$

Table 4-76: Temperature Sensor Gain Register

Temperature Sensor Gain			FCR_TS0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	R	0	**Reserved**
11:0	gain	R	*	**Temperature Sensor Gain** This field contains the unsigned gain for the temperature sensor normalization. See *Temperature Sensor* for details.

Table 4-77: Temperature Sensor Offset Register

Temperature Sensor Offset			FCR_TS1	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:14	ts_offset_sign	R	*	**Sign Extension for Offset**
13:0	offset	R	*	**Temperature Sensor Offset** This field contains the signed offset for the temperature sensor normalization. See *Temperature Sensor* for details.

Table 4-78: ADC 1.25V Reference Trim Register

ADC 1.25V Reference Trim			FCR_ADCREFTRIM0	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	R	0	**Reserved**
29:24	vx2_tune		*	**Tuning Capacitor In-Line DAC** See *1.25V Internal Reference Trim for All Other Revisions* 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 All Other Revisions* for details on this field.
15	-	RO	0	**Reserved**

ADC 1.25V Reference Trim			FCR_ADCREFTRIM0	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
14:8	vrefm	R	*	**Trim Code for V_{REFM} Output of Reference Buffer** See *1.25V Internal Reference Trim for All Other Revisions* 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 All Other Revisions* for details on this field.

Table 4-79: ADC 2.048V Reference Trim Register

ADC 2.048V Reference Trim			FCR_ADCREFTRIM1	[0x001C]
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 All Other Revisions* 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 All Other Revisions* 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 All Other Revisions* 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 All Other Revisions* for details on this field.

Table 4-80: ADC External Reference Trim Register

ADC External Reference Trim			FCR_ADCREFTRIM2	[0x0020]
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 All Other Device Revisions* 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 All Other Device Revisions* 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 All Other Revisions* 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 All Other Revisions* 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 All Other Revisions* for details on this field.

ADC External Reference Trim			FCR_ADCREFTRIM2	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
3:0	idrv_1p25	R	*	**Trim Code for 1.25V Reference Buffer Drive Strength** See *1.25V Internal Reference Trim for All Other Revisions* for details on this field.

Table 4-81: ERFO Kick Start Register

ERFO Kick Start			FCR_ERFOKS	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15:0	ctrl	R/W	0	**ERFO Kick Start Control**

5. Interrupts and Exceptions

Interrupts and exceptions are managed by the Nested Vector Interrupt Controller (NVIC). The NVIC handles the interrupts, exceptions, priorities, and masking. *Table 5-1* details the MAX32672 interrupt vector table and describes each exception and interrupt.

5.1 Features

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

5.2 Interrupt Vector Table

Table 5-1 lists the interrupt and exception table for the MAX32672. There are 123 interrupt entries for the MAX32672, including reserved for future use interrupt place holders. Including the 15 system exceptions for the Arm Cortex-M4 with FPU, the total number of entries is 138.

Table 5-1: MAX32672 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]	-	Reserved
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	Timer 4 (Low Power Timer 0) Interrupt
26	[0x0068]	TMR5_IRQn	Timer 5 (Low Power Timer 1) Interrupt
27:28	[0x006C]:[0x0070]	-	Reserved
29	[0x0074]	I2C0_IRQn	I^2C Port 0 Interrupt

Exception/Interrupt Number	Offset	Name	Description
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
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]	-	Reserved
43	[0x00AC]	CRYPTO_IRQn	Crypto Interrupt
44	[0x00B0]	DMA0_IRQn	DMA Channel 0 Interrupt
45	[0x00B4]	DMA1_IRQn	DMA Channel 1 Interrupt
46	[0x00B8]	DMA2_IRQn	DMA Channel 2 Interrupt
47	[0x00BC]	DMA3_IRQn	DMA Channel 3 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
53:69	[0x00D4]:[0x0114]	-	Reserved
70	[0x0118]	GPIOWAKE_IRQn	GPIO Wakeup Interrupt
71:72	[0x011C]:[0x0120]	-	Reserved
73	[0x0124]	WDT1_IRQn	Windowed Watchdog Timer 1 Interrupt
74:77	[0x0128]:[0x0134]	-	Reserved
78	[0x0138]	I2C2_IRQn	I^2C Port 2 Interrupt
79:83	[0x013C]:[0x014C]	-	Reserved
84	[0x0150]	DMA4_IRQn	DMA Channel 4 Interrupt
85	[0x0154]	DMA5_IRQn	DMA Channel 5 Interrupt
86	[0x0158]	DMA6_IRQn	DMA Channel 6 Interrupt
87	[0x015C]	DMA7_IRQn	DMA Channel 7 Interrupt
88	[0x0160]	DMA8_IRQn	DMA Channel 8 Interrupt
89	[0x0164]	DMA9_IRQn	DMA Channel 9 Interrupt
90	[0x0168]	DMA10_IRQn	DMA Channel 10 Interrupt
91	[0x016C]	DMA11_IRQn	DMA Channel 11 Interrupt
92:97	[0x0170]:[0x0184]	-	Reserved
98	[0x0188]	ECC_IRQn	Error Correction Coding Block Interrupt
99:100	[0x018C]:[0x0190]	-	Reserved
101	[0x0194]	SCA_IRQn	Crypto Accelerator Interrupt

Exception/Interrupt Number	Offset	Name	Description
102	[0x0198]	-	Reserved
103	[0x019C]	FLC1_IRQn	Flash Controller 1 Interrupt
104	[0x01A0}	UART3_IRQn	UART3 (Low Power UART0) Interrupt
105:112	[0x01A4]:[0x01C0]	-	Reserved
113	[0x01C4]	AES_IRQn	AES Block Interrupt
114	[0x01C8]	-	Reserved
115	[0x01CC]	I2S_IRQn	I^2S Interrupt
116:121	[0x01D0]:[0x01E4]	-	Reserved
122	[0x01E8]	QDEC_IRQn	Quadrature Decoder Interrupt
123	[0x01EC]	-	Reserved

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

The GPIO pins share an individually controlled I/O mode and an AF mode. Configuring a pin for an AF supersedes its use as a controlled GPIO. However, the input data is always readable using the GPIO input register, *GPIOn_IN*, if the GPIO input is enabled.

Multiplexing between the AF and the I/O function is often static in an application, set at initialization, and dedicated as either an AF or GPIO. The software must manage dynamic multiplexing between AF1, AF2, AF3, AF4, and I/O mode. The software must manage the AF and GPIO to ensure each is set up properly when switching from a peripheral to the I/O function. Refer to the device data sheet electrical characteristics table for information on the GPIO pin behavior based on the configurations described in this document.

In GPIO mode, each I/O pin supports interrupt functionality that can be independently enabled and configured as a level triggered interrupt, a rising edge, a falling edge, or both rising and falling edge interrupt. All GPIO on the same 32-bit GPIO port share the same interrupt vector. Not all GPIO pins are available on all packages.

Note: The register set used to control the GPIO are identical across multiple Analog Devices microcontrollers; however, the behavior of several registers varies 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

The GPIO are all bidirectional digital I/O that include:

- Input Mode Features:
  - Standard CMOS or Schmitt hysteresis.
  - Input data from the input data register (*GPIOn_IN*) or to a peripheral (AF).
  - Input state selectable for floating (tri-state) or weak pullup/pulldown.
- Output Mode Features:
  - Output data from the output data register (*GPIOn_OUT*) in GPIO mode.
  - Output data driven from the peripheral if an AF is selected.
  - Standard GPIO:
    - Four drive strength modes.
    - Slow or fast slew rate selection.
- Selectable weak pullup resistor, weak pulldown resistor, or tri-state mode for standard GPIO pins.
- Selectable weak pulldown or tri-state mode for GPIO pins with I^2C as an AF.
- Wake from low-power modes on a rising edge, falling edge, or both on the I/O pins.

6.1 Instances

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

Table 6-1: GPIO Pin Count

Package	GPIO	Pins
40-TQFN	GPIO0[19:0] GPIO0[29:22]	Refer to the device data sheet for details.
56-TQFN	GPIO0[31:0] GPIO1[9:0]	Refer to the device data sheet for details.

Note: Refer to the device data sheet for a description of the AF(s) for each GPIO port pin.

Note: MAX32672 does not support the selectable GPIO voltage supply feature.

6.2 Configuration

6.2.1 Peripheral Clock Enable

Each GPIO port is disabled by default on a reset. Using a GPIO pin requires enabling the peripheral clock for the port. Enable GPIO0 by setting *GCR_PCLKDIS0.gpio0* to 0 and enable GPIO1 by setting *GCR_PCLKDIS0.gpio1* to 0.

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 *Secure Communication Protocol Bootloader (SCPBL)* for exceptions.

Following a POR event, all GPIO, except device pins that have the SWDIO and SWDCLK functions, are floating and 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.
- Input mode disabled:
  - *GPIOn_INEN.[pin]* = 0.
- Interrupt disabled:
  - *GPIOn_INTEN.[pin]* = 0.

6.2.3 Serial Wire Debug Configuration

The SWD pins are configured for SWD on POR. Perform the following steps to reconfigure the SWDIO and SWDCLK device pins for SWD mode:

1. Set the device pin P0.0 for AF1 mode:
   a. GPIO0_EN0.[0] = 0.
   b. GPIO0_EN1.[0] = 0.
   c. GPIO0_EN2.[0] = 0.
   d. *GPIOn_PADCTRL0.[pin]* = 1.
   2. e. *GPIOn_PADCTRL1.[pin]* = 0.
2. Set device pin P0.1 for AF1 mode:
   a. GPIO0_EN0.[1] = 0.
   b. GPIO0_EN1.[1] = 0.
   c. GPIO0_EN2.[1] = 0.
   d. *GPIOn_PADCTRL0.[pin]* = 1.
   e. *GPIOn_PADCTRL1.[pin]* = 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 Alternate Function Configuration

Table 6-3 shows the AF selection matrix. Write the *GPIOn_EN0* and *GPIOn_EN1* fields as shown in the table to select the desired AF. Each AF is independently selectable. Mixing functions assigned to AF1, AF2, AF3, or AF4 is supported if all the peripheral's required functions are enabled.

Table 6-2: GPIO Mode and AF Selection

GPIO Mode	*GPIOn_EN2.[pin]*	*GPIOn_EN1.[pin]*	*GPIOn_EN0.[pin]*
I/O	0	0	1
AF1	0	0	0
AF2	0	1	0
AF3	1	0	0
AF4	1	1	0

Table 6-3: GPIO Mode and AF Transition Selection

GPIO Mode	*GPIOn_EN2.[pin]*	*GPIOn_EN1.[pin]*	*GPIOn_EN0.[pin]*
I/O (transition to AF1)	0	0	1
I/O (transition to AF2)	0	1	1
I/O (transition to AF3)	1	0	1
I/O (transition to AF4)	1	1	1

Most GPIO support one or more alternate functions that are selected with the GPIO configuration enable bits shown in *Table 6-3*. The bits that select the AF must only be changed while the pin is in one of the I/O modes (*GPIOn_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 6-3* 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 6-5* if the assigned alternate function uses the pin as an input. See *Table 6-6* if the assigned alternate function uses the pin as an output.
3. Set the GPIO configuration enable bits shown in *Table 6-2* to the desired alternate function.

Table 6-4: GPIO AF Configuration Reference

Device Pin	AF Configuration Bits		
P0.0	*GPIOn_EN2.[0]*	*GPIOn_EN1.[0]*	*GPIOn_EN0.[0]*
P0.1	*GPIOn_EN2.[1]*	*GPIOn_EN1.[1]*	*GPIOn_EN0.[1]*
…	…	…	…
P0.29	*GPIOn_EN2.[29]*	*GPIOn_EN1.[29]*	*GPIOn_EN0.[29]*
P0.30	*GPIOn_EN2.[30]*	*GPIOn_EN1.[30]*	*GPIOn_EN0.[30]*

6.2.5 Input Mode configuration

Perform the following steps to configure a pin or pins for input mode:

1. Set the pin for GPIO mode:
   a. *GPIOn_EN0.[pin]* = 1.
   b. *GPIOn_EN1.[pin]* = 0.
   c. *GPIOn_EN2.[pin]* = 0.
2. Configure the pin for pullup, pulldown, or high-impedance mode. See *Table 6-5* for configuration details.
   a. GPIO pins with I^2C as an AF only support high-impedance or a weak pulldown resistor.
3. Enable the pin's input:
   a. Set *GPIOn_INEN.[pin]* = 1.
4. Read the input state of the pin using the *GPIOn_IN.[pin]* field.

A summary of the configuration of the input mode is shown in *Table 6-5*.

Table 6-5: MAX32672 Input Mode Configuration Summary

Input Mode	Mode Select	
	GPIOn_PADCTRL1.[pin]	***GPIOn_PADCTRL0.[pin]***
High-impedance	0	0
Weak Pullup to V_{DD}	0	1
Weak Pulldown to V_{SS}	1	0
Reserved	1	1

Note: Refer to the device data sheet electrical characteristics table for the value of resistors.

6.2.6 Output Mode Configuration

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

1. Set the pin for GPIO mode:
   a. *GPIOn_EN0*.*[pin]* = 1.
   b. *GPIOn_EN1*.*[pin]* = 0.
   c. *GPIOn_EN2*.*[pin]* = 0.
2. Enable the output buffer for the pin by setting *GPIOn_OUTEN*.*[pin]* to 1.
3. Set the output drive strength using the *GPIOn_DS1*.*[pin]* and *GPIOn_DS0*.*[pin]* bits.
   a. See the *GPIO Drive Strength* section for configuration details and the modes supported.
4. Set the output high or low using the *GPIOn_OUT*.*[pin]* bit.

6.2.6.1 GPIO Drive Strength

Each I/O pin supports multiple selections for drive strength. Standard GPIO pins are configured for the supported modes using the *GPIOn_DS1* and *GPIOn_DS0* registers, as shown in *Table 6-6*. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO0_DS.str[25], GPIO0_DS1.str[25]* both represent configuration for device pin P0.25. The drive strength currents shown are targets only. Refer to the device Data Sheet Electrical Characteristics table for details of the V_{OL_GPIO}, V_{OH_GPIO}, V_{OL_I2C}, and V_{OH_I2C} parameters.

Table 6-6: Standard GPIO Drive Strength Selection

Drive Strength V_{DD} = 1.71V	***GPIOn_DS1*.*[pin]***	***GPIOn_DS0*.*[pin]***
1mA	0	0
2mA	1	0
4mA	0	1
6mA	1	1

For GPIO with I^2C as an AF, *Table 6-7* shows the drive strength setting options.

Table 6-7: GPIO with I^2C AF Drive Strength Selection

Drive Strength V_{DD} = 1.71V	***GPIOn_DS0*.*[pin]***
2mA	0
10mA	1

6.3 Configuring GPIO (External) Interrupts

Each GPIO 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 AF, the interrupts are peripheral controlled.

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

The following procedure details the steps for enabling *ACTIVE* and *SLEEP* 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 GPIO by writing 0 to the *GPIOn_INTEN* register. To maintain previously enabled interrupts, read the *GPIOn_INTEN* register and save the value to memory before setting the register to 0.
2. Clear pending interrupts by writing 1 to the *GPIOn_INTFL_CLR.[pin]* bit.
3. Set *GPIOn_INTMODE.[pin]* to select either level (0) or edge triggered (1) interrupts.
   a. For level-triggered interrupts, the interrupt triggers on an input high or low.
      i. *GPIOn_INTPOL.[pin]* = 1: Input high triggers interrupt.
      3. ii. *GPIOn_INTPOL.[pin]* = 0: Input low triggers interrupt.
   b. For edge-triggered interrupts, the interrupt triggers on an edge event.
      i. *GPIOn_INTPOL.[pin]* = 1: Input falling edge triggers interrupt.
      ii. *GPIOn_INTPOL.[pin]* = 0: Input rising edge triggers interrupt.
      iii. Optionally set *GPIOn_DUALEDGE[pin]* to 1 to trigger on both the rising and falling edges of the input signal.
4. Set *GPIOn_WKEN.[pin]* to 1 to enable wake from *SLEEP* and *DEEPSLEEP* if desired.
5. Set *GPIOn_INTEN.[pin]* to 1 to enable the interrupt for the pin.

6.3.1 GPIO Interrupt Handling

Each GPIO port is assigned a dedicated interrupt vector, as shown in *Table 6-9*. Complete the following steps to handle GPIO interrupts using a software interrupt vector handler:

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 as required by the application.
3. Clear the interrupt flag in the *GPIOn_INTFL* register by writing 1 to the *GPIOn_INTFL_CLR* bit position that triggered the interrupt. If multiple bits are set in the *GPIOn_INTFL* register, all of the bits should be cleared.
4. Return from the interrupt vector handler.

Table 6-8 shows the registers and interrupt handler for standard GPIO interrupts for each supported operating mode.

Table 6-8: MAX32672 GPIO Interrupt Enable Settings for Each Supported Operating Mode

Operating Mode	*GPIOn_INTEN*	Interrupt Handler
ACTIVE	X	GPIOn_IRQn
SLEEP	X	GPIOn_IRQn
DEEPSLEEP	X	GPIOn_IRQn
Note: Wake from DEEPSLEEP, BACKUP, and STORAGE is only supported using the GPIOWAKE interrupt.		

Table 6-9: MAX32672 GPIO Port Interrupt Vector Mapping

GPIO Port	GPIO Interrupt Status Register	Interrupt Vector Number	GPIO Interrupt Vector
GPIO0	*GPIOn_INTFL*	40	GPIO0_IRQn
GPIO1	*GPIOn_INTFL*	41	GPIO1_IRQn

6.3.2 Using GPIO for Wake Up from Low Power Modes

Standard GPIO interrupts wake the device from *SLEEP*. Additionally, wake from *DEEPSLEEP*, *BACKUP, and STORAGE* is supported for GPIO using the GPIOWAKE feature. GPIOWAKE allows wake from low-power modes from external edge-triggered interrupts (rising and falling) on the GPIO ports. Level-triggered interrupts are not supported for wake up because the system clock must be active to level detection.

6.3.3 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 on both GPIO0 and GPIO1 ports. When the wake-up event occurs, the application software must interrogate the *PWRSEQ_LPWKST0* and *PWRSEQ_LPWKST1* registers to determine which GPIO port pin caused the interrupt.

Enable GPIOWAKE interrupts for all low-power modes from an external GPIO event by completing the following steps:

1. Clear pending interrupt flags by writing 0xFFFF FFFF to the low-power wake-up status registers (*PWRSEQ_LPWKST0* and *PWRSEQ_LPWKST1).*
2. Set up a GPIOWAKE_IRQn interrupt handler.
3. Enable the GPIOWAKE for each desired pin by setting *PWRSEQ_LPWKEN0.[pin]* and *PWRSEQ_LPWKEN1.[pin]* to 1.
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* register for GPIO0 or *PWRSEQ_LPWKST1* register for GPIO1.

Table 6-10: GPIO Wakeup Interrupt Vector

GPIO Wake Interrupt Source	Device Specific Interrupt Vector Number	GPIO Wake-up Interrupt Vector
GPIO0[31:0] GPIO1[9:0]	70	GPIOWAKE_IRQn

6.4 GPIO 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 6-11*. 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 6-11: GPIO Register Summary

Offset	Register Name	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*

Offset	Register Name	Description
[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 Wakeup Enable Register
[0x0050]	GPIOn_WKEN_SET	GPIO Port n Wakeup Enable Atomic Set Register
[0x0054]	GPIOn_WKEN_CLR	GPIO Port n Wakeup Enable Atomic Clear Register
[0x005C]	GPIOn_DUALEDGE	GPIO Port n Interrupt Dual Edge Mode Register
[0x0060]	GPIOn_PADCTRL0	GPIO Port n Pad Control 0 Register
[0x0064]	GPIOn_PADCTRL1	GPIO Port n Pad Control 1 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 Input Hysteresis Enable Register
[0x00AC]	GPIOn_SRSEL	GPIO Port n Slew Rate Select Register
[0x00B0]	GPIOn_DS0	GPIO Port n Drive Strength Select 0 Register
[0x00B4]	GPIOn_DS1	GPIO Port n Drive Strength Select 1 Register
[0x00B8]	GPIOn_PS	GPIO Port n Pullup/Pulldown Enable Register
[0x00C0]	GPIOn_VSSEL	GPIO Port n Voltage Select Register

6.4.1 Register Details

Table 6-12: GPIO AF 0 Select Register

GPIO AF 0 Select			GPIOn_EN0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	*	**GPIO Configuration Enable Bit 0** These bits, in conjunction with bits in *Table 6-2*, configure the corresponding device pin for digital I/O or an AF 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 the input and interrupt functionality of the associated pin.* * GPIO0_EN0 reset value is 0xFFFF FFFF. * GPIO1_EN0 reset value is 0x0000 03FF.

Table 6-13: 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** Setting a bit in this field sets the corresponding bit in the *GPIOn_EN0* register. 0: No effect 1: Corresponding bit in *GPIOn_EN0* register set to 1 *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-14: 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** Setting a bit in this field clears the corresponding bits in the *GPIOn_EN0* register. 0: No effect 1: Corresponding bits in *GPIOn_EN0* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-15: 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** Setting a bit in this field enables 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: Disabled 1: Enabled *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-16: 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** Setting a bit in this field sets the corresponding bit in the *GPIOn_OUTEN* register. 0: No effect 1: Corresponding bits in *GPIOn_OUTEN* set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-17: 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** Setting a bit in this field sets the corresponding bits in the *GPIOn_OUTEN* register. 0: No effect 1: Corresponding bits in *GPIOn_OUTEN* cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-18: GPIO Port n Output Register

GPIO Port n Output			GPIOn_OUT	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Output Level** Setting a bit in this field sets the corresponding output pin to a high state. Clearing a bit in this field clears the corresponding output pin to a low state. 0: Drive the corresponding output pin low (logic 0) 1: Drive the corresponding output pin high (logic 1) *Note: This bit is ignored if the corresponding bit position in the GPIOn_OUTEN register is not set or if the pin is configured for an AF.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-19: 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** Setting a bit in this field sets the corresponding bits in the *GPIOn_OUT* register. 0: No effect 1: Corresponding bits in *GPIOn_OUTEN* set to 1 *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-20: GPIO Port n Output Atomic Clear Register

GPIO Port n Output Atomic Clear			GPIOn_OUT_CLR	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Output Atomic Clear** Setting a bit in this field clears the corresponding bits in the *GPIOn_OUT* register. 0: No effect 1: Corresponding bits in *GPIOn_OUTEN* cleared to 0 *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-21: GPIO Port n Input Register

GPIO Port n Input			GPIOn_IN	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R	-	**GPIO Input Level** Read the state of the corresponding input pin. The input state is always readable for a pin regardless of the pin's configuration as an output or AF. 0: Input pin low (logic 0) 1: Input pin high (logic 1) *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-22: 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** Setting a bit in this field sets edge-triggered interrupts for corresponding GPIO pin. Clearing a bit in this field sets level-triggered interrupt for corresponding GPIO pin. 0: Level triggered interrupt. 1: Edge triggered interrupt. *Note: This bit has no effect unless the corresponding bit in the GPIOn_INEN register is set.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-23: 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.[pin]*= 0): 0: Input low (logic 0) triggers interrupt. 1: Input high (logic 1) triggers interrupt. Edge triggered mode (*GPIOn_INTMODE.[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_INEN register is set.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-24: GPIO Port n Input Enable Register

GPIO Port n Input Enable			GPIOn_INEN	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	*	**GPIO Input Enable** Setting a bit in this field 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. * GPIO0_INEN reset default is 0x0000 0003. * GPIO1_INEN reset default is 0x0000 0000. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-25: GPIO Port n Interrupt Enable Registers

GPIO Port n Interrupt Enable			GPIOn_INTEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Interrupt Enable** Setting a bit in this field enables the interrupt for the corresponding GPIO pin. 0: Disabled 1: 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.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-26: 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** Setting a bit in this field sets the corresponding bits in the *GPIOn_INTEN* register. 0: No effect. 1: Corresponding bits in *GPIOn_INTEN* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-27: 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** Setting a bit in this field clears the corresponding bits in the *GPIOn_INTEN* register. 0: No effect. 1: Corresponding bits in *GPIOn_INTEN* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-28: GPIO Interrupt Status Register

GPIO Interrupt Status			GPIOn_INTFL	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R	0	**GPIO Interrupt Status** An interrupt is pending for the associated GPIO pin when this bit reads 1. 0: No interrupt pending for the associated GPIO pin. 1: GPIO interrupt pending for the associated GPIO pin. *Note: Write a 1 to the corresponding bit in the GPIOn_INTFL_CLR register to clear the interrupt pending status flag.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-29: 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** Setting a bit in this field clears 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. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-30: GPIO Port n Wakeup Enable Register

GPIO Port n Wakeup Enable			GPIOn_WKEN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved, Do Not Modify**

Table 6-31: GPIO Port n Wakeup Enable Atomic Set Register

GPIO Port Wakeup Enable Atomic Set			GPIOn_WKEN_SET	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved, Do Not Modify**

Table 6-32: GPIO Port n Wakeup Enable Atomic Clear Register

GPIO Port Wakeup Enable Atomic Clear			GPIOn_WKEN_CLR	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved, Do Not Modify**

Table 6-33: 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 a bit in this field selects dual-edge mode triggered interrupts (rising and falling edge triggered) on the corresponding GPIO port device pin. The associated *GPIOn_INTMODE* bit must be set to edge-triggered. When enabled, the associated polarity (*GPIOn_INTPOL*) setting has no effect. 0: Disabled. 1: Enabled. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-34: GPIO Port n Pad Control 0 Register

GPIO Port n Pad Control 0			GPIOn_PADCTRL0	[0x0060]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	*	**GPIO Pad Configuration 0** 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 *Input Mode configuration*. * GPIO0_PADCTRL0 reset value is 0x0000 0003. * GPIO1_PADCTRL1 reset value is 0x0000 0000.

Table 6-35: GPIO Port n Pad Control 1 Register

GPIO Port n Pad Control 1			GPIOn_PADCTRL1	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	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 *Input Mode configuration*.

Table 6-36: 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 AF 1 Mode Select** These bits, in conjunction with bits in *Table 6-2*, configure the corresponding device pin for digital I/O or an AF 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 the input and the interrupt functionality of the associated pin.*

Table 6-37: 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** Setting a bit in this field sets the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-38: 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** Setting a bit in this field clears the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-39: 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 6-2*, configure the corresponding device pin for digital I/O or an AF 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 the input and the interrupt functionality of the associated pin.*

Table 6-40: 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 AF Select Atomic Set Bit 2** Setting a bit in this field sets the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-41: 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 AF Select Atomic Clear Bit 2** Setting a bit in this field clears the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-42: GPIO Port n Input Hysteresis Enable Register

GPIO Port n Input Hysteresis Enable		GPIOn_HYSEN		[0x00A8]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Input Hysteresis Enable** Setting a bit in this field enables a Schmitt input to introduce hysteresis for better noise immunity on the corresponding GPIO port device pin. 0: Disabled 1: Enabled *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-43: GPIO Port n Slew Rate Enable Register

GPIO Port n Slew Rate Select		GPIOn_SRSEL		[0x00AC]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Slew Rate Mode** Setting a bit in this field enables the slow slew rate for the corresponding GPIO port device pin. Clearing a bit in this field enables fast slew rate for the corresponding GPIO port device pin. 0: Fast slew rate selected. 1: Slow slew rate selected. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: GPIO with I^2C as an AF do not support Slew Rate Select. Refer to the symbols V_{OL_I2C} and V_{OH_I2C} in the device data sheet electrical characteristics table for details regarding which I/O pins support I^2C functionality.*

Table 6-44: 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 Drive Strength 0 Select** The output drive strength supports four modes. The mode selection is set using the combination of the *GPIOn_DS1* and *GPIOn_DS0* bits for the associated GPIO pin. See the *GPIO Drive Strength* section for the selection options on these I/O pins. *Note: GPIO with I^2C as an AF only support two different drive strengths:* *0: Low output drive strength selected.* *1: High output drive strength selected.* *Refer to the symbols V_{OL_I2C} and V_{OH_I2C} in the device data sheet electrical characteristics table for details regarding which I/O pins support I^2C functionality.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-45: 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 Drive Strength 1 Select** The output drive strength supports four modes. The mode selection is set using the combination of the *GPIOn_DS1* and *GPIOn_DS0* bits for the associated GPIO pin. See the *GPIO Drive Strength* section for details on the selection options. *Refer to the symbols V_{OL_GPIO} and V_{OH_GPIO} in the device data sheet electrical characteristics table for details of the drive strengths for these I/O pins.* *Note: GPIO with I^2C as an AF only support two different drive strengths and do not use any bits in this register for drive strength selection. See GPIOn_DS0 for details of the two different drive strength settings.* *Refer to the symbols V_{OL_I2C} and V_{OH_I2C} in the device data sheet electrical characteristics table for details regarding which I/O pins support I^2C functionality.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-46: 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	-	RO	*	**Pullup/Pulldown Resistor Select** Reserved. * GPIO0_PS reset value is 0x0000 0003. * GPIO1_PS reset value is 0x0000 0000.

Table 6-47: GPIO Port n Voltage Select Register

GPIO Port n Voltage Select			GPIOn_VSSEL	[0x00C0]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	DNM	0	**Reserved. Do Not Modify.**

7. Flash Controller (FLC)

The MAX32672 flash controller manages read, write, erase, and mass erase operations to the internal flash. The flash controller provides the following features:

- Up to 1MB total internal flash memory
  - Two banks of 512KB each
- 64 pages per bank
- 8,192 bytes per page
- 2,048 words by 128 bits per page
- 128-bit data reads and writes
- Page erase and mass erase support
- Write protection

7.1 Instances

The device provides two instances of the flash controller. The 1MB of internal flash memory is programmable through the serial wire debug interface (in-system) or directly with user software (in-application).

The flash is organized as an array of 2,048 32-bit words by 128 bits, or 8,192 bytes per page. *Table 7-1* and *Table 7-2* show the start address, end address, start address offset, and end address offset for the two flash banks.

Table 7-1: MAX32672 Internal Flash Bank 0 Memory Organization

Instance Name	Page Number	Page Size Bytes	Start Address	End Address	Start Address Offset	End Address Offset
FLC0	1	8,192	0x1000 0000	0x1000 1FFF	0x0000 0000	0x0000 1FFF
	2	8,192	0x1000 2000	0x1000 3FFF	0x0000 2000	0x0000 3FFF
	3	8,192	0x1000 4000	0x1000 5FFF	0x0000 4000	0x0000 5FFF
	4	8,192	0x1000 6000	0x1000 7FFF	0x0000 6000	0x0000 7FFF
	…	…	…	…	…	…
	63	8,192	0x1007 C000·	0x1007 DFFF	0x0007 C000	0x0007 DFFF
	64	8,192	0x1007 E000	0x1007 FFFF	0x0007 E000	0x0007 FFFF
	Flash Information Block 0	8,192	0x1080 0000	0x1080 1FFF	0x0008 0000	0x0008 1FFF
	Flash Information Block 1	8,192	0x1080 2000	0x1080 3FFF	0x0008 2000	0x0008 3FFF

Table 7-2: MAX32672 Internal Flash Bank 1 Memory Organization

Instance Name	Page Number	Page Size Bytes	Start Address	End Address	Start Address Offset	End Address Offset
FLC1	1	8,192	0x1008 0000	0x1008 1FFF	0x0000 0000	0x0000 1FFF
	21	8,192	0x1008 2000	0x1008 3FFF	0x0000 2000	0x0000 3FFF
	3	8,192	0x1008 4000	0x1008 5FFF	0x0000 4000	0x0000 5FFF
	4	8,192	0x1008 6000	0x1008 7FFF	0x0000 6000	0x0000 7FFF
	…	…	…	…	…	…

Instance Name	Page Number	Page Size Bytes	Start Address	End Address	Start Address Offset	End Address Offset
	9	8,192	0x1009 0000	0x1009 1FFF	0x0001 0000	0x0001 1FFF
	10	8,192	0x1009 2000	0x1009 3FFF	0x0001 2000	0x0001 1FFF
	…	…	…	…	…	…
	63	8,192	0x100F C000	0x100F DFFF	0x0010 C000	0x0010 DFFF
	64	8,192	0x100F E000	0x100F FFFF	0x0010 E000	0x0010 FFFF

7.2 Usage

The flash controller manages write and erase operations to the 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. Flash is also sensitive to voltage. See *Core Operating Voltage Range Selection* for details.

7.2.1 Clock Configuration

The flash controller requires a 1MHz clock for write and erase operations. Use the flash controller clock divisor to generate $f_{FLCnCLK} = 1MHz$, as shown in *Equation 7-1*. For the IPO as the system clock, the *FLCn_CLKDIV.clkdiv* should be set to 100.

Equation 7-1: Flash Controller Clock Frequency

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

7.2.2 Lock Protection

A locking mechanism prevents accidental memory writes and erases. All write and erase operations require the *FLCn_CTRL.unlock* field to be set to 2 before starting the operation. Writing any other value to the *FLCn_CTRL.unlock* field results in:

1. The flash instance remaining locked,

   or,
2. The flash instance becoming locked from the unlocked state.

Note: If a write, page erase, or mass erase operation is started, and the unlock code was not set to 2, the flash controller hardware sets the access fail flag, FLCn_INTR.af, to indicate an access violation occurred.

7.2.3 Flash Write Width

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

7.2.4 Flash Information Blocks

The MAX32672 includes two flash information blocks.

7.2.4.1 Flash Information Block 0

Flash information block 0 is a read-only page used to store the device's USN, trim settings (option configuration and analog trim), and other nonvolatile device-specific information. The USN is a 104-bit field stored from address 0x1080 0000 through 0x1080 0017. See *Figure 7-1* for the USN mapping. Read the USN using FLC0 and perform a read of addresses 0x1080 0000 through address 0x1080 0017.

Addresses from 0x1080 0018 through 0x1080 1FFF are not user accessible.

Figure 7-1: 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	0x1080_0000	USN bits 16 - 0																	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0x1080_0004	0	USN bits 47-17																														
	0x1080_0008	USN bits 64 - 48																	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0x1080_000C	0	USN bits 95 - 65																														
	0x1080_0010	0	0	0	0	0	0	0	0	0	USN bits 103 - 96								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0x1080_0014	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

7.2.4.2 Flash Information Block 1

Flash information block 1 is mapped from address 0x1080 2000 to 0x1080 3FFF. The first 4,096 bytes of flash information block 1 is write only and is used to store the System AES key. Attempting to read from address 0x1080 2000 to 0x1080 2FFF causes a fault. The second 4,096 bytes of flash information block 1 from address 0x1080 3000 through 0x1080 3FFF is read/write and is available as nonvolatile storage for the application software. Write to this range of flash information block 1 by setting the write address from 0x1080 3000 to 0x1080 3FFF.

Erasing flash information block 1 is done by setting the address offset to 0x1080 2000 and performing a page erase. See *Page Erase* for additional details. Erasing flash information block 1 erases the entirety of the memory from 0x1080 2000 through 0x1080 3FFF.

Note: If the flash magic value is set, user information block 1 cannot be erased.

7.2.4.2.1 System AES Key Storage

The System AES Key is stored in memory from address 0x1080 2008 to 0x1080 2027. After a POR, if the flash magic values are set to the values shown in *Figure 7-3*, the System AES key stored in flash information block 1 is loaded to the *SYS_AESKEYS_KEY7*:*SYS_AESKEYS_KEY0* registers and the *SIR_STATUS*.*user_cfg_err* field is set to 1. Write a key to the System AES Key storage by performing a flash write using FLC0 with a starting address of 0x1080 2008 for the first 128-bits of the key and a second flash write starting at address offset 0x1080 2018 for the second 128-bits.

Note: All 256-bits of memory dedicated to the System AES Key storage must be written regardless of the key size (128, 192, or 256). All unused bits must be set to 0.

Enable automatic loading of the System AES Key on a POR by writing the first 128-bits of the flash magic value to address offset 0x1080 2000. Write the second 128-bits of the flash magic value to address offset 0x1080 2004. See *Figure 7-3* for the values to write. See *System AES (AES)* chapter for details on using the System AES Key.

Note: Once the flash magic value is set, the system AES keys cannot be written. Additionally, the second 4,096 bytes of flash information block 1 (address 0x1080 3000 through 0x1080 3FFF) is writable, but cannot be erased.

Note: The flash magic value and System AES Key memory is erased in devices shipped from the factory. The application software must create an AES key and store it as described above and set the flash magic values for the System AES Key registers to be loaded after a POR.

Figure 7-2: Flash Magic Value and System AES Key Mapping

		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	0x1080 2000	FMV0[3]								FMV0[2]								FMV0[1]								FMV0[0]							
	0x1080 2004	FMV1[3]								FMV1[2]								FMV1[1]								FMV1[0]							
	0x1080 2008	KEY[3]								KEY[2]								KEY[1]								KEY[0]							
	0x1080 200C	KEY[7]								KEY[6]								KEY[5]								KEY[4]							
	0x1080 2010	KEY[11]								KEY[10]								KEY[9]								KEY[8]							
	0x1080 2014	KEY[15]								KEY[14]								KEY[13]								KEY[12]							
	0x1080 2018	KEY[19]								KEY[18]								KEY[17]								KEY[16]							
	0x1080 201C	KEY[23]								KEY[22]								KEY[21]								KEY[20]							
	0x1080 2020	KEY[27]								KEY[26]								KEY[25]								KEY[24]							
	0x1080 2024	KEY[31]								KEY[30]								KEY[29]								KEY[28]							

Figure 7-3: Flash Magic Value for Automatically Loading the System AES Key on POR

		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	0x1080 2000	0x2B								0x86								0xD4								0x79							
	0x1080 2004	0x2B								0x86								0xD4								0x79							

7.2.5 Flash Write

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

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 achieve a 1MHz flash clock based on the current SYS_CLK frequency.
4. Set the *FLCn_ADDR* register to a valid address offset. See *Table 7-1* and *Table 7-2* for valid address for each flash bank.
5. Set *FLCn_DATA[3]*, *FLCn_DATA[2]*, *FLCn_DATA[1]*, and *FLCn_DATA[0]* to the data to write. *FLCn_DATA[3]* is the most significant word, and *FLCn_DATA[0]* is the least significant word. Each word of the data to write follows the little-endian format where the least significant byte of the word is stored at the lowest-numbered byte, and the most significant byte is stored at the highest-numbered byte.
6. Set *FLCn_CTRL.unlock* to 2 to unlock the flash instance.
7. Set *FLCn_CTRL.wr* to 1. This field is automatically cleared by the flash controller when the write operation is finished.
8. *FLCn_INTR.done* is set by hardware when the write completes, and if an error occurred, the *FLCn_INTR.af* flag is set. These bits generate a flash interrupt if the interrupt enable bits are set.
9. Set *FLCn_CTRL.unlock* to any value other than 2 to re-lock the flash instance.

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

Note: If using the flash bank's associated ICC the ICC should be disabled before writing to the flash or flushed after writing to the flash.

***CAUTION**: Software must ensure there are no flash writes or erase operations in progress before entering a low-power mode.*

7.2.6 Page Erase

CAUTION: Care must be taken not to 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 achieve a 1MHz flash clock based on the current SYS_CLK frequency.
4. Set the *FLCn_ADDR* register to an address offset within the target page to be erased. *FLCn_ADDR*[12:0] bits are ignored by the flash controller to ensure the address is page aligned. See *Table 7-1* for valid flash address offsets for each flash bank.
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 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 re-lock the flash instance.

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

Note: If using the flash bank's associated ICC the ICC should be disabled before writing to the flash or flushed after writing to the flash.

***CAUTION:** Software must ensure there are no flash writes or erase operations in progress before entering a low-power mode.*

7.2.7 Mass Erase

CAUTION: Care must be taken not to erase the flash instance 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 achieve a 1MHz flash clock based on the current 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 re-lock the flash instance.

***CAUTION:** Software must ensure there are no flash writes or erase operations in progress before entering a low-power mode.*

7.3 Flash Error Correction Coding

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

Note: An ECC error occurs when reading from blank/erased flash memory. Until the flash is programmed with valid data, the ECC check bits are not set.

7.4 Flash Controller 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 registers. 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.

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

Table 7-3: Flash Controller Register Summary

Offset	Register Name	Description
[0x0000]	*FLCn_ADDR*	*Flash Controller 0 Address Register*
[0x0004]	*FLCn_CLKDIV*	*Flash Controller Clock Divisor Register*
[0x0008]	*FLCn_CTRL*	*Flash Controller Control Register*
[0x0024]	*FLCn_INTR*	*Flash Controller Interrupt Register*
[0x0028]	*FLCn_ECCDATA*	*Flash Controller Error Correction Code Data*
[0x0030]	*FLCn_DATA[0]*	*Flash Controller Data Register 0*
[0x0034]	*FLCn_DATA[1]*	*Flash Controller Data Register 1*
[0x0038]	*FLCn_DATA[2]*	*Flash Controller Data Register 2*
[0x003C]	*FLCn_DATA[3]*	*Flash Controller Data Register 3*
[0x0040]	*FLCn_ACTRL*	*Flash Controller Access Control*
[0x0080]	*FLCn_WELR0*	*Flash Controller Write/Erase Lock Register 0*
[0x0088]	*FLCn_WELR1*	*Flash Controller Write/Erase Lock Register 1*
[0x0090]	*FLCn_RLR0*	*Flash Controller Read Lock Register 0*
[0x0098]	*FLCn_RLR1*	*Flash Controller Read Lock Register 1*

7.4.1 Register Details

Table 7-4: Flash Controller 0 Address Register

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

Table 7-5: Flash Controller Clock Divisor Register

Flash Controller Clock Divisor Register			FLCn_CLKDIV	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	clkdiv	R/W	0x60	**Flash Controller Clock Divisor** The system clock is divided by the value in this field to generate the flash controller clock. The flash controller clock is only used during write, erase, and mass erase operations. See the section *Clock Configuration* for details.

Table 7-6: Flash Controller Control Register

Flash Controller Control Register			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	1	**Low Voltage Enable** Set this field to 1 to enable low voltage operation for the flash memory. See *Core Operating Voltage Range Selection* for detailed usage information on this setting. 0: Low voltage operation disabled (Default). 1: Low voltage operation enabled. *Note: The PWRSEQ_LPCN.ovr field must be set to 0b00 or 0b01 before setting this field to 1.*
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**

Flash Controller Control Register			FLCn_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
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: Mass erase code.
4:3	-	RO	0	**Reserved**
2	pge	R/W1O	0	**Page Erase** Write a 1 to this field to initiate a page erase at the address in *FLCn_ADDR*.*addr*. The flash must be unlocked 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: Normal operation. 1: Write a 1 to initiate a page erase. If this field reads 1, a page erase operation is in progress.
1	me	R/W1O	0	**Mass Erase** Write a 1 to this field to initiate a mass erase of the internal flash memory. The flash must be unlocked 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: Normal operation. 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 software.*

Table 7-7: Flash Controller Interrupt Register

Flash Controller Interrupt Register			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
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**

Flash Controller Interrupt Register			FLCn_INTR	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
1	af	R/W0C	0	**Flash Access Fail Interrupt Flag** This bit is set when an attempt is made to write or erase the flash while the flash is busy or locked. Only hardware can set this bit to 1. Writing a 1 to this bit has no effect. This bit is cleared by writing a 0. 0: Normal operation. 1: Access failure occurred.
0	done	R/W0C	0	**Flash Operation Complete Interrupt Flag** This flag is automatically set by hardware after a flash write or erase operation completes. 0: Operation not complete or not in process. 1: Flash operation complete.

Table 7-8: Flash Controller ECC Data Register

Flash Controller ECC Data			FLCn_ECCDATA	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:25	-	RO	0	**Reserved**
24:16	odd	R	*	**Error Correction Code Odd Data** 9-bit ECC data recorded from the last flash read memory location of the odd address of the even/odd pair of 128-bit flash memory content.
15:9	-	RO	0	**Reserved**
8:0	even	R	*	**Error Correction Code Even Data** 9-bit ECC data recorded from the last flash read memory location of the even address of the even/odd pair of 128-bit flash memory content.

Table 7-9: 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 7-10: 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 7-11: 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 7-12: 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 7-13: 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** When this register is written with the access control sequence, the information block can be accessed. See *Flash* for details.

Table 7-14: Flash Controller Write/Erase Lock Register 0

Flash Controller Write/Erase 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. *FLCn_WELR0*[0] maps to page 1 of the flash, and *FLCn_WELR0*[31] maps to page 32. Each flash page is 8,192 bytes. Write a 1 to a bit position in this register, and the corresponding page of flash is immediately locked. Only an external reset or a POR can unlock the page protection. 0: The corresponding page of flash is write-protected. 1: The corresponding page of flash is *not* write-protected.

Table 7-15. Flash Controller Write/Erase Lock Register 1

Flash Controller Write/Erase Lock 1			FLCn_WELR1	[0x0088]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	DNM	0xFFFF	**Reserved**
15:0	welr1	R/W1C	0xFFFF	Each bit in this register maps to a page of the internal flash. *FLCn_WELR1*[0] maps to page 33 of the flash, and *FLCn_WELR1*[31] maps to page 64. Each flash page is 8,192 bytes. Write a 1 to a bit position in this register, and the corresponding page of flash is immediately locked. An external reset or a POR can only unlock the page protection. 0: The corresponding page of flash is write-protected. 1: The corresponding page of flash is *not* write-protected.

Table 7-16: Flash Controller Read Lock Register 0

Flash Controller Read Lock 0			FLCn_RLR0	[0x0090]
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. *FLCn_RLR0*[0] maps to page 1 of the flash, and *FLCn_RLR0*[31] maps to page 32 of flash. Each flash page is 8,192 bytes. 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 7-17: Flash Controller Read Lock Register 1

Flash Controller Read Lock 1			FLCn_RLR1	[0x0098]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	DNM	0xFFFF	**Reserved**
15:0	rlr1	R/W1C	0xFFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. *FLCn_RLR1*[0] maps to page 33 of the flash, and *FLCn_RLR1*[31] maps to page 64 of flash. Each flash page is 8,192 bytes. 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.

8. 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 RAM address.
- From a memory (RAM or flash) address to a transmit FIFO.
- From a source memory (RAM or flash) address to a destination RAM 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

8.1 Instances

There is one instance of the DMA, referred to as DMA. The DMA provides 12 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 8-1: MAX32672 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

8.2 DMA Channel Operation (DMA_CH)

8.2.1 DMA Channel Arbitration and DMA Bursts

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

Granting is done based on priority—a higher priority request is always granted. Within a given priority level, requests are granted on a round-robin basis. The *DMA_CHn_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.

8.2.2 DMA Source and Destination Addressing

The source and destination for DMA transfers are dictated by the request select dedicated to the peripheral instance. The *DMA_CHn_CTRL*.*request* field dictates the source and destination for a channel's DMA transfer, as shown in *Table 8-2*. 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 8-2: MAX32672 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	Reserved		
0x07	I2C0	I2C0 Receive FIFO	*DMA_CHn_DST*
0x08	I2C1	I2C1 Receive FIFO	*DMA_CHn_DST*
0x09	ADC	ADC FIFO	*DMA_CHn_DST*
0x0A	I2C2	I2C2 Receive FIFO	*DMA_CHn_DST*
0x0B:0x0D	Reserved		
0x0E	UART2	UART2 Receive FIFO	*DMA_CHn_DST*
0x0F	Reserved		
0x10	AES	AES Receive	*DMA_CHn_DST*
0x11:0x1B	Reserved		
0x1C	UART3 (LPUART0)	UART3 Receive	*DMA_CHn_DST*
0x1D	Reserved		
0x1E	I^2S	I^2S Receive	*DMA_CHn_DST*
0x1F: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	Reserved		
0x27	I2C0	*DMA_CHn_SRC*	I2C0 Transmit FIFO
0x28	I2C1	*DMA_CHn_SRC*	I2C1 Transmit FIFO
0x29	Reserved		
0x2A	I2C2	*DMA_CHn_SRC*	I2C2 Transmit FIFO
0x2B	Reserved		
0x2C	CRC	*DMA_CHn_SRC*	CRC
0x2D	Reserved		
0x2E	UART2	*DMA_CHn_SRC*	UART2 Transmit FIFO
0x2F	Reserved		
0x30	AES	*DMA_CHn_SRC*	AES
0x31:0x3B	Reserved		
0x3C	UART3 (LPUART0)	*DMA_CHn_SRC*	LPUART Transmit FIFO
0x3D	Reserved		

DMA_CHn_CTRL.request	Peripheral	DMA Source	DMA Destination
0x3E	I²S	DMA_CHn_SRC	I²S Transmit FIFO
0x3F	Reserved		

8.2.3 Data Movement from Source to DMA

Table 8-3 shows the fields that control the burst movement of data into the DMA FIFO. The source is a peripheral or memory.

Table 8-3: 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.

8.2.4 Data Movement from DMA to Destination

Table 8-4 shows the fields that control the burst movement of data out of the DMA FIFO. The destination is a peripheral or memory.

Table 8-4: 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.

8.3 Usage

Use the following procedure to perform a DMA transfer from a peripheral's receive FIFO to memory, from memory to a peripheral's transmit FIFO, or from memory to memory.

1. Ensure *DMA_CHn_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*.*cnt* field 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*.*en* to 1 to start the DMA transfer immediately.
   a. If using the reload feature, set *DMA_CHn_CTRL*.*rlden* to 1 and *DMA_CHn_CTRL*.*en* to 1 in the same instruction.
7. Wait for the interrupt flag to become 1 to indicate the completion of the DMA transfer.

8.4 Count-To-Zero (CTZ) Condition

When an AHB channel burst completes, a CTZ condition exists if *DMA_CHn_CNT* is decremented to 0.

At this point, 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.

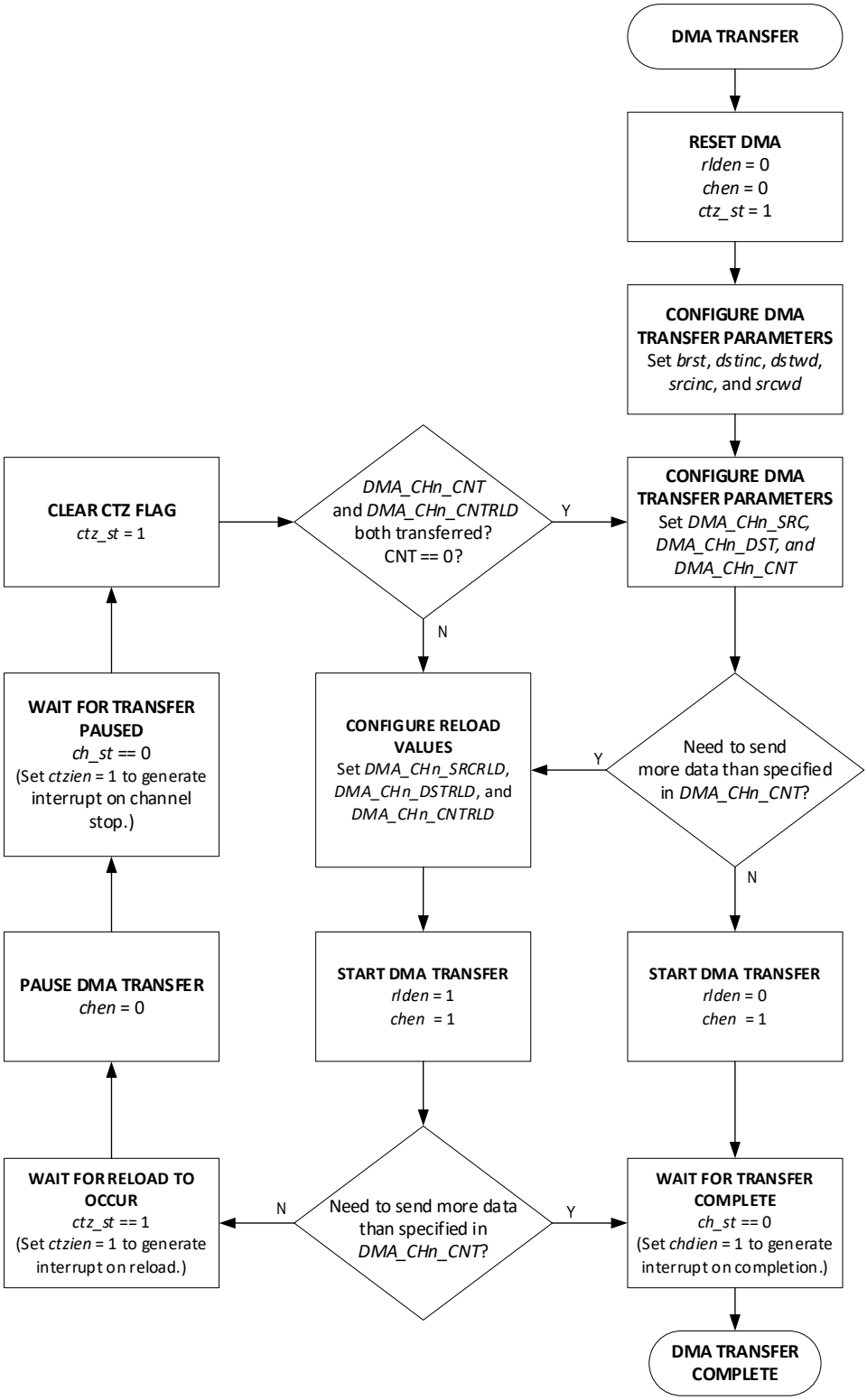
8.5 Chaining Buffers

Chaining buffers reduces the DMA interrupt response time and allows the DMA to service requests without intermediate processing from the CPU. *Figure 8-1* 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 8-1: DMA Block-Chaining Flowchart



8.6 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).

8.7 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 8-5*. 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 8-5: DMA Channel Timeout Configuration

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

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.

8.8 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.

8.9 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 8-6: DMA Register Summary

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

8.9.1 Register Details

Table 8-7: 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 8-8: 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

8.10 DMA Channel Register Summary

Table 8-9: Standard DMA Channel 0 to Channel 11 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
[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

8.11 DMA Channel Registers

See *Table 3-2* for this peripheral/module's base address. If multiple peripheral instances are provided, each instance has its own independent set of registers, shown in *Table 8-10*. 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 8-10: 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

8.11.1 Register Details

Table 8-11: 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.

DMA Channel *n* Control			DMA_CHn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
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**
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: $\frac{f_{HCLK}}{2^{24}}$

DMA Channel *n* Control			DMA_CHn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
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 - 4. 1: 7 - 8. 2: 15 - 16. 3: 31 - 32. 4: 63 - 64. 5: 127 - 128. 6: 255 - 256. 7: 511 - 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 8-2*.
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: When setting this field to 1, the DMA_CHn_CTRL.en field must also be set to 1 in the same write for proper operation.*
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 8-12: 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 has occurred.
5	-	RO	0	**Reserved**

DMA Channel *n* Status			DMA_CHn_STATUS	[0x0104]
Bits	**Field**	**Access**	**Reset**	**Description**
4	bus_err	R/W1C	0	**Bus Error** If this bit reads 1, an AHB abort occurred, and the channel was disabled by hardware. Write 1 to clear. 0: No error found. 1: An AHB bus error occurred.
3	rld_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 8-13: 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 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 8-14: 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 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 8-15: 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 field is reloaded with the contents of the *DMA_CHn_CNTRLD*.*cnt* field.

Table 8-16: 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 8-17: 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 8-18: DMA Channel n Count Reload Register

DMA Channel *n* Count Reload			DMA_CHn_CNTRLD	[0x011C]
Bits	**Field**	**Access**	**Reset**	**Description**
31	en	RO	0	**Reserved** This field is reserved and must be set to 0.
30:24	-	RO	0	**Reserved**

DMA Channel *n* Count Reload			DMA_CHn_CNTRLD	[0x011C]
Bits	**Field**	**Access**	**Reset**	**Description**
23:0	cnt	R/W	0	**Count Reload Value.** If *DMA_CHn_CTRL.rlden* = 1, then this register's value is loaded into *DMA_CHn_CNT* upon a CTZ condition.

9. Universal Asynchronous Receiver/Transmitter (UART)

The UART and the low-power UART (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:

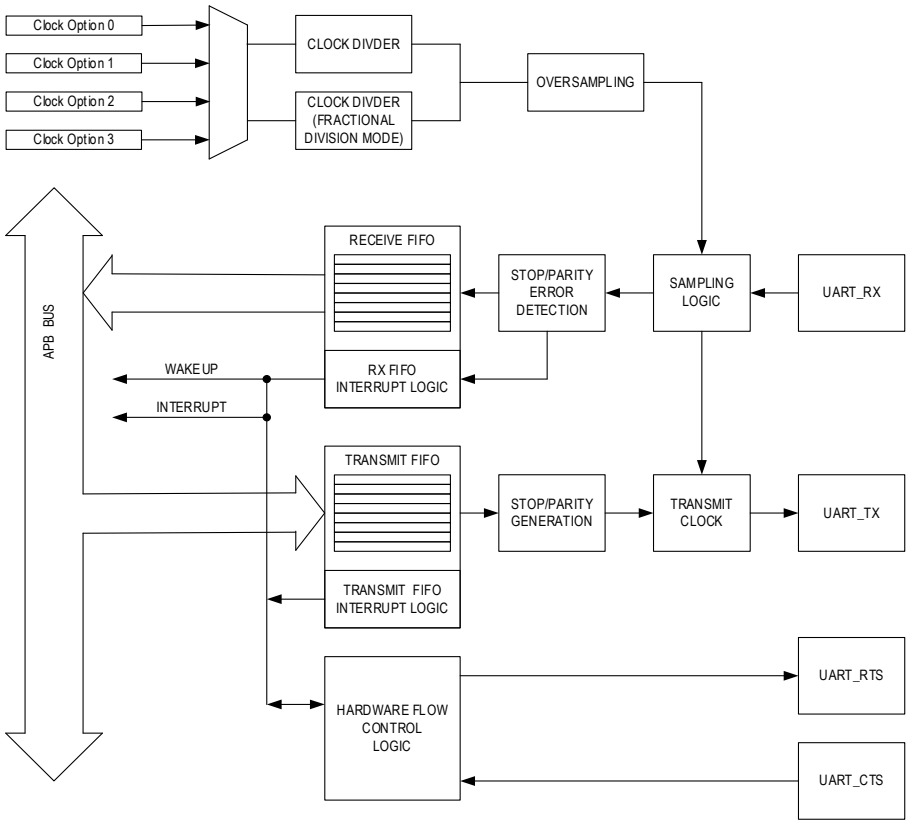
- Flexible baud rate generation
- Programmable character size of 5-bits to 8-bits
- Stop bit settings of 1, 1.5, or 2-bits
- Parity settings of even, odd, mark (always 1), space (always 0), and no parity
- Automatic parity error detection with selectable parity bias
- Automatic frame error detection
- 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*, *DEEPSLEEP*, and *BACKUP* at up to 9600 baud
- Fractional baud rate divisor improves baud rate accuracy for 9600 and lower baud rates
- Wakeup from low-power modes to *ACTIVE* on multiple receive FIFO conditions

Figure 9-1 shows a high-level diagram of the UART peripheral.

Figure 9-1: UART Block Diagram



Note: See Table 9-1 for the clock options supported by each UART instance.

9.1 Instances

Instances of the peripheral are shown in *Table 9-1*. 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 9-1: MAX32672 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	EXT_CLK1 P0.28 (AF2)	IBRO	-	Yes	8	8
UART1	UART1									
UART2	UART2									
LPUART0	UART3	Yes	*ACTIVE SLEEP*	AOD_CLK	EXT_CLK2 P0.12 (AF2)	INRO[1]	ERTCO	No		
			DEEPSLEEP BACKUP	N/A	EXT_CLK2 P0.12 (AF2)	INRO[1]	ERTCO			
1. INRO accuracy varies up to ±50% across temperature and voltage. Baud rate accuracy must be taken into account when using INRO as the clock source.										

9.2 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.

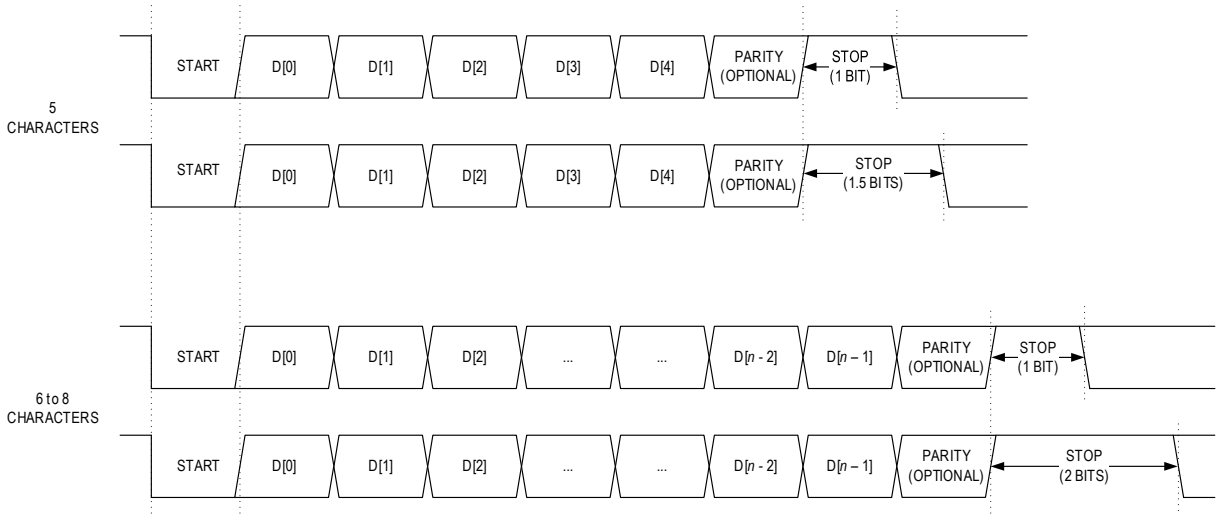
When DMA is enabled, the following describes the behavior of the DMA requests:

- 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.

9.3 UART Frame

Figure 9-2 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 9-2: UART Frame Structure



9.4 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.

9.4.1 TX FIFO Operation

Writing data to the *UARTn_FIFO.data* field increments the TX FIFO pointer, *UARTn_STATUS.tx_lvl*, and loads the data into the TX FIFO. The *UARTn_TXPEEK.data* register provides a feature that allows the software to "peek" at the current value of the write-only TX FIFO without changing the *UARTn_STATUS.tx_lvl*. Writes to the TX FIFO are ignored while *UARTn_STATUS.tx_lvl* = C_TX_FIFO_DEPTH.

9.4.2 RX FIFO Operation

Reads of the *UARTn_FIFO.data* field return the character values in the RX 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.

9.4.3 Flushing

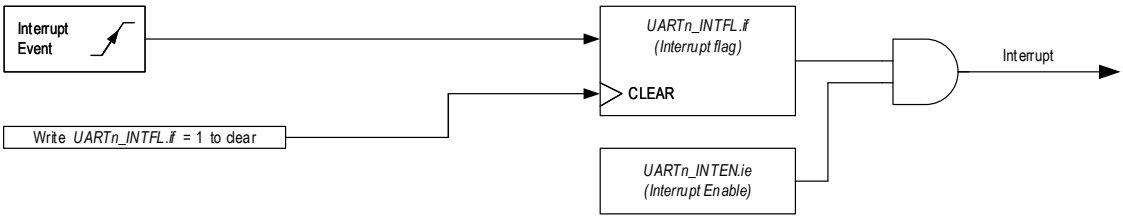
The FIFOs are flushed on the following conditions:

- Setting the *UARTn_CTRL.rx_flush* field to 1 flushes the RX FIFO by setting its pointer to 0.
- Setting the *UARTn_CTRL.tx_flush* field to 1 flushes the TX FIFO by setting its pointer to 0.
- Flush the FIFOs by setting the respective UART's reset field (*GCR_RST0*) to 1.

9.5 Interrupt Events

The peripheral generates interrupts for the events shown in *Table 9-2*. Unless noted otherwise, each instance has its own set of interrupts and higher-level flag and enable fields, as shown in *Table 9-2*

Figure 9-3: 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 9-2: MAX32672 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*

9.5.1 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 9-4*, 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 9-3* 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 9-4* for details.

Table 9-3: 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 9-4: 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	

9.5.2 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.

9.5.3 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.

9.5.4 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.

9.5.5 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*.

9.5.6 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 9-1*.

Note: When this condition occurs, verify the number of bytes in the transmit FIFO (UARTn_STATUS.tx_lvl) before refilling.

Equation 9-1: UART Transmit FIFO Half-Empty Condition

$$\left(\frac{C\_TX\_FIFO\_DEPTH}{2}+1\right) \xrightarrow{Transistions\ from} \left(\frac{C\_TX\_FIFO\_DEPTH}{2}\right)$$

9.5.7 Transmit FIFO Almost Empty

The transmit FIFO almost empty event occurs where there is one byte remaining in the transmit FIFO.

9.6 LPUART Wakeup Events

LPUART instances can receive characters while in the low-power modes listed in *Table 9-1*. If enabled, each of the receive FIFO conditions shown in *Table 9-5* wakes the device, exits the low-power mode, and returns the device to *ACTIVE*.

Unlike interrupts, wakeup activity is based on a condition, not an event. As long as the condition is true and the wakeup enable field is set to 1, the wakeup flag remains set.

Table 9-5: MAX32672 Wakeup Events

Receive FIFO Condition	Wakeup Flag *UARTn_WKFL*	Wakeup Enable *UARTn_WKEN*	Low-Power Peripheral Wakeup Flag	Low-Power Peripheral Wakeup Enable	Low-Power Clock Disable
Threshold	*rx_thd*	*rx_thd*	*PWRSEQ_LPPWKST.lpuart0*	*PWRSEQ_LPPWKEN.lpuart0*	*GCR_PM.lpuart0_we*
Full	*rx_full*	*rx_full*			
Not Empty	*rx_ne*	*rx_ne*			

9.6.1 RX FIFO Threshold

This condition persists while *UARTn_STATUS.rx_lvl* ≥ *UARTn_CTRL.rx_thd_val*.

9.6.2 RX FIFO Full

This condition persists while *UARTn_STATUS.rx_lvl* ≥ C_RX_FIFO_DEPTH.

9.6.3 RX Not Empty

This condition persists while *UARTn_STATUS.rx_lvl* > 0.

9.7 Inactive State

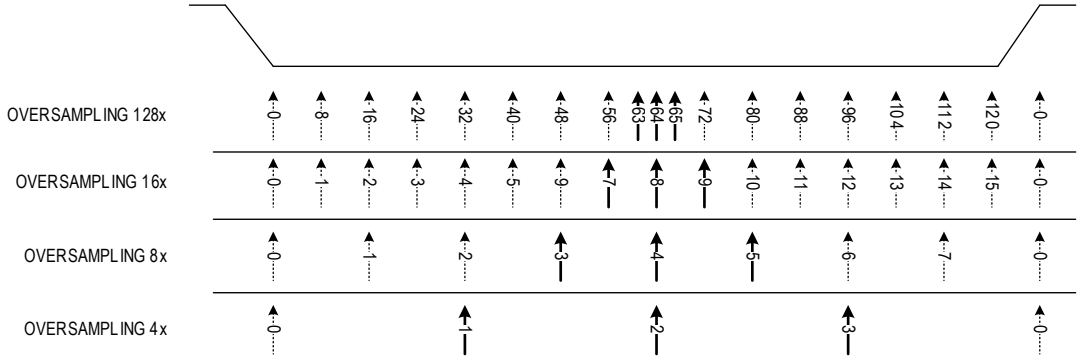
The following conditions result in the UART being inactive:

- 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

9.8 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 9-4*.

Figure 9-4: 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).

9.9 Baud Rate Generation

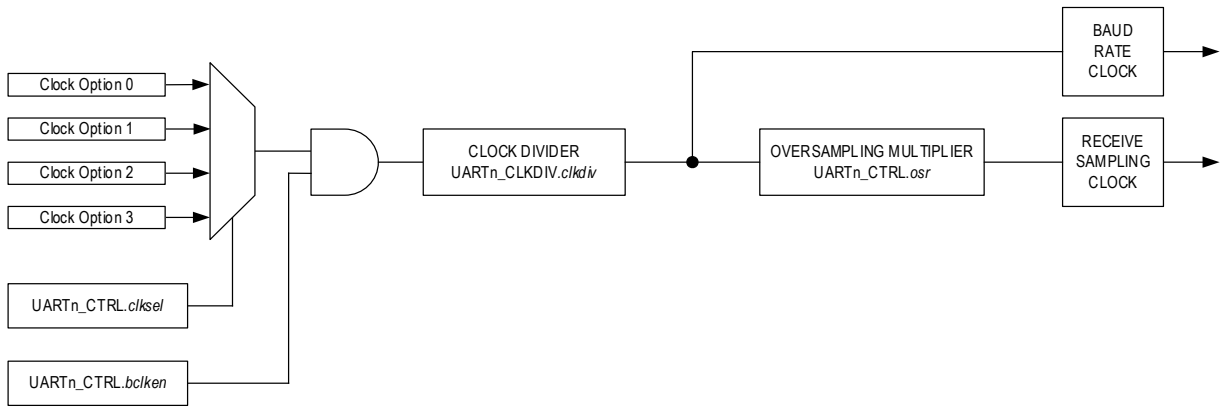
The baud rate is determined by the selected UART clock source and the value of the clock divisor. Multiple clock sources are available for each UART instance. See *Table 9-1* for available clock sources.

Note: Changing the clock source should only be done between data transfers to avoid corrupting an ongoing data transfer.

9.9.1 UART Clock Sources

Standard UART instances operate only in *ACTIVE* and *SLEEP*. Standard UART instances can only wake the device from *SLEEP*. *Figure 9-5* shows the baud rate generation path for standard UARTs.

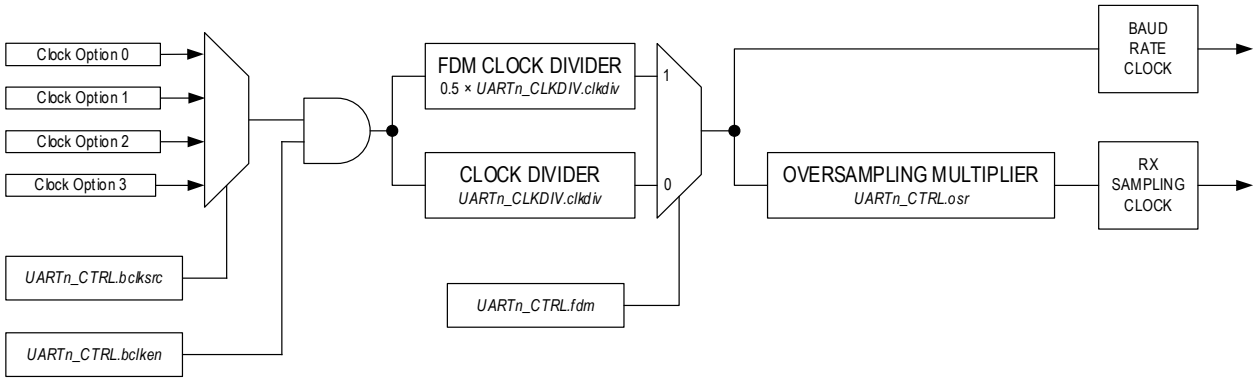
Figure 9-5: UART Baud Rate Generation



9.9.2 LPUART Clock Sources

LPUART instances support FDM and are configurable for operation at 9600 and lower baud rates for operation in *SLEEP*, *DEEPSLEEP*, and *BACKUP*. Operation in *DEEPSLEEP* and *BACKUP* require the use of the *ERTCO* as the baud rate clock source. The *ERTCO* can be configured to remain active in *DEEPSLEEP* and *BACKUP*, allowing the LPUART to receive data and serve as a wakeup source while power consumption is minimized.

Figure 9-6: LPUART Timing Generation



9.9.3 Baud Rate Calculation

The transmit and receive circuits share a common baud rate clock: the selected UART clock source divided by the clock divisor. Instances that support FDM offer a 0.5 fractional clock division when enabled by setting *UARTn_CTRL.fdm* = 1. The FDM allows for greater accuracy when operating at low baud rates and finer granularity for the oversampling rate.

Use the following formula to calculate the *UARTn_CLKDIV.clkdiv* value based on the clock source, and desired baud rate, and integer or fractional divisor.

Equation 9-2: 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 9-3: LPUART Clock Divisor Formula for UARTn_CTRL.fdm = 1

$$UARTn\_CLKDIV.clkdiv = INT\left[\frac{f_{UART\_CLK}}{Baud\ Rate}\right] \times 2$$

$$if\ f_{UART\_CLK}\%\ Baud\ rate > \frac{Baud\ rate}{2}\ or\ UARTn\_CLKDIV.clkdiv = 0, then\ UARTn\_CLKDIV.clkdiv + 1$$

For example, in a case where the UART clock is PCLK (50MHz), and the desired baud rate is 115,200bps:

$$UARTn\_CLKDIV.clkdiv = \left(\frac{50{,}000{,}000}{115{,}200}\right) = 434$$

For a low-power UART with AOD_CLK (PCLK = 50MHz) selected as the clock source, the desired baud rate is 115,200bps, *GCR_PCLKDIV.aon_clkdiv* = 3, *UARTn_CTRL.fdm* = 0, and calculate the *UARTn_CLKDIV.clkdiv* field as follows:

$$AOD\_CLK = \frac{50{,}000{,}000}{4 \times 2^3} = 1{,}562{,}500$$

$$UARTn\_CLKDIV.clkdiv = INT\left[\frac{1{,}562{,}500}{115{,}200}\right] = 13$$

IMPORTANT: UARTn_CLKDIV.clkdiv must be greater than UARTn_OSR.osr. In general, an *UARTn_OSR.osr* setting of 5 is sufficient for most applications.

9.10 Low-Power Receiver Operation

LPUART instances have the option to configure the receiver for 9600 and lower baud rates and enable the LPUART in the low-power modes *SLEEP*, *DEEPSLEEP*, and *BACKUP*. Receipt of a valid frame loads the receive FIFO and increments *UARTn_STATUS.rx_lvl*. If a wakeup event, shown in Table 9-5, is enabled, the device exits the current low-power mode and returns to *ACTIVE*.

The LPUARTs support FDM (*UARTn_CTRL.fdm*), allowing greater accuracy when operating at very low baud rates and finer granularity for the oversampling rate. If *UARTn_CLKDIV.clkdiv* is less than 16, OSR is not used, and the receive signal is sampled on the first clock edge. Dual-edge sampling is supported by setting *UARTn_CTRL.desm* to 1.

Table 9-6 uses Equation 9-3 to calculate the *UARTn_CLKDIV.clkdiv* settings and possible OSR settings for LPUART operating with FDM enabled (*UARTn_CTRL.fdm* = 1).

Table 9-6: LPUART Low Baud Rate Generation Examples (UARTn_CTRL.fdm = 1)

Clock Source	BAUD (bits/s)	Calculated *UARTn_CLKDIV.clkdiv*	*UARTn_OSR.osr*
ERTCO	9,600	6	N/A (1×)
	7,200	9	N/A (1×)
	4,800	13	N/A (1×)
	2,400	27	0: 8× 1: 12×
	1,800	36	0: 8× 1: 12× 2: 16×
	1,200	54	0: 8× 1: 12× 2: 16× 3: 20× 4: 24×

9.10.1 Configuring an LPUART for Low-Power Modes of Operation

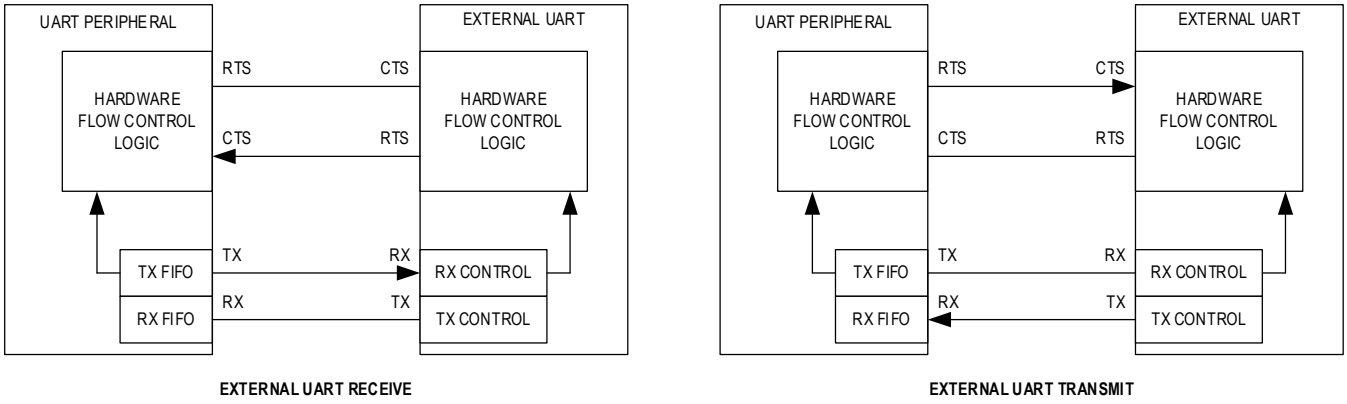
Use the following procedure to receive characters at 9600 or lower baud rates while in low-power modes:

1. Enable the LPUART for operation in *SLEEP*, *DEEPSLEEP* and *BACKUP* by setting *MCR_PCLKDIS*.*lpuart0* to 0.
2. Configure the required LPUART pins for use by enabling each pin using the *MCR_LPPIOCTRL* register.
   a. Enable the transmit pin by setting *MCR_LPPIOCTRL*.*lpuart0_tx* to 1.
   b. Enable the receive pin by setting *MCR_LPPIOCTRL*.*lpuart0_rx* to 1.
   c. If using hardware flow control, enable RTS and CTS by setting the *MCR_LPPIOCTRL*.*lpuart0_rts* and *MCR_LPPIOCTRL*.*lpuart0_cts* fields to 1.
3. Disable the baud clock by clearing *UARTn_CTRL*.*bclken* to 0.
   a. Read *UARTn_CTRL*.*bclkrdy* until it is 0.
4. Set *PWRSEQ_LPCN*.*ertco_en* to 1 to enable the ERTCO.
5. Set *UARTn_CTRL*.*ucagm* to 1.
6. Set *UARTn_CTRL*.*bclksrc* to 2 to select the ERTCO source.
7. Enable FDM by setting *UARTn_CTRL*.*fdm* to 1.
8. Set *UARTn_CLKDIV*.*clkdiv* to 6 as calculated in *Table 9-6*.
9. Set *UARTn_CTRL*.*desm* to 1 to enable receive sampling on both the rising and falling edge.
10. Enable the LPUART as a wake-up source during low-power modes by:
    a. Set *PWRSEQ_LPPWKEN*.*lpuart0* to 1.
    b. Set *GCR_PM*.*lpuart0_we* to 1.
11. Clear all the wake-up flags shown in *Table 9-5*.
12. Choose the desired wake-up condition from *Table 9-5* and set the corresponding wake-up enable field to 1.
    a. For example, to wake on the first byte received, set *UARTn_WKEN*.*rx_ne* to 1.
13. Re-enable the baud clock by setting *UARTn_CTRL*.*bclken* to 1.
14. Read the *UARTn_CTRL*.*bclkrdy* field until it reads 1.
15. Enter the desired low-power mode.

9.11 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 9-7*.

Figure 9-7: 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.

9.11.1 Automated HFC

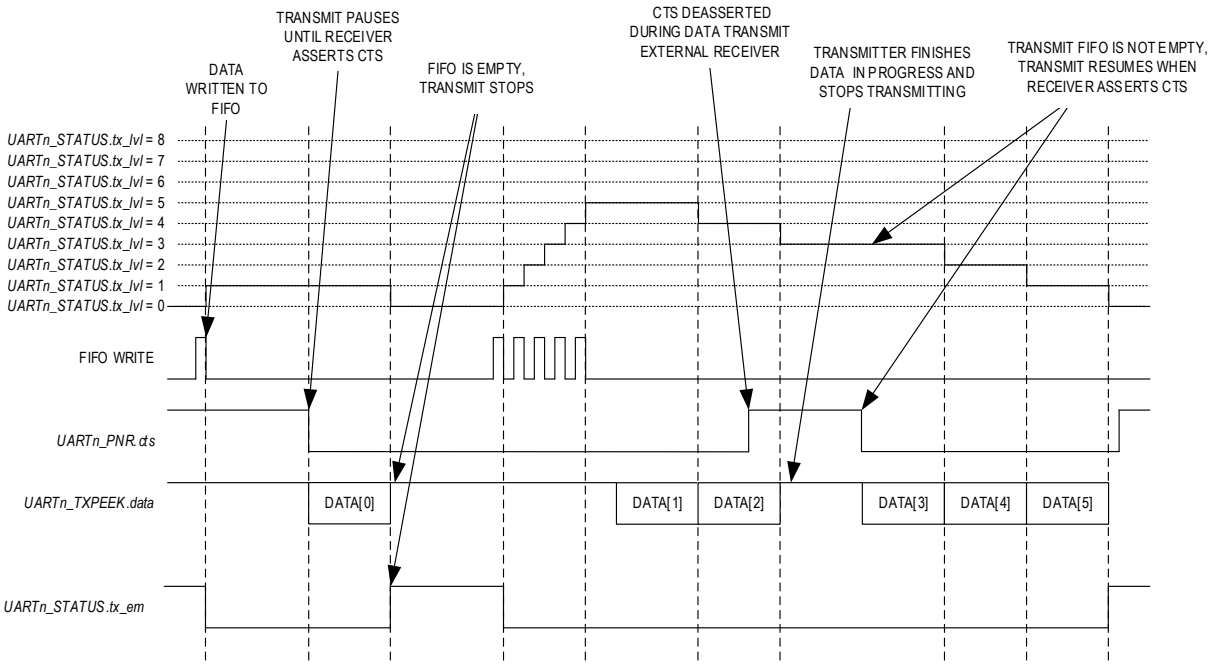
Setting *UARTn_CTRL*.*hfc_en* = 1 enables automated HFC. When automated HFC is enabled, the hardware manages the CTS and RTS signals. The deassertion of the RTS signal is configurable using the *UARTn_CTRL*.*rtsdc* field:

- *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 9-8* 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 9-8: HFC Signaling for Transmitting to an External Receiver



9.11.2 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.

9.11.2.1 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.

9.12 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 9-7*. 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 9-7: 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 Wakeup Interrupt Enable Register
[0x0038]	*UARTn_WKFL*	UART Wakeup Interrupt Flag Register

9.12.1 Register Details

Table 9-8: 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.	

UART Control				UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
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 9-1* for available clock options for each UART instance. 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.	
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	R/W	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	R/W	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 1 bits (mark). 1: Parity calculation is based on 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.	

UART Control				UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
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 9-9: UART Status Register

UART Status				UARTn_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:16	-	RO	0	**Reserved**	
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 9-10: 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	-	RO	0	**Reserved**
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 9-11: 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	-	RO	0	**Reserved**
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 9-12: 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 9-13: 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 9-14: 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 9-15: 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 9-16: UART Data Register

UART Data			UARTn_FIFO	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**

UART Data			UARTn_FIFO	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
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 9-17: 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 9-18: UART Wakeup Enable

UART Wakeup Enable			UARTn_WKEN	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	rx_thd	R/W	0	**Receive FIFO Threshold Wakeup Event Enable** 0: Disabled. 1: Enabled.
1	rx_full	R/W	0	**Receive FIFO Full Wakeup Event Enable** 0: Disabled. 1: Enabled.
0	rx_ne	R/W	0	**Receive FIFO Not Empty Wakeup Event Enable** 0: Disabled. 1: Enabled.

Table 9-19. UART Wakeup Flag Register

UART Wakeup Flag			UARTn_WKFL	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	rx_thd	R/W	0	**Receive FIFO Threshold Wakeup Event** 0: Disabled. 1: Enabled.
1	rx_full	R/W	0	**Receive FIFO Full Wakeup Event** 0: Disabled. 1: Enabled.
0	rx_ne	R/W	0	**Receive FIFO Not Empty Wakeup Event** 0: Disabled. 1: Enabled.

10. 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 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*.

10.1 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.

10.2 Instances

The two instances of the peripheral are shown in *Table 10-1*. The table lists the alternate function names of the SDA and SCL signals for each of the I²C peripherals.

Table 10-1: MAX32672 I²C Peripheral Pins

I²C Instance	Alternate Function y = Alternate Function Number (A = AF1, B = AF2, C = AF2, D = AF3, E = AF4)*
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.*	

10.3 I²C Overview

10.3.1 I²C Bus Terminology

Table 10-2 contains terms and definitions used in this chapter for the I²C bus terminology.

Table 10-2: 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.

10.3.2 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. The receiving devices sends an acknowledge bit 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.

10.3.3 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.

10.3.4 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.

10.3.5 Acknowledge and Not Acknowledge

The receiver generates an acknowledge bit (ACK), whether I^2C controller or target, after every byte received by pulling SDA low. The ACK bit is how the receiver tells the transmitter 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 I^2C 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.

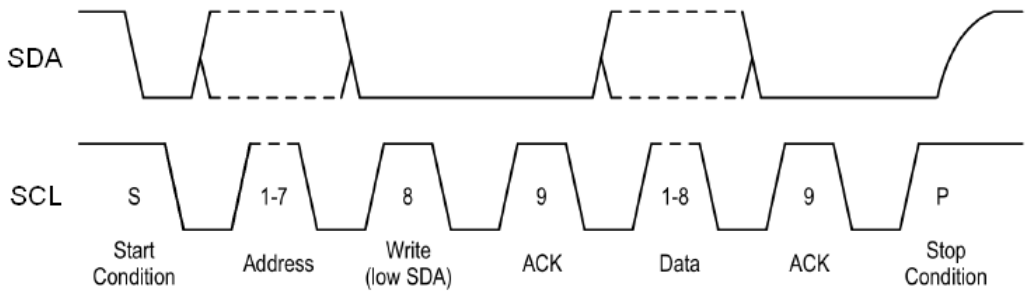
A receiver can generate a NACK after a byte transfer if any of the following conditions occur:

- 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 I^2C controller has requested data from a target, it signals the target to stop transmitting by sending a NACK following the last byte it requires.

10.3.6 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 I^2C specification states that during data transfer, the SDA line can change state only when SCL is low, that the SDA is stable, and can be read when SCL is high, as shown in *Figure 10-1*.

Figure 10-1: I^2C 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 8 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.

10.4 Configuration and Usage

10.4.1 SCL and SDA Bus Drivers

SCL and SDA are open-drain signals. In this device, once the I²C peripheral is enabled and the proper GPIO alternate function is selected, the corresponding pad circuits are automatically configured as open-drain outputs.

10.4.2 SCL Clock Configurations

The SCL frequency depends on the values of the I²C peripheral clock, and the values of the external pullup resistor and trace capacitance on the SCL clock line.

Note: An external RC load on the SCL line affects the target SCL frequency calculation.

10.4.3 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 10-2*. The SCL low time is configured in the I²C Clock Low Time register field *I2Cn_CLKLO.lo* using *Equation 10-3*. Each of these fields is 8 bits. The I²C frequency value is shown in *Equation 10-1*.

Equation 10-1: I²C Clock Frequency

$$f_{I2C\_CLK} = \frac{1}{t_{I2C\_CLK}} \; is\ either\ f_{PCLK}\ or\ f_{IBRO}$$

Equation 10-2: I²C Clock High Time Calculation

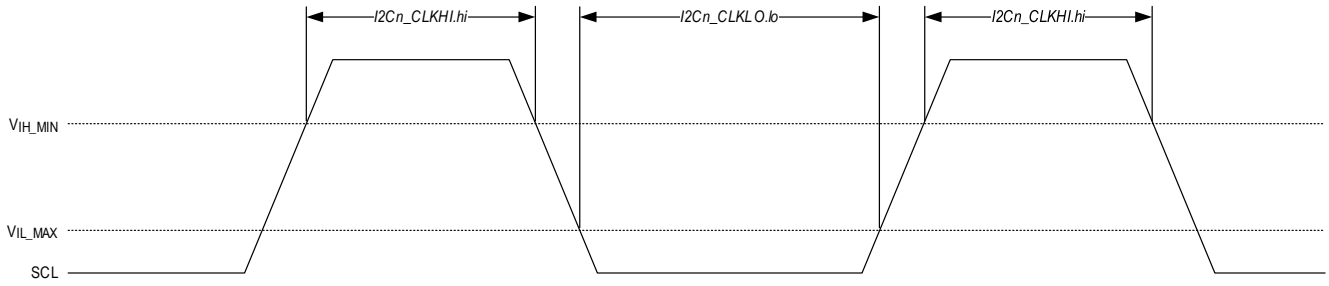
$$t_{SCL\_HI} = t_{I2C\_CLK} \times (I2Cn\_CLKHI.hi + 1)$$

Equation 10-3: I²C Clock Low Time Calculation

$$t_{SCL\_LO} = t_{I2C\_CLK} \times (I2Cn\_CLKLO.lo + 1)$$

Figure 10-2 shows the association between the SCL clock low and high times for Standard Mode, Fast Mode, and Fast Mode Plus I²C frequencies.

Figure 10-2: 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.

10.4.4 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, configure the lower speed mode as well. See *SCL Clock Generation for Standard, Fast and Fast-Plus Modes* for information regarding the configuration of lower speed modes.

10.4.4.1 Hs-Mode Timing

With I²C bus capacitances less than 100pf, the following specifications are extracted from the I²C-bus Specification User Manual Rev. 6 April 2014 *https://www.nxp.com/docs/en/user-guide/UM10204.pdf*

t_{LOW_MIN} = 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.

10.4.4.2 Hs-Mode Clock Configuration

The maximum Hs-Mode bus clock frequency can now be determined. The system clock frequency, f_{SYS_CLK}, must be known. Hs-Mode timing information from *Hs-Mode Timing* must be used.

Equation 10-4: 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 10-5: 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 10-6: 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 10-7: The Calculated Frequency of the I²C Bus Clock Using the Results of Equation 10-5 and Equation 10-6

$$Calculated\ Frequency = \left((I2Cn\_HS\_CLK.hsclk\_hi + 1) + (I2Cn\_HS\_CLK.hsclk\_lo + 1) \right) * t_{I2C\_CLK}$$

Table 10-3 shows the I²C bus clock calculated frequencies given different f_{SYS_CLK} frequencies.

Table 10-3: Calculated I²C Bus Clock Frequencies

f_{SYS_CLK} (MHz)	I2Cn_HSCLK.hi	I2Cn_HSCLK.lo	Calculated Frequency (MHz)
100	4	9	3.3
50	2	4	3.125
25	1	2	2.5

10.4.5 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 10-4: I²C Target Address Format

Target Address Bits		R/W Bit	Description
0b0000	0b000	0	General Call Address
0b0000	0b000	1	START Condition
0b0000	0b001	*x*	CBUS Address
0b0000	0b010	*x*	Reserved for different bus format
0b0000	0b011	*x*	Reserved for future purposes
0b0000	0b1*xx*	*x*	HS-mode controller code
0b1111	0b1*xx*	*x*	Reserved for future purposes
0b1111	0b0*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. It is followed by a second byte representing the remainder of the 10-bit address. If the operation is a write, it 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.

10.4.6 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, only configure the following registers when either:

1. The I^2C peripheral is disabled,

   or
2. The I^2C 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.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 following register fields can be safely (re)programmed at any time:

- All interrupt flags and interrupt enable bits
- *I2Cn_TXCTRL0.thd_lvl*
- *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*

10.4.6.1 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 controller transmits the target address 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.

10.4.6.2 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.

10.4.6.3 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 an address programmed in the target address registers (*I2Cn_SLAVE3*:*I2Cn_SLAVE0*), 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 10-3), 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.

10.4.7 Target Mode Addresses

There are two methods to set the target address for the I²C peripherals based on the devices major die revision. Up to four target addresses are supported. The following sections describe the two methods of setting target mode addresses based on the die revision. Read the die revision by reading the *GCR_REVISION* register.

10.4.7.1 Revision 0xA1 Target Addresses

If the die revision is 0xA1, use the following steps to set up to four target addresses.

1. Set the *I2Cn_SLAVE.index* field to 0 to set target address 0.
2. Set *I2Cn_SLAVE.addr* to either a 7-bit or 10-bit address.
   a. For 7-bit addressing, the address occupies the least significant 7 bits of the *I2Cn_SLAVE.addr* field.
   b. For 10-bit addressing, the 9-bits of address occupies the most significant 9 bits of the *I2Cn_SLAVE.addr* field, and the R/W bit occupies the least significant bit.
      i. Set *I2Cn_SLAVE.ext_addr_en* to 1 if using a 10-bit address.
3. Set *I2Cn_SLAVE.dis* field to 0 to enable the target address.
4. Set the *I2Cn_SLAVE.index* field to 1 to set target address 1.
5. Set *I2Cn_SLAVE.addr* field to a 7-bit address (10-bit is only supported for address index 0).
6. Set the *I2Cn_SLAVE.dis* field to 0 to enable the target address.
7. Repeat steps 4 - 6 for indexes 2 and 3 (*I2Cn_SLAVE.index* = 2 and *I2Cn_SLAVE.index* = 3).

10.4.7.2 Target Addresses for All Other Die Revisions

For all other die revisions (0xB1, 0xB2, etc.), use the following steps to set up to four target addresses.

1. 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 to the desired address for the device on the bus.
2. For extended addresses, set the corresponding *I2Cn_SLAVE3.ext_addr_en*:*I2Cn_SLAVE0.ext_addr_en* to 1 for 10-bit addressing or 0 for 7-bit addressing.
3. Enable each I²C target address register by clearing the corresponding disable field to 0 (*I2Cn_SLAVE3.dis*:*I2Cn_SLAVE0.dis*).

10.4.8 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.

10.4.8.1 Target Mode Addresses

There are two methods to set the target address for the I²C peripherals based on the devices major die revision. Up to four target addresses are supported. Follow the steps below to determine which method to use based on the die revision.

For target mode operation, configure the following register fields with the I²C peripheral disabled:

- *I2Cn_CTRL.mst_mode* = 0 for target operation.
- Set up the I²C target addresses. See *Target Mode Addresses* for details.
- *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 10-2* 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 10-6* 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_lvl* 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.

10.4.8.2 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.

10.4.8.2.1 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 up the I²C target addresses. See *Target Mode Addresses* for details.
   b. Just-in-time mode specific settings:
      i) *I2Cn_CTRL*. *clkstr_dis* = 0
      ii) *I2Cn_TXCTRL0*[5:2] = 0x8
      iii) *I2Cn_TXCTRL0*.*preload_mode* = 0.
   c. Program *I2Cn_CLKHI*.*hi* and *I2Cn_HSCLK*.*hi* with appropriate values satisfying $t_{SU;DAT}$ (and HS $t_{SU;DAT}$).
2. Set the *I2Cn_CTRL*.*en* field to 1 to enable the I²C operation.
   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 an 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 generate an interrupt.
4. After *I2Cn_INTFL0*.*addr_match* =1, the software determines if a read or write address match occurred (*I2Cn_INTFL0*.*wr_addr_match* = 1 for write and *I2Cn_INTFL0*.*rd_addr_match* = 1 for a read. In this case, assume read = 1, indicating transmit.
   a. Software can determine which target address matched by reading the *I2Cn_INTFL0*.*mami* field. Each bit in the field corresponds to each target address.
   b. 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*, along with other address match flags in the *I2Cn_INTFL0* register. Now that *I2Cn_INTFL0*.*tx_lockout* is 0, the software can begin loading the transmit data into the *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_lvl* 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 are 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.

10.4.8.2.2 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 up the I^2C target addresses. See *Target Mode Addresses* for details.
   b. 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_lvl,* 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_INTFL0*.*rd_addr_match* to determine if the transaction is a read and reads *I2Cn_INTFL0*.*wr_addr_match* to determine if the transaction is a write. In this case, assume *I2Cn_INTFL0*.*rd_addr_match* = 1, indicating a transmit.
   a. The hardware begins sending out the data 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_lvl* 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.

10.4.8.3 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 up the I²C target addresses. See *Target Mode Addresses* for details.
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.

10.4.9 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 is 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.

10.4.10 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*.

10.4.11 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. Preload the transmit FIFO 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.

10.4.12 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).

10.4.13 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.

10.4.14 Bus Timeout

The timeout field, *I2Cn_TIMEOUT.scl_to_val,* is used to detect bus errors. *Equation 10-8* and *Equation 10-9* show equations for calculating the maximum and minimum timeout values based on the value loaded into the *I2Cn_TIMEOUT.scl_to_val* field.

Equation 10-8: 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 10-9*.

Equation 10-9: 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).

10.4.15 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_lvl*) 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_lvl*, then the DMA transfers data into the transmit FIFO according to the DMA configuration.

Set the DMA burst size as shown in *Equation 10-10* to ensure the DMA does not overflow the transmit FIFO:

Equation 10-10: 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_lvl* 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. Set the DMA burst size as shown in *Equation 10-11* to ensure the DMA does not underflow the receive FIFO:

Equation 10-11: 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.

10.5 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 10-5*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 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 10-5: 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 Control 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
[0x0044]	*I2Cn_SLAVE*	I²C Target Address Register (Revision 0xA1 devices only)
[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

10.5.1 Register Details

Table 10-6: 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.

I²C Control			I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
14	-	RO	0	**Reserved**
13	-	RO	0	**Reserved**
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.
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.

I²C Control			I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
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 10-7: I²C Status Register

I²C Status			I2Cn_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**
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 10-8: 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 is accessed for a write (i.e., receive) transaction in target mode, and the address received matches the device target address. See *I2Cn_INTFL0.mami* to determine which target address match occurred. 0: No address match. 1: Address match.

I²C Interrupt Flag 0			I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
22	rd_addr_match	R/W1C	0	**Target Read Address Match Interrupt Flag** If set, the device is accessed for a read (i.e., transmit) transaction in target mode, and the address received matches the device target address. See *I2Cn_INTFL0.mami* to determine which target address match occurred. 0: No address match. 1: Address match.
21:20	-	RO	0	**Reserved**
19:16	mami	R/W1C	0	**MAMI Interrupt Flag** Each bit in this field corresponds to an address match interrupt with the corresponding target address register. For example, *I2Cn_SLAVE2* or *I2Cn_SLAVE.index* = 2, corresponds to *mami[2]* (bit 18).
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.
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.

I²C Interrupt Flag 0			I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
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 is 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.
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.

I²C Interrupt Flag 0			I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
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 10-9: 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.
21:20	-	RO	0	**Reserved**
19:16	mami	R/W	0	**MAMI Interrupt Enable** Each bit in this field enables an interrupt for the corresponding target address register. For example, *I2Cn_SLAVE0* address corresponds to *mami*[0] (bit 16). 0: Disabled. 1: Enabled.
15	tx_lockout	R/W	0	**Transmit FIFO Lock Out Interrupt Enable** 0: Disabled. 1: Enabled.
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.

I²C Interrupt Enable 0			I2Cn_INTEN0	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
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 10-10: I²C Interrupt Flag 1 Register

I²C Interrupt Status Flags 1			I2Cn_INTFL1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	start	R/W1C	0	**START Condition Status Flag** If set, a device START condition is 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.

I²C Interrupt Status Flags 1			I2Cn_INTFL1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
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 10-11: 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 10-12: 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 10-13: I²C Receive Control 0 Register

I²C Receive Control 0			I2Cn_RXCTRL0	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
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.

I²C Receive Control 0			I2Cn_RXCTRL0	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
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 is 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 10-14: 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 10-15: I²C Transmit Control 0 Register

I²C Transmit Control 0			I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**

I²C Transmit Control 0			I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
11:8	thd_lvl	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 10-16: 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 10-17: I²C Data Register

I²C Data			I2Cn_FIFO	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**

I²C Data			I2Cn_FIFO	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
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 10-18: 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 10-19: 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 10-20: I²C SCL High Control Register

I²C Clock High Control			I2Cn_CLKHI	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**

I²C Clock High Control			I2Cn_CLKHI	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
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 10-21: 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**
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 10-22: 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 10-23: I²C Target Address Register

I²C Target Address Register		I2Cn_SLAVE		[0x0044]
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. *Note: This field is only writable when I2Cn_SLAVE.index is set to 0.* *Note: This register is only valid in revision A1 material (GCR_REVISION= 0xA1).*
14:13	-	RO	0	**Reserved**
12:11	index	R/W	0	**Target Address Index** This field acts as an index to four different target addresses. This field is valid for both reads and writes to the register. For example, to set target address 1, set this field to 1 and then write the *I2Cn_SLAVE.addr* field to the desired 7-bit address. Additionally, enable the target address by writing 0 to *I2Cn_SLAVE.dis* field for each index desired to be enabled. Read each address by setting this field to the desired value and reading the address field. 0: Target address 0. 1: Target address 1. 2: Target address 2. 3: Target address 3. *Note: This register is only valid in revision A1 material (GCR_REVISION= 0xA1).*
10	dis	R/W	*	**Target Address Disable** Setting this field to 1 disables this register's target address. *Note: On reset, target address 0 is enabled, all other addresses are disabled.* *Note: This register is only valid in revision A1 material (GCR_REVISION= 0xA1).*
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. Extended address is only supported for target address 0 (*I2Cn_SLAVE.addr* = 0). *Note: I2Cn_SLAVE0.ext_addr_en controls if this field is a 7-bit or 10-bit address.* *Note: This register is only valid in revision A1 material (GCR_REVISION= 0xA1).*

Table 10-24: 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 10-25: 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.
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 10-26: 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	1	**Target Address 1 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 10-27: 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	1	**Target Address 2 Disable** Setting this field to 1 disables this register's target address.

I²C Target Address 2			I2Cn_SLAVE2	[0x0054]
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_SLAVE2.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

Table 10-28: 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	1	**Target Address 3 Disable** Setting this field to 1 disables this registers 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_SLAVE3.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

11. Inter-Integrated Sound Interface (I²S)

I²S is a serial audio interface for communicating pulse-code modulation (PCM) encoded streams between devices. The peripheral supports both master and slave 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 I²S 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 I²S 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 I²S serial data input and serial data output pins.

11.1 Instances

Table 11-1: MAX32672 I²S Instances

Instance	Supported Channels	I2S_CLK Clock Options		Receive FIFO Depth	Transmit FIFO Depth
I2S	Stereo	ERFO	PCLK	8 × 32-bits	8 × 32-bits

Note: ERFO must be enabled for master operation; in slave operation, external clocking is used for the LRCLK and BCLK input pins.

11.1.1 I²S Bus Lines and Definitions

The I²S 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 master mode and must be enabled.

Detailed pin and alternate function mapping are shown in *Table 11-2*.

Table 11-2: MAX32672 I²S Pin Mapping

Instance	I²S Signal	Pin Description	40 TQFN	Alternate Function Number	Notes
I2S	BCLK (SCK)	I²S bit clock	P0.10	AF2	Also referred to as serial clock
	LRCLK (WS)	I²S left/right clock	P0.9	AF2	Also referred to as word select
	SDI	I²S serial data input	P0.11	AF2	
	SDO	I²S serial data output	P0.8	AF2	

11.2 Details

The I²S supports full-duplex serial communication with separate SDI and SDO pins. *Figure 11-1* shows an interconnect between a peripheral configured in host mode, communicating with an external I²S slave receiver and an external I²S transmitter. In master mode, the peripheral hardware generates the BCLK and LRCLK, and each is output to each slave device.

Note: Master operation requires the use of the ERFO to generate the LRCLK and BCLK signals.

Figure 11-1: I²S Master Mode

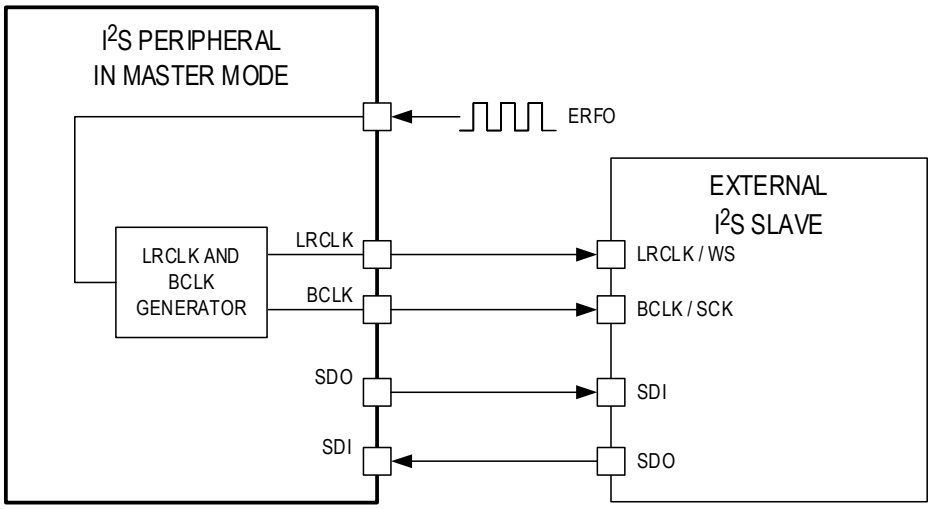
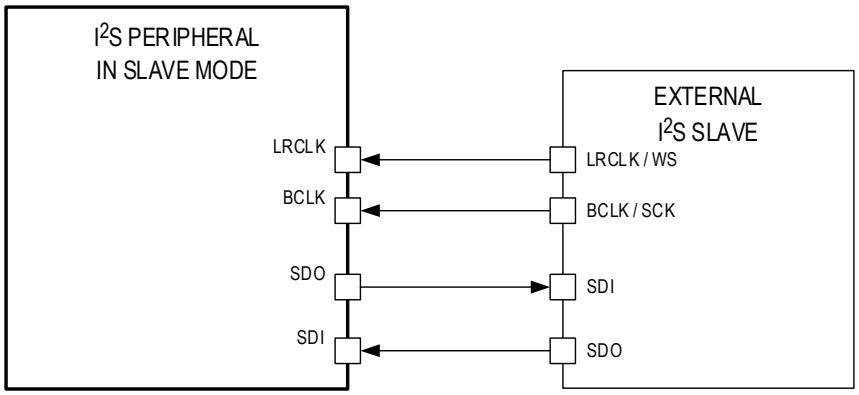


Figure 11-2 shows the I²S peripheral configured for slave operation. The LRCLK and BCLK signals are generated externally and are inputs to the I²S peripheral.

Figure 11-2: I²S Slave Mode



11.3 Master and Slave Mode Configuration

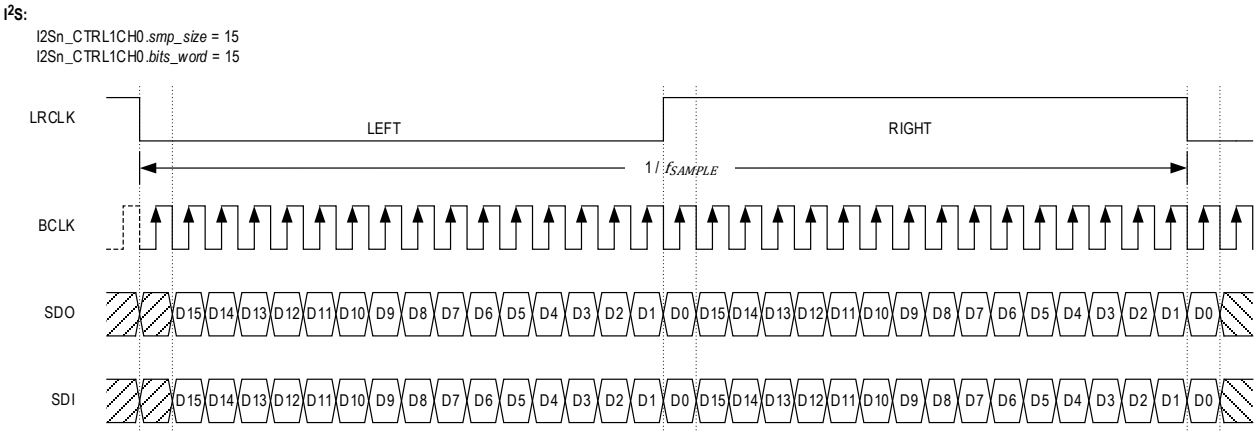
The device supports master and slave modes. In master mode, the BCLK and LRCLK signals are generated internally and output on the BCLK and LRCLK pins. In slave mode, the BCLK and LRCLK pins are configured as inputs, and the external clock source controls the peripheral timing.

Table 11-3: I²S Mode Configuration

Device Mode	*I2S_CTRL0CH0.ch_mode*	LRCLK	BCLK
Master	0	Output to slave	Output to slave
Slave	3	Input from master	Input from master

11.4 Clocking

Figure 11-3: 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 master, the BCLK and LRCLK signals are generated internally by the peripheral using the ERFO. See *Table 11-2* for details of the I²S pin mapping and alternate function selection. If using the I²S peripheral in master mode, the ERFO must generate the BCLK and LRCLK signals.

When the I²S peripheral is configured in slave mode, the BCLK and LRCLK pins must be configured as inputs. An external master generates the BCLK and LRCLK signals, which the peripheral uses to synchronize itself to the I²S bus. *Figure 11-3* 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 11-1*.

Equation 11-1: CD Audio Bit Frequency Calculation

$$f_{BCLK} = 44.1\ kHz \times 16 \times 2 = 1.4112MHz$$

11.4.1 BCLK Generation for Master Mode

As indicated by *Equation 11-1*, the requirements for determining the BCLK frequency are:

1. Audio sample frequency
2. Number of bits per sample, referred to as sample width

Equation 14-2 shows the formula to calculate the bit clock frequency for a given audio file using the above requirements.

Equation 11-2: Calculating the Bit Clock Frequency for Audio

$$f_{BCLK} = f_{SAMPLE} \times Sample\ Width \times 2$$

In master 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 11-3*.

Equation 11-3: Master Mode BCLK Generation Using the I²S External Clock

$$f_{BCLK} = \frac{f_{ERFO}}{(I2Sn\_CTRL1CH0.clkdiv + 1) \times 2}$$

Use *Equation 11-4* to determine the I²S clock divider for a target BCLK frequency.

Equation 11-4: Master Mode Clock Divisor Calculation for a Target Bit Clock Frequency

$$I2Sn\_CTRL1CH0.clkdiv = \frac{f_{ERFO}}{2 \times f_{BCLK}} - 1$$

11.4.2 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 11-3*. 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 11-5: 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 is set to operate as a master. The LRCLK frequency calculation is shown in *Equation 11-6*.

Equation 11-6: LRCLK Frequency Calculation

$$f_{LRCLK} = f_{BCLK} \times (I2Sn\_CTRL1CH0.bits\_word + 1)$$

11.5 Data Formatting

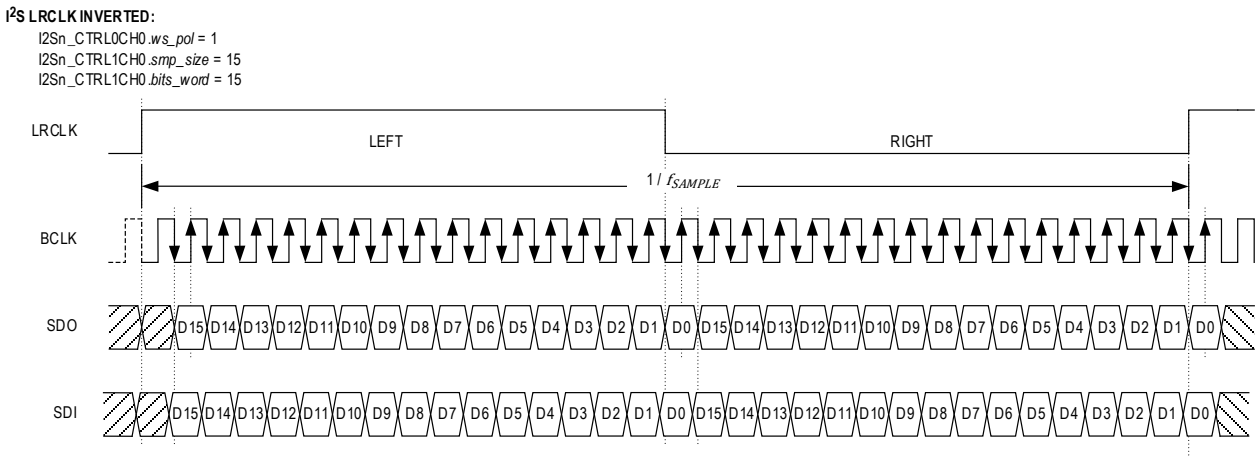
11.5.1 Sample Size

The sample size field, *I2S_CTRL1CH0.smp_size*, 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_size* to 15 collects 16 samples. See *Figure 11-6* for an example of the bits per word field's setting compared to the sample size field's setting.

11.5.2 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 11-3*. 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 11-4*.

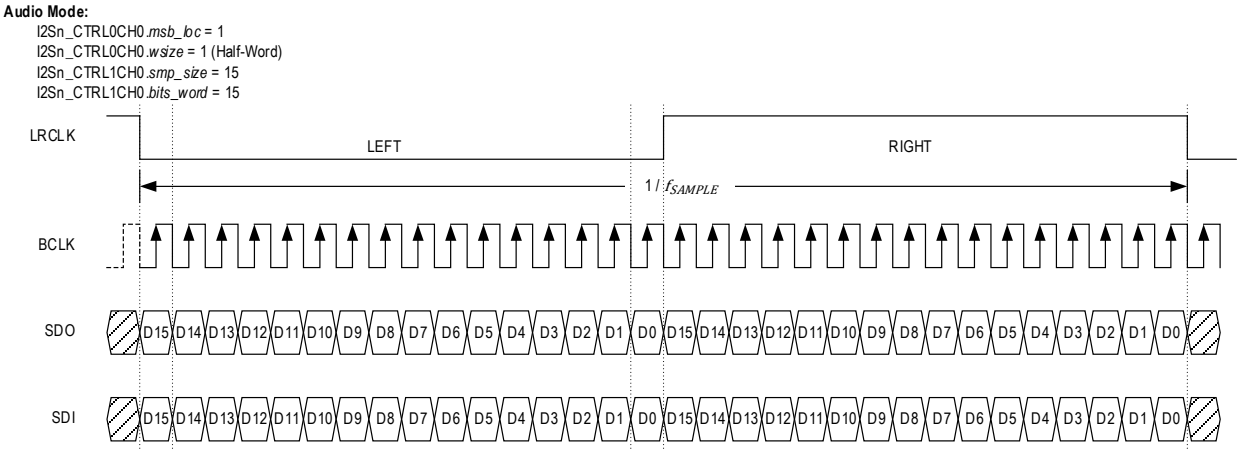
Figure 11-4: Audio Mode with Inverted Word Select Polarity



11.5.3 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 11-3* 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 11-5*. Set *I2S_CTRL0CH0.msb_loc* to 1 to left justify the data with respect to the LRCLK.

Figure 11-5: Audio Master Mode Left-Justified First Bit Location



11.5.4 Sample Adjustment

When the sample size field, *I2S_CTRL1CH0.smp_size*, is less than the bits per word field, *I2S_CTRL1CH0.bits_word*, use the *I2S_CTRL1CH0.adjust* 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 11-6* shows an example of the default adjustment, MSB, where *I2S_CTRL1CH0.smp_size* = 7 and *I2S_CTRL1CH0.bits_word* = 15. *Figure 11-7* shows the adjustment set to the LSB of the SDI/SDO data.

Figure 11-6: MSB Adjustment when Sample Size is Less Than Bits Per Word

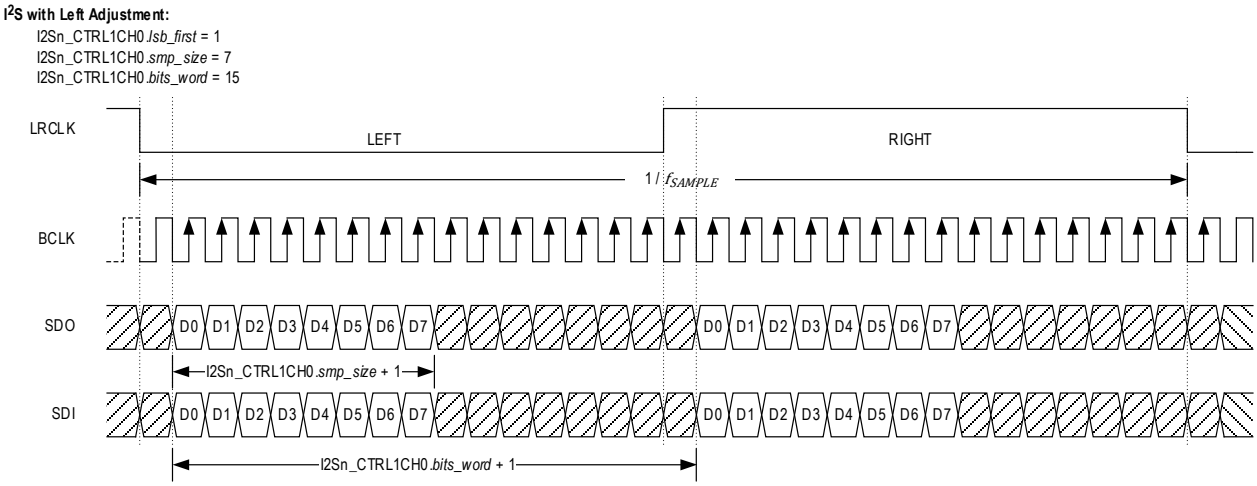
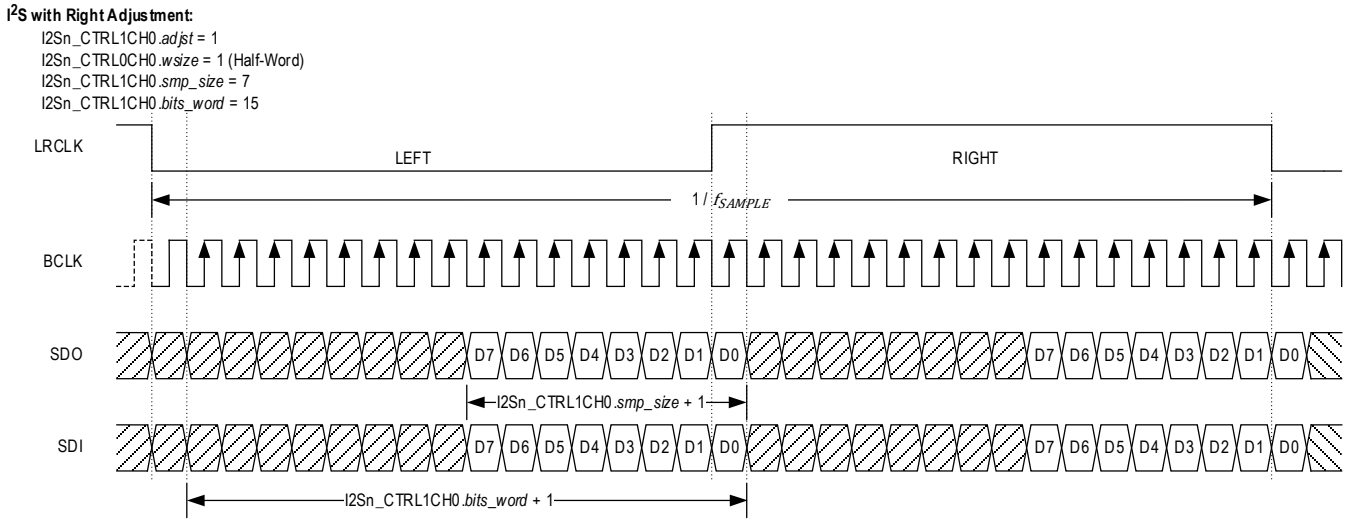


Figure 11-7: LSB Adjustment when Sample Size is Less Than Bits Per Word



11.5.5 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 11-8: I²S Mono Left Mode

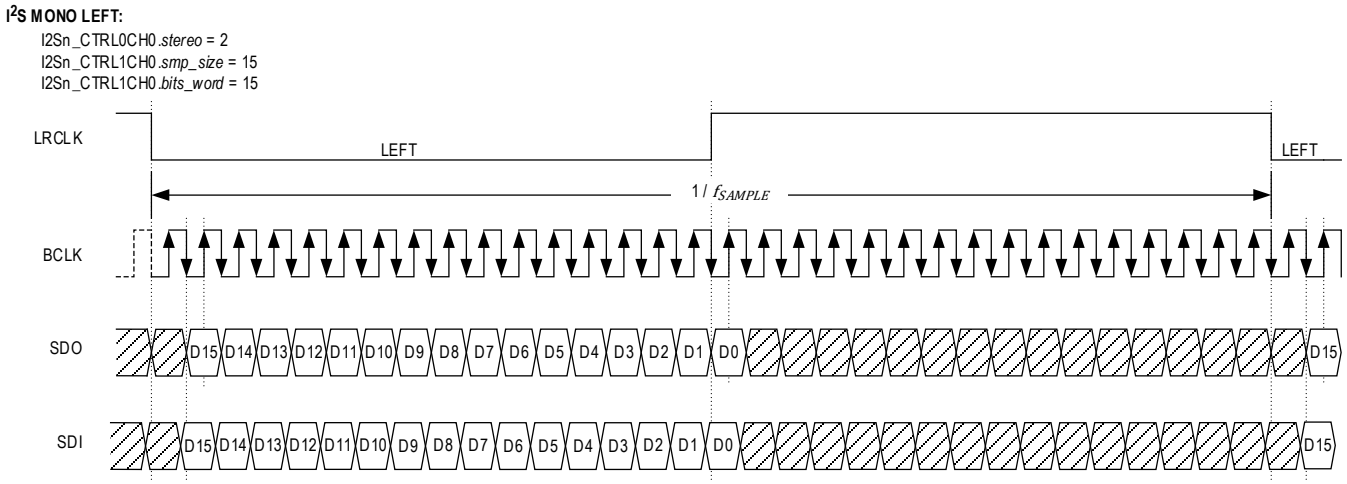
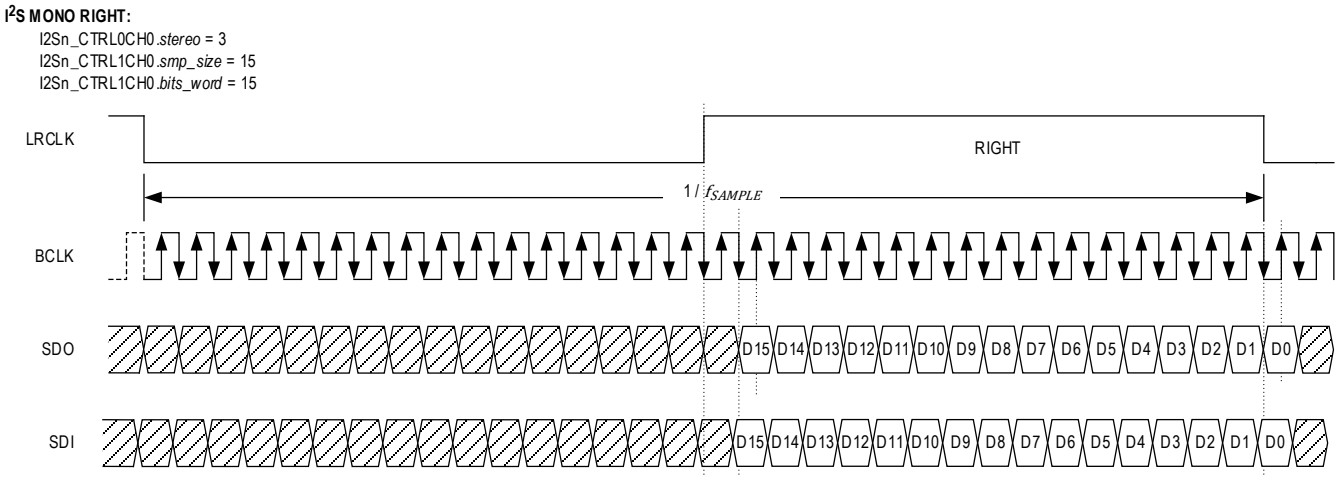


Figure 11-9: I²S Mono Right Mode



11.6 Transmit and Receive FIFOs

11.6.1 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.

11.6.2 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 was written to the transmit FIFO. Use the I²S interrupt flags to monitor the transmit FIFO status and determine when the transfer cycle(s) have been completed.

If the transmit FIFO becomes empty, an error condition occurs and results in undefined behavior.

11.6.3 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.

11.6.4 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 11-4: 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 11-5: 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 11-6: 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

11.6.5 FIFO Data Alignment

The I²S 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 is 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.

11.6.6 Typical Audio Configurations

Table 11-7 shows the relationship between the bits per word field and the sample size field. *Equation 11-7* shows the required relationship between the sample size field and the bits per word field.

Equation 11-7: Sample Size Relationship Bits per Word

$$I2Sn\_CTRL1CH0.smp\_size \leq I2Sn\_CTRL1CH0.bits\_word$$

The *I2S_CTRL1CH0.bits_word* column in *Table 11-7* 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 11-7: 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-bit / 16	8	8	7	7	0	
16-bit / 32	16	16	15	15	1	
20-bit / 40	20	20	19	19	2	sign
24-bit / 48	24	24	23	23	2	sign
24-bit / 64	32	24	31	23	2	sign
32-bit / 64	32	32	31	31	2	

Audio Sample Width/ Samples per WS Cycle	$\frac{\#\ BCLK}{Channel}$	$\frac{\#\ Samples}{Channel}$	I2S_CTRL1CH0			Sign extension (align = 1)†
			bits_word	smp_size	wsize	

† Sign Extension only applies when I2S_CTRL0CH0.align is set to 1 and I2S_CTRL1CH0.smp_size is less than the FIFO width size setting.

11.7 Interrupt Events

The I²S peripheral generates interrupts for the events shown in Table 11-8. 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 11-8. 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

11.7.1 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 has been shifted into the FIFO. The hardware automatically sets the I2S_INTFL.rx_ov_ch0 field to 1 when this event occurs.

11.7.2 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.

11.7.3 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 11-8. 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 11-8: Transmit FIFO Half-Empty Condition

$$I2Sn\_DMACH0.tx\_lvl < \left(\frac{TX\ FIFO\ DEPTH}{2}\right)$$

11.7.4 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.

11.8 Direct Memory Access

The I²S supports DMA for both transmit and receive; separate DMA channels can be connected 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.

11.9 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.i2s* to 0 to enable the I²S peripheral clock source shown in *Table 11-1*.
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_CTRL0CH0.flush* to 1 to flush the FIFO buffers.
5. Configure the *I2S_CTRL0CH0.ch_mode* to select the master or slave configuration.
    a. For master 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.adjust* 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_CTRL0CH0.rx_thd_val*. 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 master operation, load data in the transmit FIFO for transmit.
11. Re-enable the bit clock by setting *I2S_CTRL1CH0.en* to 1.

11.10 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 11-9*. 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 11-9: I²S Register Summary

Offset	Register Name	Description
[0x0000]	*I2S_CTRL0CH0*	I²S Global Mode Control 0 Register

Offset	Register Name	Description
[0x0010]	*I2S_CTRL1CH0*	I²S Master 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

11.10.1 Register Details

Table 11-10: 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
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)

I²S Control 0 Register			I2S_CTRL0CH0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
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
7:6	ch_mode	R/W	0	**Mode** Set this field to indicate master or slave I²S operation. When using master mode, the ERFO must be used to generate the LRCLK/BCLK signals. 0: Master mode, internal generation of LRCLK/BCLK using the ERFO. 1-2: Reserved for Future Use 3: Slave 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 11-11: I²S Master Mode Configuration Register

I²S Master 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 Master Mode* for detailed information. *Note: This field only applies when the I²S peripheral is set to master mode, I2S_CTRL0CH0.ch_mode = 0.*

I²S Master Mode Configuration			I2S_CTRL1CH0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
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 master mode operation, this field is used to start generating the I²S LRCLK and BCLK outputs. In slave 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 11-12: 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 11-13: 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 11-14: 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 11-15: 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

12. 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.

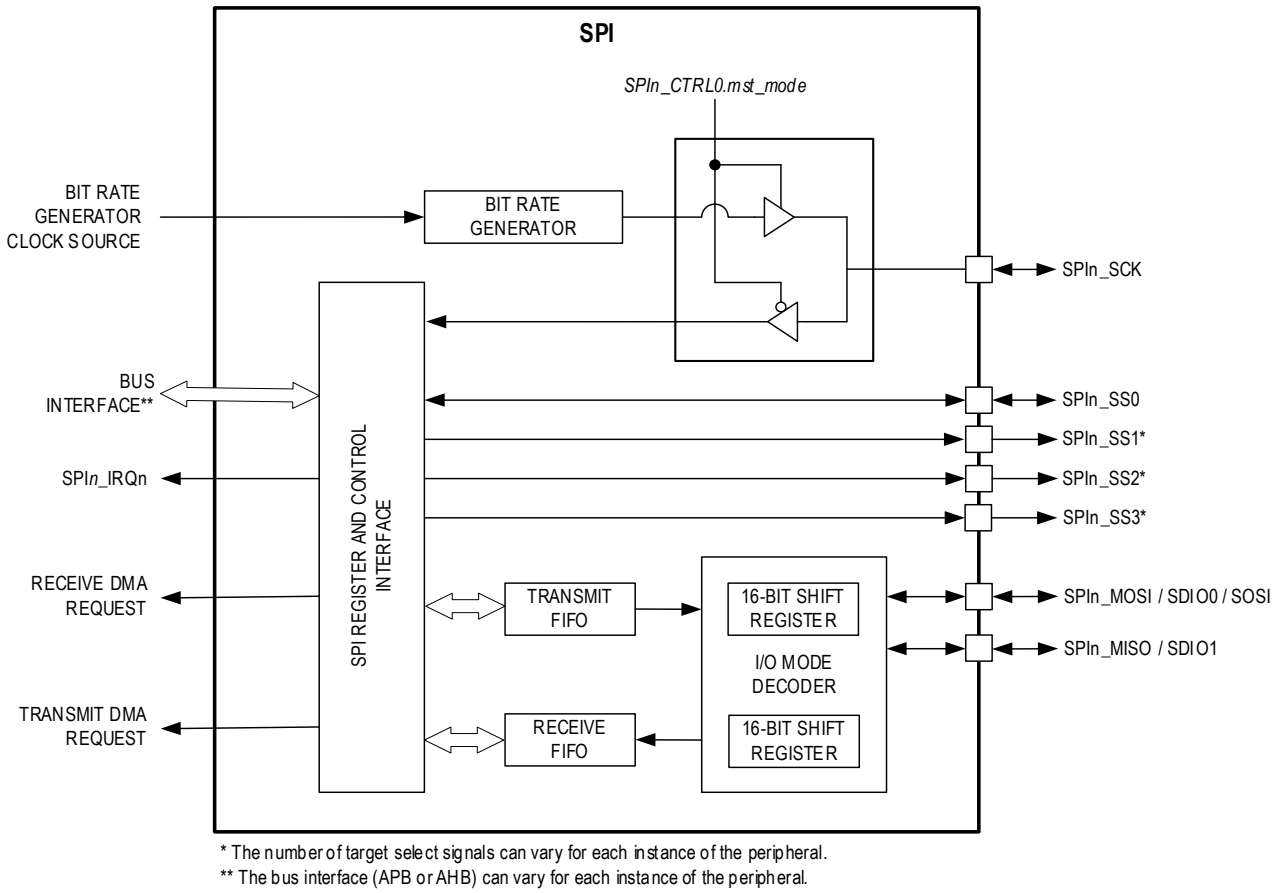
The provided SPI ports support full-duplex, bi-direction I/O, and each SPI includes a bit rate generator (BRG) for generating the clock signal when operating in controller mode. Each SPI port operates independently and requires minimal processor overhead. All instances of the SPI peripheral support both 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.
- Multi-controller mode fault detection.

Figure 12-1 shows the block diagram of the peripheral. See *Table 12-1* for the peripheral-specific bus assignment and bit rate generator clock source.

Figure 12-1: SPI Block Diagram



12.1 Instances

The following instances of the peripheral are provided.

Table 12-1: MAX32672 SPI Instances

Name	Formats				Bus Assignment	Bit Rate Generator Frequency ($f_{SPIn\_INPUT\_CLK}$)
	Three-Wire	Four-Wire	Dual	Quad		
SPI0	Yes	Yes	Yes	No	APB	f_{PCLK}
SPI1	Yes	Yes	Yes	No	APB	f_{PCLK}
SPI2	Yes	Yes	Yes	No	APB	f_{PCLK}

Note: For SPI instance availability and signal mapping, refer to the device data sheet.

12.2 Formats

12.2.1 Four-Wire SPI

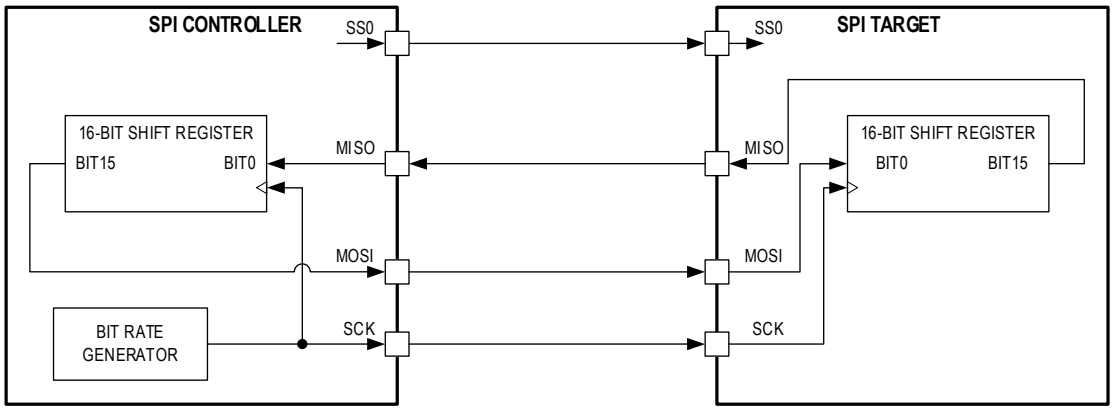
SPI devices operate as either a controller or a target device. In four-wire SPI, four signals are required for communication, as shown in *Table 12-2*.

Table 12-2: 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 12-2: Four-Wire SPI Connection Diagram



12.2.2 Three-Wire SPI

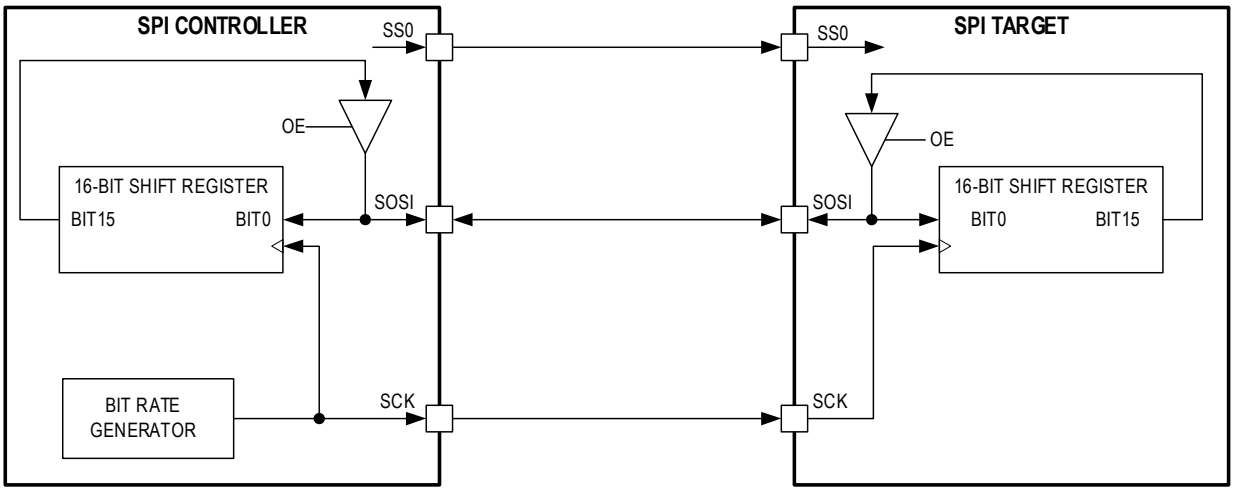
The signals in three-wire SPI operation are shown in *Table 12-3*. The MOSI signal is used as a bi-directional, half-duplex I/O referred to as target input target output (SISO). Three-wire SPI also uses a serial clock signal generated by the controller and a target select pin controlled by the controller.

Table 12-3: 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 12-3*. 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 12-3: Three-Wire SPI Controller to Target Connection



12.3 Pin Configuration

Before configuring the SPI peripheral, first, disable any SPI activity for the port by clearing the *SPIn_CTRL0.en* field to 0.

12.3.1 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.

12.3.2 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.

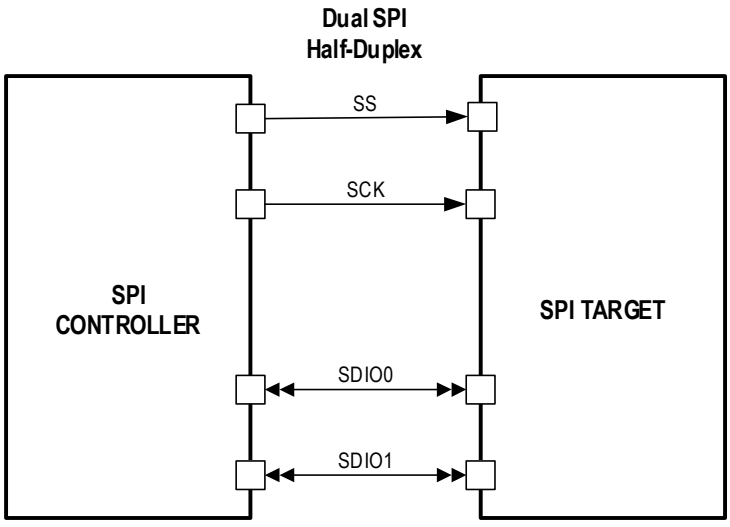
12.3.3 Three-Wire Format Configuration

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.

12.3.4 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 12-4*. 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 12-4: Dual Mode SPI Connection Diagram



12.4 Configuration

12.4.1 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.

12.4.2 Peripheral Clock

The SPI peripheral clock for each SPI port is shown in *Table 12-1*. 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 12-1*. Valid settings for *SPIn_CLKCTRL.clkdiv* are 0 to 8, allowing a divisor of 1 to 256.

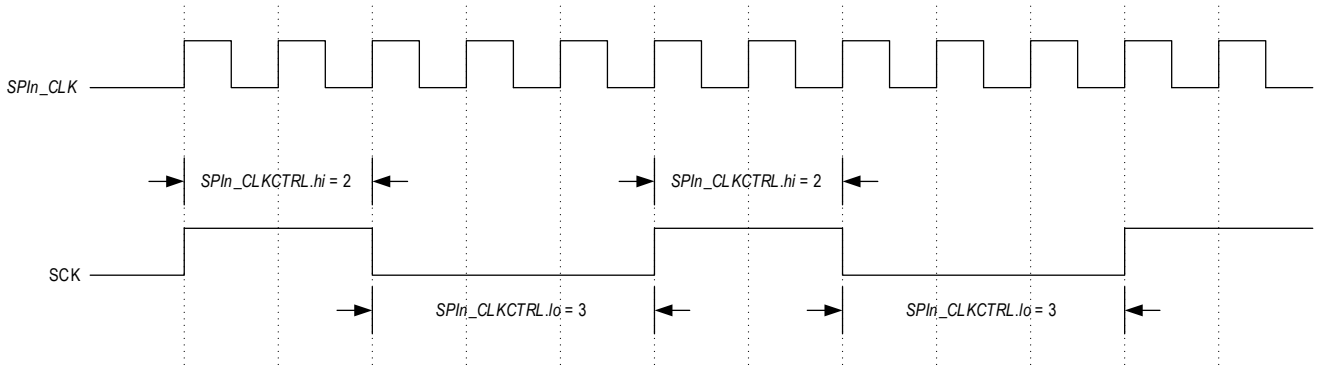
Equation 12-1: SPI Peripheral Clock

$$f_{SPIn\_CLK} = \frac{f_{SPIn\_INPUT\_CLK}}{2^{scale}}$$

12.4.3 Controller Mode Serial Clock Generation

In controller and multi-controller mode, 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_{SPIn_CLK} clocks. *Figure 12-5* 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 12-2* and *Equation 12-3* for calculating the SCK high and low time from the *SPIn_CLKCTRL.hi* and *SPIn_CLKCTRL.lo* field values.

Figure 12-5: SCK Clock Rate Control



Equation 12-2: SCK High Time

$$t_{SCK\_HI} = t_{SPIn\_CLK} \times SPIn\_CLKCTRL.hi$$

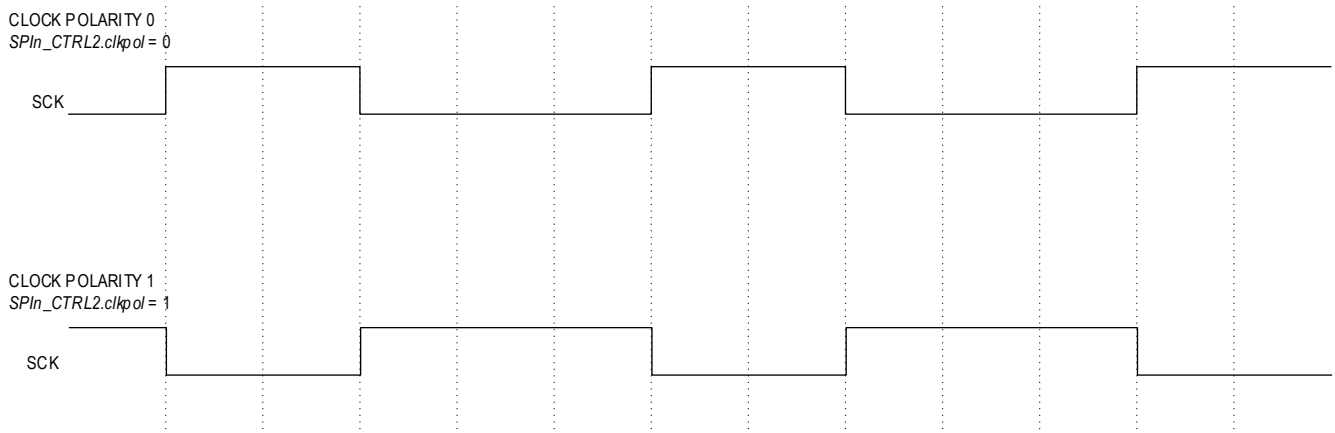
Equation 12-3: SCK Low Time

$$t_{SCK\_LOW} = t_{SPIn\_CLK} \times SPIn\_CLKCTRL.lo$$

12.4.4 Clock Phase and Polarity Control

SPI supports four combinations of clock and phase polarity, as shown in *Table 12-4*. 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 12-6*. 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 12-6: 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.

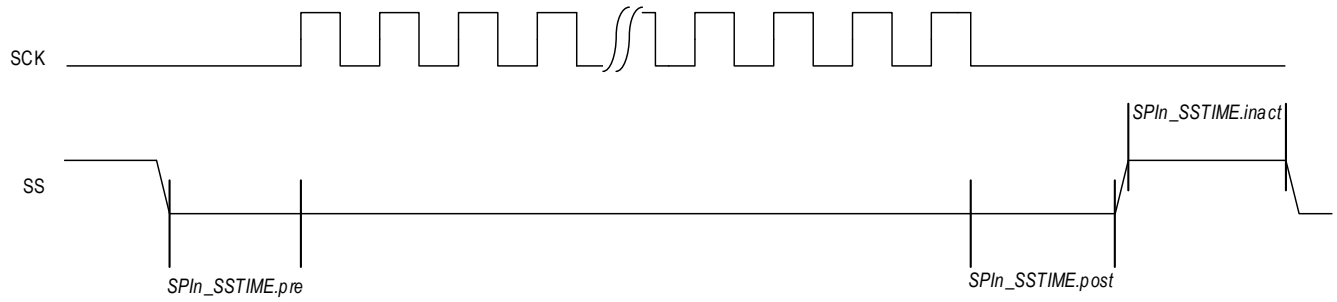
Table 12-4: SPI Modes Clock Phase and Polarity Operation

SPI Mode	*SPIn_CTRL2 .clkpol*	*SPIn_CTRL2 .clkpha*	SCK Receive Edge	SCK Transmit Edge	SCK Idle State	SCLK ≥ 20MHz
0	0	0	Rising	Falling	Low	*SPIn_CTRL2.sclk_fb_inv* = 1
1	0	1	Falling	Rising	Low	*SPIn_CTRL2.sclk_fb_inv* = 0
2	1	0	Rising	Falling	High	*SPIn_CTRL2.sclk_fb_inv* = 1
3	1	1	Falling	Rising	High	*SPIn_CTRL2.sclk_fb_inv* = 0

12.4.5 Target Select Configuration

The SPI supports additional controller mode configuration for fine-tuning the target select lines timing with respect to the time between SPI transactions and how many clock cycles between target select going active and the first SCK transition and the last SCK transition to target select going inactive. The register fields for controlling each of these portions of the target control signal (SPIn_SS) are shown in *Figure 12-7*. Each of these fields selects the number of system clocks for the delay from 1 to 256. Each of these fields defaults to the maximum setting of 256 system clocks.

Figure 12-7: Target Select Configuration Using SPIn_SSTIME Register



12.4.6 Transmit and Receive 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 both the odd and even bytes to be written to this register space before dequeuing the 16-bit result to the FIFO.

The receive FIFO hardware is 32 bytes deep. Read data width can be 8-, 16- or 32-bits. A byte read from this register dequeues one byte from the FIFO. A 16-bit read from this register dequeues two bytes from the FIFO, least significant byte first. A 32-bit read from this register dequeues four bytes from the FIFO, least significant byte first.

12.4.7 Interrupts and Wakeups

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 and external signals. The following transmission interrupts are supported:
  - SS asserted or deasserted.
  - SPI transaction complete.
    - Controller mode only.
  - Target mode transaction aborted.
  - Multi-controller 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.

12.5 SPI Registers

See *Table 3-2* for this peripheral/module's base address. If multiple peripheral instances are provided, each instance has its own independent set of registers, shown in *Table 12-5*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 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 12-5: SPI Register Summary

Offset	Register Name	Description
[0x0000]	*SPIn_FIFO32*	*SPI FIFO Data Register*
[0x0000]	*SPIn_FIFO16*	*SPI 16-bit FIFO Data Register*
[0x0000]	*SPIn_FIFO8*	*SPI 8-bit FIFO Data Register*
[0x0004]	*SPIn_CTRL0*	*SPI Controller Signals Control Register*
[0x0008]	*SPIn_CTRL1*	*SPI Transmit Packet Size Register*
[0x000C]	*SPIn_CTRL2*	*SPI Static Configuration Register*
[0x0010]	*SPIn_SSTIME*	*SPI Target Select Timing Register*
[0x0014]	*SPIn_CLKCTRL*	*SPI Controller Clock Control Register*
[0x001C]	*SPIn_DMA*	*SPI DMA Control Register*
[0x0020]	*SPIn_INTFL*	*SPI Interrupt Flag Register*
[0x0024]	*SPIn_INTEN*	*SPI Interrupt Enable Register*
[0x0028]	*SPIn_WKFL*	*SPI Wakeup Flags Register*
[0x002C]	*SPIn_WKEN*	*SPI Wakeup Enable Register*
[0x0030]	*SPIn_STAT*	*SPI Status Register*

12.5.1 Register Details

Table 12-6: SPI FIFO Data Register

SPI FIFO Data Register			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. Read from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 12-7: SPI 16-bit FIFO Register

SPI FIFO Data			SPIn_FIFO16	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15:0	data	R/W	0	**SPI 16-bit 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 width only for 16-bit FIFO access. Reading from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 12-8: SPI 8-bit FIFO Register

SPI 8-bit FIFO Data			SPIn_FIFO8	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	R/W	0	**Reserved**
7:0	data	R/W	0	**SPI 8-bit 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 for 8-bit FIFO access. Reading from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 12-9: SPI Control 0 Register

SPI Control 0 Register			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**
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**

SPI Control 0 Register			SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
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. Ensure that all pending transactions are 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 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 12-10: SPI Transmit Packet Size Register

SPI Transmit Packet Size Register			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 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 12-11: SPI Control 2 Register

SPI Control 2 Register			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**

SPI Control 2 Register			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
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 communication. ***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. 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: Reserved. 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 (not supported). 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 dual mode SPI, the character size should be divisible by the number of bits per SCK cycle.*
7:5	-	RO	0	**Reserved**
4	sclk_fb_inv	R/W	0	**Invert SCLK Feedback in Controller Mode** Set this bit to 1 to invert the SCLK feedback in controller mode for modes 0 and 2 if operating at an SCLK rate ≥ 20MHz. This field must be set to 0 for modes 1 and 3. 0: SCLK feedback is not inverted. 1: SCLK feedback is inverted.

SPI Control 2 Register			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
3:2	-	RO	0	**Reserved**
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 12-12: SPI Target Select Timing Register

SPI Target Select Timing Register			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. 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).*
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. 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).*

SPI Target Select Timing Register			SPIn_SSTIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
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 12-13: SPI Controller Clock Control Registers

SPI Controller Clock Control Register				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 by 2^{scale} to generate the SPI peripheral clock. See *Table 12-1* for details on the SPI input clock. $f_{SPIn\_CLK} = \frac{f_{SPIn\_INPUT\_CLK}}{2^{scale}}$ 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_{SPIn_CLK}, 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 12-14: SPI DMA Control Registers

SPI DMA Control Register			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 and reserved.*
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 12-15: SPI Interrupt Status Flags Registers

SPI Interrupt Status Flags Register			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 have been completed. *SPIn_CTRL1.tx_num_char* has been 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, multi-controller 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**
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 12-16: SPI Interrupt Enable Registers

SPI Interrupt Enable Register			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: Disabled. 1: Enabled.

SPI Interrupt Enable Register			SPIn_INTEN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
14	rx_ov	R/W	0	**Receive FIFO Overrun Interrupt Enable** 0: Disabled. 1: Enabled.
13	tx_un	R/W	0	**Transmit FIFO Underrun Interrupt Enable** 0: Disabled. 1: Enabled.
12	tx_ov	R/W	0	**Transmit FIFO Overrun Interrupt Enable** 0: Disabled. 1: Enabled.
11	mst_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 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_thd	R/W	0	**Receive FIFO Threshold Level Crossed Interrupt Enable** 0: Disabled. 1: Enabled.
1	tx_em	R/W	0	**Transmit FIFO Empty Interrupt Enable** 0: Disabled. 1: Enabled.
0	tx_thd	R/W	0	**Transmit FIFO Threshold Level Crossed Interrupt Enable** 0: Disabled. 1: Enabled.

Table 12-17: SPI Wakeup Status Flags Registers

SPI Wakeup Flags Register			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: Normal operation. 1: Wake condition occurred.

SPI Wakeup Flags Register			SPIn_WKFL	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
2	rx_thd	R/W1C	0	**Wake on Receive FIFO Threshold Level Crossed Flag** 0: Normal operation. 1: Wake condition occurred.
1	tx_em	R/W1C	0	**Wake on Transmit FIFO Empty Flag** 0: Normal operation. 1: Wake condition occurred.
0	tx_thd	R/W1C	0	**Wake on Transmit FIFO Threshold Level Crossed Flag** 0: Normal operation. 1: Wake condition occurred.

Table 12-18: SPI Wakeup Enable Registers

SPI Wakeup Enable Register			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 12-19: SPI Target Select Timing Registers

SPI Status Register			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: SPIn_CTRL0, SPIn_CTRL1, SPIn_CTRL2, SPIn_SSTIME, and SPIn_CLKCTRL should not be configured if this field is set.*

13. 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.
- 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 pulse-width modulated (PWM) signal generation.
- Multiple clock source options.

Instances denoted as LPTMR, shown in *Table 13-1*, 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: the timer outputs a signal until a terminal value is reached and then toggles the output signal until a second terminal value is reached and then repeats.
- 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.

13.1 Instances

Instances of the peripheral are listed in *Table 13-1*. Both the TMR and LPTMR are functionally very similar, so for convenience, they are referred to as just TMR. The LPTMR instances can function while the device is in *DEEPSLEEP* and *BACKUP*. If supported, TMR instances can operate in dual 16-bit mode or cascaded 32-bit mode. LPTMR instances provide a single 32-bit timer and can select clock sources that are available in *DEEPSLEEP* and *BACKUP*.

Table 13-1. MAX32672 TMR/LPTMR

Instance	Register Access Name	Single 32-bit Mode	Cascade 32-Bit Mode	Dual 16-Bit Mode	Power Modes	Clock Option 0	Clock Option 1	Clock Option 2	Clock Option 3
TMR0	TMR0	NO	YES	YES	*ACTIVE, SLEEP*	PCLK	EXT_CLK1	IBRO	ERFO
TMR1	TMR1								
TMR2	TMR2								
TMR3	TMR3								
LPTMR0	TMR4	YES	NO	NO	*ACTIVE, SLEEP*	AOD_CLK	EXT_CLK2	ERTCO	INRO
					DEEPSLEEP, BACKUP	N/A	N/A	ERTCO	INRO
LPTMR1	TMR5	YES	NO	NO	*ACTIVE, SLEEP*	AOD_CLK	EXT_CLK2	ERTCO	INRO
					DEEPSLEEP, BACKUP	N/A	N/A	ERTCO	INRO

Table 13-2: MAX32672 TMR/LPTMR Instances Capture Events

Instance	Capture Event 0
TMR0	Timer Input Pin
TMR1	Timer Input Pin
TMR2	Timer Input Pin
TMR3	Timer Input Pin
LPTMR0	Timer Input Pin
LPTMR1	Timer Input Pin

13.2 Basic Timer Operation

The timer modes operate by incrementing the *TMRn_CNT* register, driven by either the timer clock, an external stimulus on the timer pin, or a combination of both. The *TMRn_CNT* register is always readable, even while the timer is enabled and counting.

Each timer mode has a user-configurable timer period, which terminates on the timer clock cycle following the end of timer period condition. Each timer mode has a different response at the end of a timer period, which can include changing the state of the timer pin, capturing a timer value, reloading *TMRn_CNT* with a new starting value, or disabling the counter. The end of a timer period always sets the corresponding interrupt bit and can generate an interrupt if enabled.

In most modes, the timer peripheral automatically sets *TMRn_CNT* to 0x0000 0001 at the end of a timer period, but *TMRn_CNT* is set to 0x0000 0000 following a system reset. This means the first timer period following a system reset is one timer clock longer than subsequent timer periods if *TMRn_CNT* is not initialized to 0x0000 0001 during the timer configuration step.

13.3 32-Bit Single/32-Bit Cascade/Dual 16-Bit

Most instances contain two 16-bit timers, which can support combinations of single or cascaded 32-bit modes, and single or dual 16-bit modes, as shown in *Table 13-1*. 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 13-3*. 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 13-3: TimerA/TimerB 32-Bit Field Allocations

Register	32-Bit Single/Cascade Mode	Dual 16-Bit Mode		Single 16-Bit Mode
Timer Counter	TimerA Count = *TMRn_CNT[31:0]*	TimerA Compare = *TMRn_CNT[15:0]*	TimerB Count = *TMRn_CNT[31:16]*	TimerA Compare = *TMRn_CNT[15:0]*
Timer Compare	TimerA Compare = *TMRn_CMP[31:0]*	TimerA Compare = *TMRn_CMP[15:0]*	TimerB Compare = *TMRn_CMP[31:16]*	TimerA Compare = *TMRn_CMP[15:0]*
Timer PWM	TimerA Count = *TMRn_PWM[31:0]*	TimerA Count = *TMRn_PWM[15:0]*	TimerB Count = *TMRn_PWM[31:16]*	TimerA Count = *TMRn_PWM[15:0]*

13.4 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 13-1*. 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_CTRL0.clkdiv* 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.

Equation 13-1: Timer Peripheral Clock Equation

$$f_{CNT\_CLK} = \frac{f_{CLK\_SOURCE}}{prescaler}$$

The software configures and controls the timers by reading and writing to the timer 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 can require up to 50% of the timer's internal clock before the hardware recognizes the event.

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 mode or for a low-power timer the *TMRn_CTRL1.clksel_b* field must be set to the same value as the *TMRn_CTRL1.clksel_a* field.
3. Set the desired clock divider by setting the *TMRn_CTRL0.clkdiv* field.
4. Configure the timer for the desired operating mode. See the section *Operating Modes* for details on mode configuration.
5. 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.
6. Enable the timer.
   a. Set *TMRn_CTRL0.en* to 1 to enable the timer.
   b. Read the *TMRn_CTRL0.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.

13.5 Timer Pin Functionality

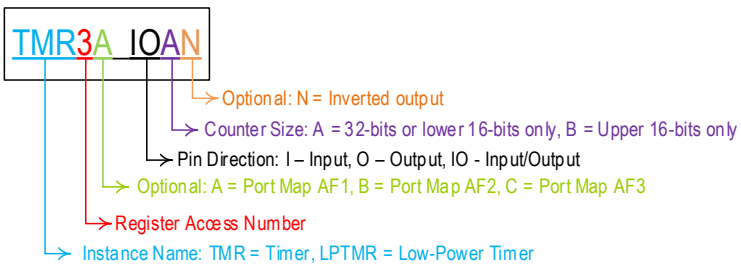
Each timer instance can have an input signal and/or output signal depending on the operating mode. Not all instances of the peripheral are available in all packages. The number of input and output signals per peripheral instance can vary as well. Refer to the data sheet for I/O signal configurations and alternate functions for each timer instance.

The physical pin location of the timer input and/or output signals can vary between packages. The timer functionality, however, is always be 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 and drive strength, 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 *GPIOn_OUT* and the *GPIOn_OUTEN* registers should be configured to match the inactive state of the timer pin for that mode. Consult the *General-Purpose I/O (GPIO) and Alternate Function (AF) Pins* chapter for details on how to configure the electrical characteristics for the pin.

Figure 13-1 shows the timer pin naming convention for the MAX32672 microcontroller.

Figure 13-1: Timer I/O Signal Naming Conventions



The TimerA output controls for modes 0, 1, 3, and 5 output signals are shown in *Figure 13-2*. The TimerA input controls for modes 2, 4, 6, 7, 8, and 14 input signals are shown in *Figure 13-3*.

Figure 13-2: MAX32672 TimerA Output Functionality, Modes 0/1/3/5

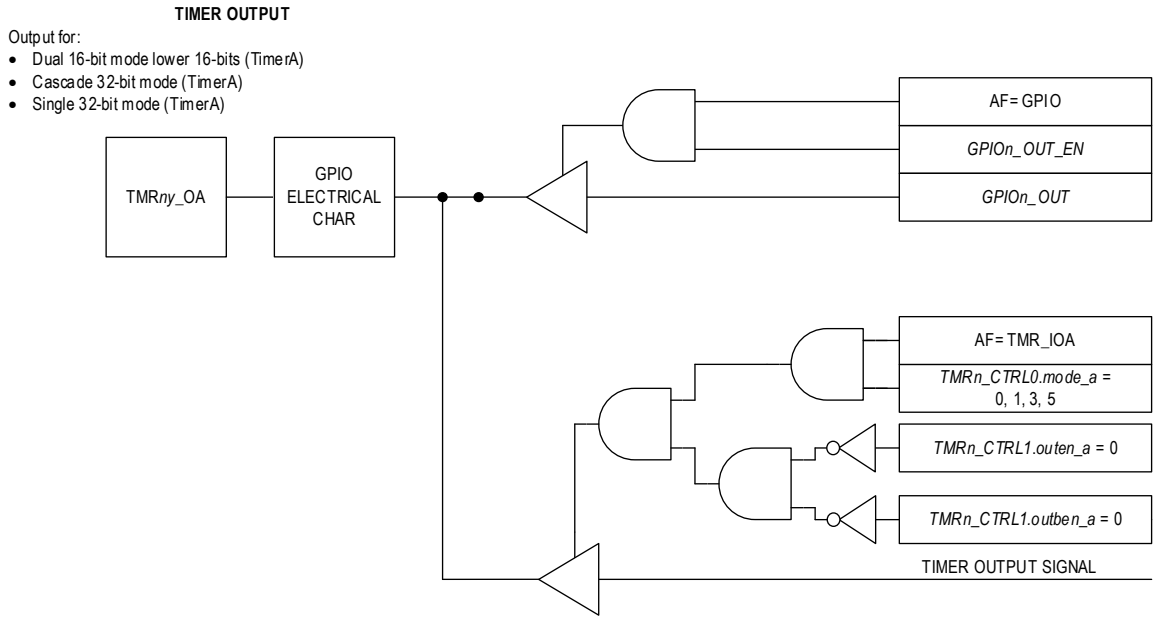
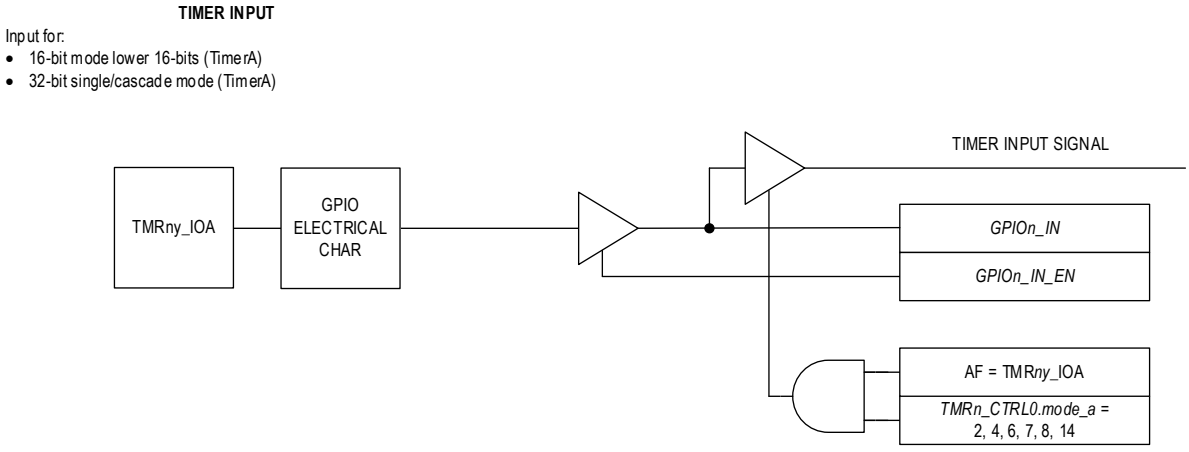


Figure 13-3: MAX32672 TimerA Input Functionality, Modes 2/4/6/7/8/14



13.6 Wakeup Events

In low-power modes, the system clock can be turned off to conserve power. In this case, a wakeup event can be configured to wakeup the clock control logic and re-enable the system clock. The wakeup conditions are the same as the interrupts.

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 wakeup 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 section *Operating Modes*.
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 wakeup event.

13.7 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 data sheet for I/O signal configurations and alternate functions for each timer instance.

In *Table 13-4* and *Table 13-5*, 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 13-1* for details of the timer's naming convention for I/O signals.

Table 13-4: MAX32672 Operating Mode Signals for Timer 0 through Timer 3

Timer Mode	TimerA	I/O Signal Name†	Pin Required
One-Shot Mode (0)	TimerA Output Signal	TMR*ny*_OA/TMR*ny*_IOA	Optional
Continuous Mode (1)	TimerA Output Signal	TMR*ny*_OA/TMR*ny*_IOA	Optional
Counter Mode (2)	TimerA Input Signal	TMR*ny*_IA/TMR*ny*_IOA	Yes
PWM Mode (3)	TimerA Output Signal	TMR*ny*_OA/TMR*ny*_IOA	Yes
Capture Mode (4)	TimerA Input Signal	TMR*ny*_IA/TMR*ny*_IOA	Yes
Compare Mode (5)	TimerA Output Signal	TMR*ny*_OA/TMR*ny*_IOA	Optional
Gated Mode (6)	TimerA Input Signal	TMR*ny*_IA/TMR*ny*_IOA	Yes
Capture/Compare Mode (7)	TimerA Input Signal	TMR*ny*_IA/TMR*ny*_IOA	Yes
Dual Edge Capture Mode (8)	TimerA Input Signal	TMR*ny*_IA/TMR*ny*_IOA	Yes
Reserved (9 - 13)	-	-	-
Inactive Gated Mode (14)	TimerA Input Signal	TMR*ny*_IA/TMR*ny*_IOA	Yes
Reserved (15)	-	-	-

† *See Figure 13-1 for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.*

Table 13-5: MAX32672 Operating Mode Signals for Low-Power Timer 0 (TMR4) and Low-Power Timer 1 (TMR5)

Timer mode	TMR4/TMR5	I/O Signal Name†	Required?
One-Shot Mode (0)	TimerA Output Signal	LPTMR*ny*_OA	Optional
Continuous Mode (1)	TimerA Output Signal	LPTMR*ny*_OA/LPTMR*ny*_IOA	Optional
Counter Mode (2)	TimerA Input Signal	LPTMR*ny*_IA/LPTMR*ny*_IOA	Yes
PWM Mode (3)	TimerA Output Signal	LPTMR*ny*_OA/LPTMR*ny*_IOA	Yes
Capture Mode (4)	TimerA Input Signal	LPTMR*ny*_IA/LPTMR*ny*_IOA	Yes
Compare Mode (5)	TimerA Output Signal	LPTMR*ny*_OA/LPTMR*ny*_IOA	Optional
Gated Mode (6)	TimerA Input Signal	LPTMR*ny*_IA/LPTMR*ny*_IOA	Yes
Capture/Compare Mode (7)	TimerA Input Signal	LPTMR*ny*_IA/LPTMR*ny*_IOA	Yes
Dual Edge Capture Mode (8)	TimerA Input Signal	LPTMR*ny*_IA/LPTMR*ny*_IOA	Yes
Reserved (9 - 13)	-	-	-
Inactive Gated Mode (14)	TimerA Input Signal	LPTMR*ny*_IA/LPTMR*ny*_IOA	Yes
Reserved (15)	-	-	-

† *See Figure 13-1 for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.*

13.7.1 One-Shot Mode (0)

In one-shot mode, the timer peripheral increments the timer's *TMRn_CNT* register until it reaches the timer's *TMRn_CMP* register, and the timer is then disabled. If the timer's output is enabled, the output signal is driven active for one timer source clock cycle. For example, if the timer souce clock (f_{CLK_SOURCE}), is PCLK, the output is driven active for 1 PCLK cycle. One-shot mode provides exactly one timer period and is automatically disabled.

The timer period ends on the timer clock following *TMRn_CNT* = *TMRn_CMP*. The timer peripheral hardware automatically performs the following actions at the end of the timer period:

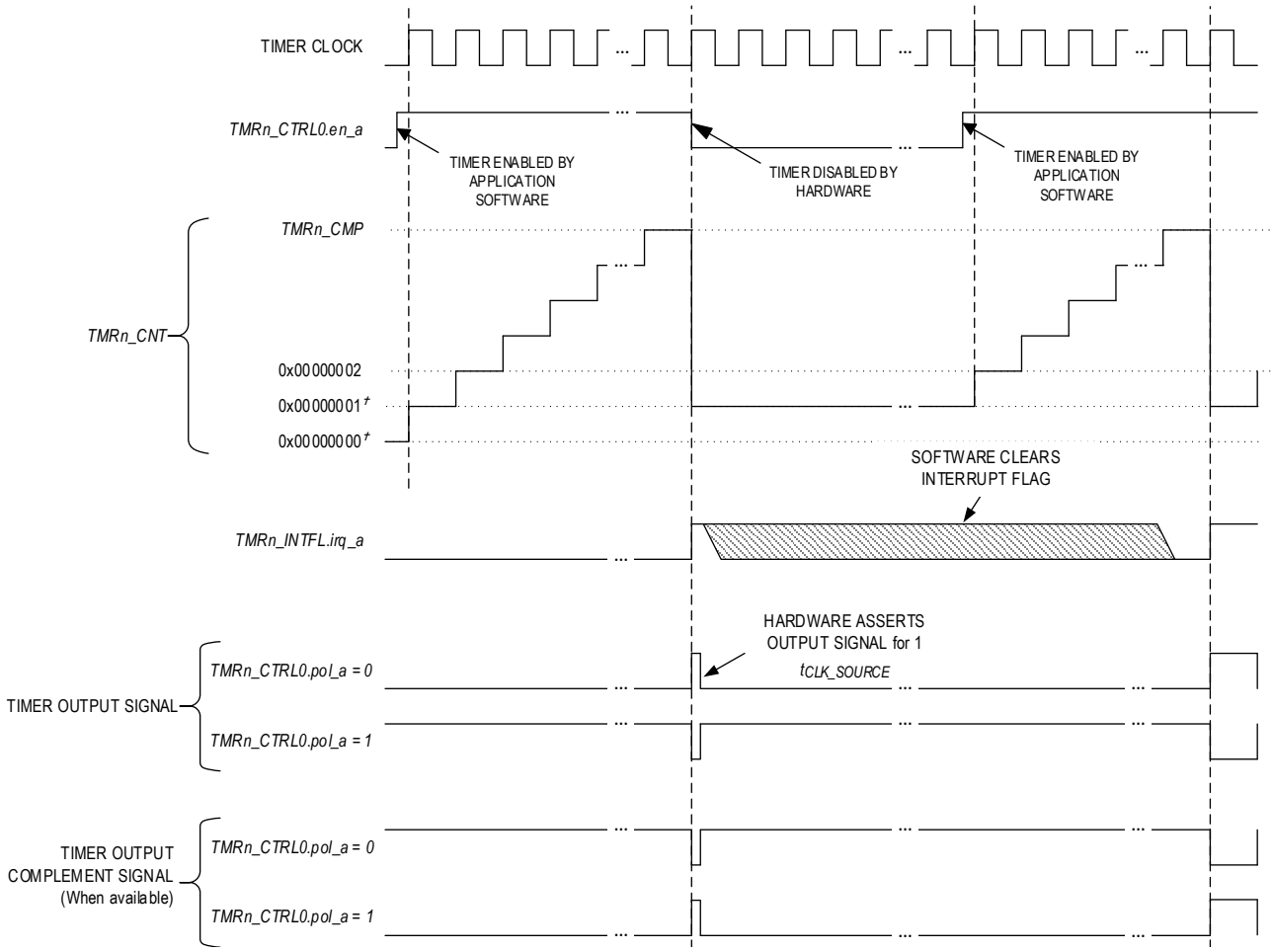
- The *TMRn_CNT* register is set to 0x0000 0001.
- The timer is disabled (*TMRn_CTRL0.en* = 0).
- The timer output, if enabled, is driven to its active state for one timer source clock period.
- The *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

The timer period is calculated using *Equation 13-2*.

Equation 13-2: One-Shot Mode Timer Period in Seconds

$$One - Shot\ mode\ timer\ period = \frac{TMRn\_CMP - TMRn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}(Hz)}$$

Figure 13-4: 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* defaults to 0x00000000 on a timer reset. *TMRn_CNT* 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 the corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to the *TMRn_CMP* register.
   a. Poll the *TMRn_INTFL*.*wrdone* field until it reads 1.
8. If desired, write an initial value to the *TMRn_CNT* register.
   a. This affects only the first period; subsequent timer periods always reset the *TMRn_CNT* register to 0x0000 0001.
9. Enable the timer peripheral as described in *Timer Clock Sources*.

13.7.2 Continuous Mode (1)

In continuous mode, the *TMRn_CNT* register increments until it equals the *TMRn_CMP* register; the *TMRn_CNT* register 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* = *TMRn_CMP*).

The timer peripheral hardware automatically performs the following actions on the timer clock cycle after the period ends:

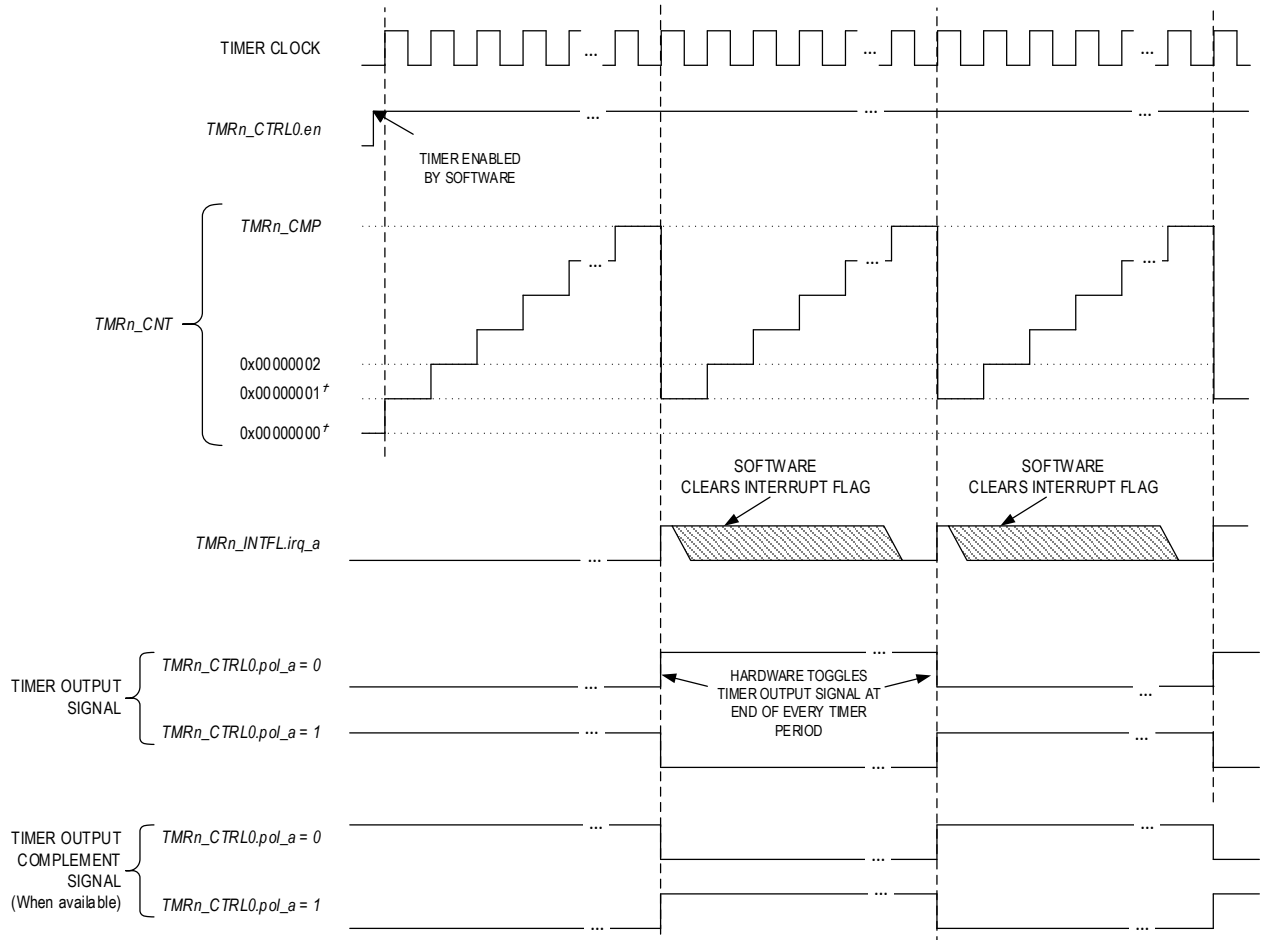
- the *TMRn_CNT* register is set to 0x0000 0001,
- if the timer output signal is toggled, the corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

The continuous mode timer period is calculated using *Equation 13-3*.

Equation 13-3: Continuous Mode Timer Period in Seconds

$$Continuous\ mode\ timer\ period = \frac{TMRn\_CMP - TMRn\_CNT_{INITIAL\_VALUE} + 1}{f_{CNT\_CLK}\ (Hz)}$$

Figure 13-5: 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* defaults to 0x00000000 on a timer reset. *TMRn_CNT* 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.
   d. Set the *TMRn_CTRL1*.*outen* field to 1.
5. If using the timer interrupt, enable the corresponding field in the *TMRn_CTRL1* register.
6. Write the compare value to the *TMRn_CMP* register.
   a. Poll the *TMRn_INTFL*.*wrdone* field until it reads 1.
7. If desired, write an initial value to the *TMRn_CNT* register.
   a. This affects only the first period; subsequent timer periods always reset the *TMRn_CNT* register to 0x0000 0001.
9. Enable the timer peripheral as described in *Timer Clock Sources*.

13.7.3 Counter Mode (2)

In counter mode, the timer peripheral increments the *TMRn_CNT* 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* reaches the *TMRn_CMP* register, the hardware automatically sets the interrupt bit to 1 (*TMRn_INTFL*.*irq)*, sets the *TMRn_CNT* register to 0x0000 0001, and continues incrementing. The timer can be configured to increment on either the rising edge or the 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 in *Equation 13-4*.

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 four PCLK cycles is required for a count to occur. The timer input signal should be greater than four PCLK cycles to guarantee a count is triggered.

Equation 13-4: 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* = *TMRn_CMP*.

The timer peripheral's hardware automatically performs the following actions at the end of the timer period:

- The *TMRn_CNT* register is set to 0x0000 0001.
- The timer output signal is toggled if the timer output pin is enabled.
- The *TMRn_INTFL*.*irq* field is set to 1, indicating a timer interrupt event occurred.
- The timer remains enabled and continues incrementing.

Note: 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.

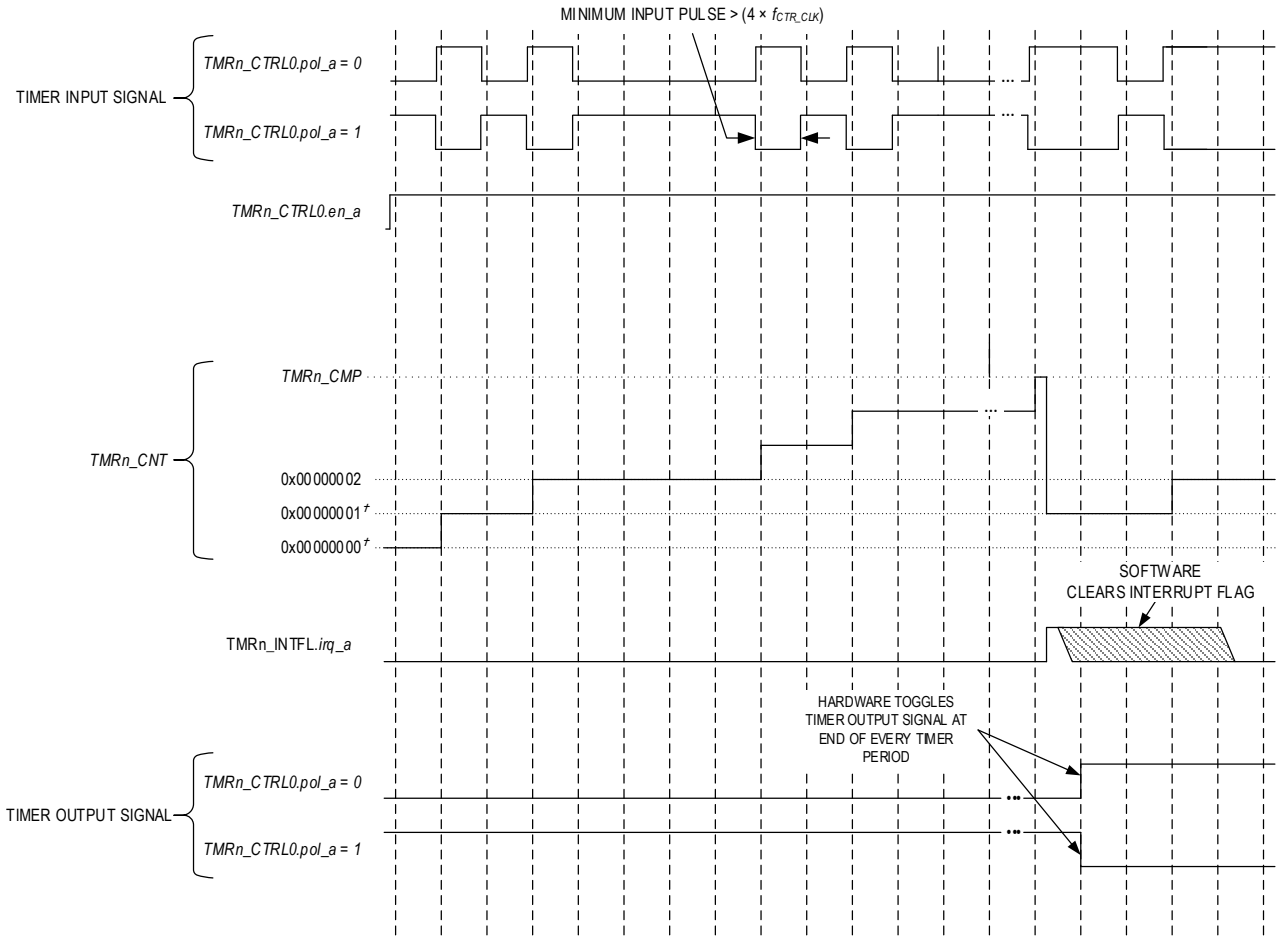
In counter mode, the number of timer input transitions that occurred during a period is equal to the *TMRn_CMP* register's value. Use *Equation 13-5* to determine the number of transitions that have occurred prior to the end of the timer's period.

Note: Equation 13-5 *is only valid during an active timer count prior to the end of the timer's period.*

Equation 13-5: Counter Mode Timer Input Transitions

$$Counter\ mode\ timer\ input\ transitions = TMR\_CNT_{CURRENT\_VALUE}$$

Figure 13-6: 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* defaults to 0x00000000 on a timer reset. *TMRn_CNT* 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* to 2 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. Select the correct alternate function mode for the timer input pin.
5. Write the compare value to *TMRn_CMP*.
   a. Poll the *TMRn_INTFL*.*wrdone* field until it reads 1.
6. If desired, write an initial value to *TMRn_CNT*. This affects only the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

13.7.4 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* register. At the end of the cycle where the *TMRn_CNT* value matches *TMRn_PWM*, the timer output signal toggles state. The timer continues counting until it reaches the *TMRn_CMP* value.

The timer period ends on the rising edge of f_{CNT_CLK} following *TMRn_CNT* = *TMRn_CMP*.

The timer peripheral automatically performs the following actions at the end of the timer period:

- The *TMRn_CNT* 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* value matches the *TMRn_PWM* value, the timer output signal remains high until the *TMRn_CNT* value reaches the *TMRn_CMP*, resulting in the timer output signal transitioning low, and the *TMRn_CNT* value resetting to 0x0000 0001.

When *TMRn_CTRL0*.*pol* = 1, the timer output signal starts high and transitions low when the *TMRn_CNT* value matches the *TMRn_PWM* value, the timer output signal remains low until the *TMRn_CNT* value reaches *TMRn_CMP*, resulting in the timer output signal transitioning high, and the *TMRn_CNT* value resetting to 0x0000 0001.

Complete the following steps to configure a timer for PWM mode and initiate 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.clkdiv* field to set the prescaler that determines the timer frequency.
5. Configure the GPIO as a timer output and configure the electrical characteristics as needed.
   a. Set *TMRn_CTRL1.outen_a* to 1 to enable the output.
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* initial value if desired.
   a. The initial *TMRn_CNT* value only affects the initial period in PWM mode, with subsequent periods always setting *TMRn_CNT* to 0x0000 0001.
9. Set the *TMRn_PWM* value to the transition period count.
   a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
10. Set the *TMRn_CMP* value for the PWM second transition period. *Note: TMRn_CMP must be greater than the TMRn_PWM value.*
    a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
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 13-6 shows the formula for calculating the timer PWM period.

Equation 13-6: Timer PWM Period in Seconds

$$PWM\ period = \frac{TMRn\_CNT}{f_{CNT\_CLK}\ (Hz)}$$

If an initial starting value other than 0x0000 0001 is loaded into the *TMRn_CNT* register, use the one-shot mode equation, *Equation 13-2*, 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 13-7*.

Equation 13-7: Timer PWM Output High Time Ratio with Polarity 0

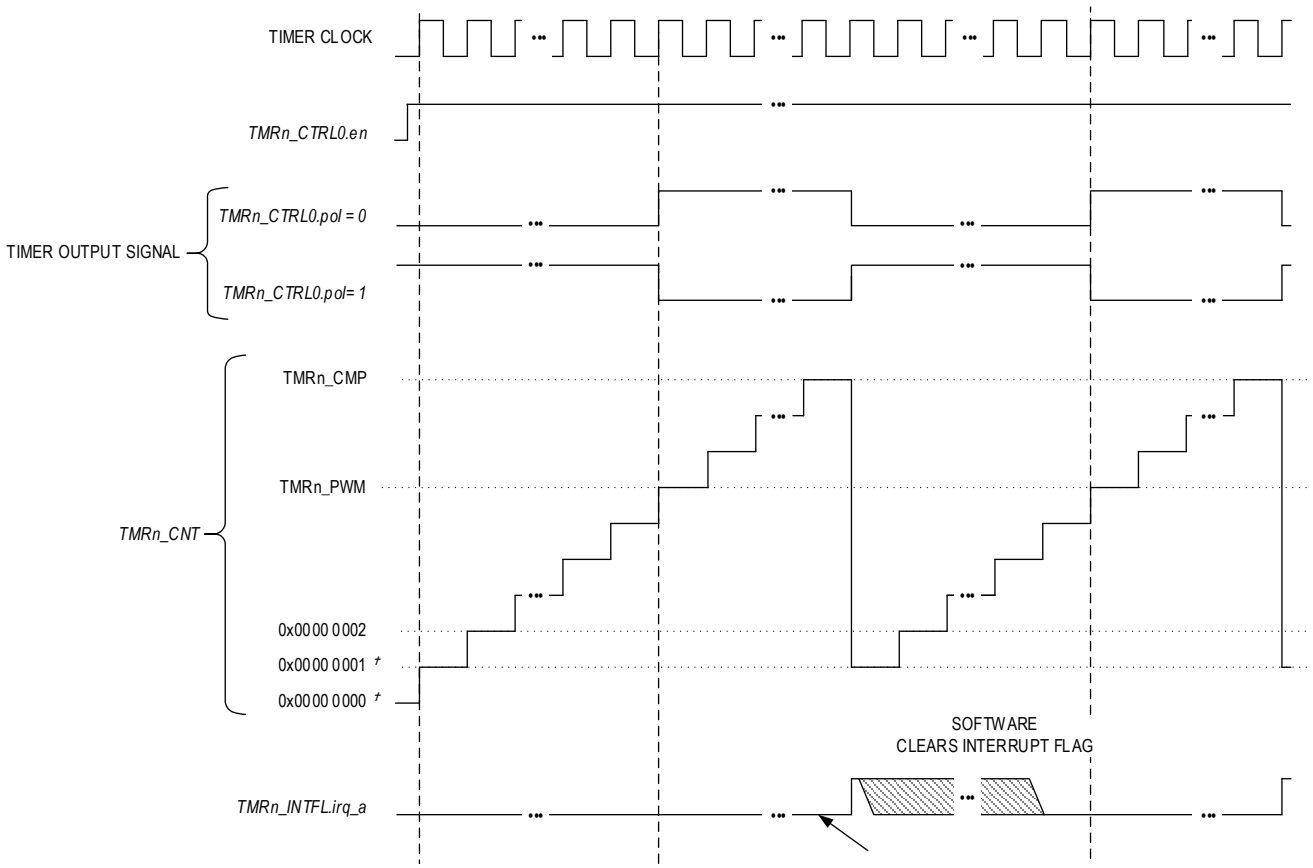
$$PWM\ output\ high\ time\ ratio\ (\%) = \frac{(TMR\_CMP - TMR\_PWM)}{TMR\_CMP} \times 100$$

If *TMRn_CTRL0.pol* is set to 1, the ratio of the PWM output high time to the total period is calculated using *Equation 13-8*.

Equation 13-8: Timer PWM Output High Time Ratio with Polarity 1

$$PWM\ output\ high\ time\ ratio\ (\%) = \frac{TMR\_PWM}{TMR\_CMP} \times 100$$

Figure 13-7: 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.

13.7.5 Capture Mode (4)

Capture mode is used to measure the time between application determined events. The timer starts incrementing the timer's count field until a transition occurs on the timer's input pin or a rollover event occurs. A capture event is triggered by hardware when the timer's input pin transitions state. When a capture event occurs, the timer hardware copies the timer's count value (*TMRn_CNT*) to the timer's PWM register (*TMRn_PWM*). *Equation 13-9* shows the formula for calculating the capture event's elapsed time.

If a capture event does not occur prior to the timer's count value reaching the timer's compare value (*TMRn_CNT* = *TMRn_CMP*), a rollover event occurs. Both the capture event and the rollover event set the timer's interrupt flag, *TMRn_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. Software must track the number of rollover events that occur prior to a capture event to determine the elapsed time of the capture event. When a capture event occurs, software should reset the count of rollover events.

Note: A capture event does not stop the timer's counter from incrementing and does not reset the timer's count value; a rollover event still occurs when the timer's count value reaches the timer's compare value.

13.7.5.1 Capture Event

When a capture event occurs, the timer hardware, on the next timer clock cycle, automatically performs the following actions:

- The *TMRn_CNT* value is copied to the *TMRn_PWM* register.
- The *TMRn_INTFL*.*irq* field is set to 1.
- The timer remains enabled and continues counting.

Note: Software must check the value of the TMRn_PWM register to determine the trigger of the timer interrupt.

Equation 13-9: Capture Mode Elapsed Time Calculation in Seconds

$$Capture\ elapsed\ time = \frac{\left(TMRn\_PWM - TMRn\_CNT_{INITIAL\_VALUE}\right) + \left((Number\ of\ rollover\ events) \times \left(TMRn\_CMP - TMRn\_CNT_{INITIAL\_VALUE}\right)\right)}{f_{CNT\_CLK}}$$

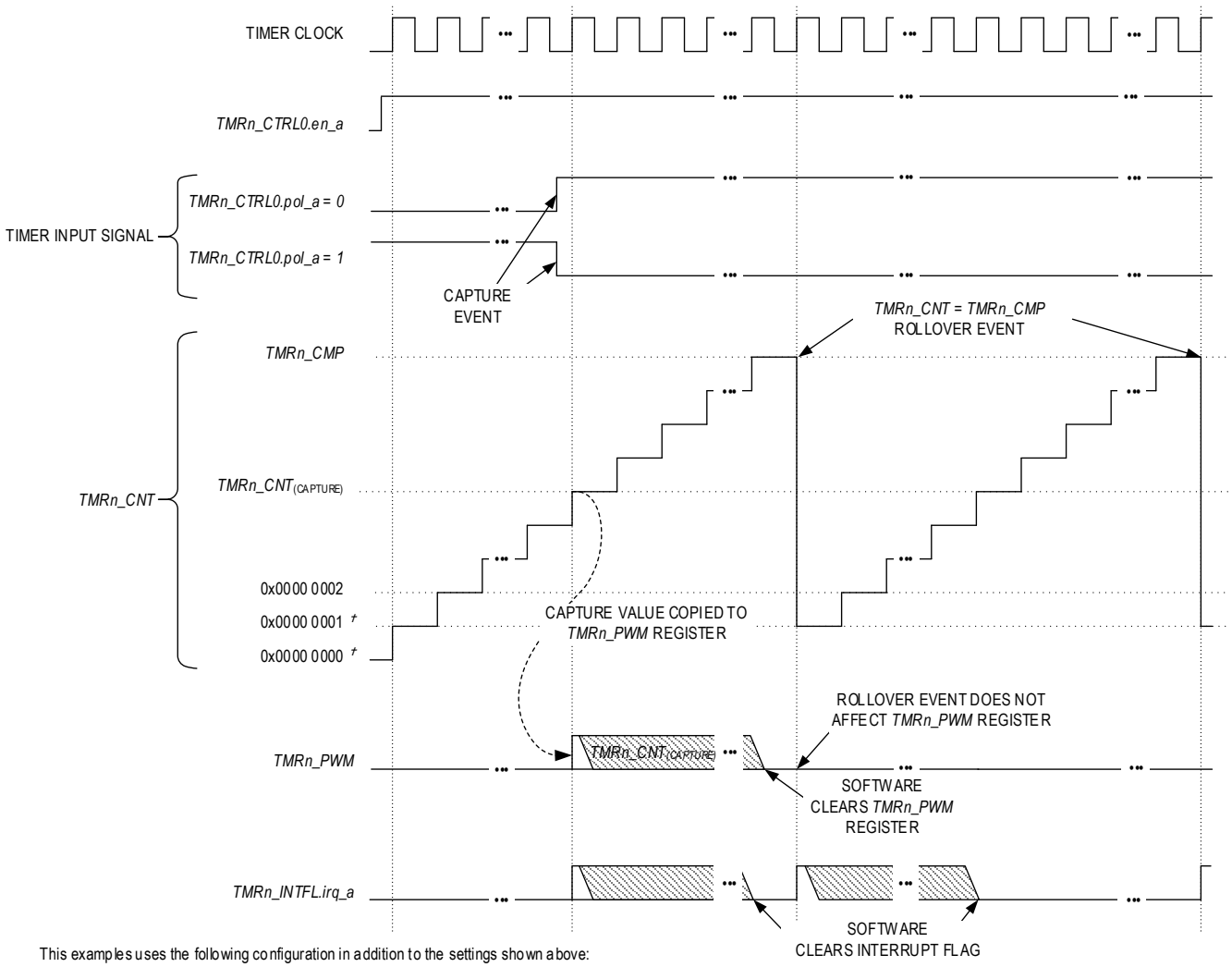
Note: The capture elapsed time calculation is only valid after the capture event occurs, and the timer stores the captured count in the TMRn_PWM register.

13.7.5.2 Rollover Event

A rollover event occurs when the timer's count value reaches the timer's compare value (*TMRn_CNT* = *TMRn_CMP*). 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* register is set to 0x0000 0001.
- The *TMRn_INTFL*.*irq* field is set to 1.
- The timer remains enabled and continues counting.

Figure 13-8: 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 = 4 (Capture)

† *TMRn_CNT* defaults to 0x00000000 on a timer reset. *TMRn_CNT* 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*, if desired.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
6 Write the compare value to the *TMRn_CMP* register.
   a. Poll the *TMRn_INTFL*.*wrdone* field until it reads 1.
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 13-10: 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.*

13.7.6 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* = *TMRn_CMP*.

The timer peripheral automatically performs the following actions when a timer period ends:

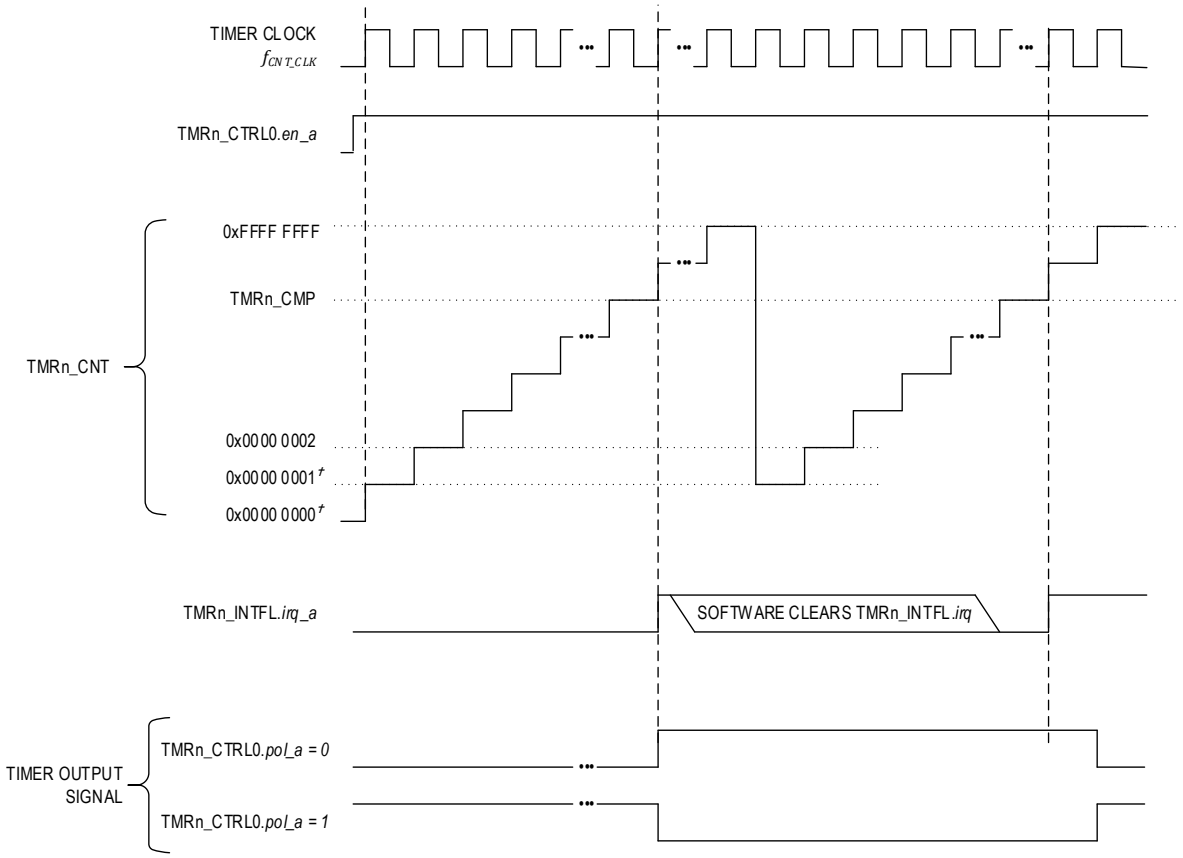
- Unlike other modes, *TMRn_CNT* is reset to 0x0000 00000, not 0x0000 0001 at the end of the timer period.
- The timer remains enabled and continues incrementing.
- The corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.
- Hardware toggles the state of the timer output signal.
- The timer output pin changes state if the timer output is enabled.

The compare mode timer period is calculated using *Equation 13-11*.

Equation 13-11: Compare Mode Timer Period in Seconds

$$Compare\ mode\ timer\ period = \frac{(TMR\_CMP - TMR\_CNT_{INITIAL\_VALUE} + 1)}{f_{CNT\_CLK}(Hz)}$$

Figure 13-9: 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)

† TMRn_CNT defaults to 0x00000000 on a timer reset. TMRn_CNT reloads to 0x00000001 for all following timer periods.

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.
   d. Set *TMRn_CTRL1*.*outen_a* to 1 to enable the output.
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*.
   a. Poll the *TMRn_INTFL*.*wrdone* field until it reads 1.
9. If desired, write an initial value to *TMRn_CNT*.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
10. Enable the timer peripheral as described in *Timer Clock Sources*.

13.7.7 Gated Mode (6)

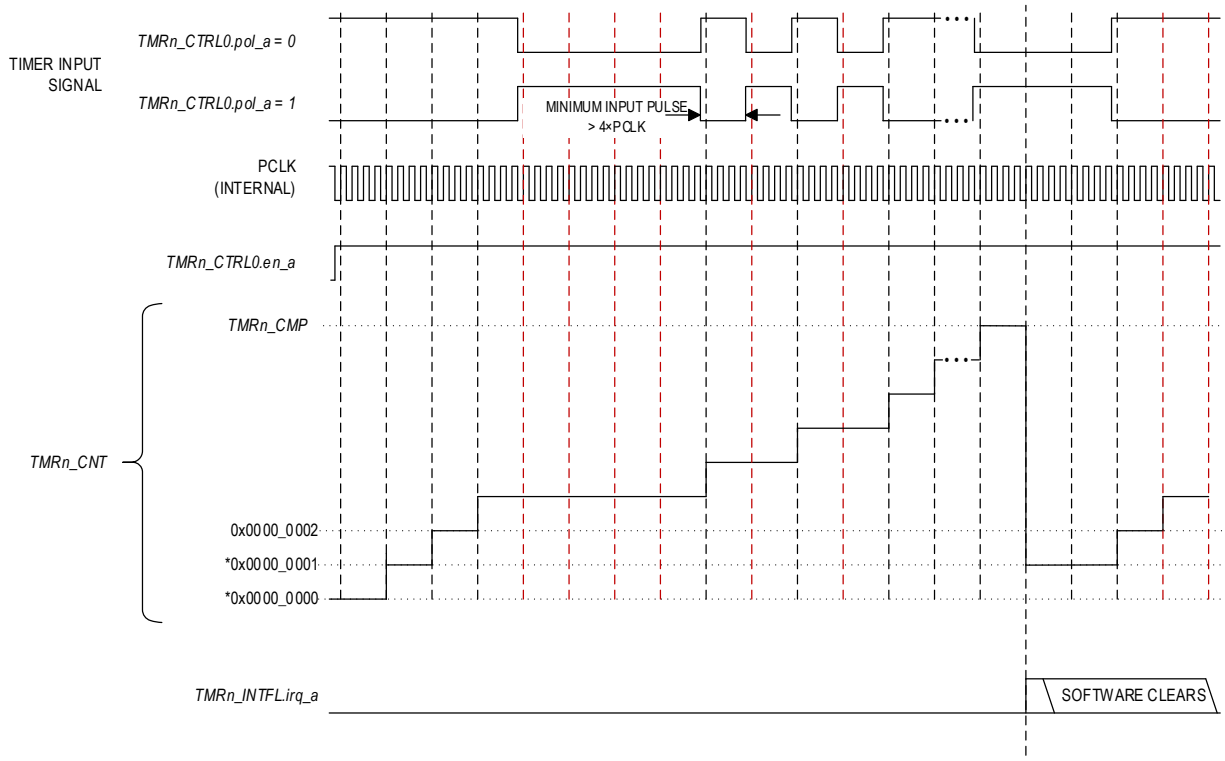
Gated mode is similar to continuous mode, except that *TMRn_CNT* only increments when the timer input signal is in its active state.

The timer period ends on the timer clock following *TMRn_CNT* = *TMRn_CMP*.

The timer peripheral automatically performs the following actions at the end of the timer period:

- The *TMRn_CNT* register is set to 0x0000 0001.
- The timer remains enabled and continues incrementing.
- If the timer output signal toggles state, 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 13-10: 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* defaults to 0x00000000 on a timer reset. *TMRn_CNT* 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* register.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
6 Write the compare value to *TMRn_CMP*.
   a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

13.7.8 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* value, writing it to the *TMRn_PWM* register (capture event). When a capture event occurs, a timer interrupt is generated, the *TMRn_CNT* value is reset to 0x0000 0001, and the timer resumes counting.

If no capture event occurs, the timer counts up to *TMRn_CMP*. At the end of the cycle, where the *TMRn_CNT* equals the *TMRn_CMP*, a timer interrupt is generated, the *TMRn_CNT* 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* = *TMRn_CMP*.

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 was caused by a transition on the timer pin, hardware automatically performs the following:

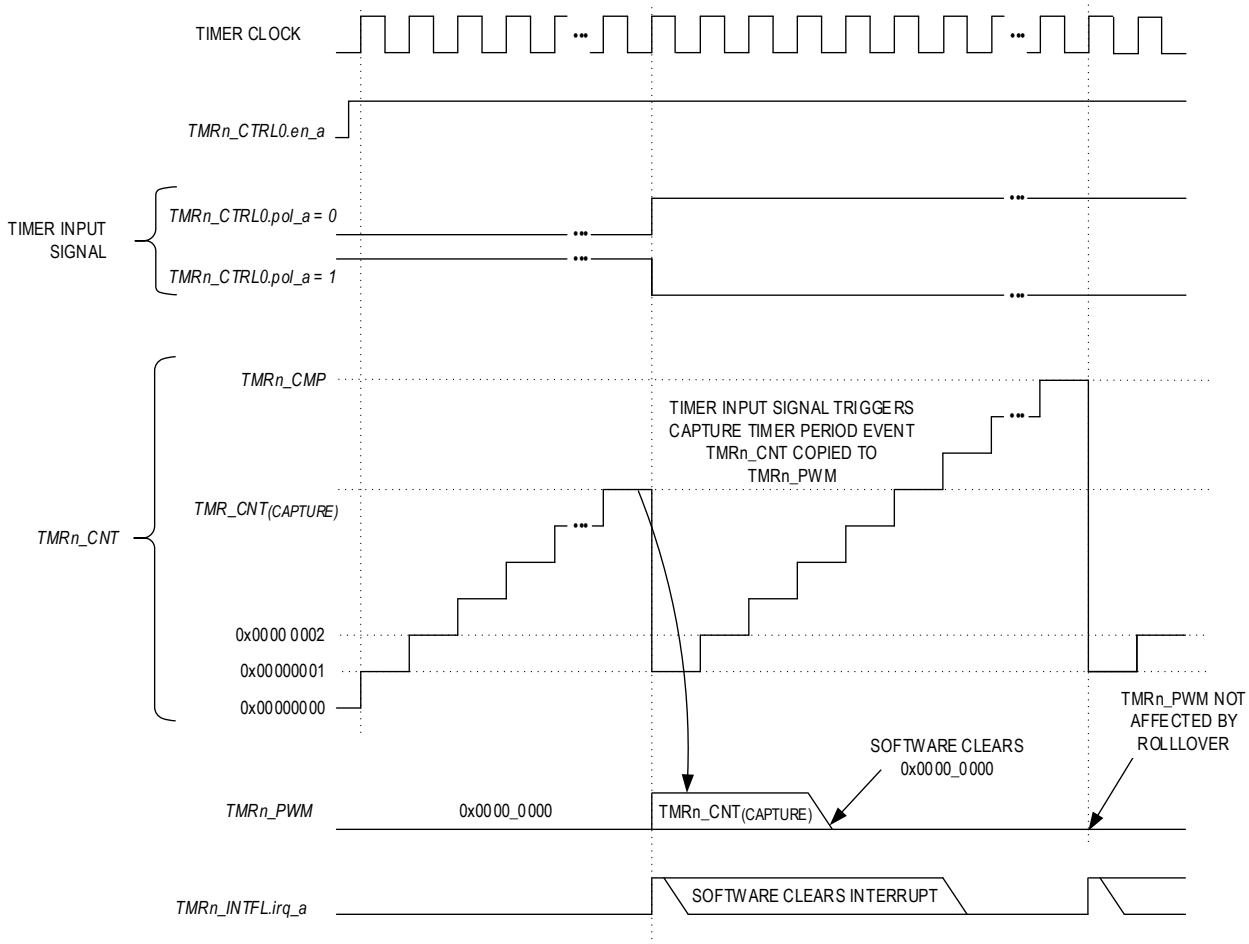
- The value in the *TMRn_CNT* register is copied to the *TMRn_PWM* register.
- The *TMRn_CNT* register is set to 0x0000 0001.
- The timer remains enabled and continues incrementing.
- The corresponding *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

In capture/compare mode, the elapsed time from the timer start to the capture event is calculated using *Equation 13-12*.

Equation 13-12: Capture Mode Elapsed Time in Seconds

$$Capture\ elapsed\ time = \frac{TMRn\_PWM - TMRn\_CNT_{INITIAL\_CNT\_VALUE}}{f_{CNT\_CLK}(Hz)}$$

Figure 13-11: 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* defaults to 0x00000000 on a timer reset. *TMRn_CNT* 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* register.
   a. This effects only the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
6 Write the compare value to *TMRn_CMP*.
   a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

Note: No interrupt is generated by the first transition of the input signal.

13.7.9 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.

13.7.10 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.

13.8 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 13-6*. 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 13-6: 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 Wakeup Status Register*

13.8.1 Register Details

Table 13-7: 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 is dependent on the 32-bit/16-bit configuration. Reads of this register always return the current value. **Note: This register is only writable if the timer clock is enabled (TMRn_CTRL0.en = 1).*

Table 13-8: 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 13-9: 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* = *TMRn_CMP*, the PWM output transitions to the second period of the PWM cycle. The second PWM period count is stored in *TMRn_CMP*. *TMRn_PWM* must be less than *TMRn_CMP* for PWM mode operation. **Timer Capture Value** In capture, compare, and capture/compare modes, this field is used to store the *TMRn_CNT* value when a capture, compare, or capture/compare event occurs. **Note: This register is only writable if the timer clock is enabled (TMRn_CTRL0.en = 1).*

Table 13-10: 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[31:16]* and *TMRn_PWM31:16]*. 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: If a timer is configured for PWM mode, the timer must be disabled before writing to the TMRn_PWM[15:0] field.* *Note: This field always reads 0 if the timer is configured as a 32-bit cascade timer.*

Timer Interrupt			TMRn_INTFL	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
24	wrdone_b	R	0	**TimerB Write Done** This field is cleared to 0 by hardware when software performs a write to *TMRn_CNT[31:16]* or *TMRn_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**
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 is used to disable write access to the *TMRn_CNT[15:0]* and *TMRn_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: If a timer is configured for PWM mode, the timer must be disabled before writing to the TMRn_PWM[15:0] field.* *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 hardware when software performs a write to *TMRn_CNT[15:0]* or *TMRn_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 13-11: 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	R/W1O	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 section *Operating Modes* 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
12	pwmckbd_a	RO	0	**Reserved**
11	nollpol_a	RO	0	**Reserved**
10	nolhpol_a	RO	0	**Reserved**

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
9	pwmsync_a	RO	0	**Reserved**
8	pol_a	R/W	0	**TimerA Polarity** This field 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 section *Operating Modes* 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 section *Operating Modes* 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
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 13-12: Timer Non-Overlapping Compare Register

Timer Non-Overlapping Compare			TMRn_NOLCMP	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	hi_b	RO	0	**Reserved**

Timer Non-Overlapping Compare			TMRn_NOLCMP	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
23:16	lo_b	RO	0	**Reserved**
15:8	hi_a	RO	0	**Reserved**
7:0	lo_a	RO	0	**Reserved**

Table 13-13: 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
30:29	-	RO	0	**Reserved**
28	we_b	R/W	0	**TimerB Wakeup Function** 0: Disabled 1: Enabled
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 13-2* 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 is set to 1 after software enables the TimerA clock by writing 1 to the *TMRn_CTRL1.clken_b* field. 0: Timer clock not ready or synchronization in progress 1: Timer clock is ready
18	clken_b	RO	0	**TimerB Clock Enable** Write this field to 1 to enable the TimerB clock. 0: Timer not enabled or synchronization in progress 1: Timer is enabled

Timer Control 1				TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**	
17:16	clksel_b	R/W	0	**TimerB Clock Source** See *Table 13-1* for the clock sources supported by each instance. *Note: In cascade 32-bit mode this field must be set to the same value as 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** This field controls the complimentary output of the timer.	
13	outen_a	R/W	0	**Output Enable** This field enables the timer output for modes 1, 3, and 5. Set this field to 1 to enable the output. Modes 2, 4, 6, and 8 automatically set the output mode on.	
12	we_a	R/W	0	**TimerA Wakeup Function** 0: Disabled 1: Enabled.	
11	sw_capevent_a	R/W	0	**TimerA Software Event Capture** 0: No software capture event-triggered 1: Trigger software capture event	
10:9	capevent_sel_a	R/W	0	**TimerA Event Capture Selection** Set this field to the desired capture event source. See *Table 13-2* 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	
3	clkrdy_a	RO	0	**TimerA Clock Ready** This field is set to 1 after 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.	

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
1:0	clksel_a	R/W	0	**Clock Source TimerA** See *Table 13-2* 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 13-14: Timer Wakeup Status Register

Timer Wakeup Status			TMRn_WKFL	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	b	R/W1C	1	**TimerB Wakeup Event** This flag is set when a wakeup event occurs for TimerB. Write 1 to clear. 0: No event 1: Wakeup event occurred
15:1	-	RO	0	**Reserved**
0	a	R/W1C	1	**TimerA Wakeup Event** This flag is set when a wakeup event occurs for TimerA. Write 1 to clear. 0: No event 1: Wakeup event occurred

14. Watchdog Timer (WDT)

The WDT protects against corrupt or unreliable software, power faults, and other system-level problems which 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 would not be detected with a traditional WDT because the end of the timeout periods would never be 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.

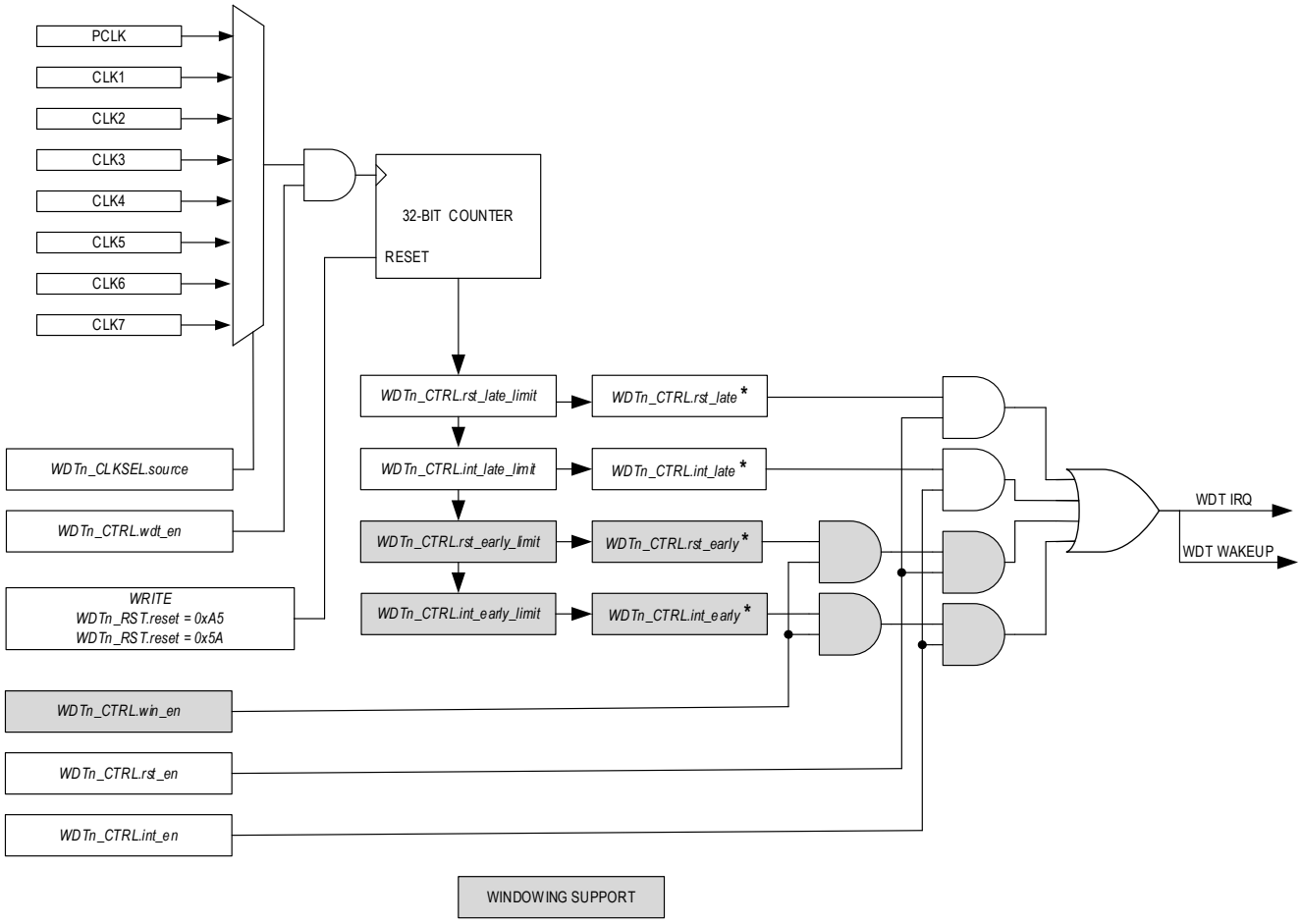
The WDT is a critical safety feature, and most fields are reset on POR or system reset events only.

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 14-1 shows a high-level block diagram of the WDT.

Figure 14-1: 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.*

14.1 Instances

Table 14-1 shows the peripheral instances, available clock sources, and windowed watchdog support.

Table 14-1: MAX32672 WDT Instances Summary

Instance	Register Access Name	Window Support	CLK0	CLK1	CLK2	CLK3	CLK4	CLK5	CLK6
WDT0	WDT0	Yes	PCLK	IPO	IBRO	INRO	ERTCO	EXT_CLK1 (P0.28 AF2)	ERFO
WDT1	WDT1								

14.2 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 which 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 to 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).

14.2.1 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.

14.2.2 Using the WDT as a Long-Interval Wakeup Timer

The WDT can be used as a very long internal wakeup source. Another application of the WDT is as a very long interval wakeup source from *SLEEP*.

14.3 WDT Protection Sequence

The WDT protection sequence protects the system against unintentional altering of the WDT count, and unintentional enabling or disabling of the timer itself. There are three different protection sequences described below.

14.3.1 WDT Reset Counter Sequence

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.

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.

14.3.2 WDT Enable Sequence

The enable sequence must 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. Write the *WDTn_RST.reset* field with 0xFE.
2. Write the *WDTn_RST.reset* field with 0xED.
3. Enable the WDT by setting *WDTn_CTRL.en* to 1.

14.3.3 WDT Disable Sequence

The disable sequence must be performed immediately before disabling the WDT to prevent accidental disabling of the WDT by software. 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. Write the *WDTn_RST.reset* field with 0xDE.
2. Write the *WDTn_RST.reset* field with 0xAD.
3. Disable the WDT by setting *WDTn_CTRL.en* to 0.

14.4 WDT Events

Multiple events are supported, as shown in *Table 14-2*. 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 14-2: WDT Event Summary

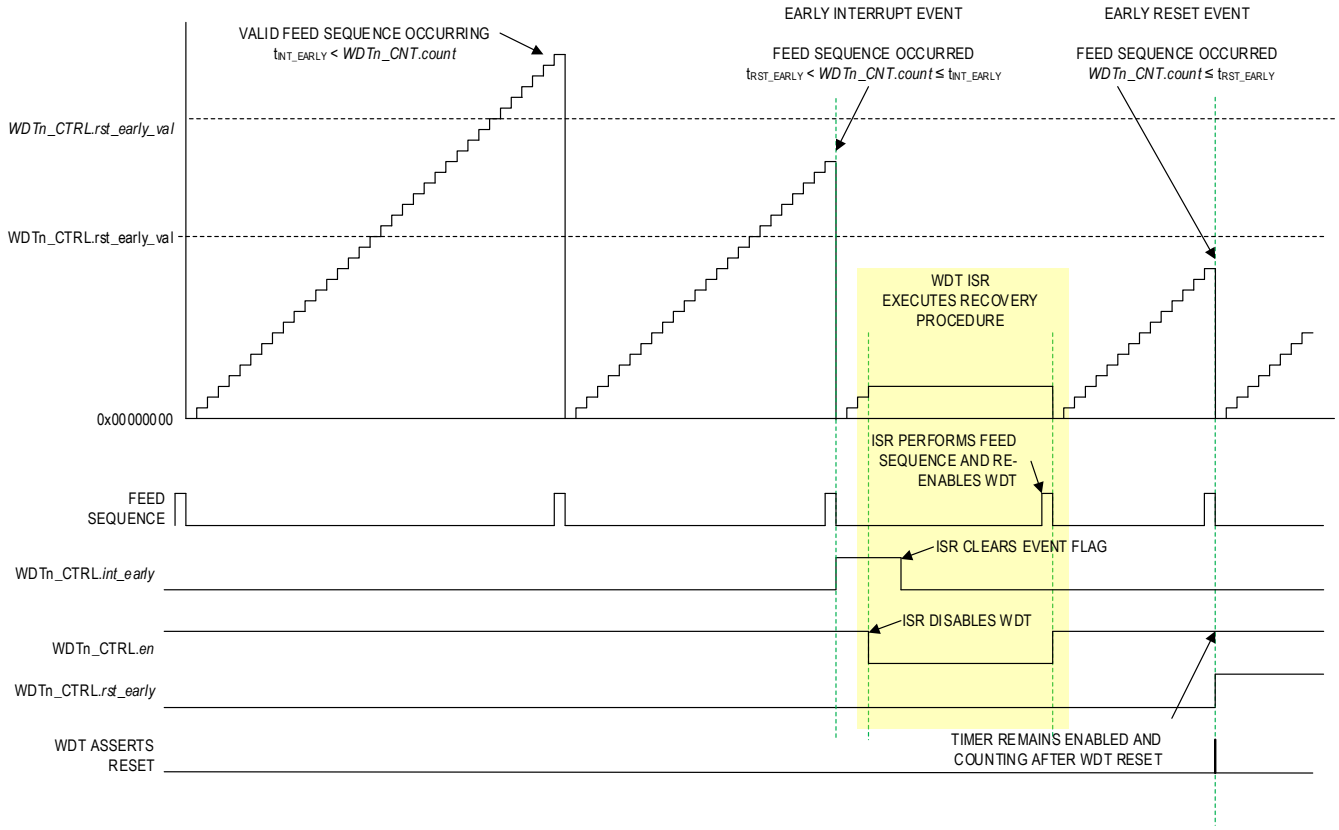
Event	Condition	Peripheral Interrupt Event Flag	Peripheral Interrupt Event Enable
Early Interrupt	Feed sequence occurs while *WDTn_CTRL.rst_early_val* ≤ *WDTn_CNT.count* < *WDTn_CTRL.int_early_val* *WDTn_CTRL.win_en = 1*	*WDTn_CTRL.int_early*	*WDTn_CTRL.wdt_int_en*
Early Reset	Feed sequence occurs while *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	

14.4.1 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 14-2 shows the sequencing details associated with an early reset event.

Figure 14-2: 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.

14.4.2 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 14-2*. *Figure 14-2* 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.

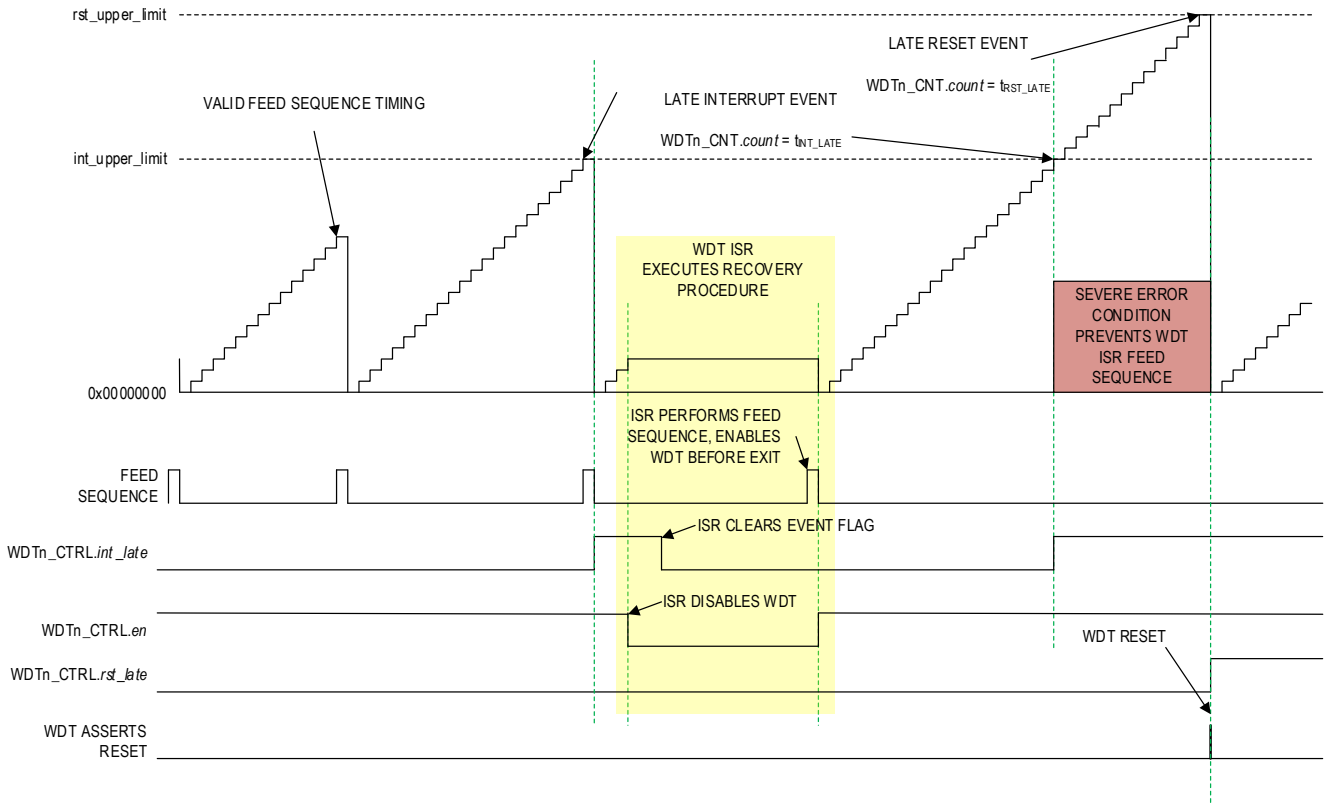
The following occurs when a WDT late interrupt event occurs:

1. The hardware sets *WDTn_CTRL.int_late* to 1.
2. The hardware initiates the WDT interrupt if enabled.

14.4.3 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 14-2*. *Figure 14-3* shows the sequencing details associated with a late reset event.

Figure 14-3: 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.

14.4.4 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 14-2*. *Figure 14-3* 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.

14.5 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.

14.6 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.

14.7 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 14-3*. 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 14-3: 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

14.7.1 Register Details

Table 14-4: 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

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
26:24	-	RO	0	**Reserved**
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

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
10	wdt_int_en	R/W	0	**WDT Interrupt Enable** 0: Disabled 1: Enabled
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.*

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
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 14-5: WDT Reset Register

WDT Reset			WDTn_RST	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved** Do not modify this field.
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 14-6: WDT Clock Source Select Register

WDT Clock Source Select			WDTn_CLKSEL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**

WDT Clock Source Select			WDTn_CLKSEL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
2:0	source	R/W	0[†]	**Clock Source Select** See *Table 14-1* 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 14-7: WDT Count Register

WDT Count			WDTn_CNT	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	count	R	0	**WDT Counter** The watchdog timer's counter value. This register is reset by system reset, as well as the watchdog feeding sequence. *Note: Before reading this register, the watchdog timer should be disabled by software (WDTn_CTRL.en = 0).*

15. Real-Time Clock (RTC)

15.1 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.

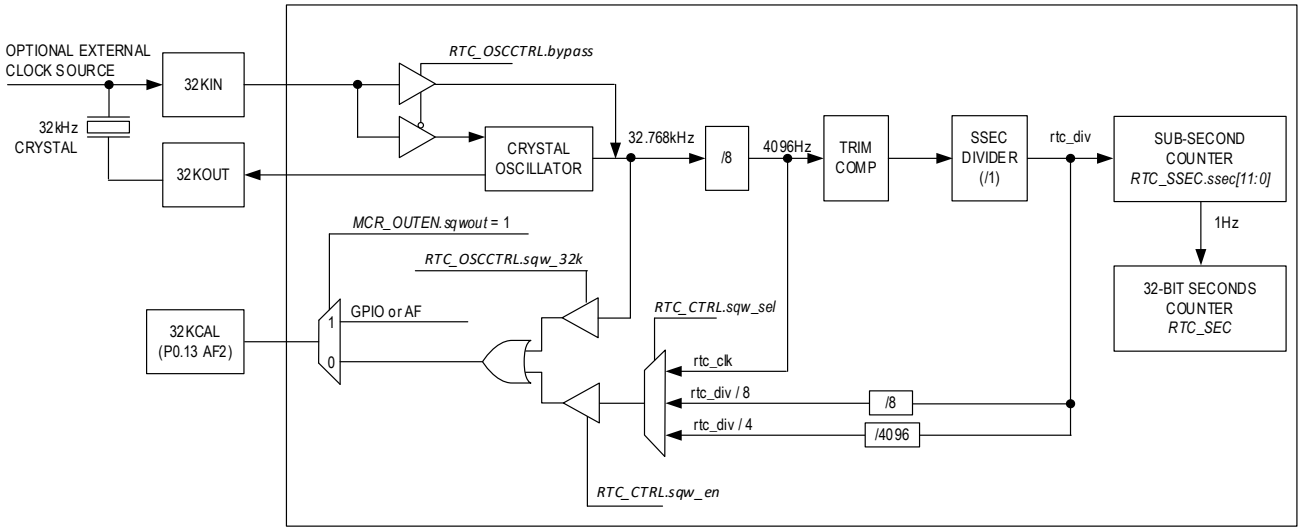
The 32-bit seconds counter register *RTC_SEC* is incremented every time there is a rollover of the *RTC_SSEC.ssec* sub-second counter field.

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 15-1: MAX32672 RTC Block Diagram



15.2 Instances

One instance of the RTC peripheral is provided. The RTC counter and alarm register details and description are shown in *Table 15-1*.

Table 15-1: 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

15.3 Register Access Control

Access protection mechanisms prevent the software from accessing critical registers and fields while RTC 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 15-2: 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.

15.3.1 RTC_SEC and RTC_SSEC Read Access Control

oftware 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.

Alternatively, 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. To ensure valid register data, the software must read the *RTC_SEC* and *RTC_SSEC* registers within 120µs.
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.

15.3.2 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.

15.4 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.

15.4.1 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.

15.4.2 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.

15.4.3 RTC Interrupt and Wakeup Configuration

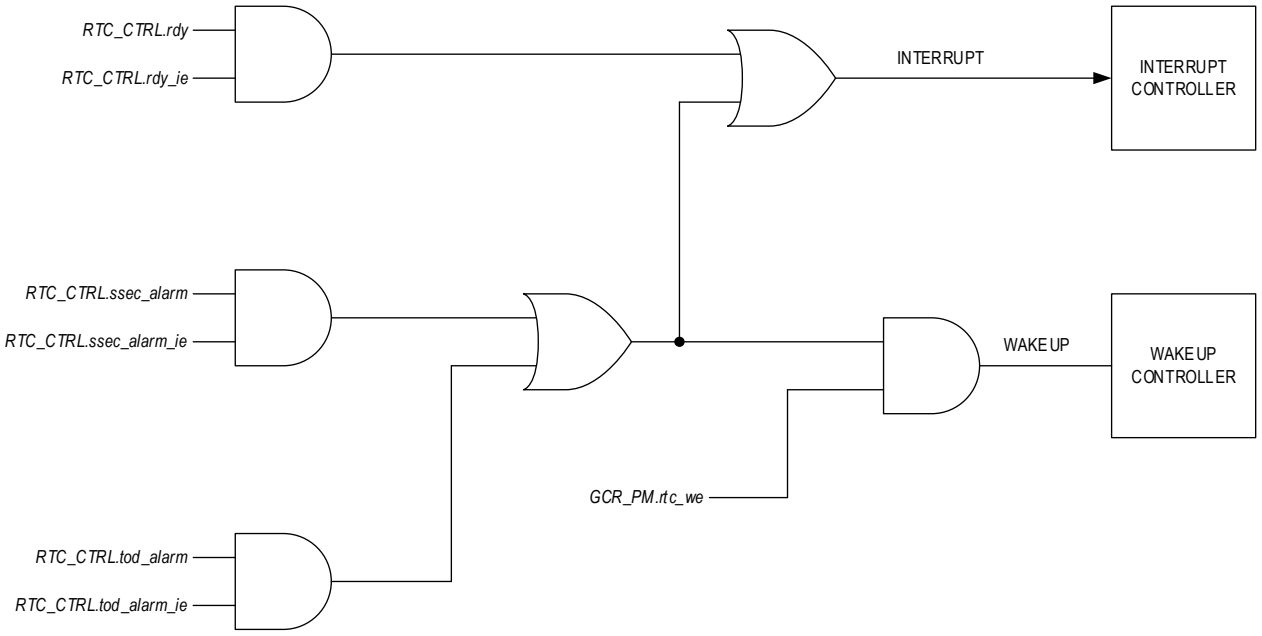
The following is a list of conditions that, when enabled, generate an RTC interrupt:

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. *BACKUP*
2. *DEEPSLEEP*
3. *SLEEP*

Figure 15-2: RTC Interrupt/Wakeup 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.

15.5 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 15-3* for the device pins, frequency options, and control fields specific to this device. Frequencies noted as compensated in *Table 15-3* are used during the RTC frequency calibration procedure because they incorporate the frequency adjustments provided by the digital trim function.

Table 15-3: MAX32672 RTC Square Wave Output Configuration

Function	Option	Control Field
Output Pin	P0.13: 32KCAL	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

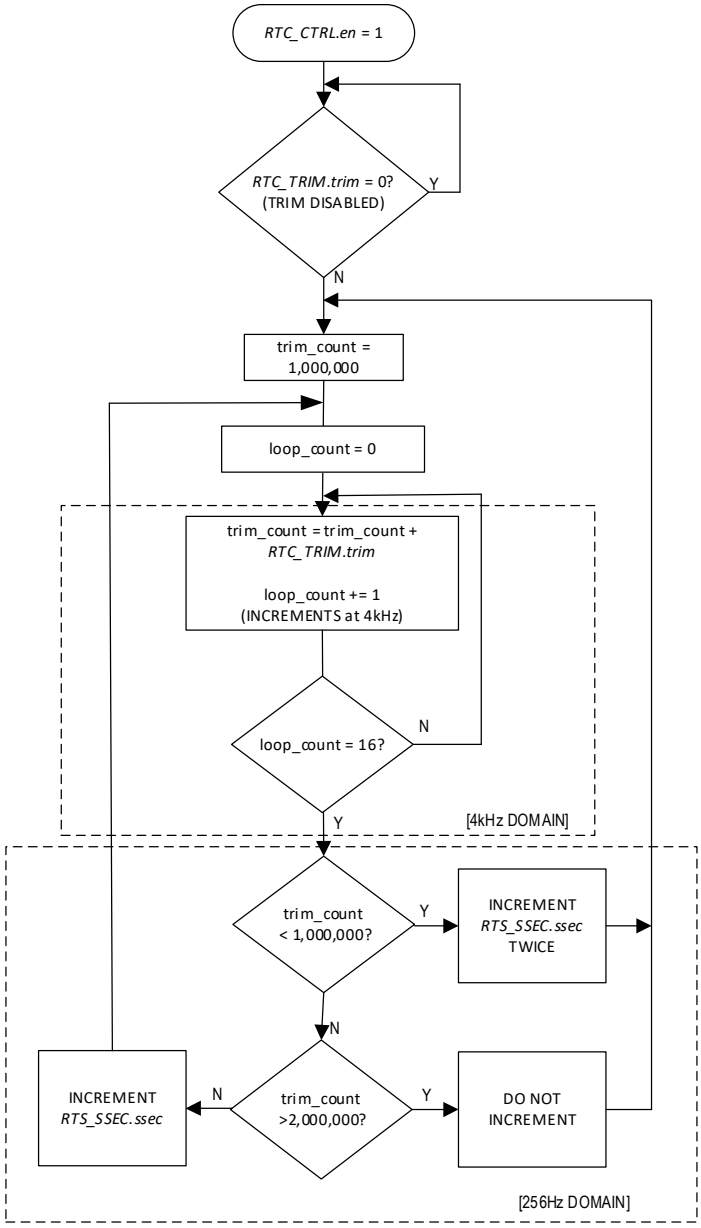
Use the following software procedure to generate and output the square wave:

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 15-3*.
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.

15.6 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, twos-complement value in the *RTC_TRIM* register as shown in *Figure 15-3*.

Figure 15-3: 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.

15.7 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 15-4: 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

15.7.1 Register Details

Table 15-5: 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 15-6: 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 15-7: 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 15-8: 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 15-9: RTC Control Register

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
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 for 0xA1 revision or MCR_RST.rtc assertion for all other revisions.*
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 15-3* 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 15-3* 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.*

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
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 for 0xA1 revision or MCR_RST.rtc assertion for all other revisions.*
4	rdy	RO	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 has 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 for 0xA1 revision or MCR_RST.rtc assertion for all other revisions.*
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.*

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
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.*
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 15-10: RTC 32kHz Oscillator Digital Trim Register

RTC 32kHz Oscillator Digital Trim			RTC_TRIM	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	vrtc_tmr	R/W	0*	**VRTC Time Counter** 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 15-11: RTC 32kHz Oscillator Control Register

RTC 32kHz 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 15-3* 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	ibias_en	R/W	1	**Current Bias Enable** Set this field to 1 to enable a higher current mode for the RTC oscillator. See *RTC_OSCCTRL.ibias_sel* for additional details.

RTC 32kHz Oscillator Control			RTC_OSCCTRL	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
2	hyst_en	R/W	0	**High Current Hysteresis Buffer Enable** This field enables a high current hysteresis buffer. 0: Disabled. 1: Enabled.
1	ibias_sel	R/W	0	**Current Bias Select** This field selects between 2× and 4× bias mode if *RTC_OSCCTRL.ibias_en* is set to 1. 0: 2× mode. 1: 4× mode.
0	filter_en	R/W	1	**Deglitch Filter Enable** Set this field to 1 to enable the analog deglitch filter for the RTC oscillator. 0: Disabled. 1: Enabled.

16. 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.

16.1 Instances

The DAP interface communicates through the serial wire debug (SWD), shown in *Table 16-1*.

Table 16-1: MAX32672 DAP Instances

Pin	Alternate Function	SWD Signal
P0.0	AF1	SWDIO
P0.1	AF1	SWDCLK

16.2 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. See *Secure Communication Protocol Bootloader (SCPBL)* for details of how to lock the DAP in all die revisions besides 0xA1. Determine the device revision by reading the *GCR_REVISION*.*revision* register.

16.2.1 Locking the DAP in Revision 0xA1

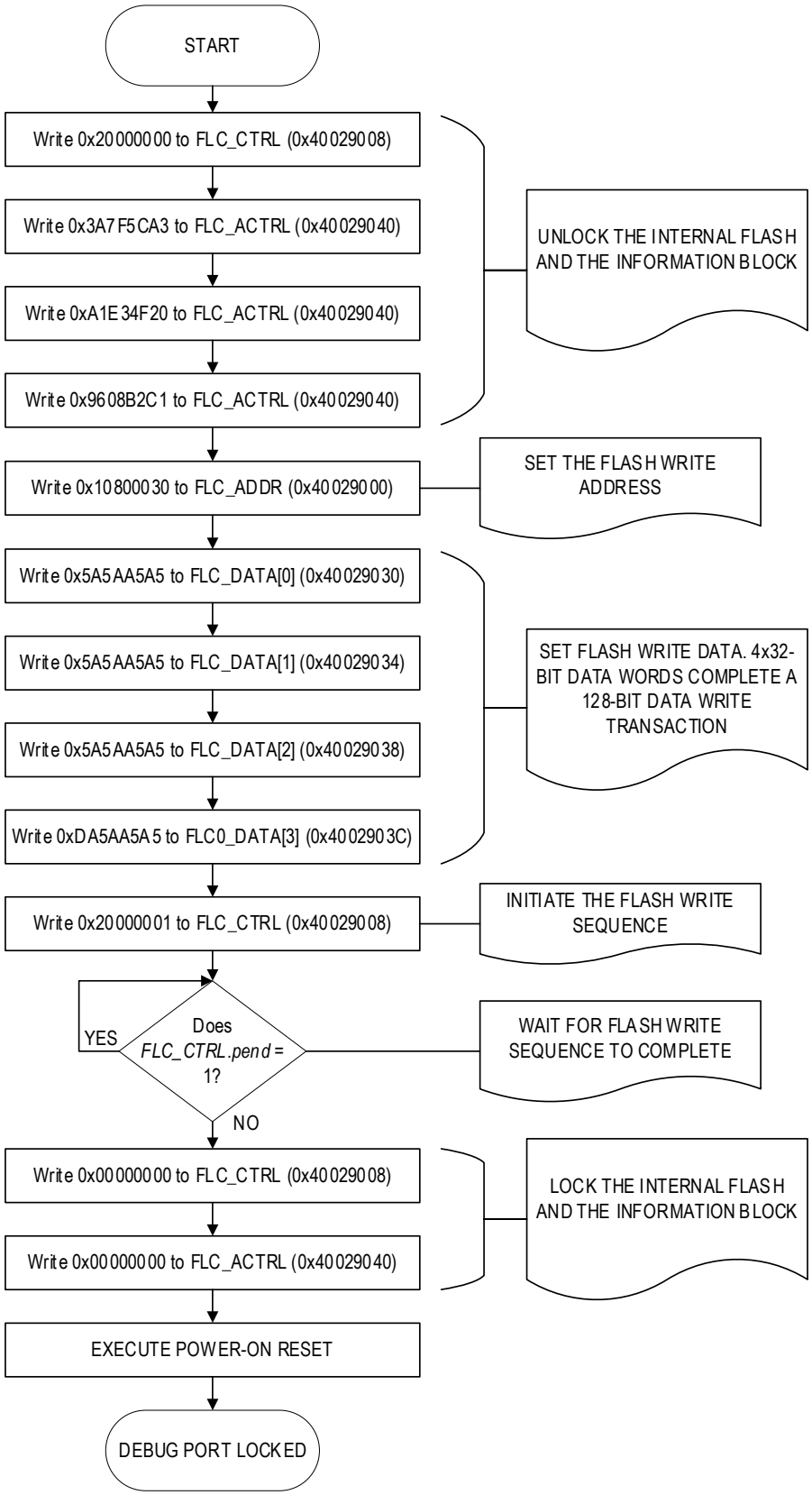
There are two options for locking out the debug access port. Option 1 locks the DAP and makes it available to be unlocked later. This is a one-time-only process. The DAP port cannot be relocked. Option 2 locks the DAP permanently.

16.2.1.1 Option 1 (Revision 0xA1 Only)

To lock the DAP and make it available to be unlocked later (one time only), follow the flow chart in *Figure 16-1*.

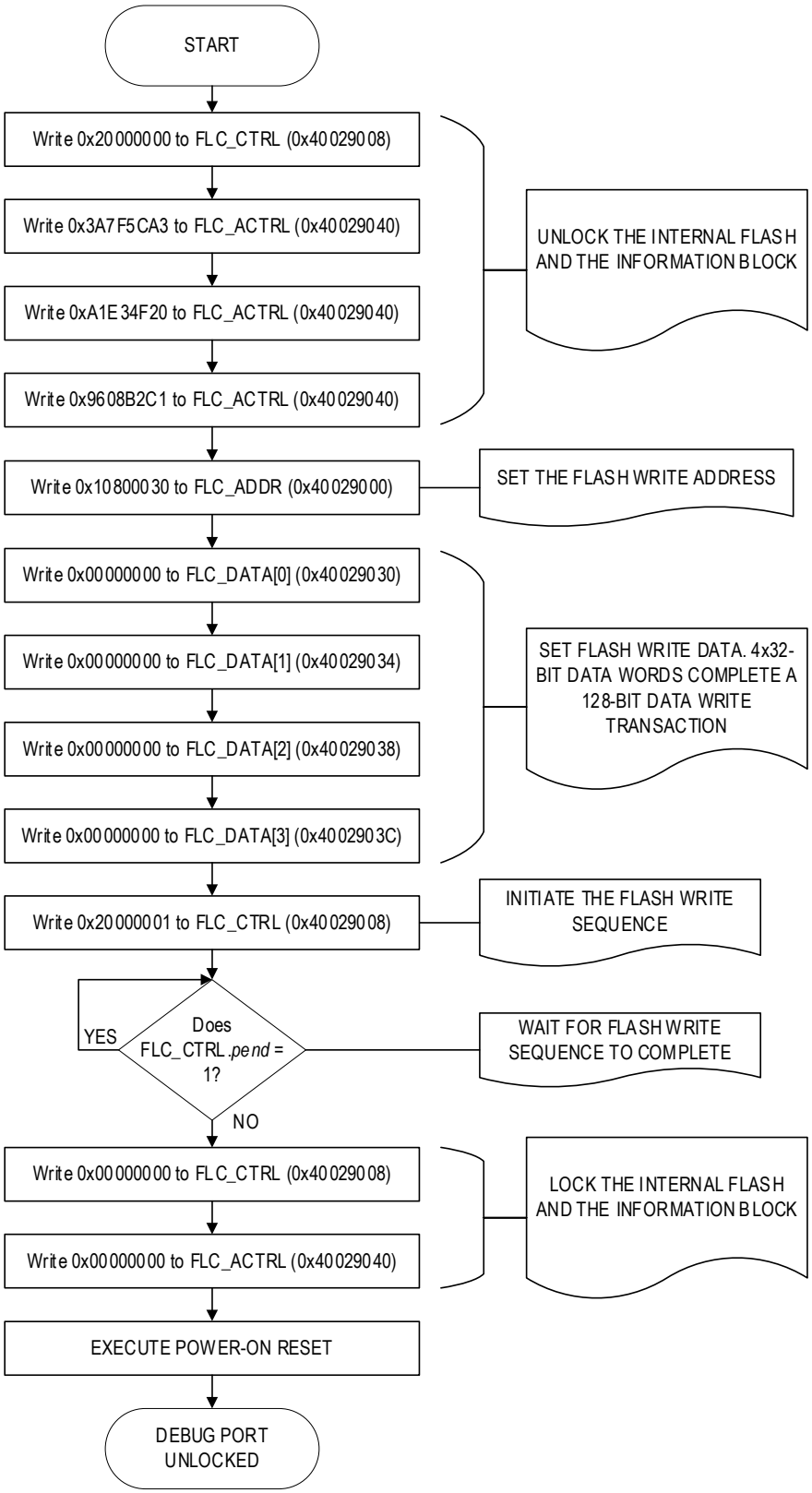
CoreSight is a registered trademark of Arm Limited.

Figure 16-1: 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 16-1*, follow the flow chart in *Figure 16-2*.

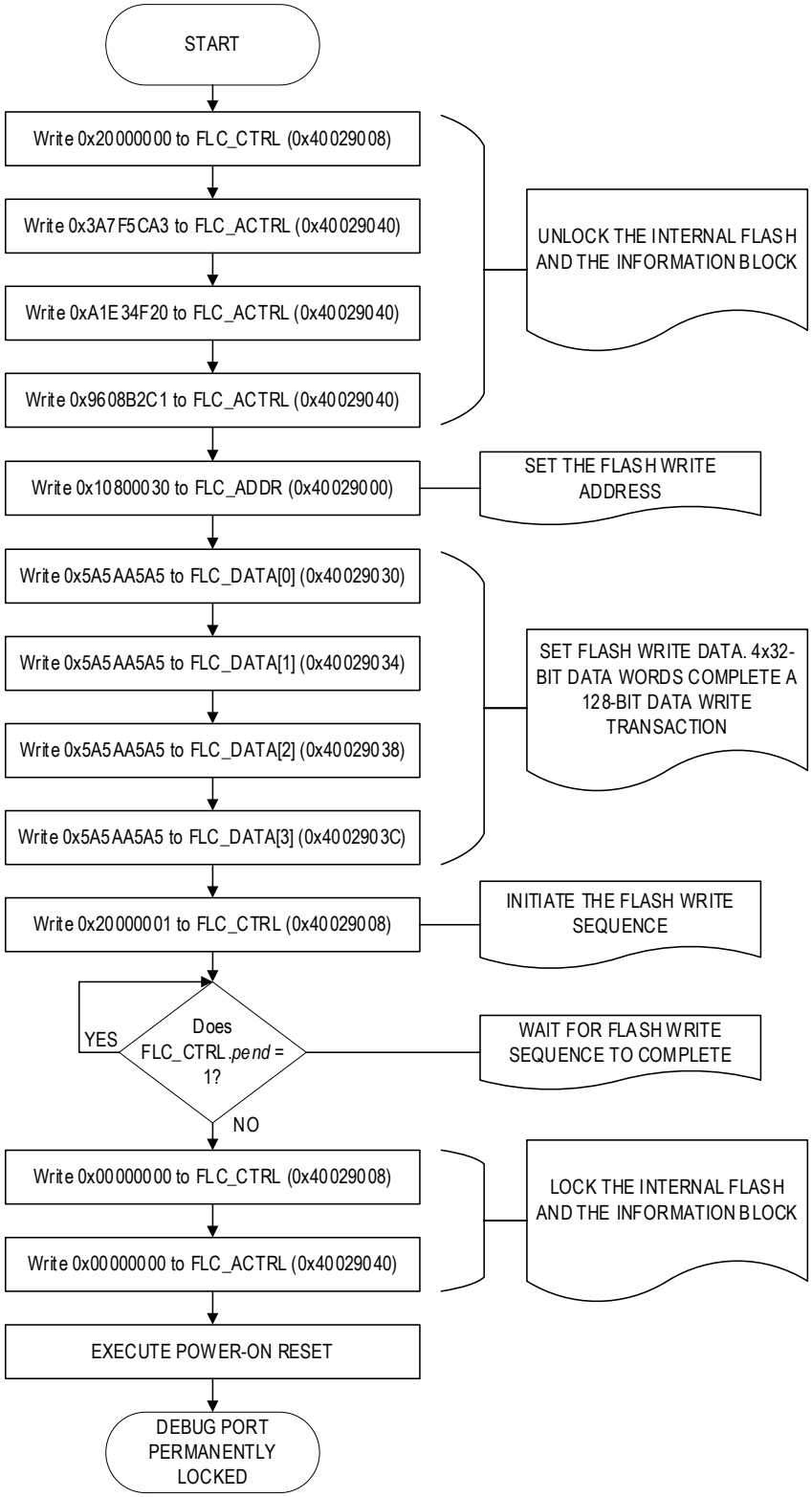
Figure 16-2: Unlocking the DAP After Being Locked as in Figure 16-1



16.2.1.2 Option 2 (Revision 0xA1 Only)

To lock the DAP permanently, follow the flow chart in *Figure 16-3*.

Figure 16-3: Locking the Debug Access Port Permanently



16.3 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.

17. Quadrature Decoder Interface (QDEC)

The quadrature decoder interface (QDEC) peripheral connects to a mechanical device that produces a digital quadrature output indicating the speed and position of a rotating mechanical device. A typical quadrature encoder has a wheel with a pattern of markings and a sensor, usually optical, that senses those markings as it turns. Three signals indicate the movement of the wheel in steps with a user-defined granularity. The QEA and QEB signals have a unique phase relationship that signals the disk movement and rotation direction. The QEI signal is active once per rotation of the disk to reference an absolute position.

The QDEC tracks position with a 32-bit counter that increments for clockwise rotations and decrements for counterclockwise rotations.

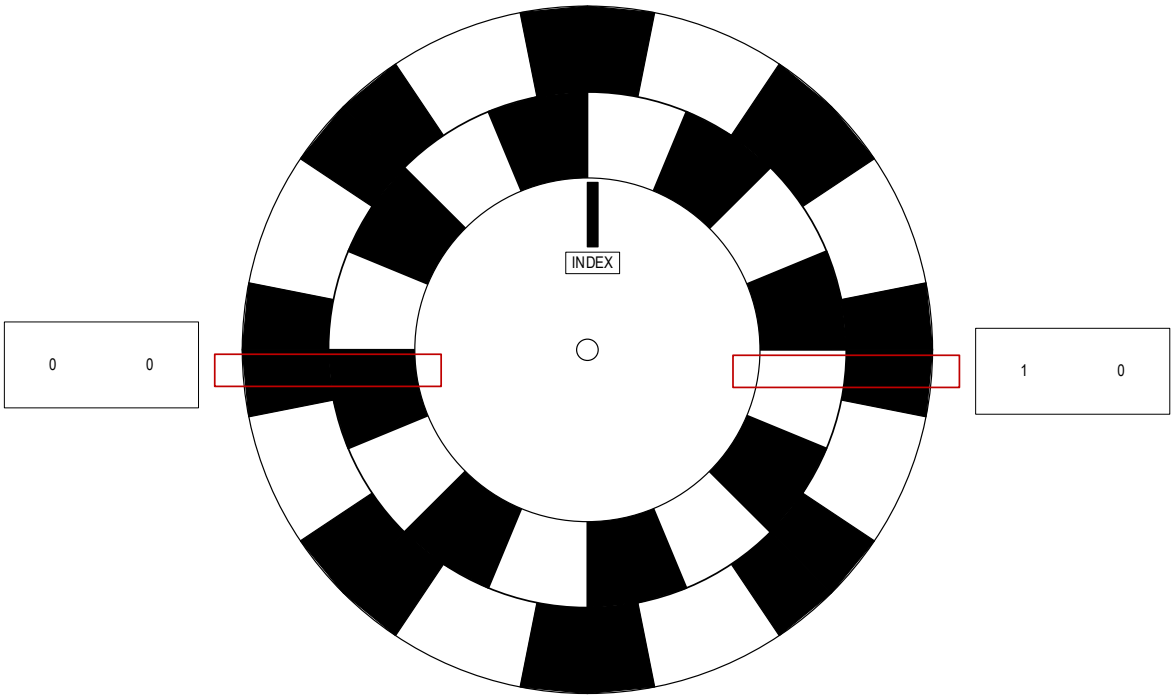
The peripheral provides the following features:

- Four signal inputs:
  - QEA (phase A)
  - QEB (phase B)
  - QEI (index pulse)
  - QES (position capture)
- Three signal outputs:
  - QMATCH (Position counter value match)
  - QERR (Error)
  - QDIR (direction of rotation)
  - QES (position capture)
- Programmable sample size to filter digital noise.
- 32-bit position counter for tracking high-resolution movements.
- Position capture on an external signal.
- Position match event generates an external signal.
- Programmable minimum value other than 0x0000 0000.
- Dual-mode position counter reset source:
  - Reset on match with a maximum count.
  - Reset on index pulse
- Programmable interrupt events including:
  - Direction change
  - Movement
  - Index pulse detected
  - Invalid phase transition

17.1 Operation

A digital encoder employs a disk or strip divided into a series of marks or slots that shifts or rotates when a mechanical device moves. A sensor, usually optical, measures two rows of markings known as Phase A and Phase B at 90 degrees out of phase, as shown in *Figure 17-1*. The sensor outputs the two phases on two signal lines, QEA and QEB, to a quadrature decoder. An optional index output pulses once per revolution as an absolute position indicator.

Figure 17-1: Quadrature Shaft Rotator Markings



The two phases produce 4 states, as shown in *Table 17-1*, with the order of the transitions through the states indicating the direction of movement. The rate of change of the state transitions indicates the speed of rotation. The number of transitions can be used to determine what fraction or number of complete rotations have occurred.

Table 17-1: State Transitions and Direction of Movement

State	QEB Input	QEA Input	Clockwise Rotation	Counterclockwise Rotation
00	0	0	↓	↑
01	0	1		
11	1	1		
10	1	0-		

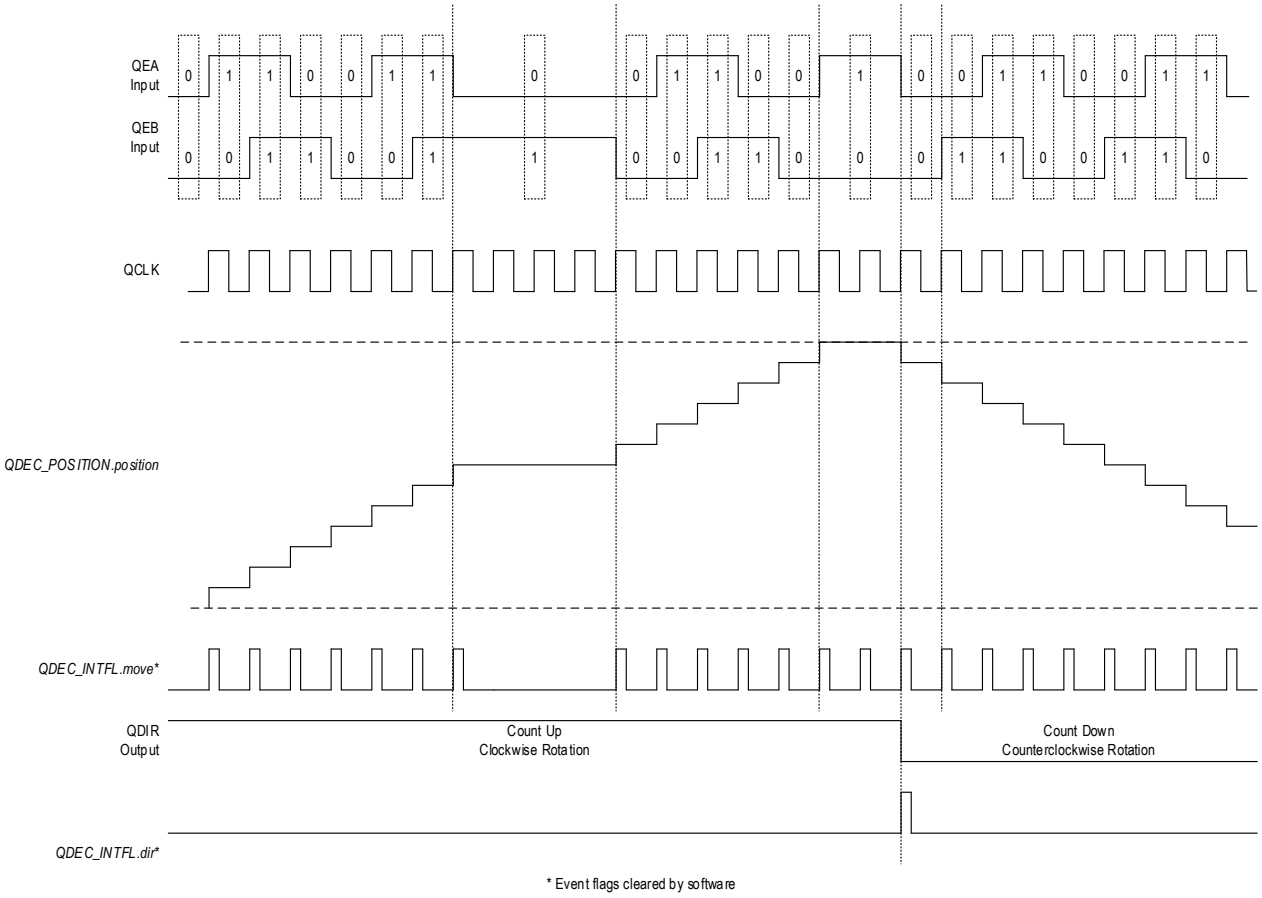
Every state transition shown in *Figure 17-2* signals a movement of the wheel that triggers the following events:

- The position counter *QDEC_POSITION*.*position* is incremented if the state transition is clockwise or decremented if the state transition is counterclockwise.
- The *QDEC_STATUS*.*dir* field is updated to indicate the rotation of the movement.
- The QDIR output is updated to match the value of *QDEC_STATUS*.*dir.*
- A move event occurs, which sets the *QDEC_INTFL*.*move* field to 1.
- A direction change event occurs if the value of *QDEC_STATUS*.*dir* has changed.

All event flags must be cleared to 0 by software.

Figure 17-2 shows the relationship of the phase inputs, counter values, and the internal flags.

Figure 17-2: Position Counter Function



The QDEC peripheral provides an option for a user-programmable minimum setting through the *QDEC_INITIAL* register, adding the flexibility of low rollover numbers other than 0x0000 0000.

17.2 Functions

The QDEC peripheral can perform multiple actions based on the value of *QDEC_POSITION*.*position*, or the state of the QES and QEI inputs. The capture and compare functions can be used simultaneously as the reset on the maximum count or the reset on index pulse functions, since capture and compare do not affect the value of *QDEC_POSITION*.*position*.

17.2.1 Reset on Maximum Count

The position counter can be programmed to reset itself to a value after a user-defined number of transitions set in *QDEC_MAXCNT*.*maxcnt* and cause a maximum count event. This value can be used to indicate a full or partial revolution.

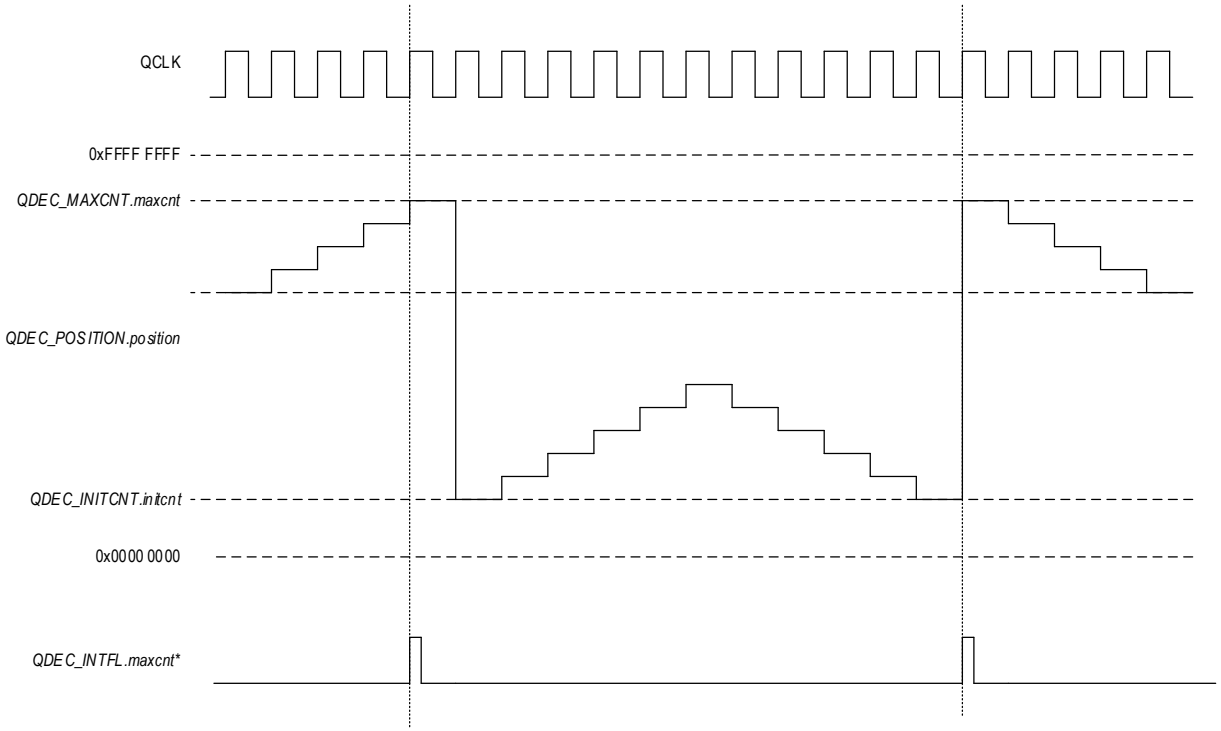
In this mode, *QDEC_POSITION.position* is automatically reloaded when *QDEC_POSITION.position* matches *QDEC_INITIAL.initial* or *QDEC_MAXCNT.maxcnt*. The reload value depends on the direction of rotation:

- If the rotation is clockwise, *QDEC_POSITION.position* is reset with *QDEC_INITIAL.initial.*
- If the rotation is counterclockwise, *QDEC_POSITION.position* is reset with *QDEC_MAXCNT.maxcnt.*

If the reset on maximum mode is not enabled the counter counts up to 0xFFFF FFFF and rollover to 0x0000 0000 (clockwise) and countdown to 0x0000 0000 and rollover to 0xFFFF FFFF (counterclockwise).

- If the rotation is clockwise, the event occurs on the clock tick when *QDEC_POSITION.position* is reset with *QDEC_MAXCNT.maxcnt.*
- If the rotation is counterclockwise, the maximum count event occurs on the clock tick following the one that resets *QDEC_POSITION.position* to *QDEC_INITIAL.initial.* This is because the maximum count event occurs when *QDEC_POSITION.position* matches *QDEC_MAXCNT.maxcnt*, which is the cycle after *QDEC_POSITION.position* matches *QDEC_INITIAL.initial.*

Figure 17-3: Reload on Maximum Count Match



Use the following procedure to configure the device for the reset on maximum count mode:

1. Clear *QDEC_CTRL.en* to 0 to disable the peripheral.
2. Configure the desired QCLK rate as described in *Clock and Sampling Selection*.
3. Set *QDEC_CTRL.mode* to the highest value that meets the timing requirements.
4. Set *QDEC_CTRL.rst_maxcnt* to 1 to enable the reset on index pulse function.
5. Clear *QDEC_INTFL.maxcnt* to 0.
6. Set *QDEC_INTEN.maxcnt* to 1 to enable interrupt generation by a reset on index pulse event.
7. Set *QDEC_CTRL.en* to 1 to enable the peripheral.

17.2.2 Reset on Index Pulse

The position counter can be programmed to reset itself when it receives a pulse on the QEI pin, indicating a full rotation of the encoder wheel. Detection of an index pulse generates an index event, reload the *QDEC_POSITION*.*position* register with the *QDEC_INITIAL*.*initial* value or the *QDEC_INTFL*.*maxcnt* value depending on the direction of rotation.

Use the following procedure to configure the device for the Reset on Index Pulse mode:

1. Clear *QDEC_CTRL*.*en* to 0 to disable the peripheral.
2. Configure the desired QCLK rate as described in *Clock and Sampling Selection*.
3. Set *QDEC_CTRL*.*mode* to the highest value that meets the timing requirements.
4. Set *QDEC_CTRL*.*rst_index* to enable the reset on index pulse function.
5. Clear *QDEC_INTFL*.*index* to 0.
6. Set *QDEC_INTEN*.*index* to 1 to enable interrupt generation by a reset on index pulse event.
7. Set *QDEC_CTRL*.*en* to 1 to enable the peripheral.

17.2.3 Capture

A valid transition on the QES pin causes the current value of the *QDEC_POSITION*.*position* to be latched into the *QDEC_CAPTURE*.*capture* and sets *QDEC_INTFL*.*capture* to 1. The counter value itself is not affected and continues counting.

This mode is used in conjunction with the reset on the maximum count or reset on index pulse modes. The software must always clear the event flag before setting the event interrupt enable field.

Use the following procedure to configure the device for the capture feature:

1. Configure the device for either *Reset on Maximum Count* or *Reset on Index Pulse* mode.
2. Before re-enabling the peripheral:
   a. Clear *QDEC_INTFL*.*capture* to 0.
   b. Set *QDEC_INTEN*.*capture* to 1 to enable interrupt generation by a capture event.
3. Set *QDEC_CTRL*.*en* to 1 to enable the peripheral.

17.2.4 Compare

A compare interrupt event occurs every time *QDEC_POSITION*.*position* changes to match the value in *QDEC_COMPARE*.*compare*. This occurs even if *QDEC_POSITION*.*position* has not been changed from its default value.

Use the following procedure to configure the device for the compare feature:

1. Configure the device for either *Reset on Maximum Count* or *Reset on Index Pulse* mode.
2. Before re-enabling the peripheral:
   a. Load *QDEC_COMPARE*.*compare* with the desired value
   b. Clear *QDEC_INTFL*.*capture* to 0.
3. Set *QDEC_INTEN*.*capture* to 1 to enable interrupt generation by a capture event.
4. Set *QDEC_CTRL*.*en* to 1 to enable the peripheral.

17.3 Interrupt Events

The following interrupt events are supported. Setting an event interrupt enable generates an interrupt when the corresponding event flag is set. The peripheral must be disabled before writing to the *QDEC_INTEN* register.

The event flag should always be cleared before its corresponding event interrupt is enabled.

Table 17-2: Interrupt Events

Event	Description	Flag	Interrupt Enable	Additional Effects
Index	Transition on QEI pin	*QDEC_INTFL.index*	*QDEC_INTEN.index*	
Error	Invalid state transition	*QDEC_INTFL.qerr*	*QDEC_INTEN.qerr*	QERR pin set to 1
Compare Match	*QDEC_POSITION.position* matches the value in *QDEC_COMPARE.compare*	*QDEC_INTFL.compare*	*QDEC_INTEN.compare*	QMATCH pin set to 1
Maximum Count Reached	*QDEC_POSITION.position* matches the value in *QDEC_MAXCNT.maxcnt*	*QDEC_INTFL.maxcnt*	*QDEC_INTEN.maxcnt*	None
Capture	Transition on QES pin	*QDEC_INTFL.capture*	*QDEC_INTEN.capture*	None
Direction Change	Change of *QDEC_STATUS.dir*	*QDEC_INTFL.dir*	*QDEC_INTEN.dir*	QMATCH pin set to 1
Movement	Valid state transition of QEA/QEB	*QDEC_INTFL.move*	*QDEC_INTEN.move*	None

17.4 Clock and Sampling Selection

17.4.1 QCLK Generation

Timing is derived from the PCLK, which is in turn derived from the selected system oscillator. A programmable clock divider scales PCLK to provide the required QDEC clock rate, QCLK. In general, the divisor should be selected to operate the peripheral at the lowest QCLK that meets the application's requirements.

The required clock rate of f_{QCLK} is determined by the number of slots corresponding to a full rotation and rotation speed. Variables such as the clock divisor and sampling mode can be configured to meet the application requirements, as shown in *Equation 11-3*.

Note: f_{PCLK} is dependent on the system oscillator source f_{SYS_OSC}. In general, the highest measurement mode multiplier possible should be used to provide the best resolution.

Equation 17-1: QCLK Generation

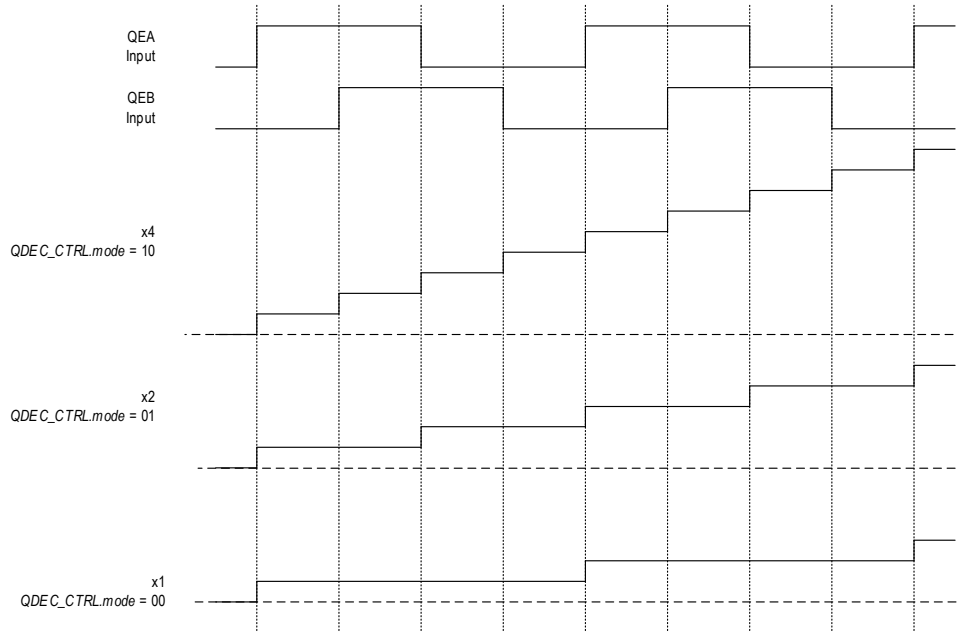
$$f_{QCLK} = \frac{f_{PCLK} \times 4}{clock\ divisor\ value} \geq \frac{f_{ROTATION} \times slots}{60}$$

The interrupt latency must be considered for high-speed, high-precision applications and is inversely proportional to f_{QCLK}. A QDEC event must be acknowledged and serviced before the next QCLK tick. The maximum latency time is application-specific and defined by the system designer.

17.4.2 Resolution

The QDEC can be adjusted to increase the encoder position's resolution by incrementing on one or more edges of the transition states, as shown in *Figure 17-4*.

Figure 17-4: QDEC Measurement Modes



The measurement mode is configured by the *QDEC_CTRL.mode* field, as shown in *Table 17-3*. The highest resolution mode, x4, increments the counter on all transitions of QEA and QEB and is preferred except in instances where the QCLK requirement can only be met with a slower clock mode.

Table 17-3: Measurement Modes

QDEC_CTRL.mode	Count Increments	Measurement Mode Multiplier
0b00	QEA rising	1
0b01	QEA rising, QEA falling	2
0b10	QEA rising, QEA falling, QEB rising, QEB falling	4
0b11	Reserved	Reserved

17.4.3 Sample Filtering

A programmable digital filter reject signal pulses shorter than a specified duration. This increases the noise immunity of the QDEC in noisy industrial environments.

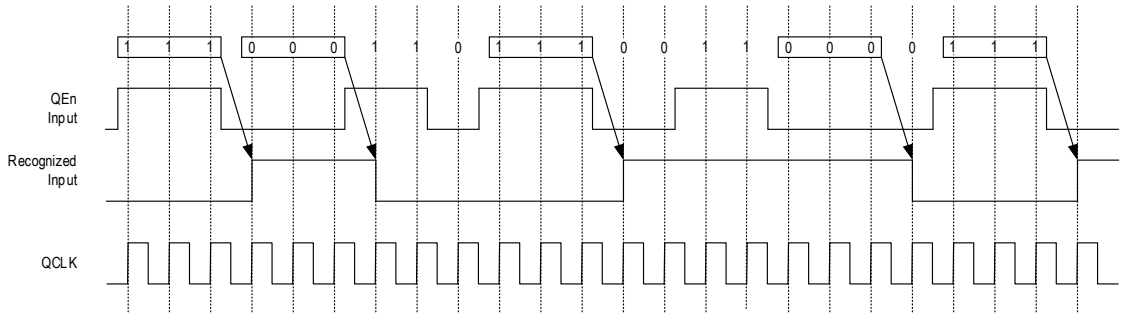
The filter can be programmed so that a transition must be valid for 1, 2, 3, or 4 consecutive QCLK ticks before being recognized. The number of ticks is configured by the *QDEC_CTRL.mode* field, as shown in *Table 17-4*.

Table 17-4: Samples per Mode

QDEC_CTRL.filter	Samples Required
0b00	1
0b01	2
0b10	3
0b11	4

Figure 17-5 is an example case showing the raw input and the recognized transitions in the three-sample mode. In this mode, when high or low for three consecutive samples in the opposite state are required before the change is recognized. *Figure 17-5* shows how pulses shorter than three QCLK periods are rejected and not recognized as a change of the input signal.

Figure 17-5: Input Sampling, Three-Sample Mode Example



17.5 State Transition Error

An error event occurs when an invalid state transition occurs, according to *Table 17-2*. An error event sets the error event interrupt flag and sets the QERR output signal to its active state.

The error event interrupt flag field can be configured to be set only on the clock tick when the error event occurs (*QDEC_CTRL*.sticky = 0) and then clear automatically or remain set until cleared by software (*QDEC_CTRL*.sticky = 1).

17.6 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, soft reset, POR, and the peripheral-specific resets.

Table 17-5: QDEC Register Summary

Offset	Register	Description
[0x0000]	*QDEC_CTRL*	QDEC Control Register
[0x0004]	*QDEC_INTFL*	QDEC Interrupt Flag Register
[0x0008]	*QDEC_INTEN*	QDEC Interrupt Enable Register
[0x000C]	*QDEC_MAXCNT*	QDEC Maximum Count Register
[0x0010]	*QDEC_INITIAL*	QDEC Initial Count Register
[0X0014]	*QDEC_COMPARE*	QDEC Compare Register
[0x0018]	*QDEC_INDEX*	QDEC Index Register
[0x001C]	*QDEC_CAPTURE*	QDEC Capture Register
[0x0020]	*QDEC_STATUS*	QDEC Status Register
[0x0024]	*QDEC_POSITION*	QDEC Count Register

17.6.1 Register Details

Table 17-6: QDEC Control Register

QDEC Control			QDEC_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:19	-	RO	0	**Reserved**
18:16	psc	R/W	0	**QDEC Clock Divisor Ratio** 0b000: $QCLK = \frac{PCLK}{1}$ 0b001: $QCLK = \frac{PCLK}{2}$ 0b010: $QCLK = \frac{PCLK}{4}$ 0b011: $QCLK = \frac{PCLK}{8}$ 0b100: $QCLK = \frac{PCLK}{16}$ 0b101: $QCLK = \frac{PCLK}{32}$ 0b110: $QCLK = \frac{PCLK}{64}$ 0b111: $QCLK = \frac{PCLK}{128}$
15:9	-	RO	0	**Reserved**
8	sticky	R/W	0	**GPIO Output Latch** 0: QMATCH and QERR pulse active for one clock cycle when the condition occurs. 1: QMATCH always mirrors the state of *QDEC_INTFL.compare,* and QERR always mirrors the state of *QDEC_INTFL.qerr*.
7	rst_maxcnt	R/W	0	**Reset on Maximum Count Match** Defines whether a maximum count event occurs when *QDEC_POSITION.position* = *QDEC_MAXCNT.maxcnt*. 0: No effect 1: A maximum count event occurs.
6	rst_index	R/W	0	**Reset on Index** Reset the counter when the index pulse is detected. 0: The state of the QEI input does not affect *QDEC_POSITION.position*. 1: *QDEC_POSITION.position* reset to *QDEC_INITIAL.initial*
5:4	filter	R/W	0	**Input Signal Filter** A state change on a signal is only recognized if it is sampled in the same state for the minimum number of QCLK cycles. 0: 1 sample (no filtering) 1: 2 samples 2: 3 samples 3: 4 samples
3	swap	R/W	0	**Phase Swap** 0: Clockwise: QEA leads QEB by 90° Counterclockwise: QEB leads QEA by 90° 1: Clockwise: QEB leads QEA by 90° Counterclockwise: QEA leads QEB by 90°
2:1	mode	R/W	0b00	**Counter Mode** 0: Mode x1 1: Mode x2 2: Mode x4 3: Reserved

QDEC Control			QDEC_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
0	en	R/W	0	**Enable** The peripheral must be disabled when writing to any of the counter registers or the interrupt enable registers. 0: Disabled 1: Enabled

Table 17-7: QDEC Interrupt Flag Register

QDEC Interrupt Flag			QDEC_INTFL	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6	move	R/W1C	0	**Shaft Moved Event** 0: Normal operation 1: Value of *QDEC_POSITION*.*position* changed
5	dir	R/W1C	0	**Direction Change Event** 0: Normal operation 1: Direction of rotation changed
4	capture	R/W1C	0	**QES/CAPTURE Event** 0: Normal operation 1: Active transition on QES input.
3	maxcnt	R/W1C	0	**Maximum Count Match Event** 0: Normal operation 1: Position = maximum count
2	compare	R/W1C	0	**Compare Match Event** A compare match event occurred. 0: COMPARE is not matched 1: COMPARE is matched
1	qerr	R/W1C	0	**Error Event** An error event occurred. 0: Normal operation 1: Error occurred
0	index	R/W1C	0	**Index Event** 0: Normal operation 1: Event occurred

Table 17-8: QDEC Interrupt Enable Register

QDEC Interrupt Enable			QDEC_INTEN	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6	move	R/W1C	0	**Shaft Moved Event Interrupt Enable** 0: Disabled 1: Enabled
5	dir	R/W1C	0	**Direction Change Event Interrupt Enable** 0: Disabled 1: Enabled
4	capture	R/W1C	0	**QES/CAPTURE Event Interrupt Enable** 0: Disabled 1: Enabled

QDEC Interrupt Enable			QDEC_INTEN	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
3	maxcnt	R/W1C	0	**Maximum Count Match Event Interrupt Enable** 0: Disabled 1: Enabled
2	compare	R/W1C	0	**Compare Match Event Interrupt Enable** 0: Disabled 1: Enabled
1	qerr	R/W1C	0	**Error Event Interrupt Enable** 0: Disabled 1: Enabled
0	Index	R/W1C	0	**Index Event Interrupt Enable** 0: Disabled 1: Enabled

Table 17-9: QDEC Maximum Count Register

QDEC Maximum Count			QDEC_MAXCNT	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	maxcnt	R/W	0xFFFF FFFF	**Position Maximum Count Value**

Table 17-10: QDEC Initial Value Register

QDEC Initial Value			QDEC_INITIAL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	initial	R/W	0	**Position Initialization Value**

Table 17-11: QDEC Compare Register

QDEC Compare			QDEC_COMPARE	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	compare	R/W	0	**Position Compare Value**

Table 17-12: QDEC Index Register

QDEC Index			QDEC_INDEX	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	index	R/W	0	**Position Compare Value**

Table 17-13: QDEC Capture Register

QDEC Capture			QDEC_CAPTURE	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	capture	R/W	0	**Position Capture Value**

Table 17-14: QDEC Status Register

QDEC Status			QDEC_STATUS	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**

QDEC Status			QDEC_STATUS	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
0	dir	R	0	**Latched QDIR** 0: Counterclockwise 1: Clockwise

Table 17-15: QDEC Position Register

QDEC Position			QDEC_POSITION	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	position	R/W	0	**Position Counter**

Table 17-16: QDEC Capture Delay Register

QDEC Capture Delay			QDEC_CAPDLY	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0x0000 03FF	**Capture Delay** This field specifies the number of PCLK cycles the CAPTURE signal is delayed. A minimum of 20µs is required.

18. Analog-to-Digital Converter (ADC)

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

The device samples any or all of the inputs in a user-defined conversion sequence which 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.
- Internal reference without external capacitor.
  - 1.25V or 2.048V selectable.
- Support for external reference from 2.048V to V_{DDA}.
- Capacitor calibration.
- Internal die temperature sensor.

18.1 Operation

Measurements are performed in a series of user-defined channel measurements called a conversion sequence. Conversion sequences can be set up as a single conversion sequence or continuous conversion sequences. 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_CHSEL3*:*ADC_CHSEL0*) and the *slot0_id* through *slot15_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*). The number of slots is zero-based, where 0 equals 1 slot.

Each measurement is pushed onto the FIFO to be read by software. Several data formats are available to the user.

18.1.1 Input Channels

Each of the input channels is shown in *Table 18-1*.

Table 18-1: MAX32672 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	AIN8	Single-ended	AIN8
9	AIN9	Single-ended	AIN9
10	AIN10	Single-ended	AIN10

Channel ID	Source	Mode	Alternate Function Name[1]
11	AIN11	Single-ended	AIN11
12	V_{DDA}	Single-ended	N/A
13	Temperature Sensor	Differential	N/A
14	V_{CORE}	Single-ended	N/A
15	V_{SS}	Single-ended	N/A

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

Table 18-2: ADC Voltage Divider Configuration for Channels 0 through 12

Setting	Divider Selection *MCR_ADC_CFG2.ch<n>*[1]	Dynamic Pullup Enable *MCR_ADC_CFG1.ch<n>_pu_dyn*[1]	Automatic Disable During Device Low Power Modes (Channels 0 to 11)[2]
Pass-through divide by 1	0	N/A	0
Voltage divide by 2, 5kΩ	1	0: Divider enabled always 1: Divider enabled only when channel active	*MCR_ADC_CFG0.lp_5k_dis* = 1
Voltage divide by 2, 50kΩ	2		*MCR_ADC_CFG0.lp_50k_dis* = 1

1. <n> = Channel number (0 to 12)

2. The disable settings only apply to channels 0 through 11. Channel 12's pullup, if enabled, is always disabled during low-power modes.

18.2 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 startup time. *Table 18-3* shows the ADC clock sources available for the device.

Table 18-3: MAX32672 ADC Clock Sources

ADC_CLKCTRL.clksel	Source (f_{ADC_SRC})
0	SYS_CLK
1	EXT_CLK1 (P0.28 / AF2)
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 18-1*. *Table 18-3* lists the available sources for f_{ADC_SRC}. The clock divider is selected using the *ADC_CLKCTRL.clkdiv* field.

Equation 18-1: ADC Clock Generation

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

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

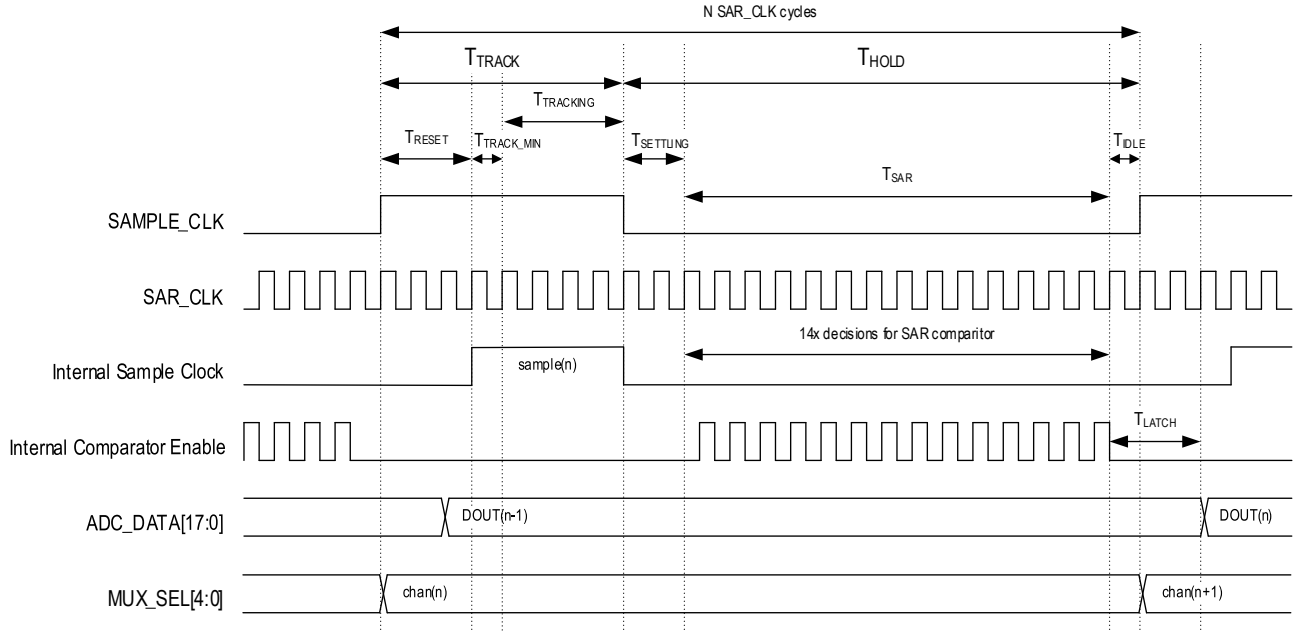
$$f_{SAR\_CLK} \leq 25MHz$$

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 18-1* shows the SAMPLE_CLK and its relationship to the track and hold values.

Equation 18-2: Sample Clock Frequency Calculation

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

Figure 18-1: 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 18-3: T_{TRACK} Calculation

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

Equation 18-4: 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 18-3* and *Equation 18-4*.

As an example, the following steps show the settings required to achieve a 1MSPS 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 18-1*.
   a. Set *ADC_CLKCTRL*.*clkdiv* to 1 (divide by 4, $f_{SAR\_CLK} \leq 25MHz$)
   b. $t_{SAR\_CLK} = 40ns$
3. Determine the SAMPLE_CLK for 1MSPS
   a. $TRACK + HOLD = \frac{25MHz}{1MHz} = 25$
4. Determine the *ADC_SAMPCLKCTRL*.*track_cnt* setting using *Equation 18-3*.
   a. *ADC_SAMPCLKCTRL*.*track_cnt* = 4 ($T_{TRACK} \geq 8$)
5. Determine the *ADC_SAMPCLKCTRL*.*idle_cnt* setting using *Equation 18-4*.
   a. HOLD = 25 - T_{TRACK} = 25 - 8 = 17
   b. *ADC_SAMPCLKCTRL*.*idle_cnt* = 0 ($T_{HOLD} \geq 17$)

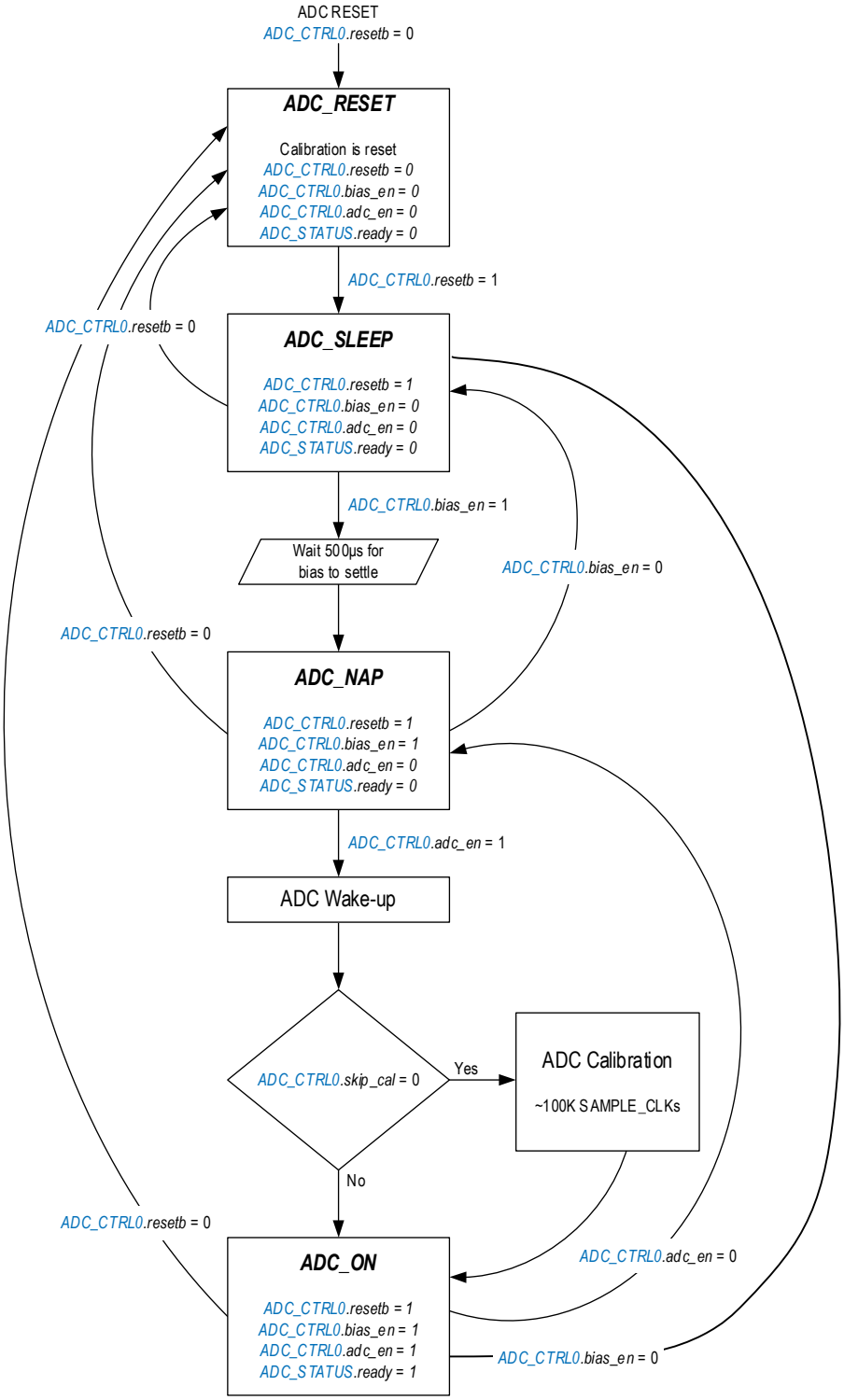
18.3 Operating Modes

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 *18.3.1* describes initializing the ADC from *ADC_RESET*. Section *18.3.1.2* describes entering *ADC_NAP* state. Section *18.3.1.3* describes the steps required to enter the *ADC_ON* state and perform an ADC capacitor calibration, and section *18.3.1.4* describes the steps to enter the *ADC_ON* state without performing a calibration. ADC calibration is only necessary after an ADC reset occurs, 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 18-2* shows the ADC operating modes state diagram. *Table 18-4* shows the configuration bits' state and the status bit's state for each operating mode.

Table 18-4: 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 18-2: 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 0, the device performs the ADC auto-calibration, which takes approximately 100ms to complete. After the auto-calibration is complete, or immediately if it was 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.

18.3.1 ADC Initialization

18.3.1.1 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 18-3* and set it to the selected clock using the *ADC_CLKCTRL.clksel* field.
4. Configure the SAR_CLK as described in *Clocks and Timing* using the *ADC_CLKCTRL.clkdiv* field.
5. Clear *PWRSEQ_LPCN.vbbmon_dis* to 0
6. Select the ADC reference source:
   - External: *MCR_ADC_CFG0.ext_ref* to 1
   - Internal, 1.25V: Clear *MCR_ADC_CFG0.ext_ref* to 0 and clear *MCR_ADC_CFG0.ref_sel* to 0.
   - Internal, 2.048: Clear *MCR_ADC_CFG0.ext_ref* to 0 and set *MCR_ADC_CFG0.ref_sel* to 1.
     - If using the 2.048V internal reference, V_{DDA} must be ≥ 2.048V.
7. If desired, enable the external input voltage dividers for the desired channels, as shown in *Table 18-2*. The voltage divider can always be active or only during the channel measurement. It can also be configured to disable the voltage divider during low-power modes.
8. Configure the GPIO associated with the desired external channels as inputs in high impedance mode. Configure the alternate function mode as indicated in *Table 18-1*.
9. Set the *ADC_CTRL0.resetb* to 1 to enter the *ADC_SLEEP* state.

18.3.1.2 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.
3. The ADC is now in the *ADC_NAP* state.

18.3.1.3 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 to perform 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, bias, and wake-up counter settings. See the *ADC SFR Interface* section 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.

18.3.1.4 Entering ADC_ON State Skipping Calibration

If calibration has previously been 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.

18.4 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.

18.4.1 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 18-5* 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 1MSPS.

1. The bias counter must be 500µs, which results in $1MHz \times 500\mu s = 500$ cycles.
   a. Referring to *Table 18-5*, the closest bias counter setting to achieve at least 500 cycles is 7 (512 clock cycles).
2. The wake-up counter must be 30µs, which results in $1MHz \times 30\mu s = 30$ cycles.
   a. Referring to *Table 18-5*, the closest wake-up counter setting to achieve at least 30 cycles is 11 (32 clock cycles).

Table 18-5: 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

18.4.2 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.

Loading of the trim data varies by device revision and reference used. Software must determine the device revision and load the trim values based on the revision for the specific reference used. Read the *GCR_REVISION* register to determine the device revision.

18.4.3 1.25V Internal Reference Trim for Device Revision 0xA1

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_ADCREFTRIM0.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. *FCR_ADCREFTRIM2.iboost_1p25* << 7 | *FCR_ADCREFTRIM0.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_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_SFRADDR* register and store it in a variable, *read_sfr*.
15. Mask off the upper two bits of the lower nibble of the *read_sfr* variable by performing a bit-wise AND of the value read with 0x0C.
    a. *read_sfr* &= 0x0C;
16. Perform the following operation on the *read_sfr* variable.
    a. *read_sfr* |= (*FCR_ADCREFTRIM2.idrv_1p25* << 4) | *FCR_ADCREFTRIM0.vcm*
17. Write the *read_sfr* variable to the *ADC_SFRWRDATA* register.
18. Write address 0x05 to *ADC_SFRADDR*.
19. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
20. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
21. Perform a bit-wise OR of the data from step 20 and the bias counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
22. Write address 0x06 to *ADC_SFRADDR*.
23. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
24. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
25. Perform a bit-wise OR of the data from step 24 and the calculated wake-up counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
26. Move the ADC into the ADC_ON state using calibration.

18.4.4 1.25V Internal Reference Trim for All Other Revisions

1. Write address 0x01 to *ADC_SFRADDR*.
2. Read the SFR data by reading a byte from the *ADC_SFRADDR* register.
3. Perform a bit-wise OR of the result of step 2 with 0x0F.
4. Write the byte from step 3 to the *ADC_SFRWRDATA* register.
5. Write address 0x0B to *ADC_SFRADDR*.
6. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
7. Mask off the upper two bits of the data read by performing a bit-wise AND of the value read with 0xC0.
8. Perform a bit-wise OR of the result of step 7 with *FCR_ADCREFTRIM0*.*vx2_tune*.
9. Write the byte from step 8 to the *ADC_SFRWRDATA* register.
10. Write *FCR_ADCREFTRIM0*.*vcm* to *MCR_ADC_CFG3*.*vcm*.
11. Write *FCR_ADCREFTRIM0*.*vrefm* to *MCR_ADC_CFG3*.*vrefm*.
12. Write *FCR_ADCREFTRIM0*.*vrefp* to *MCR_ADC_CFG3*.*vrefp*.
13. Write *FCR_ADCREFTRIM2*.*iboost_1p25* to *MCR_ADC_CFG3*.*d_iboost*.
14. Write *FCR_ADCREFTRIM2*.*idrv_1p25* to *MCR_ADC_CFG3*.*idrv*.
15. Write address 0x05 to *ADC_SFRADDR*.
16. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
17. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
18. Perform a bit-wise OR of the data from step 17 and the bias counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
19. Write address 0x06 to *ADC_SFRADDR*.
20. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
21. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
22. Perform a bit-wise OR of the data from step 21 and the calculated wake-up counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
23. Move the ADC into the ADC_ON state using calibration.

18.4.5 2.048V Internal Reference Trim for Device Revision 0xA1

1. Write address 0x01 to *ADC_SFRADDR*.
2. Read the SFR data by reading a byte from the *ADC_SFRADDR* register.
3. Perform a bit-wise OR of the result of step 2 with 0x0F.
4. Write the byte from step 3 to the *ADC_SFRWRDATA* register.
5. Write address 0x0B to *ADC_SFRADDR*.
6. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
7. Mask off the upper two bits of the data read by performing a bit-wise AND of the value read with 0xC0.
8. Perform a bit-wise OR of the result of step 7 with *FCR_ADCREFTRIM1*.*vx2_tune*.
9. Write the byte from step 8 to the *ADC_SFRWRDATA* register.
10. Write address 0x0C to *ADC_SFRADDR*.
11. Write the following byte to the *ADC_SFRWRDATA* register.
    a. *FCR_ADCREFTRIM2*.*iboost_2p048* << 7 | *FCR_ADCREFTRIM1*.*vrefp*
12. Write address 0x0D to *ADC_SFRADDR*.
13. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
14. Mask off the upper bit of the data read by performing a bit-wise AND of the value read with 0x80.
15. Perform a bit-wise OR of the result of step 14 with *FCR_ADCREFTRIM1*.*vrefm*.
16. Write the byte from step 15 to the *ADC_SFRWRDATA* register.
17. Write address 0x0E to *ADC_SFRADDR*.
18. Read the SFR data by reading a byte from the *ADC_SFRADDR* register and store it in a variable, *read_sfr*.
19. Mask off the upper two bits of the lower nibble of the *read_sfr* variable by performing a bit-wise AND of the value read with 0x0C.
    a. *read_sfr* &= 0x0C;
20. Perform the following operation on the *read_sfr* variable.
    a. *read_sfr* |= (*FCR_ADCREFTRIM2*.*idrv_2p048* << 4) | *FCR_ADCREFTRIM1*.*vcm*
21. Write the *read_sfr* variable to the *ADC_SFRWRDATA* register.
22. Write address 0x05 to *ADC_SFRADDR*.
23. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
24. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
25. Perform a bit-wise OR of the data from step 24 and the bias counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
26. Write address 0x06 to *ADC_SFRADDR*.
27. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
28. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
29. Perform a bit-wise OR of the data from step 28 and the calculated wake-up counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
30. Move the ADC into the ADC_ON state using calibration.

18.4.6 2.048V Internal Reference Trim for All Other Revisions

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

1. Write address 0x01 to *ADC_SFRADDR*.
2. Read the SFR data by reading a byte from the *ADC_SFRADDR* register.
3. Perform a bit-wise OR of the result of step 2 with 0x0F.
4. Write the byte from step 3 to the *ADC_SFRWRDATA* register.
5. Write address 0x0B to *ADC_SFRADDR*.
6. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
7. Mask off the upper two bits of the data read by performing a bit-wise AND of the value read with 0xC0.
8. Perform a bit-wise OR of the result of step 7 with *FCR_ADCREFTRIM1*.*vx2_tune*.
9. Write the byte from step 8 to the *ADC_SFRWRDATA* register.
10. Write *FCR_ADCREFTRIM1*.*vcm* to *MCR_ADC_CFG3*.*vcm*.
11. Write *FCR_ADCREFTRIM1*.*vrefm* to *MCR_ADC_CFG3*.*vrefm*.
12. Write *FCR_ADCREFTRIM1*.*vrefp* to *MCR_ADC_CFG3*.*vrefp*.
13. Write *FCR_ADCREFTRIM2*.*iboost_2p048* to *MCR_ADC_CFG3*.*d_iboost*.
14. Write *FCR_ADCREFTRIM2*.*idrv_2p048* to *MCR_ADC_CFG3*.*idrv*.
15. Write address 0x05 to *ADC_SFRADDR*.
16. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
17. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
18. Perform a bit-wise OR of the data from step 17 and the bias counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
19. Write address 0x06 to *ADC_SFRADDR*.
20. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
21. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
22. Perform a bit-wise OR of the data from step 21 and the calculated wake-up counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
23. Move the ADC into the ADC_ON state using calibration.

18.4.7 External Reference Trim for Device Revision 0xA1

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 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_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. Perform a bit-wise AND of the result of step 7 with 0xFC.
9. Perform a bit-wise OR of the data from step 8 and *FCR_ADCREFTRIM2*.*vcm* and write this byte to the *ADC_SFRWRDATA* register.
10. Write address 0x05 to *ADC_SFRADDR*.
11. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
12. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
13. Perform a bit-wise OR of the data from step 12 and the bias counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
14. Write address 0x06 to *ADC_SFRADDR*.
15. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
16. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
17. Perform a bit-wise OR of the data from step 16 and the calculated wake-up counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
18. Move the ADC into the ADC_ON state using calibration.

18.4.8 External Reference Trim for All Other Device Revisions

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

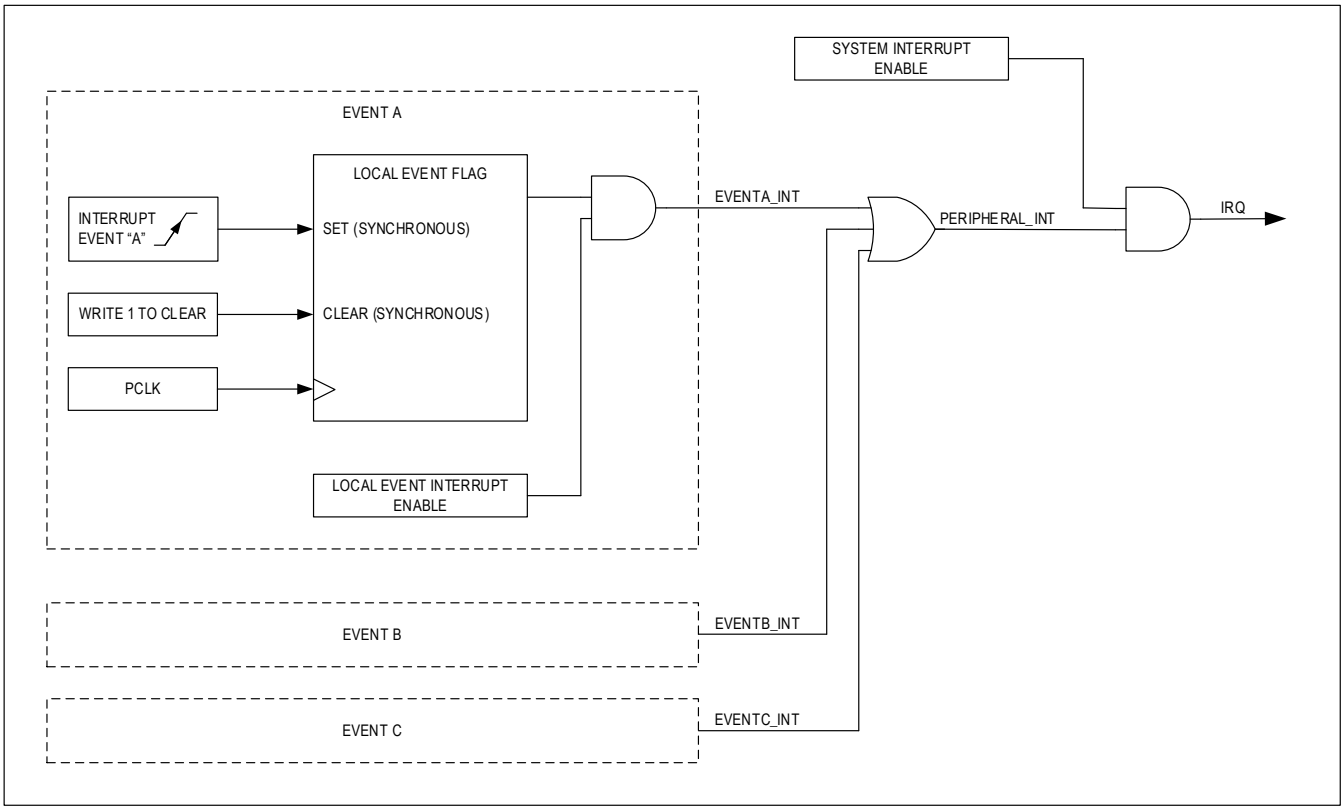
1. Write address 0x01 to *ADC_SFRADDR*.
2. Read the SFR data by reading a byte from the *ADC_SFRADDR* register.
3. Perform a bit-wise OR of the result of step 2 with 0x0F.
4. Write the byte from step 3 to the *ADC_SFRWRDATA* register.
5. Write address 0x0B to *ADC_SFRADDR*.
6. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
7. Mask off the upper bits of the data read by performing a bit-wise AND of the value read with 0xC0.
8. Perform a bit-wise OR of the result of step 7 with *FCR_ADCREFTRIM2*.*vx2_tune*.
9. Write the byte from step 8 to the SFR by writing it to the *ADC_SFRWRDATA* register.
10. Write *FCR_ADCREFTRIM2*.*vcm* to *MCR_ADC_CFG3*.*vcm*.
11. Write address 0x05 to *ADC_SFRADDR*.
12. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
13. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
14. Perform a bit-wise OR of the data from step 13 and the bias counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
15. Write address 0x06 to *ADC_SFRADDR*.
16. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
17. Mask off bits 4–7 of the data read by performing a bit-wise AND of the value read with 0xF0.
18. Perform a bit-wise OR of the data from step 17 and the calculated wake-up counter setting and write this byte to the *ADC_SFRWRDATA* register. See *Determination of Bias and Wake-up Counter Settings* to calculate this value.
19. Move the ADC into the ADC_ON state using calibration.

18.5 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 18-3* is a functional diagram showing this relationship.

Figure 18-3: 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 18-6*.

Table 18-6: MAX32672 Interrupt Events

Event	Description	Interrupt Flag	Interrupt Enable
FIFO Threshold	*ADC_STATUS.fifo_level* > *ADC_FIFODMACTRL.thresh*.	*ADC_INTFL.fifo_lvl*	*ADC_INTEN.fifo_lvl*
FIFO Overflow	Hardware FIFO write when the FIFO is full.	*ADC_INTFL.fifo_ofl*	*ADC_INTEN.fifo_ofl*
FIFO Underflow	Read from FIFO when *ADC_STATUS.fifo_level* = 0	*ADC_INTFL.fifo_ufl*	*ADC_INTEN.fifo_ufl*
Data Clipped	An ADC measurement has been 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*

18.6 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 taken and the FIFO is full (*ADC_STATUS*.*fifo_level* = 15). When a write occurs and the FIFO is full, the previous data is overwritten. It is possible to use the channel ID field in the *ADC_DATA* register to determine which measurement results have been overwritten. The FIFO should be flushed and the current conversion sequence restarted.
- A FIFO underflow event occurs when the FIFO is read when *ADC_STATUS*.*fifo_level* is equal to 0.

18.7 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.

18.8 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 18-7*. A visual representation of the corresponding format of the *ADC_DATA* register for the temperature sensor input is shown in *Figure 18-5*, and all other input channels are shown in *Figure 18-4*.

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 was beyond the ADC limits (either positive or negative). For an averaged measurement, the result is marked clipped if any of the samples were 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 18-7*.

Table 18-7: 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

Mode	*ADC_FIFODMACTRL.data_format*	Format code	Channel ID	Clipped	Invalid Flag	Data Format
Differential (Temperature Sensor Only)	0	Data and Status	Yes	Yes	Yes	12-bit signed 2's complement
	1	Data Only	No	*ADC_CHSTATUS*	No	12-bit signed 2's complement
	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 18-3*.

Figure 18-4: ADC Result Formats (Single-Ended)

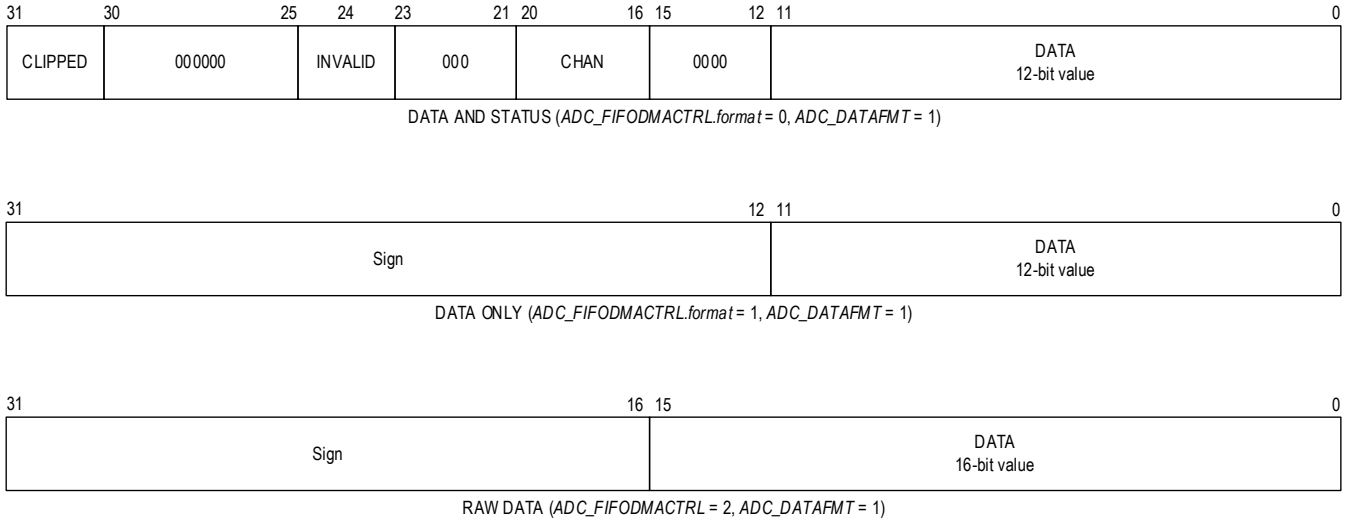
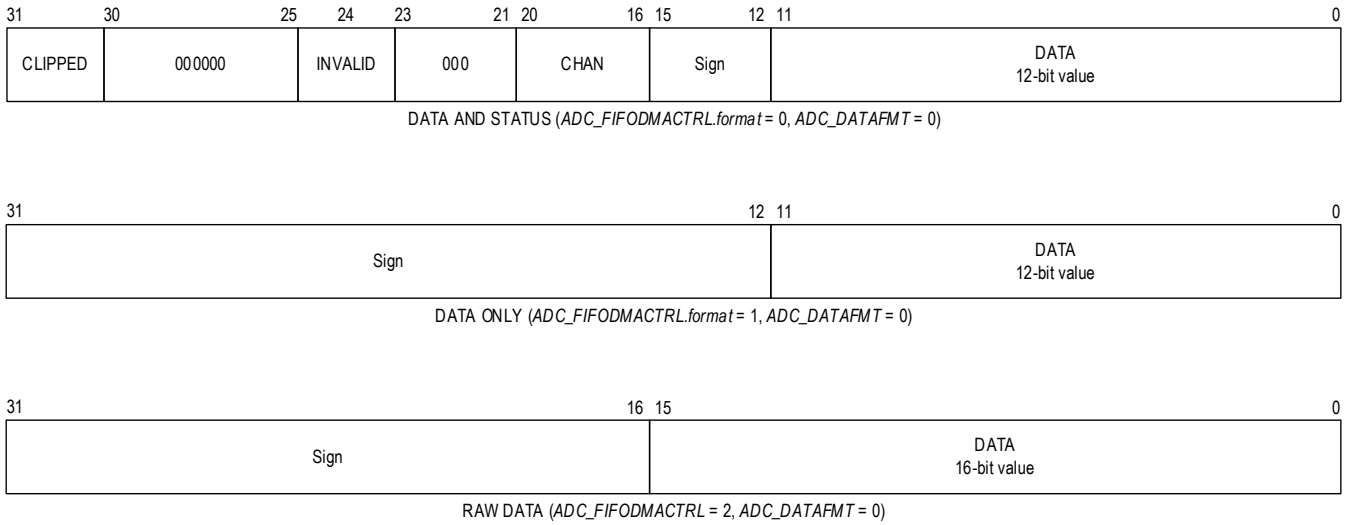


Figure 18-5: ADC Result Formats (Differential, Temperature Sensor Only)



18.9 Conversions

18.9.1 Conversion Sequence Triggers

A conversion sequence is initiated by either a software or hardware trigger. This flexibility allows either manual or on-demand measurements using a timer peripheral or an external GPIO. *Table 18-8* lists the hardware triggers available to start a conversion sequence.

Table 18-8: MAX32672 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	Temperature sensor measurement ready
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 18-8*, 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 Configuration* in the GPIO chapter for details on configuring port pin alternate functions.

Table 18-9: 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

Conversion Sequence Type	Sequence Start	ADC_CTRL1.cnv_mode	ADC_CTRL1.trig_mode
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

18.9.2 Single Conversion Sequences

18.9.2.1 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_CHSEL3*:*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 has occurred.
- Software should set *ADC_CTRL1.start* to 0 in to prevent additional conversions.
- The converted data is available in the *ADC_DATA* register. See *FIFO Operation* for details on the FIFO.

18.9.2.2 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 18-8* 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_CHSEL3*:*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.
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 has 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.

18.9.3 Continuous Conversion Sequences

18.9.3.1 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_CHSEL3*:*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 is complete.
- 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.

18.9.3.2 Hardware-Triggered, Continuous Conversion Sequence

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 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 18-8* for a list of hardware triggers.
5. Configure the selected hardware trigger. See *Timers (TMR/LPTMR)* for timer configuration and *Alternate Function Configuration* 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_CHSEL3*:*ADC_CHSEL0* registers slot fields.
8. Configure *ADC_CTRL1*.*avg* to the desired sample averaging.
    a. 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 has 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.

18.9.4 Temperature Sensor

The internal temperature sensor provides a measurement of die temperature. Depending on the application, environmental changes might necessitate recalibration of the RTC or ADC. The temperature sensor returns its results in a differential measurement format. The temperature measurement takes approximately 500µs for a measurement. This time is required after the temperature sensor is enabled. Measuring the temperature sensor requires a sequence of channels. If the actual measurement of the temperature sensor occurs before the temperature sensor is ready, the measurement is marked as invalid.

In order to perform a temperature sensor conversion, a minimum of two channels must be converted. The first channel or the channel immediately before the temperature sensor must be V_{DDA}. Additionally, the IBRO must be enabled for the temperature sensor conversion to complete correctly.

The following steps describe how to measure the temperature sensor using a hardware triggered, single conversion sequence:

1. Configure the ADC and enter the *ADC_ON* state as described in *Operating Modes*.
2. Enable the IBRO by setting *GCR_CLKCTRL.ibro_en* to 1.
3. Clear the *ADC_CTRL1.start* field to 0.
4. Set *ADC_CTRL1.trig_mode* to 1 to select hardware triggering.
5. Set *ADC_CTRL1.trig_sel* to 7 to select the temperature sensor as the hardware trigger.
6. Set *ADC_CTRL1.cnv_mode* to 0 to select a single conversion sequence.
7. Select the number of channels to convert by setting the *ADC_CTRL1.num_slots* to 2 to measure 3 channels.
8. Set *ADC_CHSEL0.slot0_id* to 12 to select V_{DDA} as the first channel in the sequence.
9. Set *ADC_CHSEL0.slot1_id* to 13 to select the temperature sensor as the second channel.
10. Set *ADC_CHSEL0.slot2_id* to 12 to select V_{DDA} as the third channel in the sequence.
11. Set *ADC_CTRL1.avg* to 0 for a single conversion.
10. Set the data format for the conversion results using the *ADC_FIFODMACTRL.data_format* field. See *Conversion Results* for details. The temperature sensor is a differential measurement and always returns a signed value.
11. Clear the interrupt flags by writing 0xFFFF FFFF to the *ADC_INTFL* register.
12. Set the *ADC_CTRL1.ts_sel* field to 1 to enable the temperature sensor.
13. Set *ADC_CTRL1.start* to 1 to arm the conversion sequence. The conversion sequence begins when the temperature sensor is ready.
14. When the sequence is triggered, hardware sets the *ADC_INTFL.seq_started* field to 1.

At the end of the temperature sensor measurement:

- Hardware sets the *ADC_INTFL.seq_done* field to 1, indicating a sequence done event has 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 FIFO.
- The first read of the *ADC_DATA* register returns the value of V_{DDA}.
- The second read of the *ADC_DATA* register returns the value of the temperature sensor conversion. Use the appropriate equation below to convert the value read to a temperature value.
- The third read of the *ADC_DATA* register returns the value of V_{CORE}.

Equation 18-5: Temperature Conversion Equation for Device Revision 0xA1

$$T(°C) = \frac{Measured\ Code \times V_{REF} \times 530.582}{Full\text{-}Scale\ Code} - 273.15$$

Equation 18-6: Temperature Conversion Equation for All Other Device Revisions

$$T(°C) = \frac{Measured\ Code}{Full\text{-}Scale\ Code} \times \frac{V_{REF}}{2.048} \times FCR\_TS0 - \frac{FCR\_TS1}{16}$$

18.10 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 of the two analog comparators supports multiple analog inputs for the positive and negative inputs. See *Table 18-10* and *Table 18-11* for each comparator's input options and selection. Refer to the device data sheet's pin description table for alternate function assignment.

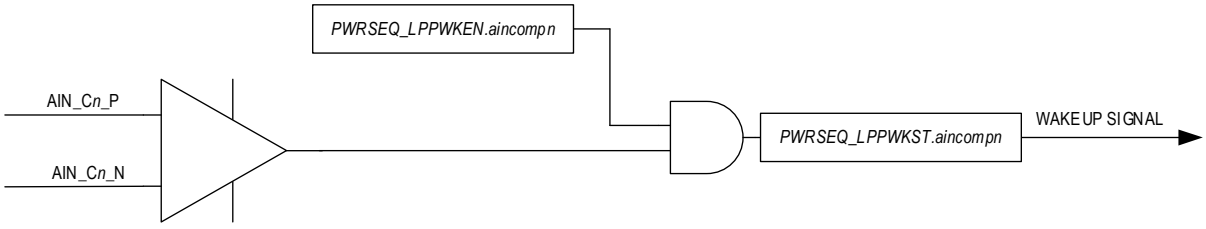
Table 18-10: MAX32672 Analog Comparator 0 Input Selection

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

Table 18-11: MAX32672 Analog Comparator 1 Input Selection

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

Figure 18-6: Analog Wakeup Comparators



The comparator status field dynamically shows the comparator output when both the corresponding positive and the 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 18-12*.

Table 18-12: MAX32672 Analog Wake-Up Comparator Fields

Comparator	Wake-up Flag	Wake-up Enable	Comparator Output
AINCOMP0	*PWRSEQ_LPPWKST.aincomp0*	*PWRSEQ_LPPWKEN.aincomp0*	*PWRSEQ_LPPWKST.aincomp0_out*
AINCOMP1	*PWRSEQ_LPPWKST.aincomp1*	*PWRSEQ_LPPWKEN.aincomp1*	*PWRSEQ_LPPWKST.aincomp1_out*

Configure the comparators as follows:

1. Select the comparator's inputs. See *Table 18-10* and *Table 18-11* for details on selecting each comparator's inputs.
2. Enable the selected input's alternate function for the GPIO. See *Alternate Function Configuration* in the GPIO chapter for details. Refer to the device data sheet alternate function table for pin assignments.
3. Set *MCR_AINCOMP.pd[0]* to 1 to enable comparator 0, set *MCR_AINCOMP.pd[1]* to 1 to enable comparator 1, or set both.
4. Clear the appropriate comparator wake-up flag:
   a. Write 1 to *PWRSEQ_LPPWKST.aincomp0* to clear comparator 0s wake-up flag
   b. Write 1 to *PWRSEQ_LPPWKST.aincomp1* to clear comparator 1s wake-up flag
5. If using the comparator as a wake-up source, set the comparator's wake-up enable flag.
   a. Write 1 to *PWRSEQ_LPPWKEN.aincomp0* to use comparator 0 as a wake-up source.
   b. Write 1 to *PWRSEQ_LPPWKEN.aincomp1* to use comparator 1 as a wake-up source.
6. Read the comparator's output.
   a. Read *PWRSEQ_LPPWKST.aincomp0_out* for comparator 0s output.
   b. Read *PWRSEQ_LPPWKST.aincomp1_out* for comparator 1s output.

18.11 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 18-13: 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
[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

Offset	Register	Description
[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

18.11.1 Register Details

Table 18-14: 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.
0	adc_en	R/W	0	**ADC Enable** 0: Disabled. 1: Enabled.

Table 18-15: 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. … : … 15: 15 slots. 16 – 31: Reserved.
15:11	-	RO	0	**Reserved**

ADC Control 1			ADC_CTRL1	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
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	ts_sel	R/W	0	**Temperature Sensor Select** 0: Disabled. 1: Enabled.
6:4	trig_sel	R/W	0	**Hardware Trigger Source** See *Table 18-8* 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 18-16: ADC Clock Control Register

ADC Clock Control			ADC_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6:4	clkdiv	R/W	3	**Clock Divider** See *Clocks and Timing* 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: Reserved.
3:2	-	RO	0	**Reserved**
1:0	clksel	R/W1C	0	**Clock Source** 0: SYS_OSC 1: EXT_CLK1 2: IBRO 3: ERFO

Table 18-17: 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 *Clocks and Timing* for details on determining the required setting for this field to achieve the desired sample rate.
15:0	track_cnt	R/W	0	**Sample Clock Track Time** The number of cycles to add to the minimum track time. See *Clocks and Timing* for details on determining the required setting for this field to achieve the desired sample rate.

Table 18-18: 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 18-19: 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**
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 18-20: 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 18-21: 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 18-22: 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 18-23: ADC Data Format Register

ADC Data Format			ADC_DATAFMT	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	mode	DNM	0xFFFF 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 except the temperature sensor, which operates in differential mode. Bit positions corresponding to unimplemented channels are ignored. 0: Differential mode. 1: Single-ended mode.

Table 18-24: 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.
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	W1O	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 18-25: 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 was 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, reading an invalid channel, reading the temperature sensor when the temperature sensor is not ready.
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.

ADC Data			ADC_DATA	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	data	R/W	0xFFFF	**Data** The format of this data is configurable. See *Conversion Results* for details.

Table 18-26: 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 18-27: 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 register identifies channels that have experienced a conversion result that was 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 18-28: 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.
9	fifo_ufl	W1C	0	**FIFO Underflow Event Interrupt Enable** 0: Disabled. 1: Enabled.

ADC Interrupt Enable			ADC_INTEN	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
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 18-29: 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.

ADC Interrupt Flags			ADC_INTFL	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
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 was started. 0: Normal operation. 1: Event occurred.
2	abort	R/W	0	**Command Aborted Event** The conversion command was aborted before conversions were 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 18-30: ADC SFR Address Offset Register

ADC SFR Address Offset			ADC_SFRADDROFFSET	[0x0060]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 18-31: ADC SFR Address Register

ADC SFR Address			ADC_SFRADDR	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 18-32: ADC SFR Write Data Register

ADC SFR Write Data			ADC_SFRWRDATA	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 18-33: ADC SFR Read Data Register

ADC SFR Read Data			ADC_SFRRDDATA	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 18-34: ADC SFR Status Register

ADC SFR Status			ADC_SFRSTATUS	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

19. Cryptographic Toolbox (CTB)

The cryptographic toolbox combines cryptographic engines and a secure cryptographic accelerator (SCA) 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.

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 with support for NIST-approved block modes:
  - DES/TDEA
- AES-128, -192, and -256 (FIPS 197).
  - CCM/GCM
- Parallel calculation of block cipher and hash functions
- Hash function accelerator supporting functions commonly used in HMAC:
  - SHA-1
- SHA-224
  - SHA-256
- SHA-384
  - SHA-512 (FIPS 180-3) values used in HMAC.
- Hamming code engine calculates hamming codes used in memory error correction code (ECC) algorithms.

The dedicated SCA performs high-speed calculations of elliptic curve functions that support:

- NIST elliptic curves secp256r1, secp384r1 and secp521r1
- Brainpool elliptic curves bp256r1 and bp384r1
- Custom elliptic curves (of 256, 384, and 521 bits)

Most functions are configurable for big- or little-endian operations.

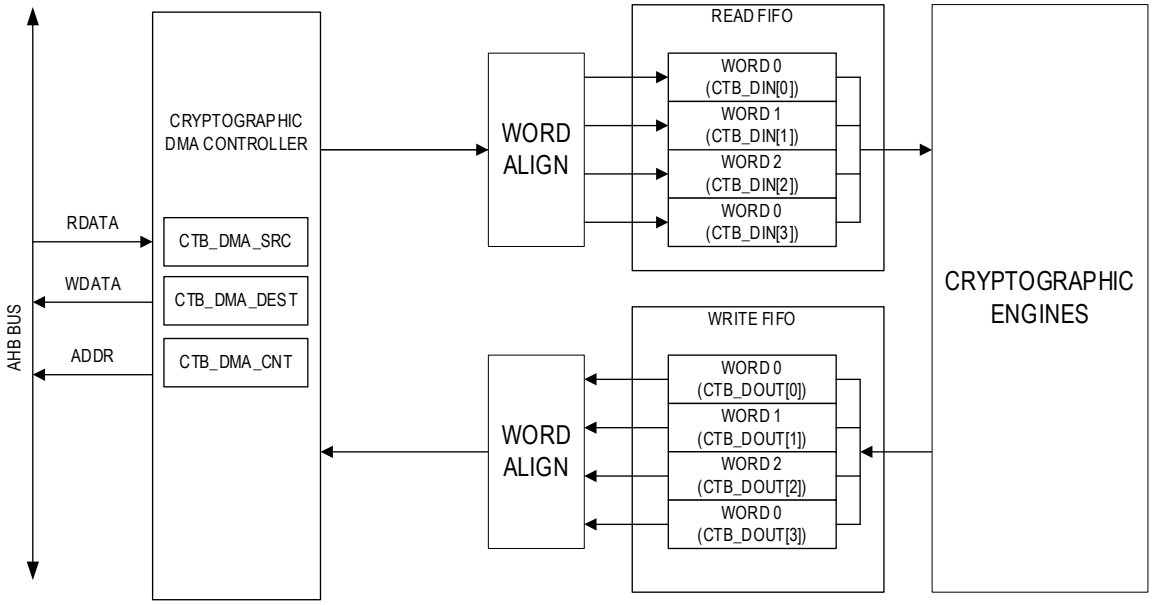
All cryptographic operations begin by resetting the CTB. The cryptographic engine functions have their own done bit and a global done bit for the cryptographic block. The engines can generate an interrupt if enabled.

19.1 Cryptographic DMA (CDMA)

A dedicated DMA engine performs high-speed accesses between the CTB and memory on the AHB bus. The DMA engine improves performance during data-intensive operations such as encryption/decryption and hashing. The source, destination, and count registers are 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 start the calculation on the next block immediately. This keeps the cipher and hash generator running continuously without waiting for data to be written to or read from the bus.

Figure 19-1: CDMA Block Diagram



19.2 FIFOs

The read FIFO and write FIFO have programmable sources shown in *Table 19-1* to allow flexibility in their operation.

Table 19-1: 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.

19.3 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
- TDEA
- DES

The symmetric block ciphers encrypt or decrypt data in blocks. The block sizes for each cipher are shown in *Table 19-2*.

Table 19-2: Symmetric Block Ciphers

Cipher	Key Size	Used Key Bits	Effective Strength (NIST SP800-57)	Block Size
DES	56-bits	*CTB_CIPHER_KEY[1]*:*CTB_CIPHER_KEY[0]*	-	64-bits
TDEA	192-bits	*CTB_CIPHER_KEY[5]*:*CTB_CIPHER_KEY[0]*	112-bits	64-bits
AES-128	128-bits	*CTB_CIPHER_KEY[3]*:*CTB_CIPHER_KEY[0]* or *USR_AESKEYS_KEY3*:*USR_AESKEYS_KEY0*	128-bits	128-bits
AES-192	192-bits	*CTB_CIPHER_KEY[5]*:*CTB_CIPHER_KEY[0]* or *USR_AESKEYS_KEY5*:*USR_AESKEYS_KEY0*	192-bits	128-bits
AES-256	256-bits	*CTB_CIPHER_KEY[7]*:*CTB_CIPHER_KEY[0]* or *USR_AESKEYS_KEY7*:*USR_AESKEYS_KEY0*	256-bits	128-bits

The accelerator supports the block cipher modes:

- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter (CTR)
- Counter with Cipher Block Chaining Message Authentication Code (CCM)
- Galois/Counter Mode GCM

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.

19.3.1 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* and *CTB_CTRL*.*dma_done* to 0.
2. Set *CTB_CIPHER_CTRL*.*src* to select the source of the keys.
3. Poll until hardware sets *CTB_CTRL*.*dma_done* to 1.
4. Clear *CTB_CTRL*.*done* and *CTB_CTRL*.*dma_done* to 0.

19.3.2 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* to 1 when complete.

1. Enable the CTB peripheral clock by clearing *GCR_PCLKDIS0*.*ctb* to 0.
2. Reset the engine by setting *CTB_CTRL*.*rst* = 1.
3. Poll until *CTB_CTRL*.*rdy* reads 1.
4. Select the cipher algorithm operation using *CTB_CIPHER_CTRL*.*cipher*.
5. Select the mode of operation using *CTB_CIPHER_CTRL*.*mode*.
6. Select encryption or decryption mode *CTB_CIPHER_CTRL*.*enc*.
7. 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.
8. Set *CTB_CTRL*.*rdsrc* to 1 to select the read FIFO source as DMA
9. Set *CTB_CTRL*.*wrsrc* to 1 to select the cipher output FIFO source as DMA
10. Load the DMA source address to *CTB_DMA_SRC*.
11. Load the DMA destination address to *CTB_DMA_DEST*.*addr*.
12. 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

19.3.3 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[3]*:*CTB_TAGMIC[0]* registers.

19.4 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 19-3*. 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.

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 CDMA.

Table 19-3: 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* 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 have been hashed.
2. If necessary, the 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.

19.4.1 Hash Operation

1. Enable the CTB peripheral clock by clearing *GCR_PCLKDIS0.ctb* to 0.
2. Reset the hash engine by setting *CTB_CTRL.rst* = 1.
3. Poll until hardware sets *CTB_CTRL.rdy* = 1.
4. Configure *CTB_HASH_CTRL.hash* to select the desired hash function.
5. Configure *CTB_HASH_MSG_SZ[0]* to the total number of bits to hash.
6. Set *CTB_CTRL.rdsrc* to 0b01 to select the read FIFO source as DMA.
7. Set *CTB_CTRL.wrsrc* to 0b01 to select the cipher output FIFO source as DMA.
8. Set *CTB_CTRL.intr* to 1 to enable an interrupt when the DMA transfer is complete.
9. Load the starting address of the input message to *CTB_DMA_SRC*.
10. Load *CTB_DMA_CNT* with the number of bits associated with whole blocks as shown in *Table 19-3*, 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.

19.5 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 SEC-DED. Three errors masquerade as a correctable single-bit error.

The Hamming code generator calculates the ECC on a block of data up to 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 errors that are uncorrectable. The engine generates even parity on even halves of bit groups.

19.5.1 Hamming Code Operation

1. Enable the CTB peripheral clock by clearing *GCR_PCLKDIS0.ctb* to 0.
2. Set *CTB_CTRL.rst* = 1 to reset the CTB.
3. Poll until hardware sets *CTB_CTRL.rdy* = 1.
4. 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.
5. Set *CTB_CRC_CTRL.hrst* = 1 to reset the hamming code generator for the next calculation.
6. 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.
7. Set *CTB_CTRL.intr* = 1 to enable the interrupt when the DMA transfer is complete.
8. Load the starting address of the block to *CTB_DMA_SRC.addr*.
9. Write the length of the block in bytes to *CTB_DMA_CNT* 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.

19.6 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 19-4* lists frequently used polynomials and their check constants.

Table 19-4: 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.

19.6.1 CRC Operation

1. Enable the CTB peripheral clock by clearing *GCR_PCLKDIS0.ctb* to 0.
2. Reset the CRC engine by setting *CTB_CTRL.rst* to 1.
3. Poll until hardware sets *CTB_CTRL.rdy* to 1.
4. 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.*
5. Clear *CTB_CTRL.dmadnemsk* and *CTB_CTRL.flag_mode* to 0.
6. Clear *CTB_CTRL.rdsrc* to 0 to select the input FIFO as the input source.
7. Set *CTB_CTRL.intr* to 1 to enable the interrupt when the DMA transfer completes.
8. Write the desired polynomial to the *CTB_CRC_POLY* register.
9. Set *CTB_CRC_CTRL.crc* to 1 to enable the CRC function.
10. Load *CTB_DMA_SRC.addr* with the starting address of the memory to CRC.
11. Load *CTB_DMA_CNT* 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)

19.7 Secure Cryptographic Accelerator (SCA)

This accelerator is dedicated to non-symmetrical crypto operations. It is optimized for elliptic curve cryptography operations. This block does not support high-level RSA operations; however, the modular multiplication can be used by software to accelerate RSA computations.

The SCA supports Jacobian coordinates to improve the security level. Different Jacobian coordinates can represent the same point: (X; Y; Z) and ($X*\lambda^2$; $Y*\lambda^3$; $Z*\lambda$) are equivalent. Therefore, the SCA converts affine coordinates to a random Jacobian representation by using a secret random λ. This countermeasure aims to counter differential power analysis (DPA), correlation power analysis (CPA), the electromagnetic equivalents, and template/comparative attacks.

The accelerator can be used to both perform high-level operations, such as ECDSA signature and verification, and low-level operations such as modular addition, subtraction, multiplication, division, and scalar operations. It also supports up to 521-bit operations for low level operations.

Internal state machines are implemented to perform high-level operations without software interaction. For low-level operations, the software has direct access to the primary operations to implement a high-level algorithm such as ECC or RSA.

High-level and low-level operations are selected through dedicated opcodes. Operands and results reside in system RAM. Flash and APB registers are used to store data pointers to system memory; therefore, SCA does not require dedicated memory.

This accelerator has the following features:

- ECC basic operations.
- High-level and low-level operations.
- APB interface for configuration.
- AHB master interface to access system memory (operands, results, etc.).
- Pointer-based architecture to easily access SCA memory without memory transfer overhead.
- Interrupt management.
- Support for Jacobian and affine coordinates.
- SPA/DPA and fault attack countermeasures.
- Pre-calculated point support to speed ECC processing for ECDSA P256.
  - For manual parameter curves, pre-computed points must be provided.
- Custom curve parameters (Brainpool, others).
- EdDSA algorithms such as Ed25519 and Ed448 can also be implemented (supported in TLS 1.3), as well as X448 and X25519.
- Dedicated input for the secure private key.

19.8 Basic Operations

SCA uses system memory to read/store operands, working buffer, and results. Data pointers must be stored in the SCA configuration registers before starting any operations.

The primary purpose of the SCA is to perform ECDSA P256 signature and verification. Some P256 parameters are hardcoded in the design to simplify and speed up these two operations. However, SCA can be used in manual mode to perform other operations to cover a wide range of elliptic curves based operations, like ECDH, various curves (Brainpool…), different curve sizes (256, 384, and 521), and other ECDSA algorithms (basic modular operations and ECC point operation are provided).

CTB_SCA_CTRL0.manparam is used to select P256 operations or to use manual parameters for the other curves. If *CTB_SCA_CTRL0.manparam* is 0, hardcoded parameters are used to deal with NIST P256 operations. If *CTB_SCA_CTRL0.manparam* is 1, the curve parameters must be set in the operation buffer (*CTB_SCA_OP_BUFF_ADDR*).

19.8.1 Manual Parameters

CTB_SCA_CTRL0.eccsize is used to select the maximum size of the curve to optimize the computation when manual parameters are used. Only curve sizes of 256-, 384-, and 521-bits are supported.

Note: Large size can be used for small curve operation, but the required system memory and processing time are not optimized. For example, CTB_SCA_CTRL0.eccsize set to 521 bits can process 256-bit or 384-bit curves, and the unused bits (MSB) of the results must be ignored.

CTB_SCA_CTRL0.eccsize bits are only available when *CTB_SCA_CTRL0.manparam* is set. Otherwise, *CTB_SCA_CTRL0.eccsize* is set to 1 (256-bit) by the hardware to deal with the NIST P256 curve.

Before starting any SCA operation, it is required to set the data pointers in the SCA registers. The SCA engine uses these data pointers to read operands, store results, and locate a working buffer.

Data pointers must be stored in the following registers:

- *CTB_SCA_PPX_ADDR*
- *CTB_SCA_PPY_ADDR*
- *CTB_SCA_PPZ_ADDR*
- *CTB_SCA_PQX_ADDR*
- *CTB_SCA_PQY_ADDR*
- *CTB_SCA_PQZ_ADDR*
- *CTB_SCA_RDSA_ADDR*
- *CTB_SCA_RES_ADDR*
- *CTB_SCA_OP_BUFF_ADDR*

Each operation has specific requirements for working memory; therefore, the system memory allocation can be optimized for each operation. See *Memory Allocation for SCA Data in System Memory* to allocate system memory resources.

Once all the data pointers have been set and the operands are available in the corresponding locations in the system memory, the operation can be started by setting *CTB_SCA_CTRL0.stc* bit.

19.8.2 SCA Modular Operations

These operations can be used to add, subtract, multiply, and divide big numbers. The maximum size of the operand is defined by *CTB_SCA_CTRL0.eccsize* when *CTB_SCA_CTRL0.manparam* is 1; otherwise 256-bit operation is selected.

The modulo must be set by using the *CTB_SCA_CTRL0.modaddr* and *CTB_SCA_MODDATA.data* fields.

For example, a 10-bit modular operation with modulo 0x32F:

CTB_SCA_CTRL0.modaddr = 0.

CTB_SCA_MODDATA.data = 0x0000 032F.

Note: Unused bits (31 to 11) must be set to 0.

For a 256-bit modular operation with the following modulo:

MODULO = 0xFC632551 F3B9CAC2 A7179E84 BCE6FAAD FFFFFFFF FFFFFFFF 00000000 FFFFFFFF

The modulo must be loaded in the SCA internal registers:

1. *CTB_SCA_CTRL0.modaddr* = 0
2. *CTB_SCA_MODDATA.data* = 0xFFFF FFFF (LSB)
3. *CTB_SCA_CTRL0.modaddr* = 1
4. *CTB_SCA_MODDATA.data* = 0x0000 0000
5. *CTB_SCA_CTRL0.modaddr* = 2
6. *CTB_SCA_MODDATA.data* = 0xFFFF FFFF
7. *CTB_SCA_CTRL0.modaddr* = 3
8. *CTB_SCA_MODDATA.data* = 0xFFFF FFFF
9. *CTB_SCA_CTRL0.modaddr* = 4
10. *CTB_SCA_MODDATA.data* = 0xBCE6 FAAD
11. *CTB_SCA_CTRL0.modaddr* = 5
12. *CTB_SCA_MODDATA.data* = 0xA717 9E84
13. *CTB_SCA_CTRL0.modaddr* = 6
14. *CTB_SCA_MODDATA.data* = 0xF3B9 CAC2
15. *CTB_SCA_CTRL0.modaddr* = 7
16. *CTB_SCA_MODDATA.data* = 0xFC63 2551 (MSB)

For 521-bit modular operation, the MSB must be loaded at 0x16:

1. *CTB_SCA_CTRL0.modaddr* = 0
2. *CTB_SCA_MODDATA.data* = MOD0 (LSB)
3. *CTB_SCA_CTRL0.modaddr* = 1
4, *CTB_SCA_MODDATA.data* = MOD1
5. …
6. *CTB_SCA_CTRL0.modaddr* = 16
7. *CTB_SCA_MODDATA.data* = MOD16 (MSB)

Note: unused bits (522 to 543) must be set to 0.

Table 19-5: SCA Opcodes for Modular Operations

Operation	Opcode	Module	Comment	Parameters[1]
MM: Modular Multiplication	0	MODOP	Calculation of a product of two long integers modulo n Inputs: PPx, PPy, and MOD Output: RESx RESx = (PPx * PPy) modulo MOD With PPx < MOD and PPy < MOD	*CTB_SCA_PPX_ADDR* *CTB_SCA_PPY_ADDR* *CTB_SCA_RES_ADDR* *CTB_SCA_MODDATA*
MD: Modular Division	1	MODOP	Calculation of a ration of two long integers modulo n Inputs: PPx, PPy, and MOD bit Output: RESx RESx = (PPx / PPy) modulo MOD With PPx < MOD and PPy < MOD **Warning**: *MOD must be primary, with non-primary modulo. The result is not guaranteed since the inverse might not exist.*	*CTB_SCA_PPX_ADDR* *CTB_SCA_PPY_ADDR* *CTB_SCA_RES_ADDR* *CTB_SCA_MODDATA* *CTB_SCA_OP_BUFF_ADDR*[2]
MA: Modular Addition	2	MODOP	Calculation of a sum of two long integers modulo n Inputs: PPx, PPy, and MOD bit Output: RESx RESx = (PPx + PPy) modulo MOD With PPx < MOD and PPy < MOD	*CTB_SCA_PPX_ADDR* *CTB_SCA_PPY_ADDR* *CTB_SCA_RES_ADDR* *CTB_SCA_MODDATA*
MS: Modular Subtraction	3	MODOP	Calculation of a difference of two long integers modulo n Inputs: PPx, PPy, and MOD bit Output: RESx RESx = (PPx - PPy) modulo MOD With PPx < MOD and PPy < MOD	*CTB_SCA_PPX_ADDR* *CTB_SCA_PPY_ADDR* *CTB_SCA_RES_ADDR* *CTB_SCA_MODDATA*

19.8.3 SCA Opcodes for NIST P256

Table 19-6 describes the operations that can be used over the NIST P256 curve. The accelerator is optimized for P256. P256 curve parameters (including modulo and pre-computed points) are hardcoded. Therefore, the usage of P256 operation is straightforward.

For other curve types or sizes, the manual parameter mode (*CTB_SCA_CTRL0.manparam* =1) must be used to load curve parameters in OPBUFF before starting the operation.

Table 19-6: SCA Opcodes for NIST P256

Operation	Opcode	Module	Comment	Parameters[1]
ECJAADD: ECC Point Operation, Jacobian + Affine	5	POP	ECC point addition, one is in Jacobian coordinates, and the other is in Affine coordinates. The result is in Jacobian coordinates. Inputs: PP(X,Y,Z) = Jacobian point PQ(x,y) = Affine point MOD Output: RES(X,Y,Z) RES(X,Y,Z) = PP(X,Y,Z) + PQ(x,y) modulo MOD	PPX, PPY, PPZ PQX, PQY OPBUFF[2] RESX, RESY, RESZ

Operation	Opcode	Module	Comment	Parameters[1]
ECJASUB: ECC Point Operation, Jacobian - Affine	6	POP	ECC point subtraction, one is in Jacobi coordinates, and the other one is in Affine coordinates. The result is in Jacobi coordinates. Inputs: PP(X,Y,Z) = Jacobi Point PQ(x,y) = Affine Point MOD Output: RES(X,Y,Z) RES(X,Y,Z) = PP(X,Y,Z) - PQ(x,y) modulo MOD	PPX, PPY, PPZ PQX, PQY OPBUFF[2] RESX, RESY, RESZ
ECJDBL: ECC Point Operation, Jacobian Double	7	POP	ECC point double in Jacobi coordinates Input: PP(X,Y,Z) and MOD Output: RES(X,Y,Z) RES(X,Y,Z) = 2*PP(X,Y,Z) modulo MOD	PPX, PPY, PPZ OPBUFF[2] RESX, RESY, RESZ
ECJACTOAFF: ECC Point Operation, coordinates conversion	10	POP	EEC Point coordinates conversion, from Jacobi to Affine Inputs: PP(X,Y,Z) and MOD Output: RES(X,Y) RESx = PP(X)/PP(Z)2 modulo MOD RESy = PP(Y)/PP(Z)3 modulo MOD	PPX, PPY, PPZ OPBUFF[2] RESX, RESY
ECAFFTOJAC: ECC Point Operation, coordinates conversion	11	POP	EEC Point coordinates conversion, from Affine to Jacobi Inputs: PP(x,y) and MOD Output: RES(X,Y,Z) RESx = PP(x)*RND2 modulo MOD RESy = PP(y)*RND3 modulo MOD RESz = RND modulo MOD	PPX, PPY OPBUFF[2] RESX, RESY, RESZ
ECPVERIF: EC point verification	12	POP	Simply verify if a given point P indeed belongs to the curve P256 Inputs: PP(X,Y,Z) (1) and MOD Output: PVF1: 0 ok, 1 if not on the curve (1) For affine PP(Z) must be set to 1	PPX, PPY, PPZ OPBUFF[2] RESX, RESY, RESZ
ECSPSCAL: EC Secure Scalar for P256 Q=kP	13	SCALOP	EC secure scalar that uses pre-calculated points from SCA memory (PC0 … PC*n*) k must be stored in PPZ Inputs: PP(z) and MOD Output: AFFJAC=0 => RES(x,y) Affine coordinates RES(x,y) = PPz * PP(x,y) RES(z) by SCA as working variable	PPZ OPBUFF[2] RESX, RESY RESZ

Operation	Opcode	Module	Comment	Parameters[1]
ECSCALQ: EC Secure Scalar for Diffie Hellman operation (kQ) With authenticated Q	14	ECDSA	EC secure scalar that uses point Q from SCA memory (PQ(x,y)) k must be stored in PPx Q must be stored in PQ(x,y) Inputs: PP(x) ,PQ(x,y) and MOD Output: RES(x,y) Affine coordinates RES(x,y) = PPz * PP(x,y) RES(z) by SCA as working variable	PPX,PPY,PPZ PQX, PQY, PQZ OPBUFF[2] RESX, RESY RESZ
ECDSASIG: ECDSA signature	15	ECDSA	ECDSA signature k (nonce) must be stored in PPz d (private key) must be stored in PQx H(m) must be stored in PQz Inputs: PP(z) ,PQ(x,z) and MOD Output: RESy=r = kP(x) RESx= $s = k^{-1}(H(m) + d.r)$ RES(z) is used by the SCA as a working variable	PPX, PPY PPZ PQX, PQY PQZ OPBUFF[2] RESX, RESY RESZ
ECDSAVER: ECDSA verification	16	ECDSA	ECDSA verification r must be stored in RDSA s must be stored in PPx H(m) must be stored in PQz Q must be stored in PQ(x,y) Inputs: RDSA, PP(x) ,PQ(x,y,z) and MOD Output: If pass, RESx =0, PVF=0 else RESx ≠0, PVF =1. RES(x,y,z) are used by the SCA as working variables	RDSA PPX, PPY, PPZ PQX, PQY, PQZ OPBUFF[2] RESX, RESY, RESZ

Operation	Opcode	Module	Comment	Parameters[1]
ECSPC2kPA2vQ: EC unsecure 2kP + 2vQ for P256	18	SCALOP	Used for ECDSA verification P+Q must be stored in affine in PP P-Q must be stored in affine in PQ k MSBs must be stored in PQz MSBs k LSBs must be stored in PPz MSBs v MSBs must be stored in PQz LSBs v LSBs must be stored in PPz LSBs Inputs: PP(x,y,z),PQ(x,y,z) and MOD Output: RES(x,y) Affine coordinates RES(x,y) = (PQz MSBs||PPz MSBs)*PP(x,y) + (PQz LSBs||PPz LSBs)*PQ(x,y) RES(z) by SCA as working variable	PPX, PPY, PPZ PQX, PQY, PQZ OPBUFF[2] RESX, RESY RESZ
ECPSQ: EC P-Q, with hardcoded P(1) and result in affine Hardcoded if *CTB_SCA_CTRL0.manparam* is not set; otherwise, P is from OPBUFF (see *Manual Parameters*)	19	SCALOP	Q must be stored in PQ(x,y) Inputs: PQ(x,y) and MOD Output: RES(x,y) Affine coordinates RES(x,y) = P-PQ RES(z) by SCA as working variable	PQX, PQY OPBUFF[2] RESX, RESY RESZ
ECPAQ: EC P+Q, with hardcoded P and result in affine Hardcoded if *CTB_SCA_CTRL0.manparam* is not set; otherwise, P is from OPBUFF (see *Manual Parameters*)	20	SCALOP	Q must be stored in PQ(x,y) Inputs: PQ(x,y) and MOD Output: RES(x,y) Affine coordinates RES(x,y) = P + PQ RES(z) by SCA as working variable	PQX, PQY OPBUFF[2] RESX, RESY RESZ

[1] *Corresponding registers must be filled with data pointers. See Memory Allocation for SCA Data in System Memory for details.*

[2] *Operation buffer is required to store/load intermediate results*

19.9 Manual Parameters

In manual parameter mode, the operation buffer (OPBUFF) must be filled with ECC parameters. The base address of the operation buffer is defined by the operational buffer address register (*CTB_SCA_OP_BUFF_ADDR*). The size of the operating buffer depends on the operand size, *CTB_SCA_CTRL0.eccsize*.

Note: The speed of the operations increases with the operand size.

RA, RB, RC, and RD are internal variables used by the SCA to store/load intermediate results and must be aligned to 1KB, i.e., they must be in the same 1KB space.

The following tables describe the OPBUFF mapping to store ECC curve parameters.

19.9.1 256-Bit Operations (CTB_SCA_CTRL0.eccsize = 0 or 1)

A total of 768 bytes must be reserved to store the operation buffer.

Table 19-7: Operation Buffer Mapping with CTB_SCA_CTRL0.eccsize = 0/1

Offset	Parameter	Size (Bits)	Description
[0x000]	Mod P	256	P modulo
[0x020]	Mod N	256	N modulo
[0x040]	CA	256	ECC – CA parameter
[0x060]	CB	256	ECC – CB parameter
[0x080]	PC0X	256	Pre-computed point 0, x coordinate
[0x0A0]	PC0Y	256	Pre-computed point 0, y coordinate
[0x0C0]	PC1X	256	Pre-computed point 1, x coordinate
[0x0E0]	PC1Y	256	Pre-computed point 1, y coordinate
[0x100]	PC2X	256	Pre-computed point 2, x coordinate
[0x120]	PC2Y	256	Pre-computed point 2, y coordinate
[0x140]	PC3X	256	Pre-computed point 3, x coordinate
[0x160]	PC3Y	256	Pre-computed point 3, y coordinate
[0x180]	PC4X	256	Pre-computed point 4, x coordinate
[0x1A0]	PC4Y	256	Pre-computed point 4, y coordinate
[0x1C0]	PC5X	256	Pre-computed point 5, x coordinate
[0x1E0]	PC5Y	256	Pre-computed point 5, y coordinate
[0x200]	PC6X	256	Pre-computed point 6, x coordinate
[0x220]	PC6Y	256	Pre-computed point 6, y coordinate
[0x240]	PC7X	256	Pre-computed point 7, x coordinate
[0x260]	PC7Y	256	Pre-computed point 7, y coordinate
[0x280]	RA	256	Reserved
[0x2A0]	RB	256	Reserved
[0x2C0]	RC	256	Reserved
[0x2E0]	RD	256	Reserved

19.9.2 384-Bit Operations (CTB_SCA_CTRL0.eccsize = 2)

A total of 1,152 bytes must be reserved to store the operation buffer.

Table 19-8: Operation Buffer Mapping with CTB_SCA_CTRL0.eccsize = 2

Offset	Parameter	Size (Bits)	Description
[0x000]	Mod P	384	P modulo
[0x030]	Mod N	384	N modulo
[0x060]	CA	384	ECC – CA parameter
[0x090]	CB	384	ECC – CB parameter
[0x0C0]	PC0X	384	Pre-computed point 0, x coordinate
[0x0F0]	PC0Y	384	Pre-computed point 0, y coordinate
[0x120]	PC1X	384	Pre-computed point 1, x coordinate

Offset	Parameter	Size (Bits)	Description
[0x150]	PC1Y	384	Pre-computed point 1, y coordinate
[0x180]	PC2X	384	Pre-computed point 2, x coordinate
[0x1B0]	PC2Y	384	Pre-computed point 2, y coordinate
[0x1E0]	PC3X	384	Pre-computed point 3, x coordinate
[0x210]	PC3Y	384	Pre-computed point 3, y coordinate
[0x240]	PC4X	384	Pre-computed point 4, x coordinate
[0x270]	PC4Y	384	Pre-computed point 4, y coordinate
[0x2A0]	PC5X	384	Pre-computed point 5, x coordinate
[0x2D0]	PC5Y	384	Pre-computed point 5, y coordinate
[0x300]	PC6X	384	Pre-computed point 6, x coordinate
[0x330]	PC6Y	384	Pre-computed point 6, y coordinate
[0x360]	PC7X	384	Pre-computed point 7, x coordinate
[0x390]	PC7Y	384	Pre-computed point 7, y coordinate
[0x3C0]	RA	384	Reserved
[0x3F0]	RB	384	Reserved
[0x420]	RC	384	Reserved
[0x450]	RD	384	Reserved

19.9.3 521-Bit Operations (CTB_SCA_CTRL0.eccsize = 3)

A total of 1,632 bytes must be reserved to store the operation buffer.

Table 19-9: Operation Buffer Mapping with CTB_SCA_CTRL0.eccsize = 3

Offset	Parameter	Size (Bits)	Description
[0x000]	Mod P	544	P modulo
[0x044]	Mod N	544	N modulo
[0x088]	CA	544	ECC – CA parameter
[0x0CC]	CB	544	ECC – CB parameter
[0x110]	PC0X	544	Pre-computed point 0, x coordinate
[0x154]	PC0Y	544	Pre-computed point 0, y coordinate
[0x198]	PC1X	544	Pre-computed point 1, x coordinate
[0x1DC]	PC1Y	544	Pre-computed point 1, y coordinate
[0x220]	PC2X	544	Pre-computed point 2, x coordinate
[0x264]	PC2Y	544	Pre-computed point 2, y coordinate
[0x2A8]	PC3X	544	Pre-computed point 3, x coordinate
[0x2EC]	PC3Y	544	Pre-computed point 3, y coordinate
[0x330]	PC4X	544	Pre-computed point 4, x coordinate
[0x374]	PC4Y	544	Pre-computed point 4, y coordinate
[0x3B8]	PC5X	544	Pre-computed point 5, x coordinate
[0x3FC]	PC5Y	544	Pre-computed point 5, y coordinate
[0x440]	PC6X	544	Pre-computed point 6, x coordinate
[0x484]	PC6Y	544	Pre-computed point 6, y coordinate

Offset	Parameter	Size (Bits)	Description
[0x4C8]	PC7X	544	Pre-computed point 7, x coordinate
[0x50C]	PC7Y	544	Pre-computed point 7, y coordinate
[0x550]	RA	544	Reserved
[0x594]	RB	544	Reserved
[0x5D8]	RC	544	Reserved
[0x61C]	RD	544	Reserved

19.10 Memory Allocation for SCA Data in System Memory

System memory must be reserved for operands, results, and various SCA variables. The following registers must be initialized by software before any SCA operation:

- *CTB_SCA_PPX_ADDR*, *CTB_SCA_PPY_ADDR*, *CTB_SCA_PPZ_ADDR*,
- *CTB_SCA_PQX_ADDR*, *CTB_SCA_PQY_ADDR*, *CTB_SCA_PQZ_ADDR*
- *CTB_SCA_RDSA_ADDR*
- *CTB_SCA_RES_ADDR*, *CTB_SCA_OP_BUFF_ADDR*

Note: All memory allocated to the SCA block must be aligned to 1KB.

Some SCA operations need extra system memory to store/load variables. This extra memory is defined as the operation buffer. The size of the OPBUFF depends on the operand size (*CTB_SCA_CTRL0*.*eccsize*) and the selected mode (*CTB_SCA_CTRL0*.*manparam*).

If *CTB_SCA_CTRL0*.*manparam* is 0, the SCA uses hardcoded parameters for NIST P256, and a restricted OPBUFF is needed. Otherwise (*CTB_SCA_CTRL0*.*manparam* =1), a larger OPBUFF is needed to store curve parameters.

CTB_SCA_CTRL0.*manparam* = 0 (NIST P256)

- OPBUFF is 4 blocks × ECC Size × 6:
- ECC size = 256: OPBUFF = 4 × 256 × 6 ⇒ 6,144 bits ⇒ 128 bytes
- ECC size = 384: OPBUFF = 4 × 384 × 6 ⇒ 9,216 bits ⇒ 192 bytes
- ECC size=521: OPBUFF = 4 × 544 × 6 ⇒ 13,056 bits ⇒ 272 bytes

CTB_SCA_CTRL0.*manparam* = 1

- OPBUFF is 24 blocks * ECC Size:
- ECC size=256: OPBUFF ⇒ 24 × 256 × 6 ⇒ 6,144 bits ⇒ 768 bytes
- ECC size=384: OPBUFF ⇒ 24 × 384 × 6 ⇒ 9,216 bits ⇒ 1152 bytes
- ECC size=521: OPBUFF ⇒ 24 × 544 × 6 ⇒ 13,056 bits ⇒ 1632 bytes

19.10.1 SCA Opcodes

Table 19-10: SCA Opcode Summary

OPCODE	Mnemonic	Description
0x00	MM	Modular Multiplication
0x01	MD	Modular Division
0x02	MA	Modular addition
0x03	MS	Modular Subtraction
0x05	ECJAADD	ECC Point Operation, Jacobian + Affine

OPCODE	Mnemonic	Description
0x06	ECJASUB	ECC Point Operation, Jacobian – Affine
0x07	ECJDBL	ECC Point Operation, Jacobian Double
0x0A	ECJACTOAFF	ECC Point Operation, Coordinates Conversion
0x0B	ECAFFTOJAC	ECC Point Operation, Coordinates Conversion
0x0C	ECPVERIF	EC Point Verification
0x0D	ECSPSCAL	EC Secure Scalar for P256 (kP)
0x0E	ECSCALQ	EC Secure Scalar for Diffie Hellman Operation (kQ)
0x0F	ECDSASIG	ECDSA Signature
0x10	ECDSAVER	ECDSA Verification
0x12	ECSPC2kPA2vQ	EC Unsecure 2kP + 2vQ for P256
0x13	ECP - Q	With Hard-Coded P and Result in Affine
0x14	ECP + Q	With Hard-Coded P and Result in Affine

19.10.2 SCA Opcode Details

The following tables define which memory location shall be reserved for each SCA operation. N/A means that the corresponding memory is not used during the operation. Therefore the specific address register (refer to *SCA Data Pointer Address Registers*) is not used by the SCA.

19.10.3 OPCODE = 0x00: MM - Modular Multiplication

Table 19-11: Opcode 0x00: MM

	Required Allocation of System RAM (Bytes)									
ECC Size	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF	Total
256	32	32	N/A	N/A	N/A	N/A	32	N/A	N/A	96
384	48	48	N/A	N/A	N/A	N/A	48	N/A	N/A	144
521	65	65	N/A	N/A	N/A	N/A	65	N/A	N/A	195

19.10.4 OPCODE = 0x01: MD - Modular Division

Table 19-12: Opcode 0x01: MD

	Required Allocation of System RAM (Bytes)									
ECC Size	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF	Total
256	32	32	N/A	N/A	N/A	N/A	32	N/A	128	224
384	48	48	N/A	N/A	N/A	N/A	48	N/A	192	336
521	65	65	N/A	N/A	N/A	N/A	65	N/A	272	467

19.10.5 OPCODE = 0x02: MA - Modular addition

Table 19-13: Opcode 0x02: MA

	Required Allocation of System RAM (Bytes)									
ECC Size	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF	Total
256	32	32	N/A	N/A	N/A	N/A	32	N/A	N/A	96
384	48	48	N/A	N/A	N/A	N/A	48	N/A	N/A	144
521	65	65	N/A	N/A	N/A	N/A	65	N/A	N/A	195

19.10.6 OPCODE = 0x03: MS - Modular Subtraction

Table 19-14: Opcode 0x03: MS

	Required Allocation of System RAM (Bytes)									
ECC Size	**PPX**	**PPY**	**PPZ**	**PQX**	**PQY**	**PQZ**	**RES**	**RDSA**	**OPBUFF**	**Total**
256	32	32	N/A	N/A	N/A	N/A	32	N/A	N/A	96
384	48	48	N/A	N/A	N/A	N/A	48	N/A	N/A	144
521	65	65	N/A	N/A	N/A	N/A	65	N/A	N/A	195

19.10.7 OPCODE = 0x05: ECJAADD - ECC Point Operation, Jacobian + Affine

Table 19-15: Opcode 0x05: ECJAADD

	Required Allocation of System RAM (Bytes)											
ECC Size	**PPX**	**PPY**	**PPZ**	**PQX**	**PQY**	**PQZ**	**RES**	**RDSA**	**OPBUFF**		**Total**	
									CTB_SCA_CTRL0 manparam		***CTB_SCA_CTRL0 manparam***	
									0	**1**	**0**	**1**
256	32	32	32	32	32	N/A	96	N/A	128	768	384	1024
384	48	48	48	48	48	N/A	144	N/A	192	1152	576	1536
521	65	65	65	65	65	N/A	195	N/A	272	1632	792	2152

19.10.8 OPCODE = 0x06: ECJASUB - ECC Point Operation, Jacobian – Affine

Table 19-16: Opcode 0x06: ECJASUB

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		*CTB_SCA_CTRL0 manparam*	
									0	1	0	1
256	32	32	32	32	32	N/A	96	N/A	128	768	384	1024
384	48	48	48	48	48	N/A	144	N/A	192	1152	576	1536
521	65	65	65	65	65	N/A	195	N/A	272	1632	792	2152

19.10.9 OPCODE = 0x07: ECJDBL - ECC Point Operation, Jacobian Double

Table 19-17: Opcode 0x07: ECJDBL

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		*CTB_SCA_CTRL0 manparam*	
									0	1	0	1
256	32	32	32	N/A	N/A	N/A	96	N/A	128	768	320	960
384	48	48	48	N/A	N/A	N/A	144	N/A	192	1152	480	1440
521	65	65	65	N/A	N/A	N/A	195	N/A	272	1632	662	2022

19.10.10OPCODE = 0x0A: ECJACTOAFF - ECC Point Operation, Coordinates Conversion

Table 19-18: Opcode 0x0A: ECJACTOAFF

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		*CTB_SCA_CTRL0 manparam*	
									0	1	0	1
256	32	32	32	N/A	N/A	N/A	96	N/A	128	768	320	960
384	48	48	48	N/A	N/A	N/A	144	N/A	192	1152	480	1440
521	65	65	65	N/A	N/A	N/A	195	N/A	272	1632	662	2022

19.10.11OPCODE = 0x0B : ECAFFTOJAC - ECC Point Operation, Coordinates Conversion

Table 19-19: Opcode 0x0B : ECAFFTOJAC

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		*CTB_SCA_CTRL0 manparam*	
									0	1	0	1
256	32	32	N/A	N/A	N/A	N/A	96	N/A	128	768	288	928
384	48	48	N/A	N/A	N/A	N/A	144	N/A	192	1152	432	1392

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		*CTB_SCA_CTRL0 manparam*	
									0	1	0	1
521	65	65	N/A	N/A	N/A	N/A	195	N/A	272	1632	597	1957

19.10.12OPCODE = 0x0C: ECPVERIF - EC Point Verification

Table 19-20: Opcode 0x0C: ECPVERIF

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		*CTB_SCA_CTRL0 manparam*	
									0	1	0	1
256	32	32	32	N/A	N/A	N/A	96	N/A	128	768	320	960
384	48	48	48	N/A	N/A	N/A	144	N/A	192	1152	480	1440
521	65	65	65	N/A	N/A	N/A	195	N/A	272	1632	662	2022

19.10.13OPCODE = 0x0D: ECSPSCAL - EC Secure Scalar for P256 (kP)

Table 19-21: Opcode 0x0D: ECSPSCAL

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		*CTB_SCA_CTRL0 manparam*	
									0	1	0	1
256	N/A	N/A	32	N/A	N/A	N/A	96	N/A	128	768	256	896
384	N/A	N/A	48	N/A	N/A	N/A	144	N/A	192	1152	384	1344
521	N/A	N/A	65	N/A	N/A	N/A	195	N/A	272	1632	532	1892

19.10.14OPCODE = 0x0E: ECSCALQ - EC Secure Scalar for Diffie Hellman Operation (kQ)

Table 19-22: Opcode 0x0E: ECSCALQ

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		*CTB_SCA_CTRL0 manparam*	
									0	1	0	1
256	32	32	32	32	32	32	96	N/A	128	768	416	1056
384	48	48	48	48	48	48	144	N/A	192	1152	624	1584
521	65	65	65	65	65	65	195	N/A	272	1632	857	2217

19.10.15OPCODE = 0x0F: ECDSASIG - ECDSA Signature

Table 19-23: Opcode 0x0F: ECDSASIG

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		CTB_SCA_CTRL0 manparam	
									0	1	0	1
256	32	32	32	32	32	32	96	N/A	128	768	416	1056
384	48	48	48	48	48	48	144	N/A	192	1152	624	1584
521	65	65	65	65	65	65	195	N/A	272	1632	857	2217

19.10.16OPCODE = 0x10: ECDSAVER - ECDSA Verification

Table 19-24: Opcode 0x10: ECDSAVER

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		CTB_SCA_CTRL0 manparam	
									0	1	0	1
256	32	32	32	32	32	32	96	32	128	768	448	1088
384	48	48	48	48	48	48	144	48	192	1152	672	1632
521	65	65	65	65	65	65	195	65	272	1632	922	2282

19.10.17OPCODE = 0x12: ECSPC2kPA2vQ - EC Unsecure 2kP + 2vQ for P256

Table 19-25: Opcode 0x12: ECSPC2kPA2vQ

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		CTB_SCA_CTRL0 manparam	
									0	1	0	1
256	32	32	32	32	32	32	96	N/A	128	768	416	1056
384	48	48	48	48	48	48	144	N/A	192	1152	624	1584
521	65	65	65	65	65	65	195	N/A	272	1632	857	2217

19.10.18OPCODE = 0x13: EC P-Q - With Hard-Coded P and Result in Affine

Table 19-26: Opcode 0x13: EC P-Q

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		CTB_SCA_CTRL0 manparam	
									0	1	0	1
256	N/A	N/A	N/A	32	32	N/A	96	N/A	128	768	288	928
384	N/A	N/A	N/A	48	48	N/A	144	N/A	192	1152	432	1392

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		*CTB_SCA_CTRL0 manparam*	
									0	1	0	1
521	N/A	N/A	N/A	65	65	N/A	195	N/A	272	1632	597	1957

19.10.19OPCODE = 0x14: EC P+Q - With Hard-Coded P and Result in Affine

Table 19-27: Opcode 0x14: EC P+Q

ECC Size	Required Allocation of System RAM (Bytes)											
	PPX	PPY	PPZ	PQX	PQY	PQZ	RES	RDSA	OPBUFF		Total	
									CTB_SCA_CTRL0 manparam		*CTB_SCA_CTRL0 manparam*	
									0	1	0	1
256	N/A	N/A	N/A	32	32	N/A	96	N/A	128	768	288	928
384	N/A	N/A	N/A	48	48	N/A	144	N/A	192	1152	432	1392
521	N/A	N/A	N/A	65	65	N/A	195	N/A	272	1632	597	1957

19.11 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 19-28: 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
[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

Offset	Register	Name
[0x0050]	*CTB_CIPHER_INIT[0]*	Cipher Initial Vector Register 0 [31:0]
[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_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[3]*	Tag/Mic Register 3

Offset	Register	Name
[0x0100]	*CTB_SCA_CTRL0*	SCA Control Register
[0x0104]	*CTB_SCA_CTRL1*	SCA Advanced Control Register
[0x0108]	*CTB_SCA_STAT*	SCA Status Register
[0x010C]	*CTB_SCA_PPX_ADDR*	PPX Address Register
[0x0110]	*CTB_SCA_PPY_ADDR*	PPY Address Register
[0x0114]	*CTB_SCA_PPZ_ADDR*	PPZ Address Register
[0x0118]	*CTB_SCA_PQX_ADDR*	PQX Address Register
[0x011C]	*CTB_SCA_PQY_ADDR*	PQY Address Register
[0x0120]	*CTB_SCA_PQZ_ADDR*	PQZ Address Register
[0x0124]	*CTB_SCA_RDSA_ADDR*	SCA RDSA Address Register
[0x0128]	*CTB_SCA_RES_ADDR*	SCA Result Address Register
[0x012C]	*CTB_SCA_OP_BUFF_ADDR*	SCA Operation Buffer Address
[0x0130]	*CTB_SCA_MODDATA*	SCA Modulo Data Input Register
[0x0134]	*CTB_SCA_NRNG*	SCA NRNG Register

19.11.1 Register Details

Table 19-29: 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: No cryptographic operations have been completed since this bit was cleared. 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: Cryptographic accelerator ready for use.
29	err	R	0	**AHB Bus Error** This bit is set if the DMA attempts to access non-existent or protected memory on the AHB bus. This bit can only be cleared by resetting the cryptographic accelerator block. 0: No error. 1: Cryptographic accelerator DMA bus error.
28	-	RO	0	**Reserved**

Cryptographic Control			CTB_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
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**
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** DNM

Cryptographic Control			CTB_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
6	wait_en	DNM	0	**Wait Pin Enable** DNM
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.
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 19-30: 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.

Cipher Control			CTB_CIPHER_CTRL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
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**
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: DES. 0b101: TDEA. 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—each field setting loads 128 bits. The load cipher key operation must be performed twice (once with 0b10 and again with 0b11) to load a full 256-bit key. 0b00: *CTB_CIPHER_KEY[7]*:*CTB_CIPHER_KEY[0]* registers. 0b01: Reserved. 0b10: *USR_AESKEYS_KEY7*:*USR_AESKEYS_KEY0* registers. 0b11: Reserved. *Note: If CTB_CIPHER_CTRL.cipher is set to DES or TDEA, the source of the cipher key is always the CTB_CIPHER_KEY registers.*

Cipher Control			CTB_CIPHER_CTRL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
1	key	R/W1O	0	**Load Cipher Key Using CDMA** 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 CDMA completes loading the key. 0: NOP. 1: Initiate key loading from DMA.
0	enc	R/W	0	**Block Cipher Encryption Mode** Set this field to 1 to select decryption and 0 for encryption. 0: Encrypt. 1: Decrypt.

Table 19-31: 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.
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** Set 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 19-32: 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 19-33: 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 19-34: 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 19-35: Cryptographic DMA Count Register

Cryptographic DMA Count			CTB_DMA_CNT	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0.	-	R/W	0	**DMA Byte Counter** Writing a non-zero value to this register initiates the DMA-based operations.

Table 19-36: Cryptographic Data In Registers

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 19-37: Cryptographic Data Out Registers

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 19-38: CRC Polynomial Register

CRC Polynomial	CTB_CRC_POLY	[0x0040]

Bits	Name	Access	Reset	Description
31:0	-	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 19-39: 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 19-40: Hamming Engine Result Register

Hamming Engine Result	CTB_HAM_ECC	[0x004C]

Bits	Name	Access	Reset	Description
31:17	-	RO	0	**Reserved**

Hamming Engine Result			CTB_HAM_ECC	[0x004C]
Bits	**Name**	**Access**	**Reset**	**Description**
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 19-41: Cipher Initial Vector Registers

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 19-42: Cipher Key Registers

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. Refer to the *CTB_CIPHER_CTRL*.*key* field for information on loading this register. This register is affected by the endian swap input control bits.

Table 19-43: Hash Message Digest Registers

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 19-44: Hash Message Size Registers

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 19-45: 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 19-46: 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 19-47: 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 19-48: 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 19-49: TAG/MIC Registers

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.

Table 19-50: SCA Control 0 Register

SCA Control			CTB_SCA_CTRL0	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
25:24	eccsize	R/W	0	**ECC Size** This field sets the maximum number of bits for ECC operations. This field is ignored if *CTB_SCA_CTRL0.manparam* = 0. 0b00: 256 bits. 0b01: 256 bits. 0b10: 384 bits. 0b11: 521 bits.
23:21	-	RO	0	**Reserved**
20:16	modaddr	R/W	0	**MODULO Address Offset** The *CTB_SCA_MODDATA.data* and *CTB_SCA_CTRL0.modaddr* fields are used to load the SCA modulo for modular operations.
15:13	-	RO	0	**Reserved**
12:8	opcode	R/W	0	**SCA Opcode** See *Table 19-10* for more information.
7	-	RO	0	**Reserved**

SCA Control			CTB_SCA_CTRL0	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
6	hwkey	R/W		**Hardware Key Select** This field selects the source of the private key for ECDSA signature and ECC scalar operations. 256-bits only available if *CTB_SCA_CTRL1.man* = 0. Do not modify this field if a hardware key is not implemented on the device. 0: Software key is used (PQx). 1: Hardware key is used.
5	manparam	R/W	0	**ECC Parameter Source** 0: ECC operation automatically performed using hardcoded NIST P256 parameters. 1: Software must load ECC parameters into OPBUFF before performing ECC operations.
4	ermem	R/W	0	**Erase Cryptographic Memory** Setting this field to 1 initiates a hardware erasure of the cryptographic memory defined by *SCA Data Pointer Address Registers* and *CTB_SCA_CTRL0.eccsize*. SCA operations in progress are automatically aborted. This bit is automatically cleared to 0 when the operation is complete. 0: Operation complete. 1: Memory erasure in progress.
3	-	RO	0	**Reserved**
2	abort	R/W		**Abort Operation.** Setting this field to 1 aborts operation in progress. The field is automatically cleared to 0 by hardware when the abort operation is complete. 0: Operation complete. 1: Abort operation in progress.
1	scaie	R/W		**SCA Interrupt Enable** 0: Disabled. 1: Enabled.
0	stc	R/W		**Start Calculation** This field functions as both the control and the status of the SCA. Setting this bit to 1 starts the calculation defined by the *opcode* field. This field bit is cleared to 0 by hardware when the calculation is complete. Clearing this field to 0 aborts the operation in progress and resets the SCA to its default state. This bit is cleared to 0 by hardware if an error is detected (*CTB_SCA_STAT.fsmerr*, *CTB_SCA_STAT.pvf2*, or *CTB_SCA_STAT.comperr*). 0: Operation complete. 1: Calculation in progress.

Table 19-51: SCA Control 1 Register

SCA Control 1			CTB_SCA_CTRL1	[0x0104]
Bits	**Name**	**Access**	**Reset**	**Description**
31:18	-	RO	0	**Reserved**
17:8	carrypos	R/W		**To set Carry Location** This field can be used if operands are less than or equal to 521 bits. According to the operand size, the *carrypos* range is from 1 to 522. *CTB_SCA_STAT.carry* is set when a carry is detected upon operation completion, based on the carry location defined by these bits. This field is ignored if *CTB_SCA_CTRL1.man* is 0. *Note: If operands are less than 256 bits, e.g., m bits with m < 256, the results are on > m bits (e.g., m+1 for add, up to 2m for multiplication), with result ≤ 256 bits.*

SCA Control 1			CTB_SCA_CTRL1	[0x0104]
Bits	**Name**	**Access**	**Reset**	**Description**
7:6	-	RO	0	**Reserved**
5	nrng	R/W	0	**NRNG** Set this field to 1 to use the nrng pointed to by the *CTB_SCA_NRNG* register. This field is only valid for OPCODE = 0x0F.
4:3	-	RO	0	**Reserved**
2	plusone	R/W		**Enable Carry propagation for the next operation.** Automatically cleared to 0 if *CTB_SCA_CTRL1.man* is 0. 0: NOP. 1: Propagate carry.
1	autocarry	R/W		**Automatically propagate the carry for the next operation.** When this bit is set, *CTB_SCA_CTRL1.plusone* is atomically set on carry detection. If this bit is not set, the software must write *CTB_SCA_CTRL1.plusone* if needed. No effect if *CTB_SCA_CTRL1.man* is 0. 0: Software has to write to *CTB_SCA_CTRL1.plusone.* 1: Autocarry.
0	man	R/W		**SCA Mode** 0: Auto mode. 1: Manual mode.

Table 19-52: SCA Status Register

SCA Status			CTB_SCA_STAT	[0x0108]
Bits	**Name**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11	aluneg2	R	0	**ALU 2 Sub Sign of the subtraction result for ALU 2** 0: Result ≥ 0 1: Result < 0
10	aluneg1	R	0	**ALU 1 Sub Sign of the subtraction result for ALU 1** 0: Result ≥ 0 1: Result < 0
9	gte2i2	R	0	**Modulo 2x Result** This bit is set if the operation result is $\geq 2 \times modulo$. Automatically cleared to 0 if *CTB_SCA_CTRL1.man* is 0. 0: Result $< 2 \times modulo$ 1: Result $\geq 2 \times modulo$
8	carry	R	0	**Carry on Ongoing Operation** 0: No carry 1: Carry
7	-	RO	0	**Reserved**
6	memerr	R	0	**SCA Memory Access Error** This error flag is set when the SCA memory access is not authorized, e.g., *CTB_SCA_PPX_ADDR.addr* = 0x0000 0000, 0x0000 0000 is ROM memory; thus, this location cannot be written, any SCA write to this location fails, and this field is set to 1. An interrupt is generated if enabled when this field is set to 1. Write 1 to clear. 0: Normal operation 1: Error

SCA Status			CTB_SCA_STAT	[0x0108]
Bits	**Name**	**Access**	**Reset**	**Description**
5	comperr	R/W1C	0	**EC Computation Error** Ongoing operation has failed. The severity depends on the processed operation. An interrupt is generated if enabled when this field is set to 1. Write 1 to clear. 0: Normal operation 1: Error
4	fsmerr	R/W1C	0	**FSM Transition Error** This error can occur upon external fault injection. An interrupt is generated if enabled when this field is set to 1. Write 1 to clear. 0: Normal operation 1: Error
3	pvf2	R/W1C	0	**Point 2 Verification Fail** This bit is set if the result of the EC point verification fails (see ECC operation table). An interrupt is generated if enabled when this field is set to 1. Write 1 to clear. 0: Normal operation 1: Error
2	pvf1	R/W1C	0	**Point 1 Verification Fail** This bit is set if the result of the EC point verification fails. An interrupt is generated if enabled when this field is set to 1. Write 1 to clear. 0: Normal operation 1: Error
1	scaif	R/W1C	0	**SCA Interrupt Flag** This bit is set by hardware when the SCA has completed an operation. Write 1 to clear. 0: Normal operation 1: Operation complete.
0	busy	R/W	0	**SCA Busy** 0: Not busy 1: Busy

19.11.1.1 SCA Data Pointer Address Registers

Table 19-53: Point P Data Pointer Registers

PPX Coordinate Data Pointer			CTB_SCA_PPX_ADDR	[0x010C]
PPY Coordinate Data Pointer			CTB_SCA_PPY_ADDR	[0x0110]
PPZ Coordinate Data Pointer			CTB_SCA_PPZ_ADDR	[0x0114]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**Point P Coordinate Data Pointer** Set this address to the start of the specific parameter in RAM.

Table 19-54: Point Q Data Pointer Registers

PQX Coordinate Data Pointer			CTB_SCA_PQX_ADDR	[0x0118]
PQY Coordinate Data Pointer			CTB_SCA_PQY_ADDR	[0x011C]
PQZ Coordinate Data Pointer			CTB_SCA_PQZ_ADDR	[0x0120]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**Point Q Coordinate Data Pointer** Set this address to the start of the specific parameter in RAM.

Table 19-55: SCA RDSA Address Register

SCA RDSA Address			CTB_SCA_RDSA_ADDR	[0x0124]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**SCA RDSA Result Address Register** This field defines the starting address of the R portion for an R, S ECDSA signature. This parameter is only used for ECDSA verification (OPCODE = 0x10).

Table 19-56: SCA Result Address Register

SCA Result Address			CTB_SCA_RES_ADDR	[0x0128]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**SCA Result Address Register** Set this address to the starting address of result storage.

Table 19-57: SCA Operation Buffer Address Register

SCA Operation Buffer Address			CTB_SCA_OP_BUFF_ADDR	[0x012C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**SCA Operation Buffer Address Register** Set this address to the starting address of the operation buffer.

19.11.1.2 SCA Modulo Data Input Register

Table 19-58: SCA Modulo Data Input Register

SCA Modulo Data Input			CTB_SCA_MODDATA	[0x0130]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**SCA Modulo Data Input Register** The *CTB_SCA_MODDATA.data* and *CTB_SCA_CTRL0.modaddr* fields are used to load the SCA modulo for modular operations.

Table 19-59: SCA NRNG Register

SCA NRNG			CTB_SCA_NRNG	[0x0134]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**NRNG Starting Address** Set this field to the starting address for NRNG stored in RAM. The random number is used when signing with ECDSA, OPCODE=0x0F. This address is only used if *CTB_SCA_CTRL1.nrng* = 1.

19.12 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 19-60: User AES Key Register Summary

Offset	Name	Description
[0x0000]	*USR_AESKEYS_KEY0*	*User AES Key 0 Register*
[0x0004]	*USR_AESKEYS_KEY1*	*User AES Key 1 Register*
[0x0008]	*USR_AESKEYS_KEY2*	*User AES Key 2 Register*
[0x000C]	*USR_AESKEYS_KEY3*	*User AES Key 3 Register*
[0x0010]	*USR_AESKEYS_KEY4*	*User AES Key 4 Register*
[0x0014]	*USR_AESKEYS_KEY5*	*User AES Key 5 Register*
[0x0018]	*USR_AESKEYS_KEY6*	*User AES Key 6 Register*
[0x001C]	*USR_AESKEYS_KEY7*	*User AES Key 7 Register*

19.12.1 Register Details

Table 19-61: User AES Key 0 Register

User AES Key 0			**USR_AESKEYS_KEY0**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 31:0**

Table 19-62: User AES Key 1 Register

User AES Key 1			**USR_AESKEYS_KEY1**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 63:32**

Table 19-63: User AES Key 2 Register

User AES Key 2			**USR_AESKEYS_KEY2**	**[0x0008]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 95:64**

Table 19-64: User AES Key 3 Register

User AES Key 3			**USR_AESKEYS_KEY3**	**[0x000C]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 127:96**

Table 19-65: User AES Key 4 Register

User AES Key 4			**USR_AESKEYS_KEY4**	**[0x0010]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 159:128**

Table 19-66: User AES Key 5 Register

User AES Key 5			USR_AESKEYS_KEY5	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 191:160**

Table 19-67: User AES Key 6 Register

User AES Key 6			USR_AESKEYS_KEY6	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 223:192**

Table 19-68: User AES Key 7 Register

User AES Key 7			USR_AESKEYS_KEY7	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 255:224**

19.13 TRNG Engine

Random numbers are a vital part of a secure application, providing random numbers that are usable for cryptographic seeds or strong cryptography keys to ensure data privacy. The TRNG is continuously updated by a high-quality, physically-unpredictable entropy source. It generates one random bit per cryptographic clock cycle. The TRNG engine can also be used to generate AES keys.

19.14 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 19-69: TRNG Register Summary

Offset	Register	Name
[0x0000]	*TRNG_CTRL*	TRNG Control Register
[0x0004]	*TRNG_STATUS*	TRNG Status Register
[0x0008]	*TRNG_DATA*	TRNG Data Register

19.14.1 TRNG Register Details

Table 19-70: 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.
14:5	-	RO	0	**Reserved**
4	aeskg_sys	R/W1O	0	**AES System Key Generate** Set this field to 1 to generate a system AES key and transfer the key to the secure system key registers, *SYS_AESKEYS_KEY7*:*SYS_AESKEYS_KEY0* , automatically without user visibility or intervention. This field is cleared by hardware automatically when the key is generated and loaded to the secure system AES key register.
3	aeskg_usr	R/W	0	**AES User Key Generate** Set this field to 1 to generate a user AES key and transfer it to the secure user AES key register, *USR_AESKEYS_KEY7*:*USR_AESKEYS_KEY0*, automatically without user visibility or intervention. This field is cleared by hardware when the key has been generated and transferred to the secure user key register.
2	health_en	R/W	0	**Health Test Interrupt Enable** Set this field to 1 to enable interrupts for the health tests.
1	rnd_ie	R/W	0	**Random Number Interrupt Enable** This bit enables an interrupt to be generated when *TRNG_STATUS.rdy* = 1.
0	odht	R/W	0	**Start On-Demand Health Test** Set this field to 1 to start the on-demand health test.

Table 19-71: TRNG Status Register

TRNG Status			TRNG_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:25	-	RO	0	**Reserved**
24	ld_cnt	RO	0	**Reserved**
23:5	-	RO	0	**Reserved**
4	aeskgd	R/W1C	0	**AES Key Transfer Status** 0: Key transfer in progress. 1: Key transfer complete.
3	srcfail	R/W1C	0	**TRNG Key Source Fail** If this field is set to 1, the TRNG entropy source failed.
2	ht	R/W	0	**Health Test Result** When an on-demand health test is complete (*TRNG_STATUS.ht* = 1) this field indicates the result of the test. 0: Pass 1: Fail
1	odht	R/W	0	**On-Demand Health Test Status** This field is set to 1 by hardware when an on-demand health test is complete.
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 19-72: 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.

20. System AES (AES)

The provided hardware system AES accelerator performs calculations on blocks up to 128 bits.

The features include:

- Multiple key sizes supported:
  - 128 bits.
  - 192 bits.
  - 256 bits.
- DMA support for both receive and transmit channels
- Supports multiple key sources:
  - Encryption using the system AES key
  - Decryption using the system AES key
  - Decryption using the locally generated decryption key

20.1 Instances

Instances of the peripheral are listed in *Table 20-1*.

Table 20-1: MAX32672 AES Instances

Instance	128-Bit Key	192-Bit Key	256-Bit Key	DMA Support
AES	Yes	Yes	Yes	Yes

20.2 Key Management

The system AES key is stored in the write-only *SYS_AESKEYS_KEY7*:*SYS_AESKEYS_KEY0* registers. The key can be loaded by software, stored and automatically loaded on POR from the information block, or generated using the TRNG. If the key is generated and stored in the system AES key registers automatically using the TRNG AES system key generation feature, the key cannot be read or stored for reuse. The TRNG system key generation is for ephemeral key generation only.

Creating a permanent system AES key should be done in a secure environment, such as the manufacturing flow. The initial key creation requires the key to be generated, read by the system software, and stored in a write-only portion of flash information block 1. Once stored, the key is automatically loaded in the system AES key registers on POR and system reset. See *System AES Key Storage* for details on storing a system AES key for automatic loading after a POR. If a valid key is stored in the information block, the *SIR_STATUS.user_cfg_err* field reads 1 after a POR.

If a valid default key is not been programmed in the flash information block, or a key other than the stored key is required, software must write the desired key to the *SYS_AESKEYS_KEY7*:*SYS_AESKEYS_KEY0* registers or use the TRNG to generate an ephemeral system AES key.

Note: The system AES key registers are write only. If the system is set up to automatically load the system AES key after a reset, overwriting the key registers requires a POR to reload the system AES key registers with the stored key.

20.3 Encryption/Decryption of 128-Bit Blocks of Data

The system AES engine automatically handles 128-bit operations using the FIFO. When software writes 128-bits of data to the FIFO, the system AES engine automatically performs an encryption or decryption operation. For large operations, the data should be fed to the FIFO 128-bits per operation until less than 128-bits of data are remaining. Manual operation is

required for blocks less than 128-bits and is described in *Encryption/Decryption of Blocks Less Than 128 Bits*. The *AES_CTRL.start* field is not used for 128-bit blocks of data.

1. Enable the system AES peripheral clock by clearing *GCR_PCLKDIS1.aes* to 0.
2. Enable the TRNG peripheral clock by clearing *GCR_PCLKDIS1.trng* to 0.
3. Wait for hardware to clear *AES_STATUS.busy* = 0.
4. Clear *AES_CTRL.en* = 0 to disable the peripheral.
5. If *AES_STATUS.input_em* is 0, set *AES_CTRL.input_flush* to 1 to flush the input FIFO.
6. If *AES_STATUS.output_em* is 0, set *AES_CTRL.output_flush* to 1 to flush the output FIFO.
7. Clear the AES interrupt flags by writing 0xFFFF FFFF to the *AES_INTFL* register.
8. Select the key size using the *AES_CTRL.key_size* field.
9. Select encryption or decryption:
    a. Set *AES_CTRL.type* to 0 to perform encryption using the system AES key registers.
    b. Set *AES_CTRL.type* to 2 to preform decryption using the locally generated decryption key.
        i. The locally generated decryption key is generated from a previous encryption operation.
10. If interrupts are desired, set the interrupt enable fields in the *AES_INTEN* register.
11. Set *AES_CTRL.en* to 1 to enable the peripheral.
12. Write four 32-bit words of data to the *AES_FIFO* register.
    a. Hardware automatically starts the selected operation when 128-bits are loaded into the FIFO.
13. If the *AES_INTEN.done* interrupt enable is set, an AES_IRQn interrupt occurs when the operation is complete.
14. If not using interrupts, software should read the *AES_STATUS.busy* field until it reads 0.
    a. Verify the operation completed successfully (*AES_INTFL.done* = 1)
15. Read four 32-bit words from the *AES_FIFO* register (least significant word first).
16. Clear the done interrupt flag by writing 1 to *AES_INTFL.done.*
17. Steps 12-16 should be repeated until all 128-bit blocks have been processed.

20.4 Encryption/Decryption of 128-Bit Blocks Using DMA

This example assumes the DMA for reads and writes of data to and from the AES FIFO, although this is not required. The AES could use DMA on one side and software on the other. For each direction, it is required that for each DMA_TX_REQ, the DMA writes four 32-bit words of data into the AES FIFO. Read operations require that for each DMA_RX_REQ, the DMA reads four 32-bit words of data out of the AES FIFO.

The *AES_CTRL.start* field is unused in this case. The state of *AES_STATUS.busy* and the *AES_INTFL.done* fields are indeterminate during DMA operations. Software must clear the done interrupt enable (*AES_INTEN.done* = 0) when using the DMA mode. Use the DMA interrupt to determine when the DMA operation is complete and read the results from the *AES_FIFO* register.

Assuming the DMA continuously fills the data input FIFO, the calculations are complete in the number of cycles shown below.

- 128-bit key: 181 SYS_CLK cycles
- 192-bit key: 213 SYS_CLK cycles
- 256-bit key: 245 SYS_CLK cycles

The procedure to use DMA encryption/decryption is:

1. Enable the system AES peripheral clock by clearing *GCR_PCLKDIS1*.*aes* to 0.
2. Enable the TRNG peripheral clock by clearing *GCR_PCLKDIS1*.*trng* to 0.
3. Initialize the AES receive and transmit channels for the DMA controller.
4. Wait for hardware to clear *AES_STATUS*.*busy* = 0.
5. Clear *AES_CTRL*.*en* to 0 to disable the peripheral.
6. If *AES_STATUS*.*input_em* is 0, set *AES_CTRL*.*input_flush* to 1 to flush the input FIFO.
7. If *AES_STATUS*.*output_em* is 0, set *AES_CTRL*.*output_flush* to 1 to flush the output FIFO.
8. Select the key size using the *AES_CTRL*.*key_size* field.
9. Select encryption or decryption
    a. Set *AES_CTRL*.*type* to 0 to perform encryption using the system AES key.
    b. Set *AES_CTRL*.*type* to 2 to perform decryption using the locally generated decryption key.
        i. The locally generated decryption key is generated from a previous encryption operation.
10. Set *AES_INTEN*.*done* to 0. This is required to use DMA.
11. Set *AES_CTRL*.*en* to 1 to enable the peripheral.
12. The DMA engine fills the FIFO, and hardware begins the AES operation.
13. When the calculation is complete, the AES signals to the DMA that the output FIFO is full and should be emptied. The hardware sets *AES_STATUS*.*output_full* to 1 if the DMA does not empty the FIFO before the next operation is complete.

Steps 12 and 13 are repeated until the DMA completes the amount of data specified.

Note: The interface from the DMA to the AES only works when the amount of data is a multiple of 128 bits. For non-multiples of 128 bits, the remainder of data after calculating all 128-bit blocks must be encrypted or decrypted manually.

20.5 Encryption/Decryption of Blocks Less Than 128 Bits

Encryption or decryption of fewer than 128 bits requires manually starting the AES operation. The following steps describe how to perform encryption/decryption for less than 128-bits of data.

1. Enable the system AES peripheral clock by clearing *GCR_PCLKDIS1.aes* to 0.
2. Enable the TRNG peripheral clock by clearing *GCR_PCLKDIS1.trng* to 0.
3. Wait for hardware to clear *AES_STATUS.busy* to 0.
4. Clear *AES_CTRL.en* to 0 to disable the peripheral.
5. If *AES_STATUS.input_em* is 0, set *AES_CTRL.input_flush* to 1 to flush the input FIFO.
6. If *AES_STATUS.output_em* is 0, set *AES_CTRL.output_flush* to 1 to flush the output FIFO.
7. Clear the AES interrupt flags by writing 0xFFFF FFFF to the *AES_INTFL* register.
8. Select the key size using the *AES_CTRL.key_size* field.
9. Select encryption or decryption:
    a. Set *AES_CTRL.type* to 0 to perform encryption using the system AES key.
    b. Set *AES_CTRL.type* to 2 to perform decryption using the locally generated decryption key.
        i. The locally generated decryption key is generated from a previous encryption operation.
10. If interrupts are desired, set the interrupt enable fields in the *AES_INTEN* register.
11. Set *AES_CTRL.en* = 1 to enable the peripheral.
12. Write one, two, or three 32-bit words of data to the *AES_FIFO* register (least significant word first).
13. Manually start the AES operation by setting *AES_CTRL.start* to 1.
14. If the *AES_INTEN.done* interrupt enable is set, an AES_IRQn interrupt occurs when the operation is complete.
15. If not using interrupts, software should read the *AES_STATUS.busy* field until it reads 0.
    a. Verify the operation completed successfully (*AES_INTFL.done* = 1)
16. Read four 32-bit words from the *AES_FIFO* FIFO (least significant word first).

20.6 Interrupt Events

Multiple interrupt events are supported. Unless specified otherwise, each event has a flag and interrupt enable field in the peripheral register set. 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 already 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 to 0 before setting its interrupt enable field.

The peripheral generates interrupts for the events shown in *Table 20-2*.

Table 20-2: MAX32672 Interrupt Events

Event	Local Interrupt Flag	Local Interrupt Enable
Data Output FIFO Overrun	*AES_INTFL.ov*	*AES_INTEN.ov*
Key All One	*AES_INTFL.key_one*	*AES_INTEN.key_one*
Key All Zero	*AES_INTFL.key_zero*	*AES_INTEN.key_zero*
Key Change	*AES_INTFL.key_change*	*AES_INTEN.key_change*
Calculation Done	*AES_INTFL.done*	*AES_INTEN.done*

20.6.1 Data Output FIFO Overrun

When an AES calculation is completed, the AES signals to the DMA that the data output FIFO is full and that it should be emptied. If the DMA does not empty the FIFO before the next calculation is complete, a data output FIFO overrun event occurs, and the corresponding local interrupt flag is set.

20.6.2 Key One

Attempting a calculation with a key of all 1s generates a key one event.

20.6.3 Key Zero

Attempting a calculation with a key of all zeros generates a key zero event.

20.6.4 Key Change

Writing to any system AES key register while *AES_STATUS.busy* is 1 generates a key change event.

20.6.5 Calculation Done

The transition of *AES_STATUS.busy* = 1 to *AES_STATUS.busy* = 0 generates a calculation done event. The calculation done event interrupt must be disabled when using DMA.

20.7 System AES 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 20-3: AES Register Summary

Offset	Name	Description
[0x0000]	*AES_CTRL*	Control Register
[0x0004]	*AES_STATUS*	Status Register
[0x0008]	*AES_INTFL*	Interrupt Flag Register
[0x000C]	*AES_INTEN*	Interrupt Enable Register
[0x0010]	*AES_FIFO*	Data FIFO

20.7.1 Register Details

Table 20-4: System AES Control Register

System AES Control			AES_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9:8	type	R/W	0	**Operation Type** 0: Encryption using *SYS_AESKEYS_KEY7*:*SYS_AESKEYS_KEY0*. 1: Reserved. 2: Decryption using the locally generated decryption key. 3: Reserved. *Note: To perform a decryption using the SYS_AESKEYS_KEY7:SYS_AESKEYS_KEY0 registers, an encryption operation must be performed first.*
7:6	key_size	R/W	0	**Key Size** 0: 128 bits. 1: 192 bits. 2: 256 bits. 3: Reserved.

System AES Control			AES_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
5	output_flush	R/W1O	0	**Flush Output FIFO** This field always reads 0. 0: Normal operation. 1: Flush.
4	input_flush	R/W1O	0	**Flush Input FIFO** This field always reads 0. 0: Normal operation. 1: Flush.
3	start	R/W1O	0	**Start AES Calculation** This field forces the start of an AES calculation regardless of the state of the input FIFO. This permits an AES calculation on less than 128-bits of data since the AES calculation normally starts when the data input FIFO is full. This field always reads 0. 0: No Action. 1: Start calculation.
2	dma_tx_en	R/W	0	**DMA Request to Write Data Input FIFO** 0: Disabled. 1: Enabled. DMA request is generated if the data input FIFO is empty.
1	dma_rx_en	R/W	0	**DMA Request to Read Data Output FIFO** 0: Disabled. 1: Enabled. DMA request is generated if the data output FIFO is full.
0	en	R/W	0	**AES Enable** 0: Disabled. 1: Enabled.

Table 20-5: System AES Status Register

System AES Status			AES_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	output_full	R	0	**Output FIFO Full** This field reads 1 if the output FIFO is full. This field is automatically cleared to 0 when the output FIFO is no longer full. 0: Not full. 1: Full.
3	output_em	R	0	**Output FIFO Empty** This field reads 1 if the output FIFO is empty. Hardware automatically clears this field when data is available in the output FIFO. 0: Not empty. 1: Empty.
2	input_full	R	0	**Input FIFO Full** This field reads 1 when the input FIFO is full. Hardware automatically clears this field when the input FIFO is no longer full. 0: Not full. 1: Full.

System AES Status			AES_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
1	input_em	R	0	**Input FIFO Empty** This field reads 1 if the input FIFO is empty. Hardware automatically clears this field to 0 when data is written to the input FIFO. 0: Not empty. 1: Empty.
0	busy	R	0	**AES Busy** This field reads 1 when an AES operation is active. Hardware automatically clears this field to 0 when the operation is complete. 0: Not busy. 1: Busy.

Table 20-6: System AES Interrupt Flag Register

System AES Interrupt Flag			AES_INTFL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	key_one	R/W1C	0	**Key All One Event Interrupt** This field is set if all 1s are loaded in the system AES key registers and an AES operation is started. Write 1 to clear. 0: Normal operation. 1: Event occurred.
3	ov	R/W1C	0	**Data Output FIFO Overrun Event Interrupt** 0: Normal operation. 1: Event occurred.
2	key_zero	R/W1C	0	**Key All Zero Event Interrupt** This field is set if all 0s are loaded in the system AES key registers and an AES operation is started. Write 1 to clear. 0: Normal operation. 1: Event occurred.
1	key_change	R/W1C	0	**Key Change Event Interrupt** 0: Normal operation. 1: Event occurred.
0	done	R/W1C	0	**Calculation Done Event Interrupt** 0: Normal operation. 1: Event occurred.

Table 20-7: System AES Interrupt Enable Register

System AES Interrupt Enable			AES_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	key_one	R/W1C	0	**Key All One Event Interrupt Enable** 0: Enabled. 1: Disabled.
3	ov	R/W1C	0	**Data Output FIFO Overrun Event Interrupt Enable** 0: Enabled. 1: Disabled.

System AES Interrupt Enable			AES_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
2	key_zero	R/W1C	0	**Key Zero Event Interrupt Enable** 0: Enabled. 1: Disabled.
1	key_change	R/W1C	0	**Key Change Event Interrupt Enable** 0: Enabled. 1: Disabled.
0	done	R/W1C	0	**Calculation Done Event Interrupt Enable** This event interrupt must be disabled when using DMA. 0: Enabled. 1: Disabled.

Table 20-8: System AES FIFO Register

System AES Data			AES_FIFO	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**AES FIFO** Writing this register puts data to the input FIFO. Hardware automatically starts a calculation after 4 words are written to this FIFO. The data should be written least significant word first. Reading this register pulls data from the output FIFO. The least significant word is read first.

20.8 System 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. The registers are only reset on a POR and the reset value is dependent configuration in information block 1. See *System AES Key* for details.

Table 20-9: System AES Key Register Summary

Offset	Name	Description
[0x0000]	*SYS_AESKEYS_KEY0*	*System AES Key 0 Register*
[0x0004]	*SYS_AESKEYS_KEY1*	*System AES Key 1 Register*
[0x0008]	*SYS_AESKEYS_KEY2*	*System AES Key 2 Register*
[0x000C]	*SYS_AESKEYS_KEY3*	*System AES Key 3 Register*
[0x0010]	*SYS_AESKEYS_KEY4*	*System AES Key 4 Register*
[0x0014]	*SYS_AESKEYS_KEY5*	*System AES Key 5 Register*
[0x0018]	*SYS_AESKEYS_KEY6*	*System AES Key 6 Register*
[0x001C]	*SYS_AESKEYS_KEY7*	*System AES Key 7 Register*

20.8.1 Register Details

Table 20-10: System AES Key 0 Register

AES Key 0			SYS_AESKEYS_KEY0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	WO	*	**System AES Key bits 31:0**

Table 20-11: System AES Key 1 Register

AES Key 1			SYS_AESKEYS_KEY1	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	WO	*	**System AES Key bits 63:32**

Table 20-12: System AES Key 2 Register

AES Key 2			SYS_AESKEYS_KEY2	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	WO	*	**System AES Key bits 95:64**

Table 20-13: System AES Key 3 Register

AES Key 3			SYS_AESKEYS_KEY3	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	WO	*	**System AES Key bits 127:96**

Table 20-14: System AES Key 4 Register

AES Key 4			SYS_AESKEYS_KEY4	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	WO	*	**System AES Key bits 159:128**

Table 20-15: System AES Key 5 Register

AES Key 5			SYS_AESKEYS_KEY5	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	WO	*	**System AES Key bits 191:160**

Table 20-16: System AES Key 6 Register

AES Key 6			SYS_AESKEYS_KEY6	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	WO	*	**System AES Key bits 223:192**

Table 20-17: System AES Key 7 Register

AES Key 7			SYS_AESKEYS_KEY7	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	WO	*	**System AES Key bits 255:224**

21. 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 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 any of the bootloader activation events shown in *Table 21-1*. The stimulus includes receipt of the synchronization pattern and may or may not involve a physical pin, as shown in *Table 21-1*. 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.

21.1 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 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.

21.2 Instances

There is only one instance of the SCPBL.

The device will attempt to start the SCPBL following any of the bootloader activation events shown in *Table 21-1*. See *MAX32672 Bootloader Activation* for details.

The stimulus pin can be reassigned to other GPIO pins as shown in *Table 21-1*. 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.

Table 21-1: MAX32672 Bootloader Characteristics

Interface	Interface ID	Interface Pins	Default Stimulus Pin	Stimulus Pin Reassignment Options	Default SCPBL Activation Interval	Bootloader Activation Events
UART (115200bps)	0x00	UART0_RX (P0.8) UART0_TX (P0.9) RSTN	P0.10 (active low)	All GPIO except: UART0A_RX UART0A_TX	6 seconds	POR System Reset (including RSTN and WDT resets) Exit from BACKUP Exit from STORAGE

Multiple methods and algorithms ensure the integrity and authority of SCPBL communication, as shown in *Table 21-2*.

Table 21-2: MAX32672 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 (secp256r1)	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 generated using an AES-CBC-MAC 128 encryption with a 16-byte key of 0x00. The value is the MSB of the 16-byte digest.	1
Data Checksum	End of transport layer	Session layer data	The data checksum is generated using an AES-CBC-MAC 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	Second nibble of the session layer	Session layer	0xA: ECDSA-256 (secp256r1)	4 bits
MRK/CRK/TCRK	Bootloader authentication keys	N/A	ECDSA-256 (secp256r1)	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 VRTC, 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.

21.3 Secure Boot

Following any of the bootloader activation events shown in *Table 21-1*, the device executes a secure boot from the ROM to verify the integrity of the code in the flash memory. This establishes the chain of trust that secure application execution relies on:

- The platform knows the application software has not been accidentally or maliciously modified.
- The platform knows the application software is from a trusted, authorized source.
- The application software ensures it is running on a trusted platform:

A device with the secure boot feature must be initialized through the SCPBL before it executes the application software:

- The application image must be created and signed with the CRK
- The CRK must be loaded.
- The application image must be loaded.

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.

21.4 Selecting the Programming Interface

Product development and debugging are typically performed over the simpler, faster JTAG interface. The SCPBL is most often used after deploying a product to the field for software updates. The SCPBL has a slower transfer rate, but it ensures that communications are trusted and authenticated.

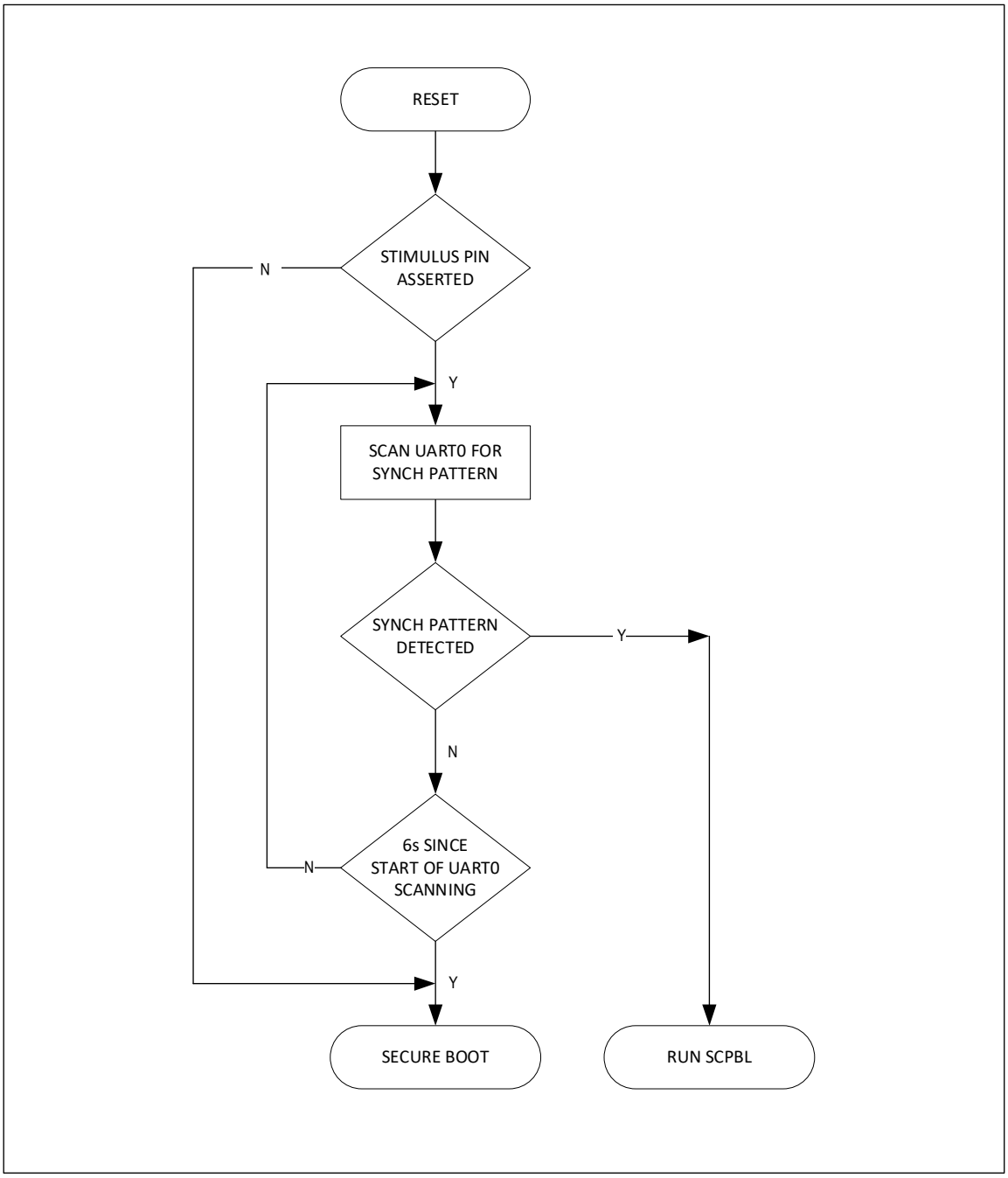
21.5 MAX32672 Bootloader Activation

Two conditions are required following any of the bootloader activation events show in *Table 21-1* to activate the bootloader:

- Detection of the stimulus, as described in *Figure 21-1*, and
- 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 21-1*. The device executes the secure boot process if the stimulus is not active or the pattern is not seen.

Figure 21-1: MAX32672 Bootloader Activation Flow



The device 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 any bootloader activation event.

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.

21.6 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 root key (TCRK) pair.

21.6.1 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.

21.6.2 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.

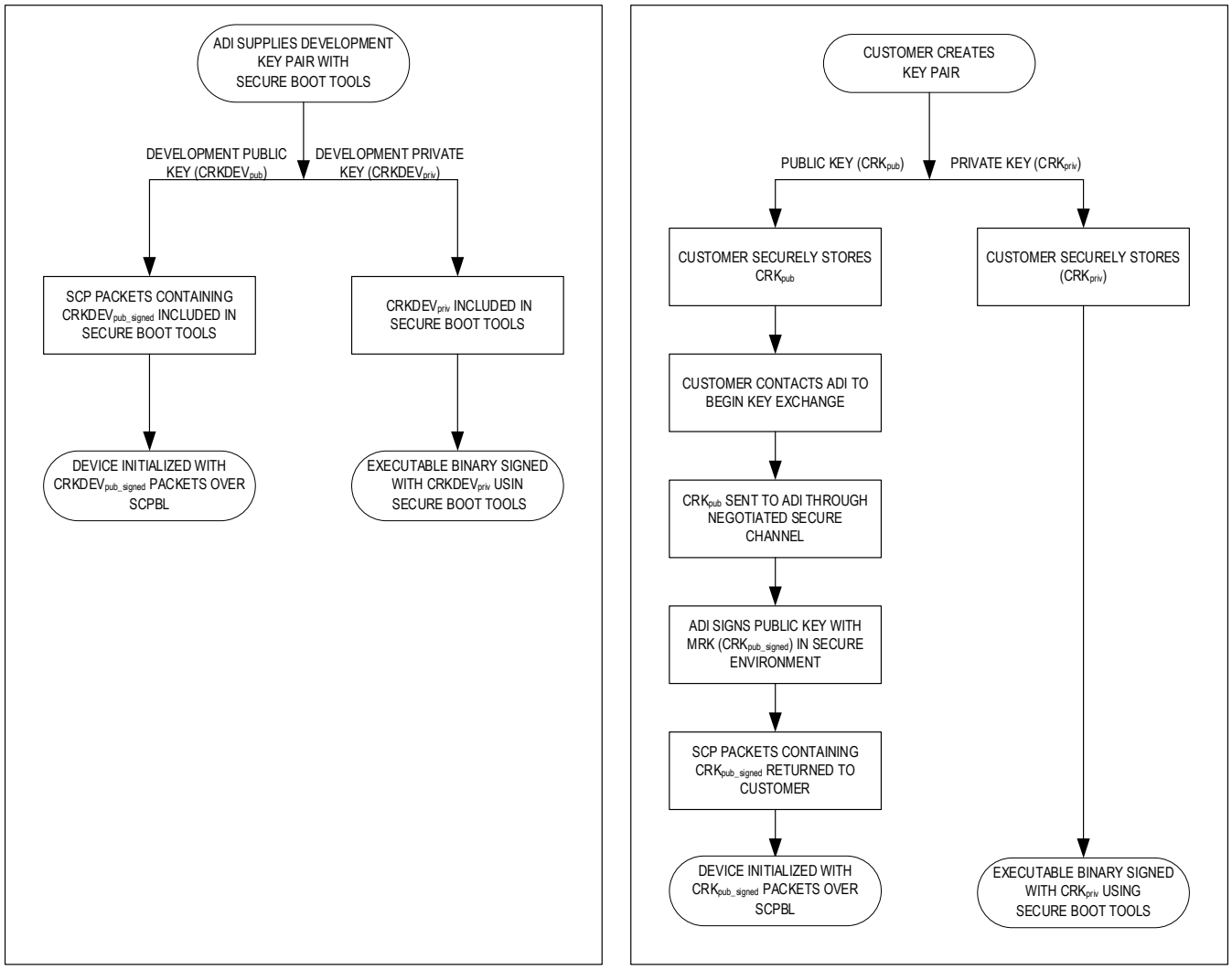
21.6.3 Test CRK (TCRK)

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 21-2 shows a high-level overview of the key generation and development described in the application note.

Figure 21-2: Customer Root and Development Key Generation and Usage



21.7 Secure Program Loading

The secure program loading procedure instructs how to configure the device, build and load the application software, and activate the security features. Steps 1 and 2 need to be done only once by the customer. Steps 3 and 4 are only performed once per device.

21.8 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 21-3*. has multiple parts.

Figure 21-3: Application Image

The elements of the image are shown graphically in *Table 21-3*.

Table 21-3: Application Image Structure (ECDSA-256)

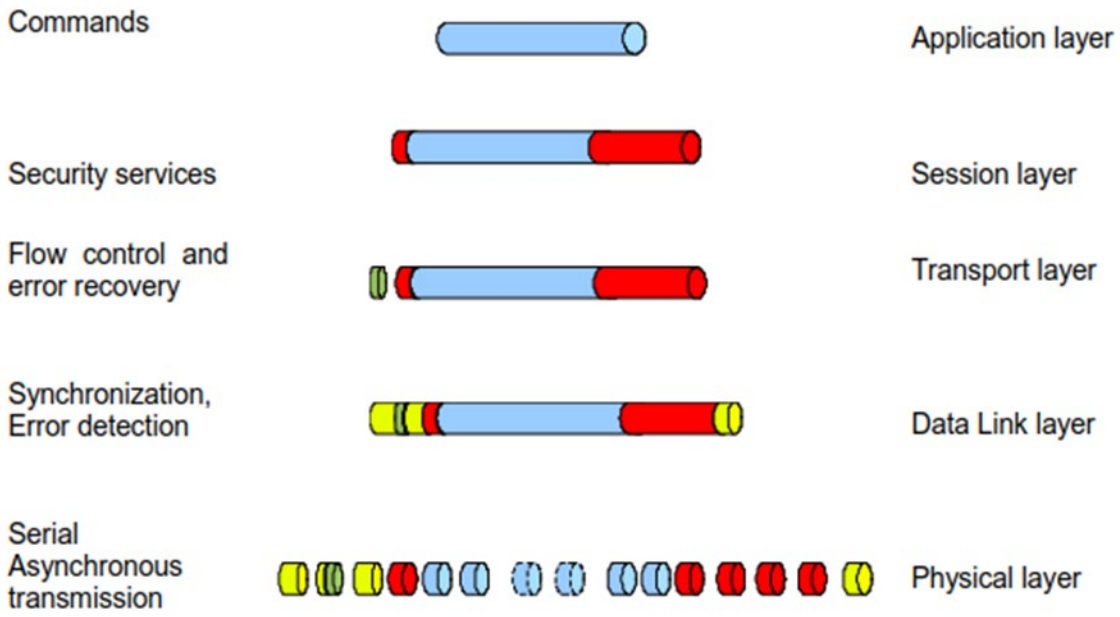
Element	Absolute Start Address Within Application Image	Length (Bytes	Value
Header ID	0x0000 0000	8	See *Table 21-2*.
Format Version (ROM_REF)	0x0000 0008	4	0x0100 0000 (default defined by device version)
Application Image Start Address	0x0000 000C	4	0x1000 0000
Application Image Signature Offset	0x0000 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 SBT is 0x0100 0000 corresponding to Version 1.0.0)
Padding	0x0000 0020	480	Padded with 0xFF out to 0x0000 01FF
Software Image	0x0000 0400	Variable	Software image
Terminal padding to 4B boundary, if necessary	End of executable binary	0 1 2 3	N/A 0xFF 0xFFFF 0xFFFFFF
Application Image Signature	End of executable binary + terminal padding	256 bits	ECDSA-256 signature

21.9 SCP Session

The SCP is a session-based protocol that securely exchanges data packets between the host and SCPBL.

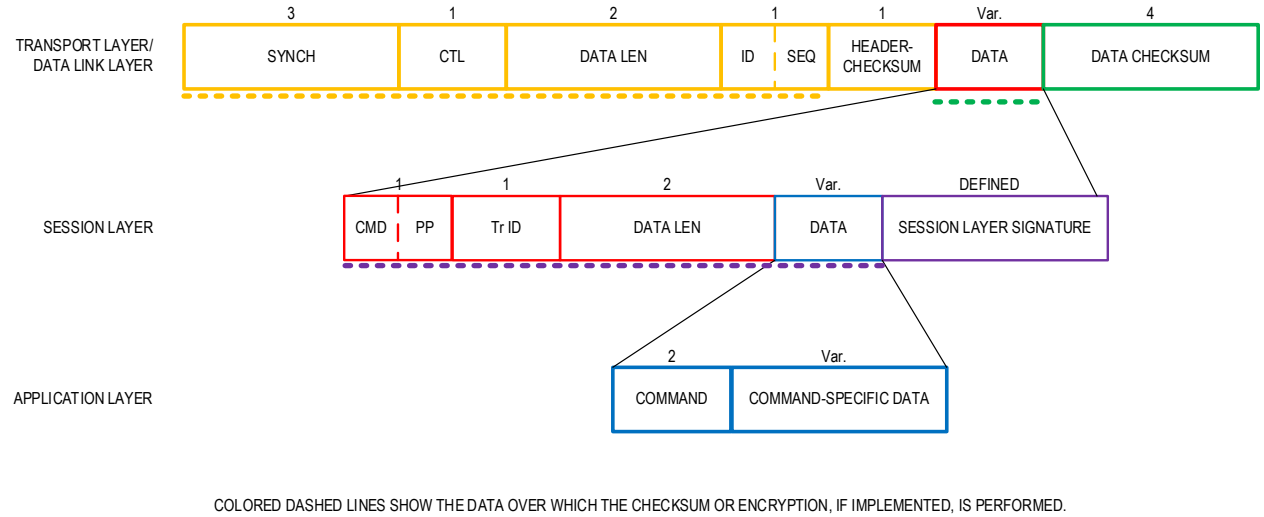
The SCP consists of a stack of layers based on the OSI model shown in Figure 29 5. The application layer contains basic commands to configure the SCPBL. It also includes the signed customer application and a set of WRITE data commands used for the SCPBL, including the write command that loads the code (the executable software) into the program memory

Figure 21-4: SCPBL Implementation of OSI Model



The general structure of a packet is shown in *Figure 21-5*. Simpler commands only require a subset of these elements. Multiple integrity and authentication steps are used throughout an SCP packet.

Figure 21-5: SCP Packet Structure



COLORED DASHED LINES SHOW THE DATA OVER WHICH THE CHECKSUM OR ENCRYPTION, IF IMPLEMENTED, IS PERFORMED.

21.9.1 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 115,200bps. Hardware flow control is not implemented.

21.9.2 Data Link Layer

The Data Link Layer accumulates bytes from the Physical Layer into a Protocol Data Unit (PDU) called a frame (logical, structured packets for data) to ensure data integrity.

Each frame contains a header identifying the type of packet and is shown in yellow in *Figure 21-5,* followed by an optional data portion. The structure of the header is shown in *Table 21-4*.

Table 21-4: 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 21-2*.

21.9.3 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

21.9.4 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.

21.10 Session, Sequence, and Transaction ID

The SCP incorporates multiple indexing schemes as an error-detection mechanism:

- The session ID in the header of the transport layer
- The sequence ID in the header of the transport layer
- The transaction ID in the header of the session layer

The session ID is set by the host in the first command CON_REQ, and remains the same through the DISC_REP command which ends the session. Any command with a session ID different that the one used in the CON_REQ command will cause the SCPBL to abort the session. The host should abort any session in which it detects a different CON_REQ value.

The SCPBL automatically increments its internal counters 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.

The incrementing rules of the SEQ and TrID values are summarized in *Table 21-6*.

21.11 Opening an SCP Session

The host initiates a new session request by sending a COM_REQ to the SCPBL. The procedure is shown in *Table 21-5*. 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 21-5: 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				

21.12 SCPBL Command Summary

Table 21-6: 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	-	-	A connection request from the host
CON_REP	0x02	N/A	N/A	Yes	No	No	A connection acceptance from the SCPBL
DISC_REQ	0x03	N/A	N/A	No	-	-	A disconnection request from the host
DISC_REP	0x04	N/A	N/A	No	-	-	A 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
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.*

21.13 Transport/Session Layer Command Details

21.13.1 CON_REQ

The host sends a CON_REQ to the SCPBL to initiate a connection.

The sequence number is always 0x0 for this command.

The session ID encoded in this command becomes the session ID for all SCP transactions.

Figure 21-6: CON_REQ Command Structure

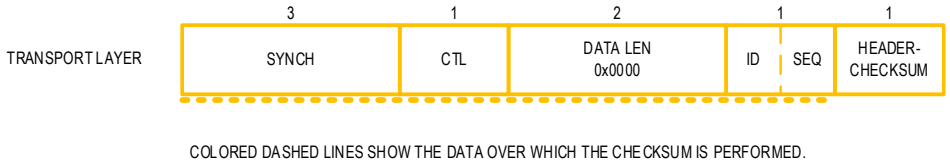


Table 21-7: CON_REQ

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

21.13.2 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 0x0 for this command.

Figure 21-7: CON_REP Command Structure

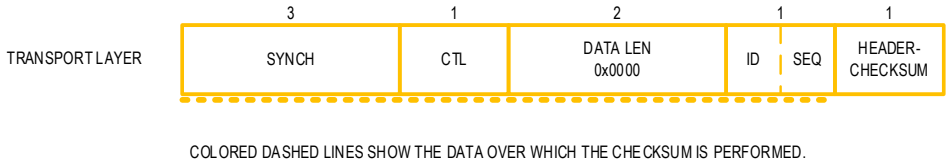


Table 21-8: CON_REP

CON_REQ	0x02	Connection Request	
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

21.13.3 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 21-8: DISC_REQ Command Structure

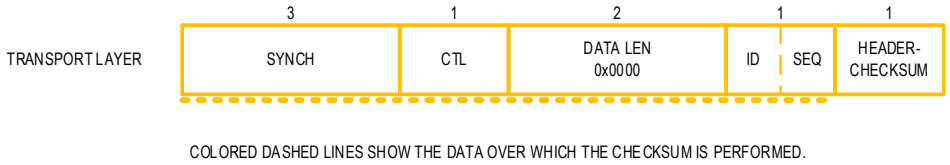


Table 21-9: DISC_REQ

DISC_REQ	0x03	Connection 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

21.13.4 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.

This command may or may not generate an ACK.

Figure 21-9: DISC_REP Command Structure

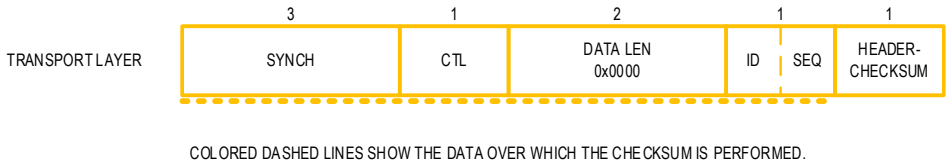


Table 21-10: DISC_REP

DISC_REQ	0x04	Connection Request	
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

21.13.5 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 21-10: ACK Command Structure

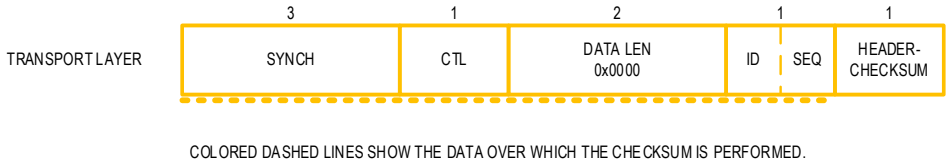


Table 21-11: ACK

ACK	0x06	Connection Request	
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

21.13.6 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 21-11: HELLO Command Structure

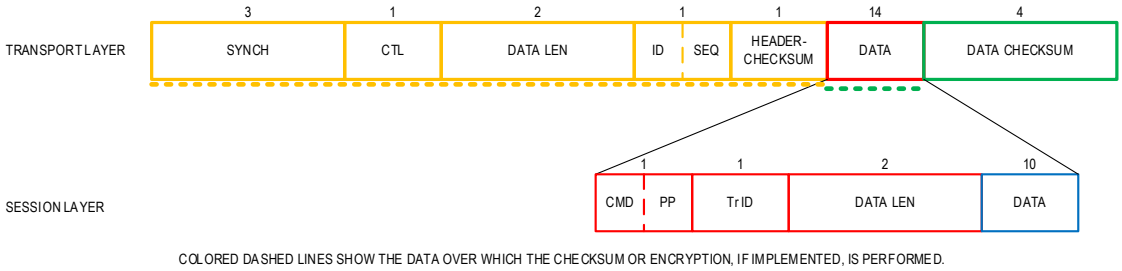


Table 21-12: HELLO Command

HELLO	0x05	Connection Request	
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	Upper 4 bits	0x1
PP		Lower 4 bits	0x0 (There is no session layer signature for this command.)
Transaction ID	0x01	1	Variable
DATA LEN	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

21.13.7 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
- JTAG status
- USN

Table 21-13: HELLO_REPLY Structure

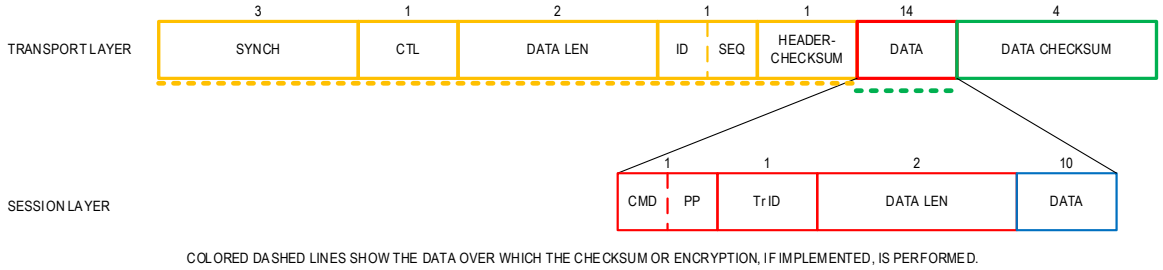


Table 21-14: HELLO_REPLY Command

HELLO_REPLY	0x05	Connection Request
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	0x0 (There is no session layer signature for this command.)
Transaction ID	0x01	1	Variable
DATA LEN	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

21.13.8 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 21-12: COMMAND_RSP Command Structure

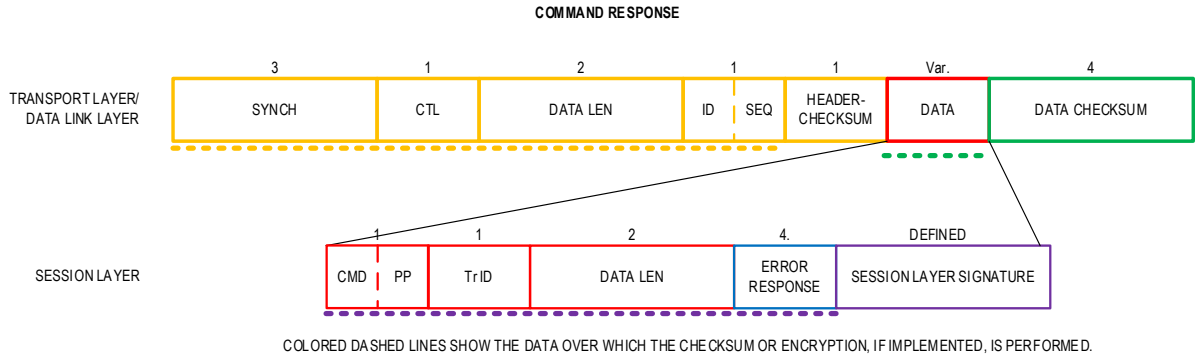


Table 21-15: COMMAND_RSP

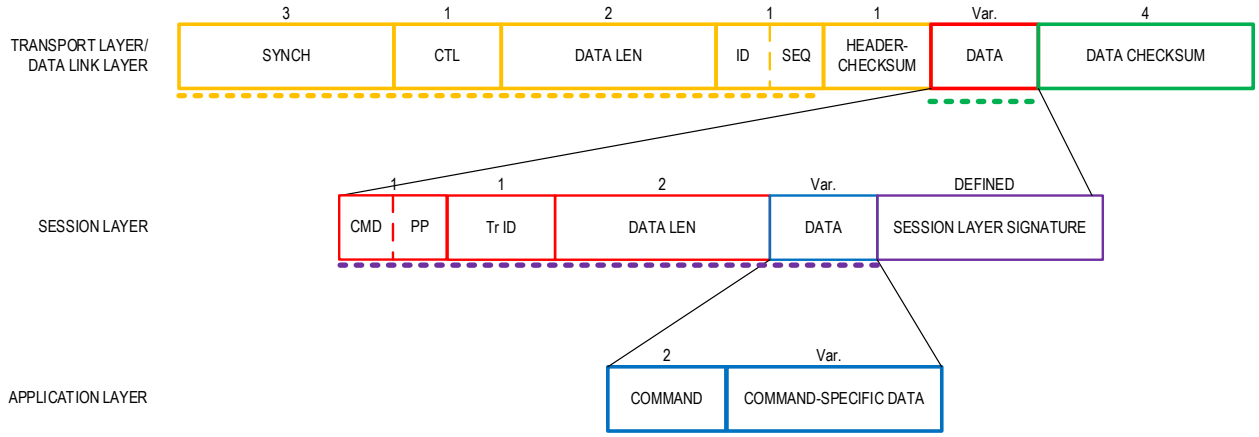
COMMAND_RSP		0x05	Success/Failure Response from Application Layer Command	
Command Format (Transport Layer)				
Element		**Offset**	**Length (Bytes)**	**Values**
SYNCH		0x00	3	0xBEEFED
CTL		0x03	1	0x05
DATA LEN		0x04	2	0x0008
ID		0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ			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 21-2* for the protection profile value.
Transaction ID		0x01	1	Varies
DATA LEN		0x04	2	0x0004
RESPONSE		0x04	4	0x00000000: Success Any other value: Failure

21.14 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 21-5* and is blue.

Application layer commands are a subset of the data transfer commands and have the general structure shown in *Figure 21-13*.

Figure 21-13: DATA_TRANSFER/Application Layer Command Structure



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.

Table 21-16: DATA_TRANSFER/Application Layer Command Structure

WRITE_DATA	**0x05**	**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 21-2* for the protection profile value.
Transaction ID	0x01	1	Varies

WRITE_DATA	0x05	Write Data To Memory or Configuration Command	
DATA LEN	0x02	2	Dependent on the application layer command.
DATA	0x04	10	User-defined
SIGNATURE	0x0E	1	Defined by the device.

21.14.1 WRITE_CRK

Table 21-17: WRITE_CRK

WRITE_CRK	0x4701	Write CRK to Device	
Description	This command writes the CRK to the nonvolatile memory of the SCPBL. This command can only be executed once per device. The CRK value is the same as the MRK until this command executes. The new CRK value does not take effect until the current session terminates and a new one starts. The customer will never directly execute this command. The CRK values can only be written using pre-generated packets supplied to the customer as part of the key signing procedure. The command is included here only for completeness of the instruction set.		
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)

21.14.2 REWRITE_CRK/RENEW_CRK

Table 21-18: REWRITE_CRK/RENEW_CRK

REWRITE_CRK RENEW_CRK	0x461A	Write a Second 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. The CRK value is the same as the MRK until this command executes. The new CRK value does not take effect until the current session terminates and a new one starts. The customer will never directly execute this command. The CRK values can only be written using pre-generated packets supplied to the customer as part of the key signing procedure. The command is included here only for completeness of the instruction set.		
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)

21.14.3 WRITE_OTP

Table 21-19: 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 (fixed)
Arguments	0x0002	Variable	

21.14.4 WRITE_TIMEOUT

Table 21-20: 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 activate 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 21-1* 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
Target	0x0002	1	See *Table 21-1*.
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.

21.14.5 WRITE_PARAMS

Table 21-21: 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 (fixed)
Target	0x0002	1	See *Table 21-1*.
Data	0x0003	Variable	The length of the field is device-specific.

21.14.6 WRITE_STIM

Table 21-22: WRITE_STIM

WRITE_STIM	0x4428			Configure SCPBL Stimulus Pin					
Description	This command allows the SCPBL to permanently change the location and polarity of the SCPBL stimulus pin. Valid locations are shown in *Table 21-1*. Regardless of specific settings, the value of this byte must not be 0xFF. This command can only be executed once.								
	Bit	**7**	**6**	**5**	**4**	**3**	**3**	**2**	**1**	**0**
		GPIO Port		**Active State**	**GPIO Pin of Selected Port**					
	Function	0b00: Port 0 0b01: Port 1 0b10: Port 2 0b11: Port 3		0: Low 1: High	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					

Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x4428
Stimulus Location	0x0002	1	See *Table 21-1* for valid pin options.

21.14.7 WRITE_SLA_VERSION

Table 21-23: 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 are 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)

21.14.8 WRITE_DEACTIVATE

Table 21-24: 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 (fixed)
Link	0x0002	1	See *Table 21-1* for the instance ID of the desired interface

21.14.9 WRITE_DATA

Table 21-25: 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

21.14.10COMPARE_DATA

Table 21-26: COMPARE_DATA

COMPARE_DATA	0x2403	Compare Data Against Internal Memory	
Description	The command compares the supplied data against the contents of the internal flash or RAM based on the target address.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x2403 (fixed)
Start Address	0x0002	4	This is the location of the target start address. The MSB of the address is sent first.
Length	0x0004	4	This is the length of the data segment in bytes. The MSB of the length is sent first.
Data	0x0006	Variable	Compare data

21.14.11ERASE _DATA

Table 21-27: ERASE_DATA

ERASE_DATA	0x4401	Erase a Range of Flash Memory	
Description	Erases the length number of bytes of the 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.

21.14.12EXECUTE_CODE

Table 21-28: EXECUTE_CODE

EXECUTE_CODE	0x2101	Execute Code From Flash Memory	
Description	The execute command is used indirectly as part of secondary-level applets. Memory commands are 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.

22. Silicon Revision Differences

The current silicon revision of the MAX32672 is B2. Read the *GCR_REVISION.revision* field to determine a device's silicon revision. For a list of known issues for each silicon revision refer to the device's errata sheet at *http://www.analog.com/MAX32672*.

22.1 Differences Between the B2 and B1 Revision

- The *GCR_REVISION.revision* field reads 0xB2.
- ECC operations fixed to work with DMA and other bus masters. (21206, 21207, 21204)
- Cache Read Buffer (CRB) is disabled.

22.2 Differences Between the B1 and A1 Revision

- The *GCR_REVISION.revision* field reads 0xB1.
- The DAP can only be disabled using the SCPBL WRITE_OTP command.
- Added DMA read access support for the flash memory.
- All GPIO default to input disabled except for the SWD pins. See *Power-On-Reset Configuration* for details.
- The I^2C target address registers were moved. See *Target Mode Addresses* for details.
- Updated watchdog tmer to support different sequences for enable, disable, and feed. See *WDT Protection Sequence* for details.
- Flash information block 1:
  - Once the flash magic word is programmed, flash information block 1 cannot be erased, and the system AES keys cannot be modified.
- Moved RTC reset from *GCR_RST0*[17] to *MCR_RST*[3]. See the *RTC_CTRL* register for additional details.
- Added *MCR_ADC_CFG3* register, which is used for loading trim values for the ADC. This addition changes how the reference trims are loaded. See *ADC SFR Interface* for details.
- Increased *ADC_SAMPCLKCTRL.track_cnt* field from 8 bits to 16 bits. See the ADC *Clocks and Timing* section for details.
- Temperature calculations changed. See *Temperature Sensor* for details.

22.3 Initial Silicon Revision A1

- The *GCR_REVISION.revision* field reads 0xA1.

23. Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION
0	10/21	Initial release
1	4/23	Updated SPI block diagram, *Figure 12-1*, to remove QSPI nomenclature. Update SPI instance table, *Table 12-1*, to show 3 SPI instances. Updated SPI *Peripheral Clock* calculations. Updated *Function Control Registers (FCR)* and added ADC trim and temperature sensor registers. Updated *System AES (AES)* to remove references to privileged mode. Added missing field (*user_cfg_err*) in *SIR_STATUS* register. Updated *System AES Key Storage* section with absolute addresses for writing system AES keys and magic values. Updated TRNG Engine. Updated *CTB_CIPHER_CTRL*.*enc* to show correct values for encryption and decryption. Renamed SYS_AES_ to AES_, see *System AES Registers* for details. Updated *AES_CTRL*.*type* field to address a missing setting. Added *MCR_ADC_CFG3* register. Updated *ADC SFR Interface* section to use *MCR_ADC_CFG3* register to load reference trim values. Updated I²C *Target Mode Addresses* to show usage of multiple target address registers. Updated the terms “master/slave” to “controller/target” throughout except for the *I2Cn_SLAVE3*:*I2Cn_SLAVE0* registers to maintain backward compatibility with existing software releases. Marked *Flash Controller Registers* as only reset on POR. Updated *Table 13-1* (timer instance table) to show correct clock options. Updated Elliptic Curve support to sizes of 256, 384, and 521 bits only, see *Cryptographic Toolbox (CTB)*. Added note that reading from blank flash results in an ECC error, see *Flash Error Correction Coding* for details. Updated *Standard DMA (DMA)* to remove references to supporting data memory only. Flash memory is now supported. Updated *Temperature Sensor* calculation. Complete chapter update for *Debug Access Port (DAP)*. Corrected reset value for *GCR_PCLKDIS0*.*gpio0* and *GCR_PCLKDIS0*.*gpio1* fields. Updated GPIO *Configuration* to add details on peripheral clock enable and to update POR configuration. Updated *Configuring GPIO (External) Interrupts* to add details on GPIO wake interrupts. Updated reset values for *GPIOn_EN0*, *GPIOn_INEN*, *GPIOn_PADCTRL0* and registers. Updated *Table 12-4* to correct SPI mode table. Updated *Figure 4-1, Figure 4-3*, *Figure 4-4*, and *Figure 4-5* to show DIV_CLK_OUT for 56-pin TQFN and other updates to power control. Updated DMA *Usage* instructions for count reload. Marked *DMA_CHn_CNTRLD*.*en* as reserved, must be set to 0. Updated Flash *Clock Configuration* section to show correct clock calculation. Added *Silicon Revision Differences* chapter to describe changes for each revision. Added *MCR_RST*.*rtc* field for resetting the RTC peripheral on all revisions except 0xA1.

REVISION NUMBER	REVISION DATE	DESCRIPTION
2	2/24	Updated *FCR_TS0* and *FCR_TS1* registers to remove equations and point to *Temperature Sensor* for details. Updated *Temperature Sensor* section to show temperature conversion equations (*Equation 18-5*, *Equation 18-6*) for revision 0xA1 and all other device revisions. Added detailed steps for each revision to show how to load the internal and external reference trims. See *ADC SFR Interface* for details. Updated *Differences Between the B1 and A1 Revision* to add temperature sensor calculations that changed from 0xA1 to 0xB1 revisions. Added missing bit *SPIn_CTRL2.sclk_fb_inv*. Updated *Table 12-4* to show settings for *SPIn_CTRL2.sclk_fb_inv*. Corrected description of *UARTn_DMA.rx_thd_val*. Added instructions for calibrating the IPO. See *IPO Calibration* for details. Added details for enabling the IPO and selecting it as the system oscillator. See *100MHz Internal Primary Oscillator (IPO)* for details. Marked *GPIOn_WKEN* register as Reserved, Do Not Modify. Updated *Device Resets* to clarify Peripheral and Soft reset behavior. Updated *TMRn_CNT* and *TMRn_PWM* registers to indicate the timer's clock must be enabled (*TMRn_CTRL0.en* = 1) before writes to these registers. Added *Enabling the ERTCO* instructions. Clarified the bootloader activation events in *Table 21-1*.