MAX32670/MAX32671 User Guide

UG7202; Rev 4; 09/23

Abstract: This user guide provides application developers information on how to use the memory and peripherals of the MAX32670/MAX32671 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.

MAX32670/MAX32671 User Guide

Table of Contents

Table of Figures

List of Tables

Table of Equations

1. Introduction

For ordering information, mechanical and electrical characteristics for the MAX32670/MAX32671 family of devices please refer to the data sheet.

1.1 Related Documentation

The MAX32670/MAX32671 data sheet and errata are available from the Analog Devices website.

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 set 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

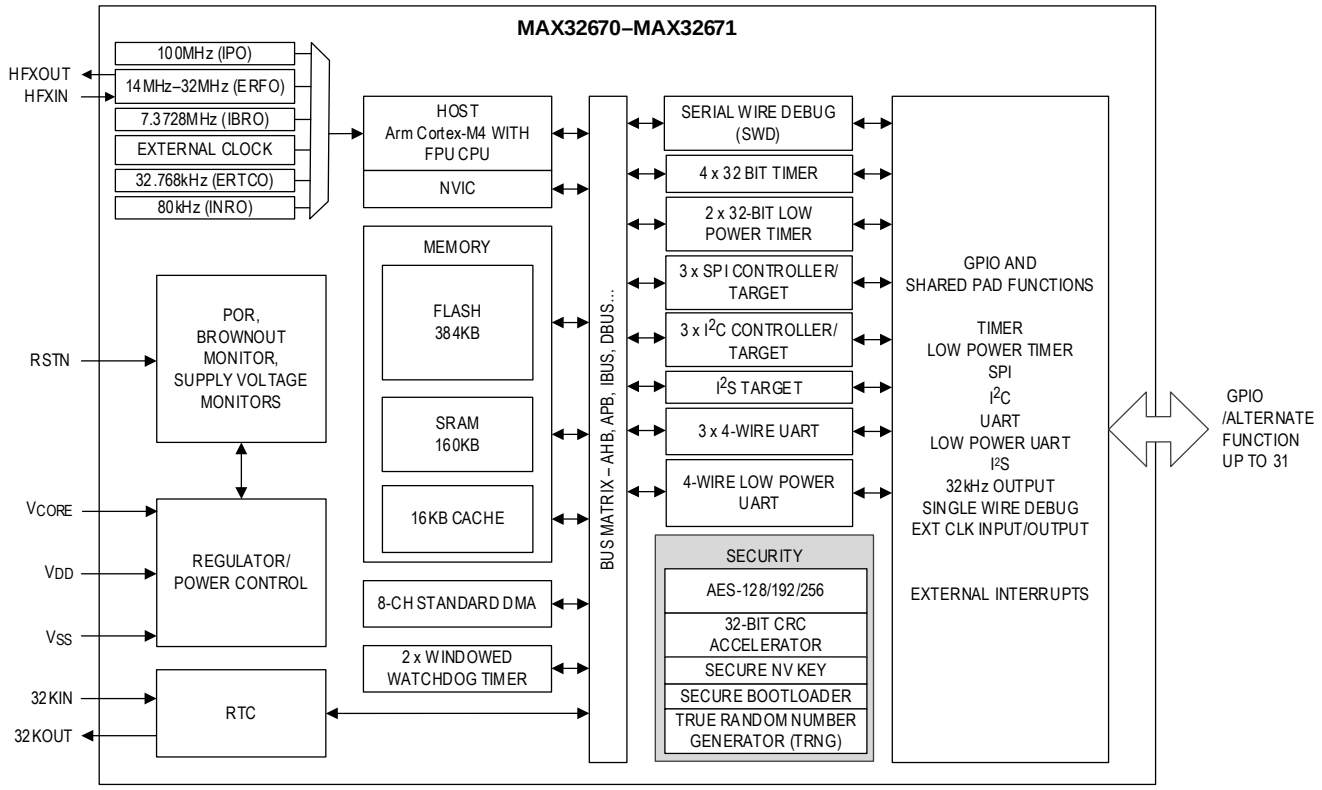
The MAX32670/MAX32671 is an ultra-low-power, cost-effective, high reliability 32-bit microcontroller enabling designs with complex sensor processing without compromising battery life. It combines a flexible and versatile power management unit with the powerful Arm Cortex-M4 core with FPU. It also offers legacy designs an easy and cost optimal upgrade path from 8- or 16-bit microcontrollers. The device integrates up to 384KB of flash memory and 160KB of SRAM to accommodate application and sensor code.

The device features five powerful and flexible power modes. It can operate from a single-supply battery or a dual-supply typically provided by a PMIC. The I^2C ports support standard, fast, fast-plus, and high-speed modes, operating up to 3400kbps. The SPI ports can run up to 48MHz in both controller and target mode, and three standard UARTs and one low power UART. Four general-purpose 32-bit timers, two low power 32-bit timers, two windowed watchdog timers, and a real-time clock (RTC) are also provided. An I^2S interface provides digital audio streaming to a codec.

The high-level block diagram for the MAX32670/MAX32671 is shown in *Figure 2-1*.

2.1 Block Diagram

Figure 2-1: MAX32670/MAX32671 Block Diagram



3. Memory, Register Mapping, and Access

3.1 Memory, Register Mapping, and Access Overview

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

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

Figure 3-1: Code Memory Mapping

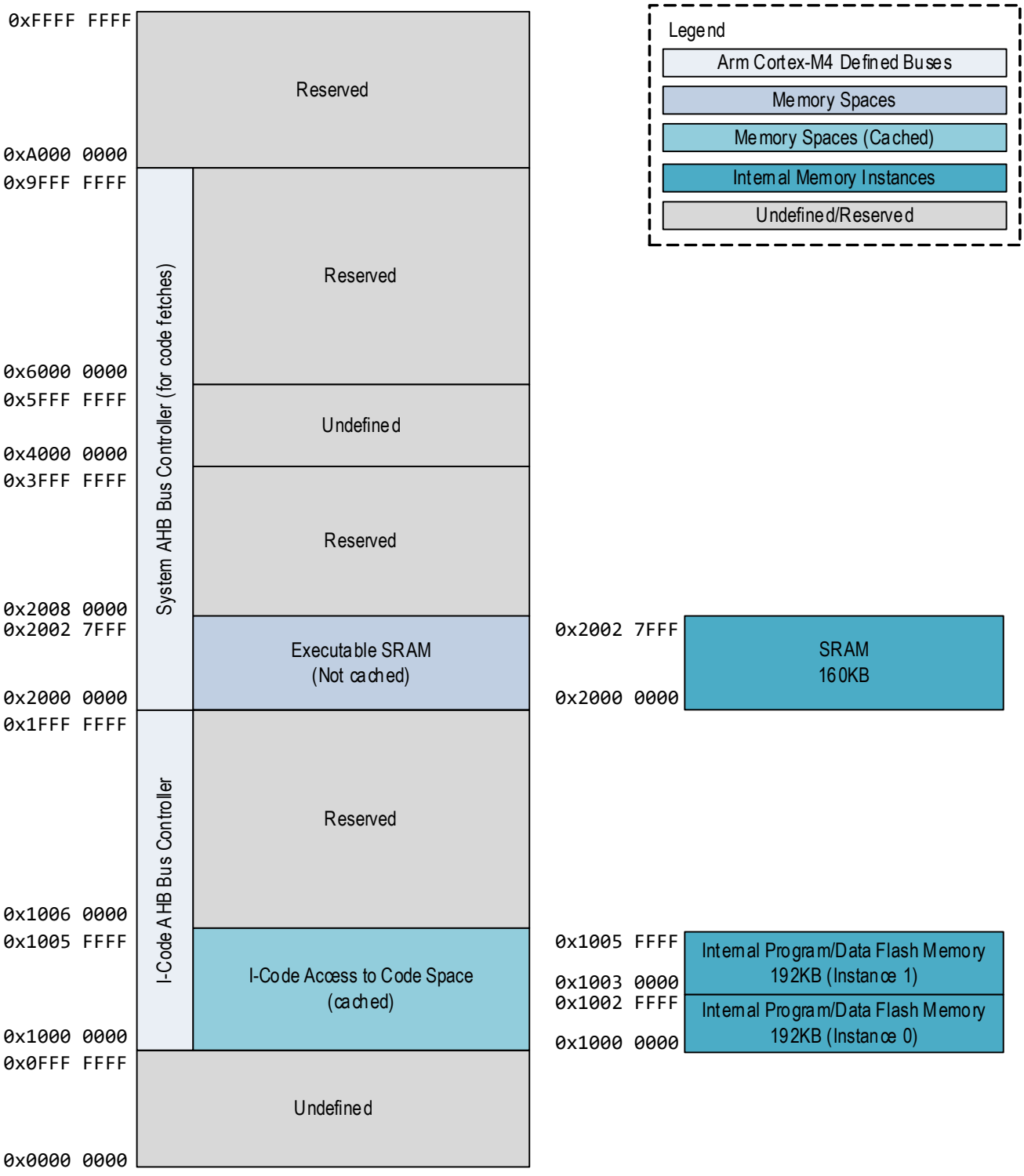
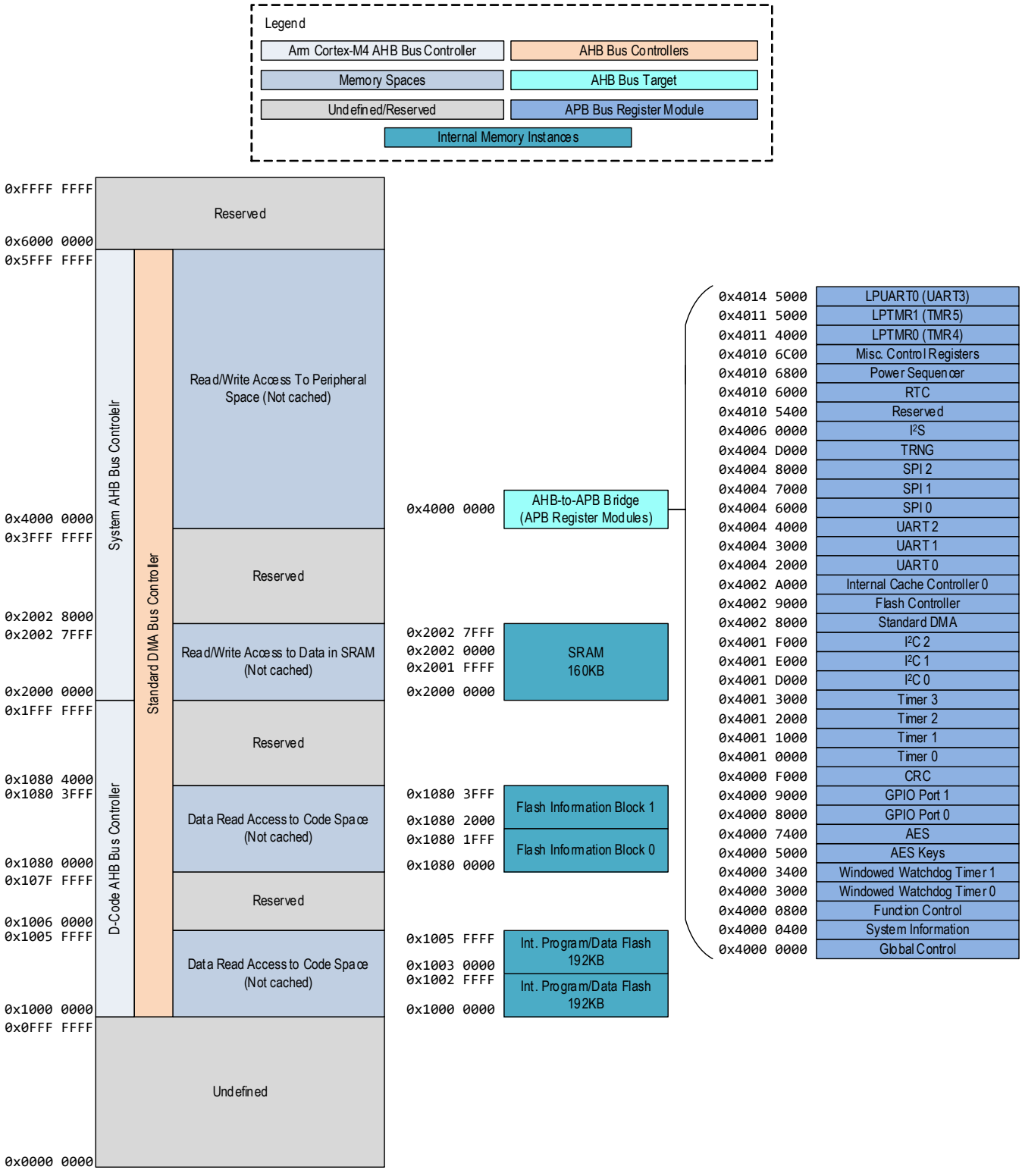


Figure 3-2: Data Memory Mapping



3.2 Device Memory Regions and Instances

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 MAX32670/MAX32671 must contain some code and data memory for software and variable/stack use, as well as certain components which are part of the instantiated core. This section details physical memory instances on the MAX32670/MAX32671 (including internal flash memory and SRAM instances) that are accessible as standalone memory regions using either the AHB or APB bus matrix. Memory areas which are only accessible through FIFO interfaces, or memory areas consisting of only a few registers for a specific peripheral, are not covered here.

3.2.1 Code Space

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

The MAX32670/MAX32671 code memory mapping is illustrated in *Figure 3-1*. The code space memory area contains the main internal flash memory, which holds most of the instruction code that will be executed on the device. The internal flash memory is mapped into both code and data space from 0x1000 0000 to 0x1005 FFFF. It is partitioned as two 192KB blocks of usable flash.

Note: The last page of flash (address 0x1005 E000 to 0x1005 FFFF) is reserved and cannot be used by software.

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 MAX32670/MAX32671 also contains the mapping for the flash information block, from 0x1080 0000 to 0x1080 3FFF. However, this mapping is generally only present during Analog Devices production test; it is disabled once the information block has been loaded with valid data and the info block lockout option has been set. This memory is accessible for data reads only and cannot be used for code execution. The flash information block is user read only accessible and contains the USN.

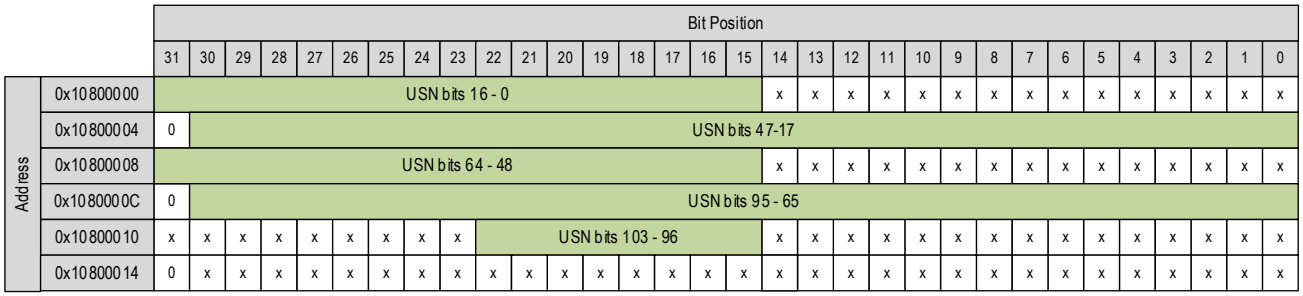
3.2.1 Instruction Cache Memory

This internal flash memory instruction cache controller (ICC) is 16,384 Bytes in size and is used to cache instructions fetched using the I-Code bus, including instructions fetched from the internal flash memory. This instruction cache controller is referred to as ICC throughout this document.

3.2.2 Information Block Flash Memory

The information block 0 is a separate flash instance of 16KB that is used to store trim settings (option configuration and analog trim) as well as other nonvolatile device-specific information. The information block 0 also contains the USN. The USN is a 104 bit field.

Figure 3-3: USN Format



Read the USN by reading from the addresses shown in *Figure 3-3*.

3.2.3 SRAM Space

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

The MAX32670/MAX32671 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 160KB.

The entirety of the SRAM memory space on the MAX32670/MAX32671 is contained within the dedicated Arm Cortex-M4 SRAM bit-banding region from 0x2000 0000 to 0x200F FFFF (1MB maximum for bit-banding). This means that the CPU can access the entire SRAM either using standard byte/word/doubleword access or using bit-banding operations. The bit-banding mechanism allows any single bit of any given SRAM byte address location to be set, cleared, or read individually by reading from or writing to a corresponding doubleword (32-bit wide) location in the bit-banding alias area.

The alias area for the SRAM bit-banding is located beginning at 0x2200 0000 and is a total of 32MB maximum, which allows the entire 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), and thus is only applicable to accesses generated by the core itself. Reads/writes to the bit-banding alias area by other (non-Arm-core) bus controllers does not trigger a bit-banding operation and will instead result in an AHB bus error.

The SRAM area on the MAX32670/MAX32671 can be used to contain executable code. Code stored in the SRAM is accessed directly for execution (using the system bus) and is not cached. The SRAM is also where the Arm Cortex-M4 stack must be located, as it is the only general-purpose SRAM memory on the device. A valid stack location inside the SRAM must be set by the system exception table (which is, by default, stored at the beginning of the internal flash memory).

The MAX32670/MAX32671 specific AHB bus controllers can access the SRAM to use as general storage or working space.

Table 3-1: SRAM Configuration

System RAM Block #	Size	Start Address	End Address
sysram0	16KB	0x2000 0000	0x2000 3FFF
sysram1	16KB	0x2000 4000	0x2000 7FFF
sysram2	32KB	0x2000 8000	0x2000 FFFF
sysram3	64KB	0x2001 0000	0x2001 FFFF
sysram4	4KB	0x2002 0000	0x2002 0FFF
sysram5	4KB	0x2002 1000	0x2002 1FFF
sysram6	8KB	0x2001 2000	0x2002 3FFF
sysram7	16KB	0x2002 4000	0x2002 7FFF

3.2.4 AES Key and Working Space Memory

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

3.2.5 Peripheral Space

The peripheral space area of memory is intended for mapping of control registers, internal buffers/working space, and other features needed for the software control of non-core peripherals. It is defined from byte address range 0x4000 0000 to 0x5FFF FFFF (0.5GB maximum). On the MAX32670/MAX32671, all device-specific module registers are mapped to this memory area, as well as any local memory buffers or FIFOs which are required by modules.

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

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

On the MAX32670/MAX32671, access to the region that contains most peripheral registers (0x4000 0000 to 0x400F FFFF) goes from the AHB bus through an AHB-to-APB bridge. This allows the peripheral modules to operate on the lower power APB bus matrix. This also ensures that peripherals with slower response times do not tie up bandwidth on the AHB bus, which must necessarily have a faster response time since it handles main application instruction and data fetching.

3.2.6 External RAM Space

The external RAM space area of memory is intended for use in mapping off-chip external memory and is defined from byte address range 0x6000 0000 to 0x9FFF FFFF (1GB maximum). The MAX32670/MAX32671 does not implement this memory area.

3.2.7 External Device Space

The external device space area of memory is intended for use in mapping off-chip device control functions onto the AHB bus. This memory space is defined from byte address range 0xA000 0000 to 0xDFFF FFFF (1GB maximum). The MAX32670/MAX32671 does not implement this memory area.

3.2.8 System Area (Private Peripheral Bus)

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

In addition to being restricted to the core, software is only allowed to 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 have access to this area.

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

3.2.9 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 MAX32670/MAX32671 does not implement this memory region.

3.3 AHB Interfaces

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

3.3.1 Core AHB Interfaces

3.3.1.1 I-Code

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

3.3.1.2 D-Code

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

3.3.1.3 System

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

3.3.2 AHB Controller

3.3.3 Standard DMA

The standard DMA bus controller 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
Watchdog Timer 0	WDT0_	0x4000 3000	0x4000 33FF
Watchdog Timer 1	WDT1_	0x4000 3400	0x4000 37FF
AES Keys	AESK_	0x4000 5000	0x4000 53FF
AES	AES_	0x4000 7400	0x4000 77FF
GPIO Port 0	GPIO0_	0x4000 8000	0x4000 8FFF
GPIO Port 1	GPIO1_	0x4000 9000	0x4000 9FFF
CRC	CRC_	0x4000 F000	0x4000 FFFF
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
I2C 0	I2C0_	0x4001 D000	0x4001 DFFF
I2C 1	I2C1_	0x4001 E000	0x4001 EFFF
I2C 2	I2C2_	0x4001 F000	0x4001 FFFF
Standard DMA	DMA_	0x4002 8000	0x4002 8FFF
Flash Controller 0	FLC0_	0x4002 9000	0x4002 93FF
Internal Cache Controller	ICC_	0x4002 A000	0x4002 A3FF

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
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
Real-Time Clock	RTC_	0x4010 6000	0x4101 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
UART 3 (Low Power UART 0)	UART3_	0x4014 5000	0x4014 5FFF

4. System, Power, Clocks, Reset

Different peripherals and subsystems support several possible 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 P0.12 AF4

4.1 Core Operating Voltage Range Selection

The MAX32670/MAX32671 supports three selections for the core operating voltage range (OVR). In 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 *FLC_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 access time of the internal flash memory, and the application software must set the flash wait states for each OVR setting as outlined in the section *Flash Wait States*. Changing the core operating voltage immediately reduces the output frequency of the IPO, 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	FLC_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 *FLC_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.
   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. Set the flash low voltage enable bit to 1 (*FLC_CTRL.lve*) to match the setting of *PWRSEQ_LPCN.ovr* since *PWRSEQ_LPCN.ovr* is not reset.

   Note: Setting the FLC_CTRL.lve to 1 should be done in the reset vector in RAM to ensure the low-voltage enable is set prior to 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

For devices that use the IPO as the default SYS_OSC (see section *Silicon Revision Differences* for details), 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* to move 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 *FLC_CTRL.lve* setting must be restored by software after any reset.

Important: *Flash reads may 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	FLC_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	FLC_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	FLC_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	FLC_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	FLC_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	FLC_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)

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

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

SYS_CLK drives the Arm Cortex-M4 with FPU core 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)

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

Note: GCR_PCLKDIV.aon_clkdiv is selectable from 0 to 3 for divisors of 1, 2, 4, or 8.

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 MAX32670/MAX32671. *Table 4-9* details the effect each reset source has on the system clock selection and the system clock prescaler settings.

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. The software should understand that all peripherals should be disabled before switching SYS_OSC or touching the SYS_OSC prescaler.

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

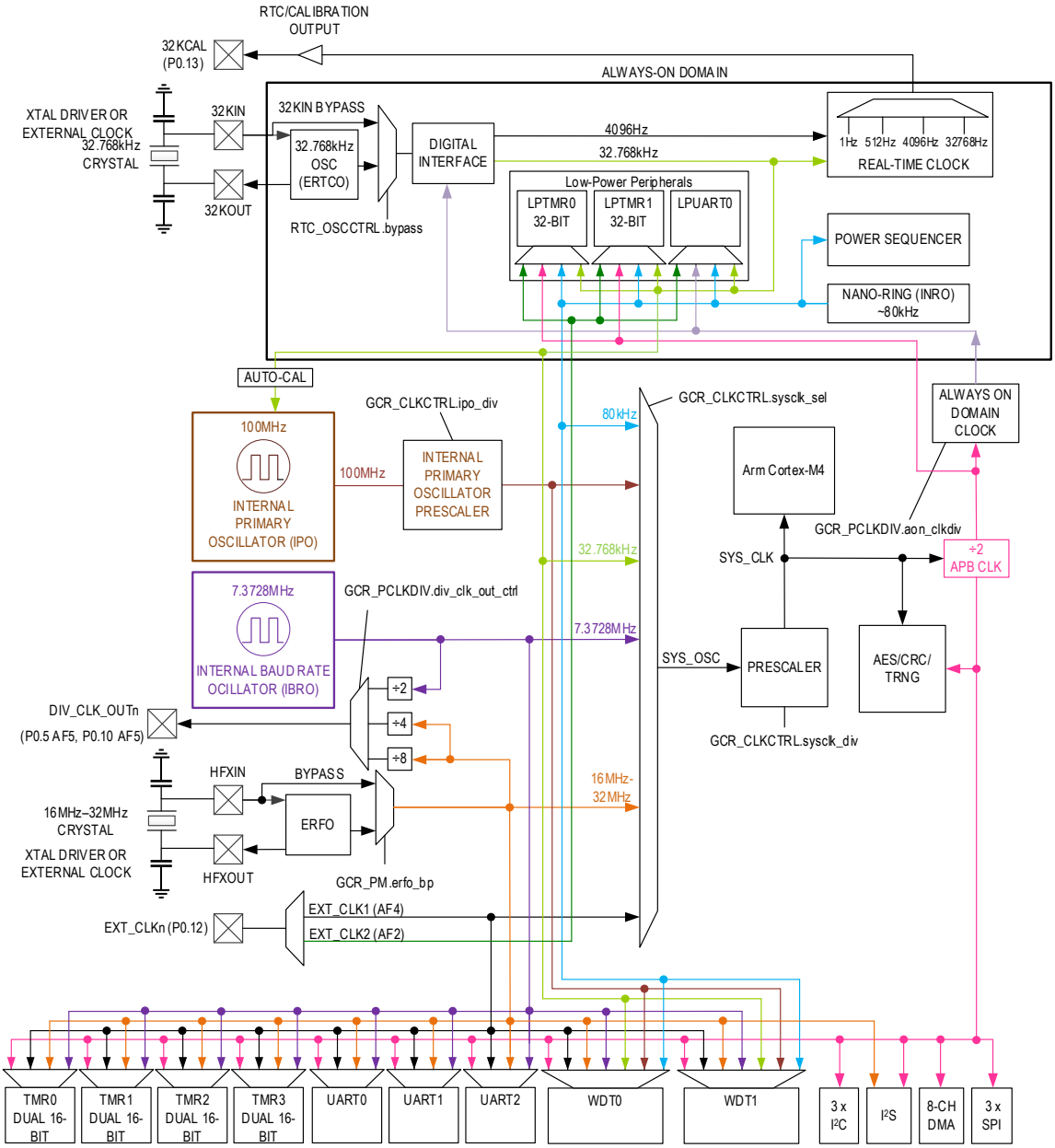
	Reset Source				
Oscillator	**POR**	**System**	**Watchdog**	**Soft**	**Peripheral**
IPO	See section *Silicon Revision Differences*			Retains State	Retains State
IBRO	See section *Silicon Revision Differences*			Retains State	Retains State
INRO	Enabled	Enabled	Enabled	Enabled	Enabled
ERTCO	Disabled	Retains 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*	Default system oscillator. See *Silicon Revision Differences* for the default oscillator by revision.			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 MAX32670/MAX32671 clock tree.

Figure 4-1: MAX32670/MAX32671 Clock Block Diagram



4.2.1 Oscillator Implementation

Following a POR or a system reset, the SYS_OSC defaults to the clock source as defined in the section *Silicon Revision Differences,* 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 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 a system oscillator 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. Before entering any low-power mode, the software must enable any oscillator needed during the low-power mode.

4.2.2 100MHz Internal Primary Oscillator (IPO)

This oscillator can be selected as SYS_OSC. The 100MHz 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 Radio Frequency Oscillator (ERFO)

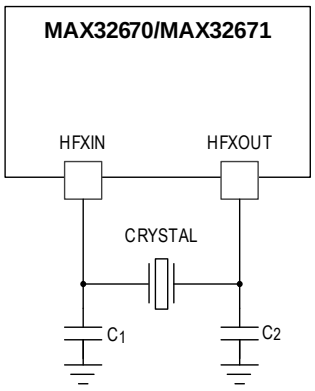
This oscillator can be selected as SYS_OSC. It is important to use the correct capacitor values on the PCB when connecting the crystal as described in *Calculating the Crystal Load Capacitor*. This oscillator is disabled by default at power-up.

Follow the steps below to use the ERFO as the system oscillator.

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

4.2.3.1 Calculating the Crystal Load Capacitor

Figure 4-2: ERFO Load Capacitors



Equation 4-5: Load Capacitance Calculation

$$C_L = \frac{C1 \times C2}{(C1 + C2)} + C_{STRAY}$$

$where$:

$C_L = the\ crystal\ capacitance$

$C_{STRAY} = the\ capacitance\ of\ the\ pins\ and\ the\ parasitics\ of\ the\ board.$

Calculate the values of C_1 and C_2 using the following steps:

1. The crystal load, C_L, as specified in the device data sheet electrical characteristics table, must be 12pF. See the spec External RF Oscillator in the data sheet. Therefore, the total capacitance seen by the crystal must equal C_L.
2. Assume $C_1 = C_2$, *Equation 4-5* can be rewritten as:
   $C_1 = C_2 = 2 \times (C_L - C_{STRAY})$
3. The device pin capacitance of the HFXOUT and HFXIN pins, respectively is 4pF each as given in the device data sheet parameter C_{IO}. Assume the circuit board stray capacitance is 0.5pF, resulting in C_{STRAY} = 4.5pF.
4. Solve for C_1 and C_2:
   $C_1 = C_2 = 2 \times (12pF - 4.5pF) = 15pF$

Checking the clock frequency accuracy on each new board design using a frequency counter is recommended. Measure the output frequency by toggling a GPIO pin with the ERFO set as the system oscillator. Adjust the load capacitance as required to adjust the crystal frequency.

4.2.4 7.3728MHz Internal Baud Rate Oscillator (IBRO)

The IBRO is a low-power internal oscillator that can be selected as the system oscillator. Some devices default to the IBRO as the system oscillator after reset. See *Silicon Revision Differences* for details. This clock can optionally be used as a dedicated baud rate clock for the UARTs. This is useful if the SYS_OSC selected does not allow the targeted UART baud rate.

The *GCR_CLKCTRL.ibro_vs* field controls the voltage source for the IBRO. The internal CPU 1V LDO core supply voltage is the default option. The V_{CORE} external pin 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 the system oscillator. 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 clock source for the RTC. If the RTC is enabled, the ERTCO must be enabled. This oscillator is disabled at power-up.

The ERTCO is disabled by a POR. 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 the *PWRSEQ_LPCN.ertco_pd* field to 0.
2. Enable the ERTCO by setting the *GCR_CLKCTRL.ertco_en* field to 1.
3. Wait until the *GCR_CLKCTRL.ertco_rdy* field reads 1.
   a. The ERTCO is now operating.
4 If setting the ERTCO as the system oscillator, set *GCR_CLKCTRL.sysclk_sel* = 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.

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

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

The INRO 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: Wake-up 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 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

This 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 remain on unless explicitly disabled before entering *SLEEP*.

4.3.2.1 Entering SLEEP

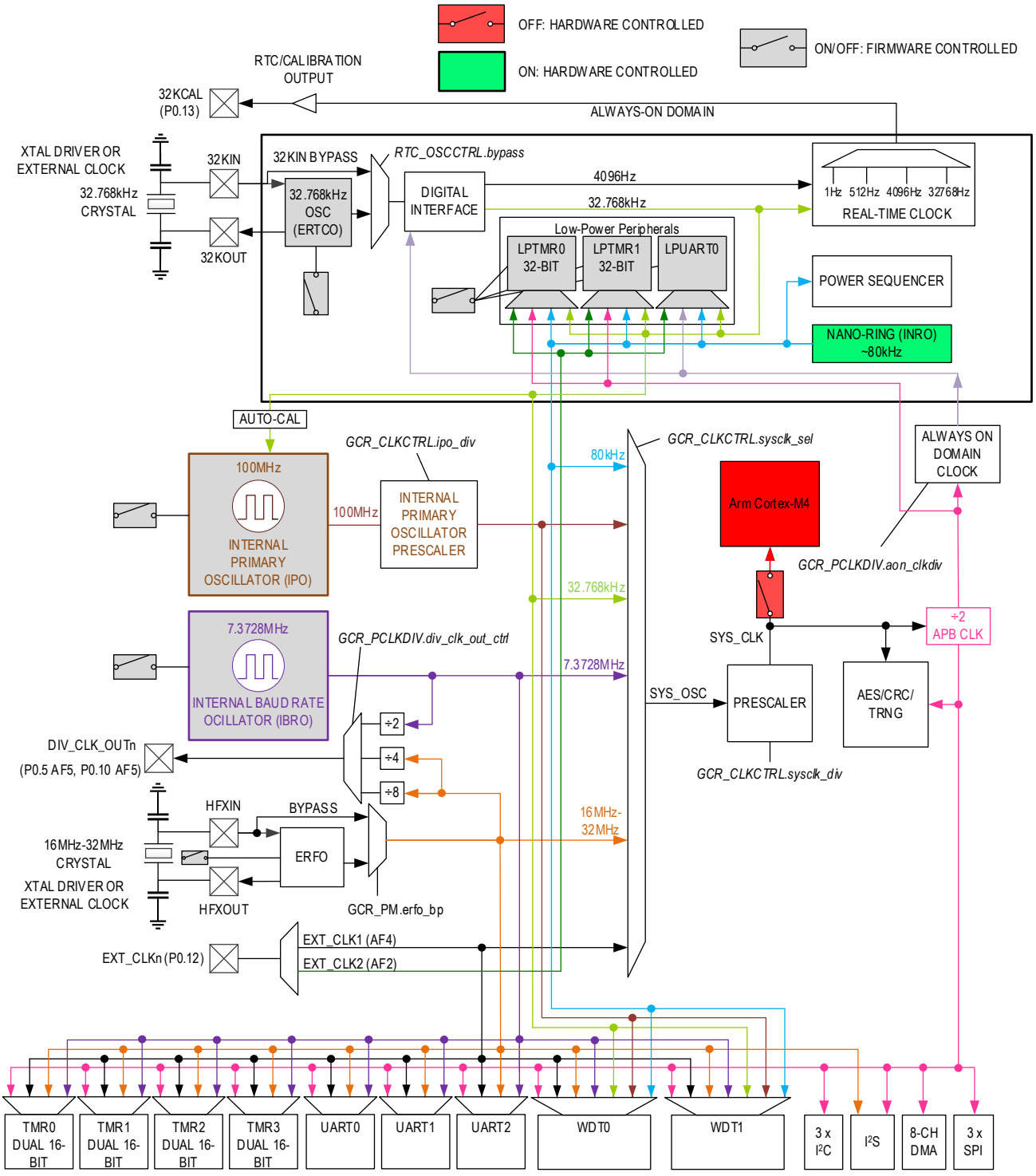
Place the CM4 in *SLEEP* by performing the following steps:

1. Configure any desired wake-up functions. See *Table 4-10* for possible wake-up sources.
2. Clear the wake-up status register by writing 0xFFFF FFFF to the *PWRSEQ_LPWKST0* register.
3. Clear the low-power peripheral wake-up status register by writing 0xFFFF FFFF 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.

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

Table 4-13 and *Table 4-15* 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: MAX32671/MAX32670 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 main bus clocks, PCLK and HCLK, are inactive. The CPU state is retained. The ERTCO can be enabled by 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 main bus clocks PCLK and HCLK are gated off, the clock source choice is limited. See *Low-Power Receiver Operation* for LPUART configuration and see *Wakeup Events* for LPTMR configuration.

The RTC, which has its own 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 4-13 and *Table 4-15* 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 register by writing 0xFFFF FFFF to each of the *PWRSEQ_LPWKST0* register.
3. Clear the low-power peripheral wake-up flags and the backup wake-up status flag by writing 0xFFFF FFFF 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.

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

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 *sysram7* can be independently configured to be state retained, as shown in *Table 4-11*.
- The low-power peripherals (LPTMR0, LPTMR1, and LPUART0) can be configured for operation and used as wake-up sources.
- 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_GP0* and *PWRSEQ_GP1* registers.
- The following oscillators are powered down:
  - IPO
  - ISO
  - IBRO
  - ERFO
- INRO is on.
- The ERTCO is software controlled.

This mode places the CPU in a static, low-power state. SYS_OSC is gated off, so HCLK, PCLK and AOD_CLK 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 main bus clocks PCLK and HCLK are gated off, the clock source choice is limited. See *Low-Power Receiver Operation* for LPUART configuration and see *Wakeup Events* for LPTMR configuration.

Table 4-11: RAM Retention By Address Range in BACKUP, System Reset, Watchdog Reset, and External Reset

System RAM Block	RAM Retention Enable Field	Address Range Retention	Amount of RAM Retained
sysram0	*PWRSEQ_LPCN.ram0ret_en*	N/A	0KB
sysram1	*PWRSEQ_LPCN.ram1ret_en*	0x2000 4000 – 0x2000 7FFF	16KB
sysram2	*PWRSEQ_LPCN.ram2ret_en*	0x2000 8000 – 0x2000 FFFF	32KB
sysram3	*PWRSEQ_LPCN.ram3ret_en*	0x2001 0000 – 0x2001 FFFF	64KB
sysram4	*PWRSEQ_LPCN.ram0ret_en*	0x2002 0000 – 0x2002 0FFF	4KB
sysram5	*PWRSEQ_LPCN.ram1ret_en*	0x2002 1000 – 0x2002 1FFF	4KB
sysram6	*PWRSEQ_LPCN.ram2ret_en*	0x2001 2000 – 0x2002 3FFF	8KB
sysram7	*PWRSEQ_LPCN.ram2ret_en*	0x2002 4000 – 0x2002 7FFF	16KB

The RTC has its own 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 may be optionally retained. The amount of RAM retained is controlled by setting the *PWRSEQ_LPCN.ram0ret_en, PWRSEQ_LPCN.ram1ret_en, PWRSEQ_LPCN.ram2ret_en*, and *PWRSEQ_LPCN.ram3ret_en* fields.

The boot ROM uses *sysram0* during a system reset, watchdog timer reset, an external reset, and an exit from *BACKUP*. The boot ROM uses this RAM to perform system checks and to determine if a bootloader activation pin is asserted. As a result, *sysram0* cannot be retained during an exit from *BACKUP*.

Table 4-13 and *Table 4-15* show the effects that *BACKUP* has on the various clock sources. *Figure 4-4* shows the clock control during *BACKUP*.

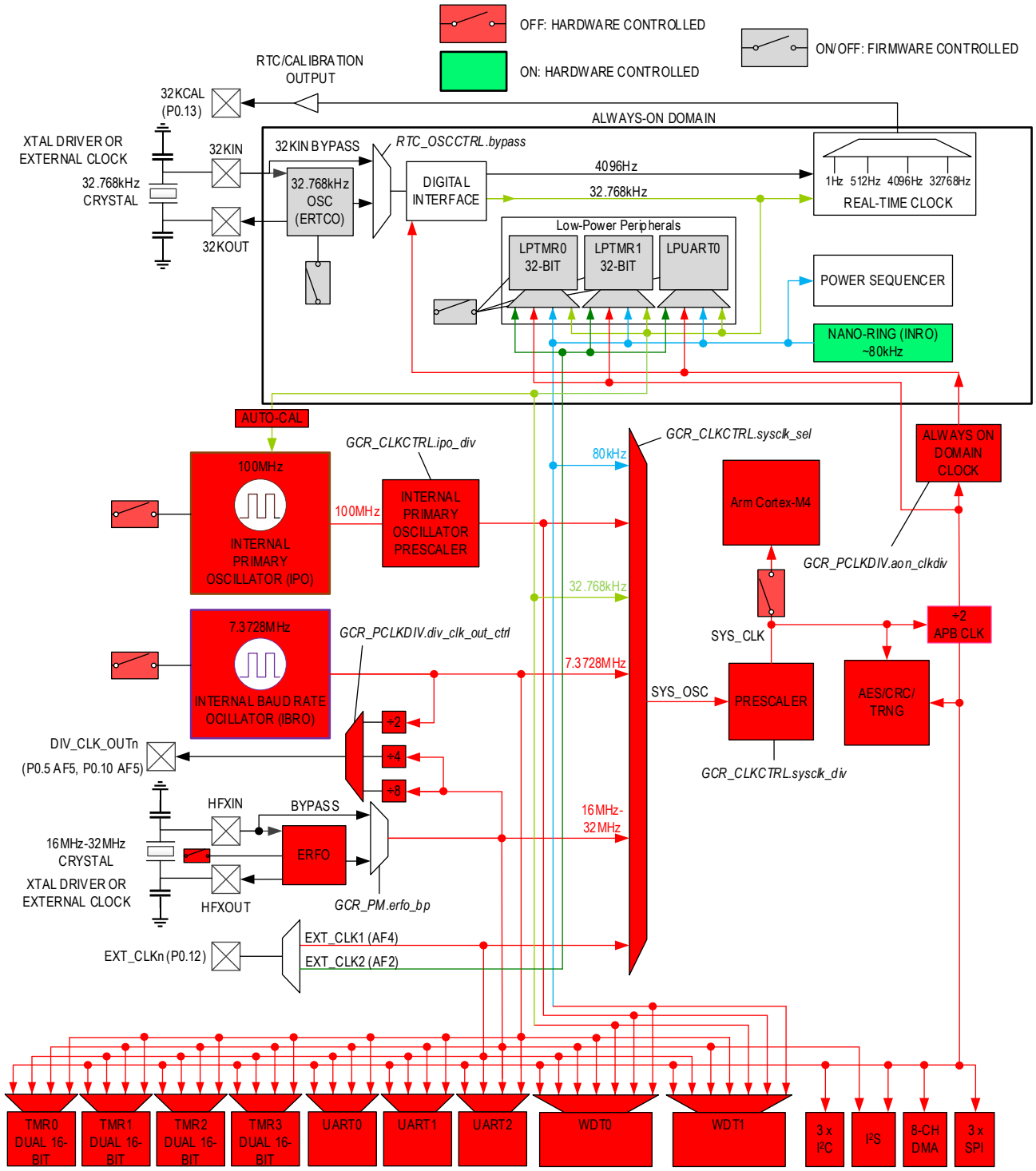
4.3.4.1 Entering BACKUP

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 4-11* for details.
3. Clear the wake-up status register by writing 0xFFFF FFFF to the *PWRSEQ_LPWKST0* register.
4. Clear the low-power peripheral wake-up flags and the backup wake-up status flag by writing 0xFFFF FFFF 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.

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

Figure 4-4: MAX32670/MAX32671 DEEPSLEEP and BACKUP Clock Control



4.3.5 STORAGE

This mode is like *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*.

The ERTCO remains enabled in *STORAGE* if it was enabled before entering *STORAGE*.

Table 4-13 and *Table 4-15* show the effects that *STORAGE* has on the various clock sources.

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

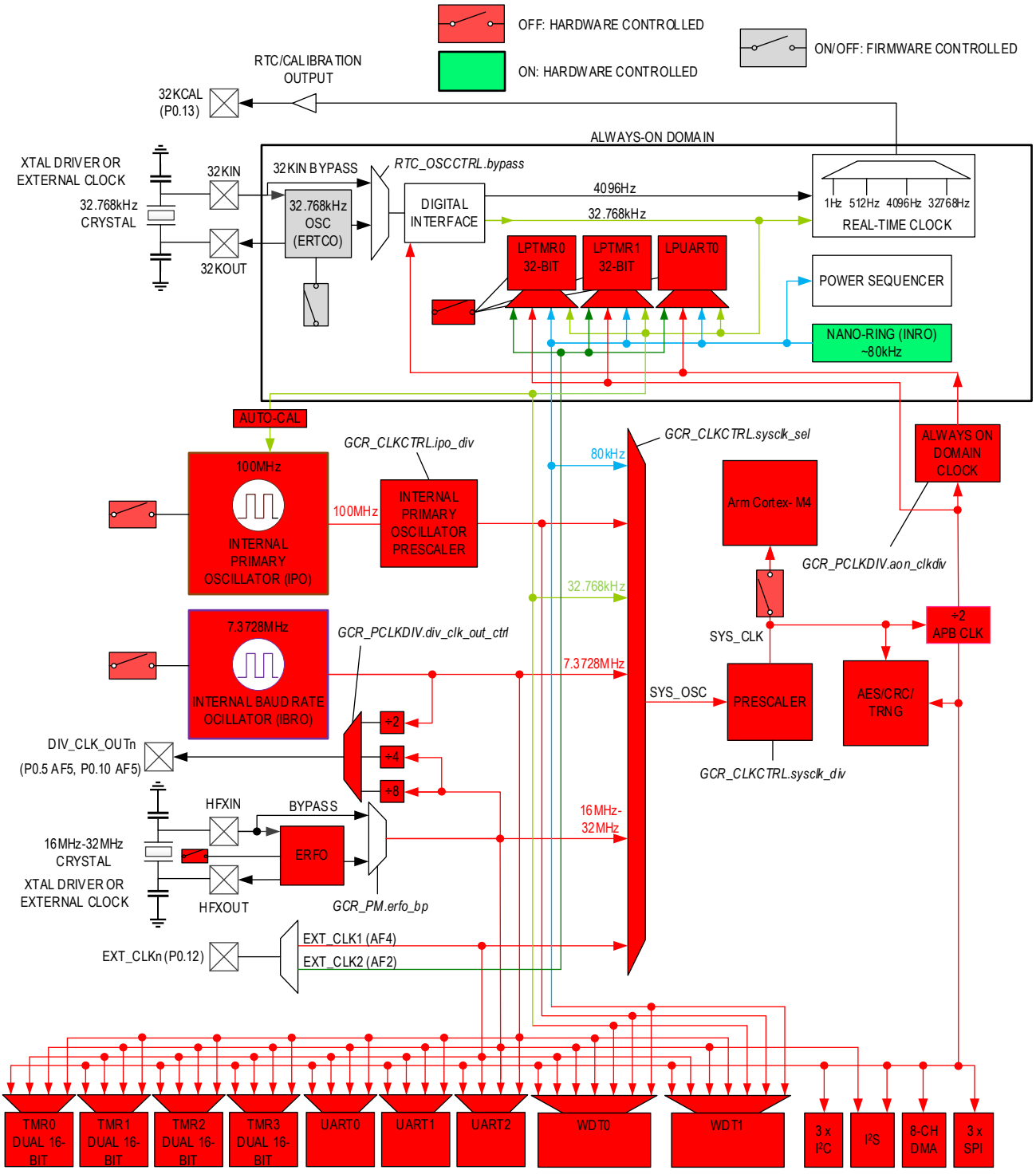
4.3.5.1 Entering STORAGE

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 the *PWRSEQ_LPWKST0* register.
3. Clear the low-power peripheral wake-up flags and the backup wake-up status flag by writing 0xFFFF FFFF 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.

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

Figure 4-5: MAX32670/MAX32671 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 wake-up sources, wake-up logic, and interrupts are disabled. The device only exits this state through a POR, which reinitializes the device.

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

4.5 Device Resets

Four device resets are available:

- Peripheral Reset
- Soft Reset
- System Reset (includes external and watchdog reset)
- POR

On completion of any of the four resets, all peripherals are reset. On completion of any reset cycle, HCLK and PCLK are operational, the CPU core receives clocks and power, and the device is in *ACTIVE*. Program execution begins at the reset vector address.

Contents of the AoD are reset only upon power cycling V_{DD} and V_{CORE}.

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

Table 4-12, *Table 4-13*, *Table 4-14*, and *Table 4-15* show the effects on the system of the four reset types and the five power modes.

Table 4-12: MAX32670/MAX32671 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	-	-	See section *Silicon Revision Differences*	
ERFO	-	-	Off	Off
IPO	-	-	See section *Silicon Revision Differences*	
SYS_CLK	On	On	On	On
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: A system reset occurs when exiting *BACKUP* or *STORAGE* low-power mode. The system reset does not affect the low-power peripherals.
3: Peripheral, soft, and system resets are initiated by software though the *GCR_RST0* register. A system reset is also triggered by the RSTN device pin and watchdog reset.

Table 4-13: MAX32670/MAX32671 Clock Source and Global Control Register Low-Power Mode Effects

	ACTIVE	*SLEEP*	*DEEPSLEEP*	*BACKUP*	*STORAGE*
GCR	R	-	-	-	-
INRO	On	On	On	On	On
ERTCO	SW	SW	SW	SW	SW
IBRO	R	-	Off	Off	Off
ERFO	R	-	Off	Off	Off
IPO	R	-	Off	Off	Off
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: A system reset occurs when exiting *BACKUP* or *STORAGE* low-power mode. The system reset does not affect the low-power peripherals.
3: Peripheral, soft, and system resets are initiated by software though the *GCR_RST0* register. System reset can also be triggered by the RSTN device pin or watchdog reset.

Table 4-14: MAX32670/MAX32671 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: A system reset occurs when exiting *BACKUP* or *STORAGE* low-power mode. The system reset does not affect the low-power peripherals.
3: Peripheral, soft, and system resets are initiated by software though the *GCR_RST0* register. System reset can also be triggered by the RSTN device pin or watchdog reset.

Table 4-15: MAX32670/MAX32671 Peripheral and CPU Low-Power Mode Effects

	ACTIVE	*SLEEP*	*DEEPSLEEP*	*BACKUP*	*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: A system reset occurs when exiting *BACKUP* or *STORAGE* low-power mode. The system reset does not affect the low-power peripherals.
3: Peripheral, soft, and system resets are initiated by software though 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 4-12* and *Table 4-14*, peripheral reset performs a reset for all peripherals. The CPU retains its state. The GPIO, watchdog timers, AoD, RAM retention, and general control registers (GCR), including the clock configuration, are unaffected.

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

4.5.2 Soft Reset

As shown in *Table 4-12* and *Table 4-14*, a soft reset is the same as a peripheral reset except that it also resets the GPIO to the POR state.

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

4.5.3 System Reset

As shown in *Table 4-12* and *Table 4-14*, a system reset is the same as a soft reset, except it also resets all GCR, resetting the clocks to their default state. The CPU state is reset as well as the watchdog timers. The AoD and RAM are unaffected.

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

4.5.4 Power-On Reset (POR)

As shown in *Table 4-12* and *Table 4-14*, a POR resets everything in the device to its default state.

4.6 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. Invalidate the flash by writing 1 to the *ICC_INVALIDATE* register.
2. Set *ICC_CTRL.en* to 1.
3. Read *ICC_CTRL.rdy* until it returns 1.

4.6.2 Disabling ICC

Disable 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 cache. Setting *GCR_SYSCTRL.icc0_flush* 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 4-16: Internal Cache Controller Register Summary

Offset	Register	Description
[0x0000]	*ICC_INFO*	Cache ID Register

Offset	Register	Description
[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 4-17: 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** Returns the ID for this cache instance.
9:6	partnum	R	*	**Cache Part Number** Returns the part number indicator for this cache instance.
5:0	relnum	R	*	**Cache Release Number** Returns the release number for this cache instance.

Table 4-18: ICC Memory Size Register

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

Table 4-19: ICC Cache Control Register

ICC Cache Control			ICC_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	-	**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	-	RO	-	**Reserved**

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

Table 4-20: 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 light sleep. See the *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 RAM memories can be zeroized:

- Internal System RAM
  - The entire system RAM can be zeroized by setting the *GCR_MEMZ.ram* field to 1.
- 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 in *Table 3-1*.

4.8.3.1 RAM LIGHTSLEEP

RAM can be placed in a low-power mode, referred to as LIGHTSLEEP, using the memory lock 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, thus reducing power consumption. LIGHTSLEEP is available for the four data RAM blocks and the ICC RAM.

4.8.3.2 RAM Shutdown

RAM can individually be shut down thus further reducing the power consumption for the device. Shutting down a memory gates off the clock and removes power to the memory. Shutting down a memory invalidates (destroys) the contents of the memory and results in a POR of the memory when it is enabled. RAM memory 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 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 4-21: Miscellaneous Control Register Summary

Offset	Register	Description
[0x0004]	*MCR_RST*	Reset Control Register
[0x0010]	*MCR_LPPIOCTRL*	Low-Power Peripheral Control Register
[0x0024]	*MCR_CLKDIS*	Clock Disable Register

4.9.1 Registers Details

Table 4-22: Reset Control Register

Reset Control			MCR_RST	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
3	rtc	R/W1O		**RTC Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. *See the Device Resets section for additional information.*
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 4-23: Low-Power Peripheral Control Register

Low-Power Peripheral Control			MCR_LPPIOCTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7	lpuart0_rts	R/W	0	**LPUART0 RTS Enable** If the LPUART0 peripheral clock is enabled (*MCR_CLKDIS.lpuart0* = 0) and this field is set to 1, the peripheral controls the associated GPIO, otherwise the associated GPIO is controlled using the GPIO registers.
6	lpuart0_cts	R/W	0	**LPUART0 CTS Enable** If the LPUART0 peripheral clock is enabled (*MCR_CLKDIS.lpuart0* = 0) and this field is set to 1, the peripheral controls the associated GPIO, otherwise the associated GPIO is controlled using the GPIO registers.

Low-Power Peripheral Control			MCR_LPPIOCTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
5	lpuart0_tx	R/W	0	**LPUART0 Transmit Enable** If the LPUART0 peripheral clock is enabled (*MCR_CLKDIS.lpuart0* = 0) and this field is set to 1, the peripheral controls the associated GPIO, otherwise the associated GPIO is controlled using the GPIO registers.
4	lpuart0_rx	R/W	0	**LPUART0 Receive Enable** If the LPUART0 peripheral clock is enabled (*MCR_CLKDIS.lpuart0* = 0) and this field is set to 1, the peripheral controls the associated GPIO, otherwise the associated GPIO is controlled using the GPIO registers.
3	lptmr1_o	R/W	0	**LPTMR1 Output Enable** If the LPTMR1 peripheral clock is enabled (*MCR_CLKDIS.lptmr1* = 0) and this field is set to 1, the peripheral controls the associated GPIO, otherwise the associated GPIO is controlled using the GPIO registers.
2	lptmr1_i	R/W	0	**LPTMR1 Input Enable** If the LPTMR1 peripheral clock is enabled (*MCR_CLKDIS.lptmr1* = 0) and this field is set to 1, the peripheral controls the associated GPIO, otherwise the associated GPIO is controlled using the GPIO registers.
1	lptmr0_o	R/W	0	**LPTMR0 Output Enable** If the LPTMR0 peripheral clock is enabled (*MCR_CLKDIS.lptmr1* = 0) and this field is set to 1, the peripheral controls the associated GPIO, otherwise the associated GPIO is controlled using the GPIO registers.
0	lptmr0_i	R/W	0	**LPTMR0 Input Enable** If the LPTMR0 peripheral clock is enabled (*MCR_CLKDIS.lptmr1* = 0) and this field is set to 1, the peripheral controls the associated GPIO, otherwise the associated GPIO is controlled using the GPIO registers.

Table 4-24: Clock Disable Register

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

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. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

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

Table 4-25: Power Sequencer and Always-On Domain Register Summary

Offset	Register	Description
[0x0000]	*PWRSEQ_LPCN*	Low-Power Control Register
[0x0004]	*PWRSEQ_LPWKST0*	GPIO0 Low-Power Wake-up Status Flags
[0x0008]	*PWRSEQ_LPWKEN0*	GPIO0 Low-Power Wake-up Enable Register
[0x000C]	*PWRSEQ_LPWKST1*	GPIO1 Low-Power Wake-up Status Flags
[0x0010]	*PWRSEQ_LPWKEN1*	GPIO1 Low-Power Wake-up Enable Register
[0x0030]	*PWRSEQ_LPPWKST*	Peripheral Low-Power Wake-up Status Flags
[0x0034]	*PWRSEQ_LPPWKEN*	Peripheral Low-Power Wake-up Enable Register
[0x0040]	*PWRSEQ_LPMEMSD*	RAM Shutdown Control Register
[0x0048]	*PWRSEQ_GP0*	General Purpose 0 Register
[0x004C]	*PWRSEQ_GP1*	General Purpose 1 Register

4.10.1 Register Details

Table 4-26: Low-Power Control Register

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31	ertco_pd	R/W	1	**ERTCO Oscillator Power Down** Set this field to 0 to power on the 32KHz oscillator circuitry. 0: 32KHz oscillator powered on. 1: 32KHz oscillator powered down.
30	-	RO	0	**Reserved.**
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. *Note: If PWRSEQ_LPCN.storage_en is 1 this field is ignored and INRO is powered off.*
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.*
27:26	-	RO	0	**Reserved**

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
25	porvddmon_dis	R/W	0	**V_{DDIO} 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**
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. Setting this field to 1 also disables the IBRO. 0: Enabled. 1: Disabled. *Note: This field cannot be modified in devices with IBRO as the default oscillator.*
19:18	-	RO	0	**Reserved**
17	vcore_ext	R/W	0	**V_{CORE} 1V Supply** Setting this bit allows the V_{CORE} device pin to be used as the 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* modes.
11	bg_dis	R/W	1	**Bandgap Disable for *DEEPSLEEP*** Setting this field to 1 turns off the bandgap during *DEEPSLEEP*. 0: Enabled. 1: Disabled.

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
10	fastwk_en	R/W	0	**Fast Wake-up Enable for *DEEPSLEEP*** Set to 1 to enable fast wake-up 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. • Resuming code execution at the next instruction after the *DEEPSLEEP* entry. When this field is 0, the device exits *DEEPSLEEP* as if a system reset occurred. Software executes from the reset vector. 0: Disabled. 1: Enabled.
9	storage_en	R/W	0	***STORAGE* 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	-	RO	0	**Reserved**
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. Bypassing the hardware detection of an external supply on V_{CORE} enables a faster wake-up time. 0: Enabled. 1: Disabled.

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
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 the 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:** 0b00: V_{CORE} = 0.9V, f_{IPO} = 12MHz. 0b01: V_{CORE} = 1.0V, f_{IPO} = 50MHz. 0b10: V_{CORE} = 1.1V, f_{IPO} = 100MHz. 0b11: Reserved. **Single-Supply Operation (V_{CORE} = GND)** 0b00: V_{LDO} = 0.9V, f_{IPO} = 12MHz. 0b01: V_{LDO} = 1.0V, f_{IPO} = 50MHz. 0b10: V_{LDO} = 1.1V, f_{IPO} = 100MHz. 0b11: Reserved.
3	ram3ret_en	R/W	0	***Sysram3* and *Sysram7* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram3* and *sysram7*. See *Table 3-1* for system RAM configuration. 0: Disabled. 1: Enabled. *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.*
2	ram2ret_en	R/W	0	***Sysram2* and *Sysram6* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram2* and *sysram6*. See *Table 3-1* for system RAM configuration. 0: Disabled. 1: Enabled. *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.*
1	ram1ret_en	R/W	0	***Sysram1* and *Sysram5* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram1* and *sysram5*. See *Table 3-1* for system RAM configuration. 0: Disabled. 1: Enabled. *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.*
0	ram0ret_en	R/W	0	***Sysram0* and *Sysram4* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram0* and *sysram4*. See *Table 3-1* for system RAM configuration. 0: Disabled. 1: Enabled. *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.* *Note: Sysram0 is used by the bootloader on exit from BACKUP and is not retained.*

Table 4-27: GPIO0 Low-Power Wake-up Status Flags

GPIO0 Low-Power Wake-up Status Flags			PWRSEQ_LPWKST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	st	R/W1C	0	**GPIO0 Pin Wake-up Status Flag** Whenever a GPIO0 pin, in any low-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_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 4-28: GPIO0 Low-Power Wake-up Enable Registers

GPIO0 Low-Power Wake-up Enable			PWRSEQ_LPWKEN0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	en	R/W	0	**GPIO0 Pin Wake-up 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 Wake-up 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 4-29: GPIO1 Low-Power Wake-up Status Flags

GPIO1 Low-Power Wake-up Status Flags			PWRSEQ_LPWKST1	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	en	R/W	0	**GPIO1 Pin Wake-up Status Flag** Whenever a GPIO1 pin, in any low-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 4-30: GPIO1 Low-Power Wake-up Enable Registers

GPIO1 Low-Power Wake-up Enable			PWRSEQ_LPWKEN1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0		DNM	0	**GPIO1 Pin Wake-up 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 Wake-up 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 4-31: Peripheral Low-Power Wake-up Status Flags

Peripheral Low-Power Wake-up Status Flags			PWRSEQ_LPPWKST	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	lpuart0	R/W1C	0	**LPUART0 Wake-up Flag** This field is set when this peripheral causes the CPU to wake to *ACTIVE*. 0: Normal operation. 1: Wake-up event detected. *Note: If the corresponding bit in PWRSEQ_LPPWKEN register is set, the event generates an interrupt to wake up the device from a low-power mode.*
1	lptmr1	R/W1C	0	**LPTMR1 Wake-up Flag** This field is set when this peripheral causes the CPU to wake to *ACTIVE*. 0: Normal operation. 1: Wake-up event detected. *Note: If the corresponding bit in PWRSEQ_LPPWKEN register is set, the event generates an interrupt to wake up the device from a low-power mode.*
0	lptmr0	R/W1C	0	**LPTMR0 Wake-up Flag** This field is set when this peripheral causes the CPU to wake to *ACTIVE*. 0: Normal operation. 1: Wake-up event detected. *Note: If the corresponding bit in PWRSEQ_LPPWKEN register is set, the event generates an interrupt to wake up the device from a low-power mode.*

Table 4-32: Peripheral Low-Power Wake-up Enable Register

Peripheral Low-Power Wake-up Enable			PWRSEQ_LPPWKEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	lpuart0	R/W	0	**LPUART0 Wake-up Enable** Setting this bit enables an interrupt and wake-up the device from any low-power mode when *PWRSEQ_LPPWKST.lpuart0* does not equal 0. 0: Disabled. 1: Enabled.

Peripheral Low-Power Wake-up Enable			PWRSEQ_LPPWKEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
1	lptmr1	R/W		**LPTMR1 Wake-up Enable** Setting this bit enables an interrupt and wake-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 Wake-up Enable** Setting this bit enables an interrupt and wake-up the device from any low-power mode when *PWRSEQ_LPPWKST.lptmr0* does not equal 0. 0: Disabled. 1: Enabled.

Table 4-33: 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** Set this field to 1 to shut down the power on the specified RAMs. Powering down the memory destroys its contents. See *Table 3-1* for system RAM configuration. 0: Enabled. 1: Shut down. *Note: See the GCR_MEMCTRL register for retention mode power settings.*
2	ram2	R/W	0	***Sysram2* and *Sysram6* Shut Down** Set this field to 1 to shut down the power on the specified RAMs. Powering down the memory destroys its contents. See *Table 3-1* for system RAM configuration. 0: Enabled. 1: Shut down. *Note: See the GCR_MEMCTRL register for retention mode power settings.*
1	ram1	R/W	0	***Sysram1* and *Sysram5* Shut Down** Set this field to 1 to shut down the power on the specified RAMs. Powering down the memory destroys its contents. See *Table 3-1* for system RAM configuration. 0: Enabled. 1: Shut down. *Note: See the GCR_MEMCTRL register for retention mode power settings.*
0	ram0	R/W	0	***Sysram0* and *Sysram4* Shut Down** Set this field to 1 to shut down the power on the specified RAMs. Powering down the memory destroys its contents. See *Table 3-1* for system RAM configuration. 0: Enabled. 1: Shut down. *Note: See the GCR_MEMCTRL register for retention mode power settings.*

Table 4-34: General Purpose 0 Register

General Purpose 0			PWRSEQ_GP0	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**General Purpose Field** The software can use this register as a general-purpose register, and the contents are retained during *SLEEP*, *DEEPSLEEP*, and *BACKUP*.

Table 4-35: General Purpose 1 Register

General Purpose 1			PWRSEQ_GP1	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**General Purpose Field** The software can use this register as a general-purpose register, and the contents are retained during *SLEEP*, *DEEPSLEEP*, and *BACKUP*.

4.11 Global Control Registers (GCR)

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

Note: The GCR are only reset on a system reset or POR. A soft reset and peripheral reset do not affect these registers.

Table 4-36: 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*	Reserved
[0x0068]	*GCR_ECCCED*	Reserved
[0x006C]	*GCR_ECCIE*	Reserved
[0x0070]	*GCR_ECCADDR*	Reserved

4.11.1 Register Details

Table 4-37: 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 bit is only valid after the ROM checksum is complete and hardware sets *GCR_SYSCTRL.cchk* to 0. 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 GCR_SYSST.icelock field is not set.*
13	cchk	R/W	0	**Calculate ROM Checksum** This bit is self-clearing when the ROM checksum calculation is complete, and the result is available at bit *GCR_SYSCTRL.chkres*. Writing a 0 has no effect. 0: No operation. 1: Start ROM checksum calculation.
12	romdone	R	1	**ROM Start Code Status** Reserved, Do Not Modify.
11:7	-	DNM	0	**Reserved, Do Not Modify**
6	icc0_flush	R/W	0	**ICC Cache Flush** Write 1 to flush all three caches. This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect. 0: Memory flush not in progress. 1: Memory flush in progress.
5	fpu_dis	R/W	0	**Floating Point Unit Disable** 0: Enabled. 1: Disabled.
4:3	-	RO	0	**Reserved**
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 4-38: 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 automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset. See the Device Resets section for additional information.
30	periph	R/W	0	**Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset. *Note: Watchdog timers, GPIO ports, the AoD, RAM retention and the general control registers (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 automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset. See the Device Resets section for additional information.
28	uart2	R/W	0	**UART2 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
27:25	-	RO	0	**Reserved**
24	trng	R/W	0	**TRNG Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
23:17	-	RO	0	**Reserved**
16	i2c0	R/W	0	**I²C0 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
15	spi2	R/W	0	**SPI2 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.

Reset 0			GCR_RST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
14	spi1	R/W	0	**SPI1 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
13	spi0	R/W	0	**SPI0 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
12	uart1	R/W	0	**UART1 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
11	uart0	R/W	0	**UART0 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
10:9	-	RO	0	**Reserved**
8	tmr3	R/W	0	**TMR3 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
7	tmr2	R/W	0	**TMR2 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
6	tmr1	R/W	0	**TMR1 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
5	tmr0	R/W	0	**TMR0 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
4	-	RO	-	**Reserved**

Reset 0			GCR_RST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
3	gpio1	R/W	0	**GPIO1 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
2	gpio0	R/W	0	**GPIO0 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
1	wdt0	R/W	0	**Watchdog Timer 0 Peripheral Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
0	dma	R/W	0	**DMA Access Block Reset** Write 1 to reset. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.

Table 4-39: System Clock Control Register

System Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31	extclk_rdy	R	1	**External Clock Ready** This bit field is set when the signal on the P0.12 device pin, driven by an external clock, is ready. 0: Not ready or not enabled. 1: External clock is ready.
30	-	RO	1	**Reserved**
29	inro_rdy		0	**Internal Nano-Ring Oscillator (INRO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
28	ibro_rdy	R	0	**Internal Baud Rate Oscillator (IBRO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
27	ipo_rdy	R	0	**Internal Primary Oscillator (IPO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
26	-	RO	0	**Reserved**
25	ertco_rdy	R	0	**32.768kHz External RTC Oscillator (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.

System Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
23:22	-	RO	0	**Reserved**
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 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	-	DNM	0	**Reserved. Do Not Modify.**
17	ertco_en	R/W	0	**ERTCO Enable** Set this field to 1 to enable the ERTCO. 0: Disabled. 1: Enabled and ready when *GCR_CLKCTRL.ertco_rdy* = 1.
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 *GCR_CLKCTRL.sysclk_sel*, there is a delay until the switchover is complete. This bit is cleared until the switchover is complete. 0: Switch to new clock source not yet complete. 1: SYS_OSC is clock source selected in *GCR_CLKCTRL. sysclk_sel*.
12	-	RO	0	**Reserved** Do not modify this field.
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 immediately clears *GCR_CLKCTRL.sysclk_rdy*. 0: Reserved. 1: Reserved. 2: ERFO. 3: INRO. 4: IPO. 5: IBRO. 6: ERTCO. 7: External Clock P0.12.

System Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
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	-	DNM	0b001000	**Reserved, DNM**

Table 4-40: 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: Clock source is crystal oscillator, driving the crystal connected between HFXIN and HFXOUT pins. 1: Clock source square wave driven into HFXIN pin.
19:18	-	DNM	0	**Reserved, Do Not Modify**
17	ibro_pd	R/W	1	**IBRO Power Down** This field must be set to 1 before entering *DEEPSLEEP*.
16	ipo_pd	R/W	1	**IPO Power Down** This field powers off the IPO in *DEEPSLEEP*. This field must be set to 1 before entering *DEEPSLEEP*.
15:13	-	DNM	0	**Reserved, Do Not Modify**
12	erfo_pd	R/W	1	**ERFO Power Down** This field powers off the ERFO in *DEEPSLEEP*. This field must be set to 1 before entering *DEEPSLEEP*.
11:9	-	DNM	0	**Reserved, Do Not Modify**
8	lpuart0_we	R/W	0	**LPUART0 Wake-Up Enable** Set this field to 1 to enable LPUART0 as a wake-up source. LPUART0 wakes the device from *SLEEP*, *DEEPSLEEP,* or *BACKUP* low-power modes. 0: Disabled. 1: Enabled.
7	lptmr1_we	R/W	0	**LPTMR1 Wake-up Enable** Set this field to 1 to enable LPTMR1 as a wake-up source. LPTMR1 wakes the device from *SLEEP*, *DEEPSLEEP,* or *BACKUP* low-power modes. 0: Disabled. 1: Enabled.
6	lptmr0_we	R/W	0	**LPTMR0 Wake-up Enable** Set this field to 1 to enable LPTMR0 as a wake-up source. LPTMR0 wakes the device from *SLEEP*, *DEEPSLEEP*, or *BACKUP* low-power modes. 0: Disabled. 1: Enabled.

Power Management			GCR_PM	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
5	rtc_we	R/W	0	**RTC Alarm Wake-up Enable** Set this field to 1 to enable an RTC alarm to wake the device. The RTC alarm wakes the device from *SLEEP*, *DEEPSLEEP, BACKUP,* or *STORAGE* low-power mode. 0: Disabled. 1: Enabled.
4	gpio_we	R/W	0	**GPIO Wake-Up Enable** Set this field to 1 to enable all GPIO pins as potential wake-up sources. Any GPIO configured for wake-up wakes the device from *SLEEP*, *DEEPSLEEP, BACKUP,* or *STORAGE* low-power mode. 0: Disabled. 1: Enabled.
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 4-41: 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** Set this field to 1 to enable the DIV_CLK_OUT signal. Refer to the device data sheet alternate function table. 0: Disabled. 1: Enabled.
15:14	div_clk_out_ctrl			**DIV_CLK_OUT Control** This field sets the source and frequency of the DIV_CLK_OUT signal. Refer to the device data sheet alternate function table. 0b00: DIV_CLK_OUT is off. 0b01: IBRO divided by 2. 0b10: ERFO divided by 4. 0b11: ERFO divided by 8.
12:2	-	RO	-	**Reserved**
1:0	aon_clkdiv	R/W	0	**AoD Clock Divider** Configures the frequency of the AoD clock. See the section *Oscillator Sources and Clock Switching* section for details.

Table 4-42: Peripheral Clock Disable Register 0

Peripheral Clock Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	DNM	1	**Reserved, Do Not Modify**
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.
27:19	-	DNM	1	**Reserved, Do Not Modify**
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	-	DNM	1	**Reserved, Do Not Modify**
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	-	DNM	1	**Reserved, Do Not Modify**
10	uart1	R/W	1	**UART1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
9	uart0	R/W	1	**UART0 Clock Disable** Disabling a clock peripheral disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.

Peripheral Clock Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
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	-	DNM	1	**Reserved, Do Not Modify**
1	gpio1	R/W	1	**GPIO1 Port and Pad Logic Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
0	gpio0	R/W	1	**GPIO0 Port and Pad Logic Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.

Table 4-43: Memory Clock Control Register

Memory Clock Control			GCR_MEMCTRL	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:14	-	RO	0	**Reserved**
13	romls_en	R/W	0	**ROM *LIGHTSLEEP* Enable** Data is unavailable for read/write operations in *LIGHTSLEEP*. 0: Disabled. 1: Enabled.
12	icc0ls_en	R/W	0	**ICC *LIGHTSLEEP* Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* but is retained. 0: Disabled. 1: Enabled.

Memory Clock Control			GCR_MEMCTRL	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
11	ram3ls_en	R/W	0	***Sysram3* and *Sysram7 LIGHTSLEEP* Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* but is retained. 0: Disabled. 1: Enabled. *Note: To put RAM in a shutdown that removes all power from the RAM and resets 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* but is retained. 0: Disabled. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and resets 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* but is retained. 0: Disabled. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and resets the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 3-1 for the 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* but is retained. 0: Disabled. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and resets the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 3-1 for the 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	**Program Flash Wait States** Number of wait-state SYS_OSC cycles per flash code read access. See *Flash Wait States* for details on this field's usage. 0 - 7: Number of flash code access wait states.

Table 4-44: Memory Zeroization Control Register

Memory Zeroization Control			GCR_MEMZ	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**

Memory Zeroization Control			GCR_MEMZ	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
2	icc0	R/W	0	**ICC Cache Data and Tag Zeroization** Write 1 to initiate the operation. 0: Normal operation. 1: Zeroize cache data and tag RAM.
1	ramcb	R/W	0	**System RAM Check Bit Block Zeroization** Write 1 to initiate the operation. 0: Normal operation. 1: Zeroize check bit RAM.
0	ram	R/W	0	**System RAM Zeroization** Write 1 to initiate the operation. 0: Normal operation. 1: Zeroize RAM.

Table 4-45: 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 4-46: Reset Register 1

Reset 1			GCR_RST1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23	i2s	R/W1O	0	**I²S Peripheral Reset** Write 1 to initiate the operation. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
22:18	-	RO	0	**Reserved**
17	i2c2	R/W1O	0	**I2C2 Peripheral Reset** Write 1 to initiate the operation. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
16:15	-	RO	0	**Reserved**
14	ac	R/W1O	0	**Auto Calibration Block Reset** Write 1 to initiate the operation. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
13:11	-	RO	-	**Reserved**

Reset 1			GCR_RST1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
10	aes	R/W1O	0	**AES Block Reset** Write 1 to initiate the operation. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
9	crc	R/W1O	0	**CRC Block Reset** Write 1 to initiate the operation. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
8	wdt1	R/W1O	0	**Watchdog Timer 1 Peripheral Reset** Write 1 to initiate the operation. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.
7:1	-	RO	0	**Reserved**
0	i2c1	R/W1O	0	**I²C1 Peripheral Reset** Write 1 to initiate the operation. This field is automatically cleared by hardware when the reset is complete. 0: Normal operation. 1: Reset.

Table 4-47: Peripheral Clock Disable Register 1

Peripheral Clock Disable 1			GCR_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	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**
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 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.

Peripheral Clock Disable 1			GCR_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
14	crc	R/W	1	**CRC 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.
13:12	-	RO	1	**Reserved**
11	icc0	R/W	0	**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	wwdt1	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	wwdt0	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 4-48: Event Enable Register

Event Enable			GCR_EVENTEN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	tx	R/W	0	**Transmit Event (TXEV) On Send Event (SEV) Enable** When set, a SEV instruction causes a TXEV event from the CPU. 0: Disabled. 1: Enabled.

Event Enable			GCR_EVENTEN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
1	rx	R/W	0	**Receive Event (RXEV) Event Enable** Set this field to 1 to enable generation of an RXEV event to wake the CPU from a WFE sleep state. 0: Disabled. 1: Enabled.
0	dma	R/W	0	**CPU DMA CTZ Wake-Up Enable** Allows a DMA0 CTZ event to generate an RXEV to wake-up CPU from a low-power mode entered with a WFE instruction. 0: Disabled. 1: Enabled.

Table 4-49: Revision Register

Revision			GCR_REVISION	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	revision	R	*	**Device Revision** Returns the chip revision ID as a packed BCD. See the section *Silicon Revision Differences* for details.

Table 4-50: 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* is set. 0: Disabled. 1: Enabled.

Table 4-51: Error Correction Coding Error Detected Register

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

Table 4-52: Error Correction Coding Correctable Error Detected Register

Error Correction Coding Correctable Error Detected			GCR_ECCCED	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 4-53: Error Correction Coding Interrupt Enable Register

Error Correction Coding Interrupt Enable			GCR_ECCIE	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 4-54: Error Correction Coding Address Register

Error Correction Coding Address			GCR_ECCADDR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

4.12 Error Correction Coding Enable Register (ECC)

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 4-55: Error Correction Coding Enable Register Summary

Offset	Register	Description
[0x0008]	*ECC_EN*	*Reserved*

4.12.1 Register Details

Table 4-56: Error Correction Coding Enable Register

Error Correction Coding Enable			ECC_EN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

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 4-57: System Initialization Register Summary

Offset	Register	Description
[0x0000]	*SIR_SIR_STATUS*	System Initialization Error Status Register
[0x0004]	*SIR_SIR_ADDR*	System Initialization Error Address Register

4.13.1 Register Details

Table 4-58: System Initialization Error Status Register

System Initialization Error Status			SIR_SIR_STATUS	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
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. Contact Maxim Integrated technical support for additional assistance providing the address contained in SIR_SIR_ADDR.addr.*

System Initialization Error Status			SIR_SIR_STATUS	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
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. Contact Analog Devices technical support for additional assistance.*

Table 4-59: System Initialization Error Address Register

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

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 4-60: 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

4.14.1 Register Details

Table 4-61: 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.

Function Control 0			FCR_FCTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
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.
19:3	-	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 4-62: Automatic Calibration 0 Register

Function Control 1			FCR_AUTOCAL0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:23	trim	R	0	**IPO Automatic Trim Value Output** This field contains the calculated trim output from an automatic calibration run.
22:20	-	RO	0	**Reserved**
19:8	gain	R/W	0	**IPO Trim Pulse Count** Load this field with the desired number of trim adjustment pulses required before the trim is updated during an atomic calibration operation. The recommended value for this field is 4.
7:5	-	RO	0	**Reserved**
4	atomic	R/W1O	0	**IPO Trim Atomic Start** Set this bit to start an automatic atomic calibration of the IPO. The calibration runs for *FCR_AUTOCAL2.runtime* milliseconds. This bit is cleared by hardware once the calibration is complete.
3	invert	RO	0	**IPO Trim Step Invert** Set this field to invert the up/down trim steps during calibration operations. 0: Trim steps are not inverted. 1: Trim steps inverted.
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 automatically cleared by hardware once the load is complete.
1	en	R/W	0	**IPO Automatic Calibration Continuous Mode Enable** 0: No effect. 1: Enabled.

Function Control 1			FCR_AUTOCAL0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
0	sel	R/W	0	**IPO Trim Select** Selects the trim value to use for the IPO. The reset default for this field uses the factory trim for the IPO. Setting this field to 1 uses the automatic calibration trim output stored in *FCR_AUTOCAL0.load*. 0: Use default trim. 1: Use automatic calibration trim values.

Table 4-63: 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 Trim Automatic Calibration Initial Trim** This field contains the initial trim setting for the IPO. Set this field to the desired initial trim value to use for an IPO automatic calibration operation. The closer this field is to the target trim value required, the faster the automatic trim operation completes. Set this field to 0x100 when performing autocalibration on the MAX32670. *Note: Valid values for this field are 1 to 0x100.*

Table 4-64: Automatic Calibration 2 Register

Automatic Calibration 2			FCR_AUTOCAL2	[0x000C]
Bits	**Name**	**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: $f_{IPO} = div \times 32{,}768$ *Note : Setting div to 0 is equivalent to setting div to 1.*
7:0	runtime	R/W	0	**IPO Trim Automatic Calibration Run Time** $Atomic\ Run\ Time = runtime\ milliseconds$

5. Interrupts and Exceptions

Interrupts and exceptions are managed by the Arm Cortex-M4 Nested Vector Interrupt Controller (NVIC). The NVIC handles the interrupts, exceptions, priorities, and masking. *Table 5-1* details the MAX32670/MAX32671 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 MAX32670/MAX32671. There are 100 interrupt entries for the MAX32670/MAX32671, 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 115.

Table 5-1: MAX32670/MAX32671 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

Exception (Interrupt) Number	Offset	Name	Description
24	[0x0060]	TMR3_IRQn	Timer 3 Interrupt
25	[0x0064]	TMR4_IRQn	LPTMR0 (TMR4) Interrupt
26	[0x0068]	TMR5_IRQn	LPTMR1 (TMR5) Interrupt
27:28	[0x006C]:[0x0070]	-	Reserved
29	[0x0074]	I2C0_IRQn	I²C Port 0 Interrupt
30	[0x0078]	UART0_IRQn	UART Port 0 Interrupt
31	[0x007C]	UART1_IRQn	UART Port 1 Interrupt
32	[0x0080]	SPI0_IRQn	SPI Port 0 Interrupt
33	[0x0084]	SPI1_IRQn	SPI Port 1 Interrupt
34	[0x0088]	SPI2_IRQn	SPI Port 2 Interrupt
35:38	[0x008C]:[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
42:43	[0x00A8]:[0x00AC]	-	Reserved
44	[0x00B0]	DMA0_IRQn	DMA0 Interrupt
45	[0x00B4]	DMA1_IRQn	DMA1 Interrupt
46	[0x00B8]	DMA2_IRQn	DMA2 Interrupt
47	[0x00BC]	DMA3_IRQn	DMA3 Interrupt
48:49	[0x00C0 : 0x00C4]	-	Reserved
50	[0x00C8]	UART2_IRQn	UART Port 2 Interrupt
51	[0x00CC]	-	Reserved
52	[0x00D0]	I2C1_IRQn	I²C 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	I²C Port 2 Interrupt
79:83	[0x013C}:[0x014C]	-	Reserved
84	[0x0150]	DMA4_IRQn	DMA4 Interrupt
85	[0x0154]	DMA5_IRQn	DMA5 Interrupt
86	[0x0158]	DMA6_IRQn	DMA6 Interrupt
87	[0x015C]	DMA7_IRQn	DMA7 Interrupt
88:97	[0x0160]:[0x0184]	-	Reserved

Exception (Interrupt) Number	Offset	Name	Description
98:103	[0x0188]: [0x019C]	-	Reserved
104	[0x01A0}	UART3_IRQn	LPUART0 Interrupt
105:112	[0x01A4]:[0x01C0]	-	Reserved
113	[0x01C4]	AES_IRQn	AES Block Interrupt
114	[0x01C8]	CRC_IRQn	CRC Block Interrupt
115	[0x01CC]	I2S_IRQn	I^2S Interrupt

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

The GPIO pins share an individually controlled I/O mode and an alternate function (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 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	Number of GPIO	Pins
40 TQFN	GPIO0[30:0]	31

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

Note: MAX32670/MAX32671 does not support the selectable GPIO voltage supply feature.

6.2 Configuration

6.2.1 Peripheral Clock Enable

The 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 .gpio0 to *GCR_RST0.gpio0* to 0.

6.2.2 Power-On-Reset Configuration

During a POR event, all I/O default to GPIO mode with input and output disabled except the SWDIO, SWDCLK, P0.4, and P0.5 pins. The SWD is enabled by default after POR with AF1 selected by hardware. See *ROM* Bootloader for exceptions.

Following a POR event, all GPIO except device pins SWDIO, SWDCLK, UART0A_RX, and UART0A_TX (P0.0, P0.1, P0.8, and P0.9) are configured with the following default settings:

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

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

1. Set the device pin P0.0 for AF1 mode:
   a. *GPIOn_EN0[0]* = 0.
   b. *GPIOn_EN1[0]* = 0.
   c. *GPIOn_EN2[0]* = 0.
2. Set device pin P0.1 for AF1 mode:
   a. *GPIOn_EN0[1]* = 0.
   b. *GPIOn_EN1[1]* = 0.
   c. *GPIOn_EN2[1]* = 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 Input Mode Configuration

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

1. Set the pin for I/O 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 6-2* for pullup and pulldown selection.
   a. GPIO pins with I^2C as an AF only support high-impedance or a weak pulldown resistor.
3. Enable the pin for input mode by setting *GPIOn_INEN[pin]* to 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 6-2*.

Table 6-2: MAX32670 Input Mode Configuration Summary

Input Mode	Pullup/Pulldown Enable *GPIOn_PADCTRL0[pin]* BITWISE OR *GPIOn_PADCTRL1[pin]*	Pullup/Pulldown Select *GPIOn_PS[pin]*
High impedance	0	N/A
Weak pullup to V_{DD}	1	1
Weak pulldown to V_{SS}	1	0

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

Note: GPIOn_PADCTRL1 reset default is 1.

6.2.5 Output Mode Configuration

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

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

6.2.6 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 6-3*. 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 6-3: Standard GPIO Drive Strength Selection

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

For GPIO with I^2C as an AF, *Table 6-4* shows the drive strength setting options.

Table 6-4: GPIO with I^2C AF Drive Strength Selection

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

6.2.7 Alternate Function Usage

Table 6-5 shows the bit settings for the *GPIOn_EN2* and *GPIOn_EN1* fields to configure the function of the GPIO port pins for a desired alternate function. For example, *GPIO0_EN1.[25]*, and *GPIO0_EN2.[25]* all represent configuration for device pin P0.25.

Note: 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 6-5: MAX32670 GPIO Mode and AF Selection

Alternate Function Selection	*GPIOn_EN2[pin]*	*GPIOn_EN1[pin]*
AF1	0	0
AF2	0	1
AF3	1	0
AF4	1	1

Most GPIO support one or more alternate functions selected with the GPIO configuration enable bits shown in *Table 6-5*. 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 following steps describe how to configure a pin for alternate function usage.

1. Set the pin to I/O mode by setting *GPIOn_EN0[pin]* to 1.
   a. This step is important to prevent selection of unintended alternate functions during configuration.
2. Set *GPIOn_EN2[pin]* and *GPIOn_EN1[pin]* to the values for the desired alternate function, as shown in *Table 6-5*.
3. Configure the electrical characteristics of the pin. See *Table 6-2* if the assigned alternate function uses the pin as an input. See *Table 6-3* if the assigned alternate function uses the pin as an output.
4. Set *GPIOn_EN0[pin]* to 0 to enable the alternate function.

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. The interrupts are peripheral controlled if the GPIO is configured as a peripheral AF. GPIO interrupts can be independently

enabled for any number of GPIO on each GPIO port. All implemented pins of a GPIO port have a single assigned/shared interrupt vector.

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

1. Disable interrupts by setting the *GPIOn_INEN[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 *GPIOn_INEN[31:0]*. To maintain previously enabled interrupts, read the *GPIOn_INEN* 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.
        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]* = 0: Input rising edge triggers interrupt.
        ii. *GPIOn_INTPOL[pin]* = 1: Input falling 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_INTEN[pin]* to 1 to enable the interrupt for the pin.

6.3.1 GPIO Interrupt Handling

Each GPIO port is assigned its own dedicated interrupt vector, as shown in *Table 6-7*. 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[pin]* bit position that triggered the interrupt. If multiple bits are set in the *GPIOn_INTFL* register, all of the corresponding the bits should be cleared.
4. Return from the interrupt vector handler.

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

Table 6-6: MAX32670 GPIO Interrupt Enable Settings for Each Supported Operating Mode

Operating Mode	*GPIOn_INTEN*	Interrupt Handler
ACTIVE	X	GPIO0_IRQn
SLEEP	X	GPIO0_IRQn
Note: Wake from DEEPSLEEP, BACKUP, and STORAGE is only supported using the GPIOWAKE interrupt.		

Table 6-7: MAX32670 GPIO Port Interrupt Vector Mapping

GPIO Wake Interrupt Source	GPIO Interrupt Status Register	Device Specific Interrupt Vector Number	GPIO Interrupt Vector
GPIO0[31:0]	GPIO0_INTFL	40	GPIO0_IRQn

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

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

external edge-triggered interrupts on the GPIO ports. Level triggered interrupts are not supported for wake-up because the system clock must be active to detect levels.

6.3.3 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. When the wake-up event occurs, the application software must interrogate the *PWRSEQ_LPWKST0* register to determine which GPIO0 port pin caused the interrupt.

Table 6-8: GPIO Wakeup Interrupt Vector

GPIO Wake Interrupt Source	GPIO Wake Interrupt Status Register	Device Specific Interrupt Vector Number	GPIO Wake-up Interrupt Vector
GPIO0[31:0]	*GPIOn_INTFL*	70	GPIOWAKE_IRQn

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

1. Clear pending interrupt flags by writing 0xFFFF FFFF to the *PWRSEQ_LPWKST0* register.
2. Set up a GPIOWAKE_IRQn interrupt handler.
3. Enable the GPIOWAKE for each desired pin by setting *PWRSEQ_LPWKEN0[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.

6.4 GPIO 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 registers, as shown in *Table 6-9*. 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 6-9: 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
[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

Offset	Register Name	Description
[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 6-10: GPIO AF 0 Select Register

GPIO AF 0 Select Register			GPIOn_EN0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	1	**GPIO Configuration Enable Bit 0** These bits, in conjunction with bits in Table 6-5, configure the corresponding device pin for digital I/O or an alternate function mode. This field can be modified directly by writing to this register or indirectly through GPIOn_EN0_SET or GPIOn_EN0_CLR. See Alternate Function for details on this register's usage. 0: Alternate function selected. 1: I/O mode 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: This register setting does not affect the input and interrupt functionality of the associated pin.*

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

GPIO Port n Configuration Enable Atomic Set Bit 0			GPIOn_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 bits in the GPIOn_EN0 register are 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 6-12: 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 the GPIOn_EN0 register are 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 6-13: 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 6-14: 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 6-15: 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 6-16: GPIO Port n Output Register

GPIO Port n Output			GPIOn_OUT	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Output 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 6-17: GPIO Port n Output Atomic Set Register

GPIO Port n Output Atomic Set			GPIOn_OUT_SET	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	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 6-18: GPIO Port n Output Atomic Clear Register

GPIO Port n Output Atomic Clear			GPIOn_OUT_CLR	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	WO	0	**GPIO Output Atomic Clear** 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 6-19: GPIO Port n Input Register

GPIO Port n Input			GPIOn_IN	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R	0	**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 6-20: 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 the corresponding GPIO pin. Clearing a bit in this field sets level-triggered interrupt for the corresponding GPIO pin. 0: Level triggered interrupt. 1: Edge triggered interrupt. *Note: This bit has no effect unless the corresponding bit in the GPIOn_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 6-21: 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 6-22: GPIO Port n Input Enable Register

GPIO Port n Input Enable			GPIOn_INEN	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0x0000 0303	**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 connected. *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 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 6-24: 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 the *GPIOn_INTEN* register are 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 6-25: 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 the *GPIOn_INTEN* register are 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 6-26: 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 associated GPIO pin. 1: GPIO interrupt pending for associated GPIO pin. *Note: Write a 1 to the corresponding bit in the GPIOn_INTFL_CLR register to clear the interrupt pending status flag.* *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 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 6-28: GPIO Port n Wakeup Enable Register

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

Table 6-30: 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**

Table 6-31: 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 6-32: 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	0	**Pullup/Pulldown Enable** Setting a bit in this field enables the weak pullup or pulldown resistor on the corresponding GPIO port device pin. The selection for pullup or pulldown resistor is set using the *GPIOn_PS* register. **CAUTION:** This field is OR'd with the corresponding bit field of *GPIOn_PADCTRL1*. 0: Input floating. 1: Pullup/Pulldown resistor selected. *Note: This field applies even if the input is disabled (GPIOn_INEN[pin] = 0).* *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 a weak pullup resistor. 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. If the corresponding GPIO with I^2C as an AF bit in the GPIOn_PS register is set to 1, setting the same bit in this register has no effect.*

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

GPIO Port n Pad Control 1			GPIOn_PADCTRL1	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**Pullup/Pulldown Enable** Setting a bit in this field enables the weak pullup or pulldown resistor on the corresponding GPIO port device pin. The selection for pullup or pulldown resistor is set using the *GPIOn_PS* register. **CAUTION:** This field is OR'd with the corresponding bit field of *GPIOn_PADCTRL0*. 0: Input floating. 1: Pullup/Pulldown resistor selected. *Note: This field is applied even if the input is disabled (GPIOn_INEN[pin] = 0).* *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 a weak pullup resistor. 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. If the corresponding GPIO with I^2C as an AF bit in the GPIOn_PS register is set to 1, setting the same bit in this register has no effect.*

Table 6-34: 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 6-5*, 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*. See *Alternate Function* for details on this register's usage. *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.*

Table 6-35: 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 the *GPIOn_EN1* register are 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 6-36: 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 the *GPIOn_EN1* register are 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 6-37: 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 6-5*, 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*. See *Alternate Function* for details on this register's usage. *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.*

Table 6-38: 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 the *GPIOn_EN2* register are 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 6-39: 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 the *GPIOn_EN2* register are 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 6-40: 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 6-41: 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 6-42: 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 I²C 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 I²C 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 6-43: 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 I²C 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 I²C 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 6-44: GPIO Port n Pulldown/Pullup Strength Select Register

GPIO Port n Pulldown/Pullup Strength Select			GPIOn_PS	[0x00B8]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**Pullup/Pulldown Resistor Select** Setting a bit in this field selects a weak pullup resistor for the corresponding GPIO port device pin. Clearing a bit in this field selects a weak pulldown resistor for the corresponding GPIO port device pin. The *GPIOn_PADCTRL0* or *GPIOn_PADCTRL1* registers must be configured to enable the pull resistor selection. 0: Pulldown resistor selected. 1: Pullup resistor selected. *Note: GPIO with I²C as an AF do not support a weak pullup resistor. As such, the bits in this register that control GPIO with I²C as an AF should always be set to 0.* *Note: 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 I²C functionality. See GPIOn_PADCTRL0 and GPIOn_PADCTRL1.*

Table 6-45: 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 MAX32670/MAX32671 Flash Controller manages read, write, and erase accesses to the internal flash. It provides the following features:

- Up to 384KB total internal flash memory.
  - Page 48 is used by the bootloader and cannot be used for application software storage.
- 48 pages.
- 8,192 bytes per page.
- 2,048 words by 128 bits per page.
- 128-bit data reads and writes.
- Page erase and mass erase support.
- Write protection.

7.1 Instances

The device provides one instance of the flash controller. The flash is programmable through the serial wire debug interface (in-system) or directly with user software (in-application).

Table 7-1 shows the start address and end address of internal flash memory.

Table 7-1: MAX32670/MAX32671 Internal Flash Memory Organization

Instance Number	Page Number	Size (per page)	Start Address	End Address
FLC	1	8,192	0x1000 0000	0x1000 1FFF
	2	8,192	0x1000 2000	0x1000 3FFF
	3	8,192	0x1000 4000	0x1000 5FFF
	4	8,192	0x1000 6000	0x1000 7FFF
	…	…	…	…
	47	8,192	0x1005 C000	0x1005 DFFF
	48	Reserved	0x1005 E000	0x1005 FFFF

7.2 Usage

The flash controller manages write and erase operations for internal flash memory and provides a lock mechanism to prevent unintentional writes to the internal flash. In-application and in-system programming, page erase, and mass erase operations are supported. 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 $f_{FLC\_CLK} = 1MHz$, as shown in *Equation 7-1*. For the IPO as the system clock, the *FLC_CLKDIV.clkdiv* should be set to 100.

***CAUTION**: The FLC_CLKDIV.clkdiv field resets to an invalid clock divisor on all forms of reset.*

Equation 7-1: FLC Clock Frequency

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

7.2.2 Lock Protection

A locking mechanism prevents accidental memory writes and erases. All writes and erase operations require the *FLC_CTRL.unlock* field to be set to 2 before starting the operation. Writing any other value to the *FLC_CTRL.unlock* field results in the flash instance becoming locked.

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, FLC_INTR.af, to indicate an access violation occurred.

7.2.3 Flash Write Width

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

7.2.4 Flash Write

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

1. If desired, enable flash controller interrupts by setting the *FLC_INTR.afie* and *FLC_INTR.doneie* bits.
2. Read the *FLC_CTRL.pend* bit until it returns 0.
3. Configure *FLC_CLKDIV.clkdiv* to achieve a 1MHz flash clock based on the current SYS_CLK frequency..
4. Set the *FLC_ADDR* register to a valid target address.
5. Set *FLC_DATA[3]*, *FLC_DATA[2]*, *FLC_DATA[1]*, and *FLC_DATA[0]* to the data to write.
    a. *FLC_DATA[3]* is the most significant word, and *FLC_DATA[0]* is the least significant word.
        i. Each word of the data to write follows the little-endian format where the least significant byte of the word is stored at the lowest-numbered byte, and the most significant byte is stored at the highest-numbered byte.
6. Set *FLC_CTRL.unlock* to 2 to unlock the flash instance.
7. Set *FLC_CTRL.wr* to 1.
    a. This field is automatically cleared by the FLC when the write operation is finished.
8. The *FLC_INTR.done* field is set to 1 by hardware when the write completes An interrupt is generated if the *FLC_INTR.doneie* field is set to 1.
    a. If an error occurred, the *FLC_INTR.af* field is set to 1 by the hardware and an interrupt is generated if the *FLC_INTR.afie* field is set to 1.
9. Set *FLC_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 the ICC is enabled, either disable it before writing to the flash or flush it 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.*

7.2.5 Page Erase

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

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

1. If desired, enable flash controller interrupts by setting the *FLC_INTR.afie* and *FLC_INTR.doneie bits*.
2. Read the *FLC_CTRL.pend* bit until it returns 0.
3. Configure *FLC_CLKDIV.clkdiv* to achieve a 1MHz flash clock based on the current SYS_CLK frequency.
4. Set the *FLC_ADDR* register to an address within the target page to be erased. *FLC_ADDR*[12:0] are ignored by the FLC to ensure the address is page aligned.
5. Set *FLC_CTRL.unlock* to 2 to unlock the flash instance.
6. Set *FLC_CTRL.erase_code* to 0x55 for page erase.
7. Set *FLC_CTRL.pge* to 1 to start the page erase operation.
8. The *FLC_CTRL.pend* bit is set by the flash controller while the page erase is in progress and the *FLC_CTRL.pge* and *FLC_CTRL.pend* are cleared by the flash controller when the page erase is complete.
9. *FLC_INTR.done* is set by hardware when the page erase completes and if an error occurred, the *FLC_INTR.af* flag is set. These bits generate a flash interrupt if enabled.
10. Set *FLC_CTRL.unlock* to any value other than 2 to re-lock the flash instance.

Note: If the ICC is enabled, either disable it before erasing a page or flush it after erasing the page.

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

7.2.6 Mass Erase

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

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

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

7.3 Flash Registers

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

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

Table 7-2: Flash Controller Register Summary

Offset	Register Name	Description
[0x0000]	*FLC_ADDR*	*Flash Controller Address Pointer Register*
[0x0004]	*FLC_CLKDIV*	*Flash Controller Clock Divisor Register*
[0x0008]	*FLC_CTRL*	*Flash Controller Control Register*
[0x0024]	*FLC_INTR*	*Flash Controller Interrupt Register*
[0x0028]	*FLC_ECCDATA*	*Reserved*
[0x0030]	*FLC_DATA[0]*	*Flash Controller Data Register 0*
[0x0034]	*FLC_DATA[1]*	*Flash Controller Data Register 1*
[0x0038]	*FLC_DATA[2]*	*Flash Controller Data Register 2*
[0x003C]	*FLC_DATA[3]*	*Flash Controller Data Register 3*
[0x0040]	*FLC_ACTRL*	*Flash Controller Access Control*
[0x0080]	*FLC_WELR0*	*Flash Controller Write/Erase Lock Register 0*
[0x0088]	*FLC_WELR1*	*Flash Controller Write/Erase Lock Register 1*
[0x0090]	*FLC_RLR0*	*Flash Controller Read Lock Register 0*
[0x0098]	*FLC_RLR1*	*Flash Controller Read Lock Register 1*

7.3.1 Register Details

Table 7-3: Flash Controller Address Pointer Register

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

Table 7-4: Flash Controller Clock Divisor Register

Flash Controller Clock Divisor Register			FLC_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 section *Clock Configuration* for details. ***CAUTION**: This field resets to an invalid clock divisor on all forms of reset.*

Table 7-5: Flash Controller Control Register

Flash Controller Control Register			FLC_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:28	unlock	R/W	0	**Flash Unlock** Write the unlock code, 2, before any flash write or erase operation to unlock the Flash. Writing any other value to this field locks the internal flash. 2: Flash unlock code.
27:26	-	RO	-	**Reserved**
25	lve	R/W	0	**Low Voltage Enable** Set this field to 1 to enable low voltage operation for the flash memory. See *Core Operating Voltage Range Selection* for detailed usage information on this setting. 0: Low voltage operation disabled. 1: Low voltage operation enabled. *Note: The PWRSEQ_LPCN.ovr field must be set to 0 or 1 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 *FLC_INTR* register are ignored by the Flash Controller. This bit will be cleared by hardware once the flash becomes accessible. *Note: If the Flash Controller is busy (FLC_CTRL.pend = 1), reads, writes and erase operations are not allowed and result in an access failure (FLC_INTR.af = 1).* 0: Flash idle. 1: Flash busy.
23:16	-	RO	0	**Reserved**
15:8	erase_code	R/W	0	**Erase Code** Prior to an erase operation this field must be set to 0x55 for a page erase or 0xAA for a mass erase. The flash must be unlocked prior to setting the erase code. This field is automatically cleared after the erase operation is complete. 0: 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 *FLC_ADDR.addr*. The flash must be unlocked prior to attempting a page erase, see *FLC_CTRL.unlock* for details. The Flash Controller hardware clears this bit when a page erase operation is complete. 0: No page erase operation in process or page erase is complete. 1: Write a 1 to initiate a page erase. If this field reads 1, a page erase operation is in progress.

Flash Controller Control Register			FLC_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
1	me	R/W1O	0	**Mass Erase** Write a 1 to this field to initiate a mass erase of the internal flash memory. The flash must be unlocked prior to attempting a mass erase, see *FLC_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 will write to the address set in the *FLC_ADDR* register. 0: No write operation in process or write operation complete. 1: Write 1 to initiate a write operation. If this field reads 1, a write operation is in progress. *Note: This field is protected and cannot be set to 0 by application code.*

Table 7-6: Flash Controller Interrupt Register

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

Table 7-7: Flash Controller ECC Data Register

Flash Controller ECC Data			FLC_ECCDATA	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 7-8: Flash Controller Data 0 Register

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

Table 7-9: Flash Controller Data Register 1

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

Table 7-10: Flash Controller Data Register 2

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

Table 7-11: Flash Controller Data Register 3

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

Table 7-12: Flash Controller Access Control Register

Flash Controller Access Control			FLC_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 *Information Block Flash Memory* for details.

Table 7-13: Flash Controller Write/Erase Lock Register 0

Flash Controller Write/Erase Lock 0			FLC_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. *FLC_WELR0*[0] maps to page 1 of the flash and *FLC_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. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding page of flash is write protected. 1: The corresponding page of flash is *not* write protected.

Table 7-14: Flash Controller Write/Erase Lock Register 1

Flash Controller Write/Erase Lock 1			FLC_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. *FLC_WELR1*[0] maps to page 33 of the flash and *FLC_WELR1*[15] maps to page 48. 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. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding page of flash is write protected. 1: The corresponding page of flash is *not* write protected.

Table 7-15: Flash Controller Read Lock Register 0

Flash Controller Read Lock 0			FLC_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. *FLC_RLR0*[0] maps to page 1 of the flash and *FLC_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 7-16: Flash Controller Read Lock Register 1

Flash Controller Read Lock 1			FLC_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. *FLC_RLR1*[0] maps to page 33 of the flash and *FLC_RLR1*[15] maps to page 48 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 standard DMA is a hardware feature that provides the ability to perform high-speed, block memory transfers of data independent of an Arm core. All DMA transactions consist of burst read from the source into the internal DMA FIFO followed by a burst write from the internal DMA FIFO to the destination.

DMA transfers are one of three types:

- From a receive FIFO to a RAM address.
- From a RAM address to a transmit FIFO.
- To a transmit FIFO from a RAM address.
- From a source SRAM address to a destination SRAM address.

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

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

8.1 Instances

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

Table 8-1: MAX32670/MAX32671 DMA and Channel Instances

DMA Instance	DMA_CHn Channel Instance
DMA	DMA_CH0
	DMA_CH1
	DMA_CH2
	DMA_CH3
	DMA_CH4
	DMA_CH5
	DMA_CH6
	DMA_CH7

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 truly 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 completed.

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 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: MAX32670/MAX32671 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	Reserved	-	-
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 FIFO	*DMA_CHn_DST*
0x11:0x1B	Reserved	-	-
0x1C	UART3 (LPUART0)	UART3 Receive FIFO	*DMA_CHn_DST*
0x1D	Reserved		-
0x1E	I^2S	I^2S Receive FIFO	*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 Transmit FIFO
0x31:0x3B	Reserved		
0x3C	UART3 (LPUART0)	*DMA_CHn_SRC*	UART3 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 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. Set *DMA_CHn_CTRL*.*en, DMA_CHn_CTRL*.*rlden*, and *DMA_CHn_STATUS*.*ctz_if* to 0.
2. If using memory DMA transfer destination, configure *DMA_CHn_DST* register to the starting address of the destination in memory.
3. If using memory for the source of the DMA transfer, 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_per* to 0x0 to disable the channel timeout feature.
6. Set *DMA_CHn_CTRL*.*en* = 1 to immediately start the DMA transfer.
   a. If using the reload feature, set *DMA_CHn_CTRL*.*en* and *DMA_CHn_CTRL*.*rlden* to 1 in a single 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, there are two possible responses depending on the value of the *DMA_CHn_CTRL*.*rlden*:

- 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.

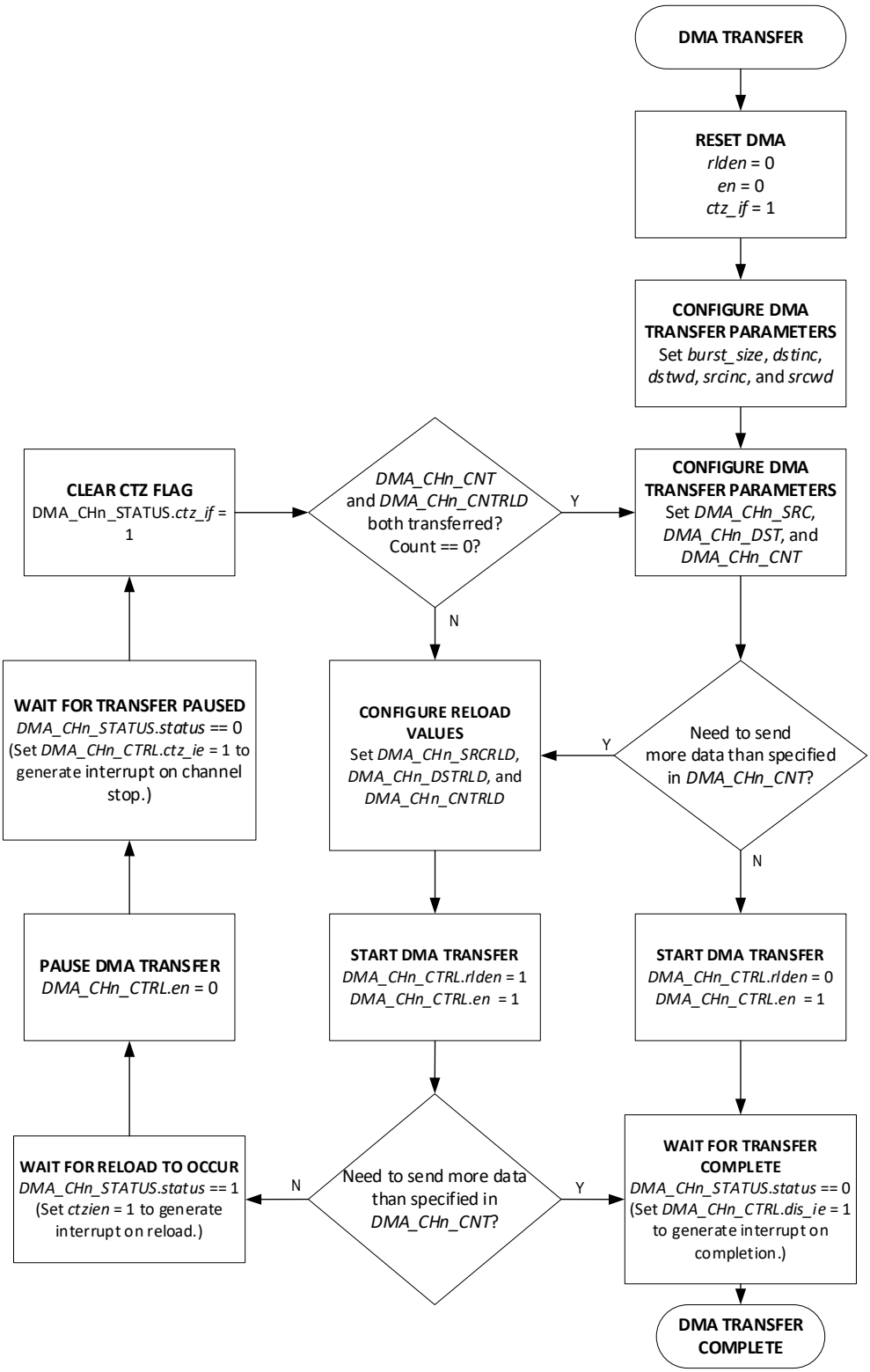
8.5 Chaining Buffers

Chaining buffers reduce the DMA ISR response time and allow 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/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 its 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: DMA Channel Timeout Configuration*. 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)**
0	Channel timeout disabled
1	$\frac{2^8 * [Value\ from\ DMA\_CHn\_CTRL.to\_per]}{f_{HCLK}}$
2	$\frac{2^{16} * [Value\ from\ DMA\_CHn\_CTRL.to\_per]}{f_{HCLK}}$
3	$\frac{2^{24} * [Value\ from\ DMA\_CHn\_CTRL.to\_per]}{f_{\mathrm{HCLK}}}$

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

- If *DMA_CHn_CTRL.to_wait* = 0, the timer begins counting immediately after *DMA_CHn_CTRL.to_per* 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, 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 always active. This means that the DMA channel generates an almost constant request for the bus until the transfer is complete. For this reason, assign a lower priority to channels executing memory-to-memory transfers to prevent starvation of other DMA channels.

Note: Memory-to-memory transfers are limited to RAM only memory.

8.9 DMA Registers

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

Table 8-6: DMA Register Summary

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

DMA Interrupt Enable			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 set only if the corresponding interrupt enable field is set in the *DMA_INTEN* register. Register bits associated with unimplemented channels should be ignored. 0: No interrupt. 1: Interrupt pending.

8.10 DMA Channel Register Summary

Table 8-9: Standard DMA Channel 0 to Channel 7 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
[0x0200]	DMA_CH5	DMA Channel 5
[0x0220]	DMA_CH6	DMA Channel 6
[0x0240]	DMA_CH7	DMA Channel 7

8.11 DMA Channel 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, as shown in *Table 8-10*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

See *Table 1-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a 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.
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.

DMA Channel *n* Control			DMA_CHn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
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** Indicates the width of each AHB transaction to the destination peripheral or memory (the actual width might be less than this if there are insufficient bytes in the DMA FIFO for the full width). 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** Indicates the width of each AHB transaction from the source peripheral or memory. The actual width might be less than this if the *DMA_CHn_CNT* register indicates a smaller value. 0: 1 byte. 1: 2 bytes. 2: 4 bytes. 3: Reserved.
15:14	to_clkdiv	R/W	0	**Timeout Timer Clock Pre-Scale Select** Selects the Pre-Scale divider for the timer clock input. 0: 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** 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 timer 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** Sets the priority of the channel relative to other channels of DMA. Channels of 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 timeout. 1: A channel timeout 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 configuration, address, and count registers for the channel. 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 Device Address** For peripheral transfers, the actual address field is either ignored or forced to zero because peripherals only have one location to read/write data based on the request select chosen. If *DMA_CHn_CTRL*.*srcinc* = 1, then this register is incremented on each AHB transfer cycle by one, two, or four bytes depending on the data width. If *DMA_CHn_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** For peripheral transfers, the actual address field is either ignored or forced to zero because peripherals only have one location to read/write data based on the request select chosen. If *DMA_CHn_CTRL.dstinc* = 1, then this register is incremented on every AHB transfer cycle by one, two, or four bytes depending on the data width. If a CTZ condition occurs while *DMA_CHn_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 register 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 the value of this register 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 the value of this register 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	ren	RO	0	**Reserved** This field is reserved and must be set to 0.

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

9. Universal Asynchronous Receiver/Transmitter (UART)

The UART and the low-power universal asynchronous receiver/transmitter (LPUART) interfaces communicate with external devices using industry-standard serial communications protocols. The UARTs are full-duplex serial ports. Each UART instance is independently configurable unless using a shared external clock source.

The LPUART is a version of the peripheral that can receive characters at up to 9600 baud while in low-power modes. The hardware loads valid received characters into the receive FIFO and wakes the device when an enabled interrupt condition occurs.

The peripheral provides the following features:

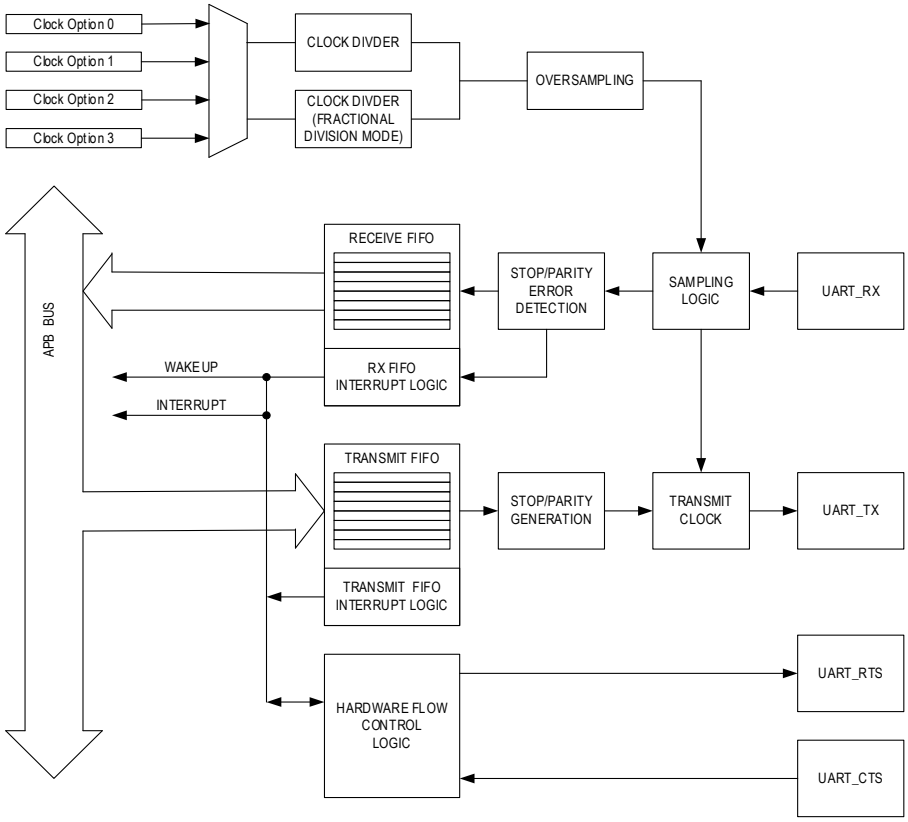
- Flexible baud rate generation for standard UART instances.
- Programmable character size of 5 to 8 bits.
- Stop bit settings of 1, 1.5, or 2 bits.
- Parity settings of even, odd, mark (always 1), space (always 0), and no parity.
- Automatic parity error detection with selectable parity bias.
- Automatic 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.*

LPUART instances provide 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.
- Wake up 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.

AOD_CLK is a scaled version of PCLK described in the section *System, Power, Clocks, Reset*.

A single external pin provides either the EXT_CLK1 source for the UART or the EXT_CLK2 external clock source for the LPUARTs. As a result, the external clock source cannot be selected by both a UART and LPUART simultaneously.

Table 9-1: MAX32670/MAX32671 UART/LPUART Instances

Instance	FDM Support	Power Modes	CLK0	CLK1	CLK2	CLK3	C_TX_FIFO_DEPTH C_RX_FIFO_DEPTH
UART0	NO	*ACTIVE SLEEP*	PCLK	EXT_CLK1 GPIO0.12 (AF4)	IBRO	ERFO	8/8
UART1							
UART2							
LPUART0	YES	*ACTIVE SLEEP*	AOD_CLK	EXT_CLK2 GPIO0.12 (AF2)	ERTCO	INRO[1]	8/8
		DEEPSLEEP BACKUP	N/A	EXT_CLK2 GPIO0.12 (AF2)	ERTCO	INRO[1]	8/8
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. The hardware automatically triggers DMA transfers based on the number of bytes in the receive FIFO and transmit FIFO.

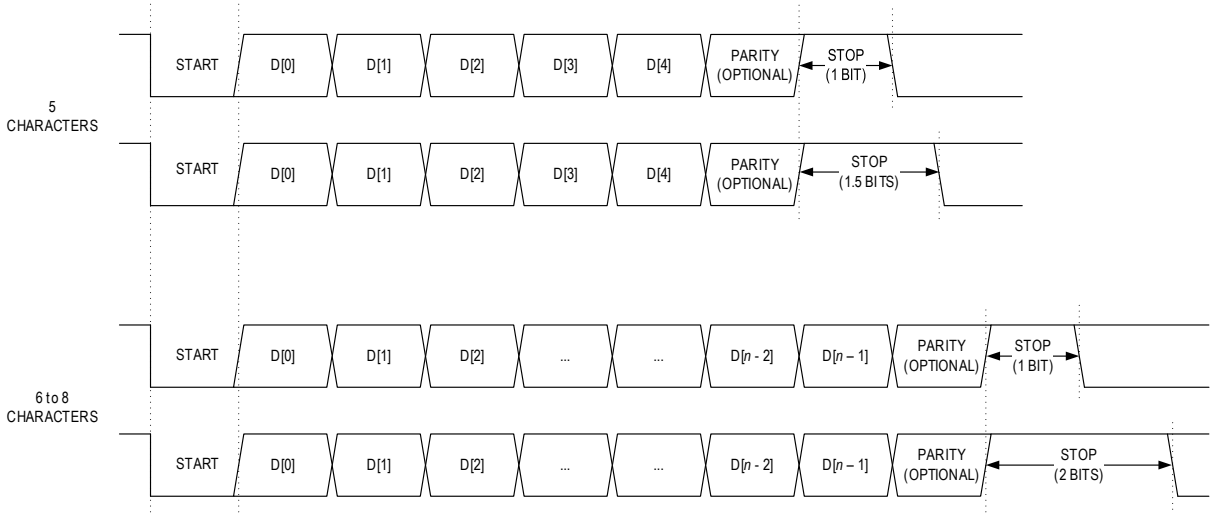
The behavior of the DMA requests are:

- 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 Transmit FIFO Operation

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

9.4.2 Receive FIFO Operation

Reads of the *UARTn_FIFO.data* field return the character values in the receive FIFO and decrement the *UARTn_STATUS.rx_lvl*. An overrun event occurs if a valid frame, including parity, is detected while *UARTn_STATUS.rx_lvl* = C_RX_FIFO_DEPTH. When an overrun event occurs, the data is discarded by hardware.

A parity error event indicates that the value read from *UARTn_FIFO.data* contains a parity error.

9.4.3 Flushing

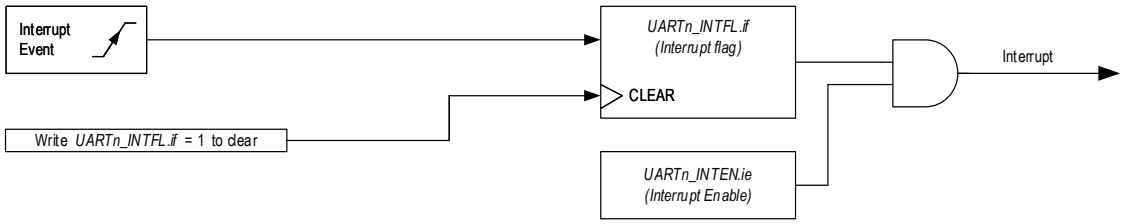
The FIFOs are flushed on the following conditions:

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

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: MAX32670/MAX32671 Interrupt Events

Event	Interrupt Flag	Interrupt Enable
Frame Error	*UARTn_INT_FL.rx_ferr*	*UARTn_INT_EN.rx_ferr*
Parity Error	*UARTn_INT_FL.rx_par*	*UARTn_INT_EN.rx_par*
CTS Signal Change	*UARTn_INT_FL.cts_ev*	*UARTn_INT_EN.cts_ev*
Receive FIFO Overrun	*UARTn_INT_FL.rx_ov*	*UARTn_INT_EN.rx_ov*
Receive FIFO Threshold	*UARTn_INT_FL.rx_thd*	*UARTn_INT_EN.rx_thd*
Transmit FIFO Half-Empty	*UARTn_INT_FL.tx_he*	*UARTn_INT_EN.tx_he*
Transmit FIFO Almost Empty	*UARTn_INT_FL.tx_ob*	*UARTn_INT_EN.tx_ob*

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 (*UARTn_CTRL.fdm* = 0).
  - The start bit is sampled three times, and all samples must be 0, or a frame error is generated.
  - Each data bit is sampled, and two of the three samples must match, or a frame error is generated.
  - If parity is enabled, the parity bit is sampled three times, and all samples must match, or a frame error is generated.
  - The stop bit is sampled three 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 three times, and all samples must be 0, or a frame error is generated.
  - Each data bit is sampled three times, and all samples must match, or a frame error is generated.
  - If parity is enabled, the parity bit is sampled three times, and all samples must match, or a frame error is generated.
  - The stop bit is sampled three 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 Inactive State

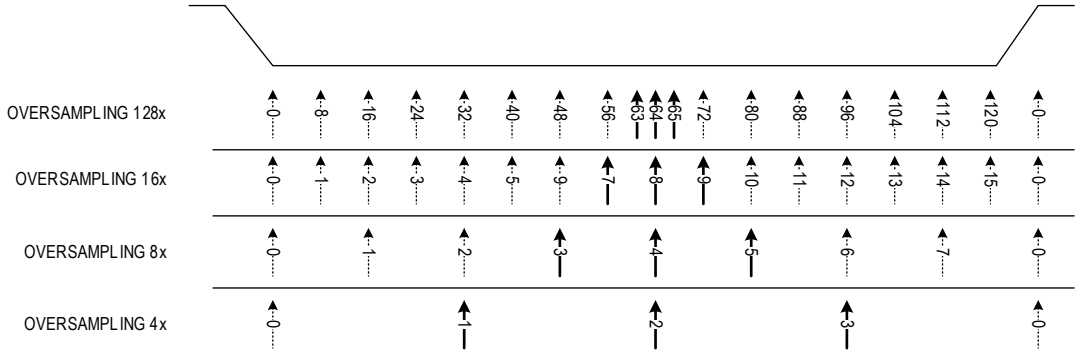
The following conditions result in the UART being inactive:

- 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

9.7 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* < 16 (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).

9.8 Baud Rate Generation

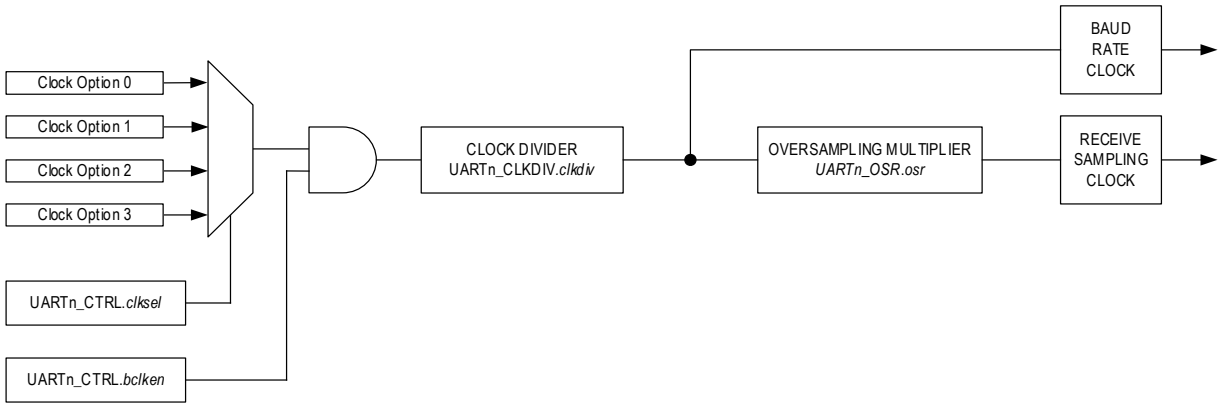
The baud rate is determined by the selected UART clock source and the value of the clock divisor. Multiple clock sources are available for each UART instance. See *Table 9-1* for available clock sources.

Note: Chang the clock source only between data transfers to avoid corrupting an ongoing data transfer.

9.8.1 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



9.8.2 Baud Rate Calculation

The standard UART transmit and receive circuits share a common baud rate clock, which is the selected UART clock source divided by the clock divisor. Similarly, the low-power UARTs support a 0.5 fractional clock divisor when *UARTn_CTRL.fdm* is set to 1. *Equation 9-2* should be used for calculating baud rates for *ACTIVE* and *SLEEP*. *Equation 9-3* is used for LPUARTs operating in *BACKUP* and *DEEPSLEEP* and requires setting *UARTn_CTRL.fdm* to 1. This allows for greater accuracy when operating at very low baud rates and finer granularity for the oversampling rate.

Equation 9-2: UART Baud Rate Equation (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: Low-Power UART Baud Rate Equation With FDM Enabled (UARTn_CTRL.fdm = 1)

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

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

For example, in a case where the UART clock is PCLK (50MHz), and the desired baud rate is 115,200bps, calculate the *UARTn_CLKDIV.clkdiv* field as follows:

$$UARTn\_CLKDIV.clkdiv = INT\left[\frac{50{,}000{,}000}{115{,}200}\right] = 434$$

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 the selected OSR setting. In general, a *UARTn_OSR.osr* setting of 5 is sufficient for most applications.

9.9 Low-Power Receiver Operation

The LPUARTs can be configured to receive up to 9600 baud in *DEEPSLEEP* and *BACKUP*. If a valid frame is received, the receive FIFO is loaded, and the receive FIFO level is incremented. If enabled, the wake-up conditions in *Table 9-6* wake the device from the low-power mode to *ACTIVE*.

The LPUARTs support FDM (*UARTn_CTRL.fdm* = 1), 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 9-5 uses *Equation 9-3* to calculate the *UARTn_CLKDIV.clkdiv* settings and possible OSR settings for LPUART operation with FDM enabled(*UARTn_CTRL.fdm* = 1).

Table 9-5: Slow Baud Rate Generation Example (FDM = 1)

Clock Source	Baud	*UARTn_CLKDIV.clkdiv*	*UARTn_OSR.osr* settings
ERTCO	9,600	6	N/A (1x)
	7,200	9	N/A (1x)
	4,800	13	N/A (1x)
	2,400	27	0: 8× 1: 12×
	1,800	36	0: 8× 1: 12× 2: 16×

Clock Source	Baud	UARTn_CLKDIV.clkdiv	UARTn_OSR.osr settings
	1,200	54	0: 8× 1: 12× 2: 16× 3: 20× 4: 24×

The following steps configure the LPUART for operation at 9,600 baud in low-power modes.

1. Enable the LPUART for operation in *DEEPSLEEP* and *BACKUP* by setting *MCR_CLKDIS.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 9-5*.
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 9-6*.
12. Choose the desired wake-up condition from *Table 9-6* 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.

9.9.1 Low-Power UART Wake-Up Conditions

Table 9-6 shows the wake-up conditions for low-power UARTs when FDM is enabled (*UARTn_CTRL.fdm* = 1).

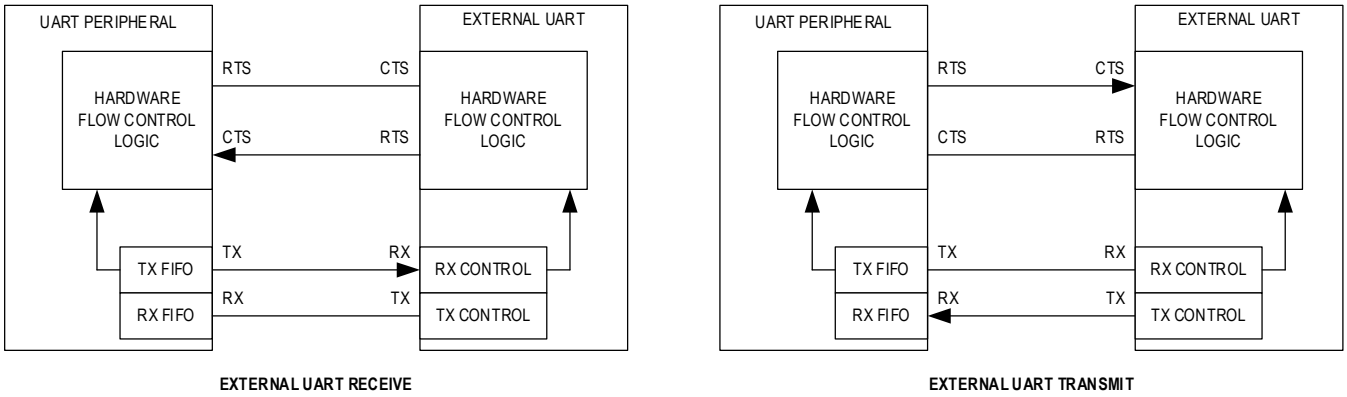
Table 9-6: MAX32670/MAX32671 Wakeup Events

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

9.10 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 9-6*.

Figure 9-6: HFC Physical Connection



A UART transmitter waits for the external device to assert its CTS pin in HFC operation. 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.

The peripheral hardware or software can fully automate HFC by directly monitoring the CTS input signal and controlling the RTS output signal.

9.10.1 Automated HFC

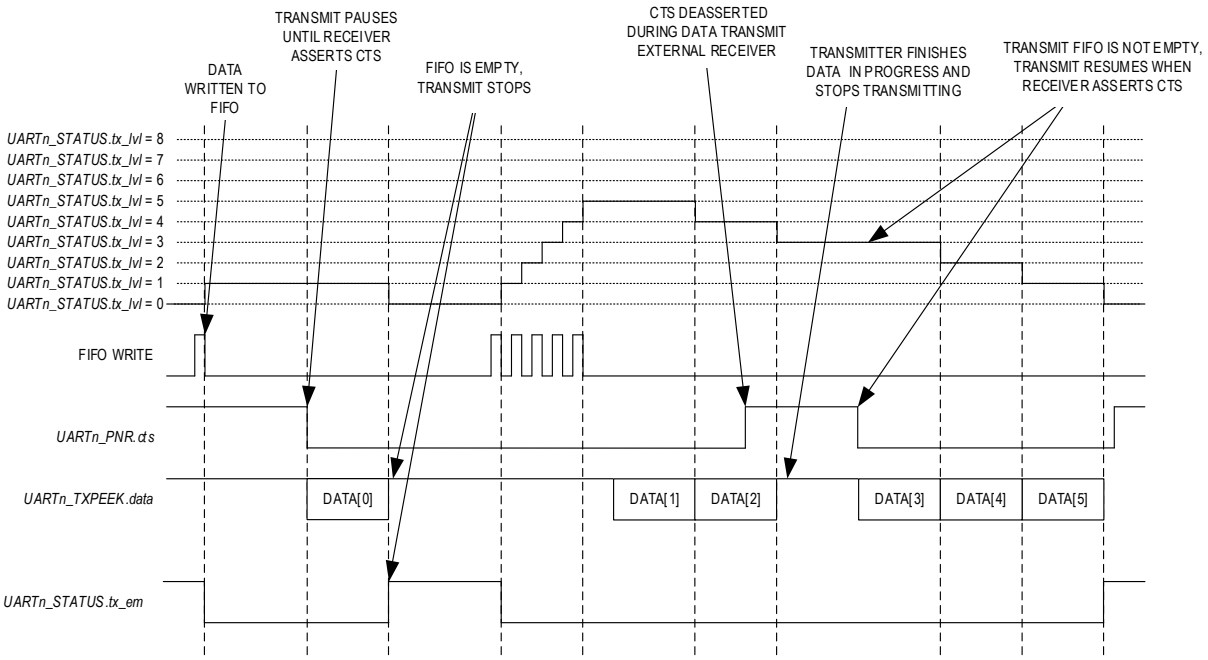
Setting *UARTn_CTRL.hfc_en* = 1 enables automated HFC. When automated HFC is enabled, the hardware manages the CTS and RTS signals. The deassertion of the RTS signal is configurable using the *UARTn_CTRL.rtsdc* field:

- *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 9-7* 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 9-7: HFC Signaling for Transmitting to an External Receiver



9.10.2 Software-Controlled HFC

Software-controlled HFC requires the software to manually control the RTS output pin and monitor the CTS input pin. To use the software-controlled HFC, disable the automated HFC 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.

9.10.2.1 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. The software can enable the CTS interrupt event by setting the *UARTn_INT_EN.cts_ev* field to 1. The hardware sets the CTS signal change interrupt flag 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 whenever 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.

9.11 UART 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 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 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 Wake-up Interrupt Enable Register
[0x0038]	*UARTn_WKFL*	UART Wake-up Interrupt Flag Register

9.11.1 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.	

UART Control				UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
19	bclkrdy	R	0	**Baud Clock Ready** 0: Baud clock not ready. 1: Baud clock ready.	
18	dpfe_en	R/W	0	**Data/Parity Bit Frame Error Detection Enable** LPUART instances only. This field is reserved in standard UART instances. 0: Disable. Do not detect receive frame errors between the start bit and stop bit. 1: Enable. Detect frame errors when receive changes at the center of a bit time.	
17:16	bclksrc	R/W	0	**Baud Clock Source** This field selects the baud clock source. See *Table 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/W1O	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/W1O	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.	

UART Control				UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
6	par_md	R/W	0	**Parity Value Select** 0: Parity calculation is based on the number of 1 bits (mark). 1: Parity calculation is based on the number of 0 bits (space).	
5	par_eo	R/W	0	**Parity Odd/Even Select** 0: Even parity. 1: Odd parity.	
4	par_en	R/W	0	**Transmit Parity Generation Enable** 0: Parity transmission disabled. 1: Parity bit is calculated and transmitted after the last character bit.	
3:0	rx_thd_val	R/W	0	**Receive FIFO Threshold** Valid settings are from 1 to C_RX_FIFO_DEPTH. 0: Reserved. 1: 1. 2: 2. 3: 3. 4: 4. 5: 5. 6: 6. 7: 7. 8: 8. 9 - 15: Reserved.	

Table 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.	

UART Status				UARTn_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
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	tx_ob	R/W	0	**Transmit FIFO Almost Empty** 0: Disabled. 1: Enabled.	
4	rx_thd	R/W	0	**Receive FIFO Threshold Event Interrupt Enable** 0: Disabled. 1: Enabled.	
3	rx_ov	R/W	0	**Receive FIFO Overrun Event Interrupt Enable** 0: Disabled. 1: Enabled.	
2	cts_ev	R/W	0	**CTS Signal Change Event Interrupt Enable** 0: Disabled. 1: Enabled.	
1	rx_par	R/W	0	**Receive Parity Event Interrupt Enable** 0: Disabled. 1: Enabled.	
0	rx_ferr	R/W	0	**Receive Frame Error Event Interrupt Enable** 0: Disabled. 1: Enabled.	

Table 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	tx_ob	R/W1C	0	**Transmit FIFO Almost Empty Interrupt Flag**	

UART Interrupt Flag			UARTn_INT_FL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
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 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	**Over Sampling Rate** LPUARTs with FDM enabled (*UARTn_CTRL.fdm* = 1): 0: 8 × 1: 12 × 2: 16 × 3: 20 × 4: 24 × 5: 28 × 6: 32 × 7: 36 × *Note: If* UARTn_CLKDIV*.clkdiv is less than 16, the hardware samples on every clock cycle and this field's setting is ignored.* For standard UARTs and LPUARTs 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**
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 (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, the hardware returns 0. 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.

UART DMA			UARTn_DMA	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
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 the UART transmit FIFO is ready to receive data from memory.

Table 9-18: UART Wake-up Enable

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

Table 9-19: UART Wake-up Flag Register

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

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 simplicity, I2Cn is used throughout this section to refer to any of the I²C peripherals.

For detailed information on I²C bus operation, refer to Analog Devices Application Note 4024 "SPI/I²C Bus Lines Control Multiple Peripherals" at *https://www.maximintegrated.com/en/app-notes/index.mvp/id/4024*.

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 three 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: MAX32670/MAX32671 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. An acknowledge bit is sent by the receiving device after each byte is transferred. When all necessary data bytes are transferred, a STOP or RESTART condition is sent by the bus controller to indicate the end of the transaction. After the STOP condition is sent, the bus is idle and ready for the next transaction. After a RESTART condition is sent, the same controller begins a new transmission. The number of bytes that can be transmitted per transfer is unrestricted.

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

An acknowledge bit (ACK) is generated by the receiver, whether I²C controller or target, after every byte received by pulling SDA low. The ACK bit is how the receiver tells the transmitter that the byte was successfully received, and another byte might be sent.

A Not Acknowledge (NACK) occurs if the receiver does not generate an ACK when the transmitter releases SDA. A NACK is generated by allowing SDA to float high during the acknowledge time slot. The I²C controller can then either generate a STOP condition to abort the transfer or generate a repeated START condition (i.e., send a START condition without an intervening STOP condition) to start a new transfer.

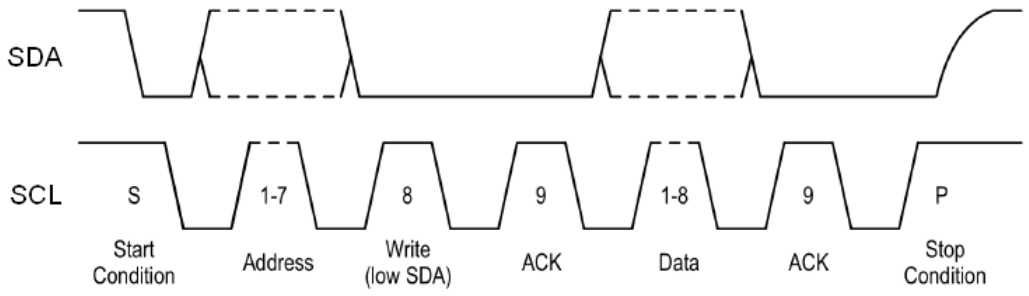
A receiver can generate a NACK after a byte transfer if any of the following conditions occur:

- No receiver is present on the bus with the transmitted address. In that case, no device responds with an acknowledge signal.
- The receiver cannot receive or transmit because it is busy and is not ready to start communication with the controller.
- During the transfer, the receiver receives data or commands it does not understand.
- During the transfer, the receiver is unable to receive any more data.
- If an I²C controller has requested data from a target, it signals the target to stop transmitting by sending a NACK following the last byte it requires.

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 I²C specification states that during data transfer, the SDA line can change state only when SCL is low and that SDA is stable and able to be read when SCL is high, as shown in *Figure 10-1*.

Figure 10-1: I²C Write Data Transfer



An example of an I²C data transfer is as follows:

1. A bus controller indicates a data transfer to a target with a START condition.
2. The controller then transmits one byte with a 7-bit target address and a single read-write bit: a zero for a write or a one for a read.
3. During the next SCL clock following the read-write bit, the controller releases SDA. During this clock period, the addressed target responds with an ACK by pulling SDA low.
4. The controller senses the ACK condition and begins transferring data. If reading from the target, it floats SDA and allows the target to drive SDA to send data. After each byte, the controller drives SDA low to acknowledge the byte. If writing to the target, the controller drives data on the SDA circuit for each of the eight bits of the byte and then floats SDA during the ninth bit to allow the target to reply with the ACK indication.
5. After the last byte is transferred, the controller indicates the transfer is complete by generating a STOP condition. A STOP condition is generated when the controller pulls SDA from low to high while SCL is high.

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

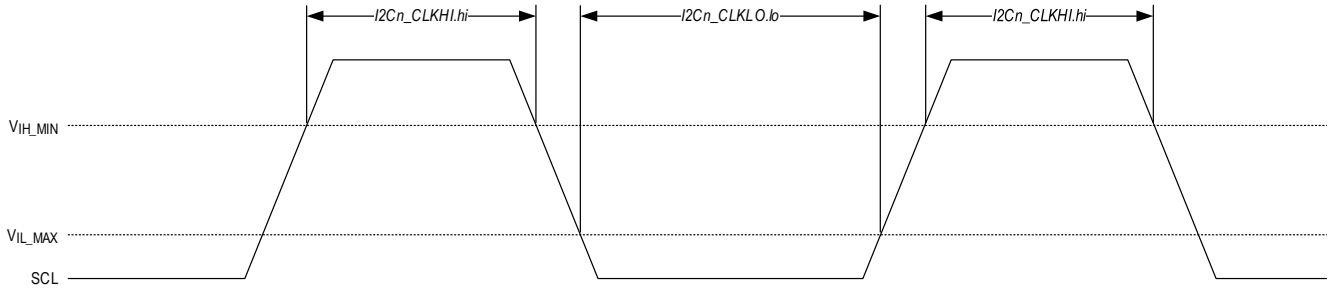
$$t_{SCL\_HI} = t_{I2C\_CLK} \times (I2Cn\_CLKHI.hi + 1)$$

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, a relevant lower speed mode must also be configured. See *SCL Clock Generation for Standard, Fast and Fast-Plus Modes* for information regarding the configuration of lower speed modes.

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
0000	000	0	General Call Address
0000	000	1	START Condition
0000	001	*x*	CBUS Address
0000	010	*x*	Reserved for different bus format
0000	011	*x*	Reserved for future purposes
0000	1*xx*	*x*	HS-mode controller code
1111	1*xx*	*x*	Reserved for future purposes
1111	0*xx*	*x*	10-bit target addressing

In 7-bit addressing mode, the controller sends one address byte. First, to address a 7-bit address target, clear the *I2Cn_MSTCTRL.ex_addr_en* field to 0, then write the address to the transmit FIFO formatted as follows, where A*n* is address A6:A0.

Controller writing to target: 7-bit address : [A6 A5 A4 A3 A2 A1 A0 0]

Controller reading from target: 7-bit address : [A6 A5 A4 A3 A2 A1 A0 1]

In 10-bit addressing mode (*I2Cn_MSTCTRL.ex_addr_en* = 1), the first byte the controller sends is the 10-bit target Addressing byte that includes the first two bits of the 10-bit address, followed by a 0 for the R/W bit. That is followed by a second byte representing the remainder of the 10-bit address. If the operation is a write, this is followed by data bytes to be written to the target. If the operation is a read, it is followed by a repeated START. The software then writes the 10-bit address again with a 1 for the R/W bit. This I²C then starts receiving data from the target device.

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, the following registers should only be configured 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 register fields below can be safely (re)programmed at any time:

- All interrupt flags and interrupt enable bits
- *I2Cn_TXCTRL0.thd_val*
- *I2Cn_RXCTRL0.thd_lvl*
- *I2Cn_TIMEOUT.scl_to_val*
- *I2Cn_DMA.rx_en*
- *I2Cn_DMA.tx_en*
- *I2Cn_FIFO.data*
- *I2Cn_MSTCTRL.start*
- *I2Cn_MSTCTRL.restart*
- *I2Cn_MSTCTRL.stop*

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 target address is transmitted by the controller from the *I2Cn_FIFO* register.
5. The I²C controller receives an ACK from the target, and the controller sets the address ACK interrupt flag (*I2Cn_INTFL0.addr_ack* = 1).
6. The I²C controller receives data from the target and automatically ACKs each byte. The software must retrieve this data by reading the *I2Cn_FIFO* register.
7. Once *I2Cn_RXCTRL1.cnt* data bytes are received, the I²C controller sends a NACK to the target and sets the Transfer Done Interrupt Status Flag (*I2Cn_INTFL0.done* = 1).
8. If *I2Cn_MSTCTRL.restart* or *I2Cn_MSTCTRL.stop* is set, then the I²C controller sends a repeated START or STOP, respectively.

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 the address programmed in the *I2Cn_SLAVE* register, 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 Operation

When in target mode, the I²C peripheral operates as a target device on the I²C bus and responds to an external controller's requests to transmit or receive data. To configure the I²C peripheral as a target, write the *I2Cn_CTRL.mst_mode* bit to zero. The controller drives the I²C clock on the bus, so the SCL device pin is driven by the external controller, and *I2Cn_STATUS.mst_busy* remains a zero. The desired target address must be set by writing to the *I2Cn_SLAVE* register.

For target mode operation, the following register fields should be configured with the I²C peripheral disabled:

- *I2Cn_CTRL.mst_mode* = 0 for target operation.
- I²C target address:
  - Set the target addresses by programming the *I2Cn_SLAVE.addr* field to the desired address for the device on the bus.
  - For extended addresses, set the *I2Cn_SLAVE.ext_addr_en* to 1 for 10-bit addressing or 0 for 7-bit addressing.
- *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_val* and *I2Cn_RXCTRL0.thd_lvl*
  - Transmit and receive FIFO threshold levels.
- *I2Cn_TXCTRL0.tx_ready_mode*
  - Transmit ready (can only be cleared by hardware).
- *I2Cn_TIMEOUT*.scl_to_val
  - Timeout control.
- *I2Cn_DMA.rx_en* and *I2Cn_DMA.tx_en*
  - Transmit and receive DMA enables.
- *I2Cn_FIFO.data*
  - FIFO access register.

10.4.7.1 Target Transmitter

The device operates as a target transmitter when the received address matches the device target address with the R/W bit set to 1. The controller is then reading from the device target. There two main modes of target transmitter operation: just-in-time mode and preload mode.

10.4.7.1.1 Just-in-Time Target Transmitter

In just-in-time mode, the software waits to write the transmit data to the transmit FIFO until after the controller addresses it for a READ transaction, just in time, to send the data to the controller. This allows the software to defer the determination of what data should be sent until the time of the address match. For example, the transmit data could be based on an immediately preceding I^2C write transaction that requests a certain block of data to be sent. The data could represent the latest, most up-to-date value of a sensor reading. Clock stretching *must* be enabled (*I2Cn_CTRL.clkstr_dis* = 0) for just-in-time mode operation.

Program flow for target transmit operation in just-in-time mode is as follows:

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers, including:
   a. Set the *I2Cn_SLAVE*.*addr* field with the desired I^2C target addresses.
   b. Set the *I2Cn_SLAVE*.*ext_addr_en* field for either 7-bit or 10-bit addressing.
   c. Just-in-time mode specific settings:
      i) *I2Cn_CTRL*. *clkstr_dis* = 0
      ii) *I2Cn_TXCTRL0*[5:2] = 0x8
      iii) *I2Cn_TXCTRL0*.*preload_mode* = 0.
   e. Program *I2Cn_CLKHI*.*hi* and *I2Cn_HSCLK*.*hi* with appropriate values satisfying $t_{SU;DAT}$ (and HS $t_{SU;DAT}$).
2. The software sets *I2Cn_CTRL*.*en* = 1.
   a. The controller is now listening for its address. For either a transmit (R/W = 1) or receive (R/W = 0) operation, the peripheral responds to its address with an ACK.
   b. When the address match occurs, the hardware sets *I2Cn_INTFL0*.*addr_match* and *I2Cn_INTFL0*.*tx_lockout*.
3. The software waits for *I2Cn_INTFL0*.*addr_match* to read 1, either through polling the interrupt flag or setting *I2Cn_INTEN0*.*addr_match* to interrupt the CPU.
4. After reading *I2Cn_INTFL0*.*addr_match* =1, the software reads *I2Cn_CTRL*.*read* to determine whether the transaction is a transmit (read = 1) or receive (read = 0) operation. In this case, assume read = 1, indicating transmit.
   a. The hardware holds SCL low until the software clears *I2Cn_INTFL0*.*tx_lockout* and loads data into the FIFO.
5. The software clears *I2Cn_INTFL0*.*addr_match* and *I2Cn_INTFL0*.*tx_lockout*. Now that *I2Cn_INTFL0*.*tx_lockout* is 0, the software can begin loading the transmit data into *I2Cn_FIFO*.
6. As soon as there is data in the FIFO, the hardware releases SCL (after counting out *I2Cn_CLKHI*.*hi*) and sends out the data on the bus.
7. While the controller keeps requesting data and sending ACKs, *I2Cn_INTFL0*.*done* remains 0, and the software should continue to monitor the transmit FIFO and refill it as needed.
   a. The FIFO level can be monitored synchronously through the transmit FIFO status/interrupt flags or asynchronously by setting *I2Cn_TXCTRL0*.*thd_val* and setting the *I2Cn_INTEN0*.*tx_thd* interrupt.
   b. If the transmit FIFO ever empties during the transaction, the hardware starts clock stretching and waits for it to be refilled.
8. The controller ends the transaction by sending a NACK. Once this happens, the *I2Cn_INTFL0*.*done* interrupt flag is set, and the software can stop monitoring the transmit FIFO.
   a. If the software needs to know how many data bytes were transmitted to the controller, it should check the transmit FIFO level as soon as *I2Cn_INTFL0*.*done* = 1 and use it to determine how many data bytes were successfully sent.

      Note: Any data remaining in the transmit FIFO is discarded before the next transmit operation; it is NOT necessary for the software to manually flush the transmit FIFO.
9. The transaction is complete. The software should clear the *I2Cn_INTFL0*.*done* interrupt flag and clear the *I2Cn_INTFL0*.*tx_thd* interrupt flag. Return to step 3, waiting on an address match.

10.4.7.1.2 Preload Mode Target Transmit

The other mode of operation for target transmit is preload mode. In this mode, it is assumed that the software knows before the transmit operation what data it should send to the controller. This data is then "preloaded" into the transmit FIFO. Once the address match occurs, this data can be sent out without any software intervention. Preload mode can be used with clock stretching either enabled or disabled, but it is the only option if clock stretching must be disabled.

To use target transmit preload mode:

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers, including:
   a. Set the *I2Cn_SLAVE*.*addr* field with the desired I^2C target addresses.
   b. Set the *I2Cn_SLAVE*.*ext_addr_en* field for either 7-bit or 10-bit addressing.
   c. Preload mode specific settings:
      i) *I2Cn_CTRL*.*clkstr_dis* = 1
      ii) *I2Cn_TXCTRL0*[5:2] = 0xF
      iii) *I2Cn_TXCTRL0*.*preload_mode* = 1.
2. The software sets *I2Cn_CTRL*.*en* = 1.
   a. Even though the controller is enabled, it does not ACK an address match with R/W equal to 1 until the software sets the *I2Cn_TXCTRL1*.*preload_rdy* field to 1.
3. The software prepares for the transmit operation by loading data into the transmit FIFO, enabling DMA, setting *I2Cn_TXCTRL0*.*thd_val,* and setting *I2Cn_INTEN0*.*tx_thd* interrupt, etc.
   a. If clock stretching is disabled, an empty transmit FIFO during the transmit operation causes a transmit underrun error. Therefore, the software should take any necessary steps to avoid an underrun *before* setting *I2Cn_TXCTRL1*.*preload_rdy* = 1.
   b. If clock stretching is enabled, then an empty transmit FIFO does not cause a transmit underrun error. However, it is recommended to follow the same preparation steps to minimize the amount of time spent clock stretching, which lets the transaction complete as quickly as possible.
4. Once the software has prepared for the transmit operation; it sets *I2Cn_TXCTRL1*.*preload_rdy* = 1.
   a. The controller is now fully enabled and responds with an ACK to an address match.
   b. The hardware sets *I2Cn_INTFL0*.*addr_match* when an address match occurs. *I2Cn_INTFL0*.*tx_lockout* is NOT set to 1 and remains 0.
5. The software waits for *I2Cn_INTFL0*.*addr_match* = 1, either through polling the interrupt flag or by setting *I2Cn_INTEN0*.*addr_match* to generate an interrupt when the event occurs.
6. After seeing *I2Cn_INTFL0*.*addr_match* =1, the software reads *I2Cn_CTRL*.*read* to determine if the transaction is a transmit (read = 1) or receive (read = 0) operation. In this case, assume *I2Cn_CTRL*.*read*, indicating a transmit.
   a. The hardware begins sending out the data that is preloaded into the transmit FIFO.
   b. Once the first data byte is sent, the hardware automatically clears *I2Cn_TXCTRL1*.*preload_rdy* to 0.
7. While the controller keeps requesting data and sending ACKs, *I2Cn_INTFL0*.*done* remains 0, and the software should continue to monitor the transmit FIFO and refill it as needed.
   a. The FIFO level can be monitored synchronously through the transmit FIFO status/interrupt flags or asynchronously by setting *I2Cn_TXCTRL0*.*thd_val* and setting *I2Cn_INTEN0*.*tx_thd* interrupt.
   b. If clock stretching is disabled and the transmit FIFO empties during the transaction, the hardware sets *I2Cn_INTFL1*.*tx_un* = 1 and sends 0xFF for all following data bytes requested by the controller.
8. The controller ends the transaction by sending a NACK, causing the hardware to set the *I2Cn_INTFL0*.*done* interrupt flag.
   a. If the transmit FIFO empties simultaneously that the controller indicates the transaction is complete by sending a NACK, this is not considered an underrun event *I2Cn_INTFL1*.*tx_un* flag remains 0.
   b. If the software needs to know how many data bytes are transmitted to the controller, check the transmit FIFO level when the *I2Cn_INTFL0*.*done* flag is set to 1.

9. The transaction is complete, the software should "clean up," which includes clearing *I2Cn_INTFL0*.*done*. Return to step 3 and prepare for the next transaction.
    a. Any data remaining in the transmit FIFO is not discarded; it is reused for the next transmit operation.
        i) If this is not desired, the software can flush the transmit FIFO. Flush the transmit and receive FIFOs by writing 0 to *I2Cn_CTRL*.*en* and the writing 1 to *I2Cn_CTRL*.*en*.

Once a target starts transmitting from the *I2Cn_FIFO*, detecting an out of sequence STOP, START, or RESTART conditions terminates the current transaction. When a transaction is terminated due to an out of sequence error, *I2Cn_INTFL0*.*start_err* or *I2Cn_INTFL0*.*stop_err* is set to 1.

If the transmit FIFO is not ready (*I2Cn_TXCTRL1*.*preload_rdy* = 0) and the I²C 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 I²C read transaction is after the address byte.

10.4.7.2 Target Receivers

The device operates as a target receiver when the received address matches the device target address with the R/W bit set to 0. The external controller is writing to the target.

Program flow for a receive operation is as follows:

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers, including:
    a. Set the *I2Cn_SLAVE*.*addr* field with the desired I²C target addresses.
    b. Set the *I2Cn_SLAVE*.*ext_addr_en* field for either 7-bit or 10-bit addressing.
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.

10.4.8 Interrupt Sources

The I^2C controller has a very flexible interrupt generator that generates an interrupt signal to the interrupt controller on any of several events. On recognizing the I^2C interrupt, the software determines the cause of the interrupt by reading the I^2C interrupt flags registers *I2Cn_INTFL0* and *I2Cn_INTFL1*. Interrupts can be generated for the following events:

- Transaction Complete (controller/target).
- Address NACK received from target (controller).
- Data NACK received from target (controller).
- Lost arbitration (controller).
- Transaction timeout (controller/target).
- FIFO is empty, not empty, or full to a configurable threshold level (controller/target).
- Transmit FIFO locked out because it is being flushed (controller/target)
- Out of sequence START and STOP conditions (controller/target).
- Sent a NACK to an external controller because the transmit or receive FIFO was not ready (target).
- Address ACK or NACK received (controller).
- Incoming address match (target)
- Transmit underflow or receive overflow (target).

Interrupts for each event can be enabled or disabled by setting or clearing the corresponding bit in the *I2Cn_INTEN0* or *I2Cn_INTEN1* interrupt enable register.

Note: Disabling the interrupt does not prevent the corresponding flag from being set by the hardware but does prevent an interrupt when the interrupt flag is set.

Note: Before enabling an interrupt, the status of the corresponding interrupt flag should be checked and, if necessary, serviced or cleared, preventing a previous interrupt event from interfering with a new I^2C communications session.

10.4.9 Transmit FIFO and Receive FIFO

There are separate transmit and receive FIFOs. Both are accessed using the FIFO data register *I2Cn_FIFO*. Writes to this register enqueue data into the transmit FIFO. Writes to a full transmit FIFO has no effect. Reads from *I2Cn_FIFO* 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*.

10.4.10 Transmit FIFO Preloading

There can be situations during target mode operation where the software wants to preload the transmit FIFO before a transmission, such as when clock stretching is disabled. In this scenario, rather than responding to an external controller requesting data with an ACK and clock stretching while the software writes the data to the transmit FIFO, the hardware responds with a NACK until the software has preloaded the requested data into the transmit FIFO.

When transmit FIFO preloading is enabled, the software controls ACKs to the external controller using the transmit ready (*I2Cn_TXCTRL1.preload_rdy*) bit. When *I2Cn_TXCTRL1.preload_rdy* is set to 0, the hardware automatically NACKs all read

transactions from the controller. Setting *I2Cn_TXCTRL1.preload_rdy* to 1 sends an ACK to the controller on the next read transaction and transmits the data in the transmit FIFO. Preloading the transmit FIFO should be complete before setting the *I2Cn_TXCTRL1.preload_rdy* field to 1.

The required steps for implementing transmit FIFO preloading in software are as follows:

1. Enable the transmit FIFO preloading by setting the field *I2Cn_TXCTRL0.preload_mode* to 1. The hardware automatically clears the *I2Cn_TXCTRL1.preload_rdy* field to 0.
2. If the transmit FIFO lockout flag (*I2Cn_INTFL0.tx_lockout*) is set to 1, write 1 to clear the flag and enable writes to the transmit FIFO.
3. Enable DMA or interrupts if required.
4. Load the transmit FIFO with the data to send when the controller sends the next read request.
5. Set *I2Cn_TXCTRL1.preload_rdy* to 1 to automatically let the hardware send the preloaded FIFO on the next read from a controller.
6. *I2Cn_TXCTRL1.preload_rdy* is cleared by the hardware once it finishes transmitting the first byte, and data is transmitted from the transmit FIFO. Once cleared, the software can repeat the preloading process or disable transmit FIFO preloading.

Note: To prevent the preloaded data from being cleared when the controller tries to read it, the software must at least set I2Cn_TXCTRL0.rd_addr_flush_dis to 1, disabling auto flush on READ address match. The software determines if the other auto flush disable bits should be set. For example, if a controller uses I²C WRITE transactions to determine what data the target should send in the following READ transactions, the software can clear I2Cn_TXCTRL0.wr_addr_flush_dis to 0. When a WRITE occurs, the transmit FIFO is flushed, giving the software time to load the new data. For the READ transaction, the external controller can poll the target address until the new data is loaded and I2Cn_TXCTRL1.preload_rdy is set, at which point the peripheral responds with an ACK.

10.4.11 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).

10.4.12 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.

10.4.13 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).

10.4.14 DMA Control

There are independent DMA channels for each transmit FIFO, and each receive FIFO. DMA activity is triggered by the transmit FIFO (*I2Cn_TXCTRL0.thd_val*) and receive FIFO (*I2Cn_RXCTRL0.thd_lvl*) threshold levels.

When the transmit FIFO byte count (*I2Cn_TXCTRL1.lvl*) is less than or equal to the transmit FIFO threshold level *I2Cn_TXCTRL0.thd_val*, then the DMA transfers data into the transmit FIFO according to the DMA configuration.

The DMA burst size should be set as shown in *Equation 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_val* setting. This fills up the FIFO more frequently but increases internal bus traffic.

When the receive FIFO count (*I2Cn_RXCTRL1.lvl*) is greater than or equal to the receive FIFO threshold level *I2Cn_RXCTRL0.thd_lvl*, the DMA transfers data out of the receive FIFO according to the DMA configuration. The DMA burst size should be set as shown in *Equation 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, as 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 0 Register
[0x0048]	*I2Cn_DMA*	I²C DMA Enable 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.
14:13	-	RO	0	**Reserved**
12	clkstr_dis	R/W	0	**Target Mode Clock Stretching** 0: Enabled. 1: Disabled.

I²C Control			I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
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.
1	mst_mode	R/W	0	**Controller Mode Enable** 0: Target mode enabled. 1: Controller mode enabled.

I²C Control			I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
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 has been accessed for a write (i.e., receive) transaction in target mode, and the address received matches the device target address. 0: No address match. 1: Address match.

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 has been accessed for a read (i.e., transmit) transaction in target mode, and the address received matches the device target address. 0: No address match. 1: Address match.
21:17	-	RO	0	**Reserved**
16	mami	R/W1C	0	**MAMI Interrupt Flag**
15	tx_lockout	R/W1C	0	**Transmit FIFO Locked Interrupt Flag** If set, the transmit FIFO is locked, and writes to the transmit FIFO are ignored. When set, the transmit FIFO is automatically flushed. Writes to the transmit FIFO are ignored until this flag is cleared. Write 1 to clear. 0: transmit FIFO not locked. 1: transmit FIFO is locked, and all writes to the transmit FIFO are ignored.
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 was received.
6	stop	R/ W1C	0	**Target Mode STOP Condition Interrupt Flag** This flag is set by hardware when a STOP condition is detected. Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: Condition occurred.
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.
21:17	-	RO	0	**Reserved**
16	mami	R/W	0	**MAMI Interrupt Enable**
15	tx_lockout	R/W	0	**Transmit FIFO Lock Out Interrupt Enable** 0: Disabled. 1: Enabled.
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 has been detected. 0: START condition not detected. 1: START condition detected.

I²C Interrupt Status Flags 1			I2Cn_INTFL1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
1	tx_un	R/W1C	0	**Target Mode Transmit FIFO Underflow Status Flag** In target mode operation, the hardware sets this flag automatically if the transmit FIFO is empty and the controller requests more data by sending an ACK after the previous byte is transferred. 0: Target mode transmit FIFO underflow condition has not occurred. 1: Target mode transmit FIFO underflow condition occurred.
0	rx_ov	R/W1C	0	**Target Mode Receive FIFO Overflow Status Flag** In target mode operation, the hardware sets this flag automatically when a receive FIFO overflow occurs. Write 1 to clear. Writing 0 has no effect. 0: Target mode receive FIFO overflow event has not occurred. 1: Target mode receive FIFO overflow condition occurred (data lost).

Table 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**

I²C Receive Control 0			I2Cn_RXCTRL0	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
11:8	thd_lvl	R/W	0	**Receive FIFO Threshold Level** Set this field to the required number of bytes to trigger a receive FIFO threshold event. When the number of bytes in the receive FIFO is equal to or greater than this field, the hardware sets the *I2Cn_INTFL0.rx_thd* bit indicating a receive FIFO threshold level event. 0: 0 bytes or more in the receive FIFO causes a threshold event. 1: 1+ bytes in the receive FIFO triggers a receive threshold event (recommended minimum value). … 8: Receive FIFO threshold event only occurs when the receive FIFO is full.
7	flush	R/W1O	0	**Flush Receive FIFO** Write 1 to this field to initiate a receive FIFO flush, clearing all data in the receive FIFO. This field is automatically cleared by the hardware when the receive FIFO flush completes. Writing 0 has no effect. 0: Receive FIFO flush complete or not active. 1: Flush the receive FIFO
6:1	-	RO	0	**Reserved**
0	dnr	R/W	0	**Target Mode Do Not Respond** Target mode operation only. If the device has been addressed for a write operation, and there is still data in the receive FIFO, then: 0: Always respond to an address match with an ACK but always respond to data bytes with a NACK. 1: NACK the address.

Table 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.

I²C Receive Control 1			I2Cn_RXCTRL1	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
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**
11:8	thd_val	R/W	0	**Transmit FIFO Threshold Level** This field sets the level for a transmit FIFO threshold event interrupt. If the number of bytes remaining in the transmit FIFO falls to this level or lower, the interrupt flag *I2Cn_INTFL0.tx_thd* is set, indicating a transmit FIFO threshold event occurred. 0: 0 bytes remaining in the transmit FIFO triggers a transmit FIFO threshold event. 1: 1 byte or fewer remaining in the transmit FIFO triggers a transmit FIFO threshold event (recommended minimum value). … 7: 7 or fewer bytes remaining in the transmit FIFO triggers a transmit FIFO threshold event
7	flush	R/W1O	0	**Transmit FIFO Flush** A transmit FIFO flush clears all remaining data from the transmit FIFO. 0: Transmit FIFO flush is complete or not active. 1: Flush the transmit FIFO. *Note: The hardware automatically clears this bit to 0 after it is written to 1 when the flush is completed.* If *I2Cn_INTFL0.tx_lockout* = 1, then *I2Cn_TXCTRL0.flush* = 1.
6	-	RO	0	**Reserved**
5	nack_flush_dis	R/W	0	**Transmit FIFO received NACK Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Received NACK at the end of a target transmit operation enabled. 1: Received NACK at the end of a target transmit operation disabled. *Note: Upon entering transmit preload mode, the hardware automatically sets this bit to 0. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 0).*

I²C Transmit Control 0			I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
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).*
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**

I²C Transmit Control Register 1			I2Cn_TXCTRL1	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
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**
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**
31:8	-	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.*

I²C Controller Control			I2Cn_MSTCTRL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
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**
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.*

I²C Hs-Mode Clock Control			I2Cn_HSCLK	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
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 0 Register

I²C Target Address			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.
14:10	-	RO	0	**Reserved**
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_SLAVE.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

Table 10-24: I²C DMA Register

I²C DMA			I2Cn_DMA	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**

I²C DMA			I2Cn_DMA	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
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.

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 controller and target modes.

Key features:

- Stereo (2 channel) and mono (left or right channel option) formats.
- Separate DMA channels for transmit and receive.
- Flexible timing
  - Configurable sampling rate from $1/65536$ to 1 of the 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: MAX32670/MAX32671 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: The ERFO must be enabled for controller operation; in target 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 controller mode and must be enabled.

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

Table 11-2: MAX32670/MAX32671 I²S Pin Mapping

I²S Signal	Pin Description	Alternate Function Name (y = A, B, or C)*	Notes
BCLK (SCK)	I²S bit clock	I2S0y_SCK	Also referred to as serial clock
LRCLK (WS)	I²S left/right clock	I2S0y_WS	Also referred to as word select
SDI	I²S serial data input	I2S0y_SDI	
SDO	I²S serial data output	I2S0y_SDO	
* Refer to the device's data sheet pin description table for alternate function mapping to pin numbers.			

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 controller mode, communicating with an external I²S target and an external I²S controller. In controller mode, the peripheral hardware generates the BCLK and LRCLK, and each is output to each target device.

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

Figure 11-1: I²S Controller Mode

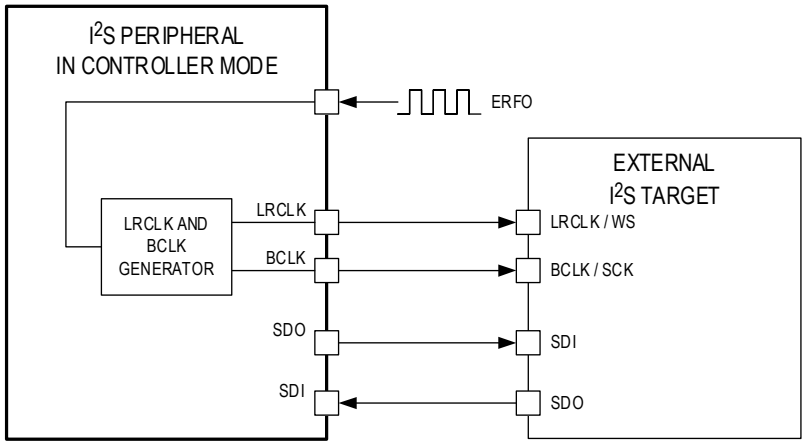
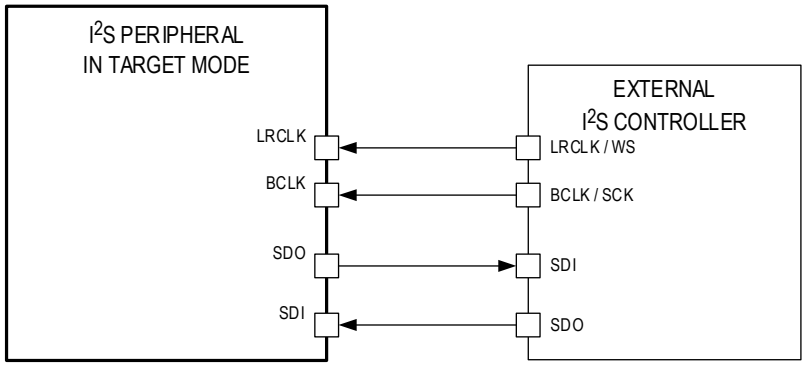


Figure 11-2 shows the I²S peripheral configured for target operation. The LRCLK and BCLK signals are generated externally by the controller and are inputs to the I²S peripheral.

Figure 11-2: I²S Target Mode



11.3 Controller and Target Mode Configuration

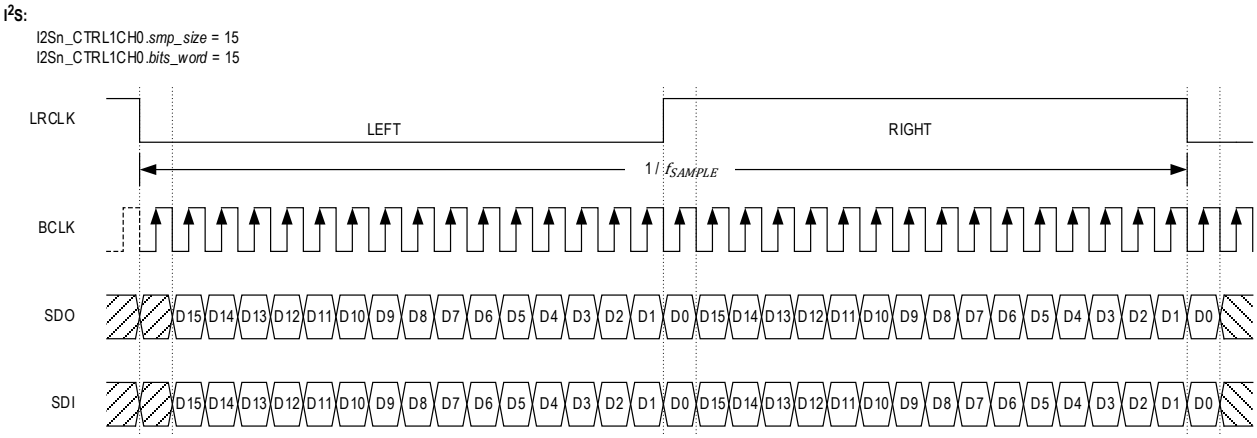
The device supports controller and target modes. In controller mode, the BCLK and LRCLK signals are generated internally and output on the BCLK and LRCLK pins. In target mode, the BCLK and LRCLK pins are configured as inputs, and the external controller's clock source controls the peripheral timing.

Table 11-3: I²S Mode Configuration

Device Mode	*I2S_CTRL0CH0.ch_mode*	LRCLK	BCLK
Controller	0	Output to target	Output to target
Target	3	Input from controller	Input from controller

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 controller, 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 controller mode, the ERFO is used to generate the BCLK and LRCLK signals. Set *GCR_CLKCTRL.erfo_en* to 1 to enable the ERFO.

When the I²S peripheral is configured in target mode, the BCLK and LRCLK pins must be configured as inputs. An external controller generates the BCLK and LRCLK signals, which the peripheral uses to synchronize itself to the I²S bus. *Figure 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 Controller 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 11-2 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 controller mode, the I²S external clock input is used to generate the BCLK frequency. The I²S external clock is divided by the *I2S_CTRL1CH0.clkdiv* field to achieve the target BCLK frequency, as shown in *Equation 11-3*.

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

$$f_{BCLK} = \frac{f_{ERFO}}{(I2S\_CTRL1CH0.clkdiv + 1) \times 2}$$

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

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

$$I2S\_CTRL1CH0.clkdiv = \frac{f_{ERFO}}{2 \times f_{BCLK}} - 1$$

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

$$I2S\_CTRL1CH0.bits\_word = Sample\ Width - 1$$

The LRCLK frequency, or word select frequency, is automatically generated by the I²S peripheral hardware when set to operate as a controller. The LRCLK frequency calculation is shown in *Equation 11-6*.

Equation 11-6: LRCLK Frequency Calculation

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

11.5 Data Formatting

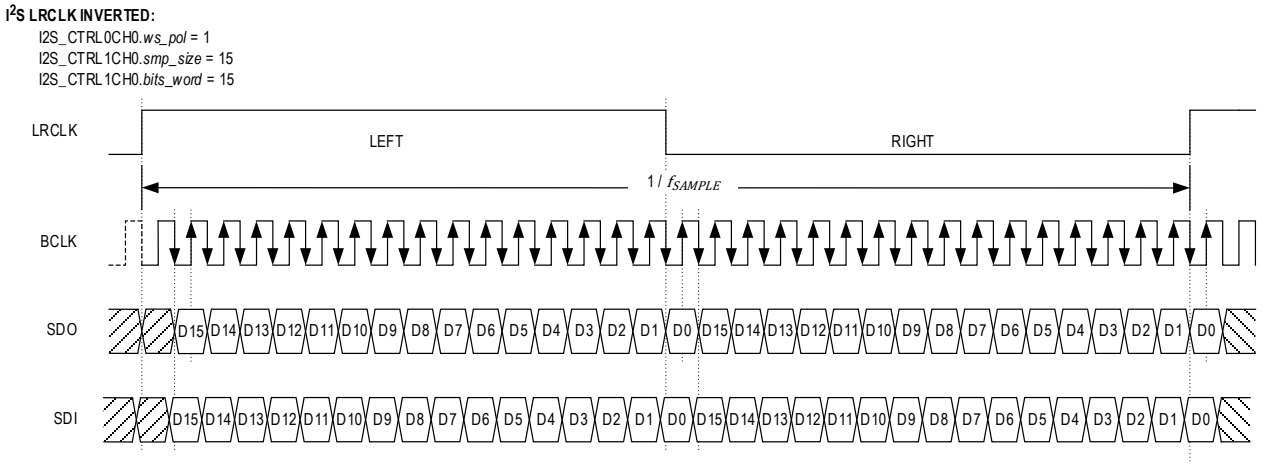
11.5.1 Sample Size

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*.

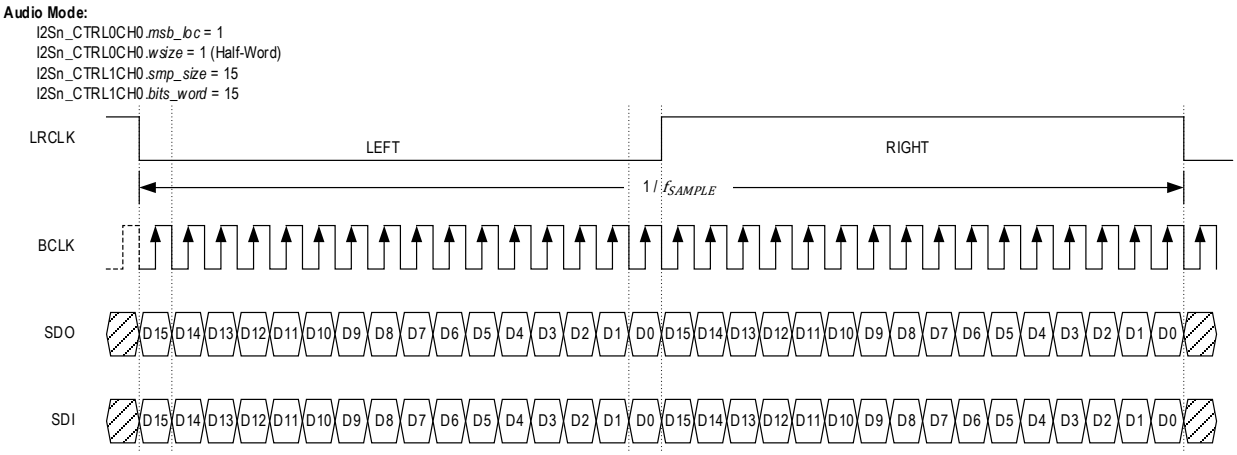
Figure 11-4: Audio Mode with Inverted Word Select Polarity



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 Controller 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.adjst* field to set which bits are stored in the receive FIFO or transmitted from the transmit FIFO, either from the first sample of the SDI/SDO line or the last sample of the SDI/SDO line for the left and right channels. *Figure 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

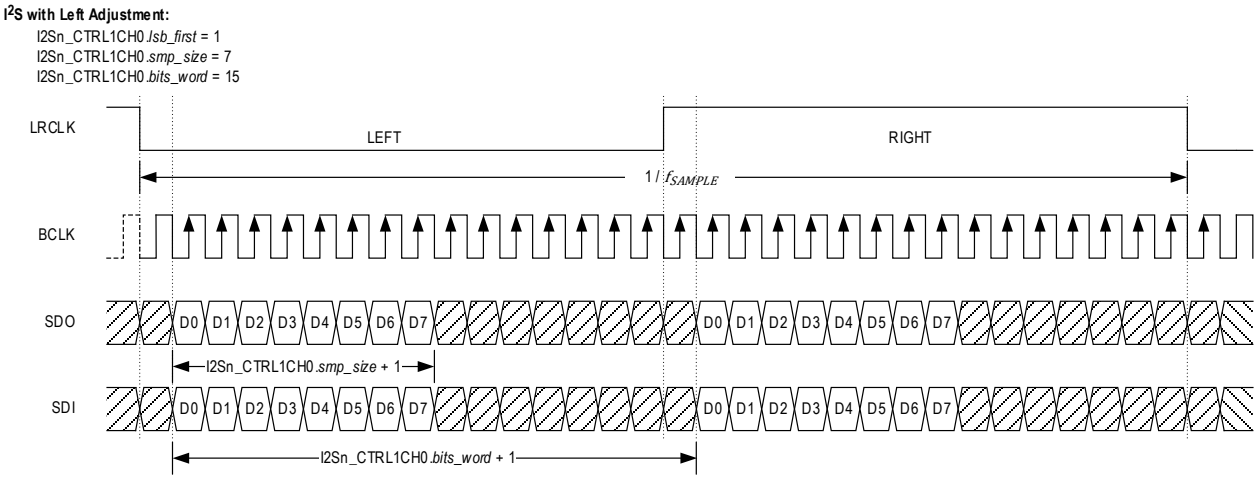
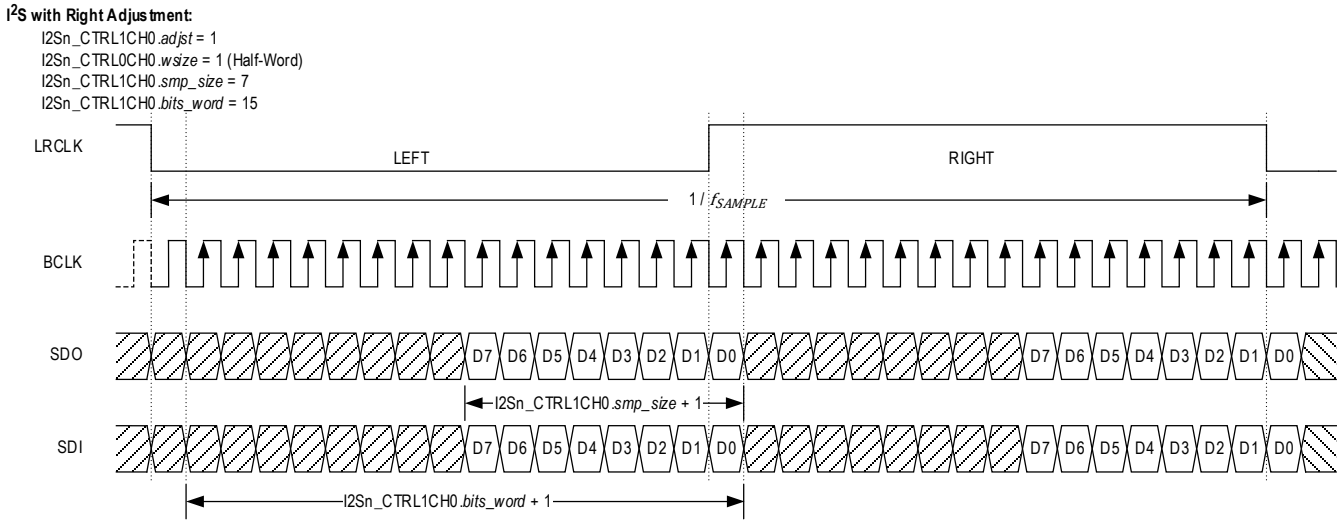


Figure 11-7: LSB Adjustment when Sample Size is Less Than Bits Per Word



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

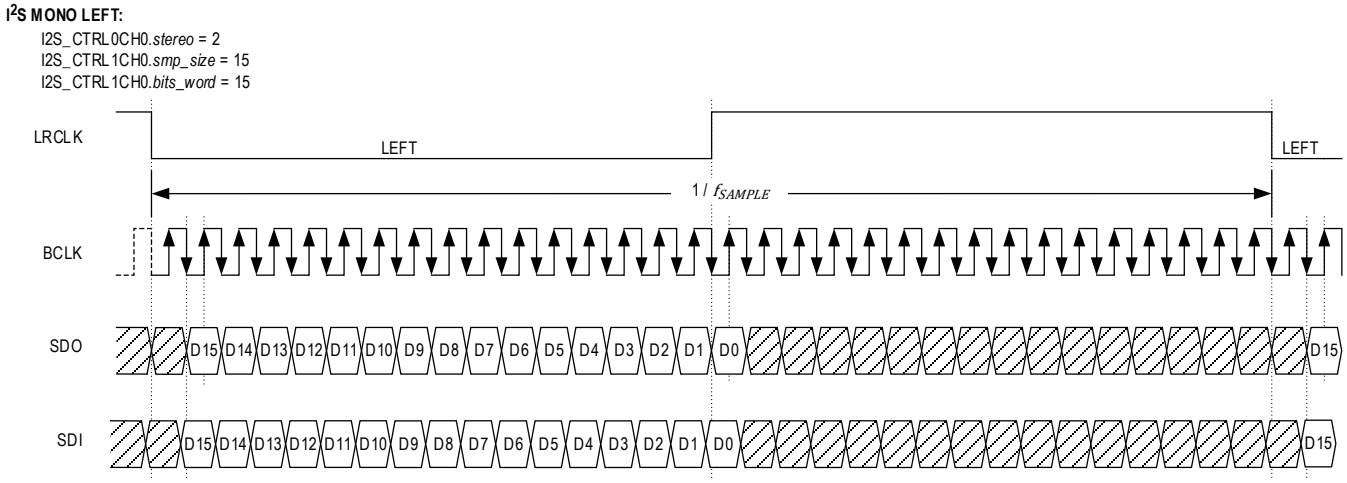
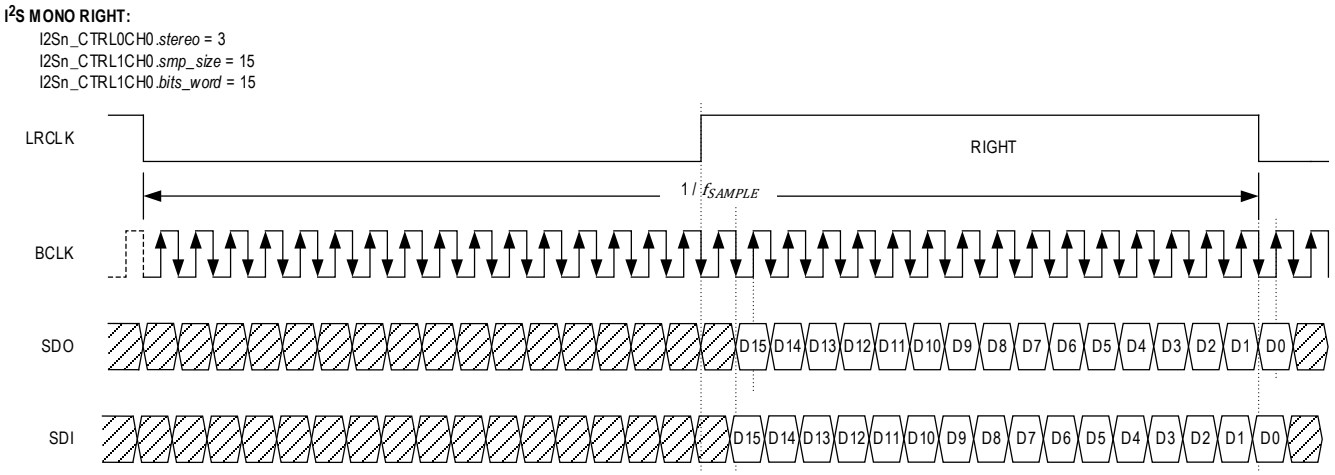


Figure 11-9: I²S Mono Right Mode



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 is written to the transmit FIFO. Use the I²S interrupt flags to monitor the transmit FIFO status and determine when the transfer cycle(s) are complete.

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

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	**MSB**			**LSB**
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 are discarded.
  - Transmit: All bits after the LSB of the FIFO data width are sent as 0.
- If the number of bits per word is less than the FIFO data width:
  - Receive: The data received is stored starting at the MSB of the FIFO entry up to the number of bits per word plus one bit.
  - Transmit: The transmit FIFO data is sent from the LSB to the number of bits plus 1.

Right aligned: *I2S_CTRL0CH0.align* = 1

- If the number of bits per word is greater than the FIFO data width:
  - Receive: The data received is stored in the receive FIFO starting with the LSB up to the FIFO data width, and any additional bits are discarded.
  - Transmit: 0 bits are transmitted for all bits greater than the FIFO data width. For example, if the bits per word field is set to 12 and the FIFO data width is 8, the first 4 bits are transmitted as 0, the 8 bits of data in the FIFO are transmitted.
- If the number of bits per word is less than the FIFO data width:
  - Receive: The data received is sign extended and saved to the receive FIFO.
  - Transmit: The transmit FIFO data is sent from the LSB to the number of bits plus 1.

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 bits / 16	8	8	7	7	0	
16 bits / 32	16	16	15	15	1	
20 bits / 40	20	20	19	19	2	sign
24 bits / 48	24	24	23	23	2	sign
24 bits / 64	32	24	31	23	2	sign
32 bits / 64	32	32	31	31	2	

† *Sign Extension only applies when I2S_CTRL0CH0.align is set to 1 and* I2S_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 is 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

$$I2S\_DMACH0.tx\_lvl < \left(\frac{TX\ FIFO\ DEPTH}{2}\right)$$

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 assigned to the receive and transmit FIFOs. The following describes the behavior of the receive and transmit DMA requests.

- A receive DMA request is asserted when the number of words in the receive FIFO is greater than or equal to the receive FIFO threshold.
- A transmit DMA request is asserted when the number of valid bytes in the transmit FIFO is less than ½ of the transmit FIFO's depth.

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 controller or target configuration.
    a. For controller mode, configure the baud rate by programming the *I2S_CTRL1CH0.clkdiv* field to achieve the required bit rate, set the *I2S_CTRL1CH0.smp_size* field to the desired sample size of the data, and the *I2S_CTRL1CH0.adjst* field if the Sample Size is smaller than the number of bits per word.
6. Configure the threshold of the receive FIFO by programming the *I2S_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 controller 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. 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
[0x0010]	*I2S_CTRL1CH0*	I²S Controller Mode Configuration Register
[0x0030]	*I2S_DMACH0*	I²S DMA Control Channel Register
[0x0040]	*I2S_FIFOCH0*	I²S FIFO Register
[0x0050]	*I2S_INTFL*	I²S Interrupt Status Register
[0x0054]	*I2S_INTEN*	I²S Interrupt Enable Register

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.

I²S Control 0 Register			I2S_CTRL0CH0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
16	tx_en	R/W	0	**Transmit Enable** Enable transmit mode for the I²S peripheral. 0: Disabled. 1: Enabled.
15:14	wsize	R/W	0x3	**Data Size When Reading/Writing FIFO** Set this field to the desired width for data writes and reads from the FIFO. 0: Byte. 1: Half-word (16 bits). 2-3: Word (32 bits).
13:12	stereo	R/W	0	**I²S Mode** Select the mode for the I²S to stereo, mono left channel only, or mono right channel only. 0-1: Stereo. 2: Mono left channel. 3: Mono right channel.
11	-	RO	0	**Reserved**
10	align	R/W	0	**FIFO Data Alignment** Set this field to control the alignment of the data in the FIFOs. This field is only used if the FIFO data width, *I2S_CTRL0CH0.wsize*, is not equal to the bits per word field. 0: MSB. 1: LSB.
9	msb_loc	R/W	0	**First Bit Location Sampling** This field controls when the first bit is transmitted/received in relation to the LRCLK. The first bit is transmitted/received on SDO/SDI on the second complete LRCLK cycle by default. Set this field to 1 to transmit/receive the first bit of data on the first complete LRCLK cycle. 0: Second complete LRCLK cycle is the first bit of the data. 1: First complete LRCLK cycle is the first bit of the data.
8	ws_pol	R/W	0	**LRCLK Polarity Select** This field determines the polarity of the LRCLK signal associated with the left channel data. Set this field to 1 to associate the left channel with the LRCLK high state. The default setting is the standard I²S association. 0: LRCLK low for the left channel. 1: LRCLK high for the left channel.
7:6	ch_mode	R/W	0	**Mode** Set this field to indicate controller or target I²S operation. When using controller mode, the ERFO must be used to generate the LRCLK/BCLK signals. 0: Controller mode, internal generation of LRCLK/BCLK using the ERFO. 1-2: Reserved. 3: Target mode, external generation of LRCLK/BCLK.
5:2	-	DNM	0	**Reserved, Do Not Modify**

I²S Control 0 Register			I2S_CTRL0CH0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
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 Controller Mode Configuration Register

I²S Controller Mode Configuration			I2S_CTRL1CH0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	clkdiv	R/W	0	**I²S Frequency Divisor** Set this field to the required divisor to achieve the desired frequency for the I²S BCLK. See *BCLK Generation for Controller Mode* for detailed information. *Note: This field only applies when the I²S peripheral is set to controller mode, I2S_CTRL0CH0.ch_mode = 0.*
15	adjust	R/W	0	**Data Justification When Sample Size is Less than Bits Per Word** This field is used to determine which bits are used if the sample size is less than the bits per word. 0: Left adjustment. 1: Right adjustment.
14	-	RO	0	**Reserved**
13:9	smp_size	R/W	0	**Sample Size** This field is the desired sample size of the data received or transmitted with respect to the Bits per Word field. In most use cases, the sample size is equal to the bits per word. However, in some situations, fewer bits are required by the application, which allows flexibility. An example use case would be for 16-bit audio being received, and the application only needs 8 bits of resolution. See *Sample Size* for additional details. *Note: The sample size is equal to I2S_CTRL1CH0.bits_word when I2S_CTRL1CH0.smp_size = 0 or I2S_CTRL1CH0.smp_size > I2S_CTRL1CH0.bits_word.*
8	en	R/W	0	**I²S Enable** For controller mode operation, this field is used to start generating the I²S LRCLK and BCLK outputs. In target mode, this field enables the peripheral to begin receiving signals on the I²S interface. 0: Disabled. 1: Enabled.
7:5	-	RO	0	**Reserved**
4:0	bits_word	R/W	0	**I²S Word Length** This field is defined as the I²S data bits per left and right channel. *Example: If the bit clocks is 16 per half frame, bits_word is 15.*

Table 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.

I²S Interrupt Flag			I2S_INTFL	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
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 peripheral is a configurable, flexible, and efficient synchronous interface between multiple SPI devices on a single bus. The SPI bus uses a single clock signal, single, dual, or quad data lines, and one or more target select lines for communication with external SPI devices.

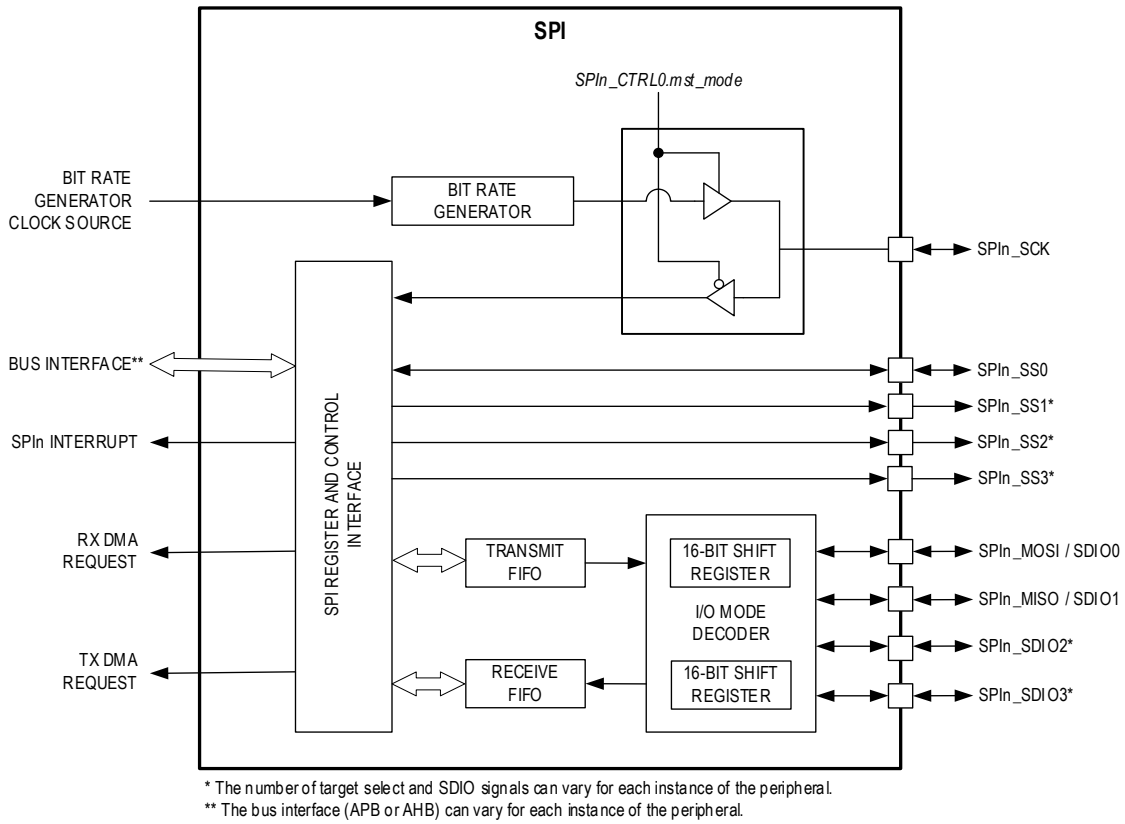
The provided SPI ports support full-duplex, bi-direction I/O, and each SPI includes a bit rate generator (BRG) for generating the clock signal when operating in controller mode. Each SPI port operates independently and requires minimal processor overhead. All instances of the SPI peripheral support both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

Features include:

- Dedicated BRG for precision serial clock generation in controller mode
  - Up to $\frac{f_{PCLK}}{2}$ for instances on the APB bus.
  - Up to $\frac{f_{HCLK}}{2}$ for instances on the AHB bus.
  - Programmable SCK duty cycle timing.
- Full-duplex, synchronous communication of 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.
- 3-wire and 4-wire SPI operation for single-bit communication.
- Single, Dual, or Quad I/O operation.
- 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 the SCK starting edge and ending edge.
- Multi-controller mode fault detection.

Figure 12-1 shows a high-level block diagram of the SPI peripheral. See *Table 12-1* for the peripheral-specific peripheral bus assignment and BRG clock source.

Figure 12-1: SPI Block Diagram



12.1 Instances

There are two instances of the SPI peripheral, as shown in *Table 12-1*.

Table 12-1: MAX32670/MAX32671 SPI Instances

Instance	Formats				Hardware Bus	Bit Rate Generator Clock Source
	3-Wire	4-Wire	Dual	Quad		
SPI0	Yes	Yes	No	No	APB	f_{PCLK}
SPI1	Yes	Yes	No	No	APB	f_{PCLK}
SPI2	Yes	Yes	No	No	APB	f_{PCLK}

Note: Refer to the device data sheet's pin description table for the list of alternate function assignments for each peripheral instance.

12.2 Formats

12.2.1 Four-Wire SPI

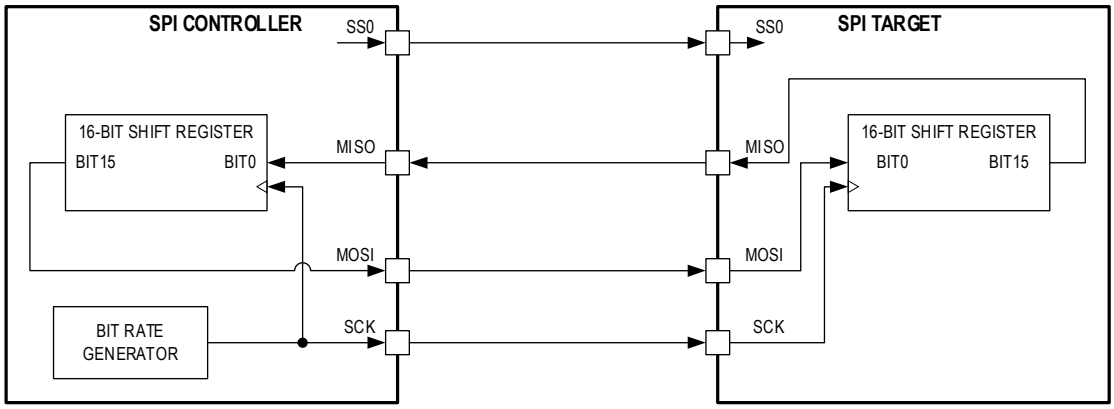
SPI devices operate as either a controller or target device. Four signals are required for communication in four-wire SPI, as shown in *Table 12-2*.

Table 12-2: Four-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The controller generates the SCK signal, an output from the controller, and an input to the target.
MOSI	Controller Output Target Input	This signal is used as an output for sending data to the target in controller mode. 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. This signal is an output for transmitting data to the controller in target mode.
SS	Target Select	This signal is an output used to select a target device before communication in controller mode. Peripherals may have multiple target select outputs to communicate with one or more external target devices. SPIn_SS0 is a dedicated input in target mode that indicates an external controller is starting communication. Other target select signals into the peripheral are ignored in target mode.

The SPI controller starts communication with a target 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 target device is required to put the SPI pins in tri-state mode.

Figure 12-2: 4-Wire SPI Connection Diagram



12.2.2 Three-Wire SPI

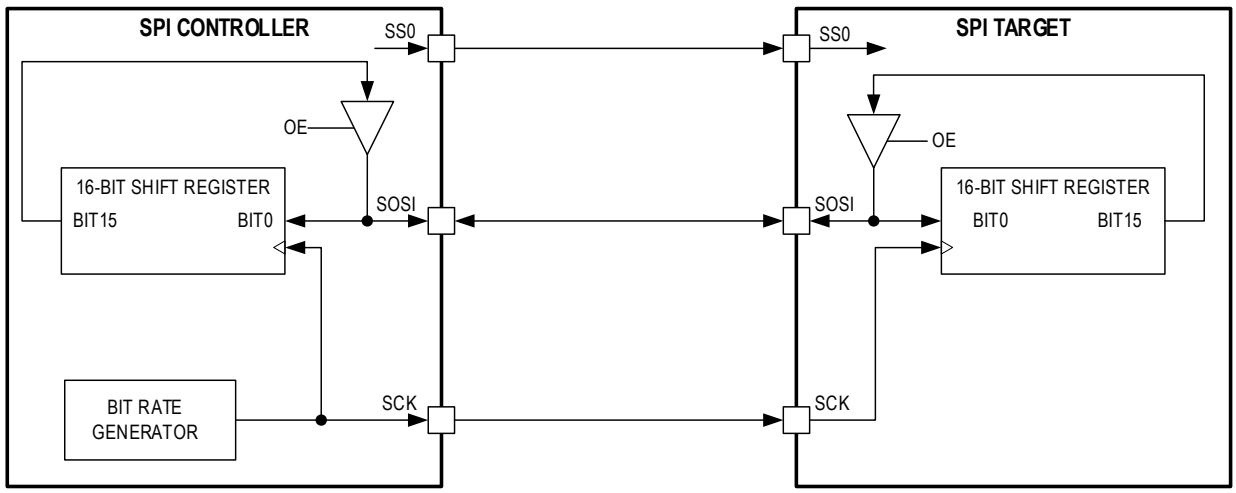
The signals in three-wire SPI operation are shown in *Table 12-3*. The MOSI signal is used as a bidirectional, half-duplex I/O referred to as target input target output (SISO). Three-wire SPI also uses a serial clock signal generated by the controller and a target select pin controlled by the controller.

Table 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	Target Output Target Input	This 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 to 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 is going to start communication. Other target select signals into the target 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 both know the intended direction of the data to prevent bus contention. For a write, the controller drives the data out the SISO pin. For a read, the controller must release the SISO line 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: Generic 3-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 SPI Alternate Function Mapping

Pin selection and configuration are required to use the SPI port. The following information applies to SPI controller and target operation as well as three-wire, four-wire, dual, and quad 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 MAX32670/MAX32671 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 may 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 historical 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 be used for any network.

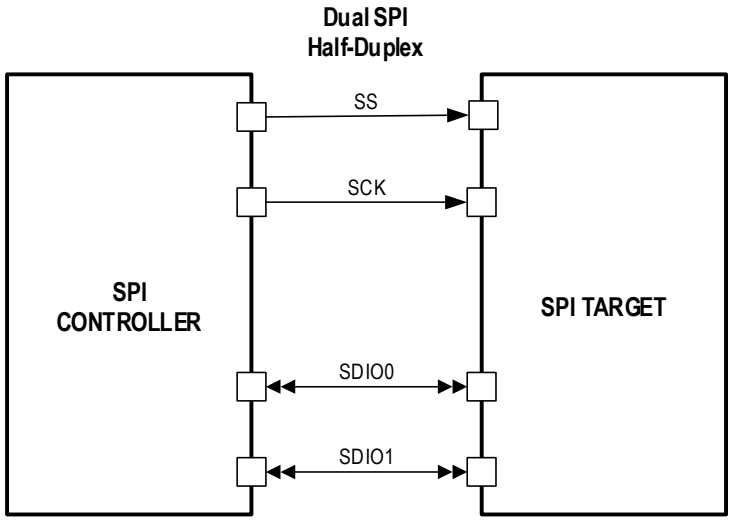
12.3.3 Three-Wire Format Configuration

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 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 are used to transmit 2-bits of data per SCK clock cycle. The communication is half-duplex, and the direction of the data transmission must be known by both the 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, and 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

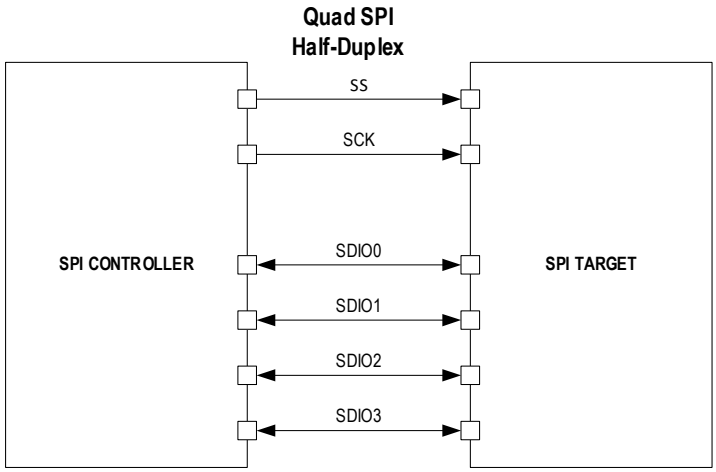


12.3.5 Quad-Mode Format Pin Configuration

Quad-mode SPI uses four I/O pins to transmit four bits of data per transaction. In quad-mode SPI, the communication is half-duplex, and the controller and target must know the direction of transmission for each transaction. Quad-mode SPI uses SCK, SDIO0, SDIO1, SDIO2, SDIO3, and one or more target select pins.

Quad-mode SPI transmits four bits per SCK cycle. Select quad-mode SPI by setting *SPIn_CTRL2.data_width* to 2.

Figure 12-5: Quad 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 an input from the external controller, and the SPI hardware synchronizes communications using the SCK input. Operating as a target, if an 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 has no effect on 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

See *Table 12-1* for the specific input clock, f_{INPUT_CLK}, used for each SPI instance. For SPI instances assigned to the AHB bus, the SPI input clock is the system clock, SYS_CLK. For SPI instances mapped to the APB bus, the SPI input clock is the system peripheral clock, PCLK. The SPI input clock drives the SPI peripheral clock. The SPI provides an internal clock, *SPI_CLK*, that is used within the SPI peripheral for the base clock to control the module and generate the SCK clock when 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

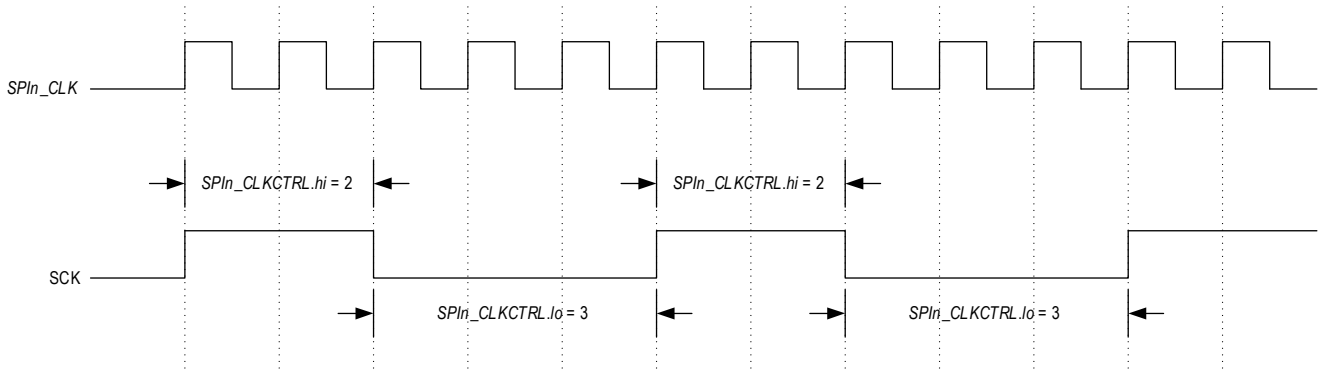
$$f_{SPI\_CLK} = \frac{f_{INPUT\_CLK}}{2^{clkdiv}}$$

12.4.3 Controller Mode Serial Clock Generation

In controller and multi-controller mode, the SCK clock is generated by the controller. The SPI peripheral provides control for both the high time and low time of the SCK clock. This control allows setting the high and low times for the SCK to duty

cycles other than 50% if required. The SCK clock uses the SPI peripheral clock as a base value, and the high and low values are a count of the number of f_{SPI_CLK} clocks. *Figure 12-6* visually represents the use of the *SPIn_CLKCTRL.hi* and *SPIn_CLKCTRL.lo* fields for a non-50% duty cycle serial clock generation. See *Equation 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 12-6: SCK Clock Rate Control



Equation 12-2: SCK High Time

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

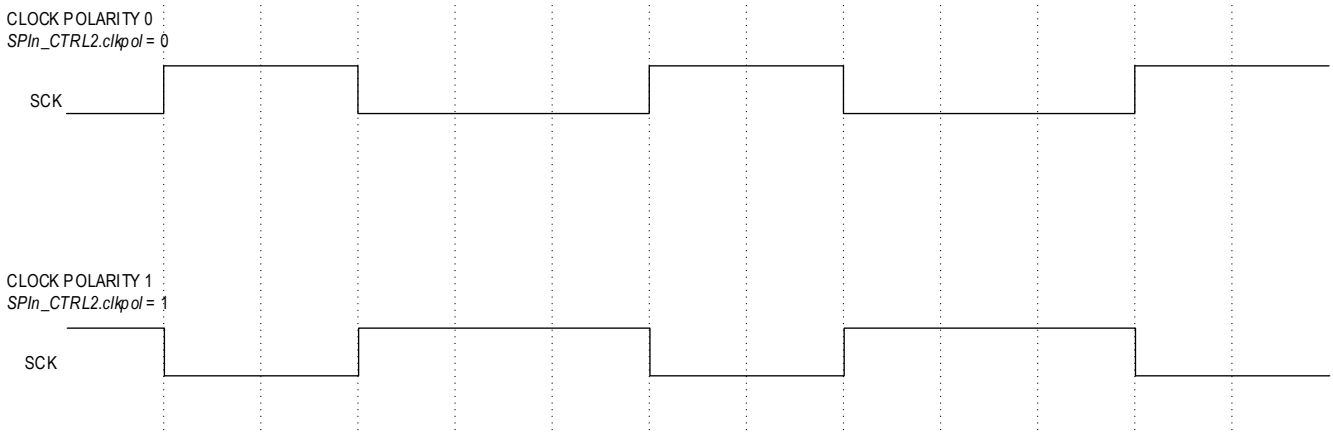
Equation 12-3: SCK Low Time

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

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 bit *SPIn_CTRL2.clkpol* and determines if the clock is active high or active low, as shown in *Figure 12-7*. Clock polarity does not affect the transfer format for SPI. The clock phase determines when the data must be stable for sampling. Setting the clock phase to 0, *SPIn_CTRL2.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 data sample occurring on the second edge of the clock regardless of clock polarity.

Figure 12-7: SPI Clock Polarity



For proper data transmission, the clock phase and polarity must be identical for the SPI controller and target. The controller always places data on the MOSI line a half-cycle before the SCK edge for the target to latch the data.

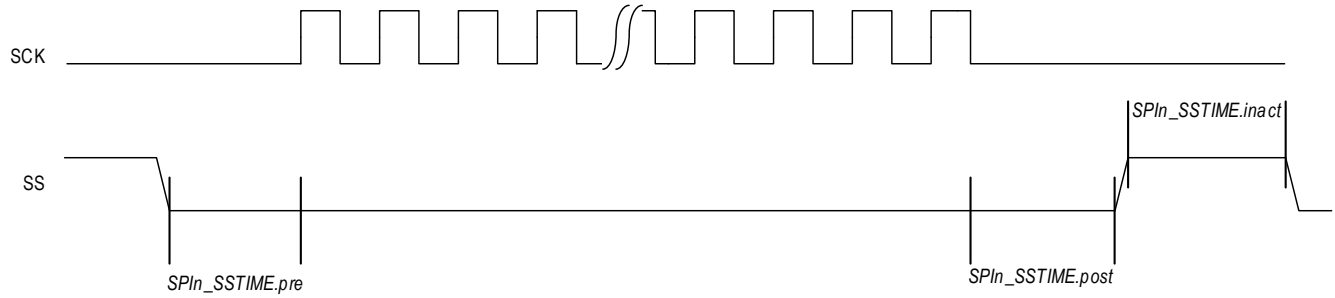
Table 12-4: SPI Modes Clock Phase and Polarity Operation

SPI Mode	SPIn_CTRL2.clkpol	SPIn_CTRL2.clkpha	SCK Transmit Edge	SCK Receive Edge	SCK Idle State	SCLK ≥ 20MHz
0	0	0	Falling	Rising	Low	SPIn_CTRL2.sclk_fb_inv = 1
1	0	1	Rising	Falling	High	SPIn_CTRL2.sclk_fb_inv = 0
2	1	0	Rising	Falling	Low	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 as well as 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 12-8*. 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 12-8: 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 status flag must be cleared by the software by writing a 1 to the interrupt flag.

The following FIFO interrupts are supported:

- Transmit FIFO Empty.
- Transmit FIFO Threshold.
- Receive FIFO Full.
- Receive FIFO Threshold.
- Transmit FIFO Underrun.
  - Target mode only, controller mode stalls the serial clock.
- Transmit FIFO Overrun.
- Receive FIFO Underrun.
- Receive FIFO Overrun.
  - Target mode only, controller mode stalls the serial clock.
- SPI supports interrupts for the internal state of the SPI as well as external signals. The following transmission interrupts are supported:
  - 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 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, as shown in *Table 12-5*. 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 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*

Offset	Register Name	Description
[0x0014]	*SPIn_CLKCTRL*	*SPI Controller Clock Configuration 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 FIFO32 Register

SPI FIFO Data			SPIn_FIFO32	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**SPI FIFO Data Register** This register is used for the SPI Transmit and Receive FIFO. Reading from this register returns characters from the Receive FIFO, and writing to this register adds characters to the Transmit FIFO. Read and write this register in either 1-byte, 2-byte, or 4-byte widths only. Reading from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 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			SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	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	-	R/W	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 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	-	R/W	0	**Reserved.**
5	start	R/W1O	0	**Controller Start Data Transmission** Set this field to 1 to start an SPI controller mode transaction. 0: No controller mode transaction active. 1: Initiate the 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	-	R/W	0	**Reserved**
1	mst_mode	R/W	0	**SPI Controller Mode Enable** This field selects between target mode 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 set the port as an SPI controller. 0: Target mode SPI operation. 1: Controller mode SPI operation.

SPI Control 0			SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
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 Control 1 Register

SPI Transmit Packet Size			SPIn_CTRL1	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	rx_num_char	R	0	**Number of Receive Characters** This field returns the number of characters to receive in receive FIFO. *Note: If the SPI port is set to operate in 4-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	0	**Number of Transmit Characters** This field returns the number of characters to transmit from transmit FIFO. *Note: If the SPI port is set to operate in 4-wire mode, this field is used for both the number of characters to receive and transmit.*

Table 12-11: SPI Control 2 Register

SPI Control 2			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved**
19:16	ss_pol	R/W	0	**Target Select Polarity** Controls the polarity of each individual 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, and Quad SPI, SPIn_CTRL2.data_width =2.*
14	-	R/W	0	**Reserved**

SPI Control 2			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
13:12	data_width	R/W	0b00	**SPI Data Width** This field controls the number of data lines used for SPI communications. *Three-wire SPI*: *data_width* = 0. Set this field to 0, indicating SPIn_MOSI is used for half-duplex communication. *Four-wire full-duplex SPI*: *data_width* = 0. Set this field to 0, indicating SPIn_MOSI and SPIn_MISO are used for the SPI data output and input, respectively. *Dual-mode SPI*: *data_width* = 1. Set this field to 1, indicating SPIn_SDIO0 and SPIn_SDIO1 are used for half-duplex communication. *Quad-mode SPI*: *data_width* = 2. Set this field to 2, indicating SPIn_SDIO0, SPIn_SDIO1, SPIn_SDIO2, and SPIn_SDIO3 are used for half-duplex communication. 0: 1-bit per SCK cycle (Three-wire half-duplex SPI and Four-wire full-duplex SPI). 1: 2-bits per SCK cycle (Dual mode SPI). 2: 4-bits per SCK cycle (Quad mode SPI). 3: Reserved. *Note: When this field is set to 0, use the field 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 and Quad 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.
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			SPIn_SSTIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	R/W	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 next 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** Set this field to the number of system clock cycles for SS to remain 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).*
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 Configuration Registers

SPI Controller Clock Configuration			SPIn_CLKCTRL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved**

SPI Controller Clock Configuration			SPIn_CLKCTRL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
19:16	clkdiv	R/W	0	**SPI Peripheral Clock Scale** Scales the SPI input clock by 2^{clkdiv} to generate the SPI peripheral clock. See *Table 12-1* for details of the SPI input clock for each instance. $f_{SPInCLK} = \frac{f_{SPIn\_INPUT\_CLK}}{2^{clkdiv}}$ Valid values for scale are 0 to 8 inclusive. Values greater than 8 are reserved. *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 - 15: The number of SPI peripheral clocks, $f_{SPInCLK}$, that SCK is high. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*
7:0	lo	R/W	0	**SCK Low Clock Cycles Control** This field controls the SCK low clock time and is used to control 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			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.
30:24	dma_rx_en	R	0	**Number of Bytes in the Receive FIFO** Read returns the number of bytes currently in the receive FIFO.
23	rx_flush	R/W1O	-	**Clear the Receive FIFO** 1: Clear the receive FIFO and any pending receive FIFO flags in *SPIn_INTFL*. This should be done when the receive FIFO is inactive. Note: Writing a 0 has no effect.
22	rx_fifo_en	R/W	0	**Receive FIFO Enabled** 0: Disabled. 1: Enabled.
21	-	R/W	0	**Reserved**

SPI DMA Control		SPIn_DMA		[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
20:16	rx_thd_val	R/W	0	**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 (*SPIn_DMA.dma_tx_en* = 1), and *SPIn_INTFL.rx_thd* is set. Valid values are 0 to 30. *Note: 31 is an invalid setting.*
15	dma_tx_en	R/W	0	**Transmit DMA Enable** 0: Disabled. Any pending DMA requests are cleared. 1: Transmit DMA is enabled.
14:8	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	-	R/W	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 (*SPIn_DMA.dma_tx_en* = 1), and *SPIn_INTFL.tx_thd* becomes set.

Table 12-15: SPI Interrupt Status Flags Registers

SPI Interrupt Status Flags			SPIn_INTFL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	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, and the transmit FIFO is full.
11	mst_done	R/W1C	0	**Controller Data Transmission Done Flag** Set if SPI is in controller mode and all transactions are complete. *SPIn_CTRL1.tx_num_char* has been reached.

SPI Interrupt Status Flags			SPIn_INTFL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
10	-	R/W	0	**Reserved**
9	abort	R/W1C	0	**Target Mode Transaction Abort Detected Flag** Set if the SPI is in target mode, and SS is deasserted before a complete character is received.
8	fault	R/W1C	0	**Multi-Controller Fault Flag** 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	-	R/W	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_lvl*. Cleared once receive FIFO level drops below *SPIn_DMA.rx_lvl.*
1	tx_em	R/W1C	1	**Transmit FIFO Empty Flag** This field is set to 1 by hardware if the transmit FIFO is empty.
0	tx_thd	R/W1C	0	**Transmit FIFO Threshold Level Crossed Flag** This field is set to 1 by hardware when the transmit FIFO is less than the value in *SPIn_DMA.tx_lvl*. This field is cleared by hardware once transmit FIFO level exceeds *SPIn_DMA.tx_lvl.*

Table 12-16: SPI Interrupt Enable Registers

SPI Interrupt Enable			SPIn_INTEN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15	rx_un	R/W	0	**Receive FIFO Underrun Interrupt Enable** 0: Disabled. 1: Enabled.
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	-	R/W	0	**Reserved**

SPI Interrupt Enable			SPIn_INTEN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
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	-	R/W	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			SPIn_WKFL	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved**
3	rx_full	R/W1C	0	**Wake on Receive FIFO Full Flag** 0: Normal operation. 1: Wake condition occurred.
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			SPIn_WKEN	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved**
3	rx_full	R/W	0	**Wake On Receive FIFO Full Enable** 0: Wake event is disabled. 1: Wake event is enabled.
2	rx_thd	R/W	0	**Wake On Receive FIFO Threshold Level Crossed Enable** 0: Wake event is disabled. 1: Wake event is enabled.
1	tx_em	R/W	0	**Wake On Transmit FIFO Empty Enable** 0: Wake event is disabled. 1: Wake event is enabled.
0	tx_thd	R/W	0	**Wake On Transmit FIFO Threshold Level Crossed Enable** 0: Wake event is disabled. 1: Wake event is enabled.

Table 12-19: SPI Target Select Timing Registers

SPI Status			SPIn_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	R/W	0	**Reserved**

SPI Status			SPIn_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
0	busy	R	0	**SPI Active Status** This field returns the SPI status. 0: SPI is not active. In controller mode, the *busy* flag is cleared when the last character is sent. In target mode, the *busy* field is cleared when the configured target select input is deasserted. 1: SPI is active. In controller mode, the *busy* flag is set when a transaction starts. In target mode, the *busy* 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 bit 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 / PWM differential.
- 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*. TMR instances can operate in dual 16-bit mode or cascaded 32-bit mode if supported. *LPTMR* instances provide a single 32-bit timer and can select clock sources that are available in *DEEPSLEEP* and *BACKUP*.

Table 13-1: MAX32670/MAX32671 TMR/LPTMR

Instance	Register Access Name	Single 32-bit Mode	Cascade 32-bit Mode	Dual 16-bit Mode	Operating Modes	CLK0	CLK1	CLK2	CLK3
TMR0	TMR0	No	Yes	Yes	*ACTIVE SLEEP*	PCLK	EXT_CLK1	BRO	ERFO
TMR1	TMR1								
TMR2	TMR2								
TMR3	TMR3								
LPTMR0	TMR4	Yes	No	No	*ACTIVE SLEEP*	AOD_CLK	EXT_CLK2	ERTCO	INRO[1]
					DEEPSLEEP BACKUP	N/A	EXT_CLK2	ERTCO	INRO[1]
LPTMR1	TMR5	Yes	No	No	*ACTIVE SLEEP*	AOD_CLK	EXT_CLK2	ERTCO	INRO[1]
					DEEPSLEEP BACKUP	N/A	EXT_CLK2	ERTCO	INRO[1]
1. INRO accuracy varies up to ±50% across temperature and voltage.									

Table 13-2: MAX32670/MAX32671 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	LPTMR0 Input Pin
LPTMR1	LPTMR1 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 the timer period condition. Each timer mode has a different response at the end of a timer period, which can include changing the state of the timer pin, capturing a timer value, reloading *TMRn_CNT* 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 may 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	Cascade 32-Bit 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 timer by reading and writing to the timer's registers. External events on timer pins are asynchronous events to the timer's clock frequency. The external events are latched on the next rising edge of the timer's clock. Since it is not possible to externally synchronize to the timer's internal clock, input events may require up to 50% of the timer's internal clock cycle before the hardware recognizes the event.

The software must configure the timer's clock source by performing the following steps:

1. Disable the timer peripheral.
   a. Clear *TMRn_CTRL0.en* to 0 to disable the timer.
   b. Read the *TMRn_CTRL1.clken* field until it returns 0, confirming the timer peripheral is disabled.
2. Set *TMRn_CTRL1.clksel* to the new desired clock source.
   Note: In cascade 32-bit mode, the TMRn_CTRL1.clksel_a and TMRn_CTRL1.clksel_b fields must be set to the same clock source for proper operation.
3. Set the desired clock divider by setting the *TMRn_CTRL0.clkdiv* field.
4. Configure the timer for the desired operating mode. See *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.

Disable the timer peripheral while changing any of the configuration registers in the peripheral.

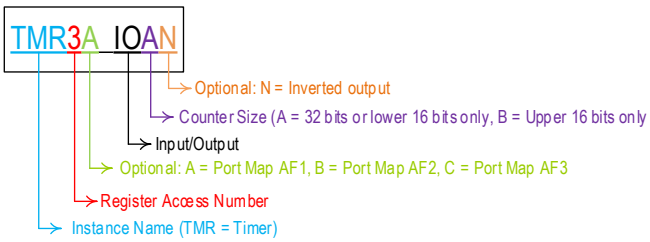
13.5 Timer Pin Functionality

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

The physical pin location of the timer input and/or output signals may vary between packages. The timer functionality, however, is always expressed on the same GPIO pin in the same alternate function mode.

The timer pin functionality is mapped as an alternate function that is shared with a GPIO. When the timer pin alternate function is enabled, the timer pin has the same electrical characteristics, such as pullup/pulldown strength, drive strength, 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 Pins* chapter for details on how to configure the electrical characteristics for the pin.

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: MAX32670/MAX32671 TimerA Output Functionality, Modes 0/1/3/5

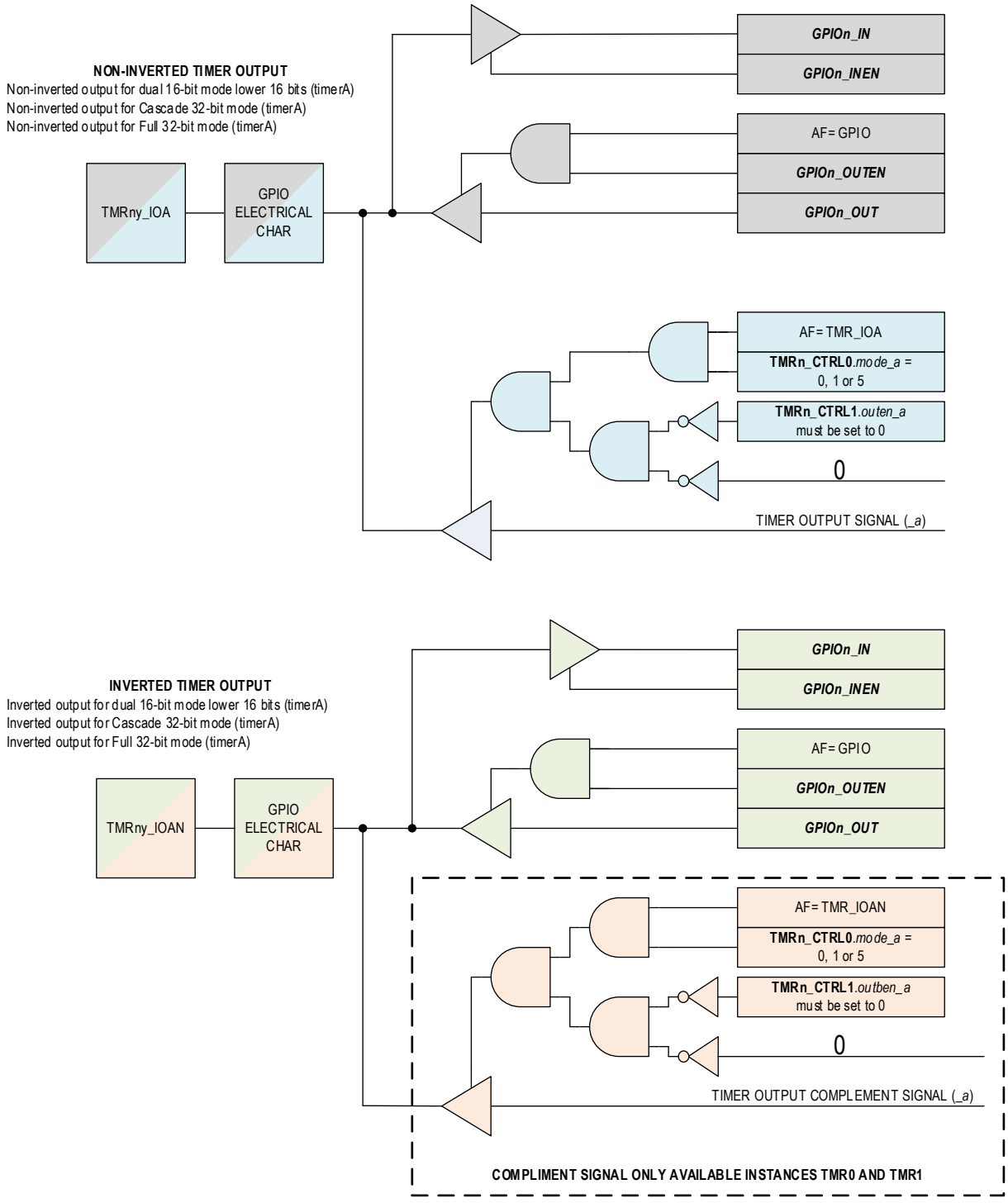
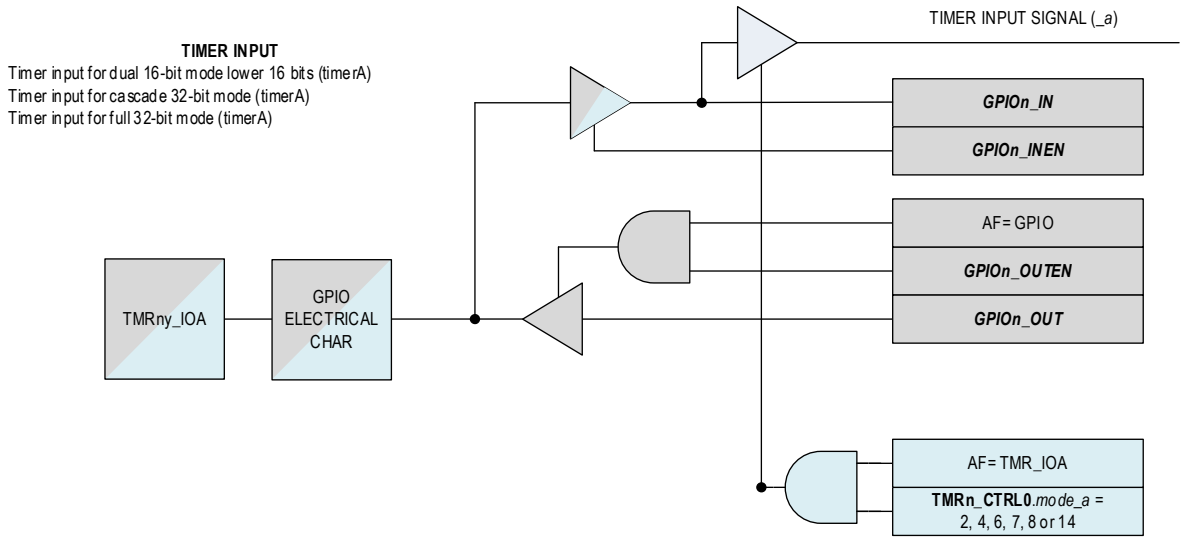


Figure 13-3: MAX32670/MAX32671 TimerA Input Functionality, Modes 2/4/6/7/8/14



13.6 Wakeup Events

The system clock may be turned off in low-power modes 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 wake-up events for the timer.
6. If desired, enable the timer interrupt and provide a TMRn_IRQn interrupt handler 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 wake-up event.

13.7 LPTMR Wakeup Events

LPTMR instances can continue to run if they are configured to run from the clock sources shown in *Table 13-1*. In this case, a wake-up event can be enabled to wake up the clock control logic and return the device to *ACTIVE*.

Each LPTMR clock must be enabled, and if using a LPTMR input or output pin, the LPTMR must also be configured for operation in *DEEPSLEEP* or *BACKUP*.

Table 13-4: MAX32670/MAX32671 Low-Power Timer Pin Configuration for DEEPSLEEP and BACKUP

Timer	Input Pin Enable	Output Pin Enable	Clock Disable
LPTMR0	*MCR_LPPIOCTRL.lptmr0_i*	*MCR_LPPIOCTRL.lptmr0_o*	*MCR_CLKDIS.lptmr0*
LPTMR1	*MCR_LPPIOCTRL.lptmr1_i*	*MCR_LPPIOCTRL.lptmr0_o*	*MCR_CLKDIS.lptmr1*

Low-power timer programming sequence example:

1. Disable the timer peripheral and set the timer clock source as described in *Timer Clock Sources*.
   a. For operation in *DEEPSLEEP* and *BACKUP*, select either the *ERTCO* or *INRO*.
2. Configure the timer operating mode as described in the section *Operating Modes*.
3. If using a timer input or output pin during low-power modes, set the corresponding enable bit shown in *Table 13-4*.
4. If using the timer during low-power modes, enable the timer's low-power clock by writing 0 to either the *MCR_CLKDIS*.*lptmr0* field or the *MCR_CLKDIS*.*lptmr1* field.
   Note: The low-power timer's clock must be disabled to return the timer's input and output to standard GPIO.
5. Enable the timer by setting *TMRn_CTRL0*.*en* to 1.
6. Poll *TMRn_CTRL1*.*clkrdy* until it reads 1.
7. Set the *TMRn_CTRL1*.*we* field to 1 to enable wake-up events for the timer.
8. If desired, enable the timer interrupt and provide a TMRn_IRQn for the timer.
9. Enter a low-power mode as described in the *Operating Modes* section.
10. When the device wakes up from the low-power mode, check the *PWRSEQ_LPPWKST* register to determine if the timer caused the wake-up event.

Table 13-5: MAX32670/MAX32671 Low-Power Timer Wake-up Events

Condition	Peripheral Wake-up Flag *TMRn_INTFL*	Peripheral Wake-up Enable	Low-Power Peripheral Wake-up Flag	Low-Power Peripheral Wake-up Enable	Power Management Wake-up Enable
Any event for LPTMR0	*irq_a*	N/A	*PWRSEQ_LPPWKST*.*lptmr0*	*PWRSEQ_LPPWKEN*.*lptmr0*	*GCR_PM*.*lptmr0_we*
Any event for LPTMR1	*irq_a*	N/A	*PWRSEQ_LPPWKST*.*lptmr1*	*PWRSEQ_LPPWKEN*.*lptmr1*	*GCR_PM*.*lptmr1_we*

13.8 Operating Modes

Multiple operating modes are supported. The availability of some operating modes is dependent on the device and package-specific implementation of the external input and output signals. Refer to the device data sheet for I/O signal configurations and alternate functions for each timer instance.

In *Table 13-6* and *Table 13-7*, 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 13-6: MAX32670/MAX32671 Operating Mode Signals for Timer 0 through Timer 3

Timer Mode	TMR0/TMR1/TMR2/TMR3	I/O Signal Name[†]	Pin Required
One-Shot Mode (0)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional
Continuous Mode (1)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional

Timer Mode	TMR0/TMR1/TMR2/TMR3	I/O Signal Name†	Pin Required
Counter Mode (2)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
PWM Mode (3)	TimerA Output Signal	TMR*ny*_IOA	Yes
	TimerB Output Signal	TMRny_IOB	Yes
Capture Mode (4)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Compare Mode (5)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional
Gated Mode (6)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Capture/Compare Mode (7)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Dual-Edge Capture Mode (8)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Reserved (9 - 13)	-	-	-
Inactive Gated Mode (14)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
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 13-7: MAX32670/MAX32671 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*_IOB	Optional
Continuous Mode (1)	TimerA Output Signal	LPTMR*ny*_IOB	Optional
Counter Mode (2)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
PWM Mode (3)	TimerA Output Signal	LPTMR*ny*_IOB	Yes
Capture Mode (4)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Compare Mode (5)	TimerA Output Signal	LPTMR*ny*_IOB	Optional
Gated Mode (6)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Capture/Compare Mode (7)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Dual-Edge Capture Mode (8)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Reserved (9 - 13)	-	-	-
Inactive Gated Mode (14)	TimerA Input Signal	LPTMR*ny*_IOB	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.*

13.8.1 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 source 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:

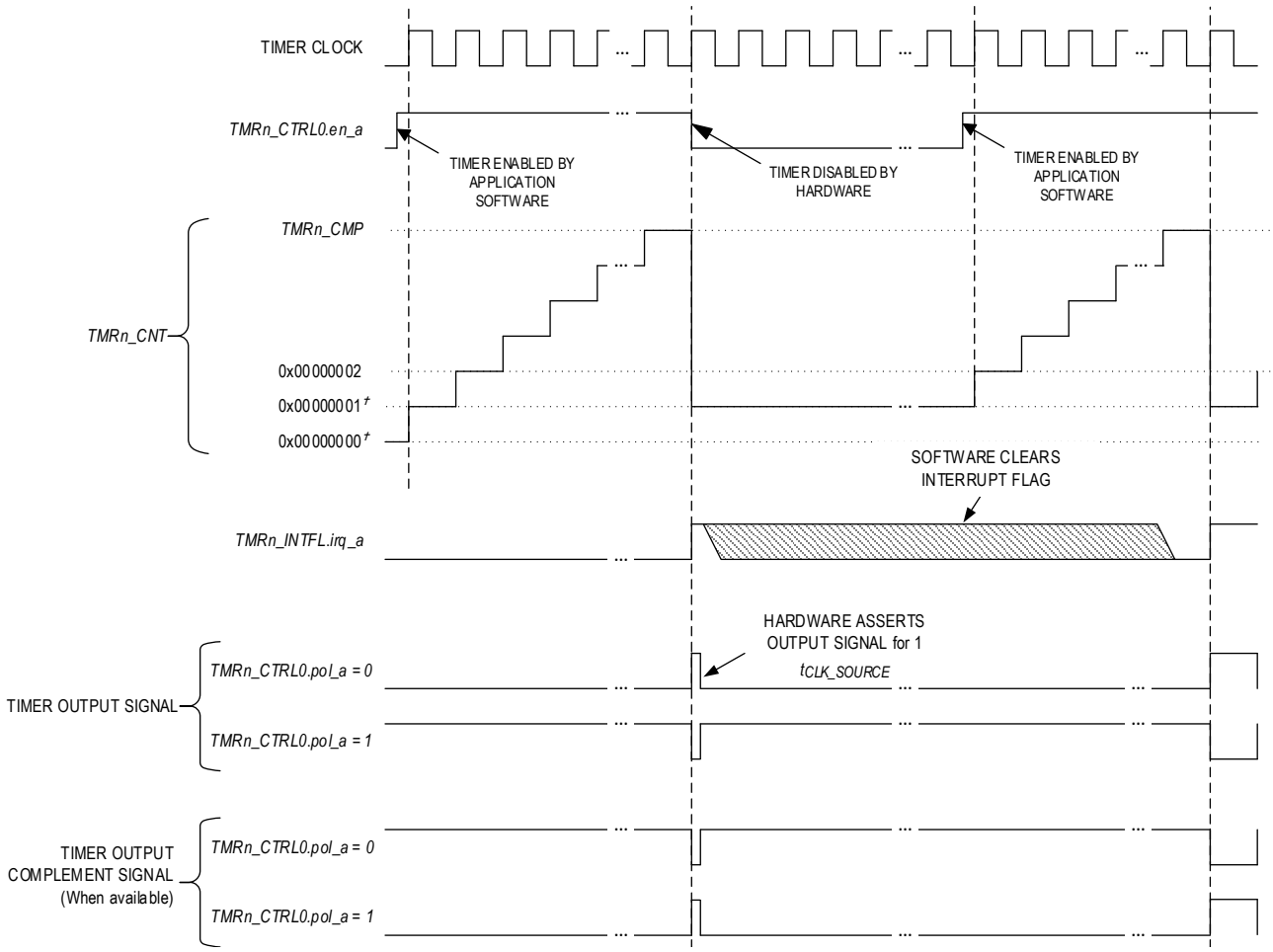
- The *TMRn_CNT* register is set to 0x0000 0001.
- The timer is disabled (*TMRn_CTRL0.en* = 0).
- The timer output, if enabled, is driven to its active state for one timer clock period.
- The *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

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

Equation 13-2: One-Shot Mode Timer Period

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

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.clkdiv* field to set the prescaler for the required timer frequency.
4. If using the timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
5. Or, if using the inverted timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.
6. If using the timer interrupt, enable corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to the *TMRn_CMP* register.
8. If desired, write an initial value to *TMRn_CNT* register.
   a. The initial value only affects the first period; subsequent timer periods always reset the *TMRn_CNT* register to 0x0000 0001.
   b. Read the *TMRn_INTFL.wrdone* field until it reads 1.

   Note: The TMRn_CNT register is only writable if the timer clock is enabled (TMRn_CTRL0.clken = 1).
9. Enable the timer by writing 1 to the *TMRn_CTRL0.en* field.
   a. Read the *TMRn_CTRL0.en* field until it returns 1 to confirm the timer is enabled.

13.8.2 Continuous Mode (1)

In continuous mode, the *TMRn_CNT* register increments until it matches 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:

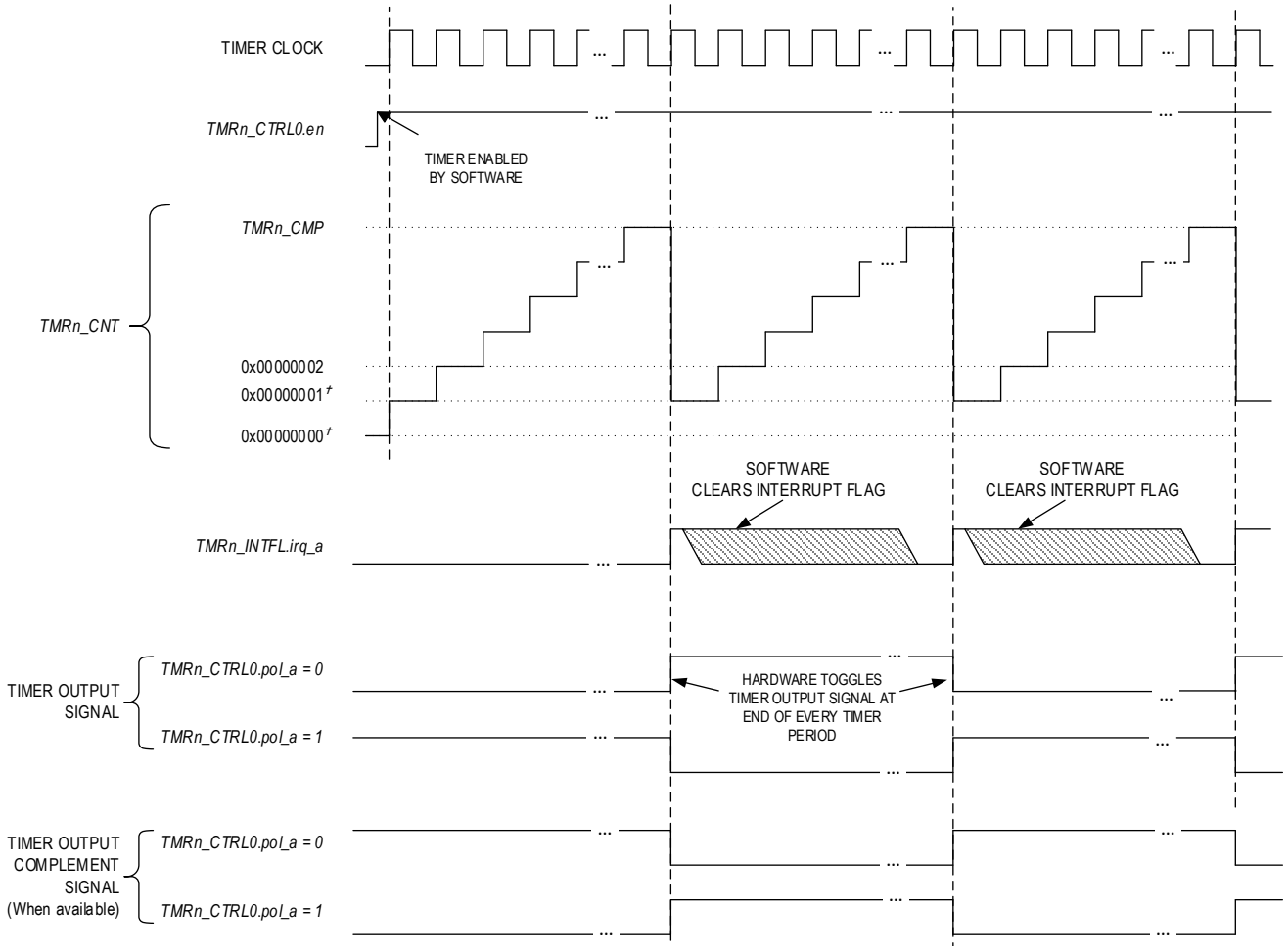
- The *TMRn_CNT* register is set to 0x0000 0001.
- If the timer output signal is toggled, the corresponding *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

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

Equation 13-3: Continuous Mode Timer Period

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

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.clkdiv* field to set the prescaler that determines the timer frequency.
4. If using the timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
5. Or, if using the inverted timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.
6. If using the timer interrupt, enable the corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to the *TMRn_CMP* register.
8. If desired, write an initial value to the *TMRn_CNT* register.
   a. The initial value only affects the first period; subsequent timer periods always reset the *TMRn_CNT* register to 0x0000 0001.
   b. Read the *TMRn_INTFL.wrdone* field until it reads 1.

   Note: The TMRn_CNT register is only writable if the timer clock is enabled (TMRn_CTRL0.clken = 1).
9. Enable the timer by writing 1 to the *TMRn_CTRL0.en* field.
   a. Read the *TMRn_CTRL0.en* field until it returns 1 to confirm the timer is enabled.

13.8.3 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 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% 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 4 PCLK cycles is required for a count to occur. To guarantee a count occurs, the timer input signal should be greater than 4 PCLK cycles in frequency.

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 to 1 indicating a timer interrupt event occurred.
- The timer remains enabled and continues incrementing.

Note: The software must clear the interrupt flag by writing 1 to the TMRn_INTFL.irq field. If the timer period ends and the interrupt flag is already set to 1, a second interrupt does not occur.

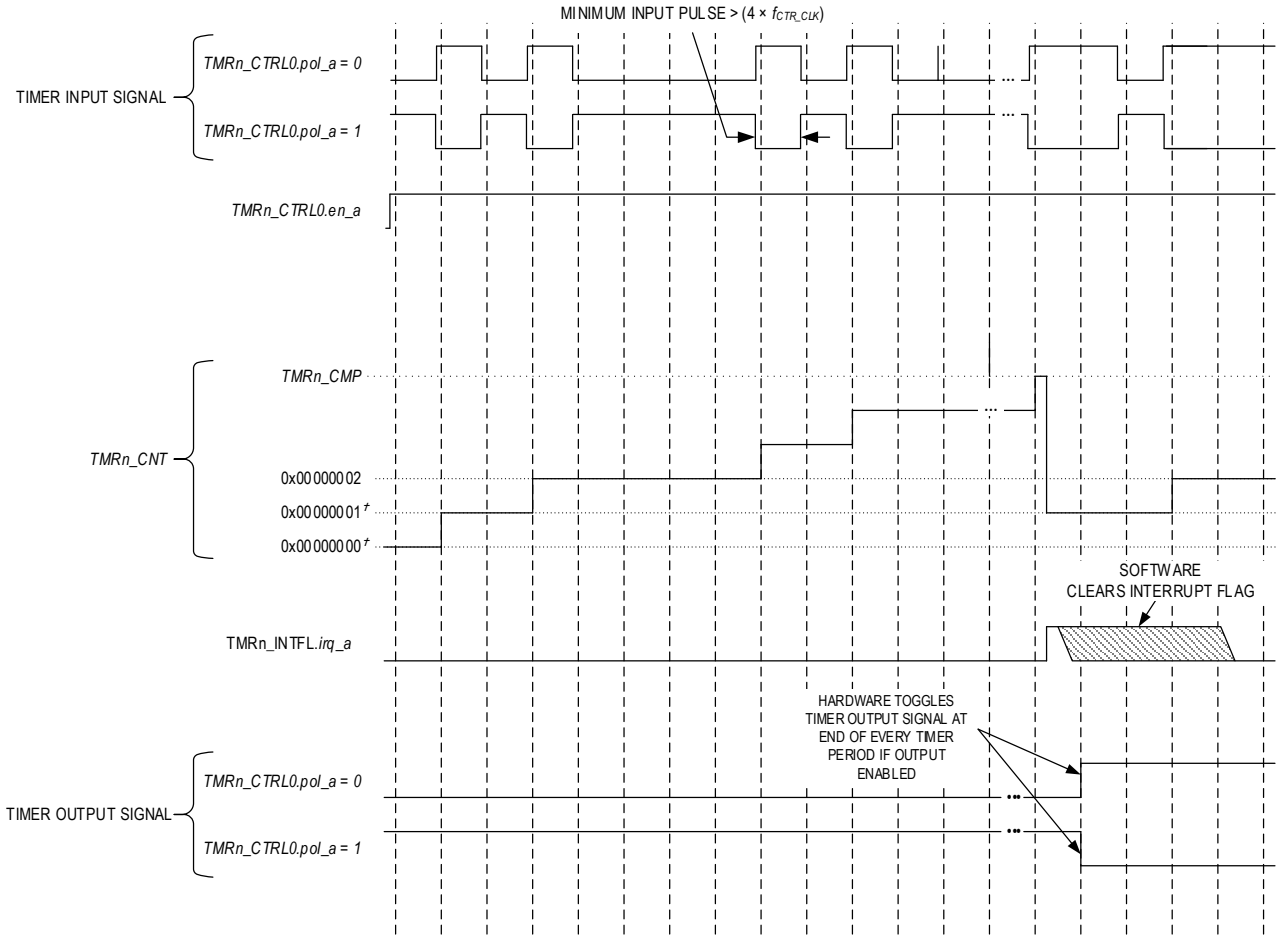
In counter mode, the number of timer input transitions that occurred during a period is equal to the *TMRn_CMP* register's setting. Use *Equation 13-5* to determine the number of transitions that occurred before the end of the timer's period.

Note: Equation 13-5 *is only valid during an active timer count before 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. Set *TMRn_CTRL1*.*outen_a* and *TMRn_CTRL1*.*outben_a* to the values shown in the *Operating Modes* section.
   d. Select the correct alternate function mode for the timer input pin.
5. Write the compare value to *TMRn_CMP*.
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.
   a. Read the *TMRn_INTFL*.*wrdone* field until it reads 1.

   Note: The TMRn_CNT register is only writable if the timer clock is enabled (TMRn_CTRL0.clk_en = 1).
7. Enable the timer by writing 1 to the *TMRn_CTRL0*.*en* field.
   a. Read the *TMRn_CTRL0*.*en* field until it returns 1 to confirm the timer is enabled.

13.8.4 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 the *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 the PWM operation:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set the *TMRn_CTRL0.mode field* to 3 to select PWM mode.
4. Set the *TMRn_CTRL0.clkdiv* field to set the prescaler that determines the timer frequency.
5. Configure the pin as a timer input and configure the electrical characteristics as needed.
6. Set *TMRn_CTRL0.pol* to match the desired initial (inactive) state.
7. Set *TMRn_CTRL0.pol* to select the initial logic level (high or low) and PWM transition state for the timer's output.
8. Set *TMRn_CNT* 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.
   b. Read the *TMRn_INTFL.wrdone* field until it reads 1.

   Note: The TMRn_CNT register is only writable if the timer clock is enabled (TMRn_CTRL0.clk_en = 1).
9. Set the *TMRn_PWM* value to the transition period count.
   a. If using the timer in dual 16-bit mode, disable both timers before writing the *TMRn_PWM* register.
   b. Read the *TMRn_INTFL.wrdone* field until it reads 1.

   Note: The TMRn_PWM register is only writable if the timer clock is enabled (TMRn_CTRL0.clk_en = 1).
10. Set the *TMRn_CMP* value for the PWM second transition period. The *TMRn_CMP* must be greater than the *TMRn_PWM* value.
    a. If using the timer in dual 16-bit mode, disable both timers before writing the *TMRn_CMP* register.
11. If using the timer interrupt, set the interrupt priority and enable the interrupt.
12. Enable the timer by writing 1 to the *TMRn_CTRL0.en* field.
    a. Read the *TMRn_CTRL0.en* field until it returns 1 to confirm the timer is enabled.

Equation 13-6 shows the formula for calculating the timer PWM period.

Equation 13-6: Timer PWM Period

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

If an initial starting value other than 0x0000 0001 is loaded into the *TMRn_CNT* 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

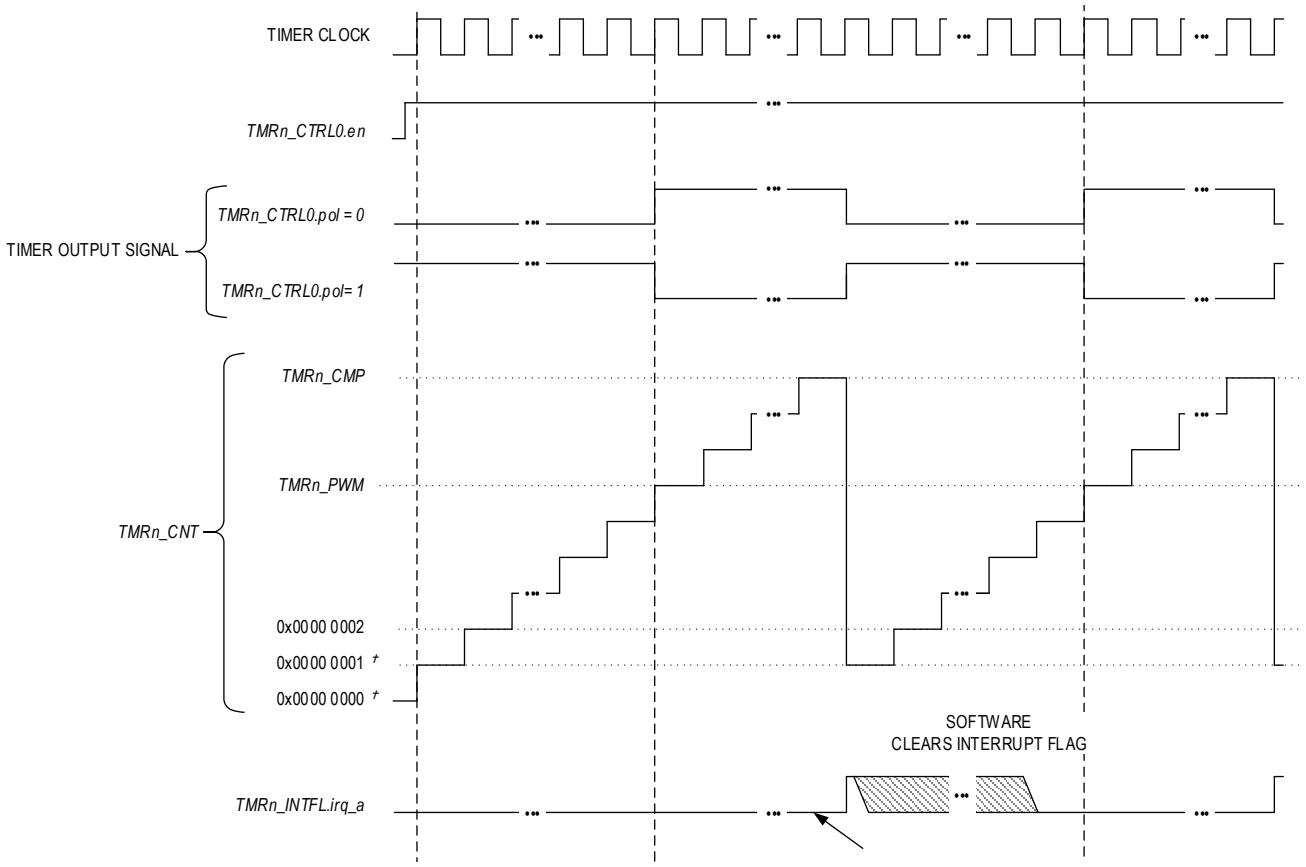
$$PWM\ output\ high\ time\ ratio\ (\%) = \frac{(TMR\_CMP - TMR\_PWM)}{TMR\_CMP} \times 100$$

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.

13.8.5 Capture Mode (4)

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

If a capture event does not occur before the timer's count value reaching the timer's compare value (*TMRn_CNT* = *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. The software must track the number of rollover events that occur before a capture event to determine the elapsed time of the capture event. When a capture event occurs, the software should reset the count of rollover events.

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

13.8.5.1 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.

The 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

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

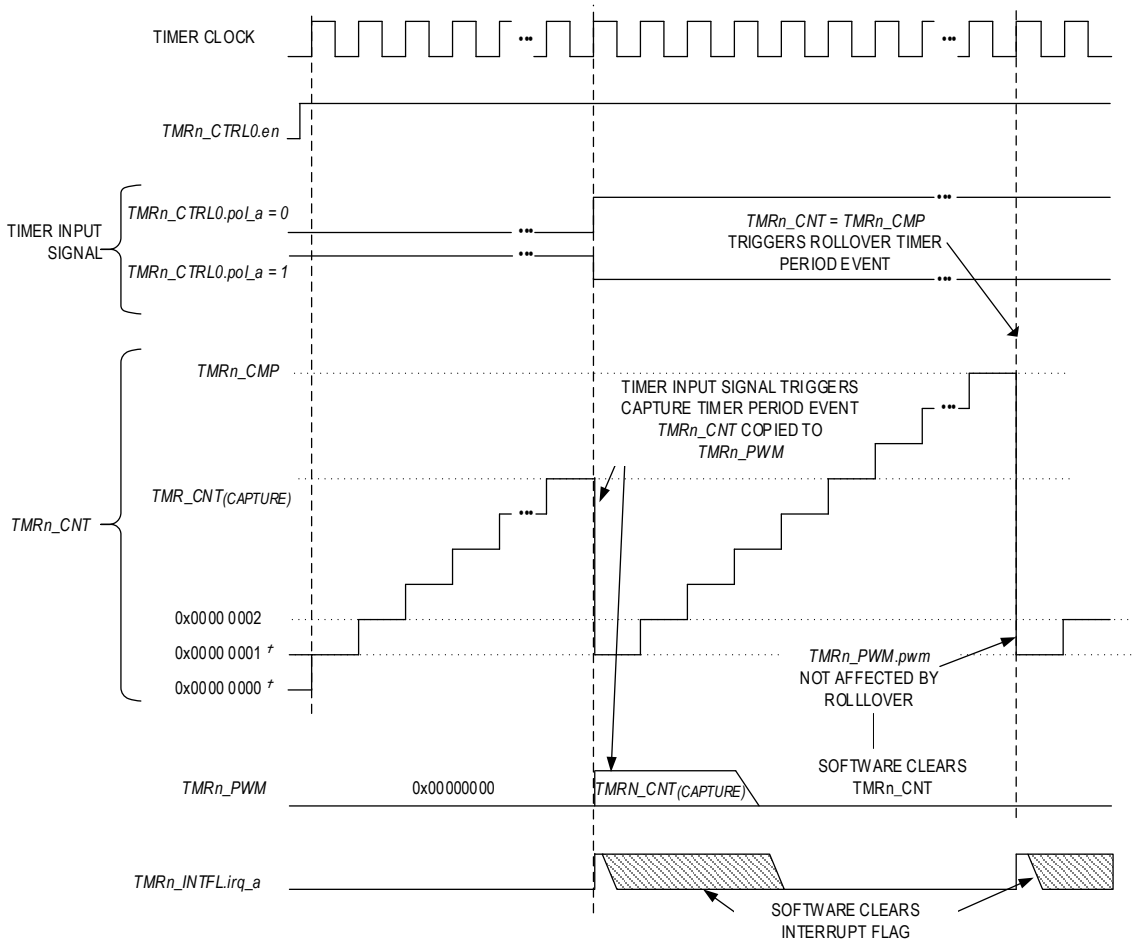
Note: The capture elapsed time calculation is only valid after the capture event occurs and the timer stores the captured count in the TMRn_PWM register.

13.8.5.2 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 = 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 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.
   b. Read the *TMRn_INTFL.wrdone* field until it reads 1.

   Note: The TMRn_CNT register is only writable if the timer clock is enabled (TMRn_CTRL0.clk_en = 1).
6 Write the compare value to the *TMRn_CMP* register.
7. Select the capture event by setting *TMRn_CTRL1.capevent_sel*.
8. Enable the timer by writing 1 to the *TMRn_CTRL0.en* field.
   a. Read the *TMRn_CTRL0.en* field until it returns 1 to confirm the timer is enabled.

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.

13.8.6 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 event occurs:

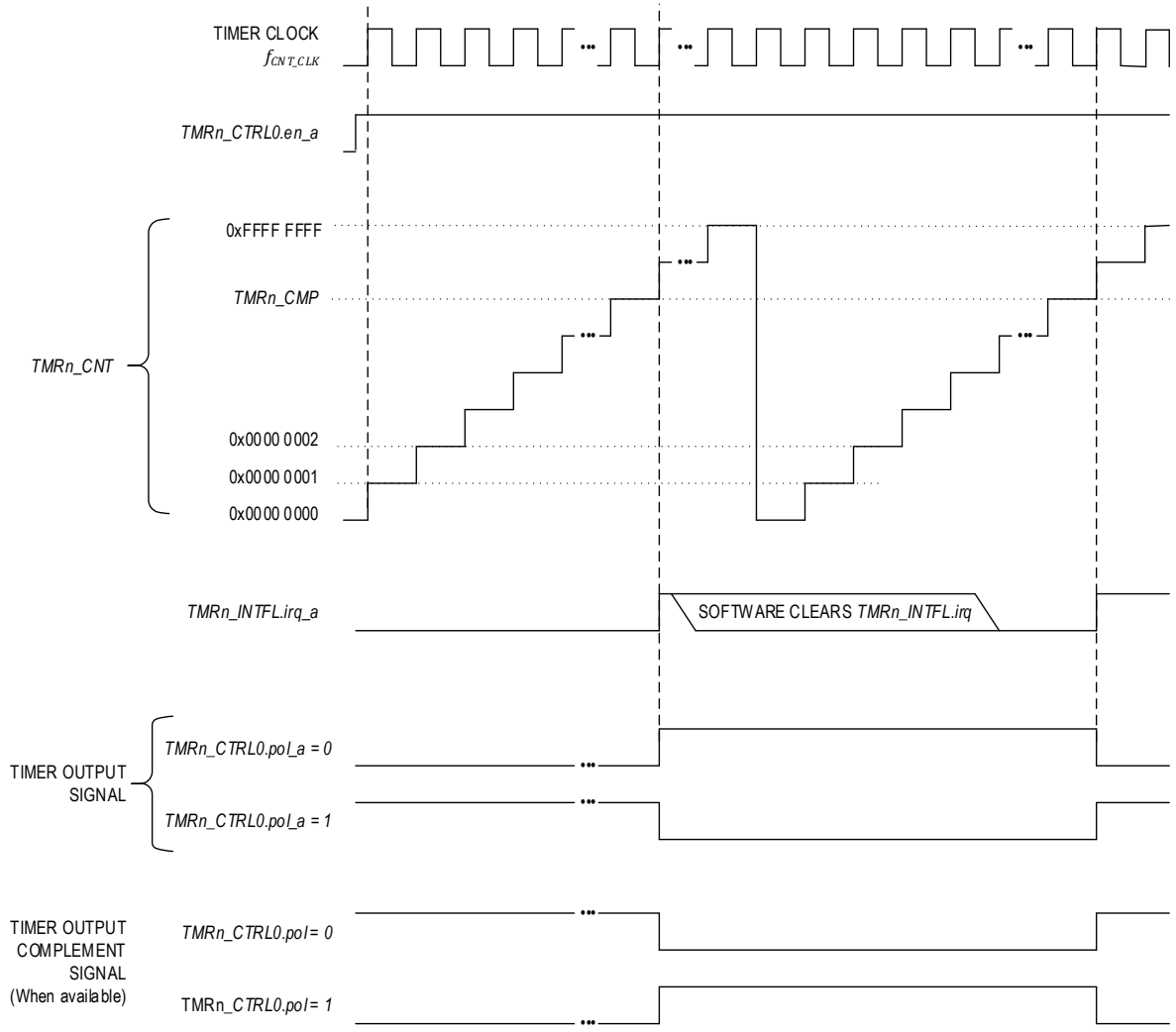
- Unlike other modes, *TMRn_CNT* is reset to 0x0000 00000, not 0x0000 0001 at the end of the timer period. The timer remains enabled and continues incrementing.
- The corresponding *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.
- The hardware toggles the state of the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

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

Equation 13-11: Compare Mode Timer Period

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

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)

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.clkdiv* to set the prescaler that determines the timer frequency.
5. If using the timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
6. If using the inverted timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.
7. If using the timer interrupt, enable the corresponding field in the *TMRn_CTRL1* register.
8. Write the compare value to *TMRn_CMP.*
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 0000.
   b. Read the *TMRn_INTFL.wrdone* field until it reads 1.

   Note: The TMRn_CNT register is only writable if the timer clock is enabled (TMRn_CTRL0.clk_en = 1).
10. Enable the timer by writing 1 to the *TMRn_CTRL0.en* field.
    a. Read the *TMRn_CTRL0.en* field until it returns 1 to confirm the timer is enabled.

13.8.7 Gated Mode (6)

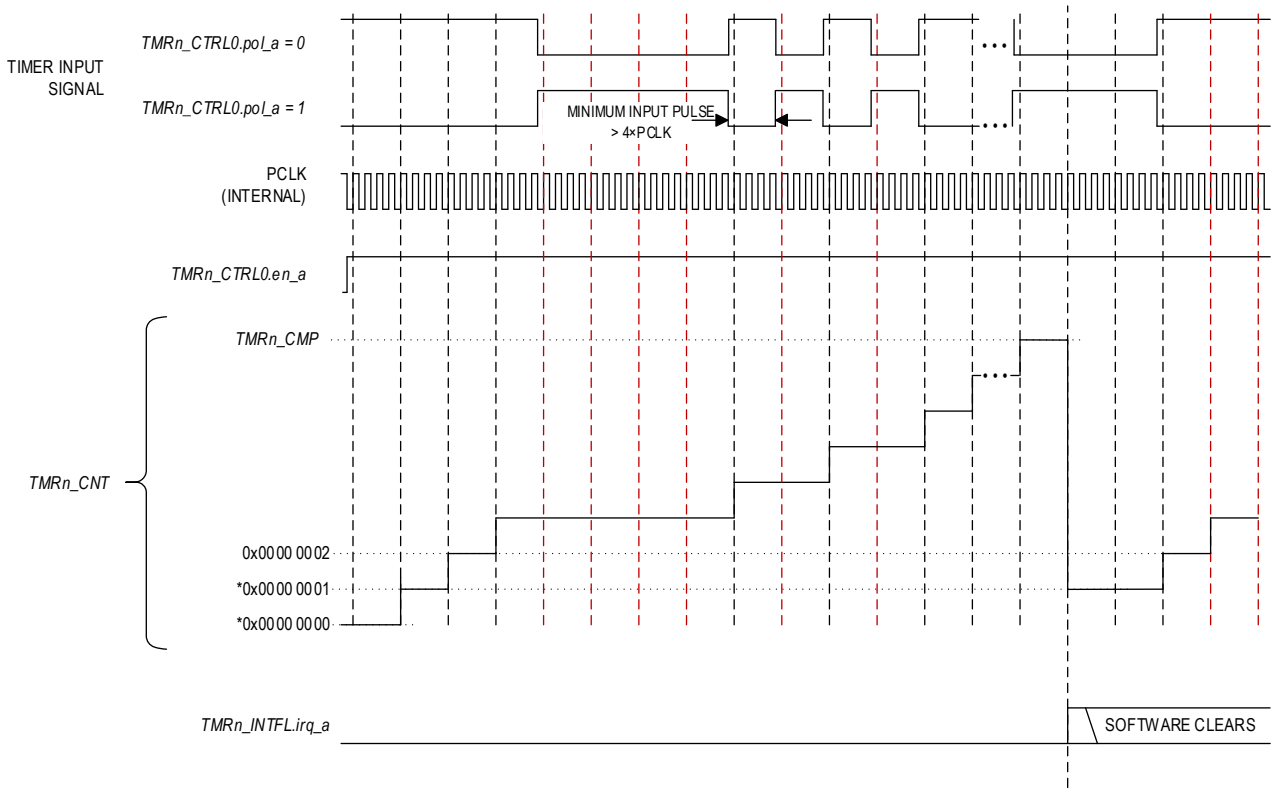
Gated mode is similar to continuous mode, except that *TMRn_CNT* only increments when the timer input signal is in its active state.

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.
- 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.
   b. Read the *TMRn_INTFL.wrdone* field until it reads 1.

   Note: The TMRn_CNT register is only writable if the timer clock is enabled (TMRn_CTRL0.clk_en = 1).
6 Write the compare value to *TMRn_CMP*.
7. Enable the timer by writing 1 to the *TMRn_CTRL0.en* field.
   a. Read the *TMRn_CTRL0.en* field until it returns 1 to confirm the timer is enabled.

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

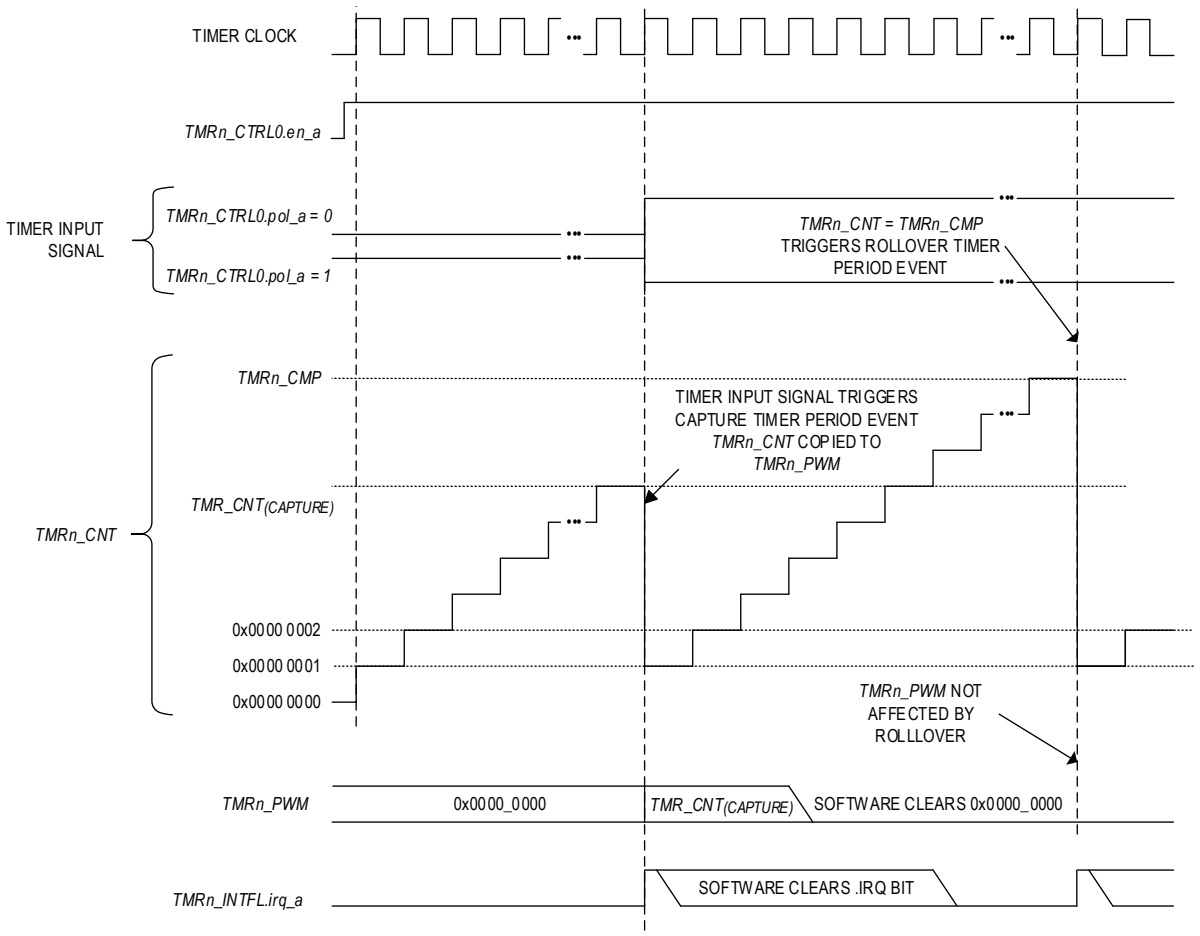
- The value in *TMRn_CNT* register is copied to the *TMRn_PWM* register.
- The *TMRn_CNT* register is set to 0x0000 0001.
- The timer remains enabled and continues incrementing.
- The corresponding *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

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

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

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 affects only the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
   b. Read the *TMRn_INTFL.wrdone* field until it reads 1.

   Note: The TMRn_CNT register is only writable if the timer clock is enabled (TMRn_CTRL0.clk_en = 1).
6 Write the compare value to *TMRn_CMP*.
7. Enable the timer by writing 1 to the *TMRn_CTRL0.en* field.
   a. Read the *TMRn_CTRL0.en* field until it returns 1 to confirm the timer is enabled.

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

13.8.9 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.

13.8.10 Inactive Gated Mode (14)

Inactive gated mode is similar to gated mode, except the interrupt is triggered when the timer input pin is in its inactive state.

13.9 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 registers, as shown in *Table 13-8*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 13-8: Timer Register Summary

Offset	Register	Description
[0x0000]	*TMRn_CNT*	Timer Counter Register
[0x0004]	*TMRn_CMP*	Timer Compare Register
[0x0008]	*TMRn_PWM*	Timer PWM Register
[0x000C]	*TMRn_INTFL*	Timer Interrupt Register
[0x0010]	*TMRn_CTRL0*	Timer Control Register
[0x0014]	*TMRn_NOLCMP*	Timer Non-Overlapping Compare Register
[0x0018]	*TMRn_CTRL1*	Timer Configuration Register
[0x001C]	*TMRn_WKFL*	Timer Wake-up Status Register

13.9.1 Register Details

Table 13-9: Timer Count Register

Timer Count			TMRn_CNT	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W*	0	**Timer Count** This field increments at a rate dependent on the selected timer operating mode. The function of the bits in this field are dependent on the 32-bit/16-bit configuration. Reads of this register always return the current value. **Note: This register is only writable if the timer clock is enabled (TMRn_CTRL0.clken = 1).*

Table 13-10: Timer Compare Register

Timer Compare			TMRn_CMP	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**Timer Compare Value** The value in this register is used as the compare value for the timer's count value. The compare field meaning is determined by the specific mode of the timer. See the timer mode's detailed configuration section for compare usage and meaning.

Table 13-11: Timer PWM Register

Timer PWM			TMRn_PWM	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	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.clken = 1).*

Table 13-12: 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_PWM[31:16]*. When this field is set to 0, 32-bit writes to the *TMRn_CNT* and *TMRn_PWM* registers only modify the lower 16 bits associated with TimerA. 0: Enabled. 1: Disabled. *Note: This field always reads 0 if the timer is configured as a 32-bit cascade timer.*

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 the hardware when the 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 disables 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: This field always reads 0 if the timer is configured as a 32-bit cascade timer.*
8	wrdone_a	R	0	**TimerA Write Done** This field is cleared to 0 by the hardware when the software performs a write to *TMRn_CNT[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 13-13: 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: Normal operation. 1: Reset Timer B.
28:24	-	RO	0	**Reserved**

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
23:20	clkdiv_b	R/W	0	**TimerB Prescaler Select** The *clkdiv_b* field selects a prescaler that divides the timer's source clock to set the timer's count clock as follows: $f_{CNT\_CLK} = f_{CLK\_SOURCE} / prescaler$ See the *Operating Modes* section for details on which timer modes use the prescaler. 0: 1. 1: 2. 2: 4. 3: 8. 4: 16. 5: 32. 6: 64. 7: 128. 8: 256. 9: 512. 10: 1024. 11: 2048. 12: 4096. 13-15: Reserved.
19:16	mode_b	R/W	0	**TimerB Mode Select** Set this field to the desired mode for TimerB. 0: One-shot. 1: Continuous. 2: Counter. 3: PWM. 4: Capture. 5: Compare. 6: Gated. 7: Capture/compare. 8: Dual-edge capture. 9-11: Reserved. 12: Internally gated. 13-15: Reserved.
15	en_a	R/W	0	**TimerA Enable** 0: Disabled. 1: Enabled.
14	clken_a	R/W	0	**TimerA Clock Enable** 0: Disabled. 1: Enabled.
13	rst_a	R/W1O	0	**TimerA Reset** 0: No action. 1: Reset TimerA.
12	pwmckbd_a	R/W	1	**TimerA PWM Output $\phi A'$ Disable** Set this field to 0 to enable the $\phi A'$ output signal. The $\phi A'$ output signal is disabled by default. 0: Enable the PWM $\phi A'$ output signal. 1: Disable PWM $\phi A'$ output signal.

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
11	nollpol_a	R/W	0	**TimerA PWM Output $\phi A'$ Polarity Bit** Set this field to 1 to invert the PWM $\phi A'$ signal. 0: Do not invert the PWM $\phi A'$ output signal. 1: Invert the PWM $\phi A'$ output signal.
10	nolhpol_a	R/W	0	**TimerA PWM Output ϕA Polarity Bit** Set this field to 1 to invert the PWM ϕA signal. 0: Do not invert the ϕA PWM output signal. 1: Invert the ϕA output signal.
9	pwmsync_a	R/W	0	**TimerA/TimerB PWM Synchronization Mode** 0: Disabled. 1: Enabled.
8	pol_a	R/W	0	**TimerA Polarity** Selects the polarity of the timer's input and output signal. This setting is not used if the GPIO is not configured for the timer's alternate function. This field's usage and settings are operating mode specific. See the *Operating Modes* section for details on the mode selected.
7:4	clkdiv_a	R/W	0	**TimerA Prescaler Select** The *clkdiv_a* field selects a prescaler that divides the timer's clock source to set the timer's count clock as follows: $f_{CNT\_CLK} = f_{CLK\_SOURCE} / prescaler$ See the *Operating Modes* section to determine which modes use the prescaler. 0: 1. 1: 2. 2: 4. 3: 8. 4: 16. 5: 32. 6: 64. 7: 128. 8: 256. 9: 512. 10: 1024. 11: 2048. 12: 4096. 13-15: Reserved.

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
3:0	mode_a	R/W	0	**TimerA Mode Select** Set this field to the desired operating mode for TimerA. 0: One-shot. 1: Continuous. 2: Counter. 3: PWM. 4: Capture. 5: Compare. 6: Gated. 7: Capture/compare. 8: Dual-edge capture. 9-11: Reserved. 12: Internally gated. 13-15: Reserved.

Table 13-14: Timer Non-Overlapping Compare Register

Timer Non-Overlapping Compare			TMRn_NOLCMP	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	hi_b	R/W	0	**TimerA Non-Overlapping High Compare 1** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output $\phi A'$ (phase A prime) and the next rising edge of the PWM output ϕA (phase A).
23:16	lo_b	R/W	0	**TimerA Non-Overlapping Low Compare 1** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output ϕA and the next rising edge of the PWM output $\phi A'$.
15:8	hi_a	R/W	0	**TimerA Non-Overlapping High Compare 0** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output $\phi A'$ and the next rising edge of the PWM output ϕA.
7:0	lo_a	R/W	0	**TimerA Non-Overlapping Low Compare 0** The 8-bit timer count value of non-overlapping time between the falling edge of the PWM output ϕA and the next rising edge of the PWM output $\phi A'$.

Table 13-15: 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 Wake-up Function** 0: Disabled. 1: Enabled.

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
27	sw_capevent_b	R/W	0	**TimerB Software Event Capture** Write this field to 1 to initiate a software event capture when operating the timer in capture mode to perform a software event capture. 0: No event. 1: Reserved.
26:25	capevent_sel_b	R/W	0	**TimerB Event Capture Selection** Set this field to the desired capture event source. See *Table 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 Edge Trigger for Event** 0: Rising edge triggered. 1: Falling edge triggered. *Note: External trigger events for rising-edge events must be active for a minimum of two timer clocks for detection.*
22:20	event_sel_b	R/W	0	**TimerB Event Selection** 0: Event disabled. 1-7: Reserved.
19	clkrdy_b	RO	0	**TimerB Clock Ready Status** This field indicates if the timer clock is ready. 0: Timer clock not ready or synchronization in progress. 1: Timer clock is ready.
18	clken_b	RO	0	**TimerB Clock Enable Status** This field indicates the status of the timer enable. 0: Timer not enabled or synchronization in progress. 1: Timer is enabled.
17:16	clksel_b	R/W	0	**TimerB Clock Source** See *Table 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** Reserved.
13	outen_a	R/W	0	**Output Enable** Reserved.
12	we_a	R/W	0	**TimerA Wake-up Function** 0: Disabled. 1: Enabled.

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
11	sw_capevent_a	R/W	0	**TimerA Software Event capture** 0: Normal operation. 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: Rising edge triggered. 1: Falling 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 the software enables the TimerA clock by writing 1 to the *TMRn_CTRL1.clken_a* field. 0: Timer not enabled or synchronization in progress. 1: TimerA clock is ready.
2	clken_a	R/W	0	**TimerA Clock Enable** Write this field to 1 to enable the TimerA clock. 0: Timer not enabled or synchronization in progress. 1: Timer is enabled.
1:0	clksel_a	R/W	0	**Clock Source TimerA** See *Table 13-1* for the available clock options for each timer instance. *Note: In cascade 32-bit mode, set TMRn_CTRL1.clksel_b to the same value as this field for proper operation.* 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.

Table 13-16: Timer Wake-up Status Register

Timer Wake-up Status			TMRn_WKFL	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**

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

14. Watchdog Timer (WDT)

The WDT protects against corrupt or unreliable software, power faults, and other system-level problems that can place the IC into an improper operating state. The software must periodically write a unique sequence to a dedicated register to confirm the application is operating correctly. Failure to reset the watchdog timer within a user-specified time frame can first generate an interrupt, allowing the application the opportunity to identify and correct the problem. If the application cannot regain normal operation, the watchdog timer can generate a system reset as a last resort.

Some instances provide a windowed timer function. These instances support an additional feature that can detect watchdog timer resets that occur too early, too late, or never. This could happen if program execution is corrupted and is accidentally forced into a tight loop of code that contains a watchdog sequence. This is not detected with a traditional WDT because the end of the timeout periods is never reached. A new set of "watchdog timer early" fields are available to support the lower limits required for windowing. Traditional watchdog timers can only detect a loss of program control that fails to reset the watchdog timer.

Each time the application performs a reset as early as possible in the application software, examine the peripheral control register 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.

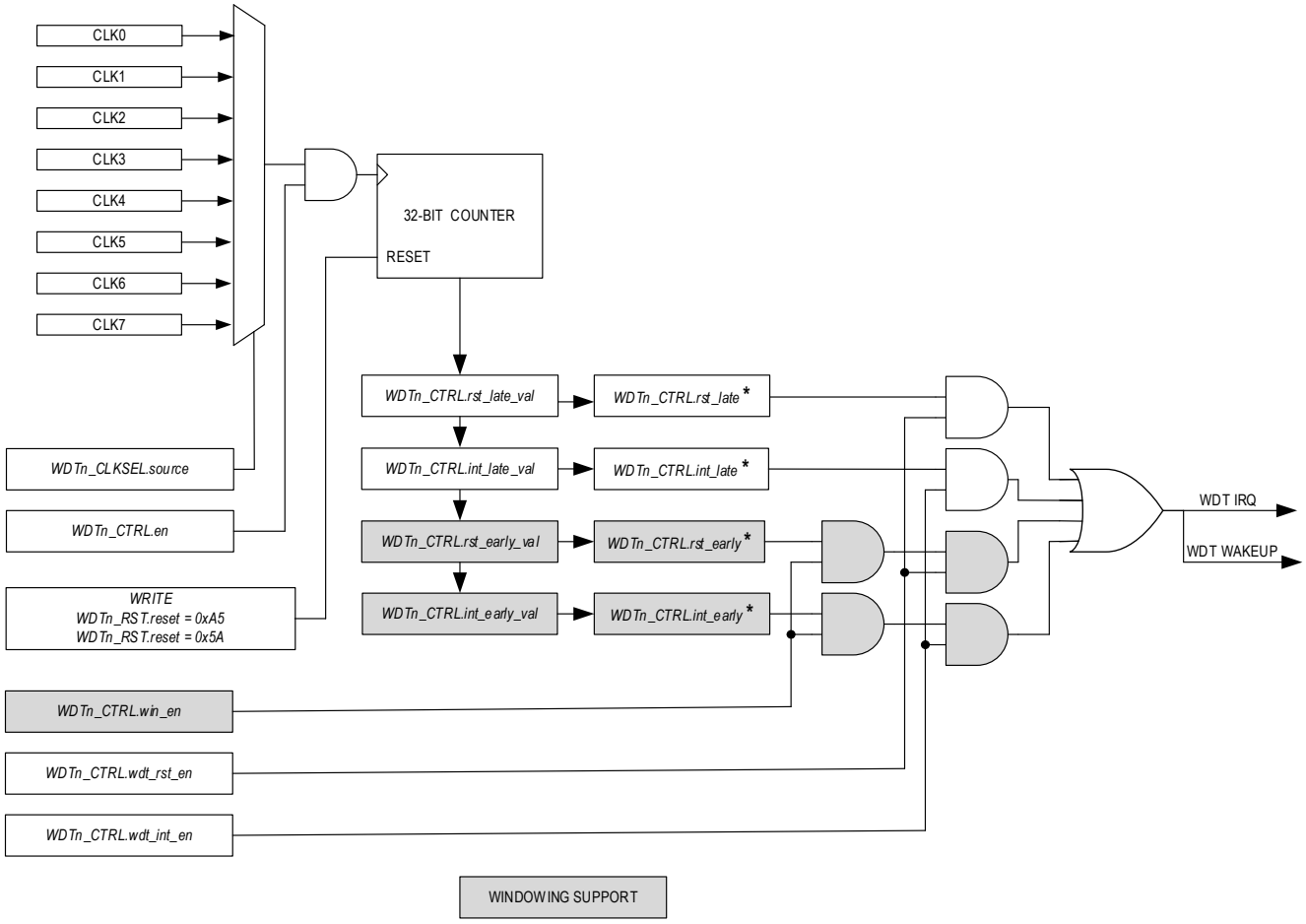
The WDT is a critical safety feature, and most fields are reset on POR or system reset events only.

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: MAX32670/MAX32671 WDT Instances Summary

Instance	Window Support	CLK0	CLK1	CLK2	CLK3	CLK4	CLK5	CLK6	CLK7
WDT0	Yes	PCLK	IPO	IBRO	INRO	ERTCO	EXT_CLK1 GPIO0.12 (AF4)	ERFO	Reserved
WDT1	Yes	PCLK	IPO	IBRO	INRO	ERTCO	EXT_CLK1 GPIO0.12 (AF4)	ERFO	Reserved

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 interrupt, the WDT continues to increment until the count reaches the WDT late reset threshold. The WDT generates a late reset event that sets the WDT late reset flag and generates a system interrupt.

Instances that support the window feature (*WDTn_CTRL.win_en* = 1) can generate a WDT early interrupt event if the WDT feed sequence occurs earlier than expected. Analogously, the device attempts to regain program control by vectoring to the dedicated WDT ISR. The WDT ISR should reset the WDT counter, perform the desired recovery activity, and then return execution to a known good address.

A WDT feed sequence that occurs earlier than the WDT early reset threshold indicates the execution error is significant enough to initiate a reset of the device to correct the problem. The WDT generates an early reset event that sets the WDT late reset flag and generates a system interrupt.

The event flags are set regardless of the corresponding interrupt or reset enable and include the early interrupt and early event flags, even if the WDT is disabled (*WDTn_CTRL.win_en* = 0).

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 Wake-up Timer

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

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 Feed Sequence

Two consecutive write instructions to the *WDTn_RST.reset* field are required to reset the *WDTn_CNT.count* = 0. Disable global interrupts immediately before and re-enable 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. Hardware automatically clears the *WDTn_CNT.count* to 0.
3. Re-enable interrupts.

14.3.2 WDT Enable Sequence

Perform the enable sequence 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. The hardware sets *WDTn_CTRL.en* to 1 automatically.

14.3.3 WDT Disable Sequence

Perform the disable sequence 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. The hardware clears *WDTn_CTRL.en* to 0 automatically.

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

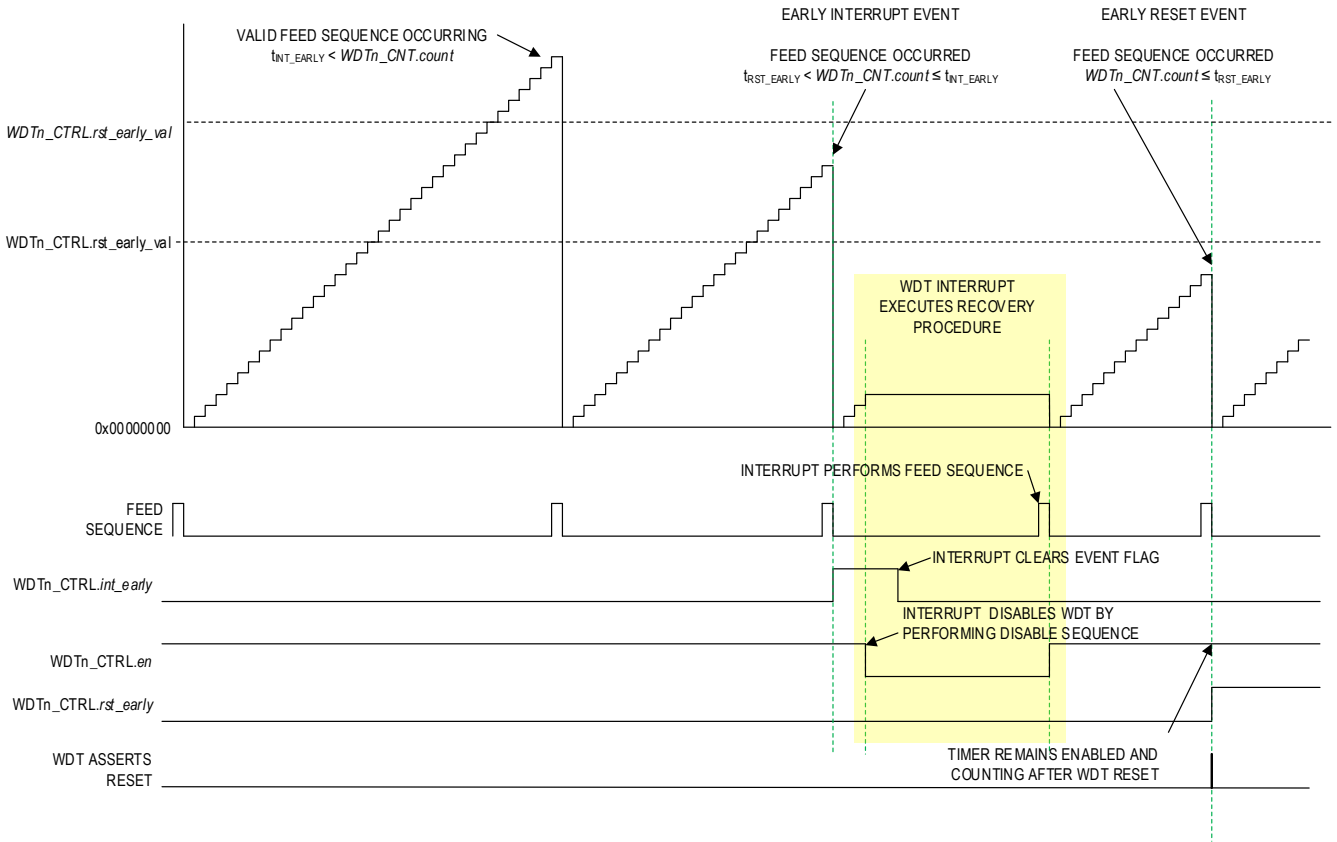
Event	Condition	Peripheral Interrupt Event Flag	Peripheral Interrupt Event Enable
Early Interrupt	Feed sequence occurs while *WDTn_CTRL.rst_early_val* ≤ *WDTn_CNT.count* < *WDTn_CTRL.int_early_val* *WDTn_CTRL.win_en = 1*	*WDTn_CTRL.int_early*	*WDTn_CTRL.wdt_int_en*
Early Reset	Feed sequence occurs while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

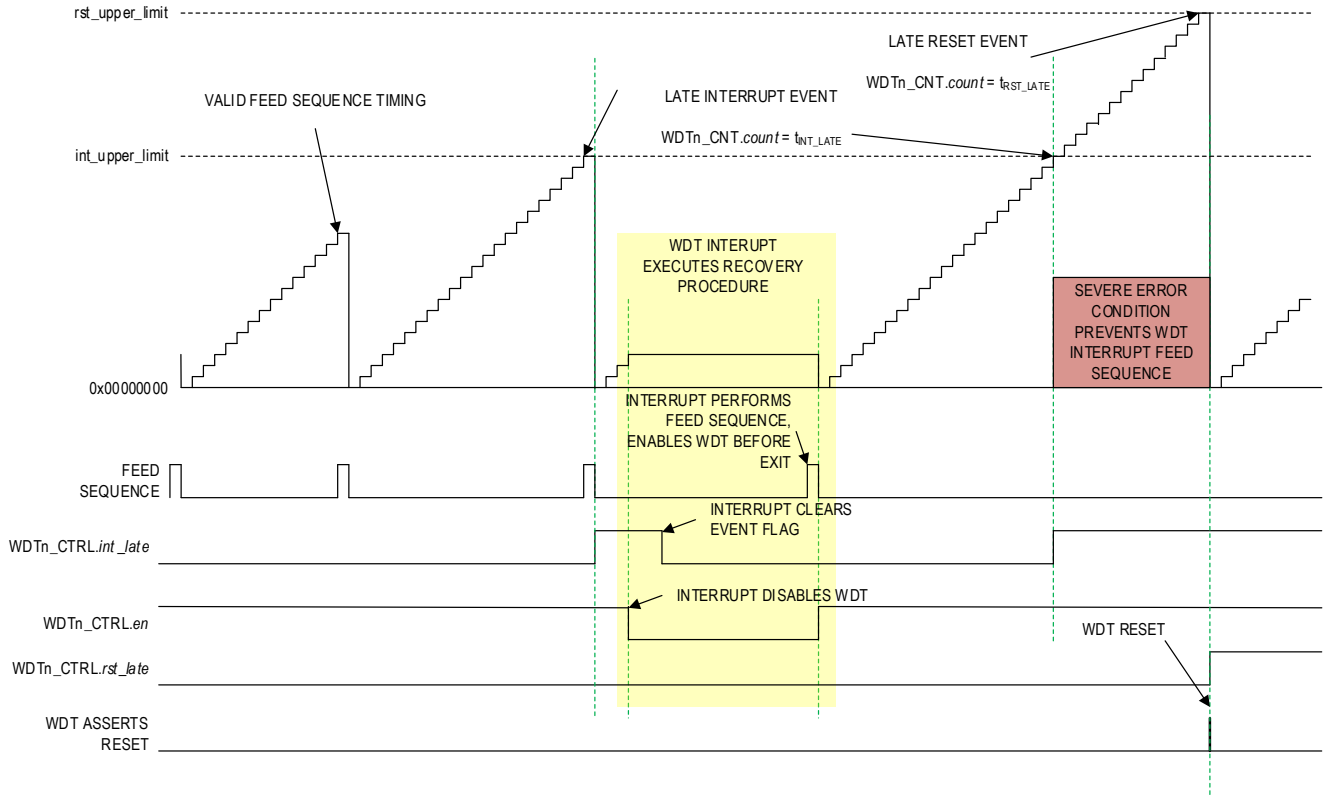
The following occurs when a WDT late interrupt event occurs:

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 disable sequence and disable the WDT:
   a. Disable global interrupts.
   b. Write *WDTn_RST.reset* to 0xDE.
   c. Write *WDTn_RST.reset* to 0xAD.
   d. The hardware automatically clears *WDTn_CTRL.en* to 0, disabling the WDT.
   e. Re-enable global interrupts.
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. Configure *WDTn_CLKSEL.source* to select the clock source.
5. 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.
6. 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.
7. 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.
8. 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.
9. Execute the WDT feed sequence to reset the WDT counter.
   a. Write *WDTn_RST.reset* to 0xA5.
   b. Write *WDTn_RST.reset* to 0x5A.

10. Execute the WDT enable sequence and enable the WDT:
    a. Disable global interrupts.
    b. Write *WDTn_RST.reset* to 0xFE.
    c. Write *WDTn_RST.reset* to 0xAD.
    d. Set *WDTn_CTRL.en* to 1 to enable the WDT.
    e. Re-enable global interrupts.
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.

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

Table 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.

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
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 status change in progress. 1: WDT status change complete.
27	clkrdy_ie	R/W	0	**Clock Switch Ready Interrupt Enable** This interrupt prevents the software from needing to poll the *WDTn_CTRL.clkrdy* field to determine when the WDT clock is ready. When the *WDTn_CTRL.clkrdy* field transitions from 1 to 0, this interrupt signals the transition is complete. 0: Disabled. 1: Enabled.
26:24	-	RO	0	**Reserved**
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.*

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
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 is performed earlier than the time determined by the *WDTn_CTRL.int_early* field. This flag is set even if *WDTn_CTRL.win_en* = 0. 0: No interrupt event. 1: Interrupt event occurred. *Note: A WDT interrupt is generated if the WDT interrupt is enabled (WDTn_CTRL.wdt_int_en = 1).*
11	wdt_rst_en	R/W	0	**WDT Reset Enable** 0: Disabled. 1: Enabled.
10	wdt_int_en	R/W	0	**WDT Interrupt Enable** 0: Disabled. 1: Enabled.
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).*

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
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 disable sequence must be performed immediately before setting this field to 0. The WDT enable sequence must be performed immediately before setting this field to 1. 0: Disabled. 1: Enabled.
7:4	rst_late_val	R/W	0	**Reset Late Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$ *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*
3:0	int_late_val	R/W	0	**Interrupt Late Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$ *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*

Table 14-5: WDT Reset Register

WDT Reset			WDTn_RST	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	reset	R/W	0†	**Reset Watchdog Timer Count** See the *WDT Protection Sequence* section for details on using this field for resetting the counter, enabling, and disabling the WDT. *†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**
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 counter value for debugging. This register is reset by system reset, as well as the watchdog feeding sequence. When the WDT clock is off, the feeding sequence generates an asynchronous reset of 1 PCLK width. When the WDT clock is on, the feeding sequence generates a synchronous reset that is a handshake to the WDT clock domain. *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before reading this field.*

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.

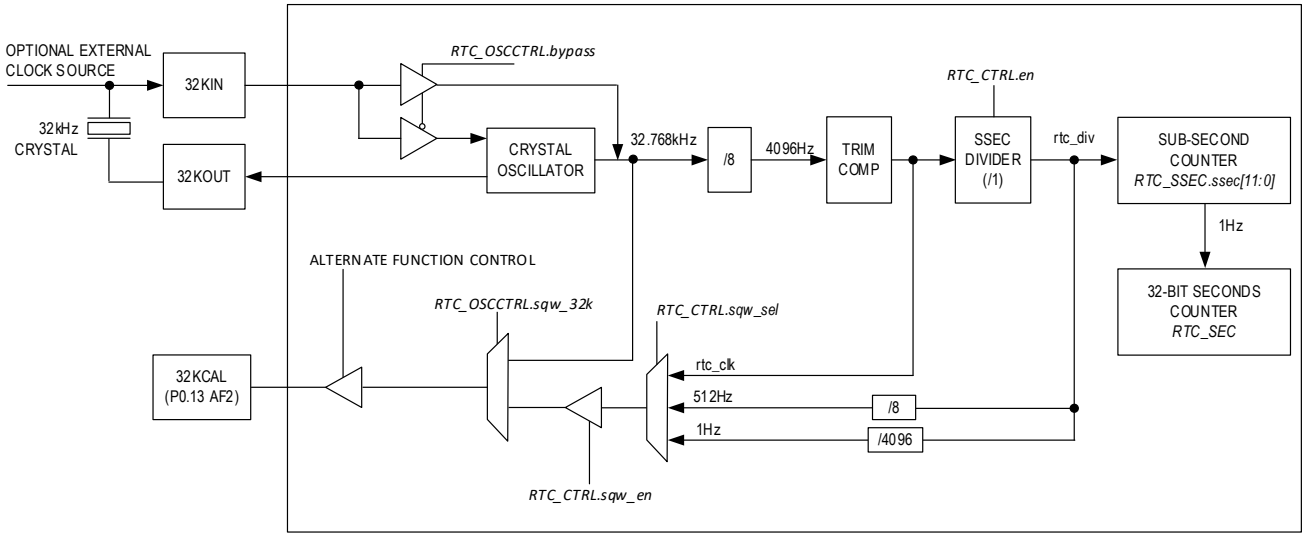
The 32-bit seconds counter register *RTC_SEC* is incremented every time there is a rollover of the *RTC_SSEC.ssec* sub-second counter field.

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: MAX32670/MAX32671 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*.

Note: See Enabling the ERTCO for details on enabling the ERTCO for use with the RTC.

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	244µs ($\frac{1}{4096Hz}$)	244µs	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	244µs ($\frac{1}{4096Hz}$)	244µs	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 and (*RTC_CTRL.tod_alarm_ie* = 0 or *RTC_CTRL.en* = 0)	Y	Time-of-day alarm
RTC_SSECA	*.ssec_alarm*	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.ssec_alarm_ie* = 0 or *RTC_CTRL.en* = 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

The software reads of the *RTC_SEC* and *RTC_SSEC* registers return invalid results if the read operation occurs on the same cycle that the register is being updated by the hardware (*RTC_CTRL.rdy* = 0). The hardware avoids this by setting the *RTC_CTRL.rdy* field to 1 for 120µs when the *RTC_SEC* and *RTC_SSEC* registers are valid and readable by the software.

Alternately, the software can set the *RTC_CTRL.rd_en* field to 1 to allow asynchronous reads of both *RTC_SEC* and *RTC_SSEC*.

Three methods are available to ensure valid results when reading *RTC_SEC* and *RTC_SSEC*:

1. The software clears the *RTC_CTRL.rdy* field to 0.
   a. The software polls the *RTC_CTRL.rdy* field until it reads 1 before reading the registers.
   b. The software must read the *RTC_SEC* and *RTC_SSEC* registers within 120µs to ensure valid register data.
2. The software sets the *RTC_CTRL.rdy_ie* field to 1 to generate an RTC interrupt when the hardware sets the *RTC_CTRL.rdy* field to 1.
   a. The software must service the RTC interrupt and read the *RTC_SEC*, *RTC_SSEC*, or both registers while the *RTC_CTRL.rdy* field is 1 to ensure valid data, avoiding the overhead associated with polling the *RTC_CTRL.rdy* field.
3. The software sets the *RTC_CTRL.rd_en* field to 1 enabling asynchronous reads of both the *RTC_SEC* register and the *RTC_SSEC* register.
   a. The software must read consecutive identical values of each of the *RTC_SEC* register and the *RTC_SSEC* register to ensure valid data.

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

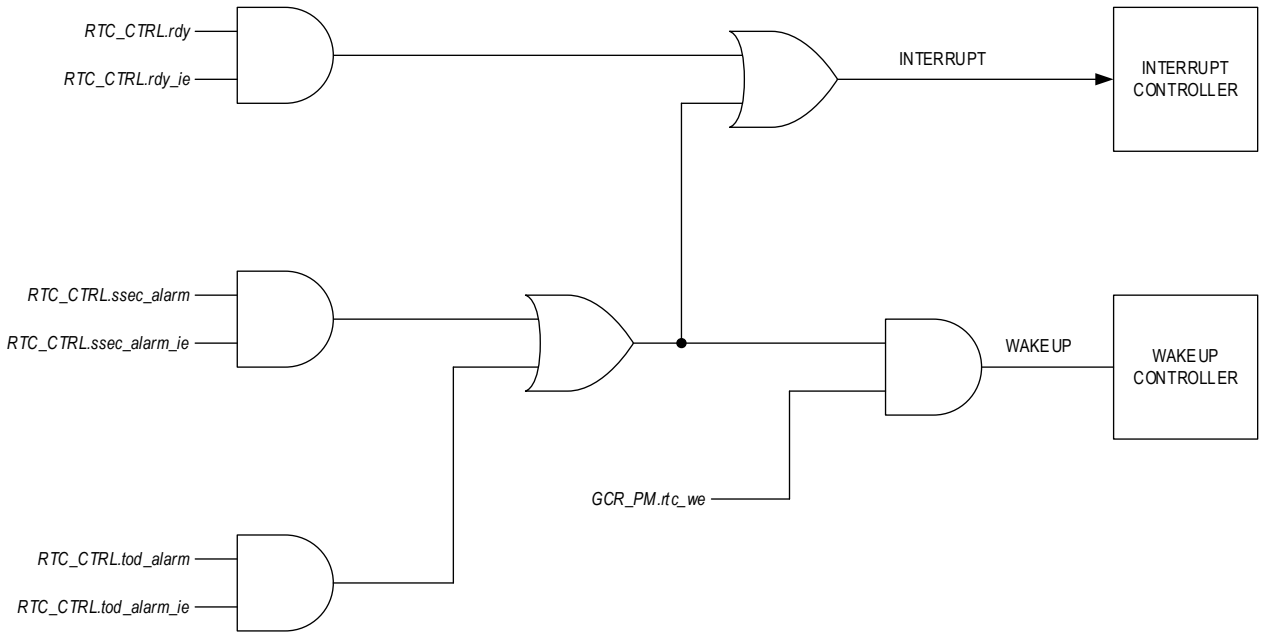
The following is a list of conditions that, when enabled, generate an RTC interrupt:

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/Wake-Up Diagram Wake-Up Function



Use this procedure to enable the RTC as a wake-up source:

1. Configure the RTC interrupt enable bits, enabling one or more interrupt conditions to generate an RTC interrupt.
2. Create an RTC interrupt handler function and register the address of the RTC_IRQn using the NVIC.
3. Set the *GCR_PM.rtc_we* field to 1 to enable system wake-up by the RTC.
4. Enter the desired low-power mode. See *Operating Modes* for details.

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: MAX32670/MAX32671 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 *RTC_CTRL.en* = 1
	512Hz (Compensated)	*RTC_CTRL.sqw_sel* = 1 *RTC_CTRL.sqw_en* = 1 *RTC_OSCCTRL.sqw_32k* = 0 *RTC_CTRL.en* = 1
	4kHz	*RTC_CTRL.sqw_sel* = 2 *RTC_CTRL.sqw_en* = 1 *RTC_OSCCTRL.sqw_32k* = 0 *RTC_CTRL.en* = N/A
	32kHz	*RTC_OSCCTRL.sqw_32k* = 1 *RTC_CTRL.en* = N/A

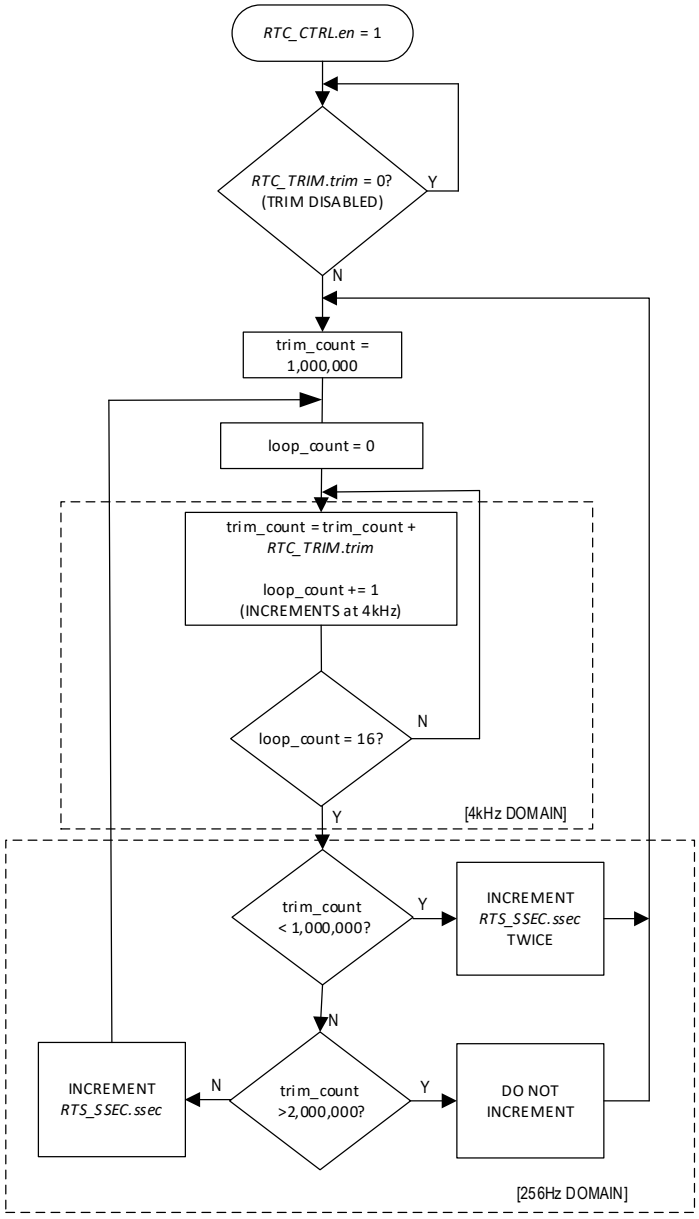
Use the following software procedure to generate and output the square wave:

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's alternate function, 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.
4. If the selected frequency is 1Hz or 512Hz, the RTC must be enabled to generate the output frequency (*RTC_CTRL.en = 1*).

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. This field is writable when *RTC_CTRL.busy* = 0 and either *RTC_CTRL.tod_alarm_ie* = 0 or *RTC_CTRL.en* = 0.

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. This field is writable when *RTC_CTRL.busy* = 0 and either *RTC_CTRL.ssec_alarm_ie* = 0 or *RTC_CTRL.en* = 0.

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.*
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.*

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
7	ssec_alarm	R/W	0*	**Sub-second Alarm Interrupt Flag** This interrupt flag is set when a sub-second alarm condition occurs. This flag is a wake-up source for the device. 0: No sub-second alarm pending. 1: Sub-second interrupt pending. **Note: Reset on POR only.*
6	tod_alarm	R/W	0*	**Time-of-Day Alarm Interrupt Flag** This interrupt flag is set by the hardware when a time-of-day alarm occurs. 0: No time-of-day alarm interrupt pending. 1: Time-of-day interrupt pending. **Note: Reset on POR only.*
5	rdy_ie	R/W	0*	**RTC Ready Interrupt Enable** 0: Disabled. 1: Enabled. **Note: Reset on system reset, soft reset, and GCR_RST0.rtc assertion.*
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.*

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
3	busy	RO	0*	**RTC Busy Flag** This field is set to 1 by the hardware while a register update is in progress. Software writes to the following registers result in this field being set to 1: • *RTC_SEC* • *RTC_SSEC* • *RTC_TRIM* The following fields cannot be written when this field is set to 1: • *RTC_CTRL.en* • *RTC_CTRL.tod_alarm_ie* • *RTC_CTRL.ssec_alarm_ie* • *RTC_CTRL.rdy_ie* • *RTC_CTRL.tod_alarm* • *RTC_CTRL.ssec_alarm* • *RTC_CTRL.sqw_en* • *RTC_CTRL.rd_en* This field is automatically cleared by the hardware when the update is complete. The software should poll this field until it reads 0 after changing the *RTC_SEC*, *RTC_SSEC*, or *RTC_TRIM* register before making any other RTC register modifications. 0: RTC not busy 1: RTC busy **Note: Reset on POR only.*
2	ssec_alarm_ie	R/W	0*	**Sub-Second Alarm Interrupt Enable** Check the *RTC_CTRL.busy* flag after writing to this field to determine when the RTC synchronization is complete. 0: Disable. 1: Enable. **Note: Reset on POR only.*
1	tod_alarm_ie	R/W	0*	**Time-of-Day Alarm Interrupt Enable** Check the *RTC_CTRL.busy* flag after writing to this field to determine when the RTC synchronization is complete. 0: Disable. 1: Enable. **Note: Reset on POR only.*
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 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.
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. Cyclic Redundancy Check (CRC)

The CRC engine performs CRC calculations on data written to the CRC data input register.

The features include:

- User-definable polynomials up to x^{32} (33 terms).
- DMA support.
- Supports automatic byte swap of data input and calculated output.
- Supports big-endian or little-endian data input and calculated output.
- Supports input reflection.

An *n*-bit CRC can detect the following types of errors:

- Single-bit errors.
- Two-bit errors for block lengths less than 2^k where k is the order of the longest irreducible factor of the polynomial.
- Odd numbers of errors for polynomials with the parity polynomial (x+1) as one of its factors (polynomials with an even number of terms).
- Burst errors less than *n*-bits.

In general, all but 1 out of 2^n errors are detected:

- 99.998% for a 16-bit CRC.
- 99.99999998% for a 32-bit CRC.

16.1 Instances

Instances of the peripheral are listed in *Table 16-1*.

Table 16-1: MAX32670/MAX32671 CRC Instances

Instance	Maximum Terms	DMA Support	Big- and Little-Endian
CRC	33 (2^{32})	Yes	Yes

16.2 Usage

A CRC value is often appended to the end of a data exchange between a transmitter and receiver. The transmitter appends the calculated CRC to the end of the transmission. The receiver independently calculates a CRC on the data it received. The result should be a known constant if the data and CRC were received error-free.

The software must configure the CRC polynomial, the starting CRC value, and the endianness of the data. Once configured, the software or the standard DMA engine transfers the data in either 8-bit, 16-bit, or 32-bit words to the CRC engine by writing to the corresponding data input register. Use the *CRC_DATAIN8* register for 8-bit data, the *CRC_DATAIN16* register for 16-bit data, and the *CRC_DATAIN32* register for 32-bit data. The hardware automatically sets the *CRC_CTRL*.*busy* field to 1 while the CRC engine is calculating a CRC over the input data. When the *CRC_CTRL*.*busy* field reads 0, the CRC result is available in the *CRC_VAL* register. The software or the standard DMA engine must track the data transferred to the CRC engine to determine when the CRC is finalized.

Because the receiving end calculates a new CRC on both the data and received CRC, 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 typically manage this by calculating everything backward. The software reverses the polynomial and does right shifts on the data. The resulting CRC is bit swapped and in the correct format.

By default, the CRC is calculated using the LSB first (*CRC_CTRL.msb* = 0.) When calculating the CRC using MSB first data (reflected), the software must set *CRC_CTRL.msb* to 1.

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 *CRC_POLY* register. This is necessary because the polynomial is always one bit larger than the resulting CRC, so a 32-bit CRC has a polynomial with 33 terms (x^0 … x^{32}).

Table 16-2: Organization of Calculated Result in the CRC_VAL.value Field

CRC_CTRL.msb	***CRC_CTRL.byte_swap_out***	**Order**
0	0	The CRC value returned is the raw value
1	0	The CRC value returned is reflected but not byte swapped
0	1	The CRC value returned is byte swapped but not reflected
1	1	The CRC value returned is reflected and then byte swapped

The CRC can be calculated on the MSB of the data first by setting the *CRC_CTRL.msb* field to 1, this is referred to as reflection. The CRC polynomial register, *CRC_POLY*, must be left-justified. The hardware implies the MSB of the polynomial just as it does when calculating the CRC LSB first. The LSB position of the polynomial must be set; this defines the length of the CRC. The initial value of the CRC, *CRC_VAL.value*, must also be left justified. When the CRC calculation is complete using MSB first, the software must right shift the calculated CRC value, *CRC_VAL.value*, by right shifting the output value if the CRC polynomial is less than 32 bits.

16.3 Polynomial Generation

The CRC can be configured for any polynomial up to x^{32} (33 terms) by writing to the *CRC_POLY.poly* field. *Table 16-3* shows common CRC polynomials.

The reset value of the *CRC_POLY.poly* field is the *CRC-32 Ethernet* polynomial. This polynomial is used by Ethernet and file compression utilities such as zip or gzip.

Note: Only write to the CRC polynomial register, CRC_POLY.poly, when the CRC_CTRL.busy field is 0.

Table 16-3: Common CRC Polynomials

Algorithm	**Polynomial Expression**	**Order**	**Polynomial**
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
CRC-CCITT	$x^{16}+x^{12}+x^5+x^0$	LSB	0x0000 8408
CRC-16	$x^{16}+x^{15}+x^2+x^0$	LSB	0x0000 A001
USB Data	$x^{16}+x^{15}+x^2+x^0$	MSB	0x8005 0000
Parity	x^1+x^0	LSB	0x0000 0001

16.4 Software CRC Calculations

The software can perform CRC calculations by writing directly to the CRC data input register. Each write to the CRC data input register triggers the hardware to compute the CRC value. The software is responsible for loading all data for the CRC into the CRC data input register. When complete, the result is read from the *CRC_VAL* register.

Use the following procedure to calculate a CRC:

1. Disable the CRC peripheral by setting the field *CRC_CTRL*.*en* to 0.
2. Configure input and output data format fields:
   a. *CRC_CTRL*.*msb*
   b. *CRC_CTRL*.*byte_swap_in*
   c. *CRC_CTRL*.*byte_swap_out*
3. Set the polynomial by writing to the *CRC_POLY*.*poly* field.
4. Set the initial value by writing to the *CRC_VAL*.*value* field.
   a. For a 32-bit CRC, write the initial value to the *CRC_VAL* register.
   b. For a 16-bit or 8-bit CRC, the unused bits in the *CRC_VAL* register must be set to 0.
5. Set the *CRC_CTRL*.*en* field to 1 to enable the peripheral.
6. Write a value to be processed to data input register.
   a. Calculate an 8-bit CRC by writing an 8-bit value to the *CRC_DATAIN8* register.
   b. Calculate a 16-bit CRC by writing a 16-bit value to the *CRC_DATAIN16* register.
   c. Calculate a 32-bit CRC by writing a 32-bit value to the *CRC_DATAIN32* register.
7. Poll the *CRC_CTRL*.*busy* field until it reads 0.
8. Repeat steps 6 and 7 until all input data is complete.
9. Disable the CRC peripheral by clearing the *CRC_CTRL*.*en* field to 0.
10. Read the CRC value from the *CRC_VAL*.*value* field.

16.5 DMA CRC Calculations

The CRC engine requests new data from the DMA controller when the fields *CRC_CTRL.en* and *CRC_CTRL.dma_en* are both set to 1. Enable the corresponding DMA channel's interrupt to receive an interrupt event when the CRC is complete. It is also possible for software to poll the DMA channel's interrupt flag directly by reading the *DMA_INTFL.ch<n>* flag until it reads 1.

Use the following procedure to calculate a CRC value using DMA:

1. Set *CRC_CTRL.en* = 0 to disable the peripheral.
2. Configure the DMA:
   a. Set *CRC_CTRL.dma_en* = 1 to enable DMA mode.
   b. See the DMA *Usage* section for details on configuring the DMA for a memory to peripheral transfer.
   c. Set the *DMA_CHn_CTRL.dstwd* field to match the size of the CRC calculation (0 for 8-bits, 1 for half-word, or 2 for word)
3. Configure the input and output data formats:
   a. *CRC_CTRL.msb*
   b. *CRC_CTRL.byte_swap_in*
   c. *CRC_CTRL.byte_swap_out*
4. Set the polynomial by writing to the *CRC_POLY.poly* field.
5. Set the initial value by writing to the *CRC_VAL* register.
   a. For a 32-bit CRC, write the initial value to the *CRC_VAL* register.
   b. For a 16-bit or an 8-bit CRC, the unused bits in the *CRC_VAL* register must be set to 0.
6. Set the *CRC_CTRL.en* field to 1 to enable the peripheral.
7. When the DMA operation completes, the hardware:
   a. Clears the *CRC_CTRL.busy* field to 0.
   b. Loads the new CRC value into the *CRC_VAL.value* field.
   c. Sets the *DMA_INTFL.ch<n>* field to 1 and generates a DMA interrupt if the *DMA_INTEN.ch<n>* field was set to 1.
8. Disable the CRC peripheral by clearing the *CRC_CTRL.en* field to 0.
9. Read the CRC value from the *CRC_VAL.value* field.

16.6 Registers

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

Table 16-4: CRC Register Summary

Offset	Name	Description
[0x0000]	*CRC_CTRL*	CRC Control Register
[0x0004]	*CRC_DATAIN8*	CRC 8-Bit Data Input Register
[0x0004]	*CRC_DATAIN16*	CRC 16-Bit Data Input Register
[0x0004]	*CRC_DATAIN32*	CRC 32-Bit Data Input Register
[0x0008]	*CRC_POLY*	CRC Polynomial Register
[0x000C]	*CRC_VAL*	CRC Value Register

16.6.1 Register Details

Table 16-5: CRC Control Register

CRC Control			CRC_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	busy	R	0	**CRC Busy** 0: Not busy. 1: Busy.
15:5	-	RO	0	**Reserved**
4	byte_swap_out	R/W	0	**Byte Swap CRC Value Output** 0: *CRC_VAL.value* is not byte swapped. 1: *CRC_VAL.value* is byte swapped.
3	byte_swap_in	R/W	0	**Byte Swap CRC Data Input** 0: The data input is processed least significant byte first. 1: The data input is processed most significant byte first.
2	msb	R/W	0	**Most Significant Bit Order** 0: LSB data first. 1: MSB data first (reflected).
1	dma_en	R/W	0	**DMA Enable** Set this field to 1 to enable a DMA request when the CRC calculation is complete (*CRC_CTRL.busy* = 0.) 0: Disabled. 1: Enabled.
0	en	R/W	0	**CRC Enable** 0: Disabled. 1: Enabled.

Table 16-6: CRC 8-Bit Data Input Register

CRC 8-Bit Data Input			CRC_DATAIN8	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	data	W	0	**CRC Data Input** Write 8-bit values to this register to calculate 8-bit CRCs. See *Table 16-2* for details on the byte and bit ordering of the data in this register. *Note: Do not write to this register if CRC_CTRL.busy = 1 or CRC_CTRL.en = 0.*

Table 16-7: CRC 16-Bit Data Input Register

CRC Data 16-Bit Input			CRC_DATAIN16	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	data	W	0	**CRC Data Input** Write 16-bit values to this register to calculate 16-bit CRCs. See *Table 16-2* for details on the byte and bit ordering of the data in this register. *Note: Do not write to this register if CRC_CTRL.busy = 1 or CRC_CTRL.en = 0.*

Table 16-8: CRC 32-Bit Data Input Register

CRC 32-Bit Data Input			CRC_DATAIN32	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	W	0	**CRC Data Input** Write 32-bit values to this register to calculate 32-bit CRCs. See *Table 16-2* for details on the byte and bit ordering of the data in this register. *Note: Do not write to this register if CRC_CTRL.busy = 1 or CRC_CTRL.en = 0.*

Table 16-9: CRC Polynomial Register

CRC Polynomial			CRC_POLY	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	poly	R/W	0xEDB8 8320	**CRC Polynomial** See *Table 16-2* for details on the byte and bit ordering of the data in this register.

Table 16-10: CRC Value Register

CRC Value			CRC_VAL	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	value	R/W	0	**Current CRC Value** The software can write to this register to set the initial state of the accelerator. This register should only be read or written when *CRC_CTRL.busy* = 0. See *Table 16-2* for details on the byte and bit ordering of the data in this register.

17. AES

The provided hardware AES accelerator performs calculations on blocks up to 128 bits.

The features include:

- Supports multiple key sizes:
  - 128 bits.
  - 192 bits.
  - 256 bits.
- DMA support for both receive and transmit channels.
- Supports multiple key sources:
  - Encryption using the external AES key.
  - Decryption using the external AES key.
  - Decryption using the locally generated decryption key.

17.1 Instances

Instances of the peripheral are listed in *Table 17-1*. Disable the peripheral by clearing *AES_CTRL.en* = 0 before writing to any register's field.

Table 17-1: MAX32670/MAX32671 AES Instances

Instance	128-Bit Key	192-Bit Key	256-Bit Key	DMA Support
AES	Yes	Yes	Yes	Yes

17.2 AES Key Generation

The TRNG supports generation of AES keys. The following steps describe how to generate an AES key and store it in the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers.

1. Clear the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers by setting *TRNG_CTRL.keywipe* to 1.
2. Set *TRNG_CTRL.aeskg_usr* to 1 to start the key generation process.
3. Read *TRNG_CTRL.aeskg_usr* until it reads 0.
4. The keys are generated and placed in the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers. The keys cannot be read by software.

17.3 AES Key Storage

The MAX32670/MAX32671 includes a dedicated memory location to store AES keys which are loaded on a POR to the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers. The following steps describe the process of saving software generated keys to the internal non-volatile memory for loading after a POR to the AES_KEY registers.

1. Generate an AES key of the desired length.
2. Write the AES key starting at address 0x1080 2008.
   a. For example, to write a 128-bit AES key, write the key to KEY[15] - KEY[0], as shown in *Figure 17-1*.
3. Software must write the AES key magic values. Once written, the AES keys are automatically loaded to the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers after each POR. Write the following values to FMV1[3]:FMV1[0] and FMV0[3]: FMV0[0].
   a. FMV0[0] = 0x79
   b. FMV0[1] = 0xD4
   c. FMV0[2] = 0x86
   d. FMV1[3] = 0x2B
   e. FMV1[0] = 0x79
   f. FMV1[1] = 0xD4
   g. FMV1[2] = 0x86
   h. FMV0[3] = 0x2B

Figure 17-1: AES KEY Storage

		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]							

17.4 Encryption of 128-Bit Blocks of Data Using FIFO

AES operations are typically performed on 128 bits of data at a time. Therefore, the simplest use case is to have software encrypt 128-bit blocks of data. The *AES_CTRL*.*start* field is unused in this case.

1. Enable the AES peripheral clock by setting *GCR_PCLKDIS1*.*aes* to 0.
2. If using the POR key, ensure the key is correctly stored as described in the *AES Key Storage* section.
3. If using a software generated key, write the key to the key registers, otherwise follow the steps in *AES Key Generation* to generate a key or use a key loaded on POR by following the steps in *AES Key Storage*.
   a. For a 128-bit key, write the key to the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers.
   b. For a 192-bit key, write the key to the *AES_KEY_AES_KEY5*:*AES_KEY_AES_KEY0* registers.
   c. For a 256-bit key, write the key to the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers.
4. Read the *AES_STATUS*.*busy* field until it reads 0.
5. Set *AES_CTRL*.*input_flush* to 1 to flush the input FIFO.
6. Set *AES_CTRL*.*output_flush* to 1 to flush the output FIFO.
7. Set *AES_CTRL*.*key_size* to the size of the loaded key.
8. Set *AES_CTRL*.*type* to 0 (encryption using the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers).
9. If interrupts are desired, set *AES_INTEN*.*done* to 1 so that an interrupt triggers at the end of the AES calculation.
10. Set *AES_CTRL*.*en* to 1 to enable the peripheral.
11. Write four 32-bit words of data to *AES_FIFO*.*data*.
    a. The hardware starts the AES calculation.
12. If *AES_INTEN*.*done* equals 1, an interrupt triggers after the AES calculation is complete.
13. If *AES_INTEN*.*done* equals 0, the software should poll until *AES_INTFL*.*done* reads 1.
14. Clear the interrupt done flag by writing 1 to *AES_INTFL*.*done*.
15. Read four 32-bit words from the *AES_FIFO*.*data* register (least significant word first).
16. Repeat steps 11-15 until all 128-bit blocks have been processed.

17.5 Encryption of 128-Bit Blocks Using DMA

For this example, it is assumed that the DMA both reads and writes data to and from the AES engine. This is not a requirement. The AES could use DMA on one side and software on the other. It is required that for each DMA transmit request the DMA writes four 32-bit words of data into the AES. Likewise, it is required that for each DMA receive request the DMA reads four 32-bit words of data out of the AES engine.

The *AES_CTRL*.*start* field is used in this case. The state of the *AES_STATUS*.*busy* and *AES_INTFL*.*done* flags are indeterminate during DMA operations. The software must clear *AES_INTEN*.*done* to 0 when using the DMA mode. Use the appropriate DMA interrupt instead to determine when the DMA operation is complete, and the results can be read from *AES_FIFO*.data.

Assuming the DMA is continuously filling the data input FIFO, the calculations complete in the following number of SYS_CLK cycles:

- 128-bit key: 181
- 192-bit key: 213
- 256-bit key: 245

The procedure to use DMA encryption/decryption is:

1. Enable the AES peripheral clock by setting *GCR_PCLKDIS1*.*aes* to 0.
2. Set the *AES_CTRL*.*en* field to 1 to enable the peripheral.
3. If using the POR key, ensure the key is correctly stored as described in the *AES Key Storage* section.
4. If using a software generated key, write the key to the key registers, otherwise follow the steps in *AES Key Generation* to generate a key or use a key loaded on POR by following the steps in *AES Key Storage*.
    a. For a 128-bit key, write the key to the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers.
    b. For a 192-bit key, write the key to the *AES_KEY_AES_KEY5*:*AES_KEY_AES_KEY0* registers.
    c. For a 256-bit key, write the key to the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers.
5. Initialize the AES receive and transmit channels for the DMA controller.
6. Read the *AES_STATUS*.*busy* field until it reads 0.
7. Set *AES_CTRL*.*input_flush* to 1 to flush the input FIFO.
8. Set *AES_CTRL*.*output_flush* to 1 to flush the output FIFO.
9. Set *AES_CTRL*.*key_size* to the size of the loaded key.
10. Select encryption or decryption:
    a. Set *AES_CTRL*.*type* to 0 (encryption using the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers).
    b. Set *AES_CTRL*.*type* to 1 (decryption using the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers).
    c. Set *AES_CTRL*.*type* to 2 (decryption using the locally generated decryption key from a previous encryption).
11. Set *AES_INTEN*.*done* to 0 for DMA operation.
12. Set *AES_CTRL*.*start* to 1 to start the AES DMA operation.
13. The DMA engine fills the FIFO, and hardware begins the AES calculation.
14. When an AES calculation is completed, the AES hardware 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, the hardware sets *AES_STATUS*.*output_full* to 1.

Step 13 and step 14 are repeated if the DMA has new data to write to the data input FIFO.

Note: The DMA interface to the AES only works when the amount of data is a multiple of 128 bits. For non-multiples of 128 bits, the remainder after calculating all 128-bit blocks must be encrypted manually using the steps in Encryption of Blocks Less Than 128 Bits.

17.6 Encryption of Blocks Less Than 128 Bits

The AES engine automatically starts a calculation when 128 bits (four writes of 32 bits) occurs. Operations of less than 128 bits use the start field to initiate the AES calculation.

1. Enable the AES peripheral clock by setting *GCR_PCLKDIS1*.*aes* to 0.
2. If using the POR key ensure the key is correctly stored as described in the *AES Key Storage* section.
3. If using a software generated key, write the key to the key registers, otherwise follow the steps in *AES Key Generation* to generate a key or use a key loaded on POR by following the steps in *AES Key Storage*.
   a. For a 128-bit key, write the key to the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers.
   b. For a 192-bit key, write the key to the *AES_KEY_AES_KEY5*:*AES_KEY_AES_KEY0* registers.
   c. For a 256-bit key, write the key to the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers.
4. Read the *AES_STATUS*.*busy* field until it reads 0.
5. Clear *AES_CTRL*.*en* to 0 to disable the peripheral.
6. Set *AES_CTRL*.*input_flush* to 1 to flush the input FIFO.
7. Set *AES_CTRL*.*output_flush* to 1 to flush the output FIFO.
8. Set *AES_CTRL*.*key_size* to the size of the loaded key.
9. Set *AES_CTRL*.*type* to 0 (encryption using the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers).
10. If interrupts are desired, set *AES_INTEN*.*done* to 1 so that an interrupt triggers at the end of the AES calculation.
11. Set *AES_CTRL*.*en* to 1 to enable the peripheral.
12. Write from one to three 32-bit words of data to the *AES_FIFO*.*data* field, least significant word first.
13. Start the calculation manually by setting *AES_CTRL*.*start* to 1.
14. If *AES_INTEN*.*done* = 1, an interrupt triggers after the AES calculation is complete.
15. If *AES_INTEN*.*done* = 0, the software should poll until *AES_INTFL*.*done* reads 1.
16. Read four 32-bit words from *AES_FIFO*.*data* (least significant word first).

17.7 Decryption

The decryption of data is very similar to encryption. The only difference is in the setting of *AES_CTRL*.*type*. There are two settings of this field for decryption:

- Decryption using the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers.
- Decryption with an internal decryption key.

The internal decryption key is generated during an encryption operation. Therefore, it is necessary to complete an encryption operation before doing the first decryption to ensure that a key is generated.

17.8 Interrupt Events

The peripheral generates interrupts for the events shown in *Table 17-2*. Unless noted otherwise, each instance has its own independent set of interrupts and higher-level flag and enable fields.

Multiple events may set an interrupt flag and generate an interrupt if the corresponding interrupt enable is set. The software must clear the flags in the interrupt handler if AES interrupts are enabled.

Table 17-2: Interrupt Events

Event	Local Interrupt Flag	Local Interrupt Enable
Data Output FIFO Overrun	*AES_INTFL*.*ov*	*AES_INTEN*.*ov*
Key Zero	*AES_INTFL*.*key_zero*	*AES_INTEN*.*key_zero*
Key Change	*AES_INTFL*.*key_change*	*AES_INTEN*.*key_change*

Event	Local Interrupt Flag	Local Interrupt Enable
Calculation Done	*AES_INTFL.done*	*AES_INTEN.done*

17.8.1 Data Output FIFO Overrun

When an AES calculation is completed, the AES engine 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.

17.8.2 Key Zero

Attempting a calculation with a key of all zeros generates a key zero event.

17.8.3 Key Change

Writing to any key register while *AES_STATUS.busy* is 1 generates a key change event.

17.8.4 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.

17.9 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, a soft reset, a POR, and the peripheral-specific resets.

Table 17-3: AES Register Summary

Offset	Name	Description
[0x0000]	*AES_CTRL*	AES Control Register
[0x0004]	*AES_STATUS*	AES Status Register
[0x0008]	*AES_INTFL*	AES Interrupt Flag Register
[0x000C]	*AES_INTEN*	AES Interrupt Enable Register
[0x0010]	*AES_FIFO*	AES Data FIFO

17.9.1 Register Details

Table 17-4: AES Control Register

AES Control			**AES_CTRL**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9:8	type	R/W	0	**Encryption Type** 0: Encryption using the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers. 1: Decryption using the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers. 2: Decryption using the locally generated decryption key. 3: Reserved.
7:6	key_size	R/W	0	**Encryption Key Size** 0: 128 bits. 1: 192 bits. 2: 256 bits. 3: Reserved.

AES Control			AES_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
5	output_flush	R/W1O	0	**Flush Data Output FIFO** This field always reads 0. 0: Normal operation. 1: Flush.
4	input_flush	R/W1O	0	**Flush Data 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 data input FIFO, allowing an AES calculation on less than 128-bits of data. By default, an AES calculation starts when the data input FIFO is full. This field always reads 0. 0: Normal operation. 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 17-5: AES Status Register

AES Status			AES_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	output_full	R	0	**Output FIFO Full** 0: Not full. 1: Full.
3	output_em	R	0	**Output FIFO Empty** 0: Not empty. 1: Empty.
2	input_full	R	0	**Input FIFO Full** 0: Not full. 1: Full.
1	input_em	R	0	**Input FIFO Empty** 0: Not empty 1: Empty
0	busy	R	0	**AES Busy** 0: Not busy. 1: Busy.

Table 17-6: AES Interrupt Flag Register

AES Interrupt Flag			AES_INTFL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	ov	W1C	0	**Data Output FIFO Overrun Event Interrupt** 0: No event. 1: Event occurred.
2	key_zero	W1C	0	**Key Zero Event Interrupt** 0: No event. 1: Event occurred.
1	key_change	W1C	0	**Key Change Event Interrupt** 0: No event. 1: Event occurred.
0	done	W1C	0	**Calculation Done Event Interrupt** 0: No event. 1: Event occurred.

Table 17-7: AES Interrupt Enable Register

AES Interrupt Enable			AES_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	ov	W1C	0	**Data Output FIFO Overrun Event Interrupt Enable** 0: Enabled. 1: Disabled.
2	key_zero	W1C	0	**Key Zero Event Interrupt Enable** 0: Enabled. 1: Disabled
1	key_change	W1C	0	**Key Change Event Interrupt Enable** 0: Enabled. 1: Disabled.
0	done	W1C	0	**Calculation Done Event Interrupt Enable** This event interrupt must be disabled when using the DMA. 0: Enabled. 1: Disabled.

Table 17-8: AES FIFO Register

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 data input FIFO. The hardware automatically starts a calculation after four words are written to this FIFO. The data should be written least significant word first. Reading this register pulls data from the data output FIFO. The least significant word is read first.

17.10 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, a soft reset, a POR, and the peripheral-specific resets.

Table 17-9: AES Register Summary

Offset	Name	Description
[0x0000]	*AES_KEY_AES_KEY0*	AES Key 0 Register
[0x0004]	*AES_KEY_AES_KEY1*	AES Key 0 Register
[0x0008]	*AES_KEY_AES_KEY2*	AES Key 0 Register
[0x000C]	*AES_KEY_AES_KEY3*	AES Key 0 Register
[0x0010]	*AES_KEY_AES_KEY4*	AES Key 0 Register
[0x0014]	*AES_KEY_AES_KEY5*	AES Key 0 Register
[0x0018]	*AES_KEY_AES_KEY6*	AES Key 0 Register
[0x001C]	*AES_KEY_AES_KEY7*	AES Key 0 Register

17.10.1 AES_KEY Register Details

Table 17-10: AES Key 0 Register

AES Key 0			**AES_KEY_AES_KEY0**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	*	*	**AES KEY 0** This register is optionally loaded with a portion of an AES key at POR if the user FMV is programmed at 0x1080 2000. See *AES Key Storage* for additional details. This register always reads 0.

Table 17-11: AES Key 1 Register

AES Key 1			**AES_KEY_AES_KEY1**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	*	*	**AES KEY 1** This register is optionally loaded with a portion of an AES key at POR if the user FMV is programmed at 0x1080 2000. See *AES Key Storage* for additional details. This register always reads 0.

Table 17-12: AES Key 2 Register

AES Key 2			**AES_KEY_AES_KEY2**	**[0x0008]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	*	*	**AES KEY 2** This register is optionally loaded with a portion of an AES key at POR if the user FMV is programmed at 0x1080 2000. See *AES Key Storage* for additional details. This register always reads 0.

Table 17-13: AES Key 3 Register

AES Key 3			AES_KEY_AES_KEY3	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	*	*	**AES KEY 3** This register is optionally loaded with a portion of an AES key at POR if the user FMV is programmed at 0x1080 2000. See *AES Key Storage* for additional details. This register always reads 0.

Table 17-14: AES Key 4 Register

AES Key 4			AES_KEY_AES_KEY4	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	*	*	**AES KEY 4** This register is optionally loaded with a portion of an AES key at POR if the user FMV is programmed at 0x1080 2000. See *AES Key Storage* for additional details. This register always reads 0.

Table 17-15: AES Key 5 Register

AES Key 5			AES_KEY_AES_KEY5	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	*	*	**AES KEY 5** This register is optionally loaded with a portion of an AES key at POR if the user FMV is programmed at 0x1080 2000. See *AES Key Storage* for additional details. This register always reads 0.

Table 17-16: AES Key 6 Register

AES Key 6			AES_KEY_AES_KEY6	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	*	*	**AES KEY 6** This register is optionally loaded with a portion of an AES key at POR if the user FMV is programmed at 0x1080 2000. See *AES Key Storage* for additional details. This register always reads 0.

Table 17-17: AES Key 7 Register

AES Key 7			AES_KEY_AES_KEY7	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	*	*	**AES KEY 1** This register is optionally loaded with a portion of an AES key at POR if the user FMV is programmed at 0x1080 2000. See *AES Key Storage* for additional details. This register always reads 0.

18. TRNG Engine

The Analog Devices supplied universal cryptographic library (UCL) provides a function to generate random numbers intended to meet the requirements of common security validations. The entropy from one or more internal noise sources continually feeds a TRNG, the output of which is then processed by software and hardware to generate the number returned by the UCL function. Analog Devices will work directly with the customer's accredited testing laboratory to provide any information regarding the TRNG that is needed to support the customer's validation requirements.

The general information in this section is provided only for completeness; customers are expected to use the Analog Devices UCL for the generation of random numbers. The TRNG engine can also be used to generate AES keys.

18.1 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, a soft reset, a POR, and the peripheral-specific resets.

Table 18-1: TRNG Register Summary

Offset	Register	Name
0x0000	*TRNG_CTRL*	TRNG Control Register
0x0004	*TRNG_STATUS*	TRNG Status Register
0x0008	*TRNG_DATA*	TRNG Data Register

18.1.1 Register Details

Table 18-2: TRNG Control Register

Cryptographic Control Register	TRNG_CTRL			[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	keywipe	R	0	**AES Key Wipe** Set this field to 1 to erase the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers. 0: Normal operation. 1: Initiate key wipe.
14:5	-	RO	0	**Reserved**
4	aeskg_sys	RO	0	**Reserved**
3	aeskg_usr	R/W	0	**AES Key Generate** Set this field to 1 to generate an AES key and store it in the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers. This bit is cleared by hardware automatically after the key is generated. 0: Normal operation. 1: Initiate AES key generation.
2	health_en	R/W	0	**Health Test Interrupt Enable** Set this field to 1 to generate an interrupt when
1	rnd_ie	R/W	0	**Random Number Interrupt Enable** This bit enables an interrupt to be generated when *TRNG_STATUS.rdy* = 1. 0: Disabled. 1: Enabled.

Cryptographic Control Register		TRNG_CTRL		[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
0	odht	R/W	0	**On-Demand Health Test** Set this field to 1 to start an on-demand health test.

Table 18-3: TRNG Status Register

TRNG Status Register			TRNG_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	aeskgd	R	0	**AES Key Generation Complete** This field is set to 0 after an AES key generation is complete and written to the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers. This field reads 1 when the key generation is in progress.
3	srcfail	R/W1C	0	**Source Fail** This field is set to 1 if the TRNG entropy source fails. An interrupt is generated if *TRNG_CTRL*.*health_en* = 1.
2	ht	R/W1C	0	**Health Test Status** This field is set to 1 if the on demand health test failed. This field is only valid once the *TRNG_STATUS*.*odht* reads 0.
1	odht	R	0	**On-Demand Health Test Complete** This field is set to 1 when the on-demand health test is complete. 0: ODHT complete. 1: ODHT in progress.
0	rdy	R	0	**Random Number Ready** Reading from *TRNG_DATA*.*data* clears this field to 0 while a new random number is generated. When new data is available, this field reads 1 and an interrupt is generated if *TRNG_CTRL*.*rnd_ie* = 1.

Table 18-4: TRNG Data Register

TRNG Data Register			TRNG_DATA	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	RO	0	**TRNG Data** The 32-bit random number generated is available here when *TRNG_STATUS*.*rdy* = 1.

19. ROM Bootloader

The ROM-based bootloader provides for program loading and verification. The physical interface between the external host and the bootloader is UART0.

All versions of the bootloader provide the ability to block access to program memory by disabling SWD.

Devices which provide the secure boot feature automatically verify the integrity of program memory after every reset.

Bootloader features:

- Command line interface.
- Programmable through UART at 115,200bps.
- LOCKED mode disables SWD and disallows any change to flash through bootloader.
- Transition from LOCK to UNLOCKED state erases all flash and the secret key before unlocking SWD.
- User-enabled PERMLOCKED state disables SWD and disables all commands except for program validation.

Devices which feature the trusted secure boot feature provide additional features:

- Automatic program memory integrity check using HMAC SHA-256 secret key after every reset. The device will halt and not execute the application software if the integrity check fails.
- Optional challenge/response protection of bootloader interface.

19.1 Instances

The dedicated pins and features of the bootloader are shown *Table 19-1*.

Table 19-1: MAX32670/MAX32671 Bootloader Instances

Part Number	Activation Pins		Bootloader	Secure Boot	Flash Memory Page Size
	UART0 RX	SWDCLK			
MAX32670GTL	P0.8	P0.1	Yes	No	8KB
MAX32671GTL	P0.8	P0.1	No	Yes	8KB

Versions incorporating secure boot functionality will not execute code unless there is a key loaded and the code has been properly signed with that key.

19.2 Bootloader Operating States

Each bootloader supports the modes shown in *Table 19-2*. Each bootloader state has a unique prompt.

Table 19-2: Bootloader Operating States and Prompts

State	Device Versions	Prompt
UNLOCKED	All device versions	“ULDR> “ <0x55> <0x4C> <0x44> <0x52> <0x3E> <0x20>
LOCKED	All device versions	“LLDR> “ <0x4C> <0x4C> <0x44> <0x52> <0x3E> <0x20>
PERMLOCK	All device versions	“PLLDR> “ <0x50> <0x4C> <0x4C> <0x44> <0x52> <0x3E> <0x20>
CHALLENGE	Only devices with secure boot feature	“<CR> “ <0x43> <0x52> <0x3E> <0x20>

State	Device Versions	Prompt
APPVERIFY	Only devices with secure boot feature	N/A

The *LOCK – Lock Device* and *PLOCK – Permanent Lock* commands do not change the bootloader prompt or take effect until the bootloader is reset.

19.2.1 UNLOCKED

The UNLOCKED state provides access to load secure keys and configuration information. Program loading, verification, and status is available in the UNLOCKED state. The SWD interface is available for use.

Transitioning from the LOCKED to UNLOCKED states erases all program memory. It also clears the challenge/response and application keys on devices with the secure boot feature.

The challenge and application keys can be erased by executing the Unlock command while in the UNLOCKED state and then resetting the device.

19.2.2 LOCKED

The LOCKED state disables access to program memory other than to verify it. It also disables the SWD interface. The application and challenge response keys cannot be changed without first changing to the UNLOCKED state.

If the optional challenge key is activated, the bootloader will start in the CHALLENGE state. Successfully completing the challenge/response will allow access to the PERMLOCKED or LOCKED prompt.

If the device provides the secure boot feature, the application and challenge key must be configured before executing the *LOCK – Lock Device* command.

19.2.3 PERMLOCKED

The PERMLOCKED state disables read/write access to program memory and keys. It also disables the SWD interface. The only functions available through the bootloader in this state are to verify program and read the USN.

If the optional challenge feature is activated, the bootloader will start in the CHALLENGE state. Successfully completing the challenge/response will access to the previous PERMLOCKED prompt.

If the device provides the secure boot feature, the application and challenge key must be configured before executing the *PLOCK – Permanent Lock* command.

19.2.4 CHALLENGE (Secure Boot Versions Only)

The CHALLENGE state provides an extra layer of security by requiring the host to authenticate itself using the HMAC SHA-256 key before executing any bootloader commands. If enabled, the device enters CHALLENGE mode following a reset if the external bootloader pins are active. CHALLENGE mode can be identified by the “CR> “ prompt.

In CHALLENGE mode, the host first requests a 128-bit random number (the challenge) from the bootloader using *GC – Get Challenge*. The host calculates the hash of the challenge using the mutually known HMAC SHA-256 key and sends it (the response) back to bootloader. The correct response transitions from CHALLENGE to the previous state of the bootloader. An incorrect response keeps the bootloader in the CHALLENGE state. The host must request a new challenge and send a response based on the new challenge. There is no limit to the number of challenge attempts.

19.2.5 APPVERIFY (Secure Boot Versions Only)

APPVERIFY is an internal state that invoked when the device is verifying the integrity of program memory.

The device performs an APPVERIFY:

- When executing a secure boot
- Immediately before executing the *LOCK – Lock Device* command
- Immediately before executing the *PLOCK – Permanent Lock* command

The device will not perform a secure boot until the HMAC SHA-256 secret-key is loaded.

Failure of the APPVERIFY process during a secure boot indicates corrupted or tampered program memory and disables code execution. If the bootloader is in the LOCKED state it can transition to the UNLOCKED state, erasing the program memory and keys so the device can be reprogrammed. There is no recovery from a secure boot failure in the PERMLOCKED state and the device must be discarded.

19.3 Creating and Loading the Motorola SREC File

The Analog Devices microSDK can directly generate SREC files that support devices with and without the secure boot feature. The information here is presented for completeness and is not necessary when using the microSDK.

19.3.1 Procedure for Devices Without the Secure Boot Feature

Devices without the secure boot feature use a standard SREC format generated directly from the binary.

1. Compile the source code and create the Motorola SREC file.
2. Activate the bootloader as described in the *Bootloader Activation* section.
3. Ensure the device is in the UNLOCKED state.
4. Execute the *L - Load* command and load the Motorola SREC file.
5. Execute the *V – Verify* command to verify the file was correctly loaded.

19.3.2 Procedure for Devices with the Secure Boot Feature

SREC files for devices with the secure boot feature must be modified to append the HMAC-256 hash to the binary before generating the SREC file as described below. Address records must be 32-bit aligned and the length of each line must be a multiple of 4 bytes. Unused memory locations within the program must be defined with 0xFF.

To generate the SREC file for devices with the secure boot feature:

1. Define the 128-bit HMAC secret key.
2. Generate the binary image.
3. Pad the binary image with 0xFF to the next 32-byte boundary.
4. Calculate the 32-byte HMAC SHA-256 hash using the secret key over the length of the padded binary image.
5. Append 32-byte hash to the binary image, after the last pad byte.
6. Generate SREC file of the modified binary image.

Follow this procedure to initialize and load a device with the secure boot feature:

1. Activate the bootloader as described in the *Bootloader Activation* section.
2. Ensure the device is in the UNLOCKED state.
3. Execute the *WL – Write Code Length* command.
4. Execute the *L - Load* command and load the SREC file.
5. Execute the *V – Verify* command to verify the file was correctly loaded.
7. Execute the *LK – Load Application Key* command to load the HMAC SHA-256 secret key.
8. Execute the *VK – Verify Application Key* command to verify the HMAC SHA-256 secret key was correctly loaded.
9. Execute the *AK – Activate Application Key* command. The device automatically verifies the program memory on all subsequent resets and attempts to execute the Lock and Plock commands.

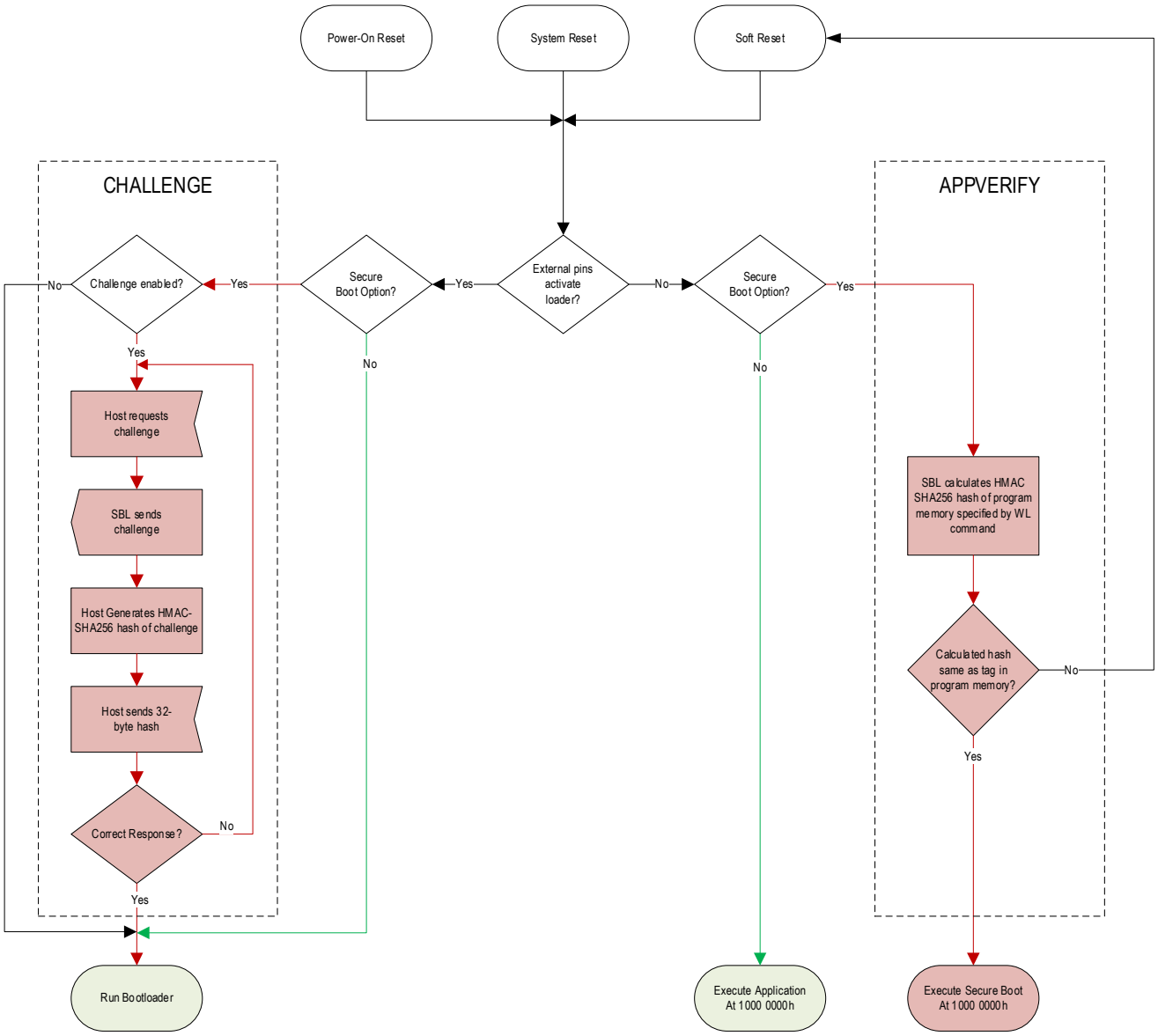
19.4 Bootloader Activation

The bootloader is invoked following a reset when the bootloader activation pin is asserted. The flow chart of the operation following a reset of the device is shown in *Figure 19-1*. Features exclusive to devices with the secure boot feature are highlighted in red.

Perform the following sequence to activate the bootloader:

1. The host asserts the UART0 Rx pin and SWDCLK pins low as shown in *Table 19-1*.
2. The host asserts RSTN pin low.
3. The host deasserts the RSTN pin.
4. Bootloader samples the UART0 Rx and SWDCLK pins immediately after reset. If they are both low, the hardware will activate the bootloader.
5. Bootloader performs its system initialization and configures the bootloader for 115,200bps.
6. The bootloader outputs the status prompt on the UART0 Tx pin. The prompt is unique for each bootloader state as shown in *Table 19-2*.
7. The host releases the UART0 Rx and SWDCLK pins once the host confirms the correct bootloader prompt.
8. The host begins bootloader session by sending commands on the UART0 Rx pin.

Figure 19-1: Combined Bootloader Flow



19.5 Secure Boot

The optional secure boot secure version of the bootloader provides additional features for secure and authenticated loading. These features are highlighted in *Figure 19-1*.

19.5.1 Secure Boot

Devices with the secure boot feature will perform a secure boot by entering the APPVERIFY state following reset in which the bootloader activation pins are not active. Failure of the secure boot will place the device in a reset loop to prevent execution of corrupted or tampered code. The device also enters APPVERIFY before completing the *LOCK – Lock Device* or *PLOCK – Permanent Lock* commands to ensure that the correct program memory and application key are loaded.

Failure of the secure boot will force the device into a continual reset state and no user code will be executed.

19.5.2 Secure Challenge/Response Authentication

The secure challenge/response authentication feature in secure boot devices provides an extra layer of security by requiring the host to authenticate itself using the mutual HMAC SHA-256 key before executing any bootloader commands. If the challenge key is activated, the device enters CHALLENGE mode following a reset if the external bootloader pins are active. The bootloader will display the CHALLENGE mode prompt shown in *Table 19-2*.

Only two commands are available in the CHALLENGE state.

Table 19-3: CHALLENGE Command Summary

Command
GC – Get Challenge
SR – Send Response

The host first requests a 128-bit random number (the challenge) from the bootloader. The host calculates the hash of the challenge using the HMAC SHA-256 key (the response) and sends it back to bootloader. The correct response transitions the bootloader from CHALLENGE mode to the LOCKED or PERMLOCKED states, depending on the last state of the bootloader.

Follow this procedure to enable the Challenge/Response feature in the UNLOCKED state:

1. The host generates the challenge/response HMAC SHA-256 secret key.
2. The host executes the LK command to load the challenge/response secret key. The key is sent in plaintext and should be done in a secure environment.
3. The host executes the VK command to verify the challenge/response secret key was correctly loaded.

The challenge/response will be required after the next device reset. It does not affect current operation until the next reset.

Follow this procedure to successfully perform the challenge/response:

1. The host executes the GC command.
2. The bootloader generates a 128-bit challenge and sends it to the host.
3. The host calculates the HMAC SHA-256 of the challenge to create the response.
4. The host executes the SR command with the calculated response. The SR command must be the first command sent to the bootloader after a GC command.

A correct response will return the prompt of the last bootloader state. An incorrect response will return an error message and the challenge/response prompt again. The host can perform steps 1-3 again to request another challenge from the bootloader. There is no limit on the number of challenge/response attempts.

Following a successful response, the bootloader will return the prompt corresponding to the last state of the bootloader.

19.6 Command Protocol

The bootloader presents a mode-specific prompt based on the current state of the loader as shown as in *Table 19-2*. The general format of commands is the ASCII character(s) of the command, followed by a <CR><LF> which is hexadecimal <0x0D><0x0A>. Commands with arguments always have a space (0x20) between the command mnemonic and the argument.

Commands arguments that are files always have the length specified in the file, so it is not necessary to follow the file with a <0x0D><0x0A>.

In general, arguments not related to security commands are prefixed with “0x” to denote hexadecimal input. Arguments for security commands in general are not prefixed with “0x”.

Always refer to the command description for the required format of the command.

19.7 General Commands

Table 19-4: General Command Summary

Command
L - Load
P – Page Erase
V – Verify
LOCK – Lock Device
PLOCK – Permanent Lock
UNLOCK – Unlock Device
H – Check Device
I – Get ID
S – Status
Q – Quit

19.7.1 General Command Details

L - Load	Load SREC File into Program Memory
Description	Load a Motorola SREC formatted file into flash program memory. After typing the L command, the bootloader will respond with “Ready to load SREC”, then transmit the file. The end of the file is detected automatically, so there is no need to send <0x0D><0x0A> at the end. If the secure boot feature is used, the files must be modified as described in *Procedure for Devices with the Secure Boot Feature*. The length reported by the success response for the modified files is the padded image plus the 32-bytes of the HMAC; this is different than the length used for the WL command.
Modes	UNLOCKED
Command	L<0x0D><0x0A> Ready to load SREC<0x0D><0x0A> [SREC File]
Response: Success	Load success, image loaded with the following parameters:<0x0D><0x0A> Base address: 0xnnnnnnnn<0x0D><0x0A> Length: 0xnnnnnnnn<0x0D><0x0A>
Response: Failure	Load failed.<0x0D><0x0A>

Table 19-5: P – Page Erase

P – Page Erase	Erase Page of Flash Program Memory
Description	Erases the page of memory associated with the 32-bit input address. Addresses must be aligned on the device-specific page boundaries.
Modes	UNLOCKED
Command	P 0xnnnnnnnn<0x0D><0x0A>
Response: Success	Erase Page Address: 0xnnnnnnnn<0x0D><0x0A>OK<0x0D><0x0A>
Response: Failure	Bad page address input<0x0D><0x0A> or Erase failed<0x0D><0x0A> or Invalid Page Address: 0xnnnnnnnn<0x0D><0x0A>

Table 19-6: V – Verify

V – Verify	Verify Flash Program Memory Against SREC File
Description	Verifies contents of flash program memory against a SREC file.
Modes	UNLOCKED
Command	V<0x0D><0x0A> Ready to verify SREC<0x0D><0x0A> [SREC File]
Response: Success	Verify success, image verified with the following parameters: <0x0D><0x0A> Base address: 0xnnnnnnnn<0x0D><0x0A> Length: 0xnnnnnnnn<0x0D><0x0A>
Response: Failure	Verify failed.<0x0D><0x0A>

Table 19-7: LOCK – Lock Device

LOCK – Lock Device	Lock Device
Description	Locks the device and disables SWD on the next device reset. See *LOCKED* section for a detailed description of this command. The effects of the Lock command do not take effect until the next time the device is reset. The bootloader will continue to display the locked prompt, but the *S – Status* command will show the Locked mode is active. The Lock command should be followed by the Q command (which generates a device reset) for the Lock command to take effect. Devices with the secure boot feature perform an APPVERIFY check before executing the Lock command. Failure of the Lock command means that the APPVERIFY check failed.
Modes	UNLOCKED
Command	LOCK<0x0D><0x0A>
Response: Success	OK<0x0D><0x0A>
Response: Failure	Failed<0x0D><0x0A>

Table 19-8: PLOCK – Permanent Lock

PLOCK – Permanent Lock	Permanently Lock Device
Description	Permanently locks the device if the argument matches the device's 13-bute USN. The effects of the Plock command do not take effect until the next time the device is reset. The bootloader will continue to display the LOCKED or UNLOCKED state prompt but the *S – Status* command will show the LOCKED or UNLOCKED state is active. The Lock command should be followed by the Q command (which generates a device reset) for the Lock command to take effect. Devices with the secure boot feature perform an APPVERIFY check before executing the PLock command. Failure of the PLock command means that the APPVERIFY check failed.
Modes	UNLOCKED LOCKED
Command	PLOCK <USN><0x0D><0x0A>
Response: Success	OK<0x0D><0x0A>
Response: Failure	Failed<0x0D><0x0A>

Table 19-9: UNLOCK – Unlock Device

UNLOCK – Unlock Device	Unlock Device
Description	Changes bootloader state to UNLOCKED if in LOCKED state. Erases all program memory and all bootloader keys. The SWD interface is re-enabled.
Modes	UNLOCKED LOCKED
Command	UNLOCK<0x0D><0x0A>
Response: Success	None. The device automatically resets itself and the bootloader will display the UNLOCKED mode prompt the next time the bootloader is activated.
Response: Failure	None.

Table 19-10: H – Check Device

H – Check Device	Perform SHA-256 Hash Over Memory Range
Description	Performs a simple SHA-256 (not HMAC SHA-256) hash of bytes starting at 32-bit address 0xnnnnnnnn to 0xmmmmmmmm. The hash must be a multiple of 64 bytes and the minimum hash input size is 512 bytes. The function returns the 32-byte hash value.
Modes	UNLOCKED LOCKED PERMLOCKED
Command	H 0xnnnnnnnn 0xmmmmmmmm<0x0D><0x0A>
Response: Success	yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>
Response: Failure	<0x0D><0x0A>

Table 19-11: I – Get ID

I – Get ID	Read Universal Serial Number
Description	Returns the 13-byte unique USN of the device.
Modes	UNLOCKED LOCKED PERMLOCKED
Command	I<0x0D><0x0A> USN: nnnnnnnnnnnnnnnnnnnnnnnnnn<0x0D><0x0A>
Response: Success	None
Response: Failure	None

Table 19-12: S – Status

S – Status	Read Device Status
Description	Returns the state of the loader and the application key and challenge key features. Executing the LOCK and PLOCK commands will immediately transition the device to that state and this command will reflect that state even before a reset occurs. The command prompt however will reflect the UNLOCKED state until the device is reset: The Lock <response> is: "Inactive" if the device is in the unlocked state. "Active" if the device is in the locked or permanent lock state. The PLock <response> is: "Inactive" if the device is in the unlocked or locked state. "Active" if the device is in the permanent lock state. In addition, devices with the secure boot feature will display: The Application Length <response> is: "Not Set" if the Write Code Length command has not previously loaded a non-zero value. "0xnnnnnnnn" which is the previously entered value using the Write Code Length command. The Application Key <response> is: "None Inactive" if no application key has been loaded using the LK command. "Loaded Inactive" if the application key has been loaded but the application key feature has not been activated by the AK command. "Loaded Active" If the application key has been loaded and the application key feature has been activated. The Challenge Key <response> is: "None Inactive" if no challenge key has been loaded using the LK command. "Loaded Inactive" if the challenge key has been loaded but the challenge key feature has not been activated by the AK command. "Loaded Active" if the challenge key has been loaded and the challenge key feature has been activated.
Modes	UNLOCKED
Command	S<0x0D><0x0A> Status<0x0D><0x0A> Lock: <response><0x0D><0x0A> PLock: <response><0x0D><0x0A> Application Length: <response><0x0D><0x0A> Application Key: <response><0x0D><0x0A> Challenge Key: <response><0x0D><0x0A>
Response: Success	None.
Response: Failure	None.

Table 19-13: Q – Quit

Q – Quit	Quit Bootloader Session
Description	Terminates the bootloader session and forces a reset of the device.
Modes	UNLOCKED LOCKED PERMLOCKED
Command	Q<0x0D><0x0A>
Response: Success	None
Response: Failure	None

19.8 Secure Commands

These commands are only supported on devices which provide the secure boot feature.

Table 19-14: Secure Command Summary

Command
LK – Load Application Key
LC – Load Challenge Key
VK – Verify Application Key
VC – Verify Challenge Key
AK – Activate Application Key
AC – Activate Challenge
WL – Write Code Length

19.8.1 Secure Command Details

Table 19-15: LK – Load Application Key

LK – Load Application Key	Load Application HMAC SHA-256 Key
Description	Write 128-bit HMAC secret key to nonvolatile memory. The key can only be written once until a *LOCK – Lock Device* or *PLOCK – Permanent Lock* command is executed.
Modes	UNLOCKED
Command	LK yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>
Response: Success	OK<0x0D><0x0A>
Response: Failure	Bad key input<0x0D><0x0A> or Key already loaded<0x0D><0x0A>

Table 19-16: LK – Load Challenge Key

LC – Load Challenge Key	Load Challenge Key
Description	Write 128-bit challenge key to nonvolatile memory. The key can only be written once until a *LOCK – Lock Device* or PLOCK – Permanent Lock command is executed.
Modes	UNLOCKED
Command	LC yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>
Response: Success	OK<0x0D><0x0A>
Response: Failure	Bad key input<0x0D><0x0A> or Key already loaded<0x0D><0x0A>

Table 19-17: VK – Verify Application Key

VK – Verify Application Key	VK – Verify Application Key
Description	Verify the application key against a value provided by the host.
Modes	UNLOCKED
Command	VK yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>
Response: Success	OK<0x0D><0x0A>
Response: Failure	Bad key input<0x0D><0x0A> or Error, no key loaded<0x0D><0x0A> or Key Mismatch<0x0D><0x0A>

Table 19-18: VC – Verify Challenge Key

VC – Verify Challenge Key	VC – Verify Challenge Key
Description	Verify the challenge key against a value provided by the host.
Modes	UNLOCKED
Command	VC yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>
Response: Success	OK<0x0D><0x0A>
Response: Failure	Bad key input<0x0D><0x0A> or Error, no key loaded<0x0D><0x0A> or Key Mismatch<0x0D><0x0A>

Table 19-19: AK – Activate Application Key

AK – Activate Application Key	Activate Application Key
Description	Activate application key. The device will perform an APPVERIFY following all resets and execution of the *LOCK – Lock Device* and PLOCK – Permanent Lock commands. The UNLOCK command deactivates the application key.
Modes	UNLOCKED
Command	AK<0x0D><0x0A>
Response: Success	OK<0x0D><0x0A>
Response: Failure	Key activate failed<0x0D><0x0A> or Error, no key loaded<0x0D><0x0A>

Table 19-20: AC – Activate Challenge Key

AC – Activate Challenge Mode	Activate Challenge Mode
Description	Activate CHALLENGE mode. All subsequent bootloader sessions in LOCKED or PERMLOCKED states will start in CHALLENGE mode. The “Key activate failed” response indicates the device has already activated the challenge/response feature.
Modes	UNLOCKED
Command	AC<0x0D><0x0A>
Response: Success	OK<0x0D><0x0A>
Response: Failure	Key activate failed<0x0D><0x0A> or Error, no key loaded<0x0D><0x0A>

Table 19-21: WL – Write Code Length

WL – Write Code Length	Write Code Length
Description	Write the length of the application code in bytes. The code length argument is address of the last pad byte as described in *Procedure for Devices with the Secure Boot Feature*. The "Write length failed" response indicates the WL command has already been performed. The host should re-enter the UNLOCKED state to clear the WL value and repeat the command.
Modes	UNLOCKED
Command	`WL 0xnnnnnnnn<0x0D><0x0A>`
Response: Success	`Length set to: 0xnnnnnnnn<0x0D><0x0A>`

Response: Failure	`Bad length input<0x0D><0x0A>`
	Or
	`Write length failed<0x0D><0x0A>`

19.9 Challenge/Response Commands

Table 19-22: Challenge/Response Command Summary

Command
GC – Get Challenge
SR – Send Response

19.9.1 Challenge/Response Command Details

Table 19-23: GC – Get Challenge

GC – Get Challenge	Get Challenge
Description	Bootloader generates a 16-byte hexadecimal ASCII challenge and transmits it to host. The challenge is sent MSB first.
Modes	LOCKED PERMLOCKED
Command	GC<0x0D><0x0A>
Response: Success	0123456789ABCDEF0123456789ABCDEF<0x0D><0x0A>
Response: Failure	None

Table 19-24: SR – Send Response

SR – Send Response	Send Response
Description	Host calculates HMAC SHA-256 on the 16-byte challenge and sends the 32-byte hexadecimal ASCII response. The response is sent MSB first.
Modes	LOCKED PERMLOCKED
Command	SR 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF<0x0D><0x0A>
Response: Success	OK<0x0D><0x0A>
Response: Failure	Bad response input<0x0D><0x0A> or Verification failed<0x0D><0x0A> or Error, request challenge<0x0D><0x0A>

20. 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.

20.1 Instances

The DAP interface communicates through the serial wire debug (SWD), shown in *Table 20-1*.

Table 20-1: MAX32670/MAX32671 DAP Instances

Instance	Pin	Alternate Function	SWD Signal	POR Default
Primary	P0.0	AF1	SWDIO	✓
	P0.1	AF1	SWDCLK	✓
Secondary	P0.20	AF4	SWDCLKB	-
	P0.22	AF4	SWDIOB	-

20.2 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.

20.2.1 Locking the DAP

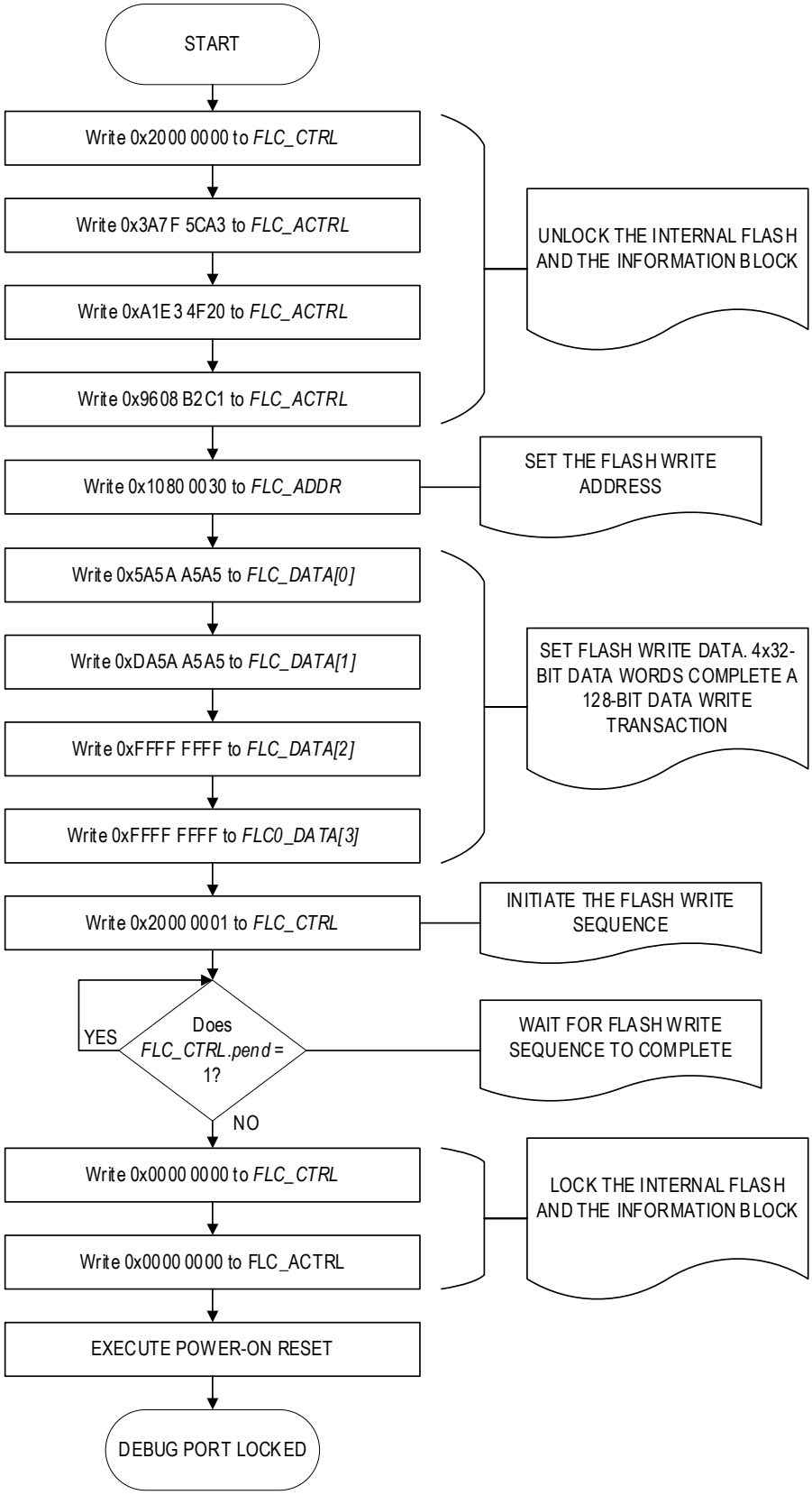
There are two options for locking out the debug access port. Option 1 locks the DAP and makes it available to be unlocked later. This is a one-time-only process. The DAP port cannot be relocked. Option 2 locks the DAP permanently.

20.2.1.1 Option 1

To lock the DAP and make it available to be unlocked later (one time only), follow the flow chart in *Figure 20-1*.

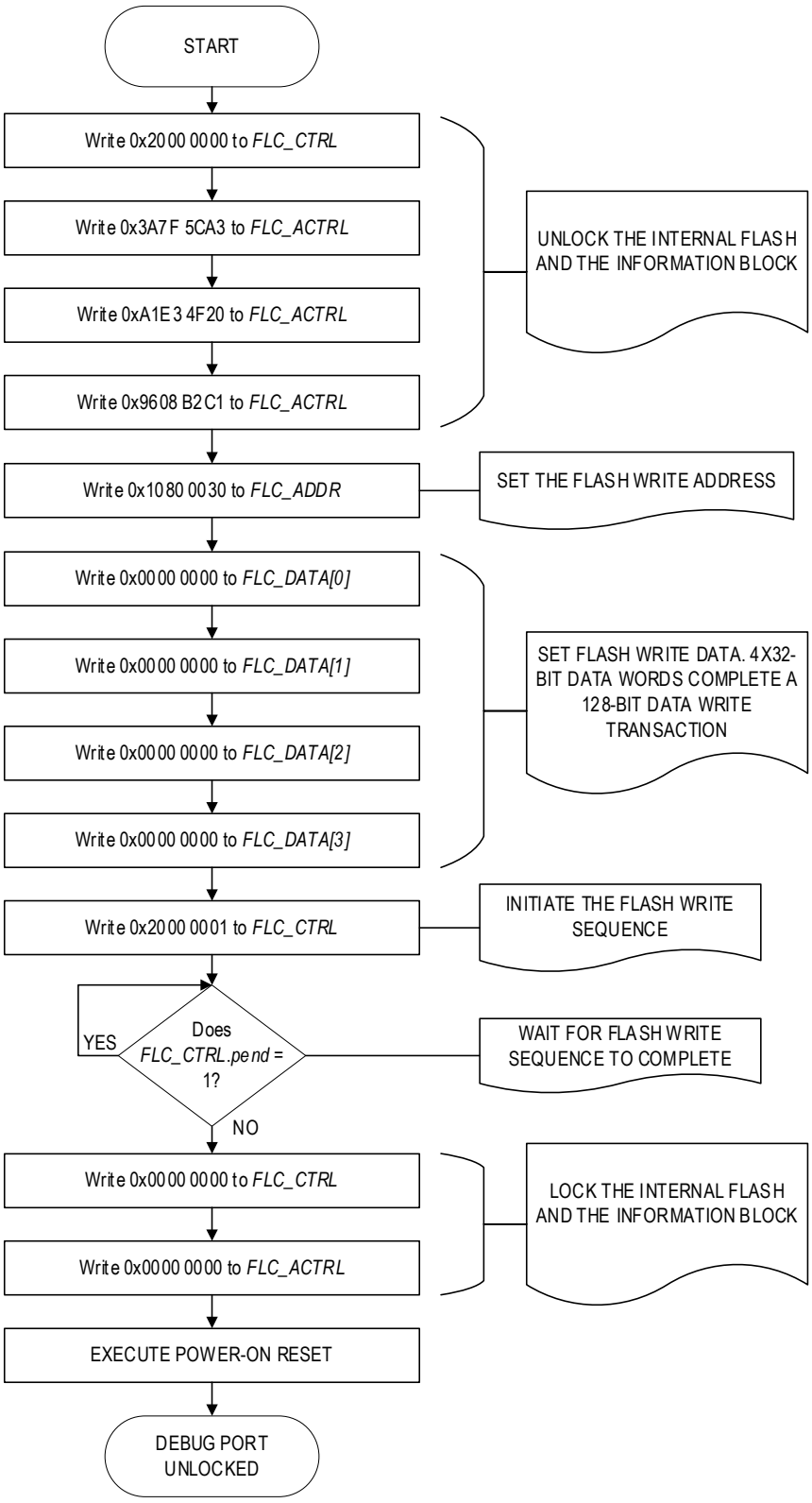
CoreSight is a registered trademark of Arm Limited.

Figure 20-1: 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 20-1*, follow the flow chart in *Figure 20-2*.

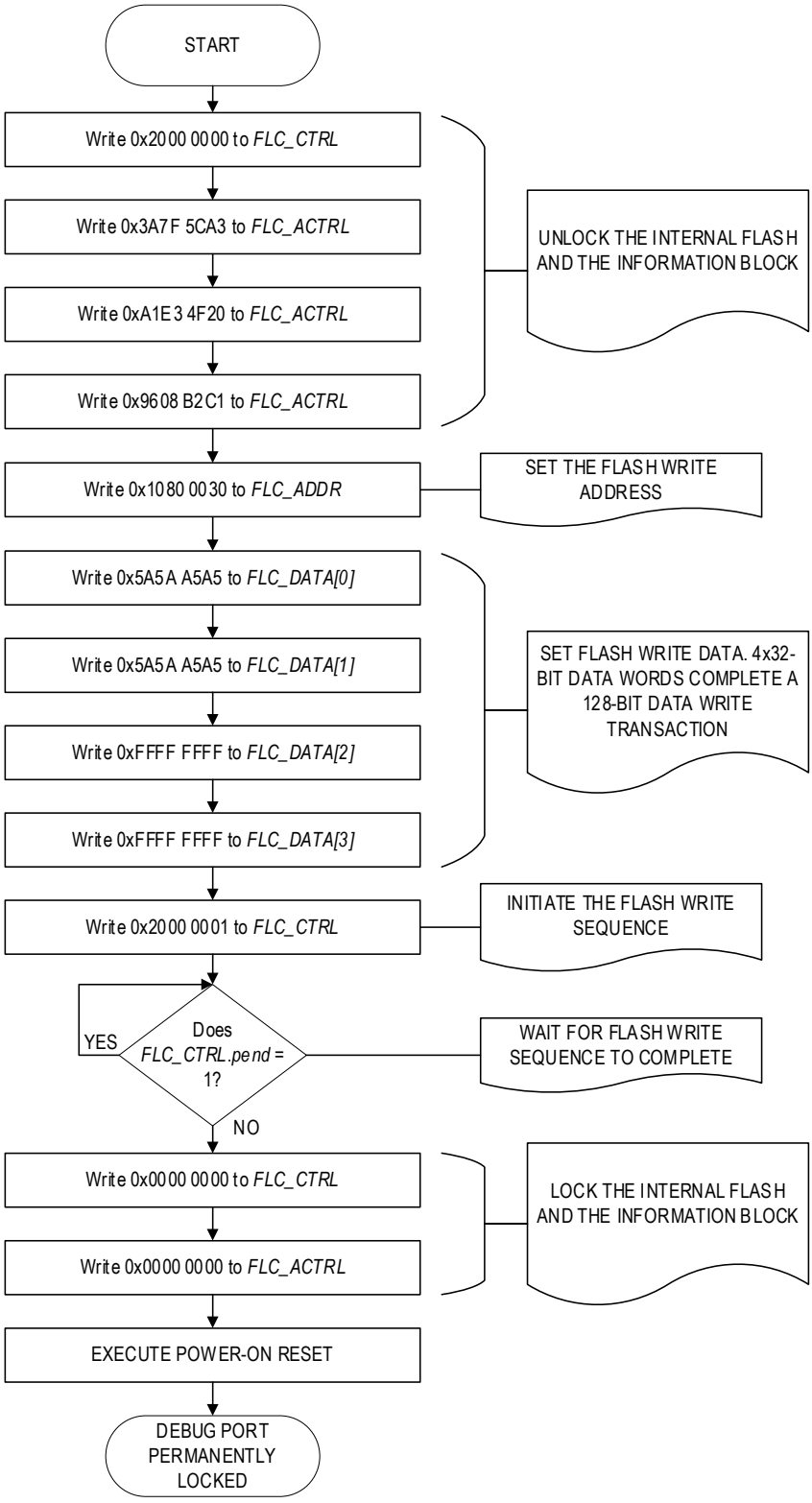
Figure 20-2: Unlocking the DAP After Being Locked as in Figure 20-1



20.2.1.2 Option 2

To lock the DAP permanently, follow the flow chart in *Figure 20-3*.

Figure 20-3: Locking the Debug Access Port Permanently



20.3 Pin Configuration

Instances of SWD signals in GPIO and alternate function matrices are for determining which GPIO pins are associated with a signal. It is not necessary to configure the default SWD pins for an alternate function to use the DAP following a POR.

20.3.1 Switching Between SWD Alternate Functions

Only one set of SWD can be enabled at a time. If software needs to switch between the primary SWD pins (P0.0 and P0.1) and the secondary SWD pins (P0.20 and P0.22), software must first disable the primary pins by setting them to I/O mode or a different alternate function, before enabling the secondary SWD pins for alternate function 4.

21. Silicon Revision Differences

The current silicon revision of the MAX32670/MAX32671 is B2 and B1. 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/MAX32670*.

21.1 Differences Between the B2 and B1 Revision

- The *GCR_REVISION*.*revision* field reads 0x01B2.
- The IBRO is the default system oscillator after a POR, system reset, and watchdog reset.
- The IPO is powered off by default after a POR, system reset, and watchdog reset.
- The *PWRSEQ_LPCN*.*vcoremon_dis* field is read-only.

21.2 Differences Between the B1 and A3 Revision

- The *GCR_REVISION*.*revision* field reads 0x00B1.
- The IPO is the default system oscillator after a POR, system reset, and watchdog reset.
- The IBRO is powered off by default after a POR, system reset, and watchdog reset.
- The *PWRSEQ_LPCN*.*ovr* setting of 0 changed from a divide by 4 to a divide by 8 of the system clock.
- Switching the system clock to the IPO from another clock source does not require waiting for the clock switch manually.
- Software must ensure that the flash is not being programmed or erased before entering a low-power mode.
- The USN is readable without having to unlock the *Information Block Flash Memory*.
- Added the *MCR_LPPIOCTRL* register, which is used to enable the low-power peripheral's to control their associated GPIO pins.
- The watchdog timer's have updated protection sequences for feed, enable, and disable. See *WDT Protection Sequence* for details. Software written for earlier revisions must be updated to work correctly with the new protection requirements.

21.3 Differences Between the A3 and A2 Revision

- The *GCR_REVISION*.*revision* field reads 0x00A3.
- The IPO is the default system oscillator after a POR, system reset, and watchdog reset.
- The IBRO is powered off by default after a POR, system reset, and watchdog reset.

21.4 Differences Between the A2 and A1 Revision

- The *GCR_REVISION*.*revision* field reads 0x01A2.
- The IBRO is the default system oscillator after a POR, system reset, and watchdog reset.
- The IPO is powered off by default after a POR, system reset, and watchdog reset.
- The *Flash Registers* are reset on a system reset as well as a POR.
- The low-power UART does not generate an extra stop bit after each packet.
- The ERTCO power-down bit moved from *GCR_CLKCTRL*[*17*] to *PWRSEQ_LPCN*[*31*].

21.5 Initial Silicon Revision A1

- The *GCR_REVISION*.*revision* field reads 0x00A1.
- The default system oscillator after a POR, system reset, and watchdog reset is the IPO.
- The IBRO is powered off by default.

22. Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION
0	06/17/2020	Initial release
1	11/02/2020	Updated all field names and register names for all chapters. Updated the *General-Purpose I/O (GPIO) and Alternate Function Pins* chapter. Updated the *System, Power, Clocks, Reset* chapter.
2	12/11/2020	Updated *General-Purpose I/O (GPIO) and Alternate Function Pins*. Description of differences between devices in the family. Updated *System, Power, Clocks, Reset*. Full chapter update. Updated *Real-Time Clock (RTC)*. Corrected RTC_CTRL register to add sqw_sel field. Update *Flash Controller (FLC)*. Added FLC_WELR0, FLC_WELR1, FLC_RLR0, and FLC_RLR1 register definitions. Updated *Inter-Integrated Sound Interface (I2S)*. Full chapter update. Updated *Timers (TMR/LPTMR)*. Full chapter update. Updated *Revision History*. Added the Device Options chapter.
3	10/22	Updated *Timers (TMR/LPTMR)* chapter. Full chapter update. Updated *Real-Time Clock (RTC)* chapter to reflect correct output frequencies. Updated *Inter-Integrated Sound Interface (I2S)* chapter to detail requirements for *Controller and Target Mode Configuration*. Updated *WDT Protection Sequence* with updated feed, enable, and disable sequences. Updated *WDT Protection Sequence* to show 8-bit writes for feed, enable, and disable. Updated to Analog Devices style template. Added *Introduction* chapter. Added *Table 4-11 BACKUP* RAM retention table. Added details on *AES Key Storage* in information block. Added on demand health test fields to TRNG registers (*TRNG_CTRL* and *TRNG_STATUS*). Added *MCR_LPPIOCTRL* register fields for LPTMR and LPUART I/O control. Updated Device Options with latest die revisions. Updated *Low-Power Receiver Operation* BAUD rate calculation example. Updated *GPIOn_WKEN* register to indicated reserved. Updated *Universal Asynchronous Receiver/Transmitter (UART)* feature list. Updated *Standard DMA (DMA)* to clarify access is limited to SRAM and not flash. Added details on *Using GPIOWAKE for Wake-Up from DEEPSLEEP, BACKUP, and STORAGE*. Corrected GPIO *Alternate Function* selection instructions. Corrected size of *sysram4* and *sysram5* in *Table 3-1*. Updated instructions for reading USN in *Information Block Flash Memory*. Added note that flash cannot be erasing or writing when entering low-power modes in *Flash Controller (FLC)*. Updated description of *GCR_SYSCTRL.romdone* field. Updated *PWRSEQ_LPCN.fastwk_en* field description. Updated the *FLC_CTRL.lve* field description. Updated *Internal Cache Controller (ICC)* usage to indicate cache should be flushed before enabling. Updated *Flash Write* and *Page Erase* usage to indicate cache should be flushed after a write to flash. Marked *GCR_RST0* bit 17 as reserved. Marked *MCR_RST.rtc* as the RTC reset field. Updated RTC block diagram (*Figure 15-1*) with correct register and field name. Updated *GCR_MEMCTRL.fws* description to indicate 0 is a valid setting. Updated I^2C usage to show target address matching in *Target Mode Operation*. Updated address offset of *I2Cn_SLAVE* register. Added note to LPUART instance table (*Table 9-1*) indicating INRO frequency varies significantly across temperature and voltage. Added *PWRSEQ_GP0* and *PWRSEQ_GP1* registers. Corrected field location of *GCR_PCLKDIV.div_clk_out_en* as bit 16 not 12.

REVISION NUMBER	REVISION DATE	DESCRIPTION
		Marked reserved fields in *GCR_PCLKDIS0* and *GCR_PCLKDIS1* registers as do not modify. Updated *Figure 3-1*, *Figure 3-2*, *Figure 4-2*, *Figure 4-3*, *Figure 4-4*, and *Figure 4-5* to improve readability. Updated the terms "master/slave" to "controller/target" throughout except for the *I2Cn_SLAVE* register to maintain backward compatibility with existing software releases. Updated *Power-On Reset (POR)* section, to indicate GPIO are only reset on POR. Added note to *GPIO Registers* and *Power Sequencer and Always-On Domain Registers (PWRSEQ)* indicating only a POR affects the register settings. Marked *Flash Registers* as only reset on POR.
4	9/23	Added PWRSEQ_LPCN.*ertco_pd* bit that is required to power on the ERTCO before enabling the ERTCO. Added *Enabling the ERTCO* instructions. Added note to instance section of *Real-Time Clock (RTC)* chapter to point to instructions for enabling the ERTCO for use with the RTC. Added *GCR_CLKCTRL*.*ertco_en* bit required to enable ERTCO. Updated *TRNG_STATUS* register to correctly document each register field. Corrected AES usage instructions to indicate clearing *GCR_PCLKDIS1*.*aes* enables the AES peripheral. Simplified GPIO *Alternate Function* selection instructions. Corrected *Power-On Reset (POR)* section to indicate GPIO are reset on peripheral reset, soft reset, system reset and POR. Removed note on *GPIO Registers* indicating only a POR affects the register settings. Added details on entering low-power modes. See *Entering SLEEP*, *Entering DEEPSLEEP*, *Entering BACKUP*, and *Entering STORAGE* for details. Removed ECC throughout. Removed RTC load capacitor section. No stability capacitors should be used on the ERTCO with this device. Updated AOD_PCLK to AOD_CLK throughout. Updated flash controller *Clock Configuration* section to clarify that the 1MHz clock requirement is for write and erase operations. Removed flash write width field from the *FLC_CTRL* register, this field is not supported. Added caution that the *FLC_CLKDIV*.*clkdiv* must be set on all forms of reset before performing a write or erase operation. Added *RTC_OSCCTRL*.*ibias_en*, *RTC_OSCCTRL*.*hyst_en*, *RTC_OSCCTRL*.*ibias_sel*, and *RTC_OSCCTRL*.*filter_en* fields. Updated *TMRn_CNT* and *TMRn_PWM* registers to indicate the timer's clock must be enabled (*TMRn_CTRL0*.*clken* = 1)before writes to those registers. Added steps for trimming the IPO to the *IPO Calibration* section. Clarified width of one-shot output pulse in Timer chapter. See for *One-Shot Mode (0)* details. Updated *Table 12-4* to show correct polarity and phase settings for each SPI mode. Updated *Table 15-2* to show correct fields for write access to *RTC_TODA* and *RTC_SSECA* registers. Updated *Table 19-1* to show correct bootloader instances and to match device data sheet. Updated *DMA_CHn_CNTRLD*.*en* to reserved. Clarified DMA *Usage* instructions to not use the *DMA_CHn_CNTRLD*.*en* field. Clarified DMA *Usage* that if using the auto reload feature, both *DMA_CHn_CTRL*.*en* and *DMA_CHn_CTRL*.*rlden* fields must be set to 1 simultaneously. Replaced Device Options chapter with *Silicon Revision Differences* and enhanced details of device options and differences between revisions. Clarified *Creating and Loading the Motorola SREC File* procedures. 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*.