MAX32660 USER GUIDE

UG6659; Rev 1; 07/2021

Abstract: This user guide provides application developers information on how to use the memory and peripherals of the MAX32660 microcontroller. Detailed information for all registers and fields in the device are covered. Guidance is given for managing the clocks, power and initialization of the product.

Table of Contents

Table of Tables

Table of Figures

Table of Equations

1. Introduction

For ordering information, mechanical and electrical characteristics for the MAX32660 family of devices, please refer to the datasheet. For information on the Arm® Cortex®-M4 with FPU core, please refer to the *Arm Cortex-M4 Processor Technical Reference Manual*.

1.1 Related Documentation

The MAX32660 datasheet and errata are available from the Maxim Integrated® website, *http://www.maximintegrated.com/MAX32660*.

1.2 Conventions Used in this Document

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/W0	**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/W1	**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 3-1* 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: Name 0 Register

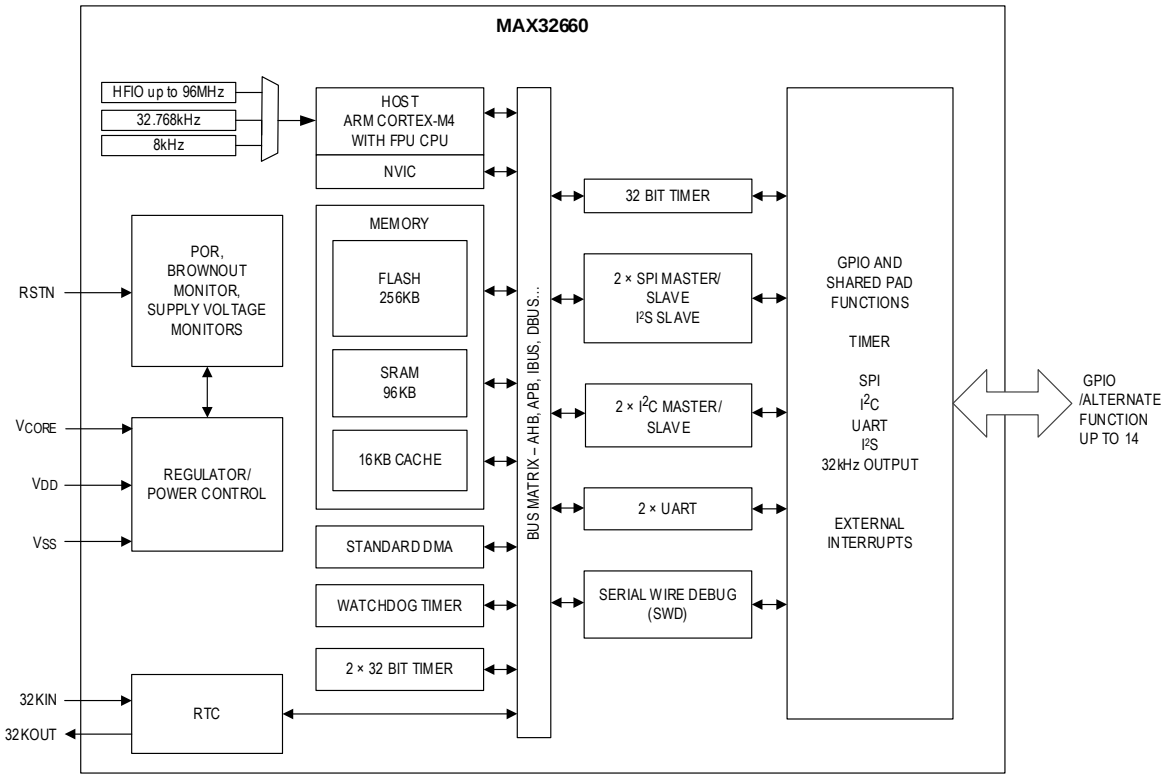
Name 0			**REG_NAME0**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	-	**Reserved**
15:0	field_name	R/W	0	**Field name description** Description of *field_name*.

2. Overview

The MAX32660 is an ultra-low-power, cost-effective, highly integrated microcontroller designed for battery-powered devices and wireless sensors. It combines a flexible and versatile power management unit with the powerful Arm Cortex-M4 with FPU (Floating Point Unit). The device enables designs with complex sensor processing without compromising batt1ery life. It also offers legacy designs an easy and cost-optimal upgrade path from 8 or 16-bit microcontrollers. The device integrates up to 256KB of flash memory and 96KB of system RAM to accommodate application software.

The device features four powerful and flexible power modes. It can operate from a single supply battery voltage or a dual supply typically provided by a PMIC. The I^2C (Inter-Integrated Circuit) port supports standard, fast, fast-plus, and high-speed modes operating up to 3400Kbps. The SPI ports can run up to 48MHz in both master and slave mode, and the UARTs can run up to 4Mbps. Three general-purpose 32-bit timers, a watchdog timer, and a real-time clock are also provided. An I^2S (Inter-IC Sound) interface provides audio streaming to or from an external audio codec.

Figure 2-1: MAX32660 High-Level Block Diagram



3. Memory, Register Mapping, and Access

3.1 Overview

The Arm Cortex-M4 core 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 can also be accessed, depending on the implementation, in 8-bit (1 byte) or 16-bit (2 byte) widths. The total range of the memory space is 32-bits in width (4GB addressable total), from addresses 0x0000 0000 to 0xFFFF FFFF.

All flash, system RAM, the CPU, and peripheral registers are mapped between address 0x0000 0000 and 0x5FFF FFFF. See *Figure 3-1* and *Figure 3-2* for details on the mapping.

Figure 3-1: Code Memory Mapping

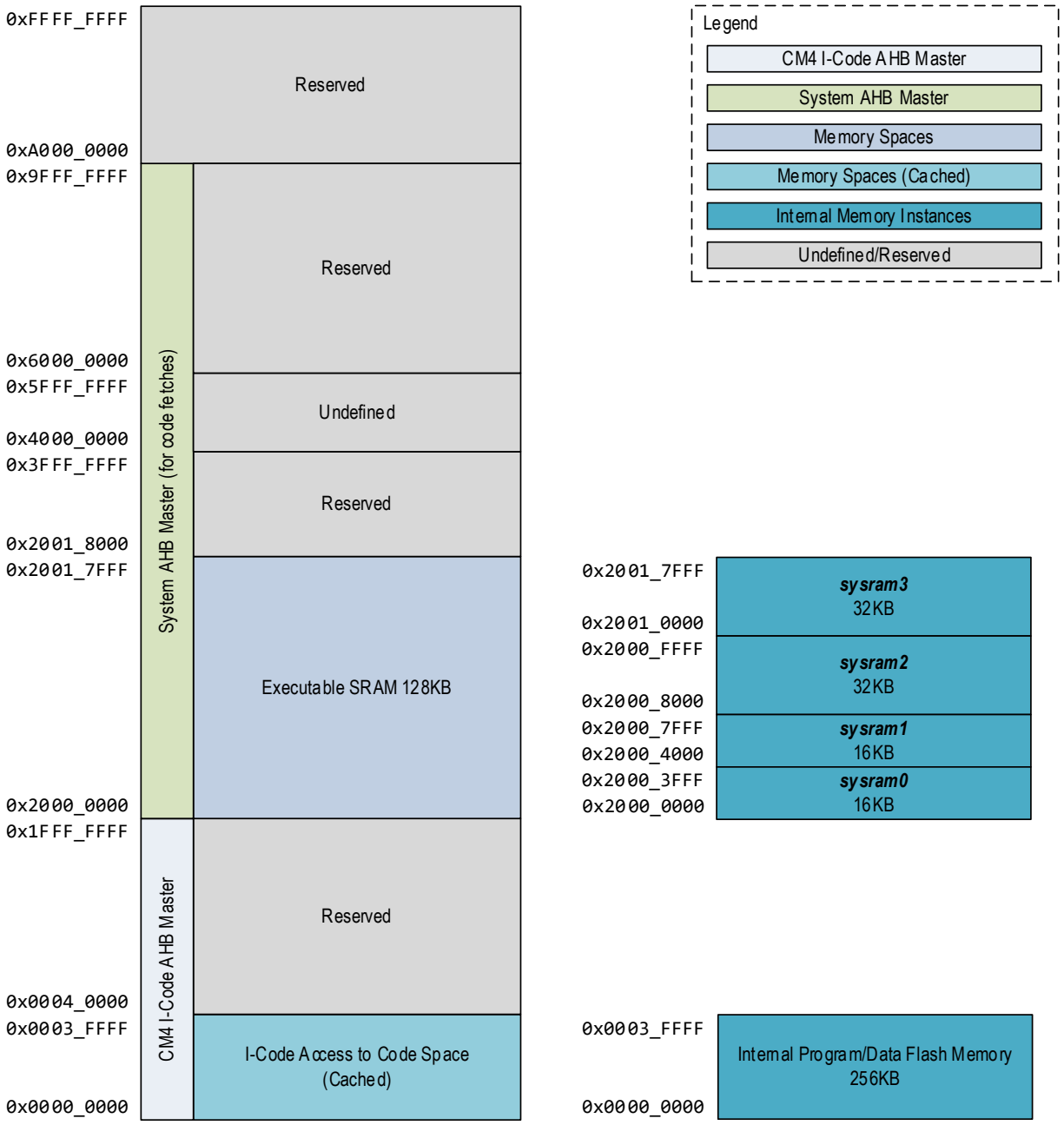
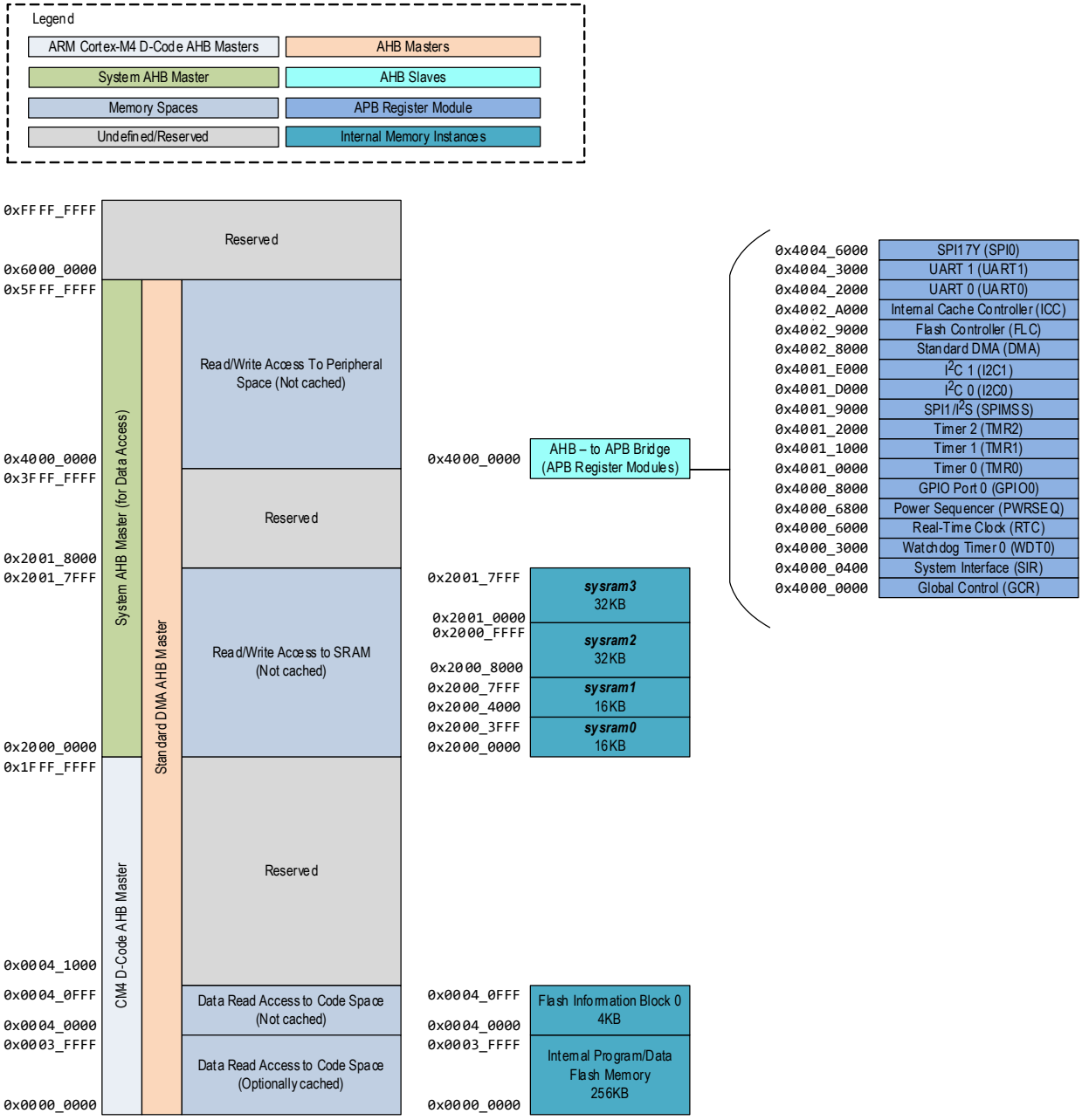


Figure 3-2: Data Memory Map



3.2 Standard Memory Regions

Several standard memory regions are defined for the Arm Cortex-M4 architecture; the use of many of these is optional for the system integrator. At a minimum, the MAX32660 must contain code and data memory for application code, stack and heap usage, and certain components of the implemented architecture.

3.2.1 Code Space

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

On the MAX32660, the code space memory area contains the main internal flash memory that typically contains the instruction code executed on the device. The internal flash memory is mapped into both code and data space from 0x0000 0000 to 0x0003 FFFF. This program memory area must also contain the default system vector table and the initial settings for all system exception handlers and interrupt handlers. The reset vector for the device is 0x0000 0000. See *Interrupts and Exceptions* for details of the device interrupt vector table.

3.2.2 SRAM Space

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

On the MAX32660, this memory area contains the main system SRAM, 96KB, that is mapped from 0x2000 0000 to 0x2001 7FFF.

The entirety of the SRAM memory space on the MAX32660 is contained within the dedicated Arm Cortex-M4's SRAM bit-banding region from 0x2000 0000 to 0x200F FFFF (1MB maximum for bit-banding). The CPU can access the entire SRAM either using standard byte/word/doubleword access or bit-banding operations. The bit-banding mechanism allows any single bit of any given SRAM byte address location to be set, cleared, or read individually by reading from or writing to a corresponding doubleword (32-bit wide) location in the bit-banding alias area.

The alias area for the SRAM bit-banding is located beginning at 0x2200 0000 and is a total of 32MB maximum, which allows the entire 1MB bit banding area to be accessed. Each 32-bit (4 byte aligned) address location in the bit-banding alias area translates into a single bit access (read or write) in the bit-banding primary area. Reading from the location performs a single bit read while writing either a 1 or 0 to the location performs a single bit set or clear.

Note: The Arm Cortex-M4 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. Bit-banding is a core function (i.e., not a function of the SRAM memory interface layer or the AHB bus layer). This is only applicable to accesses generated by the core itself. Reads/writes to the bit-banding alias area by other (non-Arm -core) bus masters such as the Standard DMA AHB bus master does not trigger a bit-banding operation and instead results in an AHB bus error.

The SRAM area on the MAX32660 is capable of code execution. 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 MAX32660 specific AHB bus masters can also access the SRAM for general storage or working space.

3.2.3 Peripheral Space

The peripheral space area of memory is intended to map control registers, internal buffers/working space, and other features needed for the firmware control of non-core peripherals. It is defined from byte address range 0x4000 0000 to 0x5FFF FFFF (0.5GB maximum). On the MAX32660, all device-specific module registers are mapped to this memory area and any local memory buffers or FIFOs required by modules.

As with the SRAM region, there is a dedicated 1MB area at the bottom of this memory region (from 0x4000 0000 to 0x400F FFFF) used for bit-banding operations by the Arm Cortex-M4. Four-byte-aligned read/write operations in the

peripheral bit-banding alias area (32MB in length, from 0x4200 0000 to 0x43FF FFFF) are translated by the core into read/mask/shift or read/modify/write operation sequences to the appropriate byte location in the bit-banding area.

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

On the MAX32660, 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 slower, easier to handle 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 primary instruction and data fetching.

3.2.4 System Area (Private Peripheral Bus)

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

In addition to being restricted to the core, application code 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.

3.3 Device Memory Instances

This section details physical memory instances on the MAX32660 (including internal flash memory and SRAM instances) accessible as standalone memory regions using either the AHB or APB bus matrix. Memory areas that are only accessible through FIFO interfaces, or memory areas consisting of only a few registers for a peripheral, are not covered here.

3.3.1 Main Program Flash Memory

The main program flash memory is 256KB in size and consists of 32 logical pages of 8,192 bytes per page.

3.3.2 Instruction Cache Memory

The internal flash memory instruction cache is 16KB in size and is used to cache instructions fetched using the I-Code bus. This includes instructions fetched from the internal flash memory.

Note: The cache is used for instruction fetches only. Data fetches (including code literal values) from the internal flash memory do not use the instruction cache.

3.3.3 Information Block Flash Memory

The information block is a separate flash instance used to store trim settings (option configuration and analog trim) and other nonvolatile device-specific information. The information block also contains the Unique Serial Number (USN). The USN has three parts labeled USN0, USN1, and USN2. USN0 is 60 bits, while USN1 and USN2 are both 64 bits. Each USN can be read but cannot be written.

Figure 3-3: Unique Serial Number Format

		Bit Position																														
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Address	0x40000	0	0	USN0 bits 29 - 0																													
	0x40004	0	0	USN0 bits 30 - 59																													
	0x40008	USN1 bits 31 - 0																															
	0x4000C	USN1 bits 32 -63																															
	0x40010	USN2 bits 31 - 0																															
	0x40014	USN bits 32 - 63																															

3.3.4 System SRAM

The system SRAM is 96KB in size and can be used for general-purpose data storage, the Arm Cortex-M4 system stack, and software code execution.

3.4 AHB Bus Matrix and AHB Bus Interfaces

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

3.4.1 Core AHB Interface

3.4.1.1 I-Code

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

3.4.1.2 D-Code

The Arm Cortex-M4 core uses the D-Code AHB master for data fetches from memory instances located in code space from byte addresses 0x0000 0000 to 0x1FFF FFFF. The D-Code AHB master has access to the entire internal flash memory.

3.4.1.3 System

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

3.4.2 AHB Slaves

3.4.2.1 Standard DMA

The standard DMA AHB slave has access to all off-core memory areas accessible by the system bus. It does not have access to the Arm Private Peripheral Bus area.

3.5 Peripheral Register Map

3.5.1 APB Peripheral Base Address Map

Table 3-1 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 register's address within the peripheral is defined as the peripheral base address plus the register's offset.

Table 3-1: APB Peripheral Base Address Map

Peripheral	Peripheral Register Prefix	Base Address	End Address
Global Control	GCR_	0x4000 0000	0x4000 03FF
System Interface	SIR_	0x4000 0400	0x4000 07FF
Watchdog Timer 0	WDT0_	0x4000 3000	0x4000 33FF
Real-Time Clock	RTC_	0x4000 6000	0x4000 63FF
Power Sequencer	PWRSEQ_	0x4000 6800	0x4000 6BFF
GPIO Port 0	GPIO0_	0x4000 8000	0x4000 8FFF
Timer 0	TMR0_	0x4001 0000	0x4001 0FFF
Timer 1	TMR1_	0x4001 1000	0x4001 1FFF
Timer 2	TMR2_	0x4001 2000	0x4001 2FFF
SPIMSS (SPI1/I^2S)	SPIMSS_	0x4001 9000	0x4001 9FFF
I^2C 0	I2C0_	0x4001 D000	0x4001 DFFF
I^2C 1	I2C1_	0x4001 E000	0x4001 EFFF
Standard DMA	DMA_	0x4002 8000	0x4002 8FFF
Flash Controller	FLC_	0x4002 9000	0x4002 93FF
Internal Cache Controller	ICC_	0x4002 A000	0x4002 AFFF
UART 0	UART0_	0x4004 2000	0x4004 2FFF
UART 1	UART1_	0x4004 3000	0x4004 3FFF
SPI0	SPI17Y_	0x4004 6000	0x4004 6FFF

4. System, Power, Clocks, Reset

The MAX32660 includes a high-frequency internal oscillator (HFIO), an 80kHz nano-ring oscillator (NANO), and support for an external 32kHz crystal or external clock. Support for selectable core operating voltage is provided, and the HFIO frequency is scaled based on the specific core operating voltage range selected.

4.1 Core Operating Voltage Range Selection

The MAX32660 supports three selections for the core operating voltage range (OVR). In a single-supply operation, changing the OVR sets the internal LDO regulator's output to the voltage shown in *Table 4-1*. In a dual-supply design, setting the OVR allows an external PMIC to dynamically provide the required V_{CORE} voltage. Changing the OVR also reduces the output frequency of the HFIO, further reducing power consumption.

Changes to the OVR affect the internal flash memory access time, 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 reduces the output frequency of the HFIO immediately, as shown in *Table 4-1*. Operating the device using dual external supplies requires special considerations and must be handled carefully in software.

Changing the core operating voltage reduces the output frequency of the HFIO immediately, as shown in *Table 4-1*. When operating the MAX32660 using dual external supplies requires special considerations and must be handled carefully in the application software.

Table 4-1: OVR Selection and the Effect on V_{CORE} and SYS_OSC

PWRSEQ_LP_CTRL ***ovr***	*FLC_CN* ***lve***	**V_{CORE} Typical**	**SYS_OSC**		
			f_{HFIO} (MHz)	**f_{NANO} (kHz)**	**f_{X32K} (kHz)**
0	1	0.9	24	80	32.768
1	1	1.0	48	80	32.768
2	0	1.1	96	80	32.768

4.1.1 Setting the Operating Voltage Range

The OVR selection is controlled using the power sequencer low-power control register field *PWRSEQ_LP_CTRL*.*ovr* which is only reset by a power-on reset (POR). This field 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 HFIO. Before adjusting the OVR settings, it is required to set the system clock to either the 80kHz NANO or the 32.768kHz external RTC oscillator. The device coordinates OVR change between the internal LDO and the HFIO 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 HFIO as the default SYS_OSC:

1. Set *PWRSEQ_LP_CTRL*.*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 32.768kHz external RTC oscillator or the 80kHz NANO as the system clock source.
   a. See the *Oscillator Sources and Clock Switching* section for details on the system clock selection.

3. Set the number of flash wait states to 5 (GCR_MEMCKCN.*fws* = 5) to ensure flash operation at any frequency.
4. Set PWRSEQ_LP_CTRL.*ovr* to either 0, 1, or 2, as shown in Table 4-1.
5. Set FLC_CN.*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 HFIO and update the system clock prescaler to the desired value.
   a. Set GCR_CLKCN.*clksel* = 0.
   b. Wait for the system clock ready bit, GCR_CLKCN.*ckrdy*, to read 1.
   c. Set GCR_CLKCN.*psc* to the desired prescaler value.
7. Set GCR_MEMCKCN.*fws* to the minimum value shown for the selected system clock and OVR. See Table 4-2, Table 4-3, or Table 4-4 for details.
8. Set GCR_RSTR0.*prst* = 1 to perform a peripheral reset.

On each subsequent non-POR reset event:

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

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

4.1.2 Flash Wait States

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

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

Table 4-2: Minimum Flash Wait State Setting for Each OVR Setting ($f_{SYSCLK} = f_{HFIO}$)

Core Operating Voltage Range Setting		Core Voltage Range	f_{HFIO} (MHz)	System Clock Prescaler	System Clock	Minimum Flash Wait State Setting
PWRSEQ_LP_CTRL.*ovr*	FLC_CN.*lve*	V_{CORE} (V)		GCR_CLKCN.*psc*	f_{SYSCLK} (MHz)	GCR_MEMCKCN.*fws*
0	1	0.9	24	0	24	2
				1	12	1
				2	6	0
				3	3	0
				4	1.5	0
				5	0.75	0
				6	0.375	0
				7	0.1875	0

Core Operating Voltage Range Setting		Core Voltage Range	f_{HFIO} (MHz)	System Clock Prescaler	System Clock	Minimum Flash Wait State Setting
PWRSEQ_LP_CTRL.ovr	FLC_CN.lve	V_{CORE} (V)		GCR_CLKCN.psc	f_{SYSCLK} (MHz)	GCR_MEMCKCN.fws
1	1	1.0	48	0	48	3
				1	24	2
				2	12	1
				3	6	0
				4	3	0
				5	1.5	0
				6	0.75	0
				7	0.375	0
2	0	1.1	96	0	96	4
				1	48	2
				2	24	1
				3	12	0
				4	6	0
				5	3	0
				6	1.5	0
				7	0.75	0

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

Core Operating Voltage Range Setting		Core Voltage Range	f_{NANO} (kHz)	System Clock Prescaler	System Clock	Minimum Flash Wait State Setting
PWRSEQ_LP_CTRL.ovr	FLC_CN.lve	V_{CORE} (V)		GCR_CLKCN.psc	f_{SYSCLK} (kHz)	GCR_MEMCKCN.fws
0	1	0.9	80	0	80	0
				1	40	0
				2	20	0
1	1	1.0	80	0	80	0
				1	40	0
				2	20	0
2	0	1.1	80	0	80	0
				1	40	0
				2	20	0

Table 4-4: Minimum Flash Wait State Setting for Each OVR Setting ($f_{SYSCLK} = f_{X32K}$)

Core Operating Voltage Range Setting		Core Voltage Range	f_{X32K} (kHz)	System Clock Prescaler	System Clock	Minimum Flash Wait State Setting
PWRSEQ_LP_CTRL.ovr	FLC_CN.lve	V_{CORE} (V)		GCR_CLKCN.psc	f_{SYSCLK} (kHz)	GCR_MEMCKCN.fws
0	1	0.9	32.768	0	32.768	0
				1	16.384	0
				2	8.192	0
1	1	1.0	32.768	0	32.768	0
				1	16.384	0
				2	8.192	0

Core Operating Voltage Range Setting		Core Voltage Range V_{CORE} (V)	f_{X32K} (kHz)	System Clock Prescaler GCR_CLKCN.psc	System Clock f_{SYSCLK} (kHz)	Minimum Flash Wait State Setting GCR_MEMCKCN.fws
PWRSEQ_LP_CTRL.ovr	FLC_CN.lve					
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 the SYS_OSC by a prescaler using the *GCR_CLKCN.psc* field as shown in *Equation 4-1*.

Equation 4-1: System Clock Scaling

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

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

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

Equation 4-2: AHB Clock

$$HCLK = SYS\_CLK$$

Equation 4-3: APB Clock

$$PCLK = \frac{SYS\_CLK}{2}$$

Equation 4-4: Always-on Domain (AoD) Clock

$$AODCLK = \frac{PCLK}{4 \times 2^{GCR\_PCKDIV.aoncd}}$$

$GCR\_PCKDIV.aoncd$ is selectable from 0 to 3, yielding denominators of 4, 8, 16, or 32.

The real-time clock (RTC) uses the 32.768kHz clock/oscillator for its clock source.

All oscillators are reset to their POR reset state during a POR, system reset, or watchdog reset. Oscillator settings are not reset during a soft reset or a peripheral reset. *Table 4-5* shows each oscillator's enabled state for each type of reset source in the MAX32660. *Table 4-6* details the effect each reset source has on the system clock selection and the system clock prescaler settings.

Table 4-5: Reset Sources and Effect on Oscillator Status

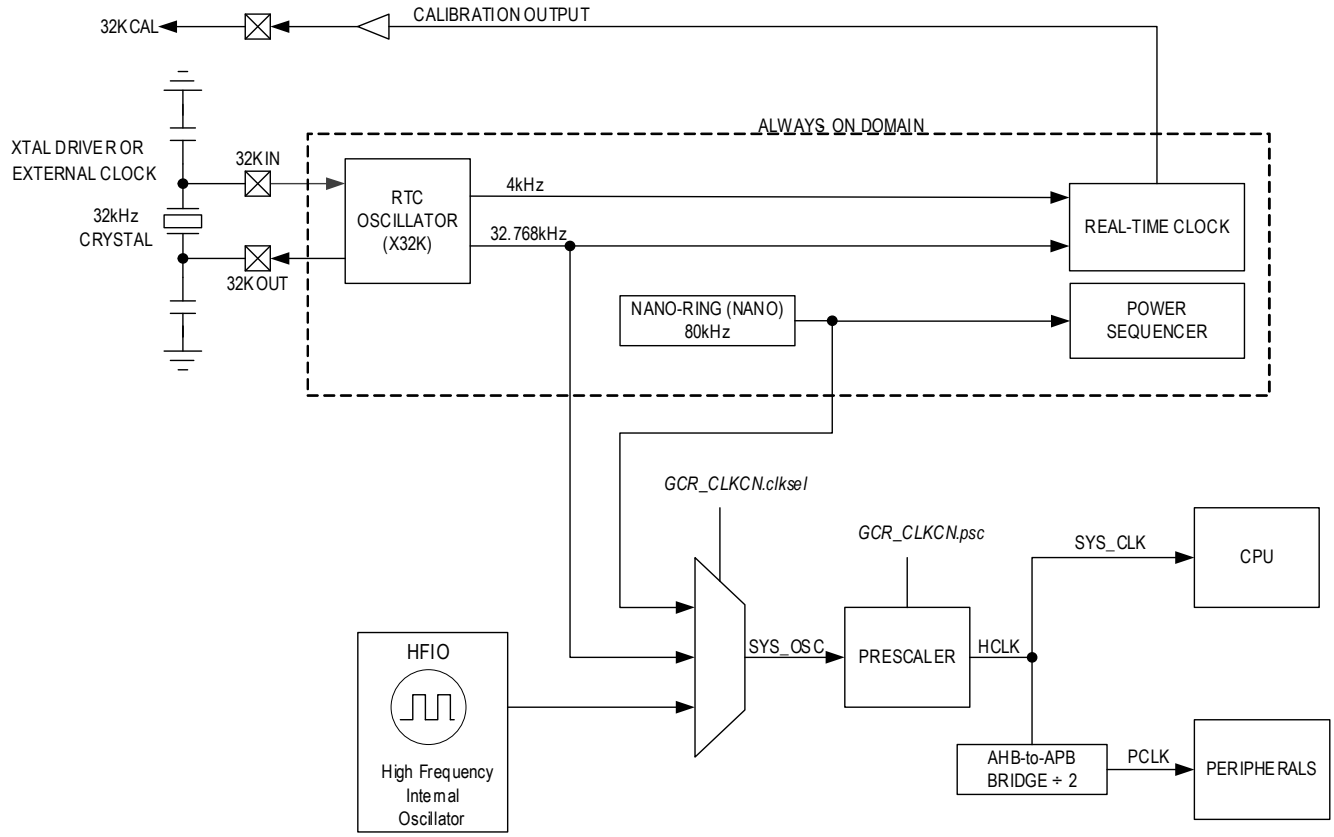
	Reset Source				
Oscillator	**POR**	**System**	**Watchdog**	**Soft**	**Peripheral**
HFIO	Enabled	Enabled	Enabled	Retains State	Retains State
NANO	Enabled	Enabled	Enabled	Enabled	Enabled
X32K	Disabled	Retains State	Retains State	Retains State	Retains State

Table 4-6: Reset Sources and Effect on System Oscillator Selection and Prescaler

	Reset Source				
Clock Field	**POR**	**System**	**Watchdog**	**Soft**	**Peripheral**
System Oscillator *GCR_CLKCN.clksel*	0 (HFIO)	0 (HFIO)	0 (HFIO)	Retains State	Retains State
System Clock Prescaler *GCR_CLKCN.psc*	1	1	1	Retains State	Retains State

*Figure 4-1*shows a high-level diagram of the MAX32660 clock tree.

Figure 4-1: Clock Tree Diagram



4.2.1 Oscillator Implementation

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

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

Any time software changes the SYS_OSC by changing the *GCR_CLKCN.clksel* field, the clock ready bit, *GCR_CLKCN.ckrdy*, is automatically cleared to 0, indicating that a SYS_OSC switchover is in progress. When the switchover is complete, *GCR_CLKCN.ckrdy* is set to 1 by hardware indicating the oscillator selected is ready for use. Immediately before entering any low-power mode, the application must enable any oscillator needed during the low-power mode.

4.2.2 High-Frequency Internal Oscillator (HFIO)

The HFIO is the default system oscillator after a POR, system reset, or watchdog reset. The HFIO operates at 96MHz and is the fastest oscillator, drawing the most power.

To enter *DEEPSLEEP*, the HFIO must be powered down by setting register bit *GCR_PM.hircpd* to 1. Failure to set this bit to 1 prevents the processor from entering *DEEPSLEEP*.

See *Table 4-5* and *Table 4-6* for details on reset sources and the effect on the HFIO oscillator.

4.2.3 32.768kHz External Crystal or Clock (X32K)

The X32K is a very low-power external oscillator that can be selected as the SYS_OSC. This oscillator can optionally use a 32.768kHz input clock instead of an external crystal. The X32K is available as an output on GPIO as an alternate function 32KCAL (P0.2, AF3).

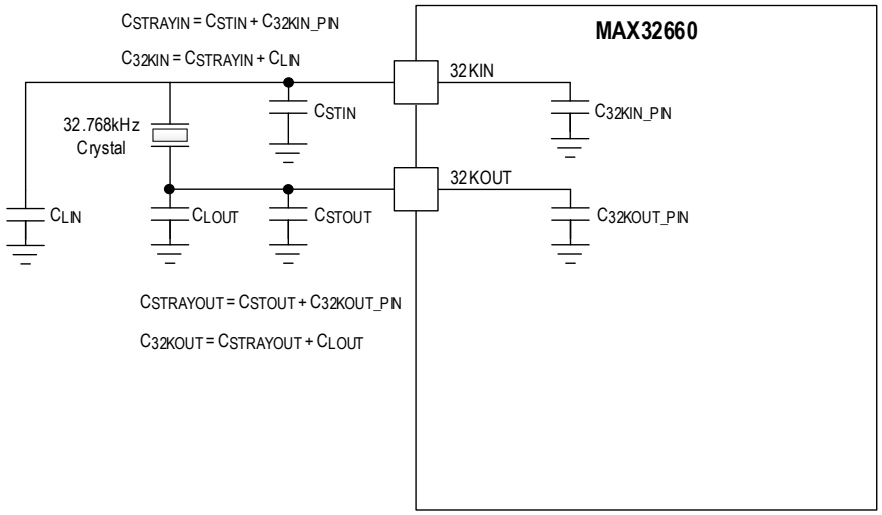
This oscillator is the dedicated clock for the RTC. If the RTC is enabled, the X32K must be enabled independent of the selection of the SYS_OSC.

When this oscillator is active, an RTC alarm can wake the device from *SLEEP* or *DEEPSLEEP* if the *GCR_PM.rtcwken* is set to 1 and the RTC alarm is configured.

A POR disables the X32K. All other forms of reset do not change the X32K enable bit. Refer to *Table 4-5* and *Table 4-6* for details on reset sources and the effect on the 32kHz oscillator.

It is important to use the correct capacitor values on the PCB when connecting the crystal. *Figure 4-2* depicts the method to determine the capacitor values C_{LIN} and C_{LOUT}.

Figure 4-2: Example 32.768kHz Crystal Capacitor Determination



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

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

Equation 4-5: Determining Load Capacitance for X32K

$$C_L = \frac{C_{32KIN} \times C_{32KOUT}}{C_{32KIN} + C_{32OUT}}$$

2. $C_{LIN} = C_{LOUT}$
3. The device pin capacitance of the 32KOUT and X32K pins are 6pF each.
4. Assume the circuit board stray capacitance represented in the diagram by C_{STIN} and C_{STOUT} are 0.5pF each.
5. Solve for C_{LOUT}
   a. $C_{LOUT} = C_{LIN} = 5.5pF$
6. Choose 6pF as the closest standard value for $C_{LOUT} = C_{LIN}$.

4.2.4 80kHz Ultra-Low Power Nano-Ring Internal Oscillator (NANO)

The NANO is a low-power internal oscillator that can be selected as SYS_OSC. This oscillator is enabled at POR and cannot be disabled by software.

4.3 Operating Modes

The device supports four operating modes, three of which are defined as low-power modes:

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

Any low-power state can wake the device to ACTIVE by a wakeup event shown in *Table 4-7*.

Table 4-7: Wakeup Sources

Low-Power Operating Mode	Wakeup Source
SLEEP	Any active peripheral interrupt or RSTN assertion
DEEPSLEEP	RTC and GPIO interrupts or RSTN assertion
BACKUP	RTC and GPIO interrupts or RSTN assertion

4.3.1 ACTIVE

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

Dynamic clocking allows the software to selectively enable or disable clocks and power to individual peripherals, providing the optimal mix of high-performance and power conservation. Internal RAM that can be enabled, disabled, or placed in low-power RAM retention mode include system RAM and the internal cache. See the section *RAM Low-Power Modes* for details on RAM power mode control.

4.3.2 Low-Power Modes

4.3.2.1 SLEEP

SLEEP is a low-power mode that suspends the CPU with a fast wakeup time to *ACTIVE*. *SLEEP* is like *ACTIVE,* except the CPU clock is disabled, which temporarily prevents the CPU from executing software. All oscillators remain active if enabled, and the AoD and RAM retention is enabled. The device returns to *ACTIVE* from any internal or external interrupt.

The Arm Cortex-M family of CPUs has two built-in low-power modes designated *SLEEP* and *DEEPSLEEP*. Implementation of these low-power modes is specific to the microcontroller's design. These modes are enabled using the System Control Register (SCR), an Arm Cortex System Control Block register.

The following pseudocode places the device in *SLEEP* mode:

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

Refer to Arm's *Cortex-M4 Technical Reference Manual* for more information on the SCR.

Table 4-9 and *Table 4-11* show *SLEEP's* effects on each clock and oscillator source.

4.3.2.2 DEEPSLEEP

This mode places the CPU in a static, low-power state. The NANO remains active in *DEEPSLEEP*. SYS_OSC is gated off, so the two primary bus clocks PCLK and HCLK, are inactive. The CPU state is retained. The X32K can be enabled using the software.

***Note**: Set GCR_PM.hircpd to 1 before entering DEEPSLEEP.*

The RTC, which has its own independent oscillator (X32K), can return the device to *ACTIVE*. The X32K remains enabled in *DEEPSLEEP* if it was enabled before entering *DEEPSLEEP*. The watchdog timers are inactive in *DEEPSLEEP*. All internal register contents and all RAM contents are preserved during *DEEPSLEEP*. The GPIO pins also retain their state while in *DEEPSLEEP*.

The Arm Cortex-M family of CPUs has two built-in low-power modes designated *SLEEP* and *DEEPSLEEP*. Implementation of these low-power modes is specific to the microcontroller's design. These modes are enabled using the System Control Register (SCR), an Arm Cortex System Control Block register.

The following pseudocode places the device in *DEEPSLEEP*:

```
GCR_PM.hircpd = 1;  // The HFIO must be powered down to enter DEEPSLEEP
SCR.sleepdeep = 1;  // DEEPSLEEP enabled
WFI (or WFE);       // Enter the low-power mode enabled by SCR.sleepdeep
```

Refer to Arm's *Cortex-M4 Technical Reference Manual* for more information on the SCR.

Table 4-9 and *Table 4-11* show *DEEPSLEEP's* effect on each clock and oscillator source.

4.3.2.3 BACKUP

This mode places the CPU in a static, low-power state. The HFIO is disabled in *BACKUP*. The NANO is enabled, and the X32K is software controlled. SYS_OSC is gated off, so PCLK and HCLK are inactive. The CPU state is not maintained.

The RTC has its own independent oscillator (X32K) and can return the device to *ACTIVE*. The X32K remains enabled in *BACKUP* if it was enabled before entering *BACKUP*.

The AoD and system RAM retention can be set to disable and clear automatically when entering this mode. Each system RAM can be independently set for data retention in *BACKUP* by setting the appropriate *PWRSEQ_LP_CTRL.ramret* bits.

Enter *BACKUP* by writing *GCR_PM.mode* to 6.

Table 4-9 and *Table 4-11* show *BACKUP's* effect on each clock and oscillator source.

4.4 Shutdown

Shutdown 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. All wakeup sources, wakeup logic, and interrupts are disabled. The device only exits the shutdown state through a POR which reinitializes the device.

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

4.5 Device Resets

Four device resets are available:

- Peripheral reset
- Soft reset
- System reset
- POR

On completion of any of the four reset cycles, all peripherals are reset. Upon 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 of the on-chip peripherals can also be reset to their POR default state using the two reset registers *GCR_RSTR0* and *GCR_RSTR1* registers.

Table 4-8, *Table 4-9*, *Table 4-10*, and *Table 4-11* show the effects on the system of the four reset types and the four power modes.

Table 4-8: Types of Resets and the Effects on Each Clock Source and the Global Control Registers

	Peripheral Reset[3]	Soft Reset[3]	System Reset[3, 4]	POR[3]
GCR	-	-	Reset	Reset
NANO	On	On	On	On
X32K	-	-	-	Off
HFIO	-	-	On[1]	On[1]
SYS_CLK	On	On	On[1, 2]	On[1, 2]

Table key:
On = Enabled by hardware (cannot be disabled).
Off = Disabled by hardware (cannot be enabled).
- = No effect.

1. A system reset occurs when exiting *BACKUP*.
2. On a system reset or POR, the HFIO is automatically selected as the SYS_OSC.
3. Peripheral, soft, and system resets are initiated by software through the *GCR_RSTR0* register.
4. System reset can also be triggered by the RSTN device pin or a watchdog reset.

Table 4-9: Low-Power Modes and the Effects on Each Clock Source and the Global Control Registers

	ACTIVE	SLEEP	DEEPSLEEP	BACKUP[1]
GCR	R	-	-	-
NANO	On	On	On	On
X32K	SW	SW	SW	SW
HFIO	R	-	Off	Off
SYS_CLK	On	On	Off	Off
CPU Clock	On	Off	Off	Off

Table key:
On = Enabled by hardware (cannot be disabled).
Off = Disabled by hardware (cannot be enabled).
- = No effect.

1. A system reset occurs when exiting *BACKUP*.

Table 4-10: Types of Resets and the Effects on Each Clock Source and the Global Control Registers

	Peripheral Reset[1]	Soft Reset[1]	System Reset[1, 2]	POR[1]
RTC	-	-	-	Reset
CPU	-	-	Reset	Reset
WDT0/WDT1	-	-	Reset	Reset
GPIO	-	Reset	Reset	Reset
Other Peripherals	Reset	Reset	Reset	Reset
AoD[3]	-	-	-	Reset
System RAM	-	-	-	Reset

Table key:
On = Enabled by hardware (cannot be disabled).
Off = Disabled by hardware (cannot be enabled).
- = No effect.

1. Peripheral, soft, and system resets are initiated by software through the *GCR_RSTR0* register.
2. System reset can also be triggered by the RSTN device pin or a watchdog reset.
3. The AoD is only reset upon power cycling V_{DD} and V_{CORE}.

Table 4-11: Low-Power Modes and the Effects on the CPU, Peripherals and Power Domains

	ACTIVE	SLEEP	DEEPSLEEP	BACKUP[1]
RTC	SW	SW	SW	SW
CPU	R	Off	Off	Off
WDT0/1	R	-	Off	Off
GPIO	R	-	-	-
Other Peripherals	R	-	Off	Off
AoD	-	-	-	-
System RAM	-	-	On	SW

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* mode; restored to system reset state when exiting *BACKUP*.

1. A system reset occurs when exiting *BACKUP*.

4.5.1 Peripheral Reset

As shown in *Table 4-8* and *Table 4-10*, 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 clock configuration, are unaffected.

To start a peripheral reset, set *GCR_RSTR0.prst* to 1. The reset is completed immediately upon setting *GCR_RSTR0.prst* to 1.

4.5.2 Soft Reset

As shown in *Table 4-8* and *Table 4-10*, a soft reset is the same as a peripheral reset, except that it also resets the GPIO to its POR state.

To perform a soft reset, set *GCR_RSTR0.srst* to 1. The reset is completed immediately upon setting *GCR_RSTR0.srst* to 1.

4.5.3 System Reset

As shown in *Table 4-8* and *Table 4-10*, a system reset is the same as a soft reset, except that it also resets all GCR, resetting the clocks to their default state. The CPU state is reset, as well as the watchdog timers. The AoD and RAM are unaffected.

A watchdog timer reset event initiates a system reset. To perform a system reset from software, set *GCR_RSTR0.system* to 1.

4.5.4 Power-On Reset

As shown in *Table 4-8* and *Table 4-10*, a POR resets everything in the device to its default state.

4.6 Instruction Cache Controller (ICC)

The ICC is used to control the internal cache of the flash memory. The ICC includes a line buffer, tag RAM, and a 16KB 2-way set associative RAM.

4.6.1 Enabling the ICC

Perform the following steps to enable the ICC:

- Set *ICC_CACHE_CTRL*.*cache_en* to 1.
- Read the *ICC_CACHE_CTRL*.*cache_rdy* field until it returns 1.

4.6.2 Disabling the ICC

Disable ICC by setting *ICC_CACHE_CTRL*.*cache_en* to 0.

4.6.3 Flushing the ICC's Cache and Tag RAM

The system control register, *GCR_SCON*, includes a field for immediately flushing the ICC's cache and tag RAM. Flush the cache and tag RAM by performing the following steps.

1. Set the *GCR_SCON*.*ccache_flush* field to 1.
   a. The hardware immediately begins a flush of the cache and tag RAM.
2. Read the *ICC_CACHE_CTRL*.*cache_rdy* field until it returns 1, indicating the ICC has completed the flush and is ready for operation.

4.6.4 Invalidating the ICC's Cache

Perform the following steps to invalidate the ICC cache, forcing hardware to flush the cache before the subsequent access.

1. Write any value to the *ICC_INVALIDATE* register.

4.6.5 Zeroizing the ICC's Cache and Tag RAM

The ICC's cache and tag RAM can be zeroized by software if desired. Set the field *GCR_MEMZCN*.*icachez* to 1 to zeroize the cache RAM and the tag RAM. Zeroizing the ICC cache automatically invalidates the cache forcing a cache load on the following access to the internal flash memory when the ICC is enabled.

4.7 ICC Registers

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

Table 4-12: ICC Registers

Offset	Register Name	Description
[0x0000]	*ICC_CACHE_ID*	Cache ID Register
[0x0004]	*ICC_MEMCFG*	Cache Memory Size Register
[0x0100]	*ICC_CACHE_CTRL*	Cache Control Register
[0x0700]	*ICC_INVALIDATE*	Cache Invalidate Register

4.7.1 ICC Register Details

Table 4-13: ICC Cache ID Register

ICC Cache ID			ICC_CACHE_ID	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	-	**Reserved**
15:10	cchid	RO	-	**ICC ID** This field returns the ICC cache ID for reference only.
9:6	partnum	RO	-	**ICC Part Number** This field returns the ICC part number indicator for reference only.
5:0	relnum	RO	-	**ICC Release Number** This field returns the ICC release number for reference only.

Table 4-14: ICC Memory Size Register

ICC Memory Size			ICC_MEMCFG	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	memsz	RO	-	**Addressable Memory Size** This field indicates the size of addressable memory by this cache controller instance in 128KB units for reference only.
15:0	cchsz	RO	-	**Cache Size** This field returns the size of the cache RAM in 1KB units for reference only. 16: 16KB Cache RAM

Table 4-15: ICC Cache Control Register

ICC Cache Control			ICC_CACHE_CTRL	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	-	**Reserved**
16	cache_rdy	RO	-	**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: Normal operation. *Note: While this field reads 0, the cache is bypassed, and reads come directly from the line buffer.*
15:1	-	R/W	-	**Reserved**
0	cache_en	R/W	0	**Enable** Set this field to 1 to enable the cache. Setting this field to 0 invalidates the cache contents. When the ICC is disabled, the cache is bypassed, and the line buffer handles the reads. 0: Disable the ICC. 1: Enable the ICC.

Table 4-16: ICC Invalidate Register

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

4.8 RAM Memory Management

This device has many features for managing internal RAMs. The internal RAM includes the system RAM instances and the peripheral FIFOs.

4.8.1 System RAM

The device includes 96KB of internal system RAM separated into four banks. *Table 4-17* lists the system RAM banks, base address, and size of each bank.

Table 4-17: System RAM Banks, Size and Base Address

System RAM Bank	Size (Words)	Base Address	Ending Address
sysram0	4K	0x2000 0000	0x2000 3FFF
sysram1	4K	0x2000 4000	0x2000 7FFF
sysram2	8K	0x2000 8000	0x2000 FFFF
sysram3	8K	0x2001 0000	0x2001 7FFF

4.8.2 RAM Zeroization

The GCR memory zeroize control register, *GCR_MEMZCN*, allows zeroizing memory by software. Zeroization writes zeros to the memory.

The following RAMs can be zeroized:

- System RAM
  - Zeroize all system RAM banks by setting the *GCR_MEMZCN.sram0z* bit to 1.
- ICC cache data and tag RAM
  - Zeroize the ICC cache and tag RAM by setting the *GCR_MEMZCN.icachez* bit to 1.

4.8.3 RAM Low-Power Modes

Each system RAM bank and the ICC cache and tag RAM can be placed in a low-power mode, referred to as *LIGHTSLEEP*, using the memory clock control register *GCR_MEMCKCN*. *LIGHTSLEEP* gates off the clock to the RAM bank and makes the RAM bank unavailable for read/write operations while the RAM bank's contents are retained, thus reducing power consumption. *LIGHTSLEEP* is available for each of the system RAM banks and the ICC cache and tag RAM.

System RAM can also be shut down for power savings using the RAM shutdown control register, *PWRSEQ_LPMEMSD*. Shutting down a RAM instance removes the power and disables the clock to the RAM. The RAM instance is inaccessible by software when in shutdown mode. Shutting down the system RAM invalidates the contents of the RAM, and the RAM is not accessible until the RAM shutdown mode is disabled. When enabling a RAM partition from a shutdown state, the RAM contents are cleared.

4.9 Global Control Registers (GCR)

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

Note: The GCR registers are reset on a system reset, a watchdog timer reset, and a POR. Soft reset and peripheral reset do not affect these registers.

Table 4-18: Global Control Registers

Offset	Register Name	Description
[0x0000]	*GCR_SCON*	System Control Register
[0x0004]	*GCR_RSTR0*	Reset 0 Register
[0x0008]	*GCR_CLKCN*	Clock Control Register
[0x000C]	*GCR_PM*	Power Management Register
[0x0018]	*GCR_PCKDIV*	Peripheral Clock Divider Register
[0x0024]	*GCR_PERCKCN0*	Peripheral Clocks Disable 0 Register
[0x0028]	*GCR_MEMCKCN*	Memory Clock Control Register
[0x002C]	*GCR_MEMZCN*	Memory Zeroize Register
[0x0034]	*GCR_SCCK*	SCCK Register
[0x0038]	*GCR_MPRI0*	MPRI0 Register
[0x003C]	*GCR_MPRI1*	MPRI1 Register
[0x0040]	*GCR_SYSST*	System Status Flags Register
[0x0044]	*GCR_RSTR1*	Reset Register 1 Register
[0x0048]	*GCR_PERCKCN1*	Peripheral Clocks Disable 1 Register
[0x004C]	*GCR_EVTEN*	Event Enable Register
[0x0050]	*GCR_REVISION*	Revision Register
[0x0054]	*GCR_SYSSIE*	System Status Interrupt Enable Register

4.9.1 Global Control Register Details

Table 4-19: System Control Register

System Control			GCR_SCON	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:15	-	RO	-	**Reserved**
14	swd_dis	R/W	0	**Serial Wire Debug (SWD) Disable** This bit is used to disable the serial wire debug interface. 0: SWD enabled. 1: SWD disabled. *Note: This bit is only writable if the FMV lock word is not programmed.*
13:7	-	RO	-	**Reserved**
6	ccache_flush	R/W1O	0	**ICC Flush** Write 1 to flush the cache RAM, tag RAM, and line buffer. This bit is automatically cleared by hardware to 0 when the flush is complete. Clearing this field to 0 does not halt the flush operation if it is in progress. 0: Normal operation. 1: Cache flush in progress.
5	fpu_dis	R/W	0	**FPU Disable** Set this field to 1 to disable the Cortex-M4 FPU. 0: Enabled 1: Disabled
4	flash_page_flip	RO	0	**Reserved**
3:2	-	RO	0	**Reserved**
1	sbusarb	RO	1	**Reserved**
0	-	RO	0	**Reserved**

Table 4-20: Reset 0 Register

Reset 0			GCR_RSTR0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31	system	R/W1O	0	**System Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See the *Device Resets* section for additional information.
30	prst	R/W1O	0	**Peripheral Reset** *Write 1 to reset. This field is cleared by hardware when the reset is complete. See the* Device Resets *section for additional information.*
29	srst	R/W1O	0	**Soft Reset** *Write 1 to reset. This field is cleared by hardware when the reset is complete. See the* Device Resets *section for additional information.*
28:18	-	RO	-	**Reserved**
17	rtc	R/W1O	0	**RTC Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
16	i2c0	R/W1O	0	**I2C0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
15	-	RO	0	**Reserved**
14	spi1	R/W1O	0	**SPIMSS (SPI1/I²S) Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
13	spi0	R/W1O	0	**SPI17Y (SPI0) Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
12	uart1	R/W1O	0	**UART1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
11	uart0	R/W1O	0	**UART0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
10:8	-	RO	0	**Reserved**
7	timer2	R/W1O	0	**TMR2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
6	timer1	R/W1O	0	**TMR1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
5	timer0	R/W1O	0	**TMR0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
4:3	-	RO	0	**Reserved**
2	gpio0	R/W1O	0	**GPIO0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
1	wdt	R/W1O	0	**WDT0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
0	dma	R/W1O	0	**DMA Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.

Table 4-21: System Clock Control Register

System Clock Control			GCR_CLKCN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:30	-	RO	3	**Reserved**
29	lirc8k_rdy	R	0	**80kHz NANO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
28:27	-	RO	0	**Reserved**
26	hirc_rdy	R	0	**HFIO Ready** 0: Oscillator not ready or not enabled. 1: Oscillator ready.
25	x32k_rdy	R	0	**X32K Ready Status** 0: Oscillator not ready or not enabled. 1: Oscillator ready.
24:19	-	RO	-	**Reserved**
18	hirc_en	R/W	1	**HFIO Enable** 0: Disabled. 1: Enabled and ready when *GCR_CLKCN.hirc_rdy* reads 1.
17	x32k_en	R/W	0	**X32K External Oscillator Enable** 0: Disabled. 1: Enabled and ready when *GCR_CLKCN.hirc_rdy* reads 1.
16:14	-	RO	-	**Reserved**
13	ckrdy	R/W	0	**SYS_OSC Select Ready** When the SYS_OSC is changed by modifying *GCR_CLKCN.clksel*, there is a delay until the switchover is complete. This bit is cleared to 0 until the switchover is complete. 0: Switch to new clock source in progress. 1: SYS_OSC is the clock source selected with the *GCR_CLKCN.clksel* field.
12	-	RO	-	**Reserved**
11:9	clksel	R/W	0	**SYS_OSC Select** Selects the SYS_OSC source used to generate SYS_CLK. Modifying this field clears the *GCR_CLKCN.ckrdy* field to 0 immediately. 0: HFIO 1: Reserved 2: Reserved 3: NANO 4: Reserved 5: Reserved 6: X32K 7: Reserved
8:6	psc	R/W	0	**SYS_OSC Prescaler** Sets the divider for generating SYS_CLK from the selected SYS_OSC. See *Equation 4-1* for details.
5:0	-	RO	8	**Reserved**

Table 4-22: Power Management Register

Power Management			GCR_PM	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**

Power Management			GCR_PM	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
15	hircpd	R/W	0	**HFIO *DEEPSLEEP* Auto Off** When set to 1, the HFIO is automatically powered off when in *DEEPSLEEP*. If the HFIO is not enabled (*GCR_CLKCN.hirc_en* = 1) when entering *DEEPSLEEP*, this field's value is ignored. 0: HFIO active, powered on, during *DEEPSLEEP* if the HFIO is enabled (*GCR_CLKCN.hirc_en* = 1). 1: HFIO is powered off in *DEEPSLEEP*. *Note: Set this field to 1 before entering DEEPSLEEP mode to achieve the lowest power numbers for the device if the HFIO is enabled (GCR_CLKCN.hirc_en = 1).*
14:6	-	R/W	0	**Reserved**
5	rtcwken	R/W	0	**RTC Alarm Wakeup Enable** Set this field to 1 to enable an RTC alarm to wake the device from low-power modes. The RTC alarm wakes the device from *SLEEP*, *DEEPSLEEP*, or *BACKUP*. 0: Disabled 1: Enabled
4	gpiowken	R/W	0	**GPIO Wakeup Enable** Set this field to 1 to enable all GPIO pins as potential wakeup sources. Any GPIO configured for wakeup can wake the device from *SLEEP*, *DEEPSLEEP*, or *BACKUP*. 0: Disabled 1: Enabled
3	-	RO	0	**Reserved**
2:0	mode	R/W	0	**Operating Mode** Configures the current operating mode for the device. 0b000: *ACTIVE* 0b001: *ACTIVE* 0b010: *ACTIVE* 0b011: *SHUTDOWN* 0b100: *BACKUP* 0b101: *BACKUP* 0b110: *BACKUP* 0b111: *SHUTDOWN*

Table 4-23: Peripheral Clock Divisor Register

Peripheral Clock Divisor			GCR_PCKDIV	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1:0	aoncd	R/W	0	**AoD Clock Divider** This field configures the frequency of the AoD clock. See the section *Oscillator Sources and Clock Switching* for details.

Table 4-24: Peripheral Clock Disable 0 Register

Peripheral Clocks Disable 0			GCR_PERCKCN0	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:29	-	RO	0	**Reserved**

Peripheral Clocks Disable 0			GCR_PERCKCN0	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
28	i2c1d	R/W	0	**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.
27:18	-	RO	0	**Reserved**
17	t2d	R/W	0	**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	t1d	R/W	0	**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	t0d	R/W	0	**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	-	RO	0	**Reserved**
13	i2c0d	R/W	0	**I2C0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
12:11	-	RO	-	**Reserved**
10	uart1d	R/W	0	**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	uart0d	R/W	0	**UART0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
8	-	RO	0	**Reserved**
7	spi1d	R/W	0	**SPIMSS (SPI1/I²S) 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	spi0d	R/W	0	**SPI17Y (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

Peripheral Clocks Disable 0			GCR_PERCKCN0	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
5	dmad	R/W	0	**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
4:1	-	RO	0	**Reserved**
0	gpio0d	R/W	0	**GPIO0 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 4-25: Memory Clock Control Register

Memory Clock Control			GCR_MEMCKCN	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:13	-	RO	0	**Reserved**
12	icachels	R/W	0	**ICC RAM *LIGHTSLEEP* Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* but is retained. 0: Normal operation. 1: *LIGHTSLEEP* mode enabled.
11	sysram3ls	R/W	0	**System RAM 3 *LIGHTSLEEP* Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* but is retained. 0: Normal operation. 1: *LIGHTSLEEP* mode enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 4-17 for base address and size information.*
10	sysram2ls	R/W	0	**System RAM 2 *LIGHTSLEEP* Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* but is retained. 0: Normal operation. 1: *LIGHTSLEEP* mode enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 4-17 for base address and size information.*
9	sysram1ls	R/W	0	**System RAM 1 *LIGHTSLEEP* Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* but is retained. 0: Normal operation. 1: *LIGHTSLEEP* mode enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 4-17 for base address and size information.*

Memory Clock Control			GCR_MEMCKCN	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
8	sysram0ls	R/W	0	**System RAM 0 *LIGHTSLEEP* Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* but is retained. 0: Normal operation. 1: *LIGHTSLEEP* mode enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 4-17 for base address and size information.*
7:3	-	RO	0	**Reserved**
2:0	fws	R/W	5	**Program Flash Wait States** This field sets the number of wait-state SYS_OSC cycles per flash code read access. See the section *Flash Wait States* for additional information. For the HFIO, the minimum wait states should be 2. For all other clock sources, the minimum wait state is 1. 0 – 7: Number of flash code access wait states.

Table 4-26: Memory Zeroization Control Register

Memory Zeroization Control			GCR_MEMZCN	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	icachez	R/W1O	0	**Cache Data and Tag RAM Zeroization** Write 1 to initiate the operation. 0: Normal operation 1: Zeroize memory
0	sram0z	R/W1O	0	**System Data RAM Zeroization** Write 1 to initiate the operation. 0: Normal operation 1: Zeroize memory

Table 4-27: SCCK Register

SCCK			GCR_SCCK	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 4-28: MPRI0 Register

MPRI0			GCR_MPRI0	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 4-29: MPRI1 Register

MPRI1			GCR_MPRI1	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	RO	0	**Reserved**

Table 4-30: System Status Flag Register

System Status Flag			GCR_SYSST	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**

System Status Flag			GCR_SYSST	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
5	scmemf	RO	0	**Reserved**
4:2	-	RO	0	**Reserved**
1	codeinterr	RO	0	**Reserved**
0	iceclock	R	0	**ICE Lock Status Flag** This field is set in the factory and, if set to 1, disables SWD access to the device. 0: Arm ICE is enabled (unlocked). 1: Arm ICE is locked (disabled).

Table 4-31: Reset Register 1

Reset 1			GCR_RSTR1	[0x0044]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	i2c1	R/W	0	**I2C1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.

Table 4-32: Peripheral Clock Disable Register 1

Peripheral Clock Disable 1			GCR_PERCKCN1	[0x0048]
Bits	**Name**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11	icached	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
3	flcd	R/W	0	**Flash Controller Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled 1: Disabled *Note: Disabling the clock to the flash controller prevents code fetches from the flash. Care must be taken to ensure software execution is performed from RAM while the flash controller peripheral clock is disabled.*
2:0	-	RO	0	**Reserved**

Table 4-33: Event Enable Register

Event Enable			GCR_EVTEN	[0x004C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	-	**Reserved**
1	rxevent	R/W	0	**RX Event Enabled** Set this field to 1 to enable generation of a receive event (RXEV) on GPIO0[8] (logic high) to wake the CPU from a WFE sleep state. 0: Disabled 1: Enabled

Event Enable			GCR_EVTEN	[0x004C]
Bits	**Name**	**Access**	**Reset**	**Description**
0	dmaevent	R/W	0	**CPU DMA CTZ Event Wakeup Enable** Allows a DMA CTZ event to generate an RXEV to wake the CPU from a WFE sleep state. 0: Disabled 1: Enabled

Table 4-34: Revision Register

Revision			GCR_REVISION	[0x0050]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	revision	R	*	**Maxim Integrated Chip Revision** This field reads the chip revision ID as a packed binary coded decimal. 0x000000A1: A1 revision 0x000000A2: A2 revision

Table 4-35: System Status Interrupt Enable Register

System Status Interrupt Enable			GCR_SYSSIE	[0x0054]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
5	scmfie	RO	0	**Reserved**
4:2	-	RO	0	**Reserved**
1	cieie	RO	0	**Reserved**
0	iceulie	R/W	0	**Arm ICE Unlocked Interrupt Enable** Generates an interrupt if the *GCR_SYSST.icelock* is set. 0: Disabled 1: Enabled

4.10 System Initialization Registers (SIR)

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

Note: The SIR registers are reset on a system reset, a watchdog timer reset, and a POR. Soft reset and peripheral reset do not affect these registers.

Table 4-36: System Initialization Registers

Offset	Register Name	Description
[0x0000]	*SIR_SISTAT*	System Initialization Status Register
[0x0004]	*SIR_ERRADDR*	System Initialization Address Error Register
[0x0100]	*SIR_FSTAT*	System Initialization Function Status Register
[0x0104]	*SIR_SFSTAT*	System Initialization Special Function Status Register

4.10.1 System Initialization Register Details

Table 4-37: Function Control Register 0

System Initialization Status			SIR_SISTAT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	-	**Reserved**

System Initialization Status			SIR_SISTAT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
1	crcerr	RO	See Description	**Configuration Error Flag** This field is set by hardware during reset if an error in the device configuration is detected. 0: Valid. 1: 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 the SIR_ERRADDR.addr field.*
0	magic	RO	See Description	**Configuration Valid Flag** This field is set to 1 by hardware during reset if the device configuration is valid. 0: Invalid. 1: Valid. *Note: If this field reads 0, the device configuration is invalid, and a device error has occurred. Contact Maxim Integrated technical support for additional assistance.*

Table 4-38: System Initialization Address Error Register

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

Table 4-39: System Initialization Function Status Register

System Initialization Function Status			SIR_FSTAT	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	addr	RO	0	**Configuration Error Address** If the *SIR_SISTAT.crcerr* field is set to 1, this register's value is the address of the configuration failure.
0	fpu	RO	1	**Floating Point Unit** 0: No floating point unit 1: Floating point unit

Table 4-40: System Initialization Function Status Register

System Initialization Special Function Status			SIR_SFSTAT	[0x0104]
Bits	**Name**	**Access**	**Reset**	**Description**
31:6	addr	RO	0	**Configuration Error Address** If the *SIR_SISTAT.crcerr* field is set to 1, this register's value is the address of the configuration failure.
5	maa	RO	*	**MAA** 0: No MAA peripheral 1: MAA peripheral **: Refer to the device data sheet for ordering information.*
4	sha	RO	*	**SHA** 0: No SHA peripheral 1: SHA peripheral **: Refer to device data sheet for ordering information.*

System Initialization Special Function Status			SIR_SFSTAT	[0x0104]
Bits	**Name**	**Access**	**Reset**	**Description**
3	aes	RO	*	**AES** 0: No AES peripheral 1: AES peripheral ** Refer to device data sheet for ordering information.*
2	trng	RO	*	**TRNG** 0: No TRNG 1: TRNG peripheral included **: Refer to device data sheet for ordering information.*
1:0	-	RO	0	**Reserved**

4.11 Power Sequencer and Always-On-Domain Registers (PWRSEQ)

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

Note: The PWRSEQ registers are reset on a POR. System reset, watchdog timer reset, soft reset, and peripheral reset do not affect these registers.

Table 4-41: Power Sequencer Low-Power Control Registers

Offset	Register Name	Description
[0x0000]	*PWRSEQ_LP_CTRL*	Low-Power Control Register
[0x0004]	*PWRSEQ_LP_WAKEFL*	GPIO0 Low-Power Wakeup Status Flags Register
[0x0008]	*PWRSEQ_LPWK_EN*	GPIO0 Low-Power Wakeup Enable Register
[0x0040]	*PWRSEQ_LPMEMSD*	RAM Shutdown Control Register

4.11.1 Power Sequencer Register Details

Table 4-42: Low-Power Voltage Control Register

Low-Power Voltage Control			PWRSEQ_LP_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	-	R/W	0	**Reserved**
25	vddio_por_dis	DNM	0	**V_{DD} (V_{DDIO}) POR Monitor Disable** Reserved. Do not modify.
24:22	-	R/W	0	**Reserved**
21	-	R/W	0	**Reserved**
20	vcore_svm_dis	DNM	0	**V_{CORE} Supply Voltage Monitor Disable** Reserved. Do Not Modify.
19:17	-	R/W	0	**Reserved**
16	ldo_dis	R/W	See Description	**LDO Disable** This field initializes to 1 (LDO disabled) 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 automatically set to 0 (LDO enabled) 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	-	R/W	0	**Reserved**

Low-Power Voltage Control			PWRSEQ_LP_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
12	vcore_por_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* and *BACKUP* operation. Disconnecting the POR signal from the core during *DEEPSLEEP* and *BACKUP* prevents the core from detecting a POR event while the device is in *DEEPSLEEP* or *BACKUP*. 0: POR signal is connected to the core during *DEEPSLEEP* and *BACKUP*. 1: POR signal is not connected to the core during *DEEPSLEEP* and *BACKUP*.
11	bg_off	R/W	1	**Bandgap Disable for *DEEPSLEEP* and *BACKUP*** Setting this field to 1 disables the bandgap during *DEEPSLEEP* and *BACKUP*. 0: Enabled 1: Disabled
10	fast_wk_en	R/W	0	**Fast Wakeup Enable for *DEEPSLEEP*** Always set to 1 to enable fast wakeup from *DEEPSLEEP*. When enabled, the system exits *DEEPSLEEP* faster by: • Bypassing the NANO warmup. • Reducing the warmup time for the HFIO. • Reducing the warmup time for the LDO. 0: Disabled 1: Enabled
9	-	RO	0	**Reserved**
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*. 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 system RAM. 1: RAM retention regulator enabled. *Note: Use the PWRSEQ_LP_CTRL.ramret_sel[3:0] fields to determine which system RAM banks have data retention enabled.*
7	-	R/W	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 wakeup time. 0: Enabled 1: Disabled

Low-Power Voltage Control			PWRSEQ_LP_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
5:4	ovr	R/W	0b10	**Output Voltage Range for Internal Regulator** Set this field to control the internal regulator's output voltage, allowing selection of the internal core operating voltage and the frequency of the HFIO. On a POR, this field defaults to 1.1V output ±10% with the f_{HFIO} = 96MHz. *Note: If V_{CORE} is connected to an external supply voltage, this field should be modified only to set it to match the input voltage on V_{CORE}. The external supply must be equal to or greater than this field setting indication.* **Dual Supply Operation:** 0b11: Reserved 0b10: V_{CORE} = 1.1V, $f_{SYS\_CLK(MAX)}$ = 96MHz. 0b01: V_{CORE} = 1.0V, $f_{SYS\_CLK(MAX)}$ = 48MHz. 0b00: V_{CORE} = 0.9V, $f_{SYS\_CLK(MAX)}$ = 24MHz. **Single Supply Operation (V_{CORE} = GND)** 0b11: Reserved 0b10: V_{LDO} = 1.1V, $f_{SYS\_CLK(MAX)}$ = 96MHz 0b01: V_{LDO} = 1.0V, $f_{SYS\_CLK(MAX)}$ = 48MHz 0b00: V_{LDO} = 0.9V, $f_{SYS\_CLK(MAX)}$ = 24MHz
3	ramret_sel3	R/W	0	**System RAM 3 Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram3*. See *Table 4-17* for system RAM configuration. 0: Data retention for *sysram3* address space disabled in *BACKUP*. 1: Data retention for *sysram3* address space enabled in *BACKUP*.
2	ramret_sel2	R/W	0	**System RAM 2 Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram2*. See *Table 4-17* for system RAM configuration. 0: Data retention for *sysram2* address space disabled in *BACKUP*. 1: Data retention for *sysram2* address space enabled in *BACKUP*.
1	ramret_sel1	R/W	0	**System RAM 1 Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram1*. See *Table 4-17* for system RAM configuration. 0: Data retention for *sysram1* address space disabled in *BACKUP*. 1: Data retention for *sysram1* address space enabled in *BACKUP*.
0	ramret_sel0	R/W	0	**System RAM 0 Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram0*. See *Table 4-17* for system RAM configuration. 0: Data retention for *sysram0* address space disabled in *BACKUP*. 1: Data retention for *sysram0* address space enabled in *BACKUP*.

Table 4-43: Low-Power Mode Wakeup Flags for GPIO0

Low-Power Mode GPIO Wakeup Flags			PWRSEQ_LP_WAKEFL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:14	-	RO	0	**Reserved**
13:0	wakest	R/W1C	0	**GPIO Pin Wakeup Status Flag** When a GPIO pin, in any power mode, transitions from low-to-high or high-to-low, the corresponding bit in this field is set. The device transitions from a low-power mode to *ACTIVE* mode if the corresponding interrupt enable bit is set in the *PWRSEQ_LPWK_EN* register and the *GCR_PM.gpiowken* bit is set to 1.

Table 4-44: Low-Power Wakeup Enable for GPIO0 Register

Low-Power Mode Wakeup Enable for GPIO0			PWRSEQ_LPWK_EN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:14	-	R/W	0	**Reserved**
13:0	wakeen	R/W	0	**GPIO0 Pin Wakeup Interrupt Enable** Setting a bit in this field causes an interrupt to be generated and wakes up the device from any low-power mode to *ACTIVE* when the corresponding bit in the *PWRSEQ_LP_WAKEFL* register is set. A wakeup event generated by GPIO generates a GPIOWAKE_IRQn interrupt. *Note: To enable the device to wake up from a low-power mode on a GPIO pin transition, first set the global GPIO wakeup enable field to 1 (GCR_PM.gpiowken = 1).*

Table 4-45: RAM Shut Down Register

RAM Shut Down			PWRSEQ_LPMEMSD	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	RO	-	**Reserved**
3	sram3_off	R/W	0	***Sysram3* Shut Down** Write 1 to shut down power to *sysram3* memory range. 0: Enabled. 1: Disabled. Affected memory is destroyed.
2	sram2_off	R/W	0	***Sysram2* Shut Down** Write 1 to shut down power to *sysram2* memory range. 0: Enabled. 1: Disabled. Affected memory is destroyed.
1	sram1_off	R/W	0	***Sysram1* Shut Down** Write 1 to shut down power to *sysram1* memory range. 0: Enabled. 1: Disabled. Affected memory is destroyed.
0	sram0_off	R/W	0	***Sysram0* Shut Down** Write 1 to shut down power to *sysram0* memory range. 0: Enabled. 1: Disabled. Affected memory is destroyed.

5. Interrupts and Exceptions

The Arm Cortex-M4 with FPU manages interrupts and exceptions using the nested vector interrupt controller (NVIC). The NVIC manages the interrupts, exceptions, priorities, and masking. *Table 5-1* details the MAX32660 interrupt vector table and describes each exception and interrupt.

5.1 Features

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

5.2 Interrupt Vector Table

Table 5-1 lists the interrupt and exception table for the MAX32660. There are 55 interrupt entries for the MAX32660, including reserved interrupt placeholders. Including the 15 system exceptions for the Arm Cortex-M4 with FPU, the total number of entries is 70. The interrupt vector table alignment must be on a 128-word boundary, the next power of 2 greater than the 70-total interrupt vector table entries.

Table 5-1: MAX32660 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	Watchdog Timer 0 Interrupt
18	[0x0048]	-	Reserved
19	[0x004C]	RTC_IRQn	Real-Time Clock Interrupt
20	[0x0050]	-	Reserved
21	[0x0054]	TMR0_IRQn	Timer 0 Interrupt
22	[0x0058]	TMR1_IRQn	Timer 1 Interrupt
23	[0x005C]	TMR2_IRQn	Timer 2 Interrupt
24:28	[0x0060]:[0x0070]	-	Reserved
29	[0x0074]	I2C0_IRQn	I^2C Port 0 Interrupt
30	[0x0078]	UART0_IRQn	UART Port 0 Interrupt
31	[0x007C]	UART1_IRQn	UART Port 1 Interrupt
32	[0x0080]	SPI0_IRQn	SPI Port 0 Interrupt

Exception (Interrupt) Number	Offset	Name	Description
33	[0x0084]	SPIMSS_IRQn	SPIMSS (SPI1/I²S) Interrupt
34:38	[0x0088]:[0x098]	-	Reserved
39	[0x009C]	FLC_IRQn	Flash Controller Interrupt
40	[0x00A0]	GPIO0_IRQn	GPIO Port 0 Interrupt
41:43	[0x00A4]:[0x00AC]	-	Reserved
44	[0x00B0]	DMA0_IRQn	DMA Channel 0 Interrupt
45	[0x00B4]	DMA1_IRQn	DMA Channel 1 Interrupt
46	[0x00B8]	DMA2_IRQn	DMA Channel 2 Interrupt
47	[0x00BC]	DMA3_IRQn	DMA Channel 3 Interrupt
48:51	[0x00C0]:[0x00CC]	-	Reserved
52	[0x00D0]	I2C1_IRQn	I²C Port 1 Interrupt
53:69	[0x00D4]:[0x0114]	-	Reserved
70	[0x0118]	GPIOWAKE_IRQn	GPIO Wakeup Interrupt

6. Debug Access Port (DAP)

Some device versions might provide an Arm debug access port (DAP) which supports debugging during application development. Refer to the device data sheet's ordering information to determine if a specific part number supports a customer-accessible DAP. Parts with a customer-accessible DAP should only be used for development and never used in a final customer product.

GCR_SYSST.icelock = 0 if the device provides a customer-accessible DAP.

6.1 Instances

The DAP interface communicates through the serial wire debug (SWD) and/or JTAG interface signals shown in *Table 6-1*.

Note: The SWDIO and SWDCLK can be mapped to two different port pin options, but only one instance of the SWD.

Table 6-1: MAX32660 DAP Instance

Instance	Pin	Alternate Function	SWD Signal	JTAG Signal	Pullup
DAP	P0.0	AF1	SWDIO	N/A	100kΩ to V_{DD}
	P0.1	AF1	SWDCLK	N/A	
	P0.8	AF2	SWDIO	N/A	100kΩ to V_{DD}
	P0.9	AF2	SWDCLK	N/A	

6.2 Access Control

6.2.1 Factory-Disabled DAP

Device versions that do not provide a DAP interface have the *GCR_SYSST.icelock* field set to 1 set at the factory, permanently disabling the DAP interface. No software action is needed to secure these devices. Contact Maxim Integrated for information on ordering information for devices with the DAP disabled.

6.2.2 Software-Accessible DAP

Device versions that provide a DAP (*GCR_SYSST.icelock* = 0) always have their interface(s) enabled and running unless the software explicitly sets the *GCR_SCON.sw_dis* field to 1. The read-only field *GCR_SYSST.icelock* is cleared to 0, and the software has read and write access to the *GCR_SCON.sw_dis* field. The *GCR_SCON.sw_dis* field resets to 0 after every POR to allow access to the DAP during development.

The software can disable the DAP by setting the *GCR_SCON.sw_dis* field to 1. The only practical application for disabling the DAP is to release the interface pins to operate as standard GPIO or in one of the supported alternate function modes in a development environment. Customers can use device versions with the DAP enabled for development but should only use device versions with the factory-disabled DAP in a final product.

6.3 Pin Configuration

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

By default, the pin associated with the bidirectional SWDIO/TMS signal is configured as a GPIO high-impedance input after any POR. While the DAP is in use, a pullup resistor should be connected to the SWDIO/TMS pin, as shown in *Table 6-1*. The pullup ensures the signal is in a known state when control of the SWDIO/TMS pin is transferred between the host and target. The pullup resistor should be removed if the associated pin is used as a GPIO to avoid unnecessary current consumption.

7. Flash Controller (FLC)

The MAX32660's flash controller manages read, write, and erase accesses to the internal flash. It provides the following features:

- Up to 256KB total internal flash memory
- 32 pages
- 8,192 bytes per page
- 2,048 words by 128 bits per page
- 128-bit data reads
- Page erase and mass erase support
- Write Protection

7.1 Instances

The device provides one instance of the FLC.

The 256KB of internal flash memory is programmable through the JTAG debug interface (in-system) or directly with user software (in-application).

The flash is organized as an array of 2,048 words by 128 bits, or 8,192 bytes per page. *Table 7-1* shows the start address and end address for the internal flash memory.

Table 7-1: Internal Flash Memory Organization

Page Number	Size in Bytes	Start Address	End Address
1	8,192	0x0000 0000	0x0000 1FFF
2	8,192	0x0000 2000	0x0000 3FFF
3	8,192	0x0000 4000	0x0000 5FFF
4	8,192	0x0000 6000	0x0000 7FFF
5	8,192	0x0000 8000	0x0000 9FFF
…	…	…	…
31	8,192	0x0003 C000	0x0003 DFFF
32	8,192	0x0003 E000	0x0003 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. All flash operations work at all core operating voltage range selections as long as the configuration outlined in section *Core Operating Voltage Range Selection* is followed.

7.2.1 Clock Configuration

The FLC requires a 1MHz APB PCLK for programming operations. See the section *Oscillator Sources and Clock Switching* for details on peripheral clock configuration. Use the FLC clock divisor to generate $f_{FLC\_CLK} = 1MHz$, as shown in *Equation 7-1*. The HFIO must be used as the system clock for flash programming and erase operation. For the HFIO as the system clock (SYS_CLK = HFIO), the *FLC_CLKDIV.clkdiv* field should be set to 48 (0x30).

Equation 7-1: FLC Clock Frequency

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

7.2.2 Lock Protection

A locking mechanism prevents accidental memory writes and erases. All write and erase operations require the *FLC_CN.unlock* field to be set to 0x2 before starting the operation. Once unlocked, the flash remains unlocked until a value other than 0x2 is written to the *FLC_CN.unlock* field. Writing any other value to the *FLC_CN.unlock* field results in:

1. The flash instance remaining locked,

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

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

7.2.3 Flash Write Width

The flash controller supports write widths of either 32-bits or 128-bits. Selection of the flash write width is controlled with the *FLC_CN.wdth* field and defaults to 128-bit width on all forms of reset. Setting *FLC_CN.wdth* to 1 selects 32-bit write widths.

In 128-bit width mode, the target address bits *FLC_ADDR*[3:0] are ignored, resulting in 128-bit alignment. In 32-bit width mode, the target address bits *FLC_ADDR*[1:0] are ignored for 32-bit address alignment. If the desired target address is not 128-bit aligned (*FLC_ADDR*[3:2] ≠ 0), 32-bit width mode is required.

Table 7-2: Valid Addresses for 32-bit and 128-bit Internal Flash Writes

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

7.2.4 Flash Write

Perform the following steps to write to the internal flash memory:

1. If desired, enable flash controller interrupts by setting the *FLC_INTR.afie* and *FLC_INTR.doneie* bits.
2. Read the *FLC_CN.pend* bit until it returns 0.
3. Configure the *FLC_CN.clkdiv* to match the SYS_CLK frequency.
4. Set the write field, *FLC_CN.wdth*, as described in *Flash Write Width*.
5. Set the *FLC_ADDR* register to a valid target address. Reference *Table 7-2*.
6. Set the data register or registers.
   a. For 32-bit write width, set *FLC_DATA* to the data to write.
   b. For 128-bit write width, set *FLC_DATA3*, *FLC_DATA2*, *FLC_DATA1*, and *FLC_DATA* to the data to write. *FLC_DATA3* is the most significant word, and *FLC_DATA* is the least significant word.
7. Set *FLC_CN.unlock* to 0x2 to unlock the internal flash.
8. Read the *FLC_CN.pend* bit until it returns 0.
9. Set *FLC_CN.wr* to 1. This field is automatically cleared by the FLC when the write operation is finished.
10. *FLC_INTR.done* is set by hardware when the write completes, and if an error occurred, the *FLC_INTR.af* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.
11. Set *FLC_CN.unlock* to any value other than 0x2 to re-lock the flash instance.

7.2.5 Page Erase

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

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

1. If desired, enable flash controller interrupts by setting the *FLC_INTR.afie* and *FLC_INTR.doneie* bits.
2. Read the *FLC_CN.pend* bit until it returns 0.
3. Configure the *FLC_CN.clkdiv* to match the SYS_CLK frequency.
4. Set the *FLC_ADDR* register to an address within the page to be erased. *FLC_ADDR*[12:0] is ignored by the FLC to ensure the address is page aligned.
5. Set *FLC_CN.unlock* to 0x2 to unlock the internal flash.
6. Read the *FLC_CN.pend* bit until it returns 0.
7. Set *FLC_CN.erase_code* to 0x55 for page erase.
8. Set *FLC_CN.pge* to 1 to start the page erase operation.
9. The *FLC_CN.pend* bit is set by the FLC while the page erase is in progress, and the *FLC_CN.pge* and *FLC_CN.pend* fields are cleared when the page erase completes.
10. *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 IRQ if the interrupt enable bits are set.
11. Set *FLC_CN.unlock* to any value other than 0x2 to re-lock the flash instance.

7.2.6 Mass Erase

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

Mass erase clears the internal flash memory on an instance base. A mass erase operation clears the entire flash memory instance. Code to perform a mass erase should be executed from RAM. The handling of reprogramming the flash memory must be performed from RAM, or the device is erased with no software loaded in the flash after the mass erase operation is completed. Perform the following steps to mass erase the internal flash:

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

7.3 Flash Controller Registers

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

Table 7-3: Flash Controller Registers

Offset	Register Name	Description
[0x0000]	*FLC_ADDR*	Flash Address Pointer Register
[0x0004]	*FLC_CLKDIV*	Flash Clock Divisor Register

Offset	Register Name	Description
[0x0008]	*FLC_CN*	Flash Control Register
[0x0024]	*FLC_INTR*	Flash Interrupt Register
[0x0030]	*FLC_DATA*	Flash Data Register 0
[0x0034]	*FLC_DATA1*	Flash Data Register 1
[0x0038]	*FLC_DATA2*	Flash Data Register 2
[0x003C]	*FLC_DATA3*	Flash Data Register 3
[0x0040]	*FLC_ACNTL*	Flash Access Control Register

7.3.1 Register Details

Table 7-3. Flash Controller Address Pointer Register

Flash Address			FLC_ADDR	[0x00]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	*	**Flash Address** This field contains the target address for a write operation. A valid internal flash memory address is required for all write operations. *The reset value for this field is always 0x0000 0000.

Table 7-4. Flash Controller Clock Divisor Register

Flash Controller Clock Divisor			FLC_CLKDIV	[0x04]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	clkdiv	R/W	0x60	**Flash Controller Clock Divisor** The APB clock is divided by the value in this field to generate the FLC peripheral clock,f_{FLC_CLK}. The FLC peripheral clock must equal 1MHz. The default on all forms of reset is 48 (0x30), resulting in $f_{FLC\_CLK} = 1MHz$. The FLC peripheral clock is only used during erase and program functions and not during read functions. See *Clock Configuration* for details.

Table 7-5. Flash Controller Control Register

Flash Controller Control			FLC_CN	[0x08]
Bits	**Name**	**Access**	**Reset**	**Description**
31:28	unlock	R/W	0	**Flash Unlock** Write the unlock code, 0x2, before any flash write or erase operation to unlock the flash. Writing any other value to this field locks the internal flash. 0x2: Flash unlock code
27	brst	RO	0	**Reserved**
26	-	RO	0	**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_LP_CTRL.ovr field must be set to 0b00 or 0b01 before setting this field to 1.*

Flash Controller Control			FLC_CN	[0x08]
Bits	**Name**	**Access**	**Reset**	**Description**
24	pend	RO	0	**Flash Busy Flag** When this field is set, writes to all flash registers except for the *FLC_INTR* register are ignored by the FLC. This bit is automatically cleared by hardware once the flash becomes accessible. 0: Flash idle 1: Flash busy *Note: If the FLC is busy (FLC_CN.pend = 1), reads, writes, and erase operations are not allowed and result in an access failure (FLC_INTR.af = 1).*
23:16	-	RO	0	**Reserved**
15:8	erase_code	R/W	0	**Erase Code** Before an erase operation, this field must be set to 0x55 for a page erase or 0xAA for a mass erase. The flash must be unlocked before setting the erase code. This field is automatically cleared after the erase operation is complete. 0x00: Erase disabled. 0x55: Page erase code. 0xAA: Enable mass erase through the JTAG debug port.
7:5	-	RO	0	**Reserved**
4	wdth	R/W	0	**Data Width Select** This field sets the data width of a write to the flash page. The flash controller supports either 32-bit wide writes or 128-bit wide writes. 0: 128-bit transactions (*FLC_DATA3*,*FLC_DATA2*, *FLC_DATA1*, *FLC_DATA*) 1: 32-bit transactions (*FLC_DATA* only)
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 before attempting a page erase. See *FLC_CN.unlock* for details. The FLC hardware clears this bit when a page erase operation is complete. 0: No page erase operation in process or page erase is complete. 1: Write a 1 to initiate a page erase. If this field reads 1, a page erase operation is in progress. *Note: This field is protected and cannot be set to 0 by application code.*
1	me	R/W1O	0	**Mass Erase** Write a 1 to this field to initiate a mass erase of the internal flash memory. The flash must be unlocked before attempting a mass erase. See *FLC_CN.unlock* for details. The FLC 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 FLC writes 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.

Table 7-4: Flash Controller Interrupt Register

Flash Controller Interrupt			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 to 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 7-5: Flash Controller Data Register 0

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

Table 7-6: Flash Controller Data Register 1

Flash Controller Data 1			FLC_DATA1	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data1	R/W	0	**Flash Data 1** Flash data for bits 63:32

Table 7-7: Flash Controller Data Register 2

flash controller Data 2			FLC_DATA2	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data2	R/W	0	**Flash Data 2** Flash data for bits 95:64

Table 7-8: Flash Controller Data Register 3

Flash Controller Data 3			FLC_DATA3	[0x003C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data3	R/W	0	**Flash Data 3** Flash data for bits 127:96.

Table 7-9: Flash Controller Access Control Register

Flash Controller Data 3			FLC_ACNTL	[0x003C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**Reserved**

8. General-Purpose I/O and Alternate Function Pins

The general-purpose I/O (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 through the GPIO input register 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, and I/O mode, and the software must manage the AF and GPIO to ensure each is set up correctly when switching from a peripheral to the I/O function. Refer to the device data sheet electrical characteristics table, http://www.maximintegrated.com, 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, 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 is identical across multiple Maxim Integrated microcontrollers; however, several registers' behavior varies depending on the specific device. The registers' behavior should not be assumed to be the same from one device to a different device. Specifically the registers GPIOn_PAD_CFG, GPIOn_PAD_CFG2, GPIOn_IS, GPIOn_SR, GPIOn_DS, 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

8.1 Instances

Table 8-1 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 8-1: GPIO Pin Count

Package	Number of GPIO	Pins
16 WLP	GPIO0[9:0]	10
20 TQFN	GPIO0[13:0]	14
24 TQFN	GPIO0[13:0]	14

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

Note: MAX32660 does not support the selectable GPIO voltage supply feature.

8.2 Configuration

8.2.1 Power-On-Reset Configuration

During a POR event, all I/O default to GPIO mode as high-impedance inputs except the SWDIO and SWDCLK pins. The SWD is enabled by default after POR with AF1 selected by hardware.

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

- GPIO mode enabled:
  - *GPIOn_EN[pin]* = 1
  - *GPIOn_EN1[pin]* = 0
  - *GPIOn_EN2[pin]* = 0
- Pullup/Pulldown disabled, I/O in Hi-Z mode
  - *GPIOn_PAD_CFG[pin]* = 0
- Output mode disabled:
  - *GPIOn_OUT_EN[pin]* = 0
- Interrupt disabled:
  - *GPIOn_INT_EN[pin]* = 0

8.2.2 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_EN[pin]* = 1
   b. *GPIOn_EN1[pin]* = 0
   c. *GPIOn_EN2[pin]* = 0
2. Configure the pin for pullup, pulldown, or high-impedance mode. See the *GPIOn_PS* register for pullup and pulldown selection.
   a. GPIO pins with I^2C as an AF only support high-impedance or a weak pulldown resistor.
3. Set *GPIOn_PAD_CFG[pin]* to 1 to enable the pull resistor or clear the bit to set the input to high-impedance mode.
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 8-2*.

Table 8-2: MAX32660 Input Mode Configuration Summary

Input Mode	Mode Select *GPIOn_PAD_CFG[pin]*	Pullup/Pulldown Strength *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.

8.2.3 Output Mode Configuration

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

1. Set the pin for I/O mode:
   a. *GPIOn_EN.config[pin]* = 1
   b. *GPIOn_EN1.config[pin]* = 0

c. *GPIOn_EN2.config[pin]* = 0

2. Enable the output buffer for the pin by setting *GPIOn_OUT_EN.en[pin]* to 1.
3. Set the output drive strength using the *GPIOn_DS1[pin]* and *GPIOn_DS[pin]* bits.
   a. See the section *GPIO Drive Strength* for configuration details and the modes supported.
   b. Reference the device data sheet for the electrical characteristics for the drive strength modes.
4. Set the output high or low using the *GPIOn_OUT[pin]* bit.

8.2.4 Serial Wire Debug Configuration

Perform the following steps to configure the SWDIO and SWDCLK device pins for SWD mode on P0.0 and P0.1:

1. Set the device pin P0.0 for AF1 mode:
   a. *GPIOn_EN.config[0]* = 0
   b. *GPIOn_EN1.config[0]* = 0
   c. *GPIOn_EN2.config[0]* = 0
2. Set device pin P0.1 for AF1 mode:
   a. *GPIOn_EN.config[1]* = 0
   b. *GPIOn_EN1.config[1]* = 0
   c. *GPIOn_EN2.config[1]* = 0

Perform the following steps to configure the SWDIO and SWDCLK device pins for SWD mode on P0.8 and P0.9:

1. Set the device pin P0.8 for AF2 mode:
   a. *GPIOn_EN.config[0]* = 0
   b. *GPIOn_EN1.config[0]* = 1
   c. *GPIOn_EN2.config[0]* = 0
2. Set device pin P0.9 for AF1 mode:
   a. *GPIOn_EN.config[1]* = 0
   b. *GPIOn_EN1.config[1]* = 1
   c. *GPIOn_EN2.config[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_SCON*.swd_dis = 1).*

8.2.5 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_DS* registers as shown in *Table 8-3*. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO0_DS.str[9], GPIO0_DS1.str[9]* both represent configuration for device pin P0.9. 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 8-3: Standard GPIO Drive Strength Selection

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

For GPIO with I²C as an AF, *Table 8-4* shows the drive strength setting options.

Table 8-4: GPIO with I²C AF Drive Strength Selection

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

8.3 Alternate Function Configuration

Table 8-5 shows the AF selection matrix. Write the *GPIOn_EN* and *GPIOn_EN1* fields as shown in the table to select the desired AF.

Note: Each AF is independently selectable. Mixing functions assigned to AF1, AF2, or AF3 is supported if all the peripheral's required functions are enabled.

Table 8-5: GPIO Mode and AF Selection

GPIO MODE	*GPIOn_EN2.config[pin]*	*GPIOn_EN1.config*[pin]	*GPIOn_EN.config*[pin]
I/O	0	0	1
AF1	0	0	0
AF2	0	1	0
AF3	0	1	1

Table 8-6: GPIO Mode and AF Transition Selection

GPIO MODE	*GPIOn_EN2.config[pin]*	*GPIOn_EN1.config*[pin]	*GPIOn_EN.config*[pin]
I/O (transition to AF1)	0	0	1
I/O (transition to AF2)	0	1	1
I/O (transition to AF3)	1	0	1

Most GPIO support one or more alternate functions selected with the GPIO configuration enable bits shown in *Table 8-6*. The bits that select the AF must only be changed while the pin is in one of the I/O modes (*GPIOn_EN* = 1). The specific I/O mode must match the desired AF. For example, if a transition to AF1 is desired, first select the setting corresponding to I/O (transition to AF1). Then enable the desired mode by selecting the AF1 mode.

1. Set the GPIO configuration enable bits shown in *Table 8-6* to the I/O mode that corresponds with the desired new AF setting. For example, select "I/O (transition to AF1)" if switching to AF1. Switching between different I/O mode settings does not affect the state or electrical characteristics of the pin.
2. Configure the electrical characteristics of the pin. See *Table 8-2* if the assigned alternate function uses the pin as an input. See *Table 8-3* if the assigned alternate function uses the pin as an output.
3. Set the GPIO configuration enable bits shown in *Table 8-6* to the desired alternate function.

Table 8-7: GPIO AF Configuration Reference

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

8.3.1 Configuring GPIO (External) Interrupts

Each GPIO supports external interrupt events when the GPIO is configured for I/O mode, and the input mode is enabled. If the GPIO is configured as a peripheral AF, the interrupts are peripheral controlled. GPIO interrupts can be 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_INT_EN[pin]* field to 0. This prevents any new interrupts on the pin from triggering but does not clear previously triggered (pending) interrupts. The application can disable all interrupts for GPIO by writing 0 to *GPIOn_INT_EN[13:0]*. To maintain previously enabled interrupts, read the *GPIOn_INT_EN* register and save the value to memory before setting the register to 0.
2. Clear pending interrupts by writing 1 to the *GPIOn_INT_CLR[pin]* bit.
3. Set *GPIOn_INT_MODE[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.
    b. *GPIOn_INT_POL[pin]* = 1: Input high triggers interrupt.
    c. *GPIOn_INT_POL[pin]* = 0: Input low triggers interrupt.
    d. For edge-triggered interrupts, the interrupt triggers on an edge event.
    e. *GPIOn_INT_POL[pin]* = 0: Input rising edge triggers interrupt.
    f. *GPIOn_INT_POL[pin]* = 1: Input falling edge triggers interrupt.
4. Optionally set *GPIOn_INT_DUAL_EDGE[pin]* to 1 to trigger on both the rising and falling edges of the input signal.
5. Set *GPIOn_INT_EN[pin]* to 1 to enable the interrupt for the pin.

8.3.2 Using GPIO for Wakeup from Low-Power Modes

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

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

Table 8-8: GPIO Wakeup Interrupt Vector

GPIO Wake Interrupt Source	GPIO Wake Interrupt Status Register	GPIO Wakeup Interrupt Vector
GPIO0[30:0]	*GPIOn_STAT*	GPIOWAKE_IRQn

Enable low-power mode wakeup (*SLEEP*, *DEEPSLEEP,* and *BACKUP*) from an external GPIO event by completing the following steps:

1. Set the polarity (rising or falling edge) by writing to the *GPIOn_INT_POL[pin]* field. The wakeup functionality uses rising and falling edge detection circuitry that operates asynchronously and does not require an active clock. Dual-edge mode is also an option to accomplish edge detection wakeup.
2. Clear pending interrupt flags by writing 0xFF to the *GPIOn_INT_CLR* register.
3. Activate the GPIO wakeup function by writing 1 to *GPIOn_WAKE_EN[pin]*.
4. Configure the power manager to use the GPIO as a wakeup source by writing 1 to the *GCR_PM.gpiowken*.

8.4 GPIO Registers

See *Table 3-1* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 8-9*. 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 8-9: GPIO Register Summary

Offset	Register Name	Description
[0x0000]	GPIOn_EN	GPIO Port n Configuration Enable Bit 0 Register
[0x0004]	GPIOn_EN_SET	GPIO Port n Configuration Enable Atomic Set Bit 0 Register
[0x0008]	GPIOn_EN_CLR	GPIO Port n Configuration Enable Atomic Clear Bit 0 Register
[0x000C]	GPIOn_OUT_EN	GPIO Port n Output Enable Register
[0x0010]	GPIOn_OUT_EN_SET	GPIO Port n Output Enable Atomic Set Register
[0x0014]	GPIOn_OUT_EN_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_INT_MODE	GPIO Port n Interrupt Mode Register
[0x002C]	GPIOn_INT_POL	GPIO Port n Interrupt Polarity Register
[0x0034]	GPIOn_INT_EN	GPIO Port n Interrupt Enable Register
[0x0038]	GPIOn_INT_EN_SET	GPIO Port n Interrupt Enable Atomic Set Register
[0x003C]	GPIOn_INT_EN_CLR	GPIO Port n Interrupt Enable Atomic Clear Register
[0x0040]	GPIOn_STAT	GPIO Port n Interrupt Status Register
[0x0048]	GPIOn_INT_CLR	GPIO Port n Interrupt Clear Register
[0x004C]	GPIOn_WAKE_EN	GPIO Port n Wakeup Enable Register
[0x0050]	GPIOn_WAKE_EN_SET	GPIO Port n Wakeup Enable Atomic Set Register
[0x0054]	GPIOn_WAKE_EN_CLR	GPIO Port n Wakeup Enable Atomic Clear Register
[0x005C]	GPIOn_INT_DUAL_EDGE	GPIO Port n Interrupt Dual Edge Mode Register
[0x0060]	GPIOn_PAD_CFG	GPIO Port n Pad Control 0 Register
[0x0064]	GPIOn_PAD_CFG2	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_IS	GPIO Port n Input Hysteresis Enable Register
[0x00AC]	GPIOn_SR	GPIO Port n Slew Rate Select Register
[0x00B0]	GPIOn_DS	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

8.4.1 Register Details

Table 8-10: GPIO AF 0 Select Register

GPIO AF 0 Select Register			GPIOn_EN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	R/W	1	**GPIO Configuration Enable Bit 0** In conjunction with bits in *Table 8-5*, these bits configure the corresponding device pin for digital I/O or an AF mode. These bits can be modified directly by writing to this register or indirectly through *GPIOn_EN_SET* or *GPIOn_EN_CLR*. *Table 8-7* depicts a detailed example of how each of these bits applies to each of the GPIO device pins. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect the input and interrupt functionality of the associated pin.*

Table 8-11: GPIO Port n Configuration Enable Atomic Set Bit 0 Register

GPIO Port n Configuration Enable Atomic Set Bit 0			GPIOn_EN_SET	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Configuration Enable Atomic Set Bit 0** Setting a bit in this field sets the corresponding bit in the *GPIOn_EN* register. 0: No effect 1: Corresponding bit in *GPIOn_EN* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-12: GPIO Port n Configuration Enable Atomic Clear Bit 0 Register

GPIO Port n Configuration Enable Atomic Clear Bit 0			GPIOn_EN_CLR	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1	0	**GPIO Configuration Enable Atomic Clear Bit 0** Setting a bit in this field clears the corresponding bits in the *GPIOn_EN* register. 0: No effect 1: Corresponding bits in *GPIOn_EN* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-13: GPIO Port n Output Enable Register

GPIO Port n Output Enable			GPIOn_OUT_EN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	en	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_OUT_EN_SET* or *GPIOn_OUT_EN_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 8-14: GPIO Port n Output Enable Atomic Set Register

GPIO Port n Output Enable Atomic Set			GPIOn_OUT_EN_SET	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Output Enable Atomic Set** Setting a bit in this field sets the corresponding bit in the *GPIOn_OUT_EN* register. 0: No effect 1: Corresponding bits in *GPIOn_OUT_EN* 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 8-15: GPIO Port n Output Enable Atomic Clear Register

GPIO Port n Output Enable Atomic Clear			GPIOn_OUT_EN_CLR	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1	0	**GPIO Output Enable Atomic Clear** Setting a bit in this field sets the corresponding bits in the *GPIOn_OUT_EN* register. 0: No effect 1: Corresponding bits in *GPIOn_OUT_EN* 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 8-16: GPIO Port n Output Register

GPIO Port n Output			GPIOn_OUT	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	level	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: The bit is ignored if the corresponding bit position in the GPIOn_OUT_EN 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 8-17: GPIO Port n Output Atomic Set Register

GPIO Port n Output Atomic Set			GPIOn_OUT_SET	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	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_OUT_EN* 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 8-18: GPIO Port n Output Atomic Clear Register

GPIO Port n Output Atomic Clear			GPIOn_OUT_CLR	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	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_OUT_EN* 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 8-19: GPIO Port n Input Register

GPIO Port n Input			GPIOn_IN	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	level	R	-	**GPIO Input Level** Read the state of the corresponding input pin. The input state is always readable for a pin regardless of its 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 8-20: GPIO Port n Interrupt Mode Register

GPIO Port n Interrupt Mode			GPIOn_INT_MODE	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	mode	R/W	0	**GPIO Interrupt Mode** Setting a bit in this field sets edge-triggered interrupts for corresponding GPIO pin. Clearing a bit in this field sets level triggered interrupt for corresponding GPIO pin. 0: Level triggered interrupt. 1: Edge triggered interrupt. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-21: GPIO Port n Interrupt Polarity Register

GPIO Port n Interrupt Polarity			GPIOn_INT_POL	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	pol	R/W	0	**GPIO Interrupt Polarity** Interrupt polarity selection bit for the corresponding GPIO pin. Level triggered mode (*GPIOn_INT_MODE*.[pin]= 0): 0: Input low (logic 0) triggers interrupt. 1: Input high (logic 1) triggers interrupt. Edge triggered mode (*GPIOn_INT_MODE*.[pin]= 1): 0: Falling edge triggers interrupt. 1: Rising edge triggers interrupt. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-22: GPIO Port n Interrupt Enable Registers

GPIO Port n Interrupt Enable			GPIOn_INT_EN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	ie	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_INT_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 8-23: GPIO Port n Interrupt Enable Atomic Set Register

GPIO Port Interrupt Enable Atomic Set			GPIOn_INT_EN_SET	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Interrupt Enable Atomic Set** Setting a bit in this field sets the corresponding bits in the *GPIOn_INT_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_INT_EN* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-24: GPIO Port n Interrupt Enable Atomic Clear Register

GPIO Port Interrupt Enable Atomic Clear			GPIOn_INT_EN_CLR	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1	0	**GPIO Interrupt Enable Atomic Clear** Setting a bit in this field clears the corresponding bits in the *GPIOn_INT_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_INT_EN* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-25: GPIO Interrupt Status Register

GPIO Interrupt Status			GPIOn_STAT	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	if	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_INT_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 8-26: GPIO Port n Interrupt Clear Register

GPIO Port Interrupt Clear			GPIOn_INT_CLR	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1C	0	**GPIO Interrupt Clear** Setting a bit in this field clears the associated interrupt status (*GPIOn_STAT*). 0: No effect on the associated *GPIOn_STAT* flag. 1: Clear the associated interrupt pending flag in the *GPIOn_STAT* 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 8-27: GPIO Port n Wakeup Enable Register

GPIO Port n Wakeup Enable Register			GPIOn_WAKE_EN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	we	R/W	0	**GPIO Wakeup Enable** Setting a bit in this field enables the corresponding GPIO pin as a wakeup from low power modes (*SLEEP*, *DEEPSLEEP*, *BACKUP*). 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 8-28: GPIO Port n Wakeup Enable Atomic Set Register

GPIO Port Wakeup Enable Atomic Set			GPIOn_WAKE_EN_SET	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Wakeup Enable Atomic Set** Setting a bit in this field sets the corresponding bits in the *GPIOn_WAKE_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_WAKE_EN* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-29: GPIO Port n Wakeup Enable Atomic Clear Register

GPIO Port Wakeup Enable Atomic Clear			GPIOn_WAKE_EN_CLR	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1	0	**GPIO Wakeup Enable Atomic Clear** Setting a bit in this field clears the corresponding bits in the *GPIOn_WAKE_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_WAKE_EN* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-30: GPIO Port n Interrupt Dual Edge Mode Register

GPIO Port n Interrupt Dual Edge Mode			GPIOn_INT_DUAL_EDGE	[0x005C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	de_en	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_INT_MODE* bit must be set to edge-triggered. When enabled, the associated polarity (*GPIOn_INT_POL*) 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 8-31: GPIO Port n Pad Control 0 Register

GPIO Port n Pad Control 0			GPIOn_PAD_CFG1	[0x0060]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	R/W	0	**GPIO0 Pull Up/Pull Down Enable** Setting a bit in this field enables either the weak pullup or weak pulldown resistor on the corresponding GPIO port device pin. The selection for pullup or pulldown resistor is set using the *GPIOn_PS* register. *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 GPIOn_PS is set to 1, setting the same bit in this register has no effect.*

Table 8-32: GPIO Port n Pad Control 1 Register

GPIO Port n Pad Control 1			GPIOn_PAD_CFG2	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	RO	0	**Reserved**

Table 8-33: 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	config	R/W	0	**GPIO AF 1 Mode Select** In conjunction with the bits in *Table 8-5*, these bits configure the corresponding device pin for digital I/O or an AF mode. This field can be modified directly by writing to this register or indirectly through *GPIOn_EN1_SET* or *GPIOn_EN1_CLR* *Table 8-7* depicts a detailed example of how each of these bits applies to each of the GPIO device pins. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect the input and interrupt functionality of the associated pin.*

Table 8-34: 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	set	R/W1	0	**GPIO Configuration Enable Atomic Set Bit 1** Setting a bit in this field sets the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-35: 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	clr	R/W1	0	**GPIO Configuration Enable Atomic Clear Bit 1** Setting a bit in this field clears the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-36: 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	config	R/W	0	**GPIO Configuration Enable Bit 2** In conjunction with the bits in *Table 8-5*, these bits configure the corresponding device pin for digital I/O or an AF mode. This field can be modified directly by writing to this register or indirectly through *GPIOn_EN2_SET*or *GPIOn_EN2_CLR*. *Table 8-7* depicts a detailed example of how each of these bits applies to each of the GPIO device pins. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect the input and interrupt functionality of the associated pin.*

Table 8-37: 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	set	R/W1	0	**GPIO AF Select Atomic Set Bit 2** Setting a bit in this field sets the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-38: 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	clr	R/W1	0	**GPIO AF Select Atomic Clear Bit 2** Setting a bit in this field clears the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-39: GPIO Port n Input Hysteresis Enable Register

GPIO Port n Input Hysteresis Enable		GPIOn_IS		[0x00A8]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	en	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 8-40: GPIO Port n Slew Rate Enable Register

GPIO Port n Slew Rate Select		GPIOn_SR		[0x00AC]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	sr_sel	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: 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 8-41: GPIO Port n Output Drive Strength Bit 0 Register

GPIO Port n Output Drive Strength Bit 0		GPIOn_DS		[0x00B0]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	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_DS* bits for the associated GPIO pin. See the *GPIO Drive Strength* section for the selection options on these I/O pins. *Note: GPIO with I^2C as an AF only support two different drive strengths:* *0: Low output drive strength selected.* *1: High output drive strength selected.* *Refer to the symbols V_{OL_I2C} and V_{OH_I2C} in the device data sheet electrical characteristics table for details regarding which I/O pins support I^2C functionality.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-42: 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	config	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_DS* 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 to determine the drive strengths for these I/O pins.* *Note: GPIO with I^2C as an AF only support two different drive strengths and do not use any bits in this register for drive strength selection. See GPIOn_DS for details of the two different drive strength settings.* *Refer to the symbols V_{OL_I2C} and V_{OH_I2C} in the device data sheet Electrical Characteristics table for details regarding which I/O pins support I^2C functionality.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-43: 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	pull_sel	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 weak pulldown resistor for the corresponding GPIO port device pin. The *GPIOn_PAD_CFG* field must be configured to enable this pull resistor selection. 0: Pulldown resistor selected. 1: Pullup resistor selected. *Note: GPIO with I^2C as an AF do not support a weak pullup resistor. As such, the bits in this register that control GPIO with I^2C 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 I^2C functionality.* *See GPIOn_PAD_CFG for additional information.*

Table 8-44: GPIO Port n Voltage Select Register

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

9. 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 the Arm core. All DMA transactions consist of a burst read from the source into the internal DMA FIFO followed by a burst write from the internal DMA FIFO to the destination.

DMA transfers are one of three types:

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

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

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

9.1 Instances

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

Table 9-1: MAX32660 DMA and Channel Instances

DMA Instance	DMA_CHn Channel Instance
DMA	DMA_CH0
	DMA_CH1
	DMA_CH2
	DMA_CH3

9.2 DMA Channel Operation (DMA_CH)

9.2.1 Channel Arbitration and DMA Bursts

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

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

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

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

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

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

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

Only an error condition can interrupt an ongoing data transfer.

9.2.2 Source and Destination Addressing

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

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

Table 9-2: DMA Source and Destination by Peripheral

DMA_CHn_CFG.reqsel	Peripheral	DMA Source	DMA Destination
0	Memory-to-Memory	*DMA_CHn_SRC*	*DMA_CHn_DST*
1	SPI0	SPI0 Receive FIFO	*DMA_CHn_DST*
2	SPIMSS (SPI1/I²S)	SPIMSS Receive FIFO-	*DMA_CHn_DST*
3	Reserved	-	-
4	UART0	UART0 Receive FIFO	*DMA_CHn_DST*
5	UART1	UART1 Receive FIFO	*DMA_CHn_DST*
6	Reserved	-	-
7	I²C 0	I2C0 Receive FIFO	*DMA_CHn_DST*
8	I²C 1	I2C1 Receive FIFO	*DMA_CHn_DST*
9-32	Reserved	-	-
33	SPI 0	*DMA_CHn_SRC*	SPI0 Transmit FIFO
34	SPIMSS (SPI1/I²S)	*DMA_CHn_SRC*	SPIMSS Transmit FIFO-
35	Reserved	-	-
36	UART 0	*DMA_CHn_SRC*	UART0 Transmit FIFO
37	UART 1	*DMA_CHn_SRC*	UART1 Transmit FIFO
38	Reserved	-	-
39	I²C 0	*DMA_CHn_SRC*	I2C0 Transmit FIFO
40	I²C 1	*DMA_CHn_SRC*	I2C1 Transmit FIFO
42-63	Reserved	-	-

9.2.3 Data Movement from Source to DMA

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

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

Register/Field	Description	Comments
DMA_CHn_SRC	Source address	If the increment enable field, *DMA_CHn_CFG.srcinc*, is set, this address 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's value is decremented on each read of a burst.
DMA_CHn_CFG.brst	Burst size (1-32)	The maximum number of bytes moved during the burst read.
DMA_CHn_CFG.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_CFG.srcinc	Source address increment enable	Increments *DMA_CHn_SRC*. This field is ignored when the DMA source is a peripheral.

9.2.4 Data Movement from DMA to Destination

Table 9-4 shows the fields that control the burst movement of data out of the DMA FIFO. The destination is a peripheral or memory.

Table 9-4: Data Movement from the DMA FIFO to Destination

Register/Field	Description	Comments
DMA_CHn_DST	Destination address	If the increment enable field, *DMA_CHn_CFG.dstinc*, is set, this address increments on every write cycle of the burst. This field is ignored when the DMA destination is a peripheral.
DMA_CHn_CFG.brst	Burst size (1-32)	The maximum number of bytes moved during a single AHB read/write burst.
DMA_CHn_CFG.dstwd	Destination width	This field determines the maximum data width used during each write of the AHB burst (one byte, two bytes, or four bytes).
DMA_CHn_CFG.dstinc	Destination address increment enable	Increments *DMA_CHn_DST*. This field is ignored when the DMA destination is a peripheral.

9.3 Usage

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

1. Ensure *DMA_CHn_CFG.chen, DMA_CHn_CFG.rlden* = 0, and *DMA_CHn_ST.ctz_st* = 0.
2. If using RAM for the destination of the DMA transfer, configure *DMA_CHn_DST* to the starting address of the destination in RAM.
3. If using RAM for the source of the DMA transfer, configure *DMA_CHn_SRC* to the starting address of the source in RAM.
4. Write the number of bytes to transfer to the *DMA_CHn_CNT* register.
5. Configure the following *DMA_CHn_CFG* register fields in one or more instructions. Do not set *DMA_CHn_CFG.chen* to 1 or *DMA_CHn_CFG.rlden* to 1 in this step:
    a. Configure *DMA_CHn_CFG.reqsel* to select the transfer operation associated with the DMA channel.
    b. Configure *DMA_CHn_CFG.brst* for the desired burst size.
    c. Configure *DMA_CHn_CFG.pri* to set the channel priority relative to other DMA channels.
    d. Configure *DMA_CHn_CFG.dstwd* to dictate the number of bytes written in each transaction.
    e. If desired, set *DMA_CHn_CFG.dstinc* to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.
    f. Configure *DMA_CHn_CFG.srcwd* to dictate the number of bytes read in each transaction.
    g. If desired, set *DMA_CHn_CFG.srcinc* to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.
    h. If desired, set *DMA_CHn_CFG.chdien* = 1 to generate an interrupt when the channel becomes disabled. The channel becomes disabled when the DMA transfer completes or a bus error occurs.
    i. If desired, set *DMA_CHn_CFG.ctzien* 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_SRC_RLD* register with the source address reload value.
        2) Load the *DMA_CHn_DST_RLD* register with the destination address reload value.
        3) Load the *DMA_CHn_CNT_RLD* register with the count reload value.
    k. If desired, enable the channel timeout feature described in *Channel Timeout Detect*. Clear *DMA_CHn_CFG.pssel* to 0 to disable the channel timeout feature.
6. Set *DMA_CHn_CFG.rlden* to 1 to enable the reload feature if using.
7. Set *DMA_CHn_CFG.chen* to 1 to immediately start the DMA transfer.
8. Wait for the interrupt flag to become 1 to indicate the completion of the DMA transfer.

9.4 Count-To-Zero (CTZ) Condition

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

At this point, there are two possible responses depending on the value of the *DMA_CHn_CFG*.*rlden*:

1. If *DMA_CHn_CFG*.*rlden* = 1, then the *DMA_CHn_SRC*, *DMA_CHn_DST*, and *DMA_CHn_CNT* registers are loaded from the reload registers, and the channel remains active and continues operating using the newly-loaded address/count values and the previously programmed configuration values.
2. If *DMA_CHn_CFG*.*rlden* = 0, then the channel is disabled, and *DMA_CHn_ST*.*ch_st* is cleared.

9.5 Chaining Buffers

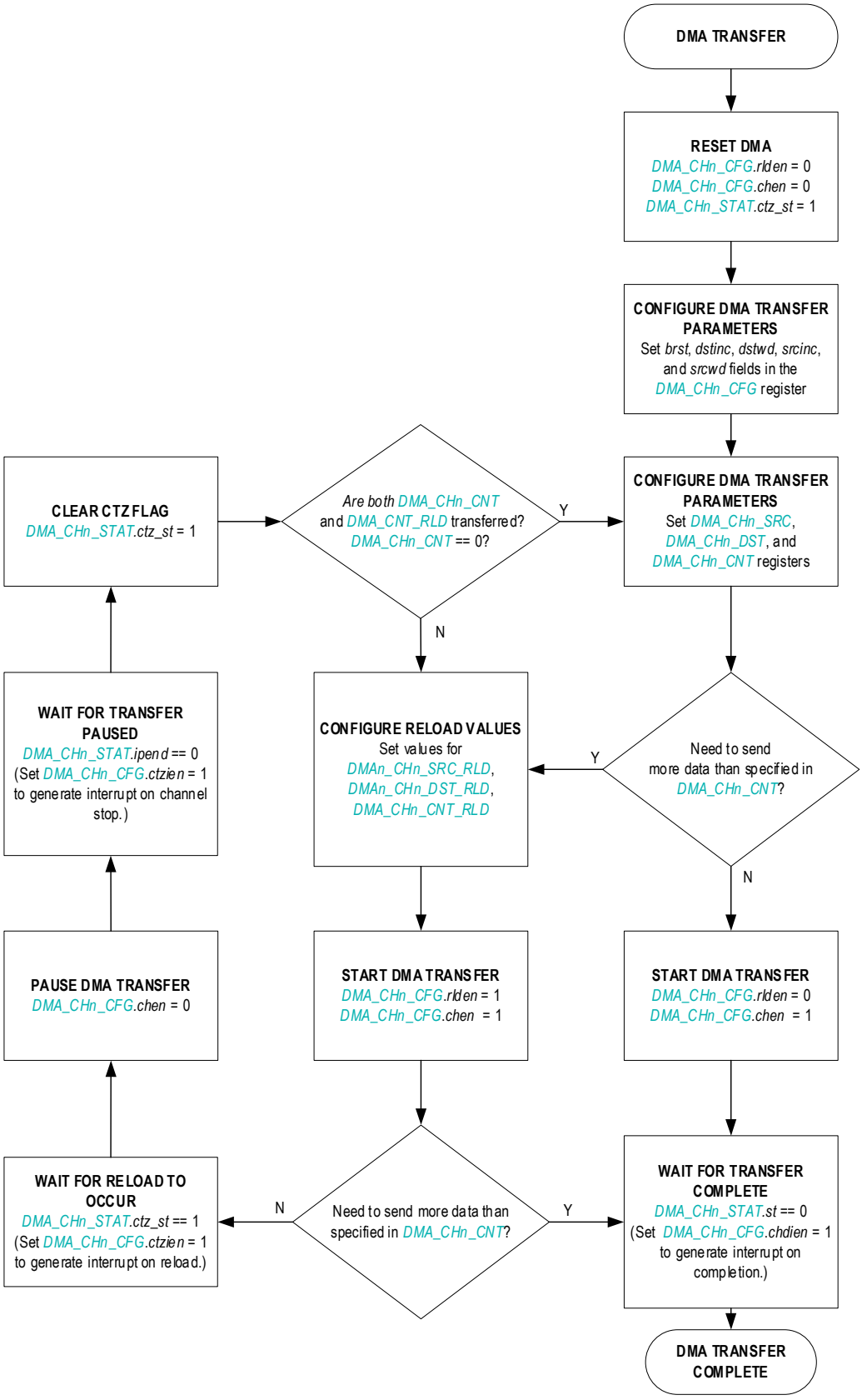
Chaining buffers reduce the DMA interrupt response time and allow the DMA to service requests without intermediate processing from the CPU. *Figure 9-1* shows the procedure for generating a DMA transfer using one or more chain buffers.

Configure the following reload registers to configure a channel for chaining:

- *DMA_CHn_SRC*
- *DMA_CHn_DST*
- *DMA_CHn_CNT*
- *DMA_CHn_SRC_RLD*
- *DMA_CHn_DST*
- *DMA_CHn_CNT_RLD*

Writing to any register while a channel is disabled is supported, but there are certain restrictions when a channel is enabled. The *DMA_CHn_ST*.*ch_st* bit indicates whether the channel is enabled or not. Because an active channel might be in the middle of an AHB read/write burst, do not write to the *DMA_CHn_SRC*, *DMA_CHn_DST*, or *DMA_CHn_CNT* registers while a channel is active (*DMA_CHn_ST*.*ch_st* = 1). To disable any DMA channel, clear the *DMA_CN*.*ch<n>_ien* bit. Then, poll the *DMA_CHn_ST*.*ch_st* bit to verify that the channel is disabled.

Figure 9-1: DMA Block-Chaining Flowchart



9.6 DMA Interrupts

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

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

- *DMA_CHn_CFG.ctzien* = 1
  - If enabled, all CTZ occurrences set the *DMA_CHn_ST.ipend* bit.
- *DMA_CHn_CFG.chdien* = 1
  - If enabled, any clearing of the *DMA_CHn_ST.ch_st* bit sets the *DMA_CHn_ST.ipend* bit. Examine the *DMA_CHn_ST* register to determine which reasons caused the disable. The *DMA_CHn_CFG.chdien* bit also enables the *DMA_CHn_ST.to_st* bit. The *DMA_CHn_ST.to_st* bit does not clear the *DMA_CHn_ST.ch_st* bit.

To clear the channel interrupt, write 1 to the cause of the interrupt (the *DMA_CHn_ST.ctz_st*, *DMA_CHn_ST.rld_st*, *DMA_CHn_ST.bus_err*, or *DMA_CHn_ST.to_st* bits).

When running in normal mode without buffer chaining (*DMA_CHn_CFG.rlden* = 0), set the *DMA_CHn_CFG.chdien* bit only. An interrupt is generated upon DMA completion or an error condition (bus error or timeout error).

When running in buffer chaining mode (*DMA_CHn_CFG.rlden* = 1), set both the *DMA_CHn_CFG.chdien* and *DMA_CHn_CFG.ctzien* bits. The CTZ interrupts occur on completion of each DMA (count reaches zero, and reload occurs). The setting of *DMA_CHn_CFG.chdien* ensures that an error condition generates an interrupt. If *DMA_CHn_CFG.ctzien* = 0, then the only interrupt occurs when the DMA completes and *DMA_CHn_CFG.rlden* = 0 (final DMA).

9.7 Channel Timeout Detect

Each channel can optionally generate an interrupt when the associated peripheral does not request a transfer in a user-configurable period. When the timeout start conditions are met, an internal 10-bit counter begins incrementing at a frequency determined by the AHB clock, *DMA_CHn_CFG.pssel*, and *DMA_CHn_CFG.tosel* shown in *Table 9-5*. A channel timeout event is generated if the timer is not reset by one of the events listed below before the timeout period expires.

Table 9-5: DMA Channel Timeout Configuration

DMA_CHn_CFG.pssel	Timeout Period (µs)
0	Channel timeout disabled.
1	$\frac{2^8 \times [Value\ from\ DMA\_CHn\_CFG.tosel]}{f_{HCLK}}$
2	$\frac{2^{16} \times [Value\ from\ DMA\_CHn\_CFG.tosel]}{f_{HCLK}}$
3	$\frac{2^{24} \times [Value\ from\ DMA\_CHn\_CFG.tosel]}{f_{HCLK}}$

DMA_CHn_CFG.reqwait controls the start of the timeout period:

- If *DMA_CHn_CFG.reqwait* = 0, the timer begins immediately counting after *DMA_CHn_CFG.tosel* is configured to a value other than 0 and the channel is enabled.
- If *DMA_CHn_CFG.reqwait* = 1, the timer begins counting when the first DMA request is received from the peripheral.

The timer is reset whenever:

- The DMA request line programmed for the channel is activated.
- The channel is disabled for any reason (*DMA_CHn_ST.ch_st* = 0).

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

9.8 Memory-to-Memory DMA

Memory-to-memory transfers are processed as if the request is always active. As a result, the DMA channel generates an almost constant request for the bus until its transfer is complete. For this reason, assign a lower priority to channels executing memory-to-memory transfers to prevent starvation of other DMA channels.

9.9 DMA Registers

See *Table 3-1* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 9-6*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 9-6: DMA Register Summary

Offset	Register	Description
[0x0000]	*DMA_CN*	DMA Control register
[0x0004]	*DMA_INTR*	DMA Interrupt Status register

9.9.1 Register Details

Table 9-7: DMA Interrupt Flag Register

DMA Interrupt Enable			DMA_CN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3:0	ch<n>_ien	R/W	0	**DMA Channel *n* Interrupt Enable** Each bit in this field enables the corresponding channel interrupt *n* in *DMA_INTR*. Register bits associated with unimplemented channels should not be changed from their default reset value. 0: Disabled 1: Enabled

Table 9-8: DMA Interrupt Enable Register

DMA Interrupt Enable			DMA_INTR	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3:0	ch<n>_ipend	R	0	**DMA Channel *n* Interrupt Flag** Each bit in this field represents an interrupt for the corresponding channel interrupt *n*. To clear an interrupt, clear the corresponding active interrupt bit in the *DMA_CHn_ST* register. An interrupt bit in this field is set only if the corresponding interrupt enable field is set in the *DMA_CN* register. Register bits associated with unimplemented channels should be ignored. 0: No interrupt 1: Interrupt pending

9.10 DMA Channel Registers

Table 9-9: Standard DMA Channel 0 to Channel 3 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

9.10.1 DMA Channel Register Details

See *Table 3-1* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 9-10*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 9-10: DMA Channel Registers Summary

Offset	Register	Description
[0x0000]	*DMA_CHn_CFG*	DMA Channel n Configuration Register
[0x0004]	*DMA_CHn_ST*	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_SRC_RLD*	DMA Channel n Source Reload Register
[0x0018]	*DMA_CHn_DST_RLD*	DMA Channel n Destination Reload Register
[0x001C]	*DMA_CHn_CNT_RLD*	DMA Channel n Count Reload Register

Table 9-11: DMA_CH n Control Register

DMA Channel n Control			DMA_CHn_CFG	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
31	ctzien	R/W	0	**CTZ Interrupt Enable** 0: Disabled 1: Enabled. *DMA_INTR.ch<n>_ipend* is set to 1 whenever a CTZ event occurs.
30	chdien	R/W	0	**Channel Disable Interrupt Enable** 0: Disabled 1: Enabled. *DMA_INTR.ch<n>_ipend* bit is set to 1 whenever *DMA_CHn_ST.ch_st* changes from 1 to 0.
29	-	RO	0	**Reserved**
28:24	brst	R/W	0	**Burst Size** The number of bytes transferred into and out of the DMA FIFO in a single burst. 0b00000: 1 byte 0b00001: 2 bytes 0b00010: 3 bytes ... 0b11111: 32 bytes
23	-	RO	0	**Reserved**
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: One byte 1: Two bytes 2: Four bytes 3: Reserved
19	-	RO	0	**Reserved**
18	srcinc	R/W	0	**Source Increment on AHB Transaction Enable** This bit enables the automatic increment of the *DMA_CHn_SRC* register upon every AHB transaction. This bit is ignored for a DMA receive from peripherals. 0: Disabled 1: Enabled
17:16	srcwd	R/W	0	**Source Width** This field 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: One byte 1: Two bytes 2: Four bytes 3: Reserved

DMA Channel n Control			DMA_CHn_CFG	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
15:14	pssel	R/W	0	**Timeout Timer Clock Pre-Scale Select** This field selects the pre-scale divider for the timer clock input. 0: Timer disabled. 1: $\frac{f_{HCLK}}{28}$ 2: $\frac{f_{HCLK}}{216}$ 3: $\frac{f_{HCLK}}{224}$
13:11	tosel	R/W	0	**Timeout Period Select** This field selects the number of pre-scaled clocks seen by the channel timer before a timeout condition is generated. The value is approximate because of synchronization delays between timers. 0: 3 to 4 1: 7 to 8 2: 15 to 16 3: 31 to 32 4: 63 to 64 5: 127 to 128 6: 255 to 256 7: 511 to 512
10	reqwait	R/W	0	**Request DMA Timeout Timer Wait Enable** This field controls when the timeout timer starts, either immediately when the timeout timer is enabled or after the first DMA transaction occurs. 0: Start timer immediately when enabled. 1: Delay timer start until after the first DMA transaction occurs.
9:4	reqsel	R/W	0	**Request Select** Selects the source and destination for the transfer as shown in *Source and Destination Addressing*.
3:2	pri	R/W	0	**Channel Priority** This field sets the priority of the channel relative to other channels of the DMA peripheral. Channels of the same priority are serviced in a round-robin fashion. 0: High 1: Medium-high 2: Medium-low 3: Low
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 upon a CTZ. When this bit is set to 0, and a CTZ occurs, the channel is disabled, and the *DMA_CHn_ST*.*ch_st* bit is set to 0. 0: The channel is disabled when a CTZ occurs, and the *DMA_CHn_ST*.*ch_st* is set to 0. 1: Automatically reload the *DMA_CHn_SRC*, *DMA_CHn_DST*, and *DMA_CHn_CNT* registers on a CTZ.

DMA Channel n Control			DMA_CHn_CFG	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
0	chen	R/W	0	**Channel Enable** This bit is automatically cleared when *DMA_CHn_ST.ch_st* changes from 1 to 0. 0: Disabled 1: Enabled

Table 9-12: DMA Status Register

DMA Channel n Status			DMA_CHn_ST	[0x0104]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6	to_st	R/W1C	0	**Timeout Interrupt Flag** Timeout. Write 1 to clear. 0: No time out. 1: A channel time out has occurred
5	-	RO	0	**Reserved**
4	bus_err	R/W1C	0	**Bus Error** If this bit reads 1, an AHB abort occurred, and the channel was disabled by hardware. Write 1 to clear. 0: No error found 1: An AHB bus error occurred
3	rld_st	R/W1C	0	**Reload Interrupt Flag** Write 1 to clear. 0: Reload has not occurred. 1: Reload occurred.
2	ctz_st	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	ch_st	RO	0	**Channel Status** This bit indicates when it is safe to change the channel's configuration, address, and count registers. Whenever this bit is cleared by hardware, the *DMA_CHn_CFG.chen* bit is also cleared. 0: Disabled 1: Enabled.

Table 9-13: 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_CFG.srcinc* = 1, then this register is incremented on each AHB transfer cycle by one, two, or four bytes depending on the data width. If *DMA_CHn_CFG.srcinc* = 0, this register remains constant. If a CTZ condition occurs while *DMA_CHn_CFG.rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_SRC_RLD* register.

Table 9-14: DMA Channel n Destination Register

DMA Channel n Destination			DMA_CHn_DST	[0x010C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**Destination Device Address** For peripheral transfers, the actual address field is either ignored or forced to zero because peripherals only have one location to read/write data based on the request select chosen. If *DMA_CHn_CFG.dstinc* = 1, then this register is incremented on every AHB transfer cycle by one, two, or four bytes depending on the data width. If a CTZ condition occurs while *DMA_CHn_CFG.rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_DST_RLD* register.

Table 9-15: DMA Channel n Count Register

DMA Channel n Count			DMA_CHn_CNT	[0x0110]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23:0	cnt	R/W	0	**DMA Counter** Load this register with the number of bytes to transfer. This field decreases on every AHB access to the DMA FIFO. The decrement is one, two, or four bytes depending on the data width. When the counter reaches 0, a CTZ condition is triggered. If a CTZ condition occurs while *DMA_CHn_CFG.rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_CNT_RLD.cnt_rld* field.

Table 9-16: DMA Channel n Source Reload Register

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

Table 9-17: DMA Channel n Destination Reload Register

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

Table 9-18: DMA Channel n Count Reload Register

DMA Channel n Count Reload			DMA_CHn_CNT_RLD	[0x011C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23:0	cnt_rld	R/W	0	**Count Reload Value** If *DMA_CHn_CFG.rlden* = 1, this register's value is loaded into the *DMA_CHn_CNT* register on a CTZ condition.

10. UART

The industry standard full-duplex universal asynchronous receiver/transmitter (UART) ports communicate with external devices using standard serial communications protocols. Each UART instance supports identical functionality and registers unless expressly noted otherwise.

The following features are provided:

- Flexible baud rate generation
- Programmable character size of 5-bits to 8-bits
- Stop bit settings of 1, 1.5, or 2-bits
- Parity settings of even, odd, mark (always 1), space (always 0), and no parity
- Automatic parity error detection with selectable parity bias
- Automatic framing error detection
- Separate 8-byte deep transmit and receive FIFOs
- Flexible interrupt conditions
- Hardware flow control for RTS and CTS
- Null modem support
- Break generation and detection
- Wakeup from *DEEPSLEEP* on UART edge
- Receive timeout detection
- DMA capable

10.1 Instances

Two instances of the UART are provided: UART0 and UART1. Refer to the device data sheet for alternate function assignments for each of the UART instances.

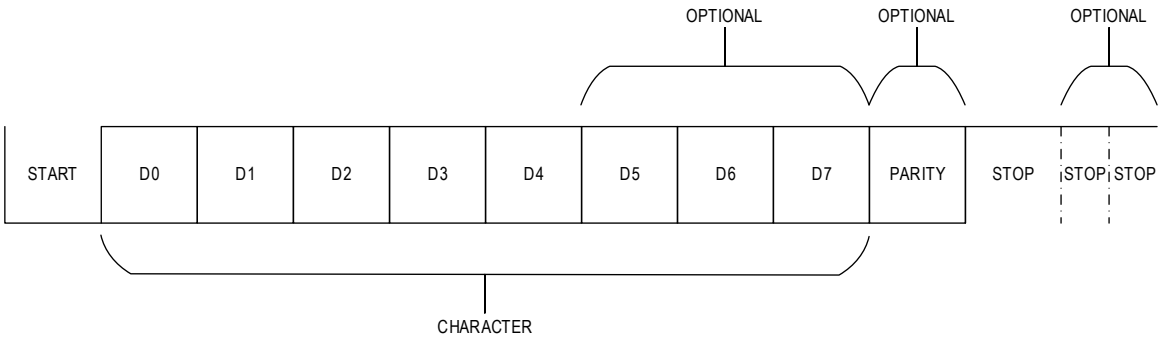
Each instance supports identical functionality and registers unless expressly noted otherwise. For simplicity, the UARTs are referenced in the documentation as UARTn where *n* = 0 or 1.

Table 10-1: MAX32660 UART Instances

Instance	Power Modes	Peripheral Clock	C_TX_FIFO_DEPTH C_RX_FIFO_DEPTH
UART0	*ACTIVE, Wakeup from SLEEP*	PCLK	8/8
UART1			

10.2 UART Frame

Figure 10-1: UART Frame Diagram



Character sizes of 5 to 8 bits are supported. The field *UARTn_CTRL.charsize* is used to select the character size.

Stop bit support includes 1, 1.5, and 2 stop bits selected with the register field *UARTn_CTRL*.

Parity support includes even, odd, mark, space, or none. For no parity, set field *UARTn_CTRL.parity_en* to 0. For all other parity options, select one of the four parity options using the *UARTn_CTRL.parity* field and enable parity (*UARTn_CTRL.parity_en*=1). Parity can be based on the number of 1 bits or 0 bits in the receive characters as set in the register bit *UARTn_CTRL.parmd*.

Break frames are transmitted by setting the field *UARTn_CTRL.break* to 1. A break sets all bits in the frame to 0.

When a break frame is received, two interrupts are available, *UARTn_INT_FL.break* is set to 1 when the first received break character is received, and *UARTn_INT_FL.last_break* is set when the last break character is received. This prevents the system from being overloaded with multiple interrupts that could occur after the first break character and up to the Nth break character received.

Note: A break character does not set the frame error flag because breaks are not valid UART characters.

10.3 UART Interrupts

Interrupts can be generated for the conditions in the following table:

Table 10-2: UART Interrupt Conditions

Interrupt	Condition
Transmit FIFO Level	The transmit FIFO level transitions from being greater than to equal to the set transmit threshold.
Receive FIFO Level	The receive FIFO level is equal to or greater than the set receive threshold.
Receive FIFO Overrun	The receive FIFO is full but is still receiving data.
CTS State Change during hardware flow control	CTS is deasserted, which tells the UART to pause transmitting data. CTS is asserted, which tells the UART to resume transmitting data.
Receive Parity	A receive parity error occurred.
Receive Frame	START or STOP bits were not detected.
Receive Timeout	No characters were received within the set timeout period.
Break	Beginning and end of break.

10.4 UART Bit Rate Calculation

The peripheral clock, f_{PCLK} , is used as the input clock to the UART bit rate generator. The following fields are used to set the target bit rate for the UART instance.

UARTn_BAUD0.factor: Selects the bit rate clock divisor.

UARTn_BAUD0.ibaud: Sets the integer portion of the bit rate divisor.

UARTn_BAUD1.dbaud: Sets the decimal portion of the bit rate divisor.

The following equations are used to determine the values for each of the bit rate fields required to achieve a target bit rate for the UART instance.

Equation 10-1: UART Bit Rate Divisor Equation

$$DIV = \frac{f_{PCLK}}{\left(2^{(7-UARTn\_BAUD0.factor)} \times Target\ Baud\ Rate\right)}$$ *where,*

$Target\ Baud\ Rate\ is\ the\ desired\ UART\ interface\ speed$

Note: UARTn_BAUD0.factor should be set to the lowest value that results in $\lfloor DIV \rfloor \geq 1$ to achieve the highest accuracy for the target bit rate. $\lfloor x \rfloor$ is a function that takes a real number x as an input and gives the greatest integer less than or equal to x as output.

Equation 10-2: Bit Rate Integer Calculation

$$UARTn\_BAUD0.ibaud = \lfloor DIV \rfloor$$

Equation 10-3: Bit Rate Remainder Calculation

$$y = \lfloor (DIV - UARTn\_BAUD.ibaud) \times 128 \rfloor$$

if (y > 3), $UART1\_BAUD1.dbaud = y - 3$

else, $UART1\_BAUD1.dbaud = y + 3$

Example Baud Rate Calculation:

$$Target\ Bit\ Rate = 1{,}843{,}200\ bits\ per\ second\ (1.8\ Mbps)$$

$$f_{PCLK} = 48\ MHz$$

$$DIV = \frac{48{,}000{,}000}{(2^{(7-UARTn\_BAUD1.factor)} \times 1{,}843{,}200)}$$

Table 10-3: Example Baud Rate Calculation Results, Target Bit Rate = 1.8Mbps

UARTn_BAUD0.factor	DIV	UARTn_BAUD0.ibaud	UARTn_BAUD1.dbaud
3	1.63	1	77
2	0.81	0 (Must be 1 or greater. The value selected for UARTn_BAUD0.factor is not valid)	
1	0.41	0 (Must be 1 or greater. The value selected for UARTn_BAUD0.factor is not valid)	
0	0.20	0 (Must be 1 or greater. The value selected for UARTn_BAUD0.factor is not valid)	

Table 10-3 shows the resulting DIV for each of the UARTn_BAUD0.*factor* field settings. With UARTn_BAUD0.*factor* set to 3, the resulting DIV value is greater than 1. Setting UARTn_BAUD0.*factor* to 3 generates the most accurate target bit rate because it is the smallest value that results in DIV ≥ 1. Using 3 for UARTn_BAUD0.*factor*, UARTn_BAUD0.*ibaud* is 1, which is the integer portion of the 1.63 DIV calculation. The UARTn_BAUD1.*dbaud* field calculation based on UARTn_BAUD0.*factor* = 3, UARTn_BAUD0.*ibaud* = 1, and DIV = 1.63 is:

$$UARTn\_BAUD1.dbaud = \lfloor (1.63 - 1) \times 128 \rfloor - 3$$

The resulting field settings for the example 1,843,200 bps rate are:

UARTn_BAUD0.*factor* = 3
UARTn_BAUD0.*ibaud* = 1
UARTn_BAUD1.*dbaud* = 77

10.5 UART DMA Using the Transmit and Receive FIFOs

Each UART has an 8-byte transmit FIFO with a dedicated DMA channel and an 8-byte receive FIFO with a dedicated DMA channel. The DMA channels are configured using the DMA configuration register, UARTn_DMA. The receive FIFO DMA channel and transmit FIFO DMA channels operate independently, and each can be enabled or disabled individually. Enable the receive FIFO DMA channel by setting UARTn_DMA.*rxdma_en* to 1 and enable the transmit FIFO DMA channel by setting

the *UARTn_DMA.txdma_en* to 1. DMA transfers are automatically triggered based on the number of bytes in the receive or transmit FIFO as described in the following two sections.

10.5.1 Receive FIFO DMA Operation

UARTn_DMA.rxdma_level configures the number of entries in the receive FIFO that triggers a DMA transfer from the receive FIFO to system RAM. If the number of entries in the receive FIFO is equal to or greater than the configured value, a DMA transfer is triggered from the receive FIFO to system RAM. If *UARTn_DMA.rxdma_level* = 1, then a transfer is triggered when there is one byte in the FIFO. A *UARTn_DMA.rxdma_level* = 0 is not allowed and results in erroneous operation.

Note: The receive DMA level must be set to a value less than 8 to avoid a receive FIFO overrun condition that results in the loss of received data.

10.5.2 Transmit FIFO DMA Operation

UARTn_DMA.txdma_level sets the number of entries (level) in the transmit FIFO that trigger a DMA transfer from system RAM to the transmit FIFO. If the number of entries (level) in the transmit FIFO falls below this value, a transmit DMA transfer is automatically triggered from system RAM to the transmit FIFO.

Note: Set the transmit DMA level (UARTn_DMA.txdma_level) greater than 1 to avoid stalling the UART transfer.

10.6 Flushing the UART FIFOs

The FIFOs can be flushed independently by setting *UARTn_CTRL.rx_flush* to 1 for the receive FIFO and *UARTn_CTRL.tx_flush* to 1 for the transmit FIFO. The transmit FIFO and receive FIFO are automatically flushed if the UART is disabled by clearing the *UARTn_CTRL.enable* field (*UARTn_CTRL.enable* = 0).

10.7 Hardware Flow Control

When hardware flow control is enabled, the CTS (Clear-to-send) and RTS (Request-to-Send) external signals are directly managed by hardware without CPU intervention. RTS and CTS are active when flow control is enabled by setting the register bit *UARTn_CTRL.flow_ctrl* = 1. The CTS/RTS signal polarity is configured with register bit *UARTn_CTRL.flow_pol* and can be active low or active high.

In operation, the host UART that wants to transmit data asserts its RTS output pin and waits for its CTS input pin to be asserted. If CTS is asserted, then the host UART begins transmitting data to the slave UART. If during the transmission the host UART notices CTS is deasserted, the host UART finishes transmitting the current character and then pauses to wait for CTS to return to an asserted level before transmitting more data.

If this UART is receiving data, and the receive FIFO reaches the level set in the 6-bit register field *UARTn_THRESH_CTRL.rts_fifo_thresh*, then the RTS signal of this UART is deasserted, informing the transmitting UART to stop sending data to this UART to prevent data overflow. Transmission resumes when the level of the receive FIFO drops below *UARTn_THRESH_CTRL.rts_fifo_thresh*, which automatically asserts RTS.

10.8 UART Registers

See *Table 3-1* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 10-4*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 10-4. UART Register Offsets, Names, Access and Descriptions

Offset	Register Name	Description
[0x0000]	*UARTn_CTRL*	Control Register
[0x0004]	*UARTn_THRESH_CTRL*	Threshold Control Register
[0x0008]	*UARTn_STATUS*	Status Register
[0x000C]	*UARTn_INT_EN*	Interrupt Enable Register
[0x0010]	*UARTn_INT_FL*	Interrupt Flag Register
[0x0014]	*UARTn_BAUD0*	Baud Rate Integer Register
[0x0018]	*UARTn_BAUD1*	Baud Rate Decimal Register
[0x001C]	*UARTn_FIFO*	FIFO Read/Write Register
[0x0020]	*UARTn_DMA*	DMA Configuration Register
[0x0024]	*UARTn_TX_FIFO*	Transmit FIFO Register

10.8.1 UART Register Details

Table 10-5: UART Control 0 Register

UART Control			UARTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23:16	rx_to	R/W	0	**Receive Timeout Frame Count** If the receive FIFO contains data, a receive timeout condition occurs if the time for the number of frames in this register passes without the FIFO receiving any new data. If a timeout occurs, the hardware sets the receive timeout flag to 1 (*UARTn_INT_FL*.*rx_timeout* = 1).
15	clksel	DNM	0	**Reserved, Do Not Modify**
14	break	R/W	0	**Transmit BREAK Frame** Set this field to 1 to send a BREAK frame. A BREAK frame transmits a character with all bits set to 0. 0: Normal UART operation. 1: Transmit BREAK frame.
13	null_modem		0	**Null Modem Support** 0: Normal operation for RTS/CTS and TX/RX 1: Null modem mode: RTS/CTS swapped, TX/RX swapped
12	flow_pol	R/W	0	**RTS/CTS Polarity** 0: RTS/CTS asserted is 0. 1: RTS/CTS asserted is 1.
11	flow_ctrl	R/W	0	**Hardware Flow Control Enable** 0: Hardware flow control disabled. 1: Hardware RTS/CTS flow control enabled.

UART Control			UARTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
10	stopbits	R/W	0	**STOP Bit Mode Select** 0: 1 STOP bit. 1: 1.5 STOP bits for 5-bit character size or 2 STOP bits for all other character sizes.
9:8	char_size	R/W	0	**Character Size** Set the number of data bits per frame. 0: 5 data bits 1: 6 data bits 2: 7 data bits 3: 8 data bits
7	bitacc	R/W	0	**Frame or Bit Accuracy Select** This field selects between either frame accuracy or bit accuracy for transmitting data. Frame Accuracy: Individual frame bit durations can be varied by hardware to meet the target frame period. Bit accuracy: Bit width is fixed by hardware. The frame accuracy of data transmitted can be reduced if bit accuracy is prioritized. 0: Frame accuracy. 1: Bit accuracy. *Note: A frame includes the start, stop, all data bits, and parity bit/bits for the character being transmitted.*
6	rx_flush	R/W1O	0	**Receive FIFO Flush** Write 1 to flush the receive FIFO. Cleared to 0 by hardware when flush is completed.
5	tx_flush	R/W1O	0	**Transmit FIFO Flush** Write 1 to flush the Transmit FIFO. Cleared to 0 by hardware when flush is completed.
4	parmd	R/W	0	**Parity Level Select** 0: Parity is based on the number of 0 bits in the character. 1: Parity is based on the number of 1 bits in the character.
3:2	parity	R/W	0	**Parity Mode Select** 0: Even parity 1: Odd Parity 2: Mark parity 3: Space parity
1	parity_en	R/W	0	**Parity Enable** 0: No parity 1: Parity enabled as charsize+1 bit
0	enable	R/W	0	**UART Enable** 0: UART disabled. FIFOs are flushed, bit rate generator is off. 1: UART enabled, bit rate generator is active.

Table 10-6: UART Threshold Control 1 Register

UART Threshold Control 1			UARTn_THRESH_CTRL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:22	0	RO	0	**Reserved**
21:16	rts_fifo_thresh	R/W	0	**RTS Receive FIFO Threshold Level** When the receive FIFO level is equal to or greater than this value, deassert the RTS output signal to inform the transmitting UART to stop sending data to this UART. Valid values are 1 to 7.

UART Threshold Control 1			UARTn_THRESH_CTRL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
15:14	-	RO	0	**Reserved**
13:8	tx_fifo_thresh	R/W	0	**Transmit FIFO Threshold Level** When the transmit FIFO level is less than or equal to this value, set *UARTn_INT_FL.tx_fifo_thresh* interrupt flag. Valid values are 1 to 7. Set this field greater than 1 to avoid a stall condition when transmitting UART data.
7:6	-	RO	0	**Reserved**
5:0	rx_fifo_thresh	R/W	0	**Receive FIFO Threshold Level** When the receive FIFO level is equal to or greater than this value, the hardware sets the *UARTn_INT_FL.rx_fifo_thresh* interrupt flag is set. Valid values are 1 to 7. Set this field to less than 7 to avoid a receive FIFO overrun condition.

Table 10-7: UART Status Register

UART Status			UARTn_STATUS	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:25	-	RO	0	**Reserved**
24	rx_to	RO	0	**Receive Timeout** This field is set to 1 when a receive timeout occurs. This field is set by hardware when the condition occurs and is automatically cleared when the condition is no longer valid.
23:22	-	RO	0	**Reserved**
21:16	tx_fifo_cnt	RO	0	**Number of Bytes in the Transmit FIFO** Read this field to determine the number of bytes in the transmit FIFO.
15:14	-	RO	0	**Reserved**
13:8	rx_fifo_cnt	RO	0	**Number of Byes in Receive FIFO** Read this field to determine the number of bytes in the receive FIFO.
7	tx_full	RO	0	**Transmit FIFO Full Status Flag** This field reads 1 when the transmit FIFO is full. This field is set by hardware when the condition occurs and is automatically cleared when the condition is no longer valid. 0: Transmit FIFO is not full. 1: Transmit FIFO is full.
6	tx_empty	RO	1	**Transmit FIFO Empty Flag** This field reads 1 when the transmit FIFO is empty. This field is set by hardware when the condition occurs and is automatically cleared when the condition is no longer valid. 0: Transmit FIFO is not empty (*UARTn_STATUS.tx_fifo_cnt* > 0). 1: Transmit FIFO is empty.
5	rx_full	RO	0	**Receive FIFO Full Flag** This field reads 1 when then receive FIFO is full. This field is set by hardware when the condition occurs and is automatically cleared when the condition is no longer valid. 0: Receive FIFO is not full. 1: Receive FIFO is full.
4	rx_empty	RO	1	**Receive FIFO Empty Flag** This flag reads 1 when the receive FIFO is empty.
3	break	RO	0	**Break Flag** This field is set when a break condition occurs. 0: BREAK not received. 1: BREAK condition received.

UART Status			UARTn_STATUS	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
2	parity	RO	0	**Parity Bit State** This field returns the state of the parity bit in the most recently received character. 0: Parity bit is 0. 1: Parity bit is 1.
1	rx_busy	RO	0	**Receive Busy** This field reads 1 when the UART is receiving data. 0: UART is not actively receiving data. 1: UART is actively receiving data.
0	tx_busy	RO	0	**Transmit Busy** This field reads 1 when the UART is transmitting data. 0: UART is not actively transmitting data. 1: UART is transmitting data.

Table 10-8: UART Interrupt Enable Register

UART Interrupt Enable			UARTn_INT_EN	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	last_break	R/W	0	**Last Break Interrupt Enable** When the UART receives a series of BREAK frames, this enables an interrupt when the last BREAK frame is received.
8	rx_timeout	R/W	0	**Receive Timeout Interrupt Enable** Enable the receive timeout interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
7	break	R/W	0	**Received BREAK Interrupt Enable** This field enables the BREAK interrupt for the first BREAK received on the UART. 0: Interrupt disabled. 1: Interrupt enabled.
6	tx_fifo_thresh	R/W	0	**Transmit FIFO Threshold Level Interrupt Enable** Enables the transmit FIFO threshold interrupt when the number of entries in the transmit FIFO is equal or less than the value set in *UARTn_THRESH_CTRL.tx_fifo_thresh*. 0: Interrupt disabled. 1: Interrupt enabled.
5	tx_fifo_almost_empty	R/W	0	**Transmit FIFO One Byte Remaining Interrupt Enable** 0: Interrupt disabled 1: Interrupt enabled
4	rx_fifo_thresh	R/W	0	**Receive FIFO Threshold Level Interrupt Enable** Enables an interrupt when the number of entries in the receive FIFO is greater than or equal to *UARTn_THRESH_CTRL.rx_fifo_thresh*. 0: Interrupt disabled. 1: Interrupt enabled.
3	rx_overrun	R/W	0	**Receive FIFO Overrun Interrupt Enable** Enables an interrupt when a write is made to a full receive FIFO. 0: Interrupt disabled. 1: Interrupt enabled.

UART Interrupt Enable			UARTn_INT_EN	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
2	cts_change	R/W	0	**CTS State Change Interrupt Enable** This field enables the CTS level change interrupt event, often referred to as modem status interrupt. 0: Interrupt disabled. 1: Interrupt enabled.
1	rx_parity_error	R/W	0	**Receive Parity Error Interrupt Enable** 0: Interrupt disabled. 1: Interrupt enabled.
0	rx_frame_error	R/W	0	**Receive Frame Error Interrupt Enable** 0: Interrupt disabled. 1: Interrupt enabled.

Table 10-9: UART Interrupt Flags Register

UART Interrupt Flags			UARTn_INT_FL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	last_break	R/W1C	0	**Last Break Interrupt Flag** When the UART receives a series of BREAK frames, this flag is set when the last BREAK frame is received. Write 1 to clear this field. 0: Last BREAK condition has not occurred. 1: Last BREAK condition has occurred.
8	rx_timeout	R/W1C	0	**Receive Frame Timeout Interrupt Flag** This field is set when a receive frame timeout occurs. Write 1 to clear this field. 0: Receive frame timeout has not occurred. 1: The UART detected a receive frame timeout.
7	break	R/W1C	0	**Received Break Interrupt Flag** When the UART receives a series of BREAK frames, this flag is set when the first BREAK frame is received. Write 1 to clear this field. 0: Break condition not occurred. 1: Break condition occurred.
6	tx_fifo_thresh	R/W1C	0	**Transmit FIFO Threshold Interrupt Flag** Set when the number of entries in the transmit FIFO is less than or equal to the transmit FIFO level set in *UARTn_THRESH_CTRL.tx_fifo_thresh*. Write 1 to clear. 0: The transmit FIFO level is below the threshold set in *UARTn_THRESH_CTRL.tx_fifo_thresh*. 1: The transmit FIFO level is equal to or greater than the *UARTn_THRESH_CTRL.tx_fifo_thresh*.
5	tx_fifo_almost_empty	R/W1C	0	**Transmit FIFO Almost Empty Interrupt Flag** This field is set when there is one byte remaining in the Transmit FIFO. Write 1 to clear. 0: Transmit FIFO level is greater than 1. 1: Transmit FIFO is Almost Empty.

UART Interrupt Flags		UARTn_INT_FL		[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
4	rx_fifo_thresh	R/W1C	0	**Receive FIFO Threshold Interrupt Flag** Set when the number of entries in the receive FIFO is equal to or greater than the receive FIFO threshold level as set in the *UARTn_THRESH_CTRL.rx_fifo_thresh* field. Data must be read from the receive FIFO to reduce the level below the threshold to guarantee this interrupt does not occur again after clearing the flag. Write 1 to clear this field. 0: The number of bytes in the receive FIFO is below the threshold level. 1: The number of bytes in the receive FIFO is equal to or greater than the threshold level.
3	rx_overrun	R/W1C	0	**Receive FIFO Overrun Interrupt Flag** This field is set if the receive FIFO is full and an additional byte is received, resulting in a FIFO overrun condition. If this field is set, at least one byte of received data has been lost. Write 1 to clear. 0: Receive FIFO overrun has not occurred. 1: Receive FIFO overrun occurred.
2	cts_change	R/W1C	0	**CTS Interrupt Flag** Also called Modem Status Interrupt
1	parity_error	R/W1C	0	**Receive Parity Error Status Flag** Set if a parity error is detected. This flag applies to data received only. Write 1 to clear. 0: Parity error has not been detected. 1: Parity error detected.
0	frame_error	R/W1C	0	**Frame Error Status Flag** Set if a frame error occurs while receiving data. Write 1 to clear. 0: No error. 1: Frame error occurred.

Table 10-10: UART Rate Integer Register

UART Baud Rate Integer		UARTn_BAUD0		[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:18	-	RO	0	**Reserved**
17:16	factor	R/W	0	**Bit Rate Clock Divisor** This field is used to divide the bit rate clock by the selected clock divider value. *factor* / Clock Divider Value: 0 / 128; 1 / 64; 2 / 32; 3 / 16 *Note: See the UART Bit Rate Calculation section for details on determining this field's value for a given UART bit rate.*
15:12	-	RO	0	**Reserved**
11:0	ibaud	R/W	0	**Integer Portion of Baud Rate Divisor** This field contains the integer value of the bit rate divisor. See the *UART Bit Rate Calculation* section for details of determining this field's value for a given UART bit rate.

Table 10-11: UART Baud Rate Decimal Register

UART Baud Rate Decimal			UARTn_BAUD1	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6:0	dbaud	R/W	0	**Decimal Portion of Baud Rate Divisor** This field contains the remainder portion of the bit rate divisor. See the *UART Bit Rate Calculation* section for details of determining this field's value for a given UART bit rate.

Table 10-12: UART FIFO Register

UART FIFO			UARTn_FIFO	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	fifo	R/W	N/A	**UART FIFO Register** Reading this field reads data from the receive FIFO and writes to this field write to the transmit FIFO.

Table 10-13: UART DMA Configuration Register

UART DMA Configuration			UARTn_DMA	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:22	-	RO	0	**Reserved**
21:16	rxdma_level	R/W	0	**Receive FIFO Level DMA Trigger** If the receive FIFO level is equal to or greater than this value, the DMA channel transfers data from the receive FIFO into memory. DMA transfers continue until the receive FIFO is empty. To avoid a receive FIFO overrun, do not set this value to 7. Values above 7 are reserved. Do not set this value to 0. A value of 0 causes erroneous operation.
15:14	-	RO	0	**Reserved**
13:8	txdma_level	R/W	0	**Transmit FIFO Level DMA Trigger** If the transmit FIFO level is less than this value, the DMA channel transfers data from memory into the transmit FIFO. DMA transfers continue until the transmit FIFO is full. To avoid stalling a UART transmission, do not set this value to 1 or 0. *Note: Values above 7 are Reserved.*
7:2	-	RO	0	**Reserved**
1	rxdma_en	R/W	0	**Receive FIFO DMA Channel Enable** 0: Receive DMA is disabled 1: Receive DMA is enabled
0	txdma_en	R/W	0	**Transmit FIFO DMA Channel Enable** 0: Transmit DMA is disabled 1: Transmit DMA is enabled

Table 10-14: UART Transmit FIFO Data Output Register

UART Transmit FIFO Data Output			UARTn_TX_FIFO	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**

UART Transmit FIFO Data Output			UARTn_TX_FIFO	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
7:0	data	RO	0	**Transmit FIFO Data Output Peek Register** Reads from this register return the next character available for transmission at the end of the transmit FIFO. If no data is available, reads of this field return 0. Reads from this register do not affect the transmit FIFO state. 0: No data available in the transmit FIFO 1-15: Number of bytes in the transmit FIFO.

11. Real-Time Clock (RTC)

11.1 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.

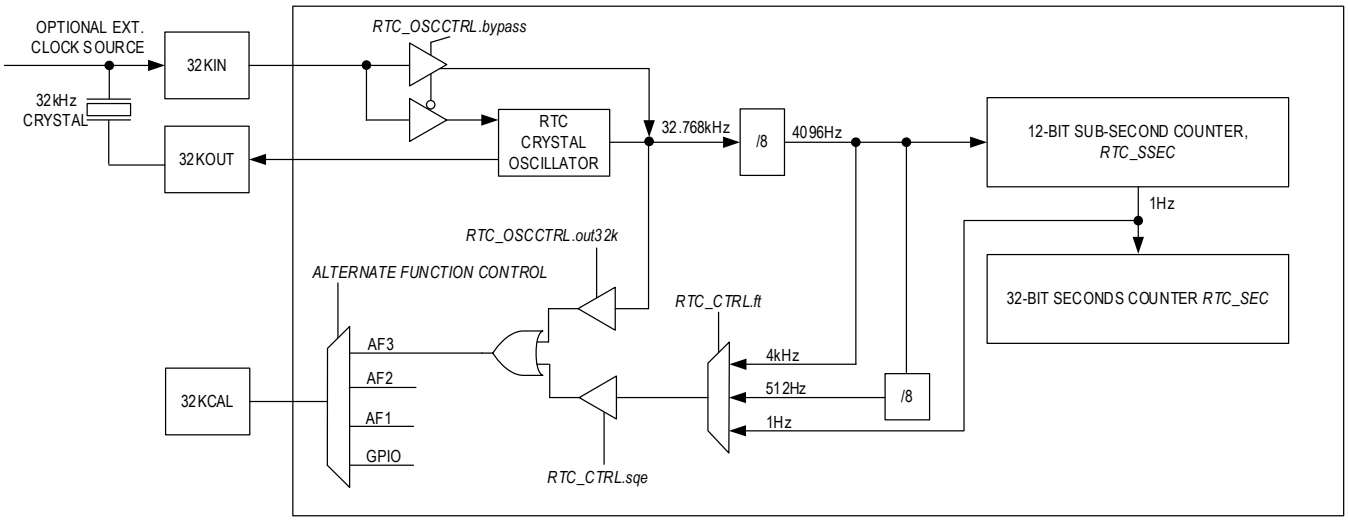
The 32-bit seconds counter register, *RTC_SEC*, is incremented on every rollover of the sub-seconds counter, *RTC_SSEC.rtss*, field.

Two alarm functions are provided:

- A programmable time-of-day alarm provides a single event, alarm timer using the *RTC_RAS* alarm register, *RTC_SEC* register, and the *RTC_CTRL.ade* field.
- A programmable sub-second alarm provides a recurring alarm using the *RTC_RSSA* and *RTC_CTRL.ase* field.

The RTC is powered in the AoD. Disabling the RTC stops incrementing the *RTC_SSEC* and *RTC_SEC* counters, but preserves their current values. The 32kHz oscillator is not affected by the *RTC_CTRL.rtce_en* field. The RTC increments the *RTC_TRIM.vbattmr* field every 32 seconds while the RTC is enabled.

Figure 11-1: MAX32660 RTC Block Diagram



11.2 Instances

One instance of the RTC peripheral is provided. The RTC counter and alarm registers are shown in *Table 11-1*.

Table 11-1: MAX32660 RTC Counter and Alarm Registers

Field	Length	Counter Increment	Minimum	Maximum	Description
RTC_SEC.sec	32	1s	1s	136 yrs	Seconds counter field
RTC_SSEC.rtss	12	244µs (1/4096Hz)	244µs	1s	Sub-seconds counter field
RTC_RAS.ras	20	1s	1s	12 days	Time-of-day alarm field
RTC_RSSA.rssa	32	244µs (1/4096Hz)	244µs	12 days	Sub-seconds alarm field

11.3 Register Access Control

Access protection mechanisms prevent the software from accessing critical registers and fields while the RTC 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 reads from registers can return valid results.

Table 11-2: RTC Register Access

Register	Field	Read Access	Write Access	*RTC_CTRL.busy* = 1 during write	Description
RTC_SEC	*.sec*	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1	Y	Seconds counter register
RTC_SSEC	*.rtss*	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1	Y	Sub-second counter field
RTC_RAS	*.ras*	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.ade* = 0	Y	Time-of-day alarm field
RTC_RSSA	*.rssa*	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.ase* = 0	Y	Sub-second alarm field
RTC_TRIM	All	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.we* = 1	Y	Trim register
RTC_OSCCTRL	All	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.we* = 1	Y	Oscillator control register
RTC_CTRL	*.rtce_en*	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.we* = 1	Y	RTC enable field
	.ade	Always	*RTC_CTRL.busy* = 0	Y	Time-of-day alarm enable field
	.ase	Always	*RTC_CTRL.busy* = 0	Y	Sub-second alarm enable field
	All other bits	Always	*RTC_CTRL.busy* = 0	Y	See *RTC_CTRL.busy* for limitations on specific bits.

11.3.1 RTC_SEC and RTC_SSEC Read Access Control

Software reads of the *RTC_SEC.sec* and *RTC_SSEC.rtss* fields return invalid results if the read operation occurs on the same cycle that the register is being updated by hardware. To avoid this, the hardware sets *RTC_CTRL.rdy* to 1 for 120µs when the *RTC_SEC.sec* and *RTC_SSEC.rtss* fields are valid and can be read.

The software can use two methods to ensure valid results when reading the *RTC_SEC.sec* and *RTC_SSEC.rtss* fields:

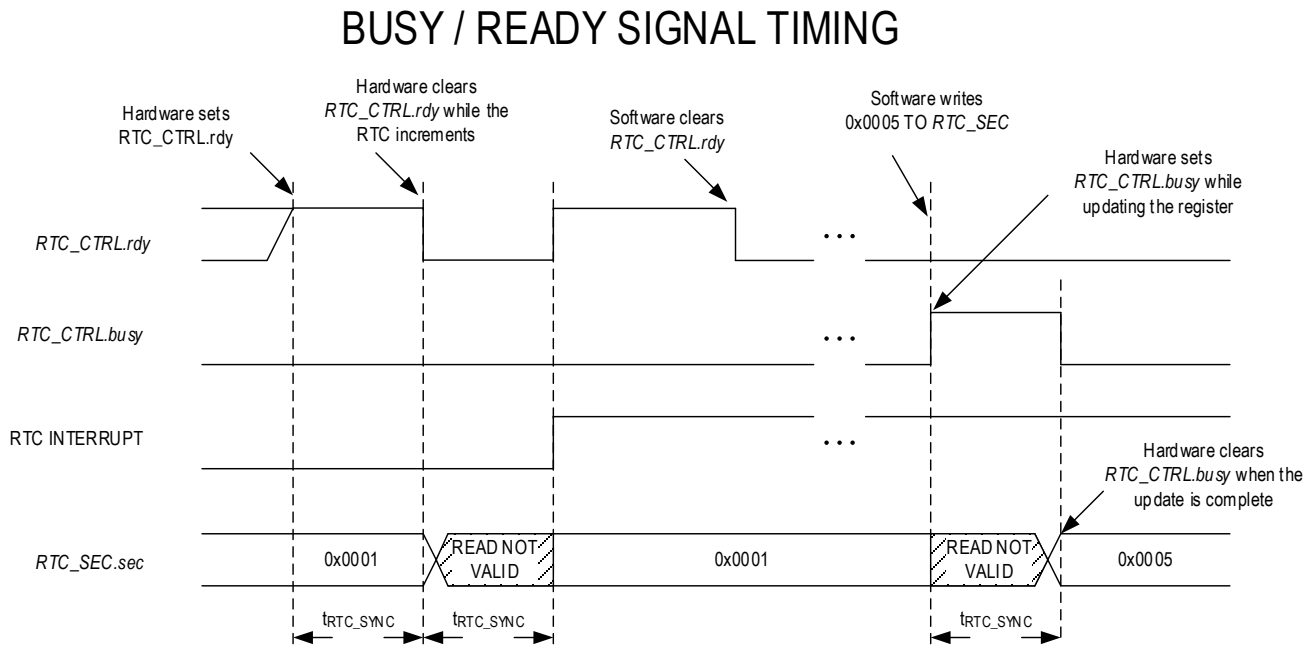
- The software clears the *RTC_CTRL.rdy* field to 0. The software then polls the *RTC_CTRL.rdy* field until it reads 1 before reading the fields. The software has approximately 120µs to read the *RTC_SEC.sec* and *RTC_SSEC.rtss* fields to ensure accuracy.
- Set the *RTC_CTRL.rdye* field to 1 to generate an RTC interrupt when the next update cycle is complete and hardware sets *RTC_CTRL.rdy* to 1. The RTC interrupt must be serviced, and the *RTC_SEC.sec* and *RTC_SSEC.rtss* fields must be read while *RTC_CTRL.rdy* = 1, within 120µs to ensure valid reads. This avoids the software overhead associated with polling to observe the state of *RTC_CTRL.rdy*.

The software can clear *RTC_CTRL.rdy* following the read of the registers to acknowledge the read.

11.3.2 RTC Write Access Control

The read-only status field *RTC_CTRL.busy* is set to 1 by hardware following a software instruction that writes to specific registers. The bit remains 1 while the software updates are being synchronized into the RTC. The software should not write to any of the registers until hardware indicates the synchronization is complete by clearing *RTC_CTRL.busy* to 0.

Figure 11-2: Busy/Ready Signal Timing Example for the RTC_SEC.sec Field



11.4 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 that is driven by the trimmed RTC clock source. See *Calibration* for details on trimming the RTC clock.

11.4.1 Time-of-Day Alarm

Program the RTC time-of-day alarm field, *RTC_RAS.ras*, to configure the time-of-day alarm. The alarm triggers when the value stored in *RTC_RAS.ras* matches the lower 20 bits of the *RTC_SEC* seconds count register. This allows programming the time-of-day alarm to any future value between 1 second and 12 days relative to the current time with a resolution of 1 second. Disable the time-of-day alarm by clearing the *RTC_CTRL.ade* field to 0 before changing the *RTC_RAS.ras* field.

When the alarm occurs, a single event sets the time-of-day alarm interrupt flag, *RTC_CTRL.aldf*, to 1.

Setting the *RTC_CTRL.aldf* bit to 1 in software results in an interrupt request to the processor if the alarm time-of-day interrupt enable, *RTC_CTRL.ade*, is set to 1.

11.4.2 Sub-Second Alarm

The *RTC_RSSA.rssa* and *RTC_CTRL.ase* fields control the sub-second alarm. Writing *RTC_RSSA.rssa* sets the starting value for the sub-second alarm counter. Writing the sub-second alarm enable, *RTC_CTRL.ase*, bit to 1 enables the sub-second alarm. Once enabled, an internal alarm counter begins incrementing from the *RTC_RSSA.rssa* value. When the counter rolls over from 0xFFFF FFFF to 0x0000 0000, hardware sets the sub-second alarm flag, *RTC_CTRL.alsf*, triggering the alarm. At the same time, the hardware also reloads the counter with the value previously written to *RTC_RSSA.rssa*.

The software must disable the sub-second alarm, *RTC_CTRL.ase*, before changing the sub-second alarm value, *RTC_RSSA.rssa*.

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. Thereafter, if the interval alarm remains enabled, the alarm triggers after each sub-second interval as defined without the first alarm uncertainty because the sub-second alarm is an

auto-reload timer. Enabling the sub-second alarm with the sub-second alarm set to 0 (*RTC_RSSA.rssa* = 0) results in the maximum sub-second alarm interval.

11.4.3 RTC Interrupt and Wakeup Configuration

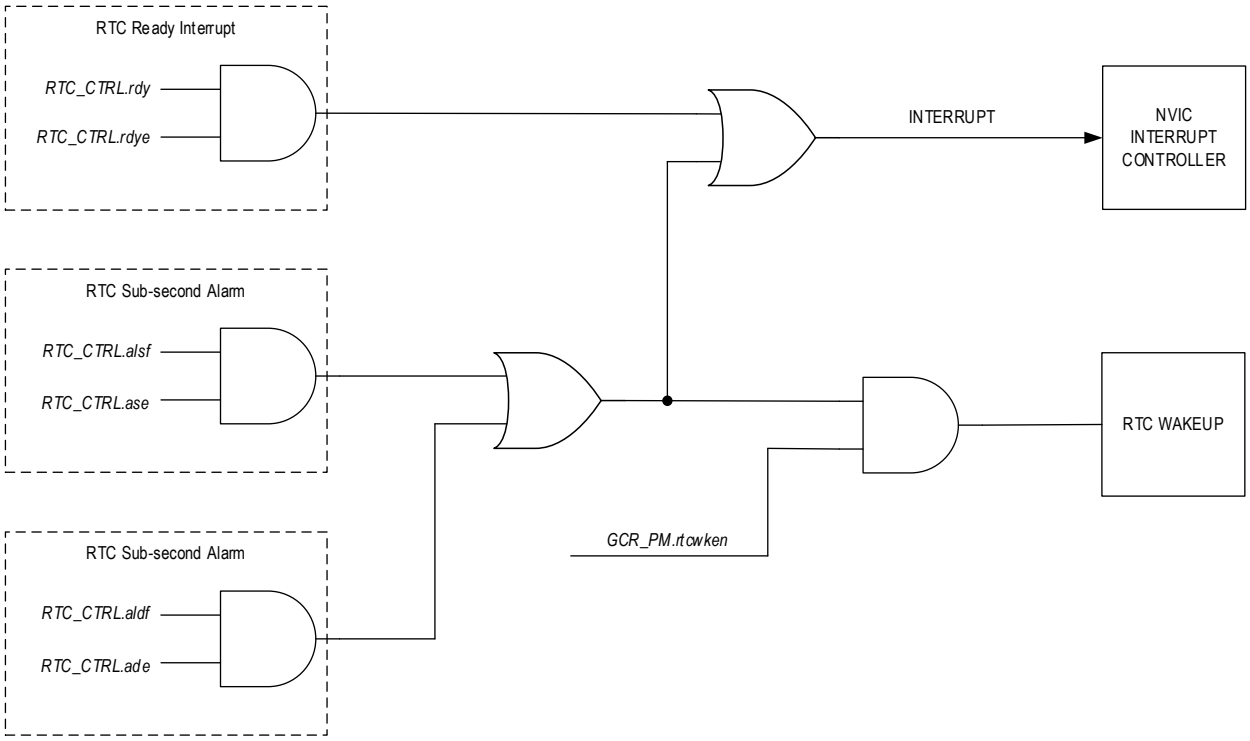
The following are a list of conditions that, when enabled, can generate an RTC interrupt:

- Time-of-day alarm
- Sub-second alarm
- *RTC_CTRL.rdy* field asserted high, signaling write access permitted

The time-of-day and sub-second alarms are configurable as a wakeup source for exiting the following low power modes:

- *SLEEP*
- *DEEPSLEEP*
- *BACKUP*

Figure 11-3: RTC Interrupt/Wakeup Diagram



Use this procedure to enable the RTC as a wakeup 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, RTC_IRQn, and register its address using the NVIC. See *Interrupts and Exceptions* for details on the RTC interrupt.
3. Set the *GCR_PM.rtcwken* field to 1 to enable system wakeup from RTC interrupts.
4. Enter the desired low-power mode. See the *Operating Modes* section for details on entering *SLEEP*, *DEEPSLEEP*, and *BACKUP*.

11.5 Square Wave Output

The RTC can output a 50% duty-cycle square wave signal derived from the 32kHz oscillator on a selected device pin. Refer to the device data sheet Alternate Function table for details on the 32KOUT pin. See *Table 11-3* for the device pins, frequency options, and control fields specific to this device. Frequencies noted as compensated are used during the RTC frequency calibration procedure because they incorporate the digital trim function's frequency adjustments.

Table 11-3: MAX32660 RTC Square Wave Output Configuration

Function	Option	Control Field
Output Pin	P0.2: 32KCAL	Configure GPIO0[2] for AF3
Output Enable	1024Hz (Compensated)	*RTC_CTRL.ft* = 0b00 *RTC_CTRL.sqe* = 1 *RTC_OSCCTRL.out32k* = 0
	512Hz (Compensated)	*RTC_CTRL.ft* = 0b01 *RTC_CTRL.sqe* = 1 *RTC_OSCCTRL.out32k* = 0
	4.096kHz	*RTC_CTRL.ft* = 0b10 *RTC_CTRL.sqe* = 1 *RTC_OSCCTRL.out32k* = 0
	32kHz	*RTC_CTRL.ft* = 0b11 *RTC_CTRL.sqe* = 1 *RTC_OSCCTRL.out32k* = 1

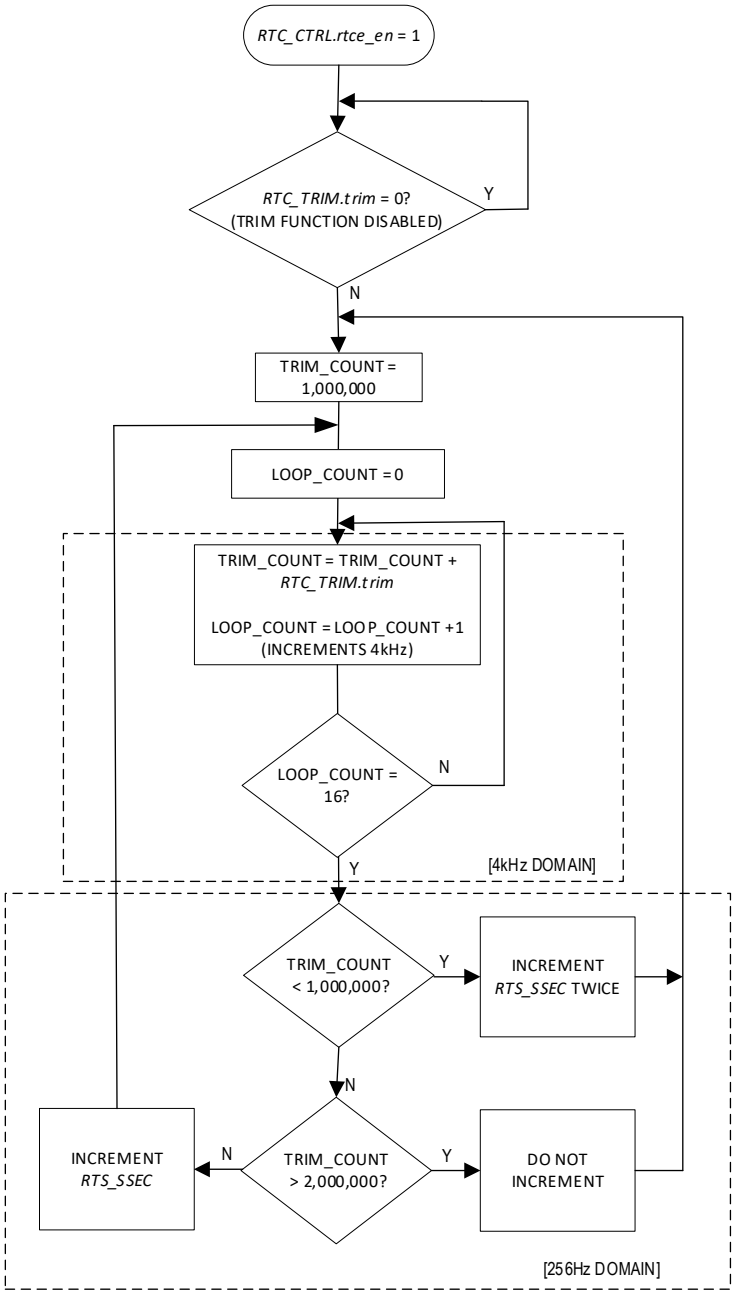
Use the following procedure to generate the square wave:

1. Configure the 32KCAL output for AF3 on GPIO0[2], see *General-Purpose I/O and Alternate Function Pins* for details.
2. Set the frequency output select field, *RTC_CTRL.ft*, for the desired frequency:
   a. Output 1024Hz by setting *RTC_CTRL.ft* to 0b00.
   b. Output 512Hz by setting *RTC_CTRL.ft* to 0b01.
   c. Output 4.096kHz by setting *RTC_CTRL.ft* to 0b10.
   d. Output 32kHz by setting *RTC_CTRL.ft* to 0b11.
3. Enable the output by configuring the *RTC_CTRL.sqe* and *RTC_OSCCTRL.out32k* fields:
   a. For 32kHz output, set *RTC_CTRL.sqe* to 1 and set *RTC_OSCCTRL.out32k* to 1.
   b. For all other frequencies, set *RTC_CTRL.sqe* to 1 and *RTC_OSCCTRL.out32k* to 0.

11.6 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 2's complement value in the *RTC_TRIM.trim* field, as shown in *Figure 11-4*.

Figure 11-4: Internal Implementation of Digital Trim



Complete the following steps to perform an RTC calibration:

1. If not already configured, the software should 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 *RTC_CTRL.rdy* = 1 by either:
   a. Setting *RTC_CTRL.rdye* to 1 to generate an interrupt when *RTC_CTRL.rdy* is set to 1, or
   b. Polling *RTC_CTRL.rdy* until the field reads 1
5. Poll until the *RTC_CTRL.busy* field reads 0 to make sure any previous operations are complete.
6. Set *RTC_CTRL.we* to 1 to allow access to *RTC_TRIM*.
7. Write to the *RTC_TRIM.trim* field as desired to correct for the measured inaccuracy.
8. Poll the *RTC_CTRL*.busy field until it reads 0.
9. Clear the *RTC_CTRL.we* to 0.

Repeat the process as needed until the desired accuracy is achieved.

11.7 RTC Registers

See *Table 3-1* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 11-4*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 11-4: RTC Register Summary

Offset	Register	Description
[0x0000]	*RTC_SEC*	RTC Seconds Counter Register
[0x0004]	*RTC_SSEC*	RTC Sub-Second Counter Register
[0x0008]	*RTC_RAS*	RTC Time-of-Day Alarm Register
[0x000C]	*RTC_RSSA*	RTC Sub-Second Alarm Register
[0x0010]	*RTC_CTRL*	RTC Control Register
[0x0014]	*RTC_TRIM*	RTC Trim Register
[0x0018]	*RTC_OSCCTRL*	RTC 32kHz Oscillator Control Register

11.7.1 Register Details

Table 11-5: 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 11-6: RTC Sub-Second Counter Register

RTC Sub-Seconds Counter			RTC_SSEC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	-	RO	0	**Sec-Seconds Counter (Bits 11:8)** This field is not readable by software.
7:0	rtss	R/W	0	**Sub-Seconds Counter (Bits 7:0)** The lower 8 bits of this counter are located in this 8-bit field. The upper four bits of the sub-seconds counter, bits 11:8, are not readable by software. The *RTC_SEC* register increments when the 12-bit sub-second counter field rolls from 0xFFF to 0.

Table 11-7: RTC Time-of-Day Alarm Register

RTC Time-of-Day Alarm			RTC_RAS	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:0	ras	R/W	0	**Time-of-Day Alarm** This field sets the time-of-day alarm from 1 second up to 12 days. When this field matches *RTC_SEC*[19:0], an RTC system interrupt is generated.

Table 11-8: RTC Sub-Second Alarm Register

RTC Sub-Second Alarm			RTC_RSSA	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	rssa	R/W	0	**Sub-second Alarm** Sets the starting and reload value of the internal sub-second alarm counter. The internal counter increments and generates an alarm when the internal counter rolls from 0xFFFF FFFF to 0x0000 0000.

Table 11-9: RTC Control Register

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	we	R/W	0*	**Write Enable** This field controls access to the RTC enable (*RTC_CTRL.rtce_en*) field. 0: Writes to the *RTC_CTRL.rtce_en* field are ignored. 1: Writes to the *RTC_CTRL.rtce_en* field are allowed. **Note: Reset on system reset, soft reset, and the peripheral specific reset (GCR_RSTR0.rtc =1).*
14:13	-	RO	0	**Reserved**
12:11	x32kmd	R/W	0	**32KHz Oscillator Mode** 0: Noise immune mode 1: Quiet mode 2: Quiet mode in low-power modes with warmup. 3: Quiet mode in low-power modes with no warmup.
10:9	ft	R/W	0	**Frequency Output Select** This field selects the RTC-derived frequency to output on the square wave output pin. See *Square Wave Output* for configuration details. 0b00: 1.024Hz (Compensated) 0b01: 512Hz (Compensated) 0b1x: 4.096kHz
8	sqe	R/W	0	**Square Wave Output Enable** This field enables the square wave output. See *Square Wave Output* for configuration details. 0: Disabled 1: Enabled
7	alsf	R/W	0	**Sub-second Alarm Interrupt Flag** This interrupt flag is set when a sub-second alarm condition occurs. This flag is a wakeup source for the processor. 0: No sub-second alarm interrupt pending. 1: Sub-second alarm interrupt pending.
6	aldf	R/W	0	**Time-of-Day Alarm Interrupt Flag** This interrupt flag is set by hardware when a time-of-day alarm occurs. 0: No Time-of-Day alarm interrupt pending. 1: Time-of-day interrupt pending.
5	rdye	R/W	0*	**RTC Ready Interrupt Enable** 0: Disabled 1: Enabled **Note: Reset on system reset, soft reset, and the peripheral specific reset (GCR_RSTR0.rtc =1).*

RTC Control Register			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
4	rdy	R/WO	0*	**RTC Ready** This bit is set to 1 for 120µs by hardware once a hardware update of the *RTC_SEC* and *RTC_SSEC* registers. 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.rdye* = 1. 0: Software reads of *RTC_SEC* and *RTC_SSEC* are invalid. 1: Software reads of *RTC_SEC* and *RTC_SSEC* are valid. **Note: Reset on System Reset, Soft Reset, and the peripheral specific reset (GCR_RSTR0.rtc =1).*
3	busy	RO	0*	**RTC Busy Flag** This bit is set to 1 by hardware to indicate a register update is in progress when the software writes to: • *RTC_SEC* • *RTC_SSEC* • *RTC_CTRL.rtce_en* • *RTC_CTRL.ade* • *RTC_CTRL.ase* The field is automatically cleared by hardware when the update is complete. The software should poll this field for 0 after changing RTC registers or fields listed above before making any other RTC register changes. 0: Normal operation. 1: RTC busy. **Note: Reset on POR only.*
2	ase	R/W	0*	**Sub-Second Alarm Interrupt Enable** Check the *RTC_CTRL*.busy flag after writing to this field to determine when the RTC synchronization is complete. 0: Disabled 1: Enabled **Note: Reset on POR only.*
1	ade	R/W	0*	**Time-of-Day Alarm Interrupt Enable** Check the *RTC_CTRL*.busy flag after writing to this field to determine when the RTC synchronization is complete. 0: Disabled 1: Enabled **Note: Reset on POR only.*
0	rtce	R/W	0*	**Real-Time Clock Enable** This field enables the RTC. The RTC write enable (*RTC_CTRL.we*) bit must be set to 1, and the RTC busy (*RTC_CTRL.busy*) field 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 11-10: RTC Trim Register

RTC Trim			RTC_TRIM	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	vbattmr	R/W	0	**VRTC Time Counter** The hardware automatically increments this field every 32 seconds while the RTC is enabled. *Note: This field is only reset on a POR.*
7:0	trim			**RTC Trim** This field specifies the 2's complement value of the trim resolution. Each increment or decrement of this field adds or subtracts 1ppm on each 4.096kHz clock value with a maximum correction of ± 127ppm.

Table 11-11: RTC 32kHz Oscillator Control Register

RTC 32kHz Oscillator Control			RTC_OSCCTRL	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	R/W	0	**Reserved**
5	out32k	R/W	0	**RTC Square Wave Output** 0: Disabled 1: Enables the 32kHz oscillator output or the external clock source output on the square wave output pin. See *Square Wave Output* for configuration details. **Note: Reset on POR only.*
4	bypass	R/W	0	**RTC Crystal Bypass** This field disables the RTC oscillator and allows an external clock source to be driven on the 32KIN pin. 0: Disable bypass. RTC time base is an external 32kHz crystal. 1: Enable bypass. RTC time base is an external square wave (driven on the 32KIN input pin). **Note: Reset on POR only.*
3	ibias_en	RO	1	**Reserved**
2	hyst_en	RO	0	**Reserved**
1	ibias_sel	RO	0	**Reserved**
0	filter_en	RO	1	**Reserved**

12. Timers (TMR)

Multiple 32-bit, reloadable timers are provided. Each timer provides 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.
- Pulse width modulated (PWM) / PWM differential.
- Capture: The timer captures a snapshot of the current timer count when the timer input edge transitions.
- Compare: The timer pin toggles when the timer exceeds the terminal count.
- Gated: The timer increments only when the timer input pin is asserted.
- Capture/Compare: The timer counts when the timer input pin is asserted and captures the timer count when input is deasserted.

The MAX32660 includes three instances of the timer peripheral (TMR0, TMR1, and TMR2).

12.1 Features

- 32-bit reload counter.
- Programmable prescaler with values from 1 to 4096.
- Capture, compare, and capture/compare capability.
- Timer pin available as alternate function on GPIO.
- Configurable input pin for event triggering, clock gating, or capture signal.
- Timer output pin for event output and PWM signal generation.
- Independent interrupt for each timer instance.

12.2 Basic Operation

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

Each timer mode has a user-configurable timer period, terminating 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, including changing the timer pin's state, 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. The first timer period following a system reset is one timer clock longer than subsequent timer periods if the *TMRn_CNT* is not initialized to 0x0000 0001 during the timer configuration step.

The timer clock frequency drives clocking of the timer functions, f_{CNT_CLK}. The timer clock frequency is a user-configurable division of the system peripheral clock, PCLK. Each timer has an independent prescaler, allowing timers to operate at different frequencies. The prescaler can be set from 1 to 4096 using the *TMRn_CN.pres3*:*TMRn_CN.pres* fields. Unless otherwise mentioned, the timer clock is generated as follows:

Equation 12-1: Timer Peripheral Clock Equation

$$f_{CNT\_CLK} = \frac{f_{PCLK}}{prescaler}$$

External events on the timer pins are asynchronous events to the slower timer clock frequency. These events are latched on the next rising edge of the timer clock. Since it is impossible to directly observe the timer clock directly, input events can have up to a 1 timer clock delay before being recognized.

12.3 Timer Pin Functionality

Most timers have an associated timer pin that can function as an optional input or output depending on the selected timer mode. The timer pin functionality is mapped as an alternate function that is shared with a GPIO. Timer pin assignments are detailed in the device data sheet.

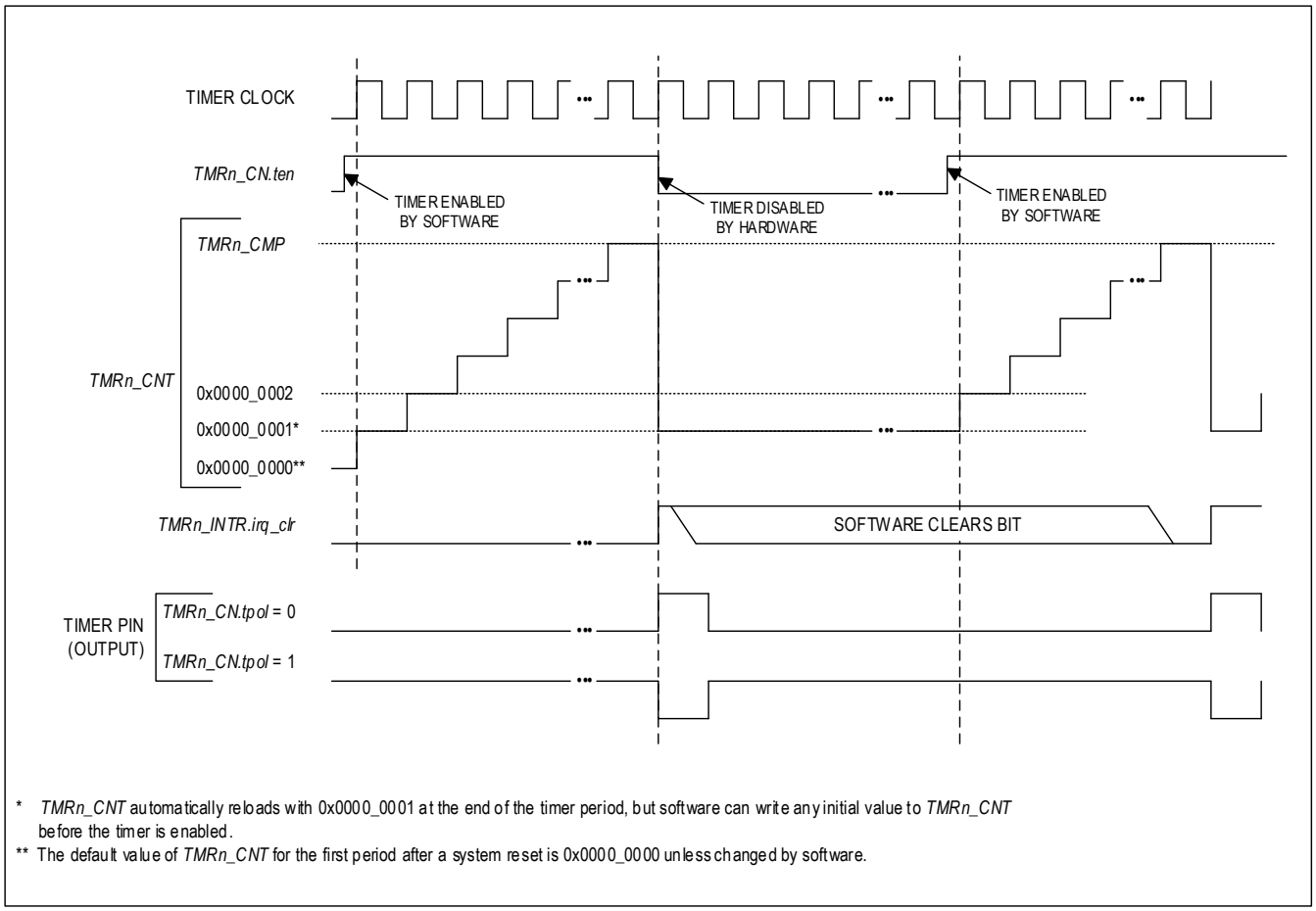
When the timer pin alternate function is enabled, the timer pin has the same electrical characteristics, such as pullup/pulldown strength and drive strength, as the GPIO mode settings for that pin. When configured as an output, the corresponding bit in the *GPIOn_OUT* register should be configured to match the timer pin's inactive state for that mode. The pin characteristics must be configured before enabling the timer. See *General-Purpose I/O and Alternate Function Pins* for details on how to configure the electrical characteristics for the pin.

Each timer has a dedicated interrupt flag, *TMRn_INTR.irq*, set at the end of a timer period. If enabled, an interrupt is generated. The interrupt flag is cleared by writing any value to *TMRn_INTR.irq*.

12.4 One-Shot Mode (0)

In One-shot mode, the timer peripheral increments *TMRn_CNT* until it matches *TMRn_CMP* and then stops incrementing and disables the timer. The timer can optionally output a pulse on the timer pin at the end of the timer period. In this mode, the timer must be re-enabled to start another one-shot mode event.

Figure 12-1: One-Shot Mode Diagram



* *TMRn_CNT* automatically reloads with 0x0000_0001 at the end of the timer period, but software can write any initial value to *TMRn_CNT* before the timer is enabled.

** The default value of *TMRn_CNT* for the first period after a system reset is 0x0000_0000 unless changed by software.

12.4.1 One-Shot Mode Timer Period

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

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

1. *TMRn_CNT* is reset to 0x0000 0001.
2. The timer is disabled by setting *TMRn_CN.ten* = 0.
3. If the timer output is enabled, the timer pin is driven to its active state for one timer clock. It then returns to its inactive state.
4. The timer interrupt bit *TMRn_INTR.irq* is set.
   a. An interrupt is generated if enabled.

12.4.2 One-Shot Mode Configuration

Configure the timer for one-shot mode by performing the following steps:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 0 to select one-shot mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. If using the timer pin:
   a. Configure the pin as a timer output and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired (inactive) state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write an initial value to *TMRn_CNT*, if desired. The initial value only affects the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* = 1 to enable the timer.

The timer period is calculated using the following equation:

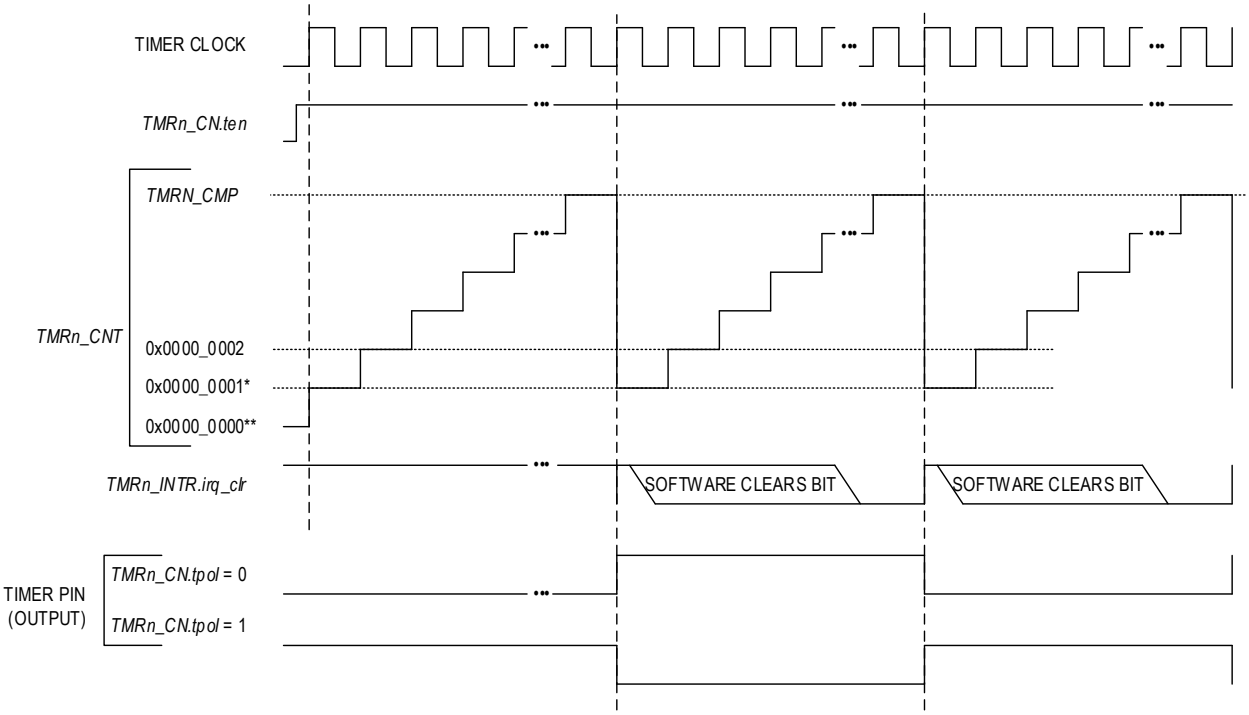
Equation 12-2: One-shot Mode Timer Period

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

12.5 Continuous Mode (1)

In continuous mode, the timer peripheral increments *TMRn_CNT* until it matches *TMRn_CMP*, resets *TMRn_CNT* to 0x0000 0001, and continues incrementing. The timer peripheral can optionally toggle the timer pin's output state at the end of the timer period.

Figure 12-2: Continuous Mode Diagram



* *TMRn_CNT* automatically reloads with 0x0000_0001 at the end of the timer period, but software can write any initial value to *TMRn_CNT* before the timer is enabled.
** The default value of *TMRn_CNT* for the first period after a system reset is 0x0000_0000 unless changed by software.

12.5.1 Continuous Mode Timer Period

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

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

1. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing.
2. If the timer output is enabled, the timer pin toggles state (low to high or high to low).
3. The timer interrupt bit *TMRn_INTR.irq* is set.
    a. An interrupt is generated if enabled.

12.5.2 Continuous Mode Configuration

Configure the timer for continuous mode by performing the steps following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 1 to select continuous mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency, f_{CNT_CLK}.
4. If using the timer pin:
    a. Configure the pin as a timer output and configure the electrical characteristics as needed.
    b. Set *TMRn_CN.tpol* to match the desired inactive state.

5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write an initial value to *TMRn_CNT*, if desired. The initial value is only used for the first period; subsequent timer periods always reset the *TMRn_CNT* register to 1.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* to 1 to enable the timer.

The continuous mode timer period is calculated using *Equation 12-3*.

Equation 12-3: Continuous Mode Timer Period

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

12.6 Counter Mode (2)

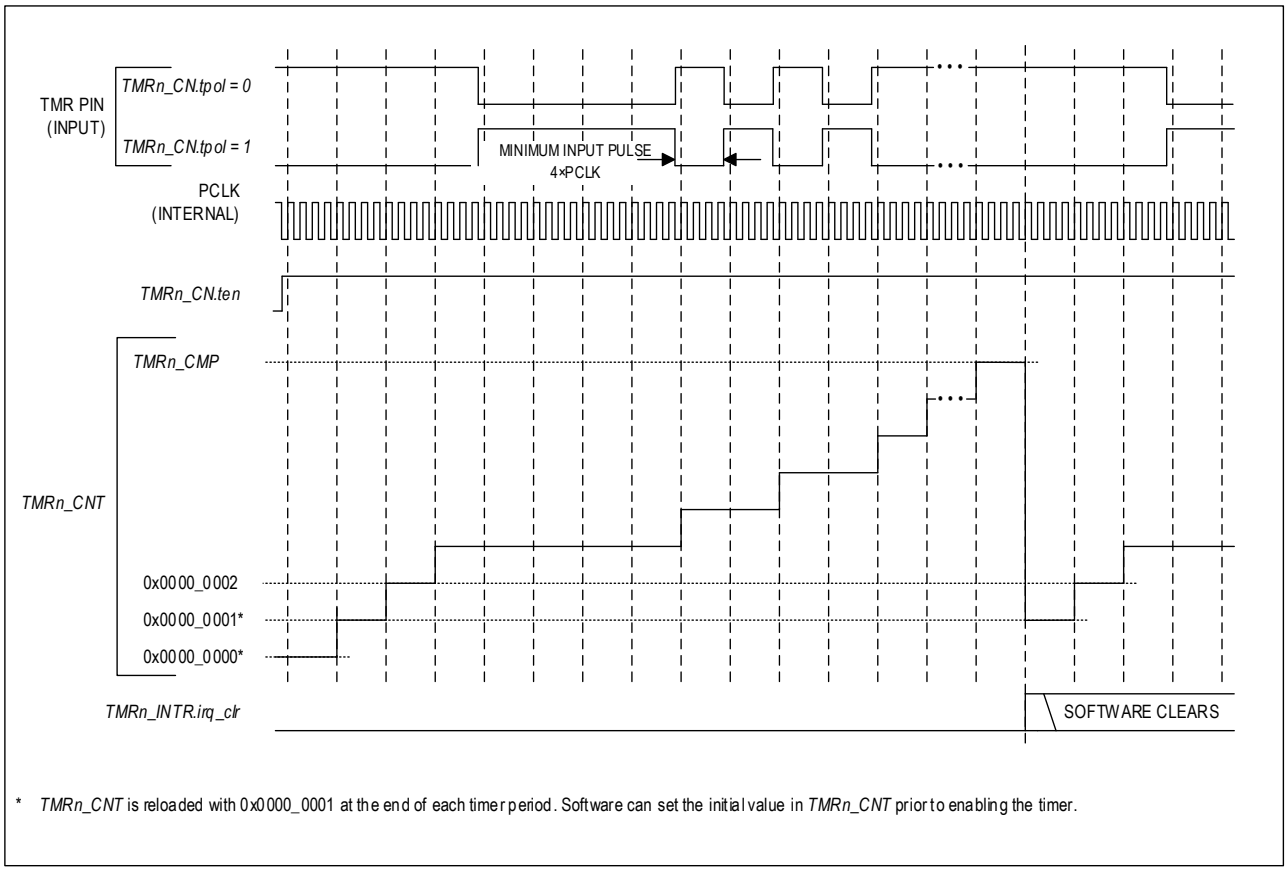
In counter mode, the timer peripheral increments *TMRn_CNT* when a transition occurs on the timer pin. When *TMRn_CNT* = *TMRn_CMP*, the interrupt bit is set, and the *TMRn_CNT* register is set to 0x0000 0001 and continues incrementing. The timer can be configured to increment on either the rising edge or the falling edge, but not both.

The timer prescaler setting has no effect in this mode. The frequency of the timer's input signal (f_{CNT_CLK}) must not exceed 25 percent of the PCLK frequency as shown in the following equation:

Equation 12-4: Counter Mode Frequency/Input Pulse Width

$$f_{CTR\_CLK} \leq \frac{f_{PCLK}\ (Hz)}{4} \quad (minimum\ input\ pulse\ width\ is\ 4 \times PCLK\ pulse\ width)$$

Figure 12-3: Counter Mode Diagram



* *TMRn_CNT* is reloaded with 0x0000_0001 at the end of each timer period. Software can set the initial value in *TMRn_CNT* prior to enabling the timer.

12.6.1 Counter Mode Timer Period

The timer period ends on the rising edge of PCLK following *TMRn_CNT* = *TMRn_CMP*.

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

1. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing on selected transitions of the timer pin.
2. The timer interrupt bit *TMRn_INTR.irq* is set.
    a. An interrupt is generated if enabled.

12.6.2 Counter Mode Configuration

Configure the timer for counter mode by doing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 2 to select counter mode.
3. Configure the timer pin:
    a. Configure the pin as a timer input and configure the electrical characteristics as needed.
    b. Set *TMRn_CN.tpol* to match the desired initial (inactive) state.
4. If using the timer interrupt, set the interrupt priority and enable the interrupt.
5. Write an initial value to *TMRn_CNT*, if desired. The initial value is only used for the first timer period; subsequent timer periods always reset *TMRn_CNT* to 1.
6. Write the compare value to *TMRn_CMP*.
7. Set *TMRn_CN.ten* = 1 to enable the timer.

In counter mode, the number of timer input transitions since timer start is calculated using the following equation:

Equation 12-5: Counter Mode Timer Input Transitions

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

12.7 PWM Mode (3)

In PWM mode, the timer sends a pulse-width modulated output using the timer's output signal. The timer first counts up to the match value stored in the *TMRn_PWM* register. At the end of the cycle, where the *TMRn_CNT* value matches the *TMRn_PWM* value, the timer's output toggles state. The timer continues counting until it reaches the *TMRn_CMP* value.

12.7.1 PWM Mode Timer Period

The timer period ends on the rising edge of PCLK following *TMRn_CNT* = *TMRn_CMP*.

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

1. The *TMRn_CNT* is reset to 0x0000 0001, and the timer resumes counting.
2. The timer output signal is toggled.
3. The timer interrupt bit *TMRn_INTR.irq_clr* is set.
    a. An interrupt is generated if enabled.

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

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

12.7.2 PWM Mode Configuration

Complete the following steps to configure a timer for PWM mode and initiate the PWM operation:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 3 to select PWM mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. Configure the timer pin:
5. Configure the pin as a timer input and configure the electrical characteristics as needed.
6. Set *TMRn_CN.tpol* to match the desired initial (inactive) state.
   a. Set *TMRn_CN.tpol* to select the initial logic level (high or low) and PWM transition state for the timer's output.
   b. Set *TMRn_CNT* to the starting count, typically 0x0000 0001. The initial *TMRn_CNT* value only affects the initial period in PWM mode, with subsequent periods always setting *TMRn_CNT* to 0x0000 0001.
   c. Set the *TMRn_PWM* value to the transition period count.
7. Set the *TMRn_CMP* value for the PWM second transition period.
   Note: TMRn_CMP must be greater than the TMRn_PWM value.
8. Optionally set the timer's interrupt priority and enable the timer's interrupt.
9. Set *TMRn_CN.ten* to 1 to enable the timer and start the PWM.

The PWM period is calculated using the following equation:

Equation 12-6: Timer PWM Period

$$PWM\ period\ in\ seconds = \frac{TMR\_CNT}{f_{CNT\_CLK}\ (Hz)}$$

If an initial starting value other than 0x0000 0001 is loaded into the *TMRn_CNT* register, use the One-Shot mode equation, *Equation 12-2*, to determine the initial PWM period.

If *TMRn_CN.tpol* is 0, the PWM output ratio high time to the total period is calculated using *Equation 12-7*.

Equation 12-7: Timer PWM Output High Time Ratio with Polarity 0

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

If *TMRn_CN.tpol* is set to 1, the PWM output ratio high time to the total period is calculated using *Equation 12-8*.

Equation 12-8: Timer PWM Output High Time Ratio with Polarity 1

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

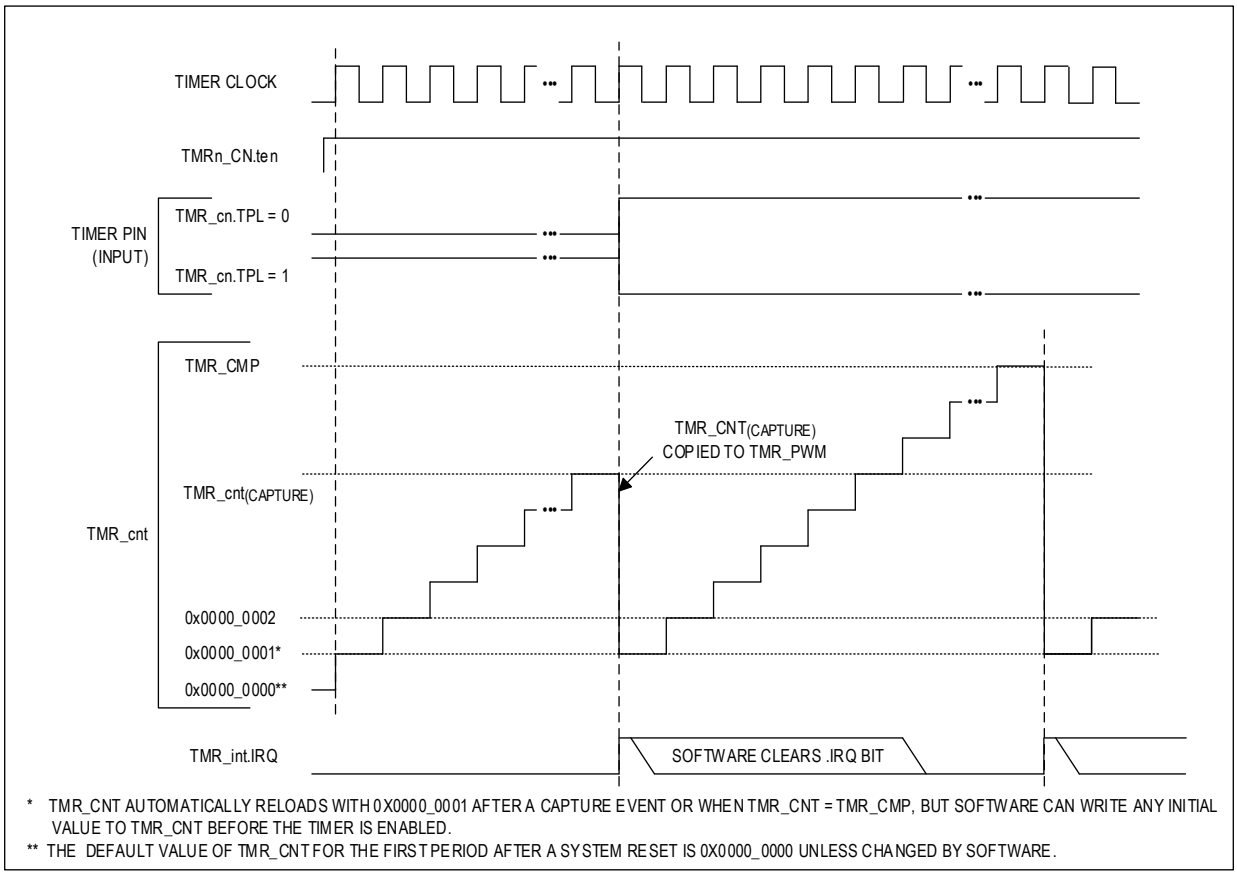
12.8 Capture Mode (4)

Capture mode is most often used to measure the time between events. The timer increments from an initial value until an edge transition occurs on the timer pin. This triggers the 'capture' event, which copies the *TMRn_CNT* value to the *TMRn_PWM.pwm* register, resets *TMRn_CNT* to 0x0000 0001, and continues incrementing. If the timer pin does not go active before *TMRn_CNT* = *TMRn_CMP*, the timer resets *TMRn_CNT* = 0x0000 0001 and continues incrementing. Either event sets the timer interrupt bit.

A capture event can occur before or after a rollover event. Software must track the number of rollover event's that occur prior to a capture event to determine the elapsed time of the capture event. When a capture event occurs, software should reset the count of rollover events.

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

Figure 12-4: Capture Mode Diagram



12.8.1 Capture Mode Timer Period

Two timer period events are possible in capture mode:

The capture event occurs on the timer clock following the selected transition on the timer pin. The timer peripheral automatically performs the following actions:

1. The value in *TMRn_CNT* is copied to *TMRn_PWM*
2. The timer interrupt bit *TMRn_INTR.irq_clr* is set.
   a. An interrupt is generated if enabled.
3. The timer remains enabled and continues incrementing.
4. The timer period ends on the timer clock following *TMRn_CNT* = *TMRn_CMP*.

The timer period event occurs on the timer clock *TMRn_CNT* = *TMRn_CMP*. The timer peripheral automatically performs the following actions when an end of timer period event occurs:

1. The value in *TMRn_CNT* is reset to 0x0000 00001. The timer remains enabled and continues incrementing.
2. The timer interrupt bit *TMRn_INTR.irq_clr* is set.
   a. An interrupt is generated if enabled.

12.8.2 Capture Mode Configuration

Configure the timer for capture mode by performing the following steps:

1. Disable the timer by setting *TMRn_CN.ten* to 0.
2. Select capture mode by setting *TMRn_CN.tmode* to 4.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. If using the timer pin:
   a. Configure the pin as a timer output and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired (inactive) state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write the initial value to *TMRn_CNT*. This affects only the first period; subsequent periods always begin with 0x0000 0001.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* = 1 to enable the timer.

The capture mode timer period is calculated using the following equation:

Equation 12-9: Capture Mode Elapsed Time Calculation in Seconds

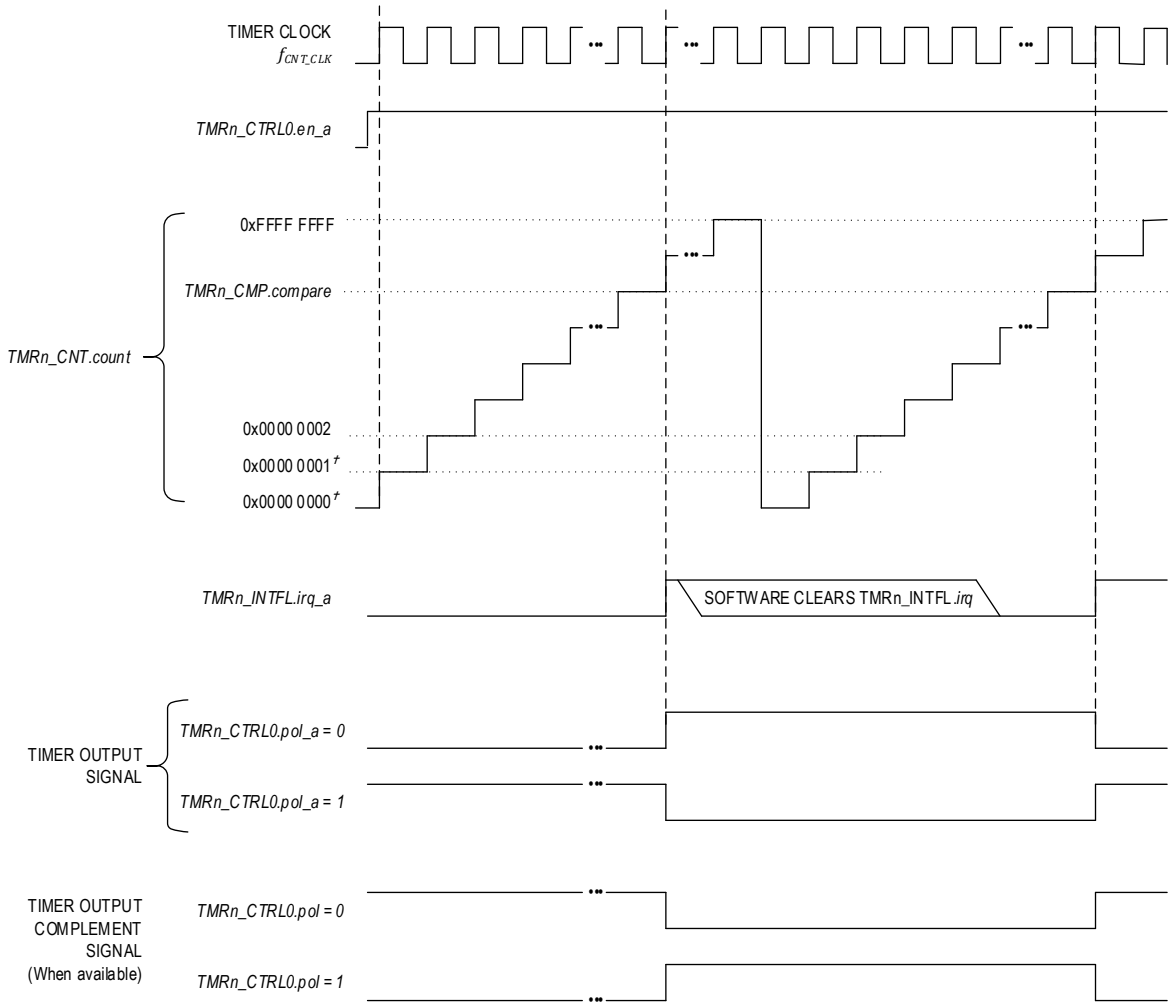
$$Capture\ elapsed\ time = \frac{\left(TMR\_PWM - TMR\_CNT_{INITIAL\_VALUE}\right) + \left((Number\ of\ rollover\ events) \times \left(TMR\_CMP - TMR\_CNT_{INITIAL\_VALUE}\right)\right)}{f_{CNT\_CLK}}$$

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.*

12.9 Compare Mode (5)

In compare mode, the timer peripheral increments continually, allowing the timer to be a programmable 32-bit programmable period timer. The end of 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.

Figure 12-5: Compare Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 5 (Compare)

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

12.9.1 Compare Mode Timer Period

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

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

1. The timer remains enabled and continues incrementing. Unlike other modes, *TMRn_CNT* is set to 0x0000 0000 at the end of the timer period.
2. If the timer output is enabled, then the timer pin toggles state (low to high or high to low).
3. The timer interrupt bit *TMRn_INTR.irq_clr* is set.
    a. An interrupt is generated if enabled.

12.9.2 Compare Mode Configuration

Configure the timer for compare mode by doing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 5 to select compare mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. If using the timer pin:
    a. Configure the pin for the timer output alternate function and configure the electrical characteristics as needed.
    b. Set *TMRn_CN.tpol* to match the desired (inactive) state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write the initial value to *TMRn_CNT*. This affects only the first period as the counter increments continuously, rolling over to 0x0000 0000 and continuing.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* = 1 to enable the timer.

The compare mode timer period is calculated using *Equation 12-10*.

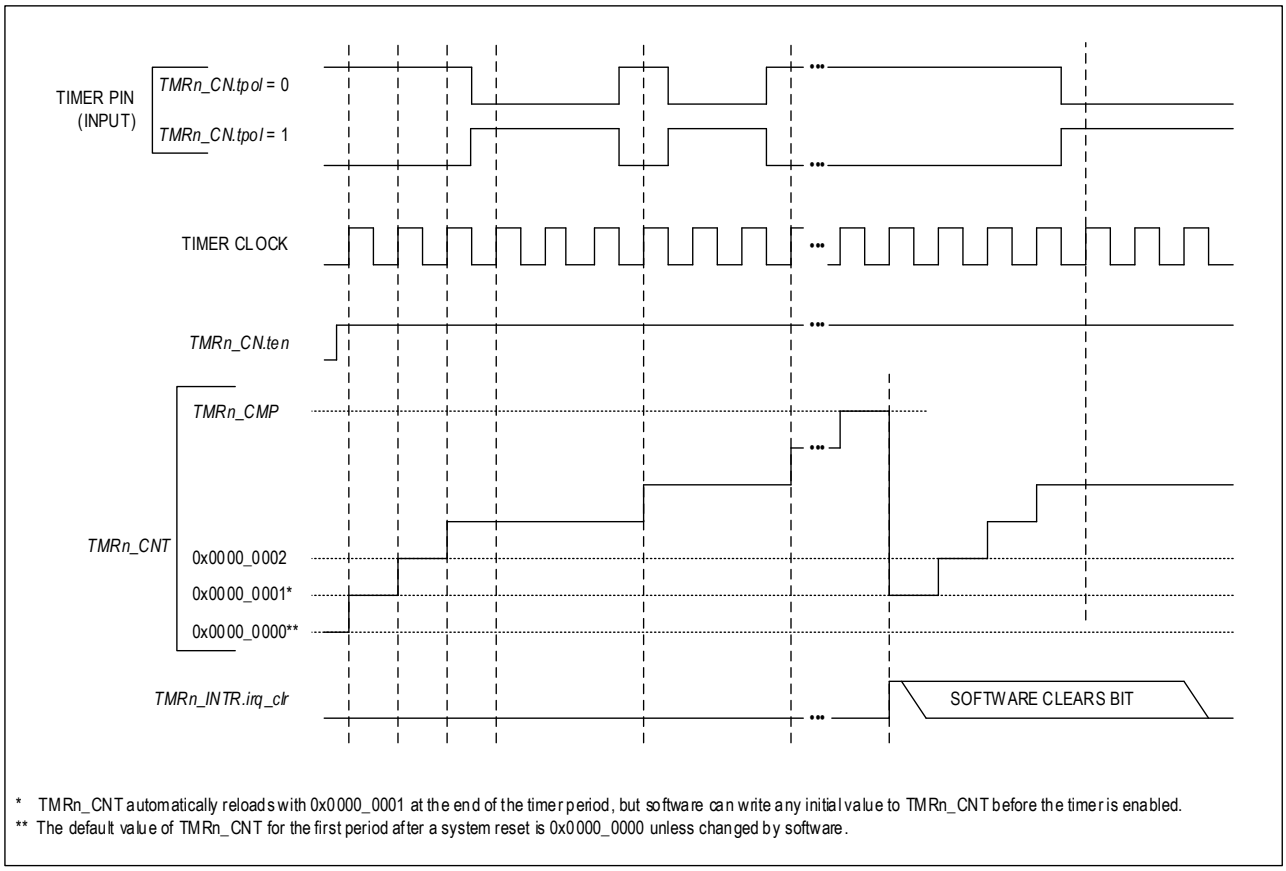
Equation 12-10: Compare Mode Timer Period

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

12.10 Gated Mode (6)

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

Figure 12-6: Gated Mode Diagram



12.10.1 Gated Mode Timer Period

The timer period ends when *TMRn_CNT* = *TMRn_CMP* and the timer automatically performs the following actions:

1. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing.
2. The timer interrupt bit *TMRn_INTR.irq_clr* is set.
   a. An interrupt is generated if enabled.

12.10.2 Gated Mode Configuration

Configure the timer for gated mode by performing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 6 to select gated mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. Configure the timer pin:
   a. Configure the pin as a timer input and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired initial (inactive) state.

5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write an initial value to *TMRn_CNT*, if desired. This affects only the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN*.*ten* = 1 to enable the timer.

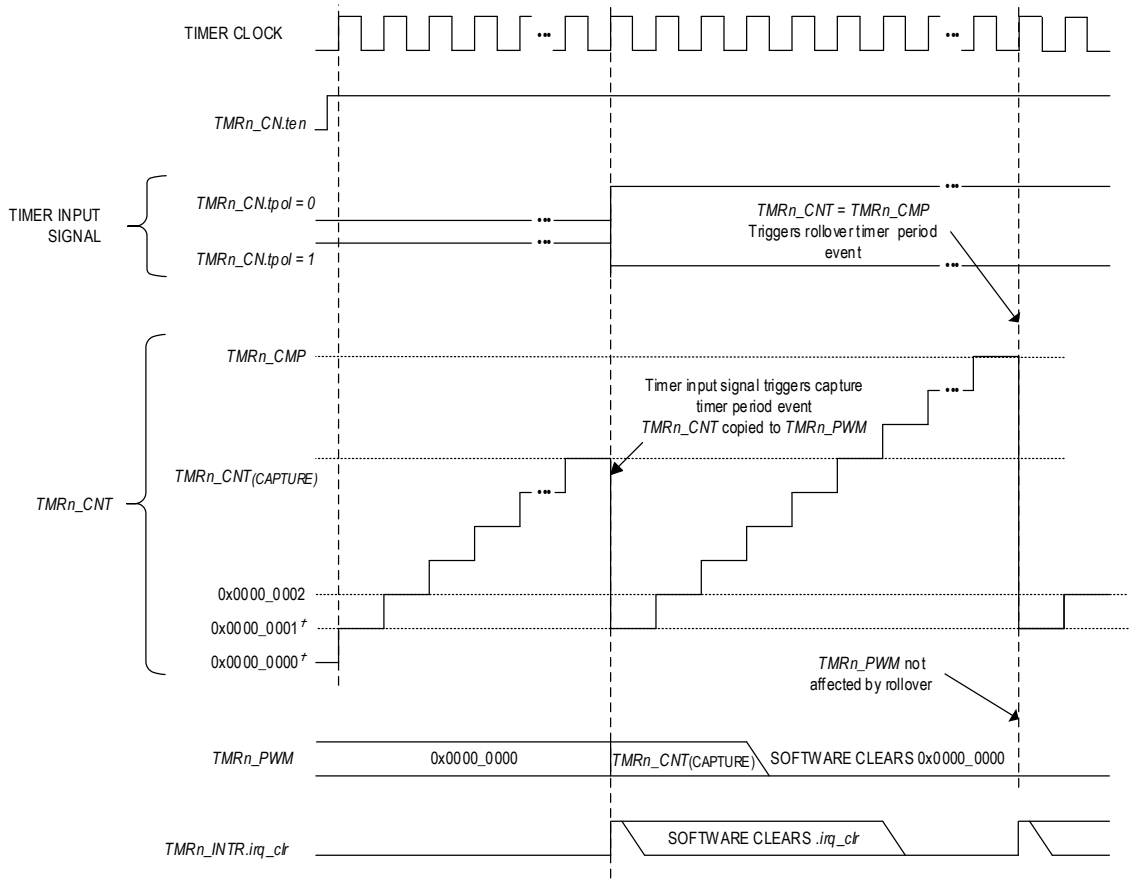
12.11 Capture/Compare Mode (7)

In capture/compare mode, the timer starts counting after the first external timer input transition occurs. The transition, a rising edge or falling edge on the timer's input signal, is set using the *TMRn_CN*.*tpol* bit.

After the first transition of the timer input signal, each subsequent transition captures the *TMRn_CNT* value, writing it to the *TMRn_PWM* register (capture event). When a capture event occurs, a timer interrupt is generated, the *TMRn_CNT* value is reset to 0x0000 0001, and the timer resumes counting.

If no capture event occurs, the timer counts up to the *TMRn_CMP* value. At the end of the cycle, where the *TMRn_CNT* equals the *TMRn_CMP* value, a timer interrupt is generated, the *TMRn_CNT* value is reset to 0x0000 0001, and the timer resumes counting.

Figure 12-7: Capture/Compare Mode Diagram



† *TMRn_CNT.count* defaults to 0x0000_0000 on a timer reset. *TMRn_CNT.count* reloads to 0x0000_0001 for all following timer periods.

12.11.1 Capture/Compare Timer Period

The timer period ends when the selected transition occurs on the timer pin or on the clock cycle following *TMRn_CNT* = *TMRn_CMP*.

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

If a transition on the timer pin caused the end of the timer period:

1. The value in *TMRn_CNT* is copied to *TMRn_PWM*.
2. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing.
3. The timer interrupt bit, *TMRn_INTR.irq_clr*, is set.
   a. An interrupt is generated if enabled.

If the end of the timer period is caused by transition on the timer pin:

1. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing.
2. The timer interrupt bit *TMRn_INTR.irq_clr* is set.
   a. An interrupt is generated if enabled.

12.11.2 Capture/Compare Configuration

Configure the timer for capture/compare mode by performing the following steps:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 7 to select capture/compare mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. Configure the timer pin:
   a. Configure the pin as a timer input and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to select the positive edge (*TMRn_CN.tpol* = 0) or negative edge (*TMRn_CN.tpol* = 0) transition causes the capture event.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write an initial value to *TMRn_CNT*, if desired. This affects only the first period; subsequent timer periods always reset *TMRn_CNT*= 0x0000 0001.
7. Set *TMRn_CN.ten* to 1 to enable the timer. Counting starts after the first transition of the timer's input signal.

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

In capture/compare mode, the elapsed time from the timer start to the capture event is calculated using *Equation 12-11*.

Equation 12-11: Capture Mode Elapsed Time

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

12.12 Timer Registers

See *Table 3-1* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in the register summary below. 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 12-1: Timer Register Summary

Offset	Register Name	Description
[0x0000]	*TMRn_CNT*	Timer Counter Register
[0x0004]	*TMRn_CMP*	Timer Compare Register
[0x0008]	*TMRn_PWM*	Timer PWM Register
[0x000C]	*TMRn_INTR*	Timer Interrupt Register
[0x0010]	*TMRn_CN*	Timer Control Register
[0x0014]	*TMRn_NOLCMP*	Timer Non-Overlapping Compare Register

12.12.1 Register Details

Table 12-2: Timer Count Registers

Timer Count Register			TMRn_CNT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	count	R/W	0	**Timer Count Value** The current count value for the timer. This field increments as the timer counts. Reads of this register are always valid. Before writing this field, disable the timer by clearing bit *TMRn_CN.ten*.

Table 12-3: Timer Compare Registers

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

Table 12-4: Timer PWM Registers

Timer PWM Register			TMRn_PWM	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	pwm	R/W	0	**Timer PWM Match** This field sets the count value for the first transition period of the PWM cycle in PWM mode. At the end of the cycle, where *TMRn_CNT* equals *TMRn_CMP*, the PWM output transitions to the second period of the PWM cycle. The second PWM period count is stored in the *TMRn_CMP* register. The value set for *TMRn_PWM.pwm* must be less than the value set in *TMRn_CMP* for PWM mode operation. **Timer Capture Value** In capture, compare, and capture/compare modes, this field is used to store the *TMRn_CNT* value when a capture, compare, or capture/compare event occurs.

Table 12-5: Timer Interrupt Registers

Timer Interrupt Register			TMRn_INTR	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	irq_clr	R/WC	0	**Timer Interrupt** If this field reads 1, a timer interrupt condition occurred. Writing any value to this bit clears the timer's interrupt. 0: Timer interrupt is not active. 1: Timer interrupt occurred.

Table 12-6: Timer Control Registers

Timer Control Register			TMRn_CN	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:13	-	RO	0	**Reserved**
12	pwmckbd	RO	1	**Reserved**
11	nollpol	RO	0	**Reserved**
10	nolhpol	RO	0	**Reserved**
9	pwmsync	RO	0	**Reserved**

Timer Control Register			TMRn_CN	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
8	pres3	R/W	0	**Timer Prescaler Select MSB** See *TMRn_CN.pres* for details on this field's usage.
7	ten	R/W	0	**Timer Enable** 1: Timer enabled. 0: Timer disabled.
6	tpol	R/W	0	**Timer Polarity** This field selects the polarity of the timer's input and output signal. This setting is not used if the GPIO is not configured for the alternate function. The specific mode of the timer determines the *tpol* field meaning. See the mode's detailed configuration section for *tpol* usage.
5:3	pres	R/W	0	**Timer Prescaler Select** This field sets the timer's prescaler value. The prescaler divides the PCLK input to the timer and sets the timer's count clock, $f_{CNT\_CLK} = {PCLK\ (Hz)}/{prescaler}$. The timer's prescaler setting is a 4-bit value with *pres3* as the most significant bit and *pres* as the three least significant bits. The table below shows the prescaler values based on *pres3*:*pres*.

pres3	pres	Prescaler	f_{CNT_CLK}
0	0b000	1	${f_{PCLK}\ (Hz)}/{1}$
0	0b001	2	${f_{PCLK}\ (Hz)}/{2}$
0	0b010	4	${f_{PCLK}\ (Hz)}/{4}$
0	0b011	8	${f_{PCLK}\ (Hz)}/{8}$
0	0b100	16	${f_{PCLK}\ (Hz)}/{16}$
0	0b101	32	${f_{PCLK}\ (Hz)}/{32}$
0	0b110	64	${f_{PCLK}\ (Hz)}/{64}$
0	0b111	128	${f_{PCLK}\ (Hz)}/{128}$
1	0b000	256	${f_{PCLK}(Hz)}/{256}$
1	0b001	512	${f_{PCLK}\ (Hz)}/{512}$
1	0b010	1024	${f_{PCLK}\ (Hz)}/{1024}$
1	0b011	2048	${f_{PCLK}\ (Hz)}/{2048}$
1	0b100	4096	${f_{PCLK}\ (Hz)}/{4096}$
1	0b101	Reserved	Reserved
1	0b110	Reserved	Reserved
1	0b111	Reserved	Reserved

Timer Control Register			TMRn_CN	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
2:0	tmode	R/W	0	**Timer Mode Select** This field sets the timer's operating mode. *tmode* / Timer Mode: 0b000 One-shot 0b001 Continuous 0b010 Counter 0b011 PWM 0b100 Capture 0b101 Compare 0b110 Gated 0b111 Capture/Compare

Table 12-7: Timer Non-Overlapping Compare Register

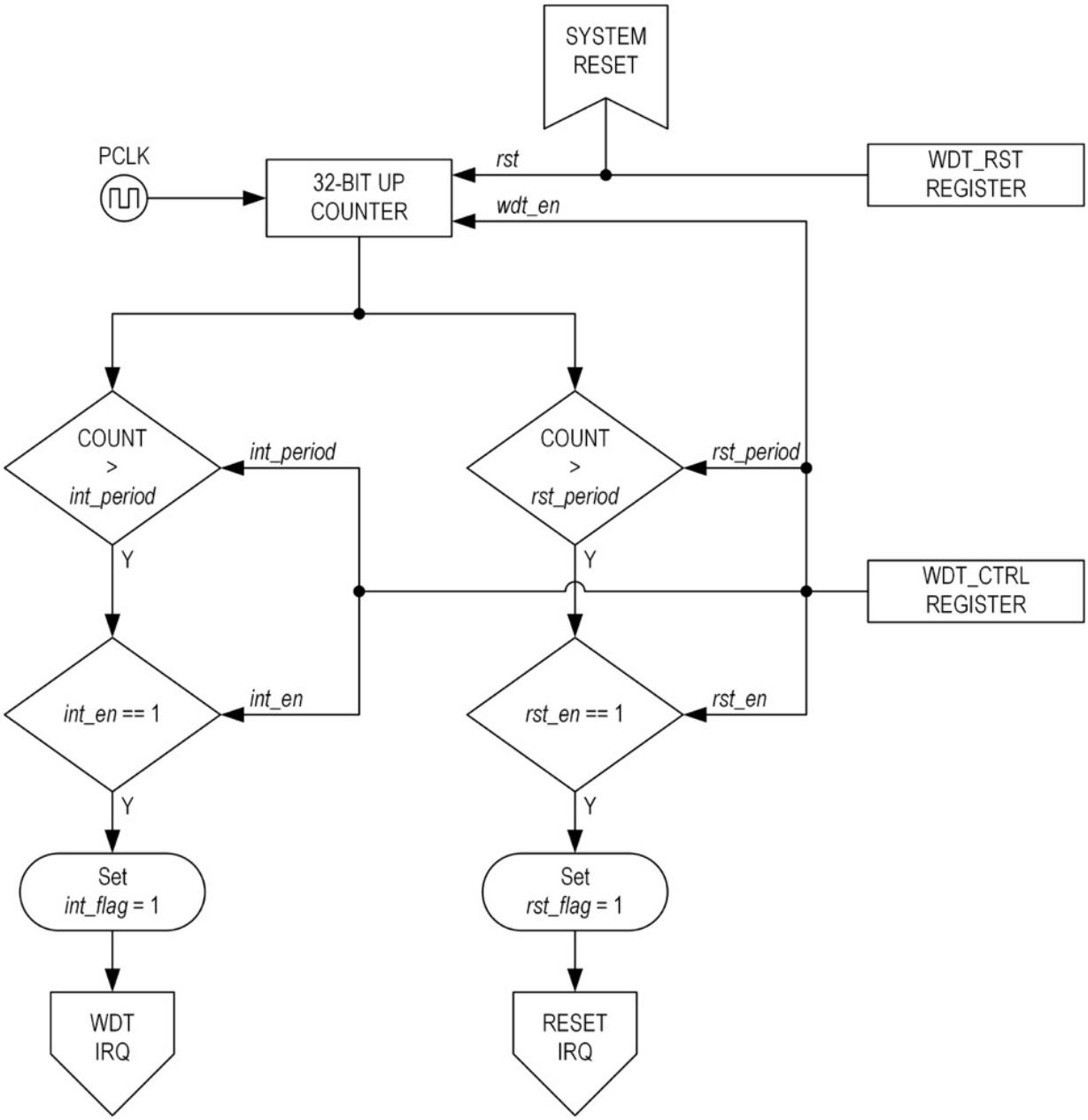
Timer Non-Overlapping Compare			TMRn_NOLCMP	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:8	nolhcpm	RO	0	**Reserved**
7:0	nollcmp	RO	0	**Reserved**

13. Watchdog Timer (WDT)

The watchdog timer protects against corrupt or unreliable software, power faults, and other system-level problems, which can place the microcontroller into an improper operating state. When the application is executing properly, application software periodically resets the watchdog counter. If the watchdog timer interrupt is enabled and the software does not reset the counter within the interrupt time period, *WDTn_CTRL.int_period*, the watchdog timer generates a watchdog timer interrupt. If the watchdog timer reset is enabled and the software does not reset the counter within the reset time period, *WDTn_CTRL.rst_period*, the watchdog timer generates a system reset. See *Operating Modes* for details on the availability of the WDT in each mode.

Figure 13-1 shows a flow chart of the watchdog timer operation.

Figure 13-1: Watchdog Timer Flow Chart



13.1 Features

- Sixteen programmable time periods for the watchdog interrupt:
  - 2^{16} through 2^{31} PCLK cycles
- Sixteen programmable time periods for the watchdog reset:
  - 2^{16} through 2^{31} PCLK cycles
- The watchdog timer counter is reset on all forms of reset.

13.2 Usage

Utilizing the watchdog timer in the application software is straightforward. As early as possible in the application software, enable the watchdog timer interrupt and watchdog timer reset. Periodically the application software must write to the WDT_RST register to reset the watchdog counter. If program execution becomes lost, the watchdog timer interrupt occurs, giving the system a "last chance" to recover from whatever circumstance caused the improper code execution. The interrupt routine can either attempt to repair the situation or allow the watchdog timer reset to occur. In the event of a system software failure, the interrupt is not executed, and the watchdog system reset recovers operation.

As soon as possible after a reset, the application software should interrogate the *WDTn_CTRL.rst_flag* to determine if the reset event resulted from a watchdog timer reset. If so, application software should assume that there was a program execution error and take whatever steps necessary to guard against a software corruption issue.

13.3 Interrupt and Reset Period Timeout Configuration

Each watchdog timer supports two independent timeout periods, the interrupt period timeout and reset period timeout.

- **Interrupt Period Timeout:** *WDTn_CTRL.int_period* sets the number of PCLK cycles until a watchdog timer interrupt is generated. This period must be less than the reset period timeout for the watchdog timer interrupt to occur.
- **Reset Period Timeout:** *WDTn_CTRL.rst_period* sets the number of PCLK cycles until a system reset event occurs.

The interrupt and reset period timeouts are calculated using *Equation 13-1* and *Equation 13-2* respectively, where $f_{PCLK} = {f_{SYSCLK}}/{2}$.

Equation 13-1: Watchdog Timer Interrupt Period

$$t_{INT\_PERIOD} = \left(\frac{1}{f_{PCLK}}\right) \times 2^{(31-WDTn\_CTRL.int\_period)}$$

Equation 13-2: Watchdog Timer Reset Period

$$t_{RST\_PERIOD} = \left(\frac{1}{f_{PCLK}}\right) \times 2^{(31-WDTn\_CTRL.rst\_period)}$$

13.4 Enabling the Watchdog Timer

Set *WDTn_CTRL.wdt_en* to 1 to enable the watchdog timer.

13.4.1 Enabling the Watchdog Timer Interrupt and Reset Functionality

The watchdog timer supports a watchdog timer interrupt and a watchdog timer reset feature. Set the *WDTn_CTRL.int_en* field to 1 to enable the watchdog timer interrupt. Register an interrupt handler for the WDT_IRQn interrupt. Performing a watchdog timer reset in the interrupt handler resets the timer and allows continued application execution. The interrupt time is set using *Equation 13-1*.

Enable the watchdog timer reset feature by setting the *WDTn_CTRL.rst_en* field to 1. When the watchdog timer reset is enabled, failure to reset the watchdog timer in the interval set using *Equation 13-2* results in a watchdog timer reset.

13.5 Disabling the Watchdog Timer

The watchdog timers can be disabled by the application code manually or by the microcontroller automatically as shown below.

13.5.1 Manual Disable

Setting *WDTn_CTRL.wdt_en* to 0 disables the watchdog timer.

13.5.2 Automatic Disable

A POR event automatically disables the watchdog timer. The watchdog timer's retain their enabled state in all other forms of reset.

13.6 Resetting the Watchdog Timer

To prevent a watchdog interrupt, a watchdog reset, or both, the software must write the reset sequence, shown below, to the *WDTn_RST.wdt_rst* field before a WDT interrupt or reset timeout occurs.

13.6.1 Reset Sequence

1. Write 0xA5 to *WDTn_RST.wdt_rst*.
2. Write 0x5A to *WDTn_RST.wdt_rst*.

13.7 Detection of a Watchdog Reset Event

During system start-up, the software should check the *WDTn_CTRL.rst_flag* to determine if the reset was the result of a watchdog reset. The software is responsible for taking appropriate actions if a watchdog reset occurred.

13.8 Watchdog Timer Registers

See *Table 3-1* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 13-1*. 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. The WDT registers are only reset on a POR.

Table 13-1: Watchdog Timer Register Offsets, Names and Descriptions

Offset	Register Name	Description
[0x0000]	*WDTn_CTRL*	Watchdog Timer Control Register
[0x0004]	*WDTn_RST*	Watchdog Timer Reset Register

13.8.1 Watchdog Timer Register Details

Table 13-2: Watchdog Timer Control Register

Watchdog Timer Control			**WDTn_CTRL**	**[0x0000]**
Bits	**Name**	**Access**	**Reset**	**Description**
31	rst_flag	R/W	0	**WDT Reset Flag** If set, a watchdog system reset occurred. Write 0 to clear the status of this field. 0: Watchdog did not cause reset event. 1: Watchdog reset occurred.
30:12	-	RO	0	**Reserved**

Watchdog Timer Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
11	rst_en	R/W	0	**Reset Enable** Enable/Disable system reset if the *rst_period* expires. Only reset by power on reset. 0: Disabled 1: Enabled
10	int_en	R/W	0	**Interrupt Enable** Enable or Disable the watchdog interrupt. 0: Disabled 1: Enabled
9	int_flag	R/W	0	**Interrupt Flag** If set, the watchdog interrupt period has occurred. Write 0 to clear this field. 0: Normal operation. 1: Interrupt period expired. Generates a WDT_IRQn interrupt if *WDTn_CTRL.int_en* = 1.
8	wdt_en	R/W	0	**Enable** Enable or disable the watchdog timer. Only reset by a power on reset. To enable the watchdog timer, the following sequence of writes must be performed. 1) Write *WDTn_RST* to 0xA5. 2) Write *WDTn_RST* to 0x5A. 3) Write *wdt_en* to 1. 0: Disabled 1: Enabled
7:4	rst_period	R/W	0	**Reset Period** Sets the number of PCLK cycles until a system reset occurs if the watchdog timer is not reset. 0xF: $2^{16} \times t_{PCLK}$ 0xE: $2^{17} \times t_{PCLK}$ 0xD: $2^{18} \times t_{PCLK}$ 0xC: $2^{19} \times t_{PCLK}$ 0xB: $2^{20} \times t_{PCLK}$ 0xA: $2^{21} \times t_{PCLK}$ 0x9: $2^{22} \times t_{PCLK}$ 0x8: $2^{23} \times t_{PCLK}$ 0x7: $2^{24} \times t_{PCLK}$ 0x6: $2^{25} \times t_{PCLK}$ 0x5: $2^{26} \times t_{PCLK}$ 0x4: $2^{27} \times t_{PCLK}$ 0x3: $2^{28} \times t_{PCLK}$ 0x2: $2^{29} \times t_{PCLK}$ 0x1: $2^{30} \times t_{PCLK}$ 0x0: $2^{31} \times t_{PCLK}$

Watchdog Timer Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
3:0	int_period	R/W	0	**Interrupt Period** Sets the number of PCLK cycles until a watchdog timer interrupt is generated. 0xF: $2^{16} \times t_{PCLK}$ 0xE: $2^{17} \times t_{PCLK}$ 0xD: $2^{18} \times t_{PCLK}$ 0xC: $2^{19} \times t_{PCLK}$ 0xB: $2^{20} \times t_{PCLK}$ 0xA: $2^{21} \times t_{PCLK}$ 0x9: $2^{22} \times t_{PCLK}$ 0x8: $2^{23} \times t_{PCLK}$ 0x7: $2^{24} \times t_{PCLK}$ 0x6: $2^{25} \times t_{PCLK}$ 0x5: $2^{26} \times t_{PCLK}$ 0x4: $2^{27} \times t_{PCLK}$ 0x3: $2^{28} \times t_{PCLK}$ 0x2: $2^{29} \times t_{PCLK}$ 0x1: $2^{30} \times t_{PCLK}$ 0x0: $2^{31} \times t_{PCLK}$

Table 13-3: Watchdog Timer Reset Register

Watchdog Timer Reset			WDTn_RST	[0x0004]
Bits	**Register Field**	**Access**	**Reset**	**Description**
31:8-	-	RO	0	**Reserved**
7:0	wdt_rst	R/W	0	**Reset Register** Writing the watchdog counter reset sequence to this register resets the watchdog counter. The following is the required reset sequence to reset the watchdog and prevent a watchdog timer interrupt or watchdog system reset. • Write *WDTn_RST* to 0xA5. • Write *WDTn_RST* to 0x5A.

14. I²C Master/Slave Serial Controller (I2C)

The I²C peripherals can be configured as either an I²C master or an I²C slave 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 Maxim Application Note 4024 "SPI/I²C Bus Lines Control Multiple Peripherals" at *https://www.maximintegrated.com/en/app-notes/index.mvp/id/4024*

14.1 I²C Master/Slave Features

Each I²C master/slave is compliant with the I²C Bus Specification and include the following features:

- Communicates through a serial data address (SDA) and a serial clock line (SCL).
- Operates as either a master or slave device as a transmitter or receiver.
- Supports I²C Standard Mode, Fast Mode, Fast Mode Plus, and High Speed (Hs) Mode.
- Transfers data at rates up to:
  - 100kbps in Standard Mode.
  - 400kbps in Fast Mode.
  - 1Mbps in Fast Mode Plus.
  - 3.4Mbps in Hs Mode.
- Supports multi-master systems, including support for arbitration and clock synchronization for Standard 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 firmware.
- 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.

14.2 Instances

The three instances of the peripheral shown in *Table 14-1* list the locations of the SDA and SCL signals for each of the I2Cn peripherals per package.

Table 14-1: MAX32660 I²C Peripheral Pins

I²C Instance	Alternate Function	Alternate Function #	Package 16-WLP	Package 20-TQFN/24-TQFN
I2C0	I2C0_SCL	AF1	P0.8	P0.8
	I2C0_SDA	AF1	P0.9	P0.9
I2C1	I2C1_SCL	AF1	P0.2	P0.2
	I2C1_SDA	AF1	P0.3	P0.3

14.3 I²C Overview

14.3.1 I²C Bus Terminology

Table 14-2 contains terms and definitions used in this chapter for the I²C bus terminology.

Table 14-2: I²C Bus Terminology

Term	Definition
Transmitter	The device that sends data to the bus.
Receiver	The device that receives data from the bus.
Master	The device that initiates a transfer, generates clock signals, and terminates a transfer.
Slave	The device addressed by a master.
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message.
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one can do so and the resulting message is not corrupted.
Synchronization	Procedure to synchronize the clock signals of two or more devices.
Clock Stretching	When a slave device holds SCL low to pause a transfer until it is ready. This feature is optional according to the I²C Specification; thus, a master does not have to support slave clock stretching if none of the slaves in the system are capable of clock stretching.

14.3.2 I²C Transfer Protocol Operation

The I²C protocol operates over a two-wire bus: a clock circuit (SCL) and a data circuit (SDA). I²C is a half-duplex protocol: only one device is allowed to transmit on the bus at a time.

Each transfer is initiated when the bus master sends a START or repeated START condition. It is followed by the I²C slave address of the targeted slave device plus a read/write bit. The master can transmit data to the slave (a 'write' operation) or receive data from the slave (a 'read' operation). Information is sent most-significant-bit (MSB) first. Following the slave address, the master indicates a read or write operation and then exchanges data with the addressed slave. An acknowledge bit is sent by the receiving device after each byte is transferred. When all necessary data bytes have been transferred, a STOP or RESTART condition is sent by the bus master to indicate the end of the transaction. After the STOP condition has been sent, the bus is idle and ready for the next transaction. After a RESTART condition is sent, the same master begins a new transmission. The number of bytes that can be transmitted per transfer is unrestricted.

14.3.3 START and STOP Conditions

A START condition occurs when a bus master pulls SDA from high to low while SCL is high, and a STOP condition occurs when a bus master allows SDA to be pulled from low to high while SCL is high. Because these are unique conditions that cannot occur during normal data transfer, they are used to denote the beginning and end of the data transfer.

14.3.4 Master Operation

I²C transmit and receive data transfer operations occur through the *I2Cn_FIFO* register. Writes to the register load the transmit FIFO and reads of the register return data from the receive FIFO. If a slave sends a NACK in response to a write operation, the I²C master generates an interrupt. The I²C controller can be configured to issue a STOP condition to free the bus.

The receive FIFO contains the received data. If the receive FIFO is full or the transmit FIFO is empty, the I²C master stops the clock to allow time to read bytes from the receive FIFO or load bytes into the transmit FIFO.

14.3.5 Acknowledge and Not Acknowledge

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

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

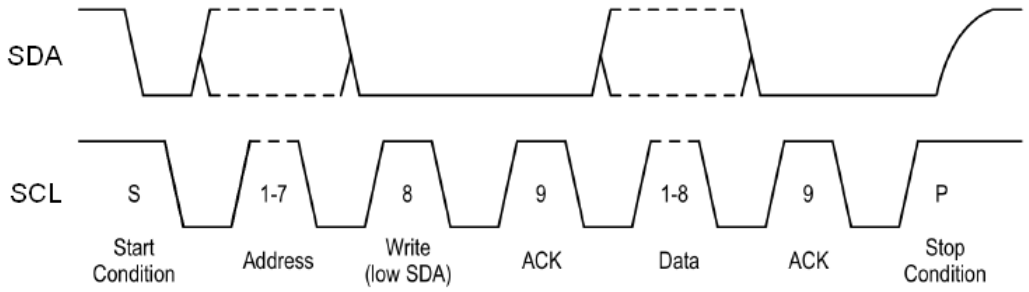
A receiver can generate a NACK after a byte transfer if any of the following conditions occur:

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

14.3.6 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 14-1*.

Figure 14-1: I²C Write Data Transfer



An example of an I²C data transfer is as follows:

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

14.4 Configuration and Usage

14.4.1 SCL and SDA Bus Drivers

SCL and SDA are open-drain signals. In this device, once the I²C peripheral is enabled and the proper GPIO alternate function is selected, the corresponding pad circuits are automatically configured as open-drain outputs. However, SCL can also be optionally configured as a push-pull driver to conserve power and avoid the need for any pullup resistor. This should only be used in systems where no I²C slave device can hold SCL low, such as for clock stretching. Push-pull operation is enabled by setting *I2Cn_CTRL.scl_pp_mode* to 1. SDA, on the other hand, always operates in open-drain mode.

14.4.2 SCL Clock Configurations

The SCL frequency is dependent upon the values of 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.

14.4.3 SCL Clock Generation for Standard, Fast and Fast-Plus Modes

The master generates the I²C clock on the SCL line. When operating as a master, application code must configure the *I2Cn_CLK_HI* and *I2Cn_CLK_LO* registers for the desired I²C operating frequency.

The SCL high time is configured in the I²C clock high time register field *I2Cn_CLK_HI.hi* using *Equation 14-2*. The SCL low time is configured in the I²C clock low time register field *I2Cn_CLK_LO.lo* using *Equation 14-3*. Each of these fields is 8-bits. The I²C frequency value is shown in *Equation 14-1*.

Equation 14-1: I²C Clock Frequency

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

Equation 14-2: I²C Clock High Time Calculation

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

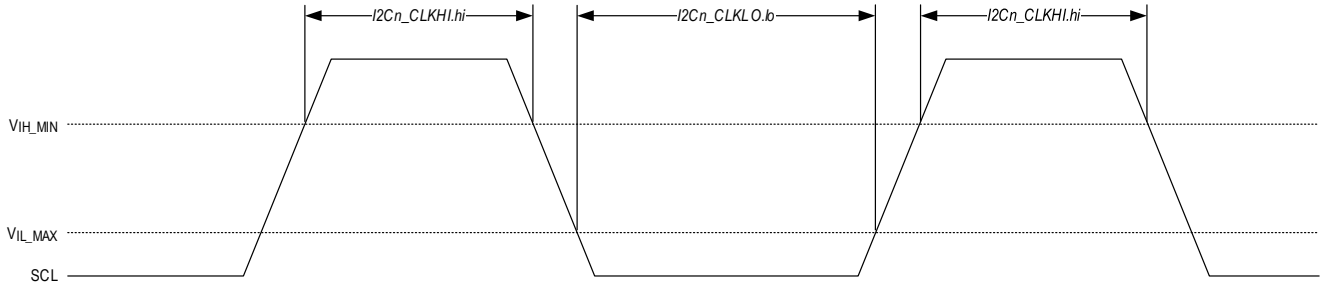
Equation 14-3: I²C Clock Low Time Calculation

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

Figure 14-2 shows the association between the SCL clock low and high times for Standard Mode, Fast Mode, and Fast Mode Plus I²C frequencies.

Figure 14-2 shows the association between the SCL clock low and high times for Standard Mode, Fast Mode, and Fast Mode Plus I²C frequencies.

Figure 14-2: I²C SCL Timing for Standard, Fast and Fast-Plus Modes



During synchronization, external masters or external slaves can drive SCL simultaneously. This affects the SCL duty cycle. By monitoring SCL, the controller determines if an external master or slave is holding SCL low. In either case, the controller waits until SCL is high before starting to count the number of SCL high cycles. Similarly, if an external master pulls SCL low before the controller has finished counting SCL high cycles, then the controller starts counting SCL low cycles and releases SCL once the time period, *I2Cn_CLK_LO.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.

14.4.4 SCL Clock Generation for Hs-mode

To operate the I²C interface in Hs-mode at its maximum speed (~3.4MHz), values to be programmed into the *I2Cn_HS_CLK.hsclk_lo* register and *I2Cn_HS_CLK.hsclk_hi* register must be determined. Since the Hs-mode operation is entered by first using one of the lower speed modes for preamble, a relevant lower speed mode must also be configured. See *SCL Clock Generation for Standard, Fast and Fast-Plus Modes* for information regarding configuration of lower speed modes.

14.4.4.1 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.

14.4.4.2 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 14-4: I²C Target SCL Frequency

Desired Target Maximum I²C *Frequency:* $f_{SCL} = \frac{1}{t_{SCL}}$.

In Hs-mode, the analog glitch filter (AF_MIN) within the device adds a minimum delay of t_{AF_MIN} = 10ns.

Equation 14-5: Determining the I2Cn_HS_CLK.hsclk_lo Register Value

$$I2Cn\_HS\_CLK.hsclk\_lo = MAX\left\{\left\lfloor\left(\frac{t_{LOW\_MIN} + t_{FCL\_MAX} + t_{I2C\_CLK} - t_{AF\_MIN}}{t_{I2C\_CLK}}\right)\right\rfloor - 1, \quad \frac{t_{SCL}}{t_{I2C\_CLK}} - 1\right\}$$

Equation 14-6: Determining the I2Cn_HS_CLK.hsclk_hi Register Value

$$I2Cn\_HS\_CLK.hsclk\_hi = \left\lfloor\left(\frac{t_{HIGH\_MIN} + t_{rCL\_MAX} + t_{I2C\_CLK} - t_{AF\_MIN}}{t_{I2C\_CLK}}\right)\right\rfloor - 1$$

Equation 14-7: The Calculated Frequency of the I²C Bus Clock Using the Results of Equation 14-5 *and* Equation 14-6

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

Table 14-3 shows the I²C bus clock calculated frequencies given different f_{SYS_CLK} frequencies.

Table 14-3: Calculated I²C Bus Clock Frequencies

f_{SYS_CLK} (MHz)	*I2Cn_HS_CLK.hsclk_hi*	*I2Cn_HS_CLK.hsclk_lo*	Calculated Frequency (MHz)
100	4	9	3.3
50	2	4	3.125
25	1	2	2.5

14.4.5 Addressing

After a START condition the I²C slave address byte is transmitted by the hardware. The I²C slave address is composed of a slave address followed by a read/write bit.

Table 14-4: I²C Slave Address Format

Slave Address Bits		R/W Bit	Description
0000	000	0	General Call Address
0000	000	1	START Condition
0000	001	*x*	CBUS Address
0000	010	*x*	Reserved for different bus format
0000	011	*x*	Reserved for future purposes
0000	1*xx*	*x*	HS-mode master code
1111	1*xx*	*x*	Reserved for future purposes
1111	0*xx*	*x*	10-bit slave addressing

In 7-bit addressing mode, the master sends one address byte. To address a 7-bit address slave, first clear the *I2Cn_MASTER_CTRL.ex_addr_en* field to 0, then write the address to the transmit FIFO formatted as follows where A*n* is address A6:A0.

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

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

In 10-bit addressing mode (*I2Cn_MASTER_CTRL.ex_addr_en* = 1), the first byte the master sends is the 10-bit slave 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 slave. If the operation is a read, it is followed by a repeated START. The software then writes the 10-bit address again with a 1 for the R/W bit. This I²C then starts receiving data from the slave device.

14.4.6 Master Mode Operation

The peripheral operates in master mode when master mode enable *I2Cn_CTRL.mst* = 1. To initiate a transfer, the master generates a START condition by setting *I2Cn_MASTER_CTRL.start* = 1. If the bus is busy, it does not generate a START condition until the bus is available.

A master can communicate with multiple slave devices without relinquishing the bus. Instead of generating a STOP condition after communicating with the first slave, the master generates a Repeated START condition, or RESTART, by setting *I2Cn_MASTER_CTRL.restart* = 1. If a transaction is in progress, the peripheral finishes the transaction before generating a RESTART. The peripheral then transmits the slave address stored in the transmit FIFO. The *I2Cn_MASTER_CTRL.restart* bit is automatically cleared to 0 as soon as the master begins a RESTART condition.

I2Cn_MASTER_CTRL.start is automatically cleared to 0 after the master has completed a transaction and sent a STOP condition.

The master can also generate a STOP condition by setting *I2Cn_MASTER_CTRL.stop* = 1.

If both START and RESTART conditions are enabled at the same time, 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 at the same time, 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 at the same time, a START condition is first generated. At the end of the first transaction, a RESTART condition is generated. The *I2Cn_MASTER_CTRL.stop* bit is cleared and ignored.

A slave cannot generate START, RESTART, or STOP conditions. Therefore, when master mode is disabled, the *I2Cn_MASTER_CTRL*.*start*, *I2Cn_MASTER_CTRL*.*restart*, and *I2Cn_MASTER_CTRL*.*stop* bits are all cleared to 0.

For master 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 master on the bus, then changing the registers outside of a transaction (*I2Cn_MASTER_CTRL*.*start* = 0) satisfies this requirement:

- *I2Cn_CTRL*.*mst_mode*
- *I2Cn_CTRL*.*rx_mode*
- *I2Cn_CTRL*.*scl_pp_mode*
- *I2Cn_CTRL*.*hs_mode*
- *I2Cn_RX_CTRL1*.*cnt*
- *I2Cn_MASTER_CTRL*.*sl_ex_addr*
- *I2Cn_MASTER_CTRL*.*master_code*
- *I2Cn_CLK_LO*.*lo*
- *I2Cn_CLK_HI*.*hi*
- *I2Cn_HS_CLK*.*hsclk_lo*
- *I2Cn_HS_CLK*.*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_TX_CTRL0*.*tx_thresh*
- *I2Cn_RX_CTRL0*.*rx_thresh*
- *I2Cn_TIMEOUT*.*scl_to_val*
- *I2Cn_DMA*.*rx_en*
- *I2Cn_DMA*.*tx_en*
- *I2Cn_FIFO*.*data*
- *I2Cn_MASTER_CTRL*.*start*
- *I2Cn_MASTER_CTRL*.*restart*
- *I2Cn_MASTER_CTRL*.*stop*

14.4.6.1 I²C Master Mode Receiver Operation

When in master mode, initiating a master receiver operation begins with the following sequence:

1. Write the number of data bytes to receive to the I²C receive count field (*I2Cn_RX_CTRL1.cnt*).
2. Write the I²C slave address byte to the *I2Cn_FIFO* register with the R/W bit set to 1.
3. Send a START condition by setting *I2Cn_MASTER_CTRL.start* = 1.
4. The slave address is transmitted by the controller from the *I2Cn_FIFO* register.
5. The I²C controller receives an ACK from the slave and the controller sets the address ACK interrupt flag (*I2Cn_INT_FL0.addr_ack* = 1).
6. The I²C controller receives data from the slave and automatically ACKs each byte. The software must retrieve this data by reading the *I2Cn_FIFO* register.
7. Once *I2Cn_RX_CTRL1.cnt* data bytes have been received, the I²C controller sends a NACK to the slave and sets the Transfer Done Interrupt Status Flag (*I2Cn_INT_FL0.done* = 1).
8. If *I2Cn_MASTER_CTRL.restart* or *I2Cn_MASTER_CTRL.stop* is set, then the I²C controller sends a repeated START or STOP, respectively.

14.4.6.2 I²C Master Mode Transmitter Operation

When in master mode, initiating a master transmitter operation begins with the following sequence:

1. Write the I²C slave address byte to the *I2Cn_FIFO* register with the R/W bit set to 0.
2. Write the desired data bytes to the *I2Cn_FIFO* register, up to the 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 prior to starting the transaction.)
3. Send a START condition by setting *I2Cn_MASTER_CTRL.start* = 1.
4. The controller transmits the slave address byte written to the *I2Cn_FIFO* register.
5. The I²C controller receives an ACK from the slave and the controller sets the address ACK interrupt flag (*I2Cn_INT_FL0.addr_ack* = 1).
6. The *I2Cn_FIFO* register data bytes are transmitted on the SDA line.
   a. The I²C controller receives an ACK from the slave after each data byte.
   b. As the transfer proceeds, the software should refill the transmit FIFO by writing to the *I2Cn_FIFO* register as needed.
   c. If the transmit FIFO goes empty during this process, the controller pauses at the beginning of the byte and waits for the software to either write more data or instruct the controller to send a RESTART or STOP condition.
7. Once the software writes all the desired bytes to the *I2Cn_FIFO* register, the software should set either *I2Cn_MASTER_CTRL.restart* or *I2Cn_MASTER_CTRL.stop*.
8. Once the controller sends all the remaining bytes and empties the transmit FIFO, it sets *I2Cn_INT_FL0.done* and proceeds to send out either a RESTART condition, if *I2Cn_MASTER_CTRL.restart* was set, or a STOP condition, if *I2Cn_MASTER_CTRL.stop* was set.

14.4.6.3 I²C Multi-Master Operation

The I²C protocol supports multiple masters on the same bus. When the bus is free, it is possible that two (or more) masters might try to initiate communication at the same time. This is a valid bus condition. If this occurs and the two masters want to transmit different data and/or address different slaves, only one master can remain in master mode and complete its transaction. The other master must back off transmission and wait until the bus is idle. This process by which the winning master is determined is called bus arbitration.

To determine which master wins the arbitration, for each address or data bit, the master compares the data being transmitted on SDA to the value observed on SDA. If a master attempts to transmit a 1 on SDA (that is, the master lets SDA float) but senses a 0 instead, then that master loses arbitration, and the other master that sent a zero continues with the transaction. The losing master cedes the bus by switching off its SDA and SCL drivers.

Note: This arbitration scheme works with any number of bus masters: if more than two masters begin transmitting simultaneously, the arbitration continues as each master cedes the bus until only one master remains transmitting. Data is not corrupted because as soon as each master realizes it has lost arbitration, it stops transmitting on SDA, leaving the following data bits sent on SDA intact.

If the I²C master peripheral detects it has lost arbitration, it stops generating SCL; sets the *I2Cn_INT_FL0.arb_er* field, sets *I2Cn_INT_FL0.tx_lock_out* field (flushing any remaining data in the transmit FIFO). The hardware also clears the *I2Cn_MASTER_CTRL.start*, the *I2Cn_MASTER_CTRL.restart*, and the *I2Cn_MASTER_CTRL.stop* fields to 0. So long as the peripheral is not itself addressed by the winning master, the I²C peripheral stays in master mode (*I2Cn_CTRL.mst = 1)*. If at any time another master addresses this peripheral using the address programmed in *I2Cn_SLAVE_ADDR.addr*, then the I²C peripheral clears the *I2Cn_CTRL.mst* field to 0 and begins responding as a slave. This can even occur during the same address transmission during which the peripheral lost arbitration.

Note: Arbitration loss is considered an error condition, and like the other error conditions sets I2Cn_INT_FL0.tx_lock_out. Therefore, after an arbitration loss, the software needs to clear I2Cn_INT_FL0.tx_lock_out and reload the transmit FIFO.

Also, in a multi-master environment, the software does *not* need to wait for the bus to become free before attempting to start a transaction (writing 1 to *I2Cn_MASTER_CTRL.start*). If the bus is free when *I2Cn_MASTER_CTRL.start* is set to 1, the transaction begins immediately. If instead the bus is busy, then the peripheral:

1. Waits for the other master to complete the transaction(s) by sending a STOP,
2. Counts out the bus free time using $t_{BUF} = t_{SCL\_LO}$ (see *Equation 14-3), and then*
3. Sends a START condition and begins transmitting the slave address byte(s) in the transmit FIFO, followed by the rest of the transfer.

The I²C master peripheral is compliant with all bus arbitration and clock synchronization requirements of the I²C specification; this operation is automatic, and no additional programming is required.

14.4.7 Slave Mode Operation

When in slave mode, the I²Cn peripheral operates as a slave device on the I²C bus and responds to an external master's requests to transmit or receive data. To configure the I²Cn peripheral as a slave, write the *I2Cn_CTRL.mst* bit to zero. The I²Cn clock is driven by the master on the bus, so the SCL device pin is driven by the external master and *I2Cn_STATUS.clk_mode* remains a zero. The desired slave address must be set by writing to the *I2Cn_SLAVE_ADDR.addr* register.

For slave mode operation, the following register fields should be configured with the I2Cn peripheral disabled:

- *I2Cn_CTRL.mst* = 0 for slave operation.
- *I2Cn_CTRL.gen_call_addr*
- *I2Cn_CTRL.rx_mode*
  - The recommended value for this field is 0. *Note that a setting of 1 is incompatible with slave mode operation with clock stretching disabled (I2Cn_CTRL.scl_clk_stretch_dis = 1).*
- *I2Cn_CTRL.scl_clk_stretch_dis*
- *I2Cn_CTRL.hs_mode*
- *I2Cn_RX_CTRL0.dnr*
  - SMBus/PMBus applications should set this to 0, while other applications should set this to 1.
- *I2Cn_TX_CTRL0.preload*
  - Recommended value is 0 for applications that can tolerate slave clock stretching (*I2Cn_CTRL.scl_clk_stretch_dis* = 0).
  - Recommend value is 1 for applications that do not allow slave clock stretching (*I2Cn_CTRL.scl_clk_stretch_dis* = 1).
- *I2Cn_CLK_HI.hi*
  - Applies to slave mode when clock stretching is enabled (*I2Cn_CTRL.scl_clk_stretch_dis* = 0)
    - This is used to satisfy $t_{SU;DAT}$ after clock stretching; program it so that the value defined by *Equation 14-2* is >= $t_{SU;DAT(min)}$.
- *I2Cn_HS_CLK.hsclk_hi*
  - Applies to slave mode in Hs Mode when clock stretching is enabled (*I2Cn_CTRL.scl_clk_stretch_dis* = 0)
    - This is used to satisfy $t_{SU;DAT}$ after clock stretching during Hs-Mode operation; program it so that the value defined by *Equation 14-6* is >= $t_{SU;DAT(min)}$.
- *I2Cn_SLAVE_ADDR.addr*
- *I2Cn_SLAVE_ADDR.ext_addr_en*

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_TX_CTRL0.tx_thresh* and *I2Cn_RX_CTRL0.rx_thresh*
  - Transmit and receive FIFO threshold levels.
- *I2Cn_TX_CTRL1.tx_ready*
  - Transmit ready (can only be cleared by hardware).
- *I2Cn_TIMEOUT.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.

14.4.7.1 Slave Transmitter

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

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

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

1. With *I2Cn_CTRL*.*i2c_en* = 0, initialize all relevant registers, including specifically for this mode *I2Cn_CTRL*.*clkstr_dis* = 0 and *I2Cn_TX_CTRL0*.*preload_mode* = 0. Don't forget to program *I2Cn_CLK_HI*.*hi* and *I2Cn_HS_CLK*.*hsclk_hi* with appropriate values satisfying $t_{SU;DAT}$ (and HS $t_{SU;DAT}$).
2. The software sets *I2Cn_CTRL*.*i2c_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_INT_FL0*.*addr_match* and *I2Cn_INT_FL0*.*tx_lock_out*.
3. The software waits for *I2Cn_INT_FL0*.*addr_match* =1, either through polling the interrupt flag or setting *I2Cn_INT_EN0*.*addr_match* to interrupt the CPU.
4. After reading *I2Cn_INT_FL0*.*addr_match* =1, 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_INT_FL0*.*tx_lock_out* and loads data into the FIFO.
5. The software clears *I2Cn_INT_FL0*.*addr_match* and *I2Cn_INT_FL0*.*tx_lock_out*. Now that *I2Cn_INT_FL0*.*tx_lock_out* 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_CLK_HI*.*hi*) and sends out the data on the bus.
7. While the master keeps requesting data and sending ACKs, *I2Cn_INT_FL0*.*done* remains 0, and the software should continue to monitor the transmit FIFO and refill it as needed.
    a. The FIFO level can be monitored synchronously through the transmit FIFO status/interrupt flags, or asynchronously by setting *I2Cn_TX_CTRL0*.*tx_thresh* and setting the *I2Cn_INT_EN0*.*tx_thresh* interrupt.
    b. If the transmit FIFO ever empties during the transaction, the hardware starts clock stretching and waits for it to be refilled.
8. The master ends the transaction by sending a NACK. Once this happens, the *I2Cn_INT_FL0*.*done* interrupt flag is set, and the software can stop monitoring the transmit FIFO.
    a. If the software needs to know how many data bytes were transmitted to the master, it should check the transmit FIFO level as soon as *I2Cn_INT_FL0*.*done* = 1 and use it to determine how many data bytes were successfully sent.
        i. Note: Any data remaining in the transmit FIFO is discarded prior to 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_INT_FL0*.*done* interrupt flag, and clear the *I2Cn_INT_FL0*.*tx_thresh* interrupt flag. Return to step 3, waiting on an address match.

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

To use slave transmit preload mode:

1. With *I2Cn_CTRL*.*i2c_en* = 0, initialize all relevant registers, including specifically for this mode *I2Cn_CTRL*.*scl_clk_stretch_dis* = 1 and *I2Cn_TX_CTRL0*.*preload_mode* = 1.
2. The software sets *I2Cn_CTRL*.*i2c_en* = 1.
    a. Even though the controller is enabled, at this point it does not ACK an address match with R/W = 1 until the software sets *I2Cn_TX_CTRL1*.*tx_ready* = 1.

3. The software prepares for the transmit operation by loading data into the transmit FIFO, enabling DMA, setting *I2Cn_TX_CTRL0.tx_thresh* and setting *I2Cn_INT_EN0.tx_thresh* interrupt, etc.
   a. If clock stretching is disabled, 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 *prior* to setting *I2Cn_TX_CTRL1.tx_preload* = 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_TX_CTRL1.tx_preload* = 1.
   a. The controller is now fully enabled and responds with an ACK to an address match.
   b. The hardware sets *I2Cn_INT_FL0.addr_match* when an address match occurs. *I2Cn_INT_FL0.tx_lock_out* is NOT set to 1 and remains 0.
5. The software waits for *I2Cn_INT_FL0.addr_match* = 1, either through polling the interrupt flag or setting *I2Cn_INT_EN0.addr_ack* to interrupt the CPU.
6. After seeing *I2Cn_INT_FL0.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 was preloaded into the transmit FIFO.
   b. Once the first data byte is sent, the hardware automatically clears *I2Cn_TX_CTRL1.tx_preload* to 0.
7. While the master keeps requesting data and sending ACKs, *I2Cn_INT_FL0.done* remains 0 and the software should continue to monitor the transmit FIFO and refill it as needed.
   a. The FIFO level can be monitored synchronously through the transmit FIFO status/interrupt flags, or asynchronously by setting *I2Cn_TX_CTRL0.tx_thresh* and setting *I2Cn_INT_EN0.tx_thresh* interrupt.
   b. If clock stretching is disabled and the transmit FIFO empties during the transaction, the hardware sets *I2Cn_INT_FL1.tx_underflow* = 1 and sends 0xFF for all following data bytes requested by the master.
8. The master ends the transaction by sending a NACK, causing the hardware to set the *I2Cn_INT_FL0.done* interrupt flag.
   a. If the transmit FIFO empties at the same time that the master indicates the transaction is complete by sending a NACK, this is not considered an underrun event, and the *I2Cn_INT_FL1.tx_underflow* flag remains 0.
   b. If the software needs to know how many data bytes were transmitted to the master, check the transmit FIFO level when the *I2Cn_INT_FL0.done* flag is set to 1.
9. The transaction is complete, the software should "clean up", which should include clearing *I2Cn_INT_FL0.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.
      1) If this is not desired, the software can flush the transmit FIFO. Flush the transmit and receive FIFOs by writing 0 to *I2Cn_CTRL.i2c_en* and the writing 1 to *I2Cn_CTRL.i2c_en*.

Once a slave starts transmitting from the *I2Cn_FIFO*, detection of out of sequence STOP, START, or RESTART conditions terminate the current transaction. When a transaction is terminated as a result of an out of sequence error, *I2Cn_INT_FL0.start_er* or *I2Cn_INT_FL0.stop_er* is set to 1.

If the transmit FIFO is not ready (*I2Cn_TX_CTRL1.tx_preload* = 0) and the I²C controller receives a data read request from the master, 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.

14.4.7.2 Slave Receivers

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

Program flow for a receive operation is as follows:

1. With *I2Cn_CTRL*.*i2c_en* = 0, initialize all relevant registers.
2. Set *I2Cn_CTRL*.*i2c_en* = 1.
   a. If an address match with R/W=0 occurs, and the receive FIFO is empty, the peripheral responds with an ACK and the *I2Cn_INT_FL0*.*addr_match* flag is set.
   b. If the receive FIFO is not empty, then depending on the value of *I2Cn_RX_CTRL0*.*dnr*, the peripheral NACKs either the address byte (*I2Cn_RX_CTRL0*.*dnr* = 1) or the first data byte (*I2Cn_RX_CTRL0*.*dnr* = 0).
3. Wait for *I2Cn_INT_FL0*.*addr_match* = 1, either by polling or by enabling the *wr_addr_match* interrupt. Once a successful address match occurs, the hardware sets *I2Cn_INT_FL0*.*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_INT_FL0*.*addr_match*, and while the master keeps sending data, *I2Cn_INT_FL0*.*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_RX_CTRL0*.*rx_thresh* and enabling the *I2Cn_INT_FL0*.*rx_thresh* interrupt.

   b. If the receive FIFO ever fills up during the transaction, then the hardware sets *I2Cn_INT_FL1*.*rx_overflow* and then either:
      i. If *I2Cn_CTRL*.*scl_clk_stretch_dis* = 0, start clock stretching and wait for software to read from the receive FIFO, *or*
      ii. If *I2Cn_CTRL*.*scl_clk_stretch_dis* = 1, respond to the master with a NACK and the last byte is discarded.
6. The master ends the transaction by sending a RESTART or STOP. Once this happens, the*I2Cn_INT_FL0*.*done* interrupt flag is set, and the software can stop monitoring the receive FIFO.
7. Once a slave starts receiving into its receive FIFO, detection of an out of sequence STOP, START, or RESTART condition releases the I^2C bus to the Idle state and the hardware sets the *I2Cn_INT_FL0*.*start_er* field or *I2Cn_INT_FL0*.*stop_er* field to 1 based on the specific condition.

If software has not emptied the data in the receive FIFO from the previous transaction by the time that a master addresses it for another write (i.e., receive) transaction, then the controller does *not* participate in the transaction, and no additional data is written into the FIFO. Although a NACK *is* sent to the master, the software can control if the NACK is sent with the initial address match, or if instead it is sent at the end of the first data byte. Setting *I2Cn_RX_CTRL0*.*dnr* to 1 chooses the former, while setting *I2Cn_RX_CTRL0*.*dnr* to 0 chooses the latter.

14.4.8 Interrupt Sources

The I²C controller has a very flexible interrupt generator that generates an interrupt signal to the interrupt controller on any of several events. On recognizing the I²C interrupt, the software determines the cause of the interrupt by reading the I²C interrupt flags registers *I2Cn_INT_FL0* and *I2Cn_INT_FL1*. Interrupts can be generated for the following events:

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

Interrupts for each event can be enabled or disabled by setting or clearing the corresponding bit in the *I2Cn_INT_EN0* or *I2Cn_INT_EN1* interrupt enable register.

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

Note: Prior to enabling an interrupt, the status of the corresponding interrupt flag should be checked and, if necessary, serviced or cleared. This prevents a previous interrupt event from interfering with a new I²C communications session.

14.4.9 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 have no effect. Reads from *I2Cn_FIFO* dequeue data from the receive FIFO. Writes to a full transmit FIFO have no effect and reads from an empty receive FIFO return 0xFF.

The transmit and receive FIFO only read or write one byte at a time. Transactions larger than 8 bits can still be performed, however. A 16- or 32-bit write to the transmit FIFO stores just the lowest 8 bits of the write data. A 16- or 32-bit read from the receive FIFO has the valid data in the lowest 8 bits and 0's in the upper bits. In any case, the transmit and receive FIFOs only accept 8 bits at a time for either reads or writes.

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 master operation is a READ, and during slave 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 master mode transaction, then the hardware sets the *I2Cn_INT_FL1.rx_overflow* the *I2Cn_INT_FL1.rx_overflow* bit and one of two things happen depending on the value of *I2Cn_CTRL.scl_clk_stretch_dis*:

- If clock stretching is enabled (*I2Cn_CTRL.scl_clk_stretch_dis* = 0), then the hardware stretches the clock until the software makes space available in the receive FIFO by reading from *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 master is free to continue the transaction.
- If clock stretching is disabled (*I2Cn_CTRL.scl_clk_stretch_dis* = 1), then the hardware responds to the master with a NACK and the data byte is lost. The master can return the bus to idle with a STOP condition or start a new transaction with a RESTART condition.

During a transmit transaction (which during master operation is a WRITE, and during slave operation is a READ), either 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 threshold.
- Receive FIFO level greater than or equal to 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.i2c_en* = 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_TX_CTRL0.tx_flush* = 1) or the receive FIFO flush bit (*I2Cn_RX_CTRL0.rx_flush* = 1), respectively. In addition, under certain conditions, the transmit FIFO is automatically locked by 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:

- During operation as a slave transmitter, a NACK is received.
- Any of the following interrupts: arbitration error, timeout error, master mode address NACK error, master mode data NACK error, start error, and stop error. Automatic flushing cannot be disabled for these conditions.

When the above conditions occur, the transmit FIFO is flushed so that data intended for a previous transaction is not transmitted unintentionally for a new transaction. In addition to flushing the transmit FIFO, the transmit lockout flag is set (*I2Cn_INT_FL0.tx_lockout* = 1) and writes to the transmit FIFO are ignored until the software acknowledges the external event by clearing *I2Cn_INT_FL0.tx_lockout*.

14.4.10 Transmit FIFO Preloading

There can be situations during slave mode operation where the software wants to preload the transmit FIFO prior to a transmission, such as when clock stretching is disabled. In this scenario, rather than responding to an external master requesting data with an ACK and clock stretching while 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 master using the transmit ready (*I2Cn_TX_CTRL1.tx_preload*) bit. When *I2Cn_TX_CTRL1.tx_preload* is set to 0, the hardware automatically NACKs all read transactions from the master. Setting *I2Cn_TX_CTRL1.tx_preload* to 1 sends an ACK to the master on the next read transaction and transmits the data in the transmit FIFO. Preloading the transmit FIFO should be complete prior to setting the *I2Cn_TX_CTRL1.tx_preload* 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_TX_CTRL0.tx_preload* to 1. The hardware automatically clears the *I2Cn_TX_CTRL1.tx_ready* field to 0.
2. If the transmit FIFO lockout flag (*I2Cn_INT_FL0.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 master sends the next read request.
5. Set *I2Cn_TX_CTRL1.tx_preload* to 1 to automatically let the hardware send the preloaded FIFO on the next read from a master.
6. *I2Cn_TX_CTRL1.tx_preload* is cleared by the hardware once it finishes transmitting the first byte, and data is transmitted from the transmit FIFO. Once cleared, the application firmware can repeat the preloading process or disable transmit FIFO preloading.

14.4.11 Interactive Receive Mode (IRXM)

In some situations, the I2Cn might want to inspect and respond to each byte of received data. In this case, interactive receive mode (IRXM) can be used. IRXM is enabled by setting *I2Cn_CTRL.rx_mode* = 1. If IRXM is enabled, it must occur before any I^2C 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_INT_FL0.rx_mode* = 1). Application firmware must read the data and generate a response (ACK or NACK) by setting the IRXM Acknowledge (*I2Cn_CTRL.rx_mode_ack*) bit accordingly. Send an ACK by clearing the *I2Cn_CTRL.rx_mode_ack* bit to 0. Send a NACK by setting the *I2Cn_CTRL.rx_mode_ack* bit to 1.

After setting the *I2Cn_CTRL.rx_mode_ack* bit, clear the IRXM interrupt flag. Write 1 to *I2Cn_INT_FL0.rx_mode* to clear the interrupt flag. When the IRXM interrupt flag is cleared, the I2Cn peripheral hardware releases the SCL line and sends the *I2Cn_CTRL.rx_mode_ack* on the SDA line.

While the I2Cn peripheral is waiting for the application firmware to clear the *I2Cn_INT_FL0.rx_mode* flag, the software can disable IRXM and, if operating as a master, load the remaining number of bytes to be received for the transaction. This allows the software to examine the initial bytes of a transaction, which might be a command, and then disable IRXM to receive the remaining bytes in normal operation.

During IRXM, received data is not placed in the receive FIFO. Instead, the *I2Cn_FIFO* address is repurposed to directly read the receive shift register, bypassing the receive FIFO. Therefore, before disabling IRXM, the software must first read the data byte from *I2Cn_FIFO.data.* If the IRXM byte is not read, the byte is lost and the next read from the receive FIFO returns 0xFF.

Note: IRXM 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 slave, clock stretching must remain enabled (I2Cn_CTRL.scl_clk_stretch_dis = 0).

14.4.12 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 I^2C Bus Specification defines the term ‘clock stretching’ to only apply to a slave device holding the SCL line low, this section describes situations where the I2Cn peripheral holds the SCL line low in either slave *or* master 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.rx_mode* = 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 master or slave), when the transmit FIFO is empty, SCL is automatically held low after the ACK bit and before the next data byte begins. To stop clock stretching and continue the transaction, the software must write data to *I2Cn_FIFO.data*. If operating in master mode, however, instead of sending more data, the software can also set either *I2Cn_MASTER_CTRL.stop* or *I2Cn_MASTER_CTRL.restart* to send a STOP or RESTART condition, respectively.

For a receive operation (as either master or slave), when both the receive FIFO and the receive shift register are full, SCL is automatically held low until at least one data byte is read from the receive FIFO. To stop clock stretching and continue the transaction, the software must read data from *I2Cn_FIFO.data*. If operating in master mode and this is the final byte of the transaction, as determined by *I2Cn_RX_CTRL1.cnt*, then the software must also set either *I2Cn_MASTER_CTRL.stop* or *I2Cn_MASTER_CTRL.restart* to send a STOP or RESTART condition, respectively. This must be done in addition to reading from the receive FIFO, since the peripheral cannot start sending the STOP or RESTART until the last data byte has been

moved from the receive shift register into the receive FIFO. This occurs automatically once there is space in the receive FIFO.

*Note: Since some masters do not support other devices stretching the clock, it is possible to completely disable all clock stretching during slave mode by setting I2Cn_CTRL.scl_clk_stretch_dis to 1 and clearing I2Cn_CTRL.*rx_mode *to 0. In this case, instead of clock stretching the I2Cn peripheral sends a NACK if receiving data or sends 0xFF if transmitting data.*

Note: The clock synchronization required to support other I²C master or slave devices stretching the clock is built into the peripheral and requires no intervention from the software to operate correctly.

14.4.13 Bus Timeout

The timeout field, *I2Cn_TIMEOUT.scl_to_val,* is used to detect bus errors. *Equation 14-8* and *Equation 14-9* show equations for calculating the maximum and minimum timeout values based on the value loaded into the *I2Cn_TIMEOUT.scl_to_val* field.

Equation 14-8: 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 14-9*.

Equation 14-9: 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 has 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_INT_FL0.to_er* = 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).

14.4.14 DMA Control

There are independent DMA channels for each transmit FIFO and each receive FIFO. DMA activity is triggered by the transmit FIFO (*I2Cn_TX_CTRL0.tx_thresh*) and receive FIFO (*I2Cn_RX_CTRL0.rx_thresh*) threshold levels.

When the transmit FIFO byte count (*I2Cn_TX_CTRL1.tx_fifo*) is less than or equal to the transmit FIFO threshold level *I2Cn_TX_CTRL0.tx_thresh*, then the DMA transfers data into the transmit FIFO according to the DMA configuration.

To ensure the DMA does not overflow the transmit FIFO, the DMA burst size should be set as follows:

Equation 14-10: DMA Burst Size Calculation for I²C Transmit

$$DMA\ Burst\ Size\ \leq TX\ FIFO\ Depth - I2Cn\_TX\_CTRL0.tx\_thresh = 8 - I2Cn\_TX\_CTRL0.tx\_thresh$$

$$where\ 0\ \leq I2Cn\_TX\_CTRL0.tx\_thresh\ \leq 7$$

Applications trying to avoid transmit underflow and/or clock stretching should use a smaller burst size and higher *I2Cn_TX_CTRL0.tx_thresh* setting. This fills up the FIFO more frequently but increases internal bus traffic.

When the receive FIFO count (*I2Cn_RX_CTRL1.lvl*) is greater than or equal to the receive FIFO threshold level *I2Cn_RX_CTRL0.rx_thresh*, the DMA transfers data out of the receive FIFO according to the DMA configuration. To ensure the DMA does not underflow the receive FIFO, the DMA burst size should be set as follows:

Equation 14-11: 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_RX_CTRL0.rx_thresh*. This results in reading from the Receive FIFO more frequently but increases internal bus traffic.

Note for receive operations, the length of the DMA transaction (in bytes) must be an integer multiple of I2Cn_RX_CTRL0.rx_thresh. Otherwise, the receive transaction ends with some data still in the receive FIFO, but not enough to trigger an interrupt to the DMA, leaving the DMA transaction incomplete. One easy way to ensure this for all transaction lengths is to set burst size to 1 (I2Cn_RX_CTRL0.rx_thresh = 1).

To enable DMA transfers, enable the transmit DMA channel (*I2Cn_DMA.tx_en*) and/or the receive DMA channel (*I2Cn_DMA.rx_en*).

14.5 I²C Registers

See *Table 3-1* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 14-5*. 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 14-5: I²C Registers

Offset	Register Name	Description
[0x0000]	*I2Cn_CTRL*	I²C Control 0 Register
[0x0004]	*I2Cn_STATUS*	I²C Status Register
[0x0008]	*I2Cn_INT_FL0*	I²C Interrupt Flags 0 Register
[0x000C]	*I2Cn_INT_EN0*	I²C Interrupt Enable 0 Register
[0x0010]	*I2Cn_INT_FL1*	I²C Interrupts Flags 1 Register
[0x0014]	*I2Cn_INT_EN1*	I²C Interrupts Enable 1 Register
[0x0018]	*I2Cn_FIFO_LEN*	I²C FIFO Length Register
[0x001C]	*I2Cn_RX_CTRL0*	I²C Receive Control 0 Register
[0x0020]	*I2Cn_RX_CTRL1*	I²C Receive Control 1 Register 1
[0x0024]	*I2Cn_TX_CTRL0*	I²C Transmit Control 0 Register 0
[0x0028]	*I2Cn_TX_CTRL1*	I²C Transmit Control 1 Register 1
[0x002C]	*I2Cn_FIFO*	I²C Transmit and Receive FIFO Register
[0x0030]	*I2Cn_MASTER_CTRL*	I²C Master Mode Register
[0x0034]	*I2Cn_CLK_LO*	I²C Clock Low Time Register
[0x0038]	*I2Cn_CLK_HI*	I²C Clock High Time Register

Offset	Register Name	Description
[0x003C]	*I2Cn_HS_CLK*	I²C Hs-Mode Clock Control Register
[0x0040]	*I2Cn_TIMEOUT*	I²C Timeout Register
[0x0044]	*I2Cn_SLAVE_ADDR*	I²C Slave Address Register
[0x0048]	*I2Cn_DMA*	I²C DMA Enable Register

14.5.1 Register Details

Table 14-6: I²C Control Register

I²C Control			I2Cn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15	hs_mode	R/W	0	**High Speed Mode** Set this field to 1 to enable High-speed mode (Hs-mode) operation. This field must be set to 0 for Standard, Fast or Fast-Plus operation. 0: Hs-mode not enabled. 1: Hs-mode enabled.
14	-	R/W	0	**Reserved**
13	scl_pp_mode	R/W	0	**SCL Push-Pull Mode Enable** Setting this field enables push-pull mode for the SCL hardware pin. This field should not be set unless any external slave device never actively drives SCL low. 0: SCL operates in standard I²C open-drain mode 1: SCL operates in push-pull mode without the need for a pullup resistor. Only recommended when in master mode and external slaves do not drive SCL low.
12	scl_clk_stretch_dis	R/W	0	**SCL Clock Stretch Control** 0: Enable Slave clock stretching 1: Disable Slave clock stretching
11	read	R	0	**Read/Write Bit Status** Returns the logic level of the R/W bit on a received address match (*I2Cn_INT_FL0.addr_match* = 1) or general call match (*I2Cn_INT_FL0.gen_call_addr* = 1). This bit is valid for three SCL clock cycles after the address match status flag is set.
10	sw_out_en	R/W	0	**Software output Enabled** When set, pins SDA and SCL are directly controlled by the fields *I2Cn_CTRL.sda_out* and *I2Cn_CTRL.scl_out*, rather than the I²C controller. Setting this field to 1 enables software bit bang control of I²C. 0: The I²C controller manages the SDA and SCL pins in hardware. 1: SDA and SCL are controller by firmware using the *I2Cn_CTRL.sda_out* and *I2Cn_CTRL.scl_out* fields.
9	sda	RO	-	**SDA Status** Returns the current logic level of the SDA pin. 0: SDA pin is logic low. 1: SDA pin is logic high.
8	scl	RO	-	**SCL Status** Returns the current logic level of the SCL hardware pin. 0: SCL pin is logic low. 1: SCL pin is logic high.

I²C Control			I2Cn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
7	sda_out	R/W	0	**SDA Pin Control** Set the state of the SDA hardware pin (actively pull low or float). 0: Pull SDA Low 1: Release SDA *Note: Only valid when I2Cn_CTRL.sw_out_en = 1*
6	scl_out	R/W	0	**SCL Pin Control** Set the state of the SCL hardware pin (actively pull low or float). 0: Pull SCL low 1: Release SCL *Note: Only valid when I2Cn_CTRL.sw_out_en=1*
5	-	R/W	0	**Reserved**
4	rx_mode_ack	R/W	0	**Interactive Receive Mode (IRXM) Acknowledge** If IRXM is enabled (*I2Cn_CTRL.rx_mode* = 1), this field determines if the hardware sends an ACK or a NACK to an IRM transaction. 0: Respond to IRM with ACK 1: Respond to IRM with NACK
3	rx_mode	R/W	0	**IRXM Enable** When receiving data, 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 *Interactive Receive Mode* section for detailed information. 0: Disabled 1: Enabled *Note: Only set this field when the I²C bus is inactive.*
2	gen_call_addr	R/W	0	**General Call Address Enable** Set this field to 1 to enable General Call Address Acknowledgement. 0: Ignore General Call Address 1: Acknowledge General Call Address
1	mst	R/W	0	**Master Mode Enable** Setting this field to 1 enables Master mode operation for the I²C peripheral. Setting this field to 0 enables the I²C peripheral for Slave mode operation. 0: Slave mode enabled 1: Master mode enabled
0	i2c_en	R/W	0	**I²C Enable** Set this field to 1 to enable the I²C peripheral. 0: I²C peripheral disabled 1: I²C peripheral enabled

Table 14-7: I²C Status Registers

I²C Status			I2Cn_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:12	-	R/W	-	**Reserved**

<table>
<tr><th colspan="3">I²C Status</th><th>I2Cn_STATUS</th><th>[0x0004]</th></tr>
<tr><th>Bits</th><th>Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td>11:8</td><td>status</td><td>RO</td><td>0</td><td>I²C Controller Status
This field indicates the status of the I²C controller.
<table><tr><th>stat</th><th>Controller Status</th></tr><tr><td>0</td><td>Idle</td></tr><tr><td>1</td><td>Master Transmit address</td></tr><tr><td>2</td><td>Master Receive address ACK</td></tr><tr><td>3</td><td>Master Transmit extended address</td></tr><tr><td>4</td><td>Master Receive extended address ACK</td></tr><tr><td>5</td><td>Slave Receive address</td></tr><tr><td>6</td><td>Slave Transmit address ACK</td></tr><tr><td>7</td><td>Slave Receive extended address</td></tr><tr><td>8</td><td>Slave transit extended address ACK</td></tr><tr><td>9</td><td>Transmit data (Master or Slave)</td></tr><tr><td>10</td><td>Receive data ACK (Master or Slave)</td></tr><tr><td>11</td><td>Receive data (Master or Slave)</td></tr><tr><td>12</td><td>Transmit data ACK (Master or Slave)</td></tr><tr><td>13</td><td>NACK stage (Master or Slave)</td></tr><tr><td>14</td><td>Reserved</td></tr><tr><td>15</td><td>Another master is addressing another slave. The transaction is ongoing but the I²C controller is not actively part of the transaction.</td></tr></table></td></tr>
<tr><td>5</td><td>clk_mode</td><td>RO</td><td>0</td><td>SCL Drive Status
This field indicates if an external device is behaving as a master by actively driving the SCL line.
0: External device not driving SCL
1: External device is a Master actively driving the SCL pin</td></tr>
<tr><td>4</td><td>tx_full</td><td>RO</td><td>0</td><td>TX FIFO Full
When set, the TX FIFO is full.
0: TX FIFO is not full
1: TX FIFO full</td></tr>
<tr><td>3</td><td>tx_empty</td><td>RO</td><td>1</td><td>TX FIFO Empty
If set, the TX FIFO is empty.
0: TX FIFO is not empty
1: TX FIFO is empty</td></tr>
<tr><td>2</td><td>rx_full</td><td>RO</td><td>0</td><td>RX FIFO Full
If set, the RX FIFO is full.
0: RX FIFO not full
1: RX FIFO Full</td></tr>
<tr><td>1</td><td>rx_empty</td><td>RO</td><td>1</td><td>RX FIFO Empty
If set, the RX FIFO is empty.
0: RX FIFO is not empty
1: RX FIFO is empty</td></tr>
<tr><td>0</td><td>bus</td><td>RO</td><td>0</td><td>Bus Busy
If set, the I²C bus is active.
0: Bus is idle
1: Bus is busy</td></tr>
</table>

Table 14-8: I²C Interrupt Status Flags Registers 0

I²C Interrupt Status Flags 0			I2Cn_INT_FL0	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15	tx_lock_out	R/W1C	0	**TX FIFO Locked Interrupt Flag** If set, the TX FIFO is locked and writes to the TX FIFO are ignored. This field is set to 1 by hardware to prevent stale data from being transmitted from the TX FIFO. When set, the TX FIFO is automatically flushed. Writes to the TX FIFO are ignored until this flag is cleared. Write 1 to clear. 0: TX FIFO not locked. 1: TX FIFO is locked and all writes to the TX FIFO are ignored.
14	stop_er	R/W1C	0	**Out of Sequence STOP Interrupt Flag** This flag is set if a STOP condition occurs on the I²C Bus out of expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence STOP condition occurred.
13	start_er	R/W1C	0	**Out of Sequence START Interrupt Flag** This flag is set if a START condition occurs on the I²C Bus out of expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence START condition occurred.
12	do_not_resp_er	R/W1C	0	**Slave Mode Do Not Respond Interrupt Flag** This occurs if an address match is made, but the TX FIFO or RX FIFO is not ready. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: I²C address match has occurred and either the TX or RX FIFO is not configured.
11	data_er	R/W1C	0	**Master Mode: Data NACK from External Slave Interrupt Flag** This flag is set by hardware if a NACK is received from a slave on the I²C bus. This flag is only valid if the I2Cn peripheral is configured for master mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Data NACK received from a slave.
10	addr_nack_er	R/W1C	0	**Master Mode: Address NACK from Slave Error Flag** This flag is set by hardware if an Address NACK is received from a slave on the I²C bus. This flag is only valid if the I2Cn peripheral is configured for master mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Address NACK received from a slave.
9	to_er	R/W1C	0	**Timeout Error Interrupt Flag** This occurs when this device holds SCL low longer than the programmed timeout value. Applies to both Master and slave mode. Write 1 to clear. Write 0 has no effect. 0: Timeout error has not occurred. 1: Timeout error occurred.
8	arb_er	R/W1C	0	**Master Mode Arbitration Lost Interrupt Flag** Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: Condition occurred.

I²C Interrupt Status Flags 0			I2Cn_INT_FL0	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
7	addr_ack	R/W1C	0	**Master Mode: Address ACK from External Slave Interrupt Flag** This field is set when a slave address ACK is received. Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: The slave device ACK for the address was received.
6	stop	R/W1C	0	**Slave Mode: STOP Condition Interrupt Flag** This flag is set by hardware when a STOP condition is detected on the I²C bus. Write 1 to clear. Write 0 has no effect. 0: Stop condition has not occurred 1: Stop condition occurred
5	tx_thresh	RO	1	**TX FIFO Threshold Level Interrupt Flag** This field is set by hardware if the number of bytes in the Transmit FIFO is less than or equal to the Transmit FIFO threshold level. Write 1 to clear. This field is automatically cleared by hardware when the TX FIFO contains fewer bytes than the TX threshold level. 0: TX FIFO contains more bytes than the TX threshold level. 1: TX FIFO contains TX threshold level or fewer bytes (Default).
4	rx_thresh	RO	1	**RX FIFO Threshold Level Interrupt Flag** This field is set by hardware if the number of bytes in the Receive FIFO is greater than or equal top the Receive FIFO threshold level. This field is automatically cleared when the RX FIFO contains fewer bytes than the RX threshold setting. 0: RX FIFO contains fewer bytes than the RX threshold level. 1: RX FIFO contains at least RX threshold level of bytes (Default).
3	addr_match	R/W1C	0	**Slave Mode: Address Match Status Interrupt Flag** In slave mode operation, a slave mode address match condition has occurred. Write 1 to clear. Writing 0 has no effect. 0: Slave address match has not occurred. 1: Slave address match occurred.
2	gen_call_addr	R/W1C	0	**Slave Mode: General Call Address Match Received Interrupt Flag** In slave mode operation, a general call address match condition has occurred. Write 1 to clear. Writing 0 has no effect. 0: General call address match has not occurred. 1: General call address match occurred.
1	rx_mode	R/W1C	0	**Interactive Receive Mode Interrupt Flag** Write 1 to clear. Writing 0 is ignored. 0: Interrupt condition has not occurred. 1: Interrupt condition occurred.
0	done	R/W1C	0	**Transfer Complete Interrupt Flag** This flag is set for both Master and Slave mode for both transmit and receive operations on the SCL falling edge after an ACK is received or sent. Write 1 to clear. Writing 0 has no effect. 0: Transfer is not complete. 1: Transfer complete.

Table 14-9: I²C Interrupt Enable 0 Registers

I²C Interrupt Enable 0			I2Cn_INT_EN0	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**

I²C Interrupt Enable 0			I2Cn_INT_EN0	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
15	tx_lock_out	R/W	0	**TX FIFO Locked Out Interrupt Enable** Set this field to enable events for TX FIFO lock events. 0: Interrupt disabled. 1: Interrupt enabled.
14	stop_er	R/W	0	**Out of Sequence STOP condition detected Interrupt Enable** Set this field to enable events for an out of sequence STOP condition event. 0: Interrupt disabled. 1: Interrupt enabled.
13	start_er	R/W	0	**Out of Sequence START condition detected Interrupt Enable** Set this field to enable events for an out of sequence START condition event. 0: Interrupt disabled. 1: Interrupt enabled.
12	do_not_resp_er	R/W	0	**Slave Mode Do Not Respond Interrupt Enable** Set this field to enable events in slave mode when the Do Not Respond condition occurs. 0: Interrupt disabled. 1: Interrupt enabled.
11	data_er	R/W	0	**Master Mode Received Data NACK from Slave Interrupt Enable** Set this field to enable events for master mode external device data NACK events. 0: Interrupt disabled. 1: Interrupt enabled.
10	addr_er	R/W	0	**Master Mode Received Address NACK from Slave Interrupt Enable** Set this field to enable events for master mode slave device address NACK events. 0: Interrupt disabled. 1: Interrupt enabled.
9	to_er	R/W	0	**Timeout Error Interrupt Enable** Set this field to enable events for a timeout error interrupt event. 0: Interrupt disabled. 1: Interrupt enabled.
8	arb_er	R/W	0	**Master Mode Arbitration Lost Interrupt Enable** Set this field to enable events in master mode for arbitration lost events. 0: Interrupt disabled. 1: Interrupt enabled.
7	addr_ack	R/W	0	**Received Address ACK from Slave Interrupt Enable** Set this field to enable events for master mode slave device address ACK events. 0: Interrupt disabled. 1: Interrupt enabled.
6	stop	R/W	0	**STOP Condition Detected Interrupt Enable** Set this field to enable interrupt events for STOP conditions. 0: Interrupt disabled. 1: Interrupt enabled.
5	tx_thresh	R/W	0	**TX FIFO Threshold Level Interrupt Enable** Set this field to enable interrupt events when a TX FIFO threshold event occurs. 0: Interrupt disabled. 1: Interrupt enabled.

I²C Interrupt Enable 0			I2Cn_INT_EN0	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
4	rx_thresh	R/W	0	**RX FIFO Threshold Level Interrupt Enable** Set this field to enable interrupt events when an RX FIFO threshold event occurs. 0: Interrupt disabled. 1: Interrupt enabled.
3	addr_match	R/W	0	**Slave Mode Incoming Address Match Interrupt Enable** Set this field to enable the slave mode address match interrupt event. 0: Interrupt disabled. 1: Interrupt enabled.
2	gen_ctrl_addr	R/W	0	**Slave Mode General Call Address Match Received Interrupt Enable** Set this field to enable the slave mode general call address match received interrupt event. 0: Interrupt disabled. 1: Interrupt enabled.
1	rx_mode	R/W	0	**Interactive Receive Interrupt Enable** Set this field to enable the interactive receive interrupt event. 0: Interrupt disabled. 1: Interrupt enabled.
0	done	R/W	0	**Transfer Complete Interrupt Enable** Set this field to enable the transfer complete interrupt event. 0: Interrupt disabled. 1: Interrupt enabled.

Table 14-10: I²C Interrupt Status Flags 1 Registers

I²C Interrupt Status Flags 1			I2Cn_INT_FL1	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	R/W	-	**Reserved**
1	tx_underflow	R/W1C	0	**Slave Mode: TX FIFO Underflow Status Flag** In slave mode operation, the hardware sets this flag automatically if the TX FIFO is empty and the master requests more data by sending an ACK after the previous byte transferred. 0: Slave mode TX FIFO underflow condition has not occurred. 1: Slave mode TX FIFO underflow condition occurred.
0	rx_overflow	R/W1C	0	**Slave Mode: RX FIFO Overflow Status Flag** In slave mode operation, the hardware sets this flag automatically when an RX FIFO overflow occurs. Write 1 to clear. Writing 0 has no effect. 0: Slave mode RX FIFO overflow event has not occurred. 1: Slave mode RX FIFO overflow condition occurred (data lost).

Table 14-11: I²C Interrupt Enable Registers 1

I²C Interrupt Enable 1			I2Cn_INT_EN1	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	R/W	0	**Reserved**
1	tx_underflow	R/W	0	**Slave Mode TX FIFO Underflow Interrupt Enable** In slave mode operation, set this field to enable the TX FIFO underflow interrupt. 0: Interrupt disabled. 1: Interrupt enabled.

I²C Interrupt Enable 1			I2Cn_INT_EN1	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
0	rx_overflow	R/W	0	**Slave Mode RX FIFO Overflow Interrupt Enable** In slave mode operation, set this field to enable the RX FIFO overflow interrupt. 0: Interrupt disabled. 1: Interrupt enabled.

Table 14-12: I²C FIFO Length Registers

I²C FIFO Length			I2Cn_FIFO_LEN	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15:8	tx_len	RO	8	**TX FIFO Length** Returns the length of the TX FIFO. 8: 8-byte TX FIFO.
7:0	rx_len	RO	8	**RX FIFO Length** Returns the length of the RX FIFO. 8: 8-byte RX FIFO.

Table 14-13: I²C Receive Control Registers 0

I²C Receive Control 0			I2Cn_RX_CTRL0	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:12	-	R/W	0	**Reserved**
11:8	rx_thresh	R/W	0	**RX FIFO Threshold Level** Set this field to the required number of bytes to trigger a RX FIFO threshold event. When the number of bytes in the RX FIFO is equal to or greater than this field, the hardware sets the *I2Cn_INT_FL0.rx_thresh* bit indicating an RX FIFO threshold level event. 0: 0 bytes or more in the RX FIFO causes a threshold event. 1: 1+ bytes in the RX FIFO triggers a receive threshold event (recommended minimum value). … 8: RX FIFO threshold event only occurs when the RX FIFO is full.
7	rx_flush	R/W1O	0	**Flush RX FIFO** Write 1 to this field to initiate a RX FIFO flush, clearing all data in the RX FIFO. This field is automatically cleared by hardware when the RX FIFO flush completes. Writing 0 has no effect. 0: RX FIFO flush complete or not active. 1: Flush the RX FIFO
6:1	-	R/W	0	**Reserved**
0	dnr	R/W	0	**Do Not Respond** Slave mode operation only. 0: If the RX FIFO contains data and an external master requests a WRITE transaction, respond to an address match with an ACK but NACK the subsequent data byte(s). (No additional data is written into the RX FIFO.) 1: If the RX FIFO contains data and a master requests a WRITE transaction, do not respond to an address match and send a NACK instead.

Table 14-14: I²C Receive Control 1 Registers

I²C Receive Control 1			I2Cn_RX_CTRL1	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:12	-	R/W	0	**Reserved**
11:8	rx_fifo	R	0	**RX FIFO Byte Count** Returns the number of bytes currently in the RX FIFO. 0: No data in the RX FIFO. ... 8: 8 bytes in the RX FIFO (max value).
7:0	rx_cnt	R/W	1	**RX FIFO Transaction Byte Count** When in master mode, write the number of bytes to be received in a transaction from 1 to 256. 0 represents 256. 0: 256 byte receive transaction. 1: 1 byte receive transaction. 2: 2 byte receive transaction. … 255: 255 byte receive transaction.

Table 14-15: I²C Transmit Control Registers 0

I²C Transmit Control 0			I2Cn_TX_CTRL0	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:12	-	R/W	-	**Reserved**
11:8	tx_thresh	R/W	0	**TX FIFO Threshold Level** Sets the level for a Transmit FIFO threshold event interrupt. If the number of bytes remaining in the TX FIFO falls to this level or lower the interrupt flag *I2Cn_INT_FL0.tx_thresh* is set indicating a TX FIFO Threshold Event occurred. 0: 0 bytes remaining in the TX FIFO triggers a TX FIFO threshold event. 1: 1 byte or less remaining in the TX FIFO triggers a TX FIFO threshold event (recommended minimum value). ... 7: 7 or fewer bytes remaining in the TX FIFO triggers a TX FIFO threshold event
7	tx_flush	R/W1O	0	**TX FIFO Flush** Write this field to 1 to initiate a TX FIFO flush, clearing all remaining data from the transmit FIFO. 0: TX FIFO flush is complete or not active. 1: Flush the TX FIFO *Note: Hardware automatically clears this bit to 0 after it is written to 1 when the flush is completed.* *Note: If I2Cn_INT_FL0.tx_lockout = 1, then* I2Cn_TX_CTRL0*.tx_flush = 1.*
6:2	-	R/W	0	**Reserved**
1	tx_ready_mode	R/W	0	**Transmit FIFO Ready Manual Mode** 0: The hardware control *I2Cn_TX_CTRL1.tx_ready*. 1: The software controls the *I2Cn_TX_CTRL1.tx_ready.*
0	tx_preload	R/W	0	**TX FIFO Preload Mode Enable** 0: Normal operation. An address match in slave mode, or a general call address match, flushes and locks the TX FIFO so it cannot be written and sets *I2Cn_INT_FL0.tx_lockout.* 1: TX FIFO preload mode. An address match in slave mode, or a general call address match, does not lock the TX FIFO and does not set *I2Cn_INT_FL0.tx_lockout.* This allows firmware to preload data into the TX FIFO. The status of the I²C is controllable using *I2Cn_TX_CTRL1.txrdy.*

Table 14-16: I²C Transmit Control Registers 1

I²C Transmit Control 1			I2Cn_TX_CTRL1	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:12	-	R/W	0	**Reserved**
11:8	tx_fifo	RO	0	**Transmit FIFO Byte Count Status** Contains the number of bytes remaining in the TX FIFO
7:2	-	R/W	0	**Reserved**
1	tx_last	R/W1O	0	**Slave Mode Transmit Last** This bit decides what to do if the I²C is in slave mode, is transmitting data to a master, and the TX FIFO is empty. 0: Hold SCL low. This pauses transmission until data is written to the TX FIFO. 1: End transaction by releasing SCL. Cleared on a STOP/RESTART condition, or if *I2Cn_INT_FL0.tx_lockout* = 1 (transmit FIFO locked for writing).
0	tx_ready	R/W1O	1	**Transmit FIFO Preload Ready Status** When transmit FIFO preload mode is enabled, *I2Cn_TX_CTRL1.tx_preload* = 1, this bit is automatically cleared to 0. While this bit is 0, if the I2Cn hardware receives a slave 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 slave address match. When transmit FIFO preload mode is disabled, *I2Cn_TX_CTRL1.tx_preload* = 0, this bit is forced to 1 and the I2Cn hardware behaves normally.

Table 14-17: I²C Data Registers

I²C Data			I2Cn_FIFO	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	R/W	0	**Reserved**
7:0	data	R/W	0xFF	**I²C FIFO Data Register** Reading from this register pops data from the RX FIFO and writes to this register pushes data onto the TX FIFO. If the RX FIFO is empty, reading this field returns 0xFF. *Note: If the RX FIFO is empty, reads from this field return 0xFF.* *Note: If the TX FIFO is full, writes to this field are ignored.*

Table 14-18: I²C Master Mode Control Registers

I²C Master Mode Control			I2Cn_MASTER_CTRL	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved**
10:8	master_code	R/W	0	
7	sl_ex_addr	R/W	0	**Slave Extended Addressing** 0: Send a 7-bit address to the slave 1: Send a 10-bit address to the slave
6:3	-	R/W	0	**Reserved**
2	stop	R/W1O	0	**Send STOP Condition** 0: Stop condition completed or inactive. 1: Send a STOP Condition *Note: This bit is automatically cleared by hardware when the STOP condition begins.*

I²C Master Mode Control			I2Cn_MASTER_CTRL	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
1	restart	R/W1O	0	**Send Repeated START Condition** After sending data to a slave, instead of sending a STOP condition the master can send another START to retain control of the bus. 0: Repeated start condition complete or inactive. 1: Send a Repeated START *Note: This bit is automatically cleared by hardware when the repeated START condition begins.*
0	start	R/W1O	0	**Start Master Mode Transfer** 0: Master mode transfer inactive. 1: Start master mode Transfer *Note: This bit is automatically cleared by hardware when the transfer is completed or aborted.*

Table 14-19: I²C SCL Low Control Register

I²C Clock Low Control			I2Cn_CLK_LO	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:9	-	R/W	0	**Reserved**
8:0	scl_lo	R/W	1	**Clock Low Time** In master mode, this configures the SCL low time. $t_{SCL\_LOW} = f_{I2C\_CLK} \times (scl\_lo + 1)$ *Note: 0 is not a valid setting for this field.*

Table 14-20: I²C SCL High Control Register

I²C Clock High Control			I2Cn_CLK_HI	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**
31:9	-	R/W	0	**Reserved**
8:0	scl_hi	R/W	1	**Clock High Time** In master mode, this configures the SCL high time. $t_{SCL\_HIGH} = {}^{1}/_{f_{I2C\_CLK}} \times (scl\_hi + 1)$ In both Master and slave mode, this also configures the time SCL is held low after new data is loaded from the TX FIFO or after firmware clears *I2Cn_INT_FL0.rx_mode* during interactive receive mode. *Note: 0 is not a valid setting for this field.*

Table 14-21: I²C Timeout Registers

I²C Timeout			I2Cn_HS_CLK	[0x003C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15:8	hs_clk_hi	R/W	0	**Hs-Mode Clock High Time** This field sets the Hs-Mode clock low count. In Slave mode, this is the time SCL is held low after data is output on SDA. The following equation defined the clock low time. $t_{HS\_CLK\_HI} = {}^{1}/_{f_{I2C\_CLK}} \times (hs\_clk\_hi + 1)$ *Note: Refer to section SCL Clock Generation for Hs-mode for details on the requirements for the Hs-mode clock high and low times.*

I²C Timeout			I2Cn_HS_CLK	[0x003C]
Bits	**Name**	**Access**	**Reset**	**Description**
7:0	hs_clk_lo	R/W	0	**Hs-Mode Clock Low Time** This field sets the Hs-Mode clock low count. In Slave mode, this is the time SCL is held low after data is output on SDA. The following equation defined the clock low time. $t_{HS\_CLK\_LO} = {}^{1}/_{f_{I2C\_CLK}} \times (hs\_clk\_lo + 1)$ *Note: Refer to section SCL Clock Generation for Hs-mode for details on the requirements for the Hs-mode clock high and low times.*

Table 14-22: I²C Timeout Registers

I²C Timeout			I2Cn_TIMEOUT	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15:0	to	R/W	0	**Bus Error SCL Timeout Period** Set this value to the number of I²C clock cycles desired to cause a bus timeout error. The I2Cn peripheral timeout timer starts when it pulls SCL low. After the I2Cn peripheral releases the line, if the line is not pulled high prior to the timeout number of I²C clock cycles, a bus error condition is set (*I2Cn_INT_FL0.to_er* = 1) and the I2Cn peripheral releases the SCL and SDA lines 0: Timeout disabled. All other values result in a timeout calculation of: $t_{BUS\_TIMEOUT} = {}^{1}/_{f_{I2C\_CLK}} \times to$ *Note: The timeout counter monitors the I2Cn controller's driving of the SCL pin, not an external I²C master driving the SCL pin.*

Table 14-23: I²C Slave Address Register

I²C Slave Address			I2Cn_SLAVE_ADDR	[0x0044]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15	ea	R/W	0	**Slave Mode Extended Address Select** When this I²C is operating in slave mode, this bit selects whether sla contains a 7-bit or 10-bit address. 0: 7-bit addressing 1: 10-bit addressing
14:10	-	R/W	0	**Reserved**
9:0	sla	R/W	0	**Slave Mode Slave Address** When I²C peripheral is operating in slave mode, *I2Cn_CTRL.mst* = 0, this field must be set as the desired slave address for the peripheral.

Table 14-24: I²C DMA Register

I²C DMA			I2Cn_DMA	[0x0048]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	R/W	0	**Reserved**
1	rxen	R/W	0	**RX DMA Channel Enable** 0: Disable RX DMA channel 1: Enable RX DMA channel

I²C DMA			I2Cn_DMA	[0x0048]
Bits	**Name**	**Access**	**Reset**	**Description**
0	txen	R/W	0	**TX DMA Channel Enable** 0: Disable TX DMA channel 1: Enable TX DMA channel

15. Serial Peripheral Interface (SPI): SPI0 (SPI17Y)

The Serial Peripheral Interface (SPI) is a highly configurable, flexible, and efficient synchronous interface between multiple SPI devices on a single bus. The SPI bus uses a single clock signal, and a single data line, and a slave select lines for communication with external SPI devices.

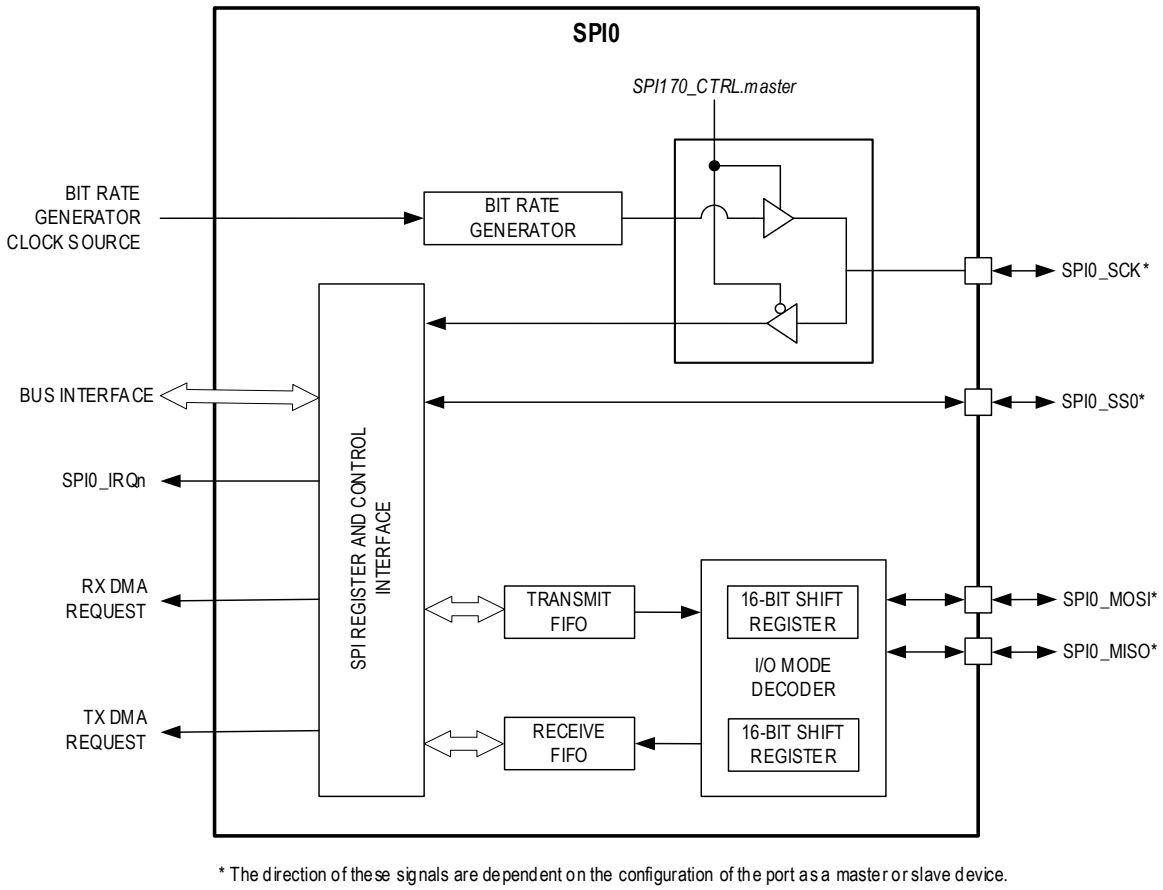
The provided SPI ports support full-duplex, bi-direction I/O and each SPI includes a Bit Rate Generator (BRG) for generating the clock signal when operating in master mode. Each SPI port operates independently and requires minimal processor overhead. All instances of the SPI peripheral support both master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

Features include:

- Dedicated bit rate generator for precision serial clock generation in master mode
  - Up to $\frac{f_{PCLK}}{2}$ for instances on the APB bus.
  - Up to $\frac{f_{HCLK}}{2}$ for instances on the AHB bus.
  - Programmable SCK duty cycle timing.
- Full-duplex, synchronous communication of 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. *SPI17Y_CLK_CFG.scale* must be > 0.
- 3-wire and 4-wire SPI operation for single-bit communication.
- Byte-wide transmit and receive FIFOs with 32-byte depth.
  - For character sizes greater than 8, each character uses 2 entries per character resulting in 16 entries for the transmit and receive FIFO.
- Transmit and receive DMA support.
- SPI modes 0, 1, 2, 3.
- Configurable slave select lines:
  - Programmable slave select level.
- Programmable slave select timing with respect to SCK starting edge and ending edge.
- Multi-master mode fault detection.

Figure 15-1 shows a high-level block diagram of the SPI peripheral. See *Table 15-1* for the peripheral-specific peripheral bus assignment and bit rate generator clock source.

Figure 15-1: SPI0 Block Diagram



* The direction of these signals are dependent on the configuration of the port as a master or slave device.

15.1 Instances

There are two instances of the SPI peripheral as shown in *Table 15-1*. *Table 15-2* lists the locations of the SPI signals for each of the SPI instances.

Table 15-1: SPI0 Instances

Instance	Formats		Hardware Bus	Bit Rate Generator Clock Source
	3-Wire	4-Wire		
SPI0	Yes	Yes	APB	f_{PCLK}

Note: Refer to the MAX32660 data sheet for the definitive list of alternate function assignments for each peripheral.

Table 15-2: SPI0 Peripheral Pins

Instance	Signal Description	Alternate Function	Alternate Function Number	16-WLP	20-TQFN 24-TQFN
SPI0	SPI Clock	SPI0_SCK	AF1	P0.6	P0.6
	MOSI	SPI0_MOSI	AF1	P0.5	P0.5
	MISO	SPI0_MISO	AF1	P0.4	P0.4
	Slave Select	SPI0_SS0	AF1	P0.7	P0.7

15.2 Formats

15.2.1 Four-Wire SPI

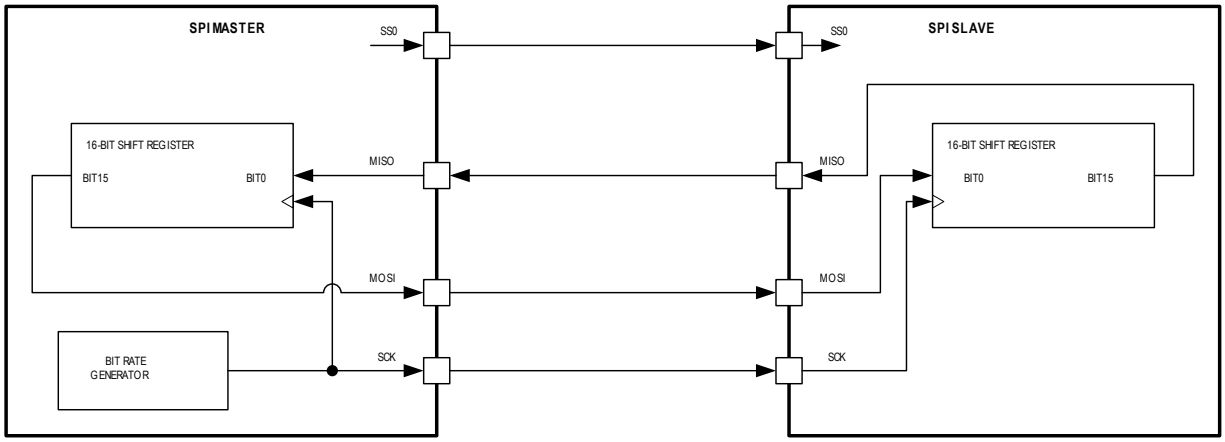
SPI devices operate as either a master or slave device. In four-wire SPI, four signals are required for communication as shown in *Table 15-3*.

Table 15-3: Four-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The master generates the Serial Clock signal, which is an output from the master and an input to the slave.
MOSI	Master Output Slave Input	In master mode, this signal is used as an output for sending data to the slave. In slave mode this is the input data from the master.
MISO	Master Input Slave Output	In master mode, this signal is used as an input for receiving data from the slave. In slave mode, this signal is an output for transmitting data to the master.
SS	Slave Select	In master mode, this signal is an output used to select a slave device prior to communication. Peripherals can have multiple slave select outputs to communicate with one or more external slave devices. In slave mode, SPI0_SS0 is a dedicated input that indicates an external master is going to start communication. Other slave select signals into the peripheral are ignored in slave mode.

The SPI master starts communication with a slave by asserting the slave select output. The master then starts the SPI clock through the SCK output pin. When a slave device's slave select pin is deasserted, the slave device is required to put the SPI pins in tri-state mode.

Figure 15-2: 4-Wire SPI Connection Diagram



15.2.2 Three-Wire SPI

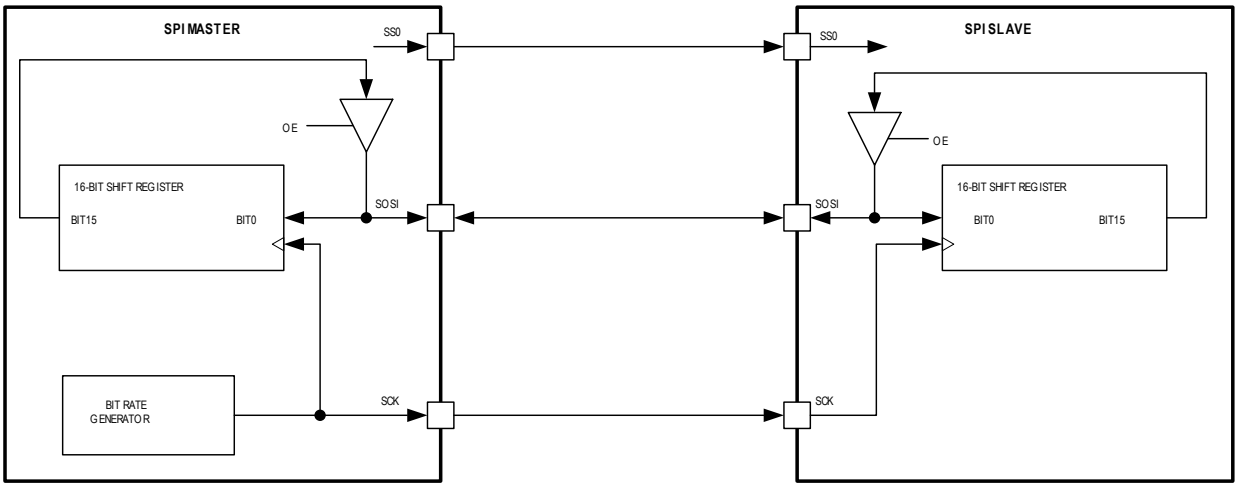
The signals in three-wire SPI operation are shown in *Table 15-4*, The MOSI signal is used as a bidirectional, half-duplex I/O referred to as slave input slave output (SISO). Three-wire SPI also uses a serial clock signal generated by the master and a slave select pin controlled by the master.

Table 15-4: Three-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The master generates the serial clock signal, which is an output from the master and an input to the slave.
MOSI	Slave Input Slave Output	This is a half-duplex, bidirectional I/O pin used for communication between the SPI master and slave. This signal is used to transmit data from the master to the slave and to receive data from the slave by the master.
SS	Slave Select	In master mode, this signal is an output used to select a slave device prior to communication. In slave mode, SPI0_SS0 is a dedicated input that indicates an external master is going to start communication. Other slave select signals into the peripheral are ignored in slave mode

A three-wire SPI network is shown in *Figure 15-3*. The master device selects the slave device using the slave select output. The communication starts with the master asserting the slave select line and then starting the clock (SCK). In three-wire SPI communication, the master and slave must both know the intended direction of the data to prevent bus contention. For a write, the master drives the data out the SISO pin. For a read, the master must release the SISO line and let the slave 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 15-3: Generic 3-Wire SPI Master to Slave Connection



15.3 Pin Configuration

Before configuring the SPI peripheral, first disable any SPI activity for the port by clearing the *SPI17Y_CTRL0.en* field to 0.

15.3.1 SPI0 Alternate Function Mapping

Pin selection and configuration is required to use the SPI port. The following information applies to SPI master and slave operation as well as three-wire and four-wire communications. Determine the pins required for the SPI type and mode in the application and configure the required GPIO as described in the following sections. Refer to the device data sheet for pin availability for a specific package.

When the SPI port is disabled, *SPI17Y_CTRL0.en* = 0, the GPIO pins enabled for SPI alternate function are placed in high-impedance input mode.

15.3.2 Four-Wire Format Configuration

Four-wire SPI uses SCK, MISO, MOSI, and one or more SS pins. Four-wire SPI can use more than one slave 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.

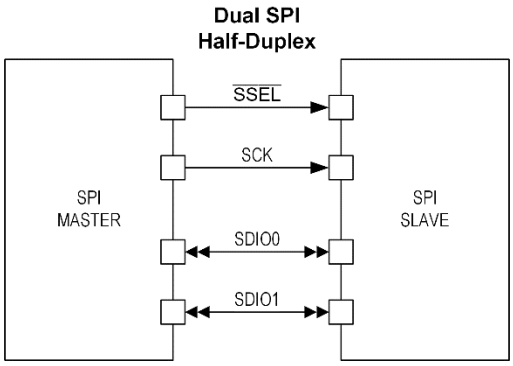
15.3.3 Three-Wire Format Configuration

Three-wire SPI uses SCK and MOSI, and one or more slave select pins for an SPI transaction. Three-wire SPI configuration is identical to the four-wire configuration except SPI0_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 SPI0 transmit and receive FIFO enables. Enabling the receive FIFO (*SPI17Y_DMA.rx_fifo_en* = 1) and disabling the transmit FIFO (*SPI17Y_DMA.tx_fifo_en* = 0) indicates a read transaction. Enabling the transmit FIFO (*SPI17Y_DMA.tx_fifo_en* = 1)and disabling the receive FIFO (*SPI17Y_DMA.rx_fifo_en* = 1) indicates a write transaction. It is an illegal condition to enable both the transmit and receive FIFOs in three-wire SPI operation.

15.3.4 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 master and slave for a given transaction. Dual-mode SPI uses SCK, SDIO0, SDIO1, and one or more slave select lines as shown in *Figure 15-4*. The configuration of the GPIO pins for dual-mode SPI is identical to four-wire SPI and the mode is controlled by setting *SPI17Y_CTRL2.data_width* to 1, indicating to the SPIn hardware to use SDIO0 and SDIO1 for half-duplex communication rather than full-duplex communication.

Figure 15-4: Dual Mode SPI Connection Diagram



15.4 Clock Configuration

15.4.1 Serial Clock

The SCK signal synchronizes data movement in and out of the device. The master drives SCK as an output to the slave's SCK pin. When SPI0 is set to master mode, the SPI0 bit rate generator creates the serial clock and outputs it on the configured SPI0_SCK pin. When SPI0 is configured for slave operation, the SPI0_SCK pin is an input from the external master and the SPI0 hardware synchronizes communications using the SCK input. Operating as a slave, if a SPI0 slave select input is not asserted, the SPI0 ignores any signals on the serial clock and serial data lines.

In both master and slave devices, data is shifted on one edge of the SCK and is sampled on the opposite edge where data is stable. Data availability and sampling time is controlled using the SPI phase control field, *SPI17Y_CTRL2.cpha*. The SCK clock polarity field, *SPI17Y_CTRL2.cpol*, controls if the SCK signal is idle high or idle low.

The SPI0 peripheral supports four combinations of SCK phase and polarity referred to as SPI modes 0, 1, 2, and 3. Clock Polarity (*SPI17Y_CTRL2.cpol*) selects an idle low/high clock and has no effect on the transfer format. Clock Phase (*SPI17Y_CTRL2.cpha*) selects one of two different transfer formats.

For proper data transmission, the clock phase and polarity must be identical for the SPI master and slave. The master always places data on the MOSI line a half-cycle before the SCK edge for the slave to latch the data. See section *Clock Phase and Polarity Control* for additional details.

15.4.2 Peripheral Clock

See *Table 15-1* 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 SPI0 peripheral clock. The SPI0 provides an internal clock, *SPI0_CLK*, that is used within the SPI peripheral for the base clock to control the module and generate the SCK clock when in master mode. Set the SPI0 internal clock using the field *SPI17Y_CLK_CFG.scale* as shown in *Equation 15-1*. Valid settings for *SPI17Y_CLK_CFG.scale* are 0 to 8, allowing a divisor of 1 to 256.

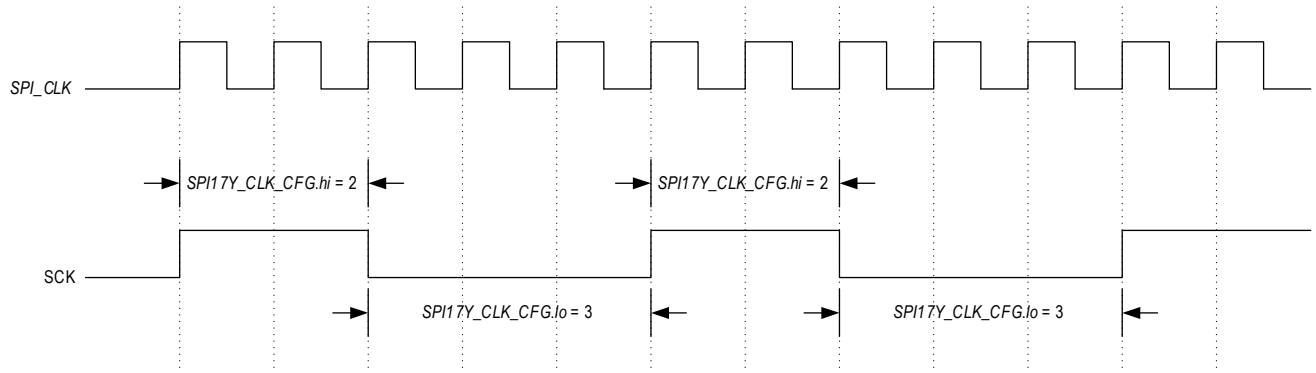
Equation 15-1: SPI Peripheral Clock

$$f_{SPI0\_CLK} = \frac{f_{INPUT\_CLK}}{2^{scale}}$$

15.4.3 Master Mode Serial Clock Generation

In master and multi-master mode, the SCK clock is generated by the master. The SPI0 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 SPI0 peripheral clock as a base value, and the high and low values are a count of the number of f_{SPI0_CLK} clocks. *Figure 15-5*, visually represents the use of the *SPI17Y_CLK_CFG.hi* and *SPI17Y_CLK_CFG.lo* fields for a non-50% duty cycle serial clock generation. See *Equation 15-2* and *Equation 15-3* for calculating the SCK high and low time from the *SPI17Y_CLK_CFG.hi* and *SPI17Y_CLK_CFG.lo* field values.

Figure 15-5: SCK Clock Rate Control



Equation 15-2: SCK High Time

$$t_{SCK\_HI} = t_{SPI0\_CLK} \times SPI17Y\_CLK\_CFG.hi$$

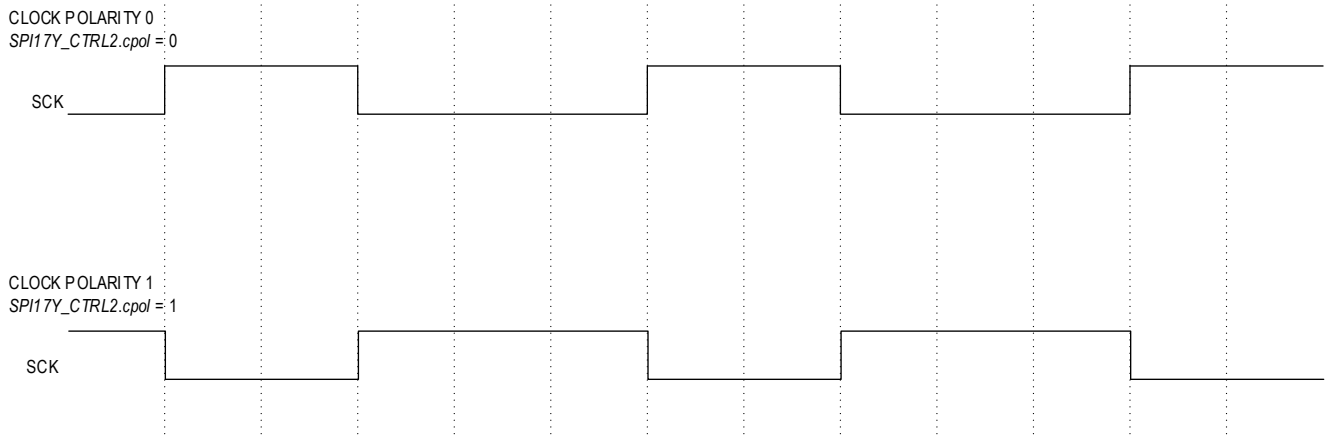
Equation 15-3: SCK Low Time

$$t_{SCK\_LOW} = t_{SPI0\_CLK} \times SPI17Y\_CLK\_CFG.lo$$

15.4.4 Clock Phase and Polarity Control

SPI0 supports four combinations of clock and phase polarity as shown in *Table 15-5*. Clock polarity is controlled using the bit *SPI17Y_CTRL2.cpol* and determines if the clock is idle high or idle low as shown in *Figure 15-6*. Clock polarity does not affect the transfer format for SPI. Clock phase determines when the data must be stable for sampling. Setting the clock phase to 0, *SPI17Y_CTRL2.cpha* = 0, dictates the SPI data is sampled on the initial SPI clock edge regardless of clock polarity. Phase 1, *SPI17Y_CTRL2.cpha* = 1, results in data sample occurring on the second edge of the clock regardless of clock polarity.

Figure 15-6: SPI Clock Polarity



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

Table 15-5: SPI Modes Clock Phase and Polarity Operation

SPI Mode	SPI17Y_CTRL2 cpha	SPI17Y_CTRL2 cpol	SCK Transmit Edge	SCK Receive Edge	SCK Idle State
0	0	0	Falling	Rising	Low
1	0	1	Rising	Falling	High
2	1	0	Rising	Falling	Low
3	1	1	Falling	Rising	High

15.4.5 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.

15.4.6 Interrupts and Wakeups

SPI0 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
  - Slave mode only, master mode stalls the serial clock
- Transmit FIFO Overrun
- Receive FIFO Underrun
- Receive FIFO Overrun
  - Slave mode only, master mode stalls the serial clock
- SPI0 supports interrupts for the internal state of the SPI as well as external signals. The following transmission interrupts are supported:
  - SS0 asserted or deasserted
- SPI transaction complete
  - Master mode only
- Slave mode transaction aborted
- Multi-master fault

The SPI0 port can wake up the microcontroller from *SLEEP* when the wake event is enabled. SPI0 events that can wake the microcontroller are:

- Receive FIFO full
- Transmit FIFO empty
- Receive FIFO threshold
- Transmit FIFO threshold

15.5 SPI0 Registers

See *Table 3-1* 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 15-6: SPI0 Master Register Addresses and Descriptions

Offset	Register Name	Description
[0x0000]	*SPI17Y_DATA*	*SPI0 FIFO Data Register*
[0x0000]	*SPI17Y_FIFO16*	*SPI0 16-bit FIFO Data Register*
[0x0000]	*SPI17Y_FIFO8*	*SPI0 8-bit FIFO Data Register*
[0x0004]	*SPI17Y_CTRL0*	*SPI0 Master Signals Control Register*
[0x0008]	*SPI17Y_CTRL1*	*SPI0 Transmit Packet Size Register*
[0x000C]	*SPI17Y_CTRL2*	*SPI0 Static Configuration Register*
[0x0010]	*SPI17Y_SS_TIME*	*SPI0 Slave Select Timing Register*
[0x0014]	*SPI17Y_CLK_CFG*	*SPI0 Master Clock Configuration Register*
[0x001C]	*SPI17Y_DMA*	*SPI0 DMA Control Register*
[0x0020]	*SPI17Y_INT_FL*	*SPI0 Interrupt Flag Register*
[0x0024]	*SPI17Y_INT_EN*	*SPI0 Interrupt Enable Register*
[0x0028]	*SPI17Y_WAKE_FL*	*SPI0 Wakeup Flags Register*
[0x002C]	*SPI17Y_WAKE_EN*	*SPI0 Wakeup Enable Register*
[0x0030]	*SPI17Y_STAT*	*SPI0 Status Register*

15.5.1 Register Details

Table 15-7: SPI0 FIFO32 Register

SPI0 FIFO Data			SPI17Y_DATA	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**SPI0 FIFO Data Register** This register is used for the SPI0 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 15-8: SPI0 16-bit FIFO Register

SPI0 FIFO Data			SPI17Y_FIFO16	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15:0	data	R/W	0	**SPI0 16-bit FIFO Data Register** This register is used for the SPI0 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 15-9: SPI0 8-bit FIFO Register

SPI0 8-bit FIFO Data			SPI17Y_FIFO8	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**Reserved**
7:0	data	R/W	0	**SPI0 8-bit FIFO Data Register** This register is used for the SPI0 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 15-10: SPI0 Control 0 Register

SPI0 Control 0			SPI17Y_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:17	-	R/W	0	**Reserved**
16	ss_active	R/W	0	**Master Slave Select** Set this field to 1 to enable the slave select pin for SPI master. When this field is set to 1, the SPI0_SS0 pin is used for the slave select output when the next SPI transaction is started (*SPI17Y_CTRL0.start* = 1). 0: SPI0_SS0 is disabled 1: SPI0_SS0 is enabled for output in master mode operation. *Note: This field is only used when the SPI0 is configured for master mode (SPI17Y_CTRL0.master = 1).*
15:9	-	R/W	0	**Reserved**

SPI0 Control 0			SPI17Y_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
8	ss_ctrl	R/W	0	**Master Slave Select Control** This field controls the behavior of the slave select pins at the completion of a transaction. The default behavior, *ss_ctrl* = 0, deasserts the slave select pin at the completion of the transaction. Set this field to 1 to leave the slave select pins asserted at the completion of the transaction. If the external device supports this behavior, leaving the slave select pins asserted allows multiple transactions without the delay associated with deassertion of the slave select pin between transactions. 0: Slave select is deasserted at the end of a transmission 1: Slave select stays asserted at the end of a transmission
7:6	-	R/W	0	**Reserved**
5	start	R/W1O	0	**Master Start Data Transmission** Set this field to 1 to start a SPI0 master mode transaction. 0: No master mode transaction active. 1: Master initiates a data transmission. Ensure that all pending transactions are complete before setting this field to 1. *Note: This field is only used when the SPI0 is configured for master mode (SPI17Y_CTRL0.master = 1).*
4	ss_io	R/W	0	**Master Slave Select Signal Direction** Set the I/O direction for 0: Slave select is an output 1: Slave select is an input *Note: This field is only used when the SPI0 is configured for master mode (SPI17Y_CTRL0.master = 1).*
3:2	-	R/W	0	**Reserved**
1	master	R/W	0	**SPI0 Master Mode Enable** This field selects between slave mode and master mode operation for the SPI0 port. Write this field to 0 to operate as an SPI0 slave. Setting this field to 1 sets the port as an SPI0 master. 0: Slave mode SPI0 operation. 1: Master mode SPI0 operation.
0	en	R/W	0	**SPI0 Enable/Disable** This field enables and disables the SPI0 port. Disable the SPI0 port by setting this field to 0. Disabling the SPI0 port does not affect the SPI0 FIFOs or register settings. Access to SPI0 registers is always available. 0: SPI0 port is disabled 1: SPI0 port is enabled

Table 15-11: SPI0 Control 1 Register

SPI0 Transmit Packet Size			SPI17Y_CTRL1	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	rx_num_char	R/W	0	**Number of Receive Characters** Number of characters to receive in receive FIFO. *Note: If the SPI0 port is set to operate in 4-wire mode, this field is ignored and the SPI17Y_CTRL1.tx_num_chars field is used for both the number of characters to receive and transmit.*

SPI0 Transmit Packet Size			SPI17Y_CTRL1	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	tx_num_char	R/W	0	**Number of Transmit Characters** Number of characters to transmit from transmit FIFO. *Note: If the SPI0 port is set to operate in 4-wire mode, this field is used for both the number of characters to receive and transmit.*

Table 15-12: SPI0 Control 2 Register

SPI0 Control 2			SPI17Y_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:25	-	RO	0	**Reserved**
24	srpol	RO	0	**Reserved**
23:17	-	RO	0	**Reserved**
16	ss_pol	R/W	0	**Slave Select Polarity** This field controls the polarity of the SS signal. 0: Active low 1: Active high
15	three_wire	R/W	0	**Three-Wire SPI0 Enable** Set this field to 1 to enable three-wire SPI0 communication. Set this field to 0 for four-wire full-duplex SPI0 communication. 0: Four-wire full-duplex mode enabled. 1: Three-wire mode enabled
14	-	R/W	0	**Reserved**
13:12	data_width	R/W	0b00	**SPI0 Data Width** This field controls the number of data lines used for SPI0 communications. *Three-wire SPI*: *data_width* = 0 Set this field to 0, indicating SPI0_MOSI is used for half-duplex communication. *Four-wire full-duplex SPI*: *data_width* = 0 Set this field to 0, indicating SPI0_MOSI and SPI0_MISO are used for the SPI0 data output and input respectively. *Dual mode SPI*: *data_width* = 1 Set this field to 1, indicating SPI0_SDIO0 and SPI0_SDIO1 are used for half-duplex communication. 0: 1-bit per SCK cycle (Three-wire half-duplex SPI0 and four-wire full-duplex SPI) 1: 2-bits per SCK cycle (Dual mode SPI) 2: Reserved 3: Reserved *Note: When this field is set to 0, use the field SPI17Y_CTRL2.three_wire to select either three-wire SPI or four-wire SPI operation.*

SPI0 Control 2			SPI17Y_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
11:8	numbits	R/W	0	**Number of Bits per Character** Set this field to the number of bits per character for the SPI0 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 master mode. SPI17Y_CLK_CFG.scale must be > 0* *Note: For Dual and Quad mode SPI, the character size should be divisible by the number of bits per SCK cycle.*
7:2	-	R/W	0	**Reserved**
1	cpol	R/W	0	**Clock Polarity** This field controls the SCK polarity. The default clock polarity is for SPI0 mode 0 and mode 1 operation and is idle high. Invert the SCK polarity for SPI0 mode 2 and mode 3 operation. 0: Standard SCK for use in SPI0 mode 0 and mode 1 1: Inverted SCK for use in SPI0 mode 2 and mode 3
0	cpha	R/W	0	**Clock Phase** 0: Data sampled on clock rising edge. Use when in SPI0 mode 0 and mode 2 1: Data sampled on clock falling edge. Use when in SPI0 mode 1 and mode 3

Table 15-13: SPI0 Slave Select Timing Register

SPI0 Slave Select Timing			SPI17Y_SS_TIME	[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 (slave select inactive) and the start of the next transaction (slave select active). 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255 *Note: The SPI17Y_SS_TIME register bit settings only apply when SPI0 is operating in master mode (SPI17Y_CTRL0.master = 1)*

SPI0 Slave Select Timing			SPI17Y_SS_TIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
15:8	post	R/W	0	**Slave Select Hold Post Last SCK** Number of system clock cycles that SS remains active after the last SCK edge. 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255 *Note: The SPI17Y_SS_TIME register bit settings only apply when SPI0 is operating in master mode (SPI17Y_CTRL0.master = 1)*
7:0	pre	R/W	0	**Slave Select Delay to First SCK** Set the number of system clock cycles the slave select is held active prior to the first SCK edge. 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255 *Note: The SPI17Y_SS_TIME register bit settings only apply when SPI0 is operating in master mode (SPI17Y_CTRL0.master = 1)*

Table 15-14: SPI0 Master Clock Configuration Registers

SPI0 Master Clock Configuration			SPI17Y_CLK_CFG	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved**
19:16	scale	R/W	0	**SPI0 Peripheral Clock Scale** Scales the SPI0 input clock (PCLK) by 2^{scale} to generate the SPI0 peripheral clock. $f_{SPInCLK} = \frac{f_{SPIn\_INPUT\_CLK}}{2^{scale}}$ Valid values for scale are 0 to 8 inclusive. Values greater than 8 are reserved. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPI17Y_CLK_CFG.scale = 0, SPI17Y_CLK_CFG.hi = 0, and SPI17Y_CLK_CFG.lo = 0, character sizes of 2 and 10 bits are not supported.*
15:8	hi	R/W	0	**SCK Hi Clock Cycles Control** 0: Hi duty cycle control disabled. Only valid if SPI17Y_CLK_CFG.scale = 0. 1 - 15: The number of SPI0 peripheral clocks, f_{SPI0_CLK}, that SCK is high. All other values are reserved. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPI17Y_CLK_CFG.scale = 0, SPI17Y_CLK_CFG.hi = 0, and SPI17Y_CLK_CFG.lo = 0, character sizes of 2 and 10 bits are not supported.*

SPI0 Master Clock Configuration			SPI17Y_CLK_CFG	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
7:0	lo	R/W	0	**SCK Low Clock Cycles Control** This field controls the SCK low clock time and is used to control the overall SCK duty cycle in combination with the *SPI17Y_CLK_CFG.hi* field. 0: Low duty cycle control disabled. Setting this field to 0 is only valid if *SPI17Y_CLK_CFG.scale* = 0. 1 to 15: The number of SPI0 peripheral clocks, f_{SPI0_CLK}, that the SCK signal is low. All other values are reserved. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPI17Y_CLK_CFG.scale = 0, SPI17Y_CLK_CFG.hi = 0, and SPI17Y_CLK_CFG.lo = 0, character sizes of 2 and 10 bits are not supported.*

Table 15-15: SPI0 DMA Control Registers

SPI0 DMA Control			SPI17Y_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31	rx_dma_en	R/W	0	**Receive DMA Enable** 0: Disabled. Any pending DMA requests are cleared. 1: Enabled
30:24	rx_fifo_cnt	R	0	**Number of Bytes in the Receive FIFO** Read returns the number of bytes currently in the receive FIFO
23	rx_fifo_clear	W1O	-	**Clear the Receive FIFO** 1: Clear the receive FIFO and any pending receive FIFO flags in *SPI17Y_INT_FL*. This should be done when the receive FIFO is inactive. Writing a 0 has no effect.
22	rx_fifo_en	R/W	0	**Receive FIFO Enabled** 0: Disabled 1: Enabled
21	-	RO	0	**Reserved**
20:16	rx_fifo_level	R/W	0x00	**Receive FIFO Threshold Level** Set this value to the desired receive FIFO threshold level. When the receive FIFO level crosses above this setting, a DMA request is triggered if enabled by setting *SPI17Y_DMA.tx_dma_en*, and *SPI17Y_INT_FL.rx_fifo_level* becomes set. Valid values are 0 to 30. *Note: 31 is an invalid setting and reserved for future use.*
15	tx_dma_en	R/W	0	**Transmit DMA Enable** 0: Disabled. Any pending DMA requests are cleared. 1: Transmit DMA is enabled.
14:8	tx_fifo_cnt	RO	0	**Number of Bytes in the Transmit FIFO** Read this field to determine the number of bytes currently in the transmit FIFO.
7	tx_fifo_clear	R/W	0	**Transmit FIFO Clear** Set this bit to clear the transmit FIFO and all transmit FIFO flags in the *SPI17Y_INT_FL* register. *Note: The transmit FIFO should be disabled (SPI17Y_DMA.tx_fifo_en = 0) prior to setting this field.* *Note: Setting this field to 0 has no effect.*
6	tx_fifo_en	R/W	0	**Transmit FIFO Enabled** 0: Disabled 1: Enabled
5	-	RO	0	**Reserved**

SPI0 DMA Control		SPI17Y_DMA		[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
4:0	tx_fifo_level	R/W	0x10	**Transmit FIFO Threshold Level** Set this value to the desired transmit FIFO threshold level. When the transmit FIFO count (*SPI17Y_DMA.tx_fifo_cnt*) falls below this value, a DMA request is triggered if enabled by setting *SPI17Y_DMA.tx_dma_en* and *SPI17Y_INT_FL.tx_fifo_level* becomes set.

Table 15-16: SPI0 Interrupt Status Flags Registers

SPI0 Interrupt Status Flags		SPI17Y_INT_FL		[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15	rx_und	R/1	0	**Receive FIFO Underrun Flag** Set when a read is attempted from an empty receive FIFO.
14	rx_ovr	R/W1C	0	**Receive FIFO Overrun Flag** Set if SPI0 is in slave mode, and a write to a full receive FIFO is attempted. If the SPI0 is in master mode, this bit is not set as the SPI0 stalls the clock until data is read from the receive FIFO.
13	tx_und	R/W1C	0	**Transmit FIFO Underrun Flag** Set if SPI0 is in slave mode, and a read from empty transmit FIFO is attempted. If SPI0 is in master mode, this bit is not set as the SPI0 stalls the clock until data is written to the empty transmit FIFO.
12	tx_ovr	R/W1C	0	**Transmit FIFO Overrun Flag** Set when a write is attempted to a full transmit FIFO.
11	m_done	R/W1C	0	**Master Data Transmission Done Flag** Set if SPI0 is in master mode, and all transactions have completed. *SPI17Y_CTRL1.tx_num_char* has been reached.
10	-	R/W	0	**Reserved**
9	abort	R/W1C	0	**Slave Mode Transaction Abort Detected Flag** Set if the SPI0 is in slave mode, and SS is deasserted before a complete character is received.
8	fault	R/W1C	0	**Multi-Master Fault Flag** Set if the SPI0 is in master mode, multi-master mode is enabled, and a slave select input is asserted. A collision also sets this flag.
7:6	-	R/W	0	**Reserved**
5	ssd	R/W1C	0	**Slave Select Deasserted Flag**
4	ssa	R/W1C	0	**Slave Select Asserted Flag**
3	rx_full	R/W1C	0	**Receive FIFO Full Flag**
2	rx_thresh	R/W1C	0	**Receive FIFO Threshold Level Crossed Flag** Set when the receive FIFO exceeds the value in *SPI17Y_DMA.rx_fifo_level*. Cleared once receive FIFO level drops below *SPI17Y_DMA.rx_fifo_level.*
1	tx_empty	R/W1C	1	**Transmit FIFO Empty Flag** The transmit FIFO is empty.
0	tx_thresh	R/W1C	0	**Transmit FIFO Threshold Level Crossed Flag** Set when the transmit FIFO is less than the value in *SPI17Y_DMA.tx_fifo_level*. Cleared once transmit FIFO level exceeds *SPI17Y_DMA.tx_fifo_level.*

Table 15-17: SPI0 Interrupt Enable Registers

SPI0 Interrupt Enable			SPI17Y_INT_EN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15	rx_und	R/W	0	**Receive FIFO Underrun Interrupt Enable** 0: Disabled 1: Enabled
14	rx_ovr	R/W	0	**Receive FIFO Overrun Interrupt Enable** 0: Disabled 1: Enabled
13	tx_und	R/W	0	**Transmit FIFO Underrun Interrupt Enable** 0: Disabled 1: Enabled
12	tx_ovr	R/W	0	**Transmit FIFO Overrun Interrupt Enable** 0: Disabled 1: Enabled
11	m_done	R/W	0	**Master Data Transmission Done Interrupt Enable** 0: Disabled 1: Enabled
10	-	R/W	0	**Reserved**
9	abort	R/W	0	**Slave Mode Abort Detected Interrupt Enable** 0: Disabled 1: Enabled
8	fault	R/W	0	**Multi-Master Fault Interrupt Enable** 0: Disabled 1: Enabled
7:6	-	R/W	0	**Reserved**
5	ssd	R/W	0	**Slave Select Deasserted Interrupt Enable** 0: Disabled 1: Enabled
4	ssa	R/W	0	**Slave Select Asserted Interrupt Enable** 0: Disabled 1: Enabled
3	rx_full	R/W	0	**Receive FIFO Full Interrupt Enable** 0: Disabled 1: Enabled
2	rx_thresh	R/W	0	**Receive FIFO Threshold Level Crossed Interrupt Enable** 0: Disabled 1: Enabled
1	tx_empty	R/W	0	**Transmit FIFO Empty Interrupt Enable** 0: Disabled 1: Enabled
0	tx_thresh	R/W	0	**Transmit FIFO Threshold Level Crossed Interrupt Enable** 0: Disabled 1: Enabled

Table 15-18: SPI0 Wakeup Status Flags Registers

SPI0 Wakeup Flags			SPI17Y_WAKE_FL	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved**

SPI0 Wakeup Flags			SPI17Y_WAKE_FL	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
3	rx_full	R/W1C	0	**Wake on Receive FIFO Full Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
2	rx_thresh	R/W1C	0	**Wake on Receive FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
1	tx_empty	R/W1C	0	**Wake on Transmit FIFO Empty Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
0	tx_thresh	R/W1C	0	**Wake on Transmit FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.

Table 15-19: SPI0 Wakeup Enable Registers

SPI0 Wakeup Enable			SPI17Y_WAKE_EN	[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_thresh	R/W	0	**Wake on Receive FIFO Threshold Level Crossed Enable** 0: Wake event is disabled 1: Wake event is enabled.
1	tx_empty	R/W	0	**Wake on Transmit FIFO Empty Enable** 0: Wake event is disabled 1: Wake event is enabled.
0	tx_thresh	R/W	0	**Wake on Transmit FIFO Threshold Level Crossed Enable** 0: Wake event is disabled 1: Wake event is enabled.

Table 15-20: SPI0 Slave Select Timing Registers

SPI0 Status			SPI17Y_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	R/W	0	**Reserved**
0	busy	R	0	**SPI0 Active Status** SPI0 status flag, see value descriptions for details of each value. 0: SPI0 is not active. In master mode, the *busy* flag is cleared when the last character is sent. In slave mode, the *busy* field is cleared when the configured slave select input is deasserted. 1: SPI0 is active. In master mode, the *busy* flag is set when a transaction starts. In slave mode, the *busy* flag is set when a configured slave select input is asserted. *Note: SPI17Y_CTRL0, SPI17Y_CTRL1, SPI17Y_CTRL2,* SPI17Y_SS_TIME*, and* SPI17Y_CLK_CFG *should not be configured if this bit is set.*

16. SPIMSS (SPI1/I²S)

The SPIMSS peripheral provides independent serial communication support for I²S (Inter-IC Sound) for 16-bit mono or stereo audio transfer to or from an external I²S audio codec. The SPIMSS can also be configured as either a SPI master (in single or multi-master systems) or a SPI slave. An SPI system has a single master and one or more slaves for any given transaction.

16.1.1 Features

- SPI mode
  - 4-wire, full-duplex communication
  - 3-Wire, half-duplex communication supported
  - Master and slave mode operation
  - Slave select with independent polarity control
    - Single slave select for SPI1 (SPI1_SS0)
  - Programmable interface timing
  - Programmable SCK frequency and duty cycle
  - 32-byte transmit and receive FIFOs
- I²S mode
  - 16-bit audio transfer
  - Slave mode
  - Normal and left-justified data alignment
  - Wakeup on FIFO status (full/empty/threshold)
  - 32-byte transmit and receive FIFOs

The block diagram shows the SPIMSS external interface signals, control unit, receive and transmit FIFOs, and single shift register common to the transmit and receive data path. Each time that a SPIMSS transfer completes, the received character is transferred to the receive FIFO.

Figure 16-1: SPIMSS Block Diagram

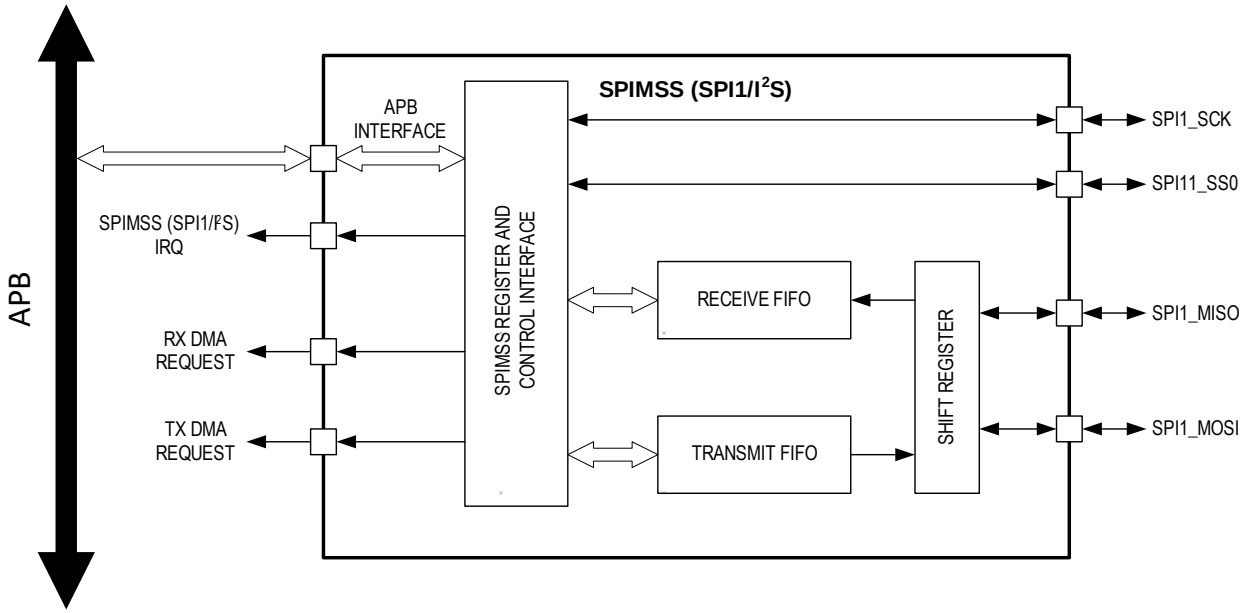
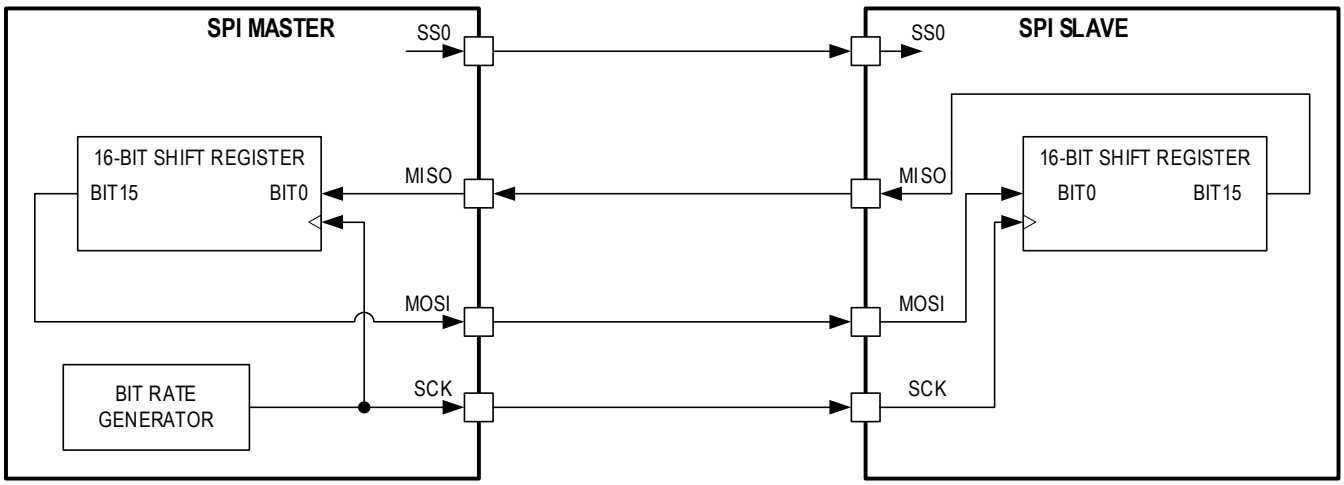


Figure 16-2: Typical SPI Network (Four-Wire)



16.2 Pin Configuration

Pin selection and configuration are required to use the SPIMSS port for either SPI1 or I²S. *Table 16-1* shows the pin selection options for the SPI1/I²S for each package available. The SPI alternate function name column maps the SPI signal name to the pin number on the MAX32660, and the I²S alternate function name column maps the I²S pin number on the MAX32660.

Note: The SPIMSS supports both SPI and I²S on the same pins and operates as either SPI or I²S based on the peripheral configuration.

Table 16-1: SPIMSS (SPI1/I²S) Pins

SPIMSS Port	SPI Signal	SPI Alternate Function Name	I2S Alternate Function Name	Alternate Function Number	GPIO	
					16-WLP	20-TQFN 24-TQFN
SPI1	SCK	SPI1_SCK	I2S_BCLK	AF2	P0.2	P0.2
	MOSI (SISO)	SPI1_MOSI	I2S_SDO	AF2	P0.1	P0.1
	MISO	SPI1_MISO	I2S_SDI	AF2	P0.0	P0.0
	SS	SPI1_SS0	I2S_LRCLK	AF2	P0.3	P0.3
	SCK	SPI1_SCK	I2S_BCLK	AF1	-	P0.12
	MOSI (SISO)	SPI1_MOSI	I2S_SDO	AF1	-	P0.11
	MISO	SPI1_MISO	I2S_SDI	AF1	-	P0.10
	SS	SPI1_SS0	I2S_LRCLK	AF1	-	P0.13

Note: SPI1/I²S is mapped to two different pin locations in the 20-TQFN and 24-TQFN packages. Only one of these locations should be used at a given time.

16.3 I²S System

In I²S mode, the slave select output is controlled by hardware and distinguishes left and right channel audio data. When operating as the I²S master, the SCK and SS signals are outputs. When operating as the I²S slave, the SCK and SS signals are inputs. This SS signal is referred to as word select signal (WS) in the I²S protocol. Normally the WS signal transitions one SCK period before the MSB of the audio data word; however, if the *SPIMSS_I2S_CTRL.i2s_lj* bit is set, the audio data word is "left justified" to be in phase with the WS signal.

16.4 SPIMSS Clock Phase and Polarity Control

The SPI supports four combinations of SCK phase and polarity. Clock polarity (*SPIMSS_CTRL.clkpol*) selects an active low/high clock and does not affect the transfer format. Clock phase (*SPIMSS_CTRL.phase*) selects one of two fundamentally different transfer formats.

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

Table 16-2: Clock Phase and Polarity Operation

SPIMSS_CTRL .phase	*SPIMSS_CTRL .clkpol*	SCK Transmit Edge	SCK Receive Edge	SCK Idle State
0	0	Falling	Rising	Low
0	1	Rising	Falling	High
1	0	Rising	Falling	Low
1	1	Falling	Rising	High

16.5 Data Movement

Data movement can be controlled in one of the following ways:

1. Software polling the *SPIMSS_STATUS.txst* bit (transfer one word at a time) or polling the *SPIMSS_DMA.tx_fifo_level* or *SPIMSS_DMA.rx_fifo_level* fields (can transfer up to 8 characters at a time).
2. The *SPIMSS_CTRL.irqe* bit can be set to enable data and error interrupts. The *SPIMSS_STATUS.str* bit can be used if desired to force a "startup" data interrupt. A data interrupt is generated on completion of each character transfer.
3. DMA control of data transferred is enabled through the *SPIMSS_DMA.rx_dma_en* and/or *SPIMSS_DMA.tx_dma_en* bits. The *SPIMSS_DMA.tx_fifo_level* and *SPIMSS_DMA.rx_fifo_level* control when DMA requests are asserted. When DMA is enabled, data interrupts are disabled (error interrupts still occur). DMA operation is beneficial for block transfers as the CPU only needs to service one DMA interrupt per block of data versus one interrupt for each character transferred.

The SPIMSS data register is used for transferring data in both incoming and outgoing directions.

For incoming data, the received data is shifted into an internal shift register. Once a full character has been shifted in, the character is automatically moved into the receive FIFO. The receive FIFO data is read through the SPIMSS data register.

The transmit data is written to the SPIMSS data register for outgoing data. When the outgoing data transfer is complete, the next entry in the FIFO is automatically transmitted.

Note: When the SPIMSS is not actively transmitting or receiving data (SPIMSS_CTRL.spien = 0), data written to the SPIMSS Data Register is stored in the FIFO as long as it is not full. Any data in the FIFO when the SPIMSS enable is set to 1 is transmitted immediately. Flush the FIFO at any time by setting the SPIMSS_DMA.tx_fifo_clear bit to 1.

With the SPI configured as a master, writing data to the *SPIMSS_DATA* register initiates the data transmission. With the SPIMSS configured as a SPI slave, writing data to the *SPIMSS_DATA* register loads the shift register in preparation for the following data transfer with the external master. In either the master or slave mode, when the transmit FIFO is full, writes to this register are ignored, and the transmit overrun error flag, *SPIMSS_STATUS.tovr*, is set in the SPIMSS Interrupt register.

Data is shifted out starting with bit 15. The last bit received resides in bit position 0. When the character length is less than 16 bits (as set by the *SPIMSS_MOD.numbits* field), the transmit character must be left-justified in the SPIMSS data register. A received character of fewer than 16 bits is right-justified (the last bit received is in bit position 0). For example, if the SPIMSS is configured for 4-bit characters, the transmit characters must be written to *SPIMSS_DATA[15:12],* and the received characters are read from *SPIMSS_DATA[3:0]*.

The software overhead to left justify the transmit data can be eliminated by setting the *SPIMSS_MOD.tx_lj* bit to 1. When *SPIMSS_MOD.tx_lj* = 1, transmit data is always written by software or DMA to *SPIMSS_DATA* in right-justified form, and hardware performs the left justify according to *SPIMSS_MOD.numbits* when the shift register is loaded. For the 4-bit character example, when *SPIMSS_MOD.tx_lj* = 1, transmit data is written to *SPIMSS_DATA[3:0],* and hardware shifts these to bits to *SPIMSS_DATA[15:12]* when the shift register is loaded. The *SPIMSS_MOD.tx_lj* bit has no effect on receive data which is always right-justified.

16.6 I²S (Inter-IC Sound) Mode

The SPIMSS block is configured for I²S mode operation by setting:

- *SPIMSS_I2S_CTRL.i2s_en* = 1
- *SPIMSS_CTRL.phase* = 0
- *SPIMSS_CTRL.clkpol* = 0
- *SPIMSS_MOD.numbits* = 0 (to select 16-bit characters)
- *SPIMSS_CTRL.spien* = 1

The *SPIMSS_CTRL.mmen* and *SPIMSS_MOD.ssio* bits are set in accordance with either master or slave mode of operation. The *SPIMSS_MOD.ssv* bit is ignored by hardware in I²S mode. In I²S, the master hardware sources the SS (known as WS in I²S

protocol) and SCK. In this mode, SS toggles between consecutive audio words. The slave select line low indicates left channel data, and the slave select line high indicates right channel audio data.

The receive and/or transmit DMA channels must be enabled when operating in I²S mode. Typically, audio data only flows in one direction as defined by the *SPIMSS_DMA.rx_dma_en* or *SPIMSS_DMA.tx_dma_en* bits; however, audio data can be transferred in both directions simultaneously if desired. The transmit buffer's data should be initialized with the first 16-bit character containing a left channel audio sample, then alternating right and left channel 16-bit audio samples. When audio data is being received, the first sample written into the receive buffer is the left channel audio sample.

16.6.1 Mute

The *SPIMSS_I2S_CTRL.i2s_mute* bit can be set by software asynchronously with respect to a DMA transfer to mute the transmit output. At the beginning of the next left channel audio sample, after *SPIMSS_I2S_CTRL.i2s_mute* is set to 1, the DMA and FIFO accesses continue; however, the data read from the transmit FIFO is discarded and replaced with zeroes. When the *SPIMSS_I2S_CTRL.i2s_mute* is set to 0, the transmit output resumes at the beginning of the next left channel audio sample.

16.6.2 Pause

The *SPIMSS_I2S_CTRL.i2s_pause* bit can be set by software asynchronously with respect to DMA transfers to halt the DMA and any FIFO accesses. At the beginning of the next left channel audio sample, after *SPIMSS_I2S_CTRL.i2s_pause* is set, both transmit and receive DMA, and FIFO accesses are halted, and the transmit data is forced to zero. At the beginning of the next left channel audio sample, after *SPIMSS_I2S_CTRL.i2s_pause* is set to 0, the DMA access resumes from the point it was previously halted. Pause takes precedence over mute.

16.6.3 Mono

The *SPIMSS_I2S_CTRL.i2s_mono* bit selects single-channel audio data or stereo format. Set *SPIMSS_I2S_CTRL.i2s_mono* to 1 to enable single-channel mono audio. In mono mode, each transmit data word read from the transmit FIFO is duplicated for both left and right channel output words. The receive channel reads the data from the left channel (SPI1_SS0 low) and ignores data in the right channel. This allows DMA buffers for mono mode to be one-half the size of DMA buffers for stereo mode.

16.6.4 Right and Left Justify

The *SPIMSS_I2S_CTRL.i2s_lj* bit selects the phase of the slave select signal versus the data. When *SPIMSS_I2S_CTRL.i2s_lj* = 0 (normal I²S mode), the audio data lags the slave select signal by one SCK period. When *SPIMSS_I2S_CTRL.i2s_lj* = 1, the audio data is "left justified" so that it is in sync with the slave select signal.

Figure 16-3: I²S Mode Right Justify Mode

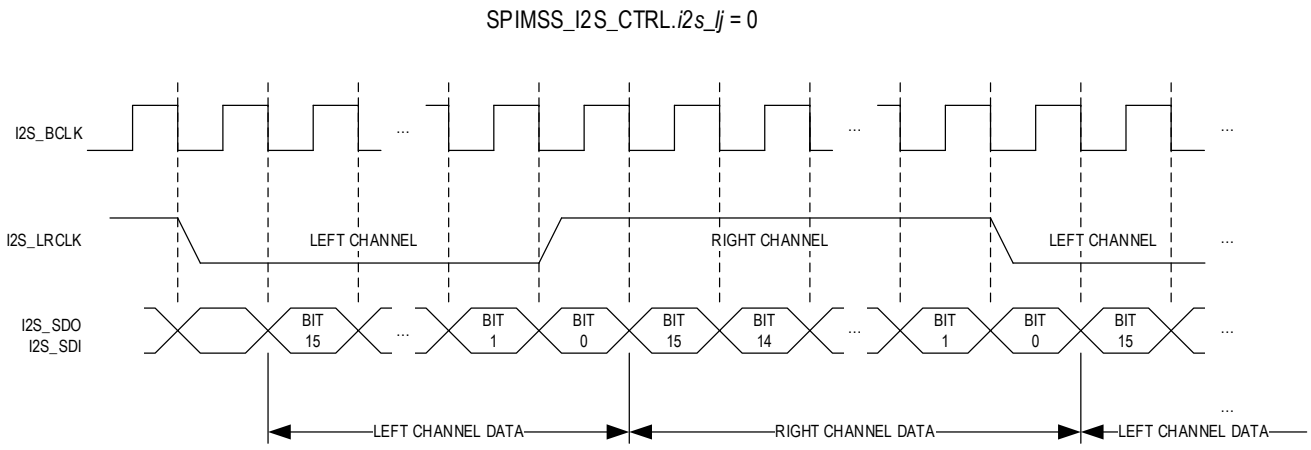
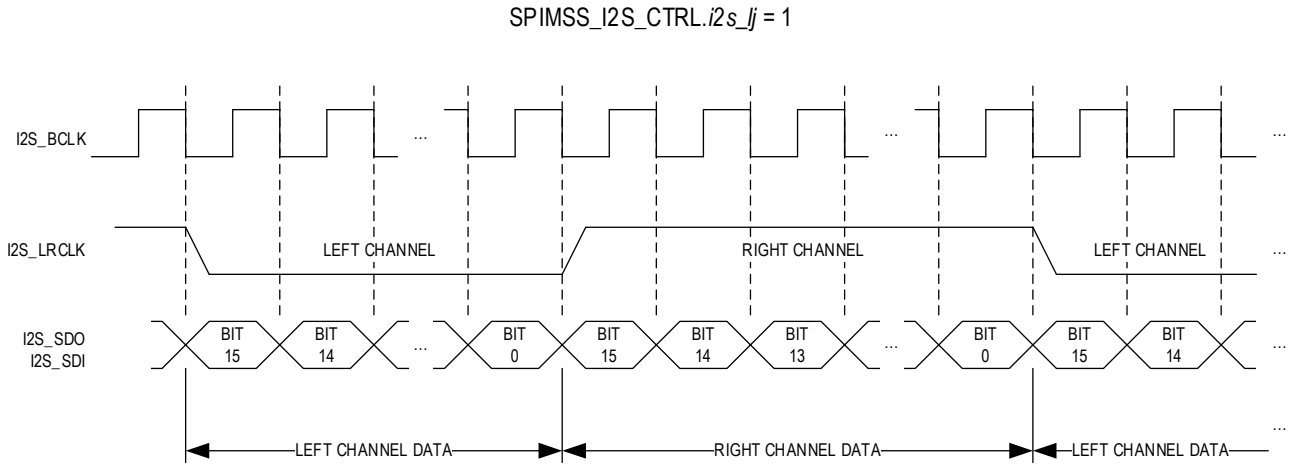


Figure 16-4: I²S Mode Left Justify Mode



16.7 Error Detection

The SPIMSS contains error detection logic to support the I²S communication protocol and recognize when communication errors have occurred. If the IRQE bit is set, error conditions generate an interrupt. The SPIMSS interrupt flag register indicates which error has been detected.

16.7.1 Transmit Overrun

A transmit overrun error indicates a write to the FIFO was attempted when the internal transmit FIFO was full in either master or slave mode. An overrun condition sets the *SPIMSS_STATUS.tovr* bit to 1. Writing a 1 to *SPIMSS_STATUS.tovr* clears this error flag.

A transmit FIFO overrun in I²S mode can result in mixing left and right channel data. Software should reinitialize the DMA channel and data buffer and restart the I²S transfer.

16.7.2 Transmit Underrun

A transmit underrun error indicates a read from the FIFO was attempted and the TX FIFO is empty in either master or slave mode. An underrun condition set the *SPIMSS_STATUS.tund* bit to 1. Write a 1 to *SPIMSS_STATUS.tund* to clear this error flag.

16.7.3 Mode Fault (Multi-Master Collision)

A mode fault indicates more than one master is trying to communicate simultaneously (a multi-master collision). The mode fault is detected when an enabled master's slave select input pin is asserted low. A mode fault sets the *SPIMSS_STATUS.col* bit to 1. Writing a 1 to *SPIMSS_STATUS.col* clears this error flag.

This error interrupt does not occur in I²S mode.

16.7.4 Slave Mode Abort

A slave mode abort indicates that the slave select pin deasserted before all bits in a character were transferred (while operating in slave mode). The next time the slave select asserts, the MISO pin outputs *SPIMSS_DATA[15]*, regardless of where the previous transaction left off. A slave mode abort sets the *SPIMSS_STATUS.abt* bit to 1. Writing a 1 to *SPIMSS_STATUS.abt* clears this error flag.

This error interrupt does not occur in I²S mode.

16.7.5 Receive Overrun

A receive overrun error indicates a write to the receive FIFO occurred when the internal receive FIFO was full (in either master or slave mode). An overrun sets *SPIMSS_STATUS.rovr* = 1. Writing a 1 to *SPIMSS_STATUS.rovr* clears this error flag.

A receive FIFO overrun in I²S mode can result in mixing left and right channel data. Software should reinitialize the DMA channel and data buffer and restart the I²S transfer.

16.8 SPIMSS Interrupts

When the SPI interrupt is enabled, *SPIMSS_CTRL.irqe* = 1, the SPIMSS generates an interrupt an enabled interrupt condition occurs. The interrupt condition is indicated by the *SPIMSS_STATUS.irq* bit. Writing a 1 to the *SPIMSS_STATUS.irq* bit clears the pending SPIMSS interrupt request.

16.8.1 Data Interrupt

A data interrupt occurs when the transmit character has been fully moved out of the shift register, and the transmit FIFO is empty (in either master or slave I²S mode). Since transmit and receive are always interlocked, there is no need for a separate receive interrupt. If either transmit or receive DMA is enabled through the *SPIMSS_DMA.rx_dma_en* and *SPIMSS_DMA.tx_dma_en* bits, the data interrupt does not occur; however, error interrupts are still enabled when using DMA. A data interrupt is indicated by *SPIMSS_STATUS.irq* = 1, and no error interrupt flags are set.

16.8.2 Forced Interrupt

Force an SPIMSS interrupt to start a transaction by writing 1 to the *SPIMSS_CTRL.str* bit.

16.8.3 Error Condition Interrupt

If any of the SPIMSS error conditions occurs as described in the *Error Detection* section, the corresponding error bit in the *SPIMSS_STATUS* register and the *SPIMSS_STATUS.irq* bit are set, and a SPIMSS interrupt (IRQ) is generated. The error status bits and the *SPIMSS_STATUS.irq* bit should be cleared at the same time by writing a 1 to the corresponding bits.

16.8.4 Bit Rate Generator Timeout Interrupt

When the SPIMSS is disabled, a SPIMSS interrupt can be generated using the SPIMSS bit rate generator timeout. Enable the SPIMSS bit rate generator by setting *SPIMSS_CTRL.birq* to 1 to use the SPIMSS BRG as a timer.

16.9 SPIMSS Bit Rate Generator

16.9.1 Slave Mode

The bit rate generator is not used in I²S slave mode. When the SPIMSS is configured as an I²S slave, the I2S_BCLK input frequency must be less than or equal to $f_{PCLK}/_{8}$.

16.9.2 Master Mode

In SPIMSS (SPI1/I²S) master mode, the bit rate generator creates a clock for data transmission synchronization between the master and the external slave. The input to the bit rate generator is the peripheral clock, PCLK.

For I²S operation, the maximum supported bit rate is $f_{PCLK}/_{4}$.

For SPI operation, the maximum supported bit rate is $f_{PCLK}/_{2}$.

The SPIMSS (SPI1/I²S) bit rate is calculated using *Equation 16-1* (for the specific case where *SPIMSS_BRG.brg* = 0, substitute 2^{16} for *SPIMSS_BRG.brg* in the equation).

Equation 16-1: SPIMSS Bit Rate Equation

$$SPIMSS\ Bit\ Rate\ \left({}^{bits}/_{sec}\right) = \left(\frac{f_{PCLK}}{2 \times SPIMSS\_BRG.brg}\right)$$

$$For\ SPIMSS\_BRG.div = 0, substitute\ 2^{16}$$

16.9.3 Timer Mode

When the SPIMSS is disabled, the bit rate generator can function as a continuous mode 16-bit timer with interrupt on timeout. Configure the bit rate generator as a timer with interrupt on timeout using the following sequence:

1. Set *SPIMSS_CTRL*.spien = 0 to disable the SPIMSS SPI or I²S activity.
2. Load the desired 16-bit count value into the SPIMSS Bit Rate Register field, *SPIMSS_BRG*.*brg*.
3. Set *SPIMSS_CTRL*.*birq* = 1 to enable the bit rate generator.
4. When the bit rate generator timer expires, the *SPIMSS_STATUS*.*irq* flag is set by hardware.

16.10 SPIMSS Registers

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

Table 16-3: SPIMSS Register Offsets, Access and Descriptions

Offset	Register Name	Description
[0x0000]	*SPIMSS_DATA*	SPIMSS Data Register
[0x0004]	*SPIMSS_CTRL*	SPIMSS Control Register
[0x0008]	*SPIMSS_STATUS*	SPIMSS Interrupt Flag Register
[0x000C]	*SPIMSS_MOD*	SPIMSS Mode Register
[0x0014]	*SPIMSS_BRG*	SPIMSS Bit Rate Register
[0x0018]	*SPIMSS_DMA*	SPIMSS DMA Register
[0x001C]	*SPIMSS_I2S_CTRL*	SPIMSS I²S Control Register

16.10.1 Register Details

Table 16-4: SPIMSS Data Register

SPIMSS Data Register			SPIMSS_DATA	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	data	R/W	0	**SPIMSS Data** See the *Data Movement* section for details.

Table 16-5: SPIMSS Control Register

SPIMSS Control Register			SPIMSS_CTRL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**

SPIMSS Control Register			SPIMSS_CTRL	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
7	irqe	R/W	0	**Interrupt Request Enable** Set to enable interrupts for the SPIMSS peripheral. 0: SPI interrupts are disabled. 1: SPI interrupts are enabled. Interrupt requests are sent to the Interrupt Controller *Note: If transmit or receive DMA is enabled, the transmit data complete interrupt is disabled, but other interrupt sources are enabled.*
6	str	R/W	0	**Start SPI Interrupt** Setting this bit starts a SPIMSS interrupt request. Setting this bit also sets *SPIMSS_STATUS.irq* to 1. Setting this bit forces the SPIMSS to send an interrupt request to the Interrupt Controller if *SPIMSS_CTRL.irqe* = 1. Write 0 to clear this bit or by clearing all pending interrupts in the *SPIMSS_STATUS* register.
5	birq	R/W	0	**Bit Rate Generator Timer Interrupt Request** Enable the bit rate generator, and the bit rate generator Interrupt if the SPIMSS is stopped, *SPIMSS_CTRL.enable* = 0. 0: Bit rate generator disabled. 1: Enable the bit rate generator and the associated bit rate generator Interrupt. *Note: If SPIMSS_CTRL.enable = 1, this bit has no effect.*
4	phase	R/W	0	**Phase Select** See the *SPIMSS Clock Phase and Polarity Control* section for details.
3	clkpol	R/W	0	**Clock Polarity** This field sets the idle state for the SCK clock pin after a character transaction. 0: SCK idles Low (0) after character transmission/reception. 1: SCK idles High (1) after character transmission/reception.
2	wor	R/W	0	**Wired OR (Open Drain) Enable** Set to enable wired OR for the SPI signal pins (SPI1_SCK, SPI1_SS0, SPI1_MOSI, SPI1_MISO). 0: Wired OR configuration disabled. 1: Wired OR configuration enabled.
1	mmen	R/W	0	**SPI Master Mode Enable** Set this field to enable Master Mode for SPI. 0: SPI set to slave mode operation 1: SPI set to master mode operation
0	spien	R/W	0	**SPI Start** Set this field to start the operation of the SPIMSS port as configured. If the FIFOs contain data, the data is considered valid by the SPIMSS peripheral and is used. 0: Stop SPIMSS operation. 1: Start SPIMSS transaction as configured. *Note: This bit should be set to 1 only after the SPIMSS is configured for operation. Setting this bit to 0 does not reset or change any configuration of the SPIMSS peripheral and does not affect any data in the FIFOs.*

Table 16-6: SPIMSS Interrupt Flag Register

SPIMSS Interrupt Flag Register			SPIMSS_STATUS	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**

SPIMSS Interrupt Flag Register			SPIMSS_STATUS	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
7	irq	R/W1C	0	**SPI Interrupt Request Flag** This bit is set by hardware when an SPI interrupt request is pending. Write 1 to clear. 0: No SPI interrupt request is pending. 1: An SPI interrupt request is pending. *Note: This field cannot be cleared unless all interrupt flags in this register are cleared.*
6	tovr	R/W1C	0	**Transmit Overrun Flag** This bit is set by hardware when a transmit FIFO overrun has occurred. Write 1 to clear. 0: No SPI interrupt request is pending 1: An SPI interrupt request is pending
5	col	R/W1C	0	**Collision Flag** This bit is set by hardware when a multi-master collision (mode fault) occurs. Write 1 to clear. 0: No multi-master collision has occurred 1: A multi-master collision has occurred
4	abt	R/W1C	0	**Slave Mode Transaction Abort Flag** This bit is set by hardware when a slave mode transaction abort occurs. Write 1 to clear. 0: No slave mode transaction abort has occurred 1: A slave mode transaction abort has occurred
3	rovr	R/W1C	0	**Receive Overrun Flag** This bit is set by hardware when a receive FIFO overrun occurs. Write 1 to clear. 0: No FIFO overrun has occurred 1: A FIFO overrun has occurred.
2	tund	R/W1C	0	**Transmit Underrun Flag** This bit is set by hardware to indicate a transmit FIFO underrun has occurred. Write 1 to clear. 0: No FIFO underrun has occurred 1: A FIFO underrun has occurred
1	txst	RO	0	**Transmit Status** This field reads 1 if a SPIMSS data transmission is currently in progress. 0: No data transmission currently in progress. 1: Data transmission currently in progress
0	slas	RO	0	**Slave Select** If the SPI is in slave mode, this bit indicates if the SPI is selected. If the SPI is in master mode, this bit has no meaning. 0: Slave SPI is selected 1: Slave SPI is not selected

Table 16-7: SPIMSS Mode Register

SPIMSS Mode Register			SPIMSS_MOD	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**

SPIMSS Mode Register			SPIMSS_MOD	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
7	tx_lj	R/W	0	**Transmit Data Alignment** Selects left or right alignment when data is loaded into the *SPIMSS_DATA.data* field for transmission if the character size is less than 16-bits. 0: Data is LSB aligned with the unused bits set to 0 up to the MSB (right aligned) 1: Data is MSB aligned with the unused bits set to 0 down to the LSB (left aligned)
6	-	RO	0	**Reserved**
5:2	numbits	R/W	0	**Number of Data Bits per Character** This field contains the number of bits to shift for each character transfer. See the *Data Movement* section for information on valid bit positions when the character length is less than 16-bits. 0b0000: 16-bits 0b0001: 1-bits 0b0010: 2-bits … 0b1110: 14-bits 0b1111: 15-bits *Note: Setting this field to 0 (default) sets the number of bits per character to 16.*
1	ssio	RO	0	**Master Mode Slave Select Output** This field must be set to 1 if the SPIMSS is configured as a master (*SPIMSS_CTRL.mmen* = 1). ***CAUTION***: This field must be set to 1.
0	ssv	R/W	0	**Slave Select Value** This field indicates the value of the SPI1_SS0 (I2S_LRCLK) pin if the SPIMSS slave select pin is configured as an output, *SPIMSS_MOD.ssio* = 1, writing this field drives the pin to the value written.

Table 16-8: SPIMSS Bit Rate Generator Register

SPIMSS Bit Rate Generator Register			SPIMSS_BRG	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	brg	R/W	0	**Bit Rate Reload Value** The SPI bit rate register is a 16-bit reload value for the SPI bit rate generator. The reload value must be greater than or equal to 2 for proper SPI1 or I²S operation (maximum bit rate is f_{PCLK} divided by 4). See the section *SPIMSS Bit Rate Generator* for calculation.

Table 16-9: SPIMSS DMA Register

SPIMSS DMA Register			SPIMSS_DMA	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
31	rx_dma_en	R/W	0	**Receive DMA Enable** Disabling clears any active request to the DMA controller. 0: Disable RX DMA requests 1: Enable RX DMA requests
30:28	-	RO	0	**Reserved**

SPIMSS DMA Register			SPIMSS_DMA	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**
27:24	rx_fifo_cnt	R/W	0	**Receive FIFO Count** 0: RX FIFO empty (0 entries) 1: RX FIFO contains 1 entry 2: RX FIFO contains 2 entries 3: RX FIFO contains 3 entries … 15: RX FIFO contains 15 entries
23:21	-	RO	0	**Reserved**
20	rx_fifo_clear	R/W1O	0	**Receive FIFO Clear** Write 1 to reset the receive FIFO. Writing 0 has no effect. 0: Normal operation 1: Reset receive FIFO or operation in progress.
19	-	RO	0	**Reserved**
18:16	rx_fifo_level	R/W	0	**Receive FIFO Level** This field sets the RX FIFO DMA request threshold, configuring the number of filled RX FIFO entries before activating an RX DMA request. 0: Request receive DMA when RX FIFO contains 1 entry 1: Request receive DMA when RX FIFO contains 2 entries 2: Request receive DMA when RX FIFO contains 3 entries … 7: Request receive DMA when RX FIFO contains 8 entries
15	tx_dma_en	R/W	0	**Transmit DMA Enable** Disabling clears any active request to the DMA controller. 0: Disable TX DMA requests 1: Enable TX DMA requests
14:12	-	RO	0	**Reserved**
11:8	tx_fifo_cnt	RO	0	**Transmit FIFO Count** 0: TX FIFO empty (0 entries) 1: TX FIFO contains 1 entry 2: TX FIFO contains 2 entries 3: TX FIFO contains 3 entries … 15: TX FIFO contains 15 entries
7:5	-	RO	0	**Reserved**
4	tx_fifo_clear	R/W1O	0	**Transmit FIFO Clear** Write 1 to reset the receive FIFO. Writing 0 has no effect. 0: Normal operation 1: Reset transmit FIFO or operation in progress.
3	-	RO	0	**Reserved**
2:0	tx_fifo_level	R/W	0	**Transmit FIFO Level** This field sets the TX FIFO DMA request threshold, configuring the number of empty TX FIFO entries before activating a transmit DMA request. 0: Request transmit DMA when TX FIFO has 1 free entry. 1: Request transmit DMA when TX FIFO has 2 free entries. 2: Request transmit DMA when TX FIFO has 3 free entries. … 7: Request transmit DMA when TX FIFO has 8 free entries.

Table 16-10: SPIMSS I²S Control Register

SPIMSS I²S Control Register			SPIMSS_I2S_CTRL	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**

SPIMSS I²S Control Register			SPIMSS_I2S_CTRL	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
4	i2s_lj	R/W	0	**I²S Left Justify** 0: Normal I²S audio protocol - audio data lags left/right channel signal by one SCK period. 1: Audio data is synchronized with the slave select transition (left/right channel signal).
3	i2s_mono	R/W	0	**I²S Monophonic Audio Mode** Set this field to enable monophonic audio mode. In this mode, each transmit data word is replicated on both left and right channels. Receive data is taken from the left channel, right channel receive data is ignored. 0: Stereophonic audio. 1: Monophonic audio format
2	i2s_pause	R/W	0	**I²S Pause Transmit/Receive** 0: Normal transmission/reception. 1: Halt transmit and receive FIFO and DMA accesses, transmit 0.
1	i2s_mute	R/W	0	**I²S Mute Transmit** 0: Normal transmit. 1: Transmit data is replaced with 0
0	i2s_en	R/W	0	**I²S Mode Enable** Set to enable I²S mode. 0: I²S mode is disabled. 1: I²S mode enabled.

17. Revision History

Table 17-1: Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	7/2018	Initial release	-
1	7/2021	Revision 1 updates	Major update for improvements, corrections, and updates for revision A2 silicon.

Trademarks

Arm and Cortex are registered trademarks of Arm Limited.

Maxim Integrated is a registered trademark of Maxim Integrated Products, Inc.