MAX32675 User Guide

UG7594; Rev 0; 01/22

Abstract: The MAX32675 user guide provides application developers information on how to use the memory and peripherals of the MAX32675 microcontroller. Detailed information for all registers and fields in the device are covered. Guidance is given for managing all the peripherals, clocks, power. and startup for the device family.

MAX32675 User Guide

Table of Contents

Table of Figures

Table of Tables

Table of Equations

1. Introduction

For ordering information, mechanical and electrical characteristics for the MAX32675 family of devices please refer to the data sheet. 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 MAX32675 data sheet and errata are available from the Analog Devices, Inc. website, *http://www.maximintegrated.com/MAX32675*.

1.2 Document Conventions

1.2.1 Number Notations

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

1.2.2 Register and Field Access Definitions

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

Table 1-1: Field Access Definitions

Access Type	Definition
RO	**Reserved** This access type is reserved for static fields. Reads of this field return the reset value. Writes are ignored.
DNM	**Reserved. Do Not Modify** Software must first read this field and write the same value whenever writing to this register.
R	**Read Only** Reads of this field return a value. Writes to the field do not affect device operation.
W	**Write Only** Reads of this field return indeterminate values. Writes to the field change the field's state to the value written and can affect device operation.
R/W	**Unrestricted Read/Write** Reads of this field return a value. Writes to the field change the field's state to the value written and can affect device operation.
RC	**Read to Clear** Reading this field clears the field to 0. Writes to the field do not affect device operation.
RS	**Read to Set** Reading this field sets the field to 1. Writes to the field do not affect device operation.
R/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 3-2* to get the register's absolute address.

Table 1-2: Example Registers

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

1.2.4 Register Detail Tables

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

Table 1-3: Example Name 0 Register

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

2. Overview

The MAX32675 is a highly integrated, mixed-signal, ultra low-power microcontroller for industrial applications and is especially suitable for 4-20mA loop powered sensors and transmitters. It is based on an ultra-low power ARM® Cortex®-M4 with Floating Point Unit (FPU) and includes 384KB of flash and 160KB of SRAM. Error correction coding (ECC), capable of single error correction and double error detection (SEC-DED), is implemented over the entire flash, SRAM, and cache to ensure ultra-reliable code execution for demanding applications. A version of the device with integrated, low-power, HART modem enables bidirectional transfer of digital data over a current loop, to/from industrial sensors for configuration and diagnostics.

Two 12-channel sigma-delta ADCs configurable as 24-bit at up to 0.4ksps or 16-bit at up to 4ksps can digitize external analog signals, as well as system temperature and supplies. An optional PGA with gains of 1× to 128× precedes each ADC. ADC outputs can be optionally converted on the fly from integer to single precision floating point format. A 12-bit DAC is also included. The integrated temperature sensor can be used with the internal sense element or an external diode for temperature compensation of sensor outputs. The device also includes a trust protection unit (TPU) providing robust security features such as an AES Engine, TRNG, and Secure Boot.

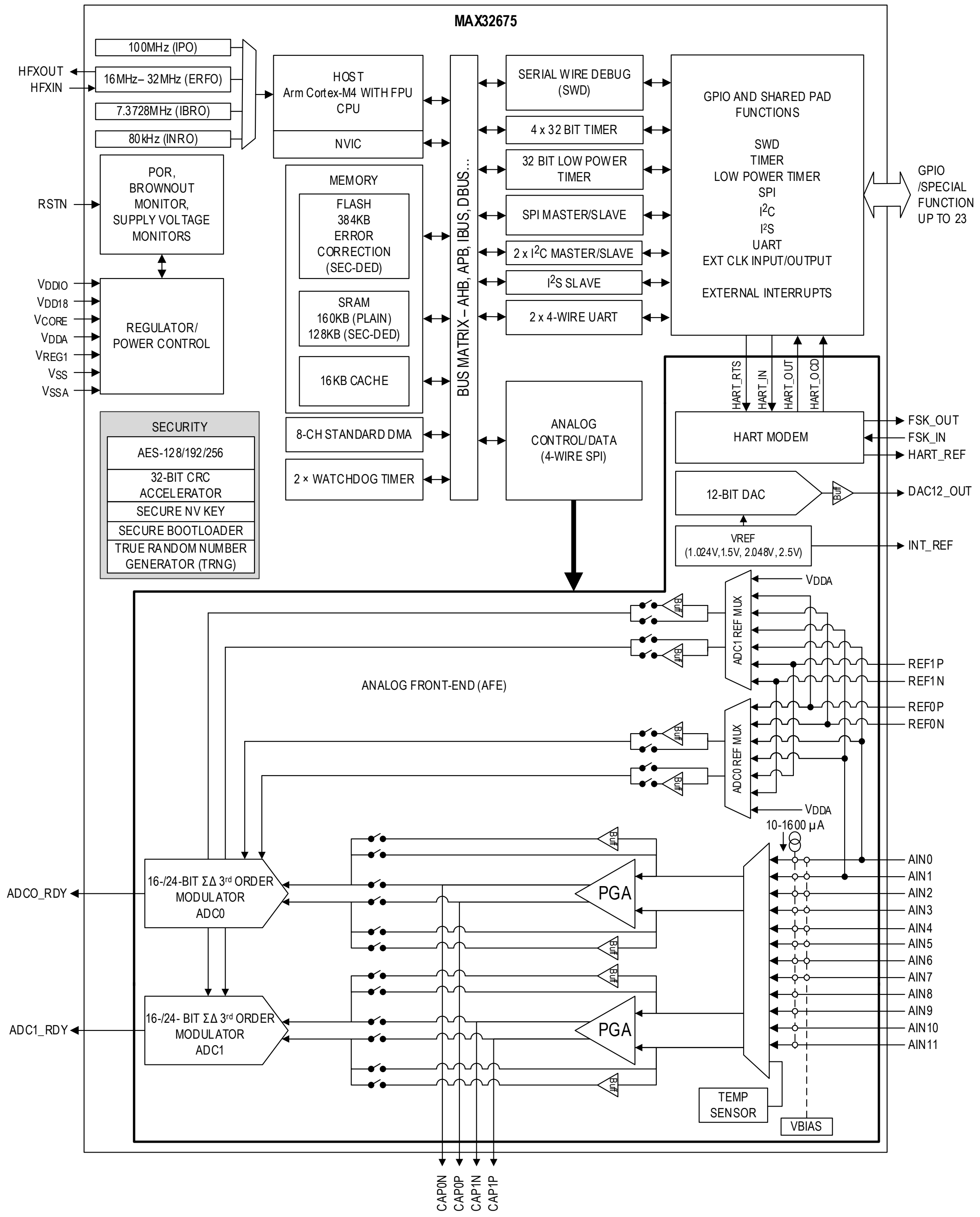
The high-level block diagram for the MAX32675 is shown in *Figure 2-1*.

Arm is a registered trademark and registered service mark of Arm Limited.

Cortex is a registered trademark of Arm Limited.

2.1 Block Diagram

Figure 2-1: MAX32675 Block Diagram



3. Memory, Register Mapping, and Access

3.1 Memory, Register Mapping, and Access Overview

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

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

Figure 3-1: Code Memory Mapping

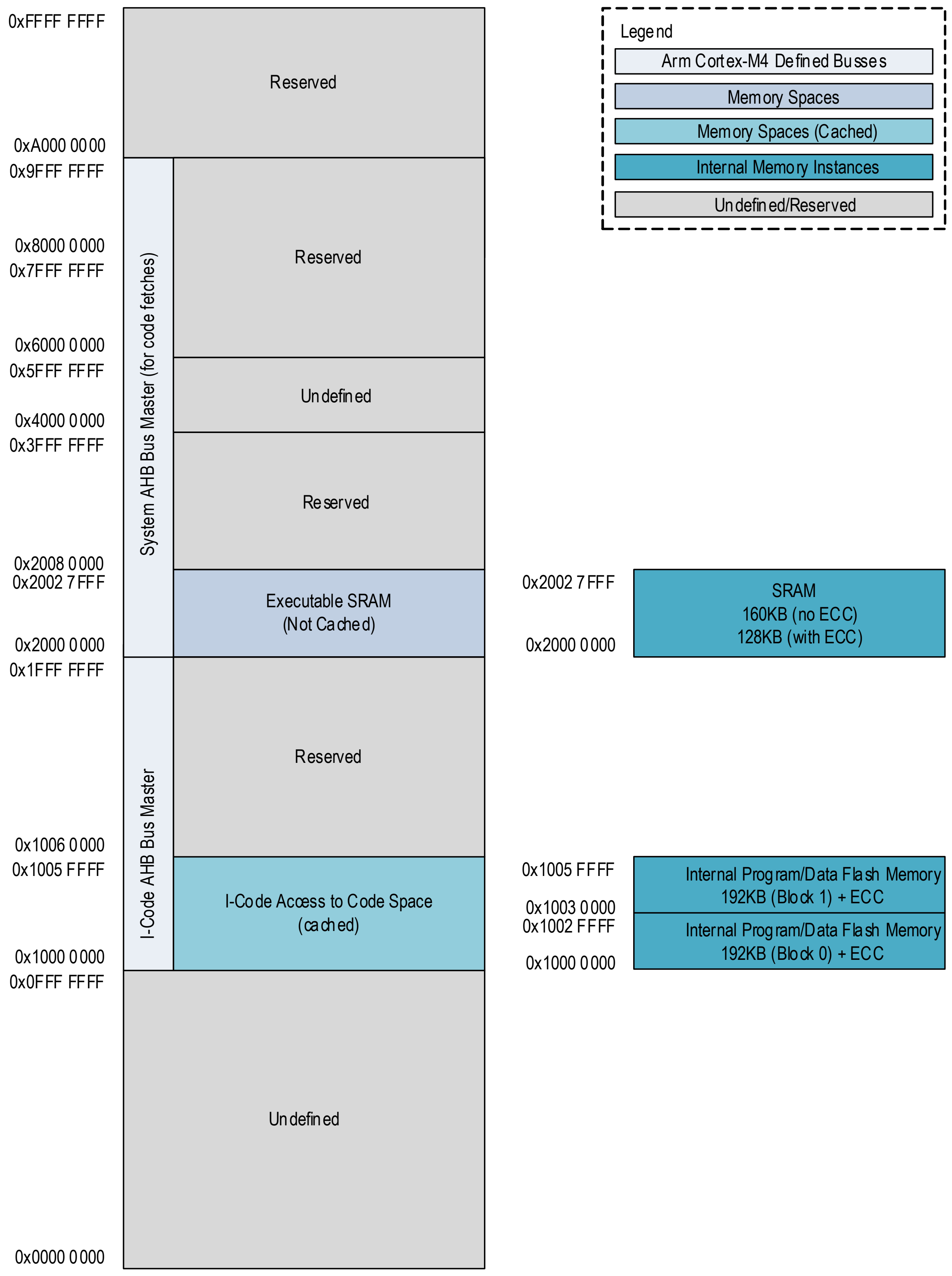
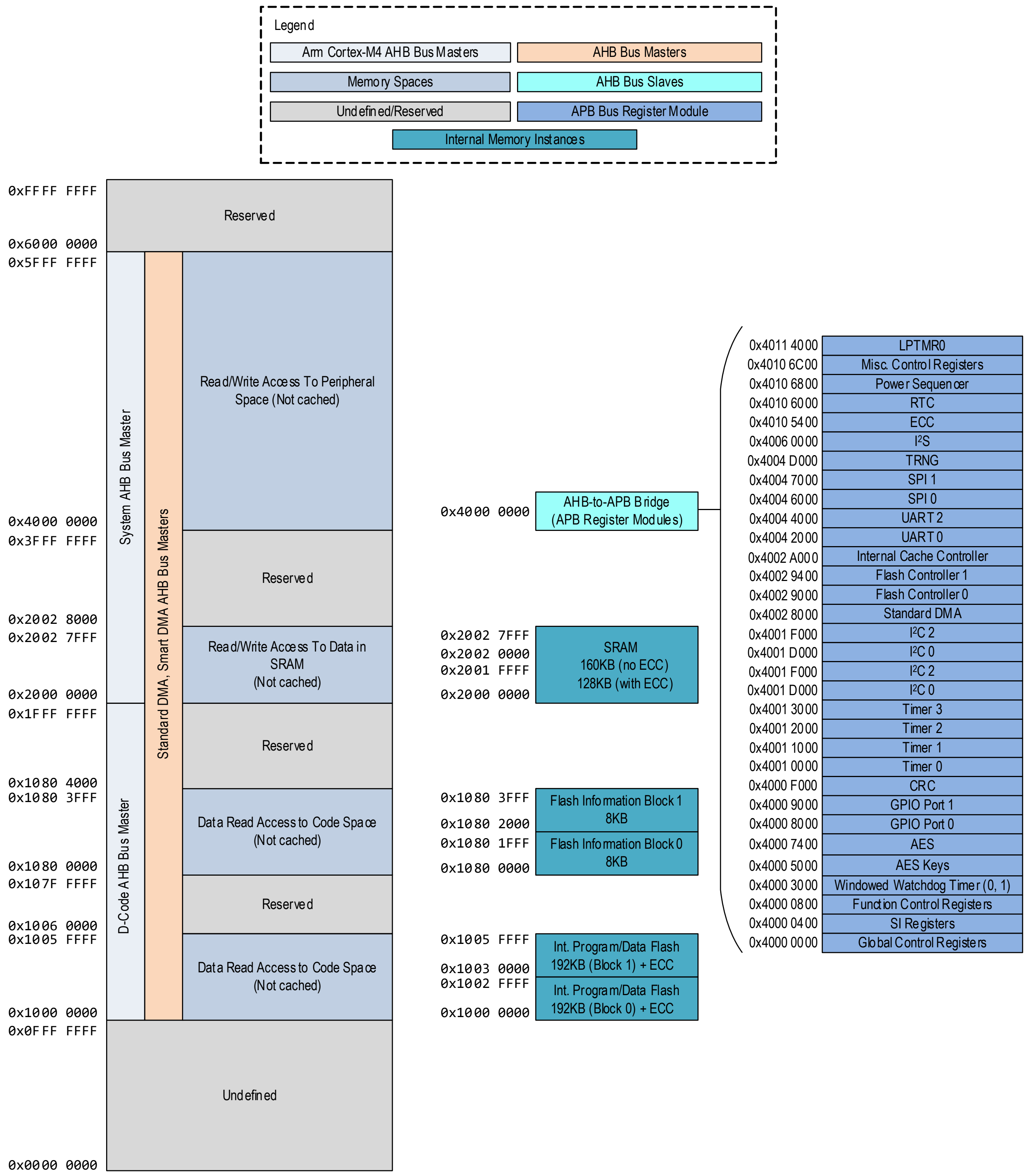


Figure 3-2: Data Memory Mapping



3.2 Standard Memory Regions

Several standard memory regions are defined for the Arm Cortex-M4 architecture. The use of many of these is optional for the system integrator. At a minimum, the MAX32675 must contain code and data memory for the software, variables, stack, and specific components that are part of the core.

3.2.1 *Code Space*

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

The MAX32675 code memory mapping is illustrated in *Figure 3-1*. The code space memory area contains the main internal flash memory, which holds the software executed on the device. The internal flash memory is mapped into both code and data space from 0x1000 0000 to 0x1005 FFFF. The main program flash memory is 384KB in size (two banks of 192KB) and consists of 48 logical pages of 8,192 bytes per page. In addition, the flash memory includes extra flash storage for Error Correction Coding (ECC) check bits if ECC is enabled. However, the ECC check bit memory is not user-accessible even when ECC is disabled.

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

The code space memory on the MAX32675 also contains the mapping for the flash information blocks. Flash information block 0 is mapped from address 0x1080 0000 to address 0x1080 1FFF, and flash information block 1 is mapped from address 0x1080 2000 to address 0x1080 3FFF.

3.2.2 *Internal Cache Memory*

The internal cache controller (ICC)is 16,384 bytes in size and is used to cache instructions fetched using the I-Code bus, including instructions fetched from the internal flash memory.

3.2.3 *Information Block Flash Memory*

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

Figure 3-3: Unique Serial Number Format

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

The information block must be unlocked to access the USN. Use the following steps to unlock this block:

1. Write 0x3A7F5CA3 to *FLCn_ACTRL*.
2. Write 0xA1E34F20 to *FLCn_ACTRL*.
3. Write 0x9608B2C1 to *FLCn_ACTRL*.
4. Information block is now read-only accessible and reads starting at address 0x1080 0000 return the USN as shown in *Figure 3-3*.

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

3.2.4 AES Key and Working Space Memory

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

3.2.5 SRAM Space

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

The MAX32675 data memory mapping is illustrated in *Figure 3-2*, and the SRAM configuration is defined in *Table 3-1*. This memory area contains the main system SRAM. The size of the internal SRAM is 160KB when not using ECC. Its address range is 0x2000 0000 to 0x2002 7FFF. If ECC is enabled, the SRAM size decreases to 128KB. The address range with ECC enabled is 0x2000 0000 to 0x2001 FFFF.

The entirety of the SRAM memory space on the MAX32675 is contained within the dedicated Arm Cortex-M4 SRAM bit-banding region from 0x2000 0000 to 0x200F FFFF (1MB maximum for bit-banding). As a result, the CPU can access the entire SRAM either using standard byte/word/doubleword access or using bit-banding operations. The bit-banding mechanism allows any single bit of any given SRAM byte address location to be set, cleared, or read individually by reading from or writing to a corresponding doubleword (32-bit wide) location in the bit-banding alias area.

The alias area for the SRAM bit-banding is located beginning at 0x2200 0000 and is a total of 32MB maximum, which allows the entire 160KB bit-banding area to be accessed. Each 32-bit (4 byte aligned) address location in the bit-banding alias area translates into a single-bit access (read or write) in the bit-banding primary area. Thus, reading from the location performs a single-bit read while writing either a 1 or 0 to the location performs a single-bit set or clear.

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

The SRAM area on the MAX32675 can be used to contain executable code. Code stored in the SRAM is accessed directly for execution (using the system bus) and is not cached. The SRAM is also where the Arm Cortex-M4 stack must be located, as it is the only general-purpose SRAM memory on the device. A valid stack location inside the SRAM must be set by the system exception table (which is, by default, stored at the beginning of the internal flash memory).

The MAX32675 specific AHB Bus Masters can access the SRAM to use as general storage or working space.

Table 3-1: SRAM Configuration

System RAM Block #	Size (Words)	Start Address	End Address	ECC SRAM Complement
sysram0	4K	0x2000 0000	0x2000 3FFF	*sysram4*
sysram1	4K	0x2000 4000	0x2000 7FFF	*sysram5*
sysram2	8K	0x2000 8000	0x2000 FFFF	*sysram6*
sysram3	16K	0x2001 0000	0x2001 FFFF	*sysram7*
sysram4	32K	0x2002 0000	0x2002 0FFF	–
sysram5	32K	0x2002 1000	0x2002 1FFF	–
sysram6	2K	0x2001 2000	0x2002 3FFF	–
sysram7	2K	0x2002 4000	0x2002 7FFF	–

3.2.6 Peripheral Space

The peripheral space area of memory is intended to map control registers, internal buffers/working space, and other features needed for the software control of non-core peripherals. It is defined from byte address range 0x4000 0000 to 0x5FFF FFFF (0.5GB maximum). On the MAX32675, all device-specific module registers are mapped to this memory area and any local memory buffers or FIFOs required by modules.

On the MAX32675, 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. Thus, mapping the peripherals to the APB bus allows lower-power operation of the peripheral modules. Furthermore, 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 application instruction and data fetching.

3.2.7 System Area (Private Peripheral Bus)

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

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

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

3.3 AHB Interfaces

The AHB memory accessibility and organization of the AHB master and slave interfaces are detailed in this section.

3.3.1 Core AHB Interfaces

3.3.1.1 I-Code

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

3.3.1.2 D-Code

The Arm core uses 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 bus master has access to the internal flash memory and the information blocks. Data fetched using the D-Code bus master are returned by the internal cache, triggering a cache line fill cycle to fetch data from the internal flash memory when a cache miss occurs.

3.3.1.3 System

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

3.3.2 AHB Masters

3.3.3 Standard DMA

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

3.4 Peripheral Register Map

3.4.1 APB Peripheral Base Address Map

Table 3-2 contains the base address for each of the APB mapped peripherals. The base address for a given peripheral is the start of the register map for the peripheral. Thus, for a peripheral, the address for a register within the peripheral is defined as the APB peripheral base address plus the register's offset.

Table 3-2: APB Peripheral Base Address Map

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
Global Control	GCR_	0x4000 0000	0x4000 03FF
System Interface	SIR_	0x4000 0400	0x4000 07FF
Function Control	FCR_	0x4000 0800	0x4000 0BFF
Watchdog Timer 0	WDT0_	0x4000 3000	0x4000 33FF
Watchdog Timer 1	WDT1_	0x4000 3400	0x4000 37FF
AES Keys	AES_KEY_	0x4000 5000	0x4000 53FF
AES	AES_	0x4000 7400	0x4000 77FF
GPIO Port 0	GPIO0_	0x4000 8000	0x4000 8FFF
GPIO Port 1	GPIO1_	0x4000 9000	0x4000 9FFF
CRC	CRC_	0x4000 F000	0x4000 FFFF
Timer 0	TMR0_	0x4001 0000	0x4001 0FFF
Timer 1	TMR1_	0x4001 1000	0x4001 1FFF
Timer 2	TMR2_	0x4001 2000	0x4001 2FFF
Timer 3	TMR3_	0x4001 3000	0x4001 3FFF
I^{2}C 0	I2C0_	0x4001 D000	0x4001 DFFF
I^{2}C 2	I2C2	0x4001 F000	0x4001 FFFF
Standard DMA	DMA	0x4002 8000	0x4002 8FFF
Flash Controller 0	FLC0_	0x4002 9000	0x4002 93FF
Flash Controller 1	FLC1_	0x4002 9400	0x4002 97FF
Internal-Cache Controller	ICC_	0x4002 A000	0x4002 A3FF
UART 0	UART0_	0x4004 2000	0x4004 2FFF
UART 2	UART2_	0x4004 4000	0x4004 4FFF
SPI0	SPI0_	0x4004 6000	0x4004 6FFF
SPI1	SPI1_	0x4004 7000	0x4004 7FFF

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
TRNG	TRNG_	0x4004 D000	0x4004 DFFF
I^{2}S	I2S_	0x4006 0000	0x4006 0FFF
ECC	ECC_	0x4010 5400	0x4010 57FF
Power Sequencer	PWRSEQ_	0x4010 6800	0x4010 6BFF
Miscellaneous Control	MCR_	0x4010 6C00	0x4010 6FFF
Timer 4 (Low Power Timer 0)	TMR4_	0x4011 4000	0x4011 4FFF

3.5 Error Correction Coding (ECC) Module

This device features an ECC module that helps ensure data integrity by detecting and correcting bit corruption of memory arrays. More specifically, this feature is single error correcting, double error detecting (SEC-DED). It corrects any single bit flip, detects 2-bit errors, and features a transparent zero wait state operation for reads.

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

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

3.5.1 SRAM

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

For example, a 32-bit by 8,192 word RAM would need a 32-bit by 2,048 word secondary RAM instance. Therefore, when ECC is enabled, each system RAM module requires an appropriately sized secondary RAM. *Table 3-3* shows the primary system RAM instances and which RAM is used as the secondary check RAM when ECC is enabled.

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

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

3.5.2 FLASH

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

3.5.3 Cache

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

3.5.4 Limitations

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

4. System, Power, Clocks, Reset

Different peripherals and subsystems use several clocks. These clocks are highly configurable by software, allowing developers to select the combination of application performance and power savings required for the target systems. In addition, support for selectable core operating voltage is provided, and the internal primary oscillator (IPO) frequency is scaled based on the specific core operating voltage range selected.

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

- 100MHz Internal Primary Oscillator (IPO)
- 7.3728MHz Internal Baud Rate Oscillator (IBRO)
- 80kHz Internal Nanoring Oscillator (INRO)
- 16MHz to 32MHz External RF Crystal Oscillator (ERFO)

4.1 Core Operating Voltage Range Selection

The MAX32675 supports three selections for the core operating voltage range (OVR). In single-supply operation, changing the OVR sets the output of the internal LDO regulator to the voltage shown in *Table 4-1*. In a dual-supply design, setting the OVR allows and external PMIC to provide the required V_{CORE} voltage dynamically. Changing the OVR also reduces the output frequency of the IPO, further reducing power consumption.

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

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

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

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

4.1.1 Setting the Operating Voltage Range

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

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

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

On each subsequent non-POR reset event:

1. Set the flash low voltage enable bit to 1 (*FLCn_CTRL.lve*) to match the setting of *PWRSEQ_LPCN.ovr* since *PWRSEQ_LPCN.ovr* is not reset.

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

4.1.2 Flash Wait States

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

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

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

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

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

Core Operating Voltage Range Setting		Core Voltage Range V_{CORE} (V)	f_{IBRO} (MHz)	System Clock Prescaler GCR_CLKCTRL.sysclk_div	System Clock f_{SYS_CLK} (MHz)	Minimum Flash Wait State Setting GCR_MEMCTRL.fws
PWRSEQ_LPCN.ovr	FLCn_CTRL.lve					
0	1	0.9	7.3728	0	7.3728	0
				1	3.6536	0
1	1	1.0	7.3728	0	7.3728	0
				1	3.6536	0
2	0	1.1	7.3728	0	7.3728	0
				1	3.6536	0
				2	2.4576	0

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

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

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

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

4.2 Oscillator Sources and Clock Switching

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

Equation 4-1: System Clock Scaling

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

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

SYS_CLK drives the Arm® Cortex®-M4 with FPU cores 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

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

Equation 4-4: AoD Clock

$$AOD\_PCLK = \frac{PCLK}{4 \times 2^{GCR\_PCLKDIV.aon\_clkdiv}}$$

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

All oscillators are reset to their POR reset default state during a POR, system reset, or watchdog reset. Oscillator settings are not reset during a soft reset or peripheral reset. *Table 4-6* shows each oscillator's enabled state for each type of reset source. *Table 4-7* details each reset source's effect on the system clock selection and the system clock prescaler settings.

Table 4-6: Reset Sources and Effect on Oscillator Status

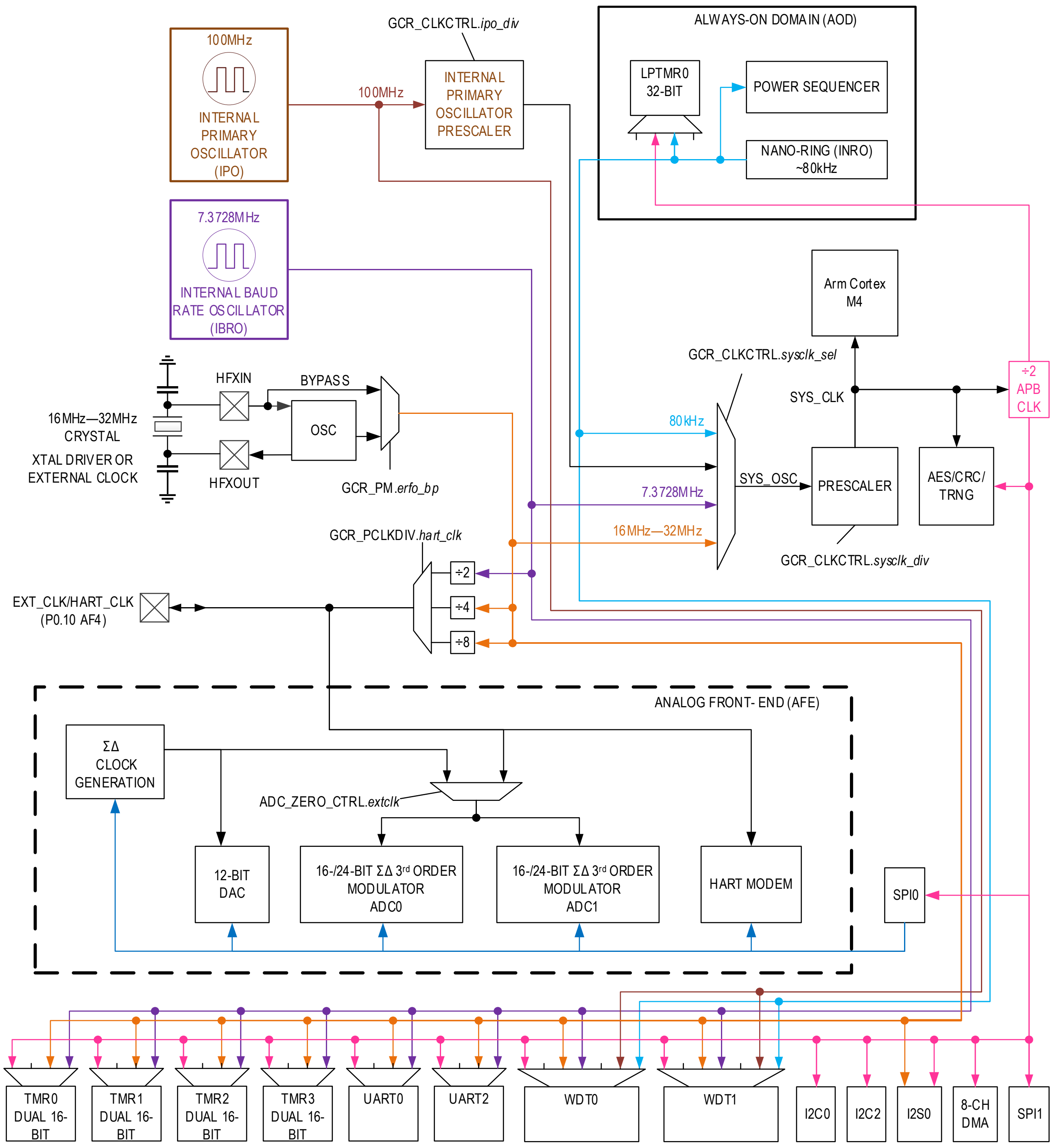
	Reset Source				
Oscillator	**POR**	**System**	**Watchdog**	**Soft**	**Peripheral**
IPO	On	On	On	Retains State	Retains State
IBRO	Off	Off	Off	Retains State	Retains State
INRO	Enabled	Enabled	Enabled	Enabled	Enabled

Table 4-7: Reset Sources and Effect on System Oscillator Selection and Prescaler

	Reset Source				
Clock Field	**POR**	**System**	**Watchdog**	**Soft**	**Peripheral**
System Oscillator *GCR_CLKCTRL.sysclk_sel*	4	4	4	Retains State	Retains State
System Clock Prescaler *GCR_CLKCTRL.sysclk_div*	1	1	1	Retains State	Retains State

Figure 4-1 shows a high-level diagram of the MAX32675 clock tree.

Figure 4-1: MAX32675 Clock Block Diagram



4.2.1 Oscillator Implementation

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

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

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

4.2.2 100MHz Internal Primary Oscillator (IPO)

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

4.2.3 16MHz to 32MHz External Radio Frequency Oscillator (ERFO)

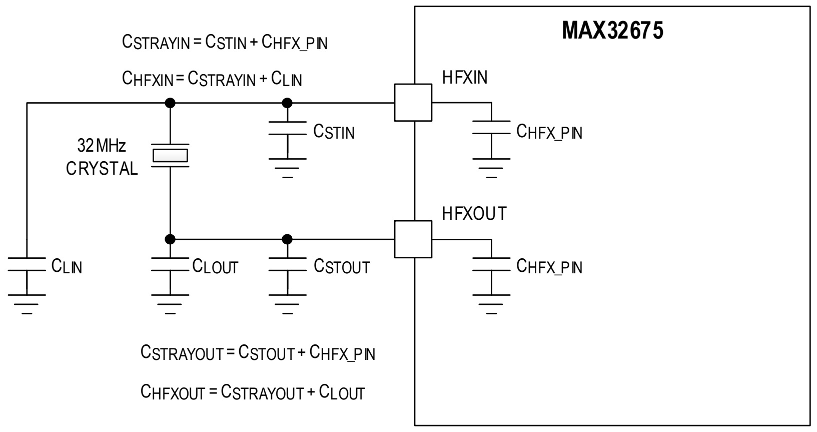
This oscillator can be selected as SYS_OSC. It is essential to use the correct capacitor values on the PCB when connecting the crystal. *Figure 4-2* depicts the method to determine the capacitor values C_{LIN} and C_{LOUT}.

This oscillator is disabled by default at power-up.

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

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

Figure 4-2: 32MHz ERFO Crystal Capacitor Determination



Equation 4-5: Determining the Load Capacitance for the ERFO

$$C_L = \left(\frac{C_{HFXIN} \times C_{HFXOUT}}{C_{HFXIN} + C_{HFXOUT}}\right)$$

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

1. The crystal load, C_L, as specified in the device data sheet electrical characteristics table is required to be 12pF. Therefore, the total capacitance seen by the crystal must equal C_L.
2. $C_{LIN} = C_{LOUT}$.
3. The device data sheet indicates the input/output pin capacitance for all pins, C_{IO} , is 4pF. Therefore, the pin capacitance of the HFXOUT and HFXIN pins are 4pF each, yielding $C_{HFX\_PIN} = 4pF$ in *Figure 4-2*.
4. The circuit board stray capacitance is represented in *Figure 4-2* as C_{STIN} and C_{STOUT}. For this example, assume C_{STIN} and C_{STOUT} are 0.5pF each.
5. Solve for C_{LOUT}
   a. $C_{LOUT} = 10pF = C_{LIN}$
6. Choose 20pF as the closest standard value for $C_{LOUT} = C_{LIN}$

4.2.4 7.3728MHz Internal Baud Rate Oscillator (IBRO)

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

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

4.2.5 80kHz Ultra-Low-Power Internal Nanoring Oscillator (INRO)

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

4.3 Operating Modes

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

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

Any low-power state can wake up to *ACTIVE* by a wake-up event shown in *Table 4-8*.

Table 4-8: Wake-Up Sources

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

4.3.1 ACTIVE

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

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

4.3.2 Low-Power Modes

4.3.2.1 SLEEP

SLEEP is a low-power mode that suspends the CPU with a fast wake-up time to *ACTIVE*. It is similar to *ACTIVE,* except the CPU clock is disabled, which temporarily prevents the CPU from executing code. All oscillators remain active if enabled, and the always-on domain (AoD) and RAM retention are enabled.

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

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

The following pseudocode places the device in *SLEEP*:

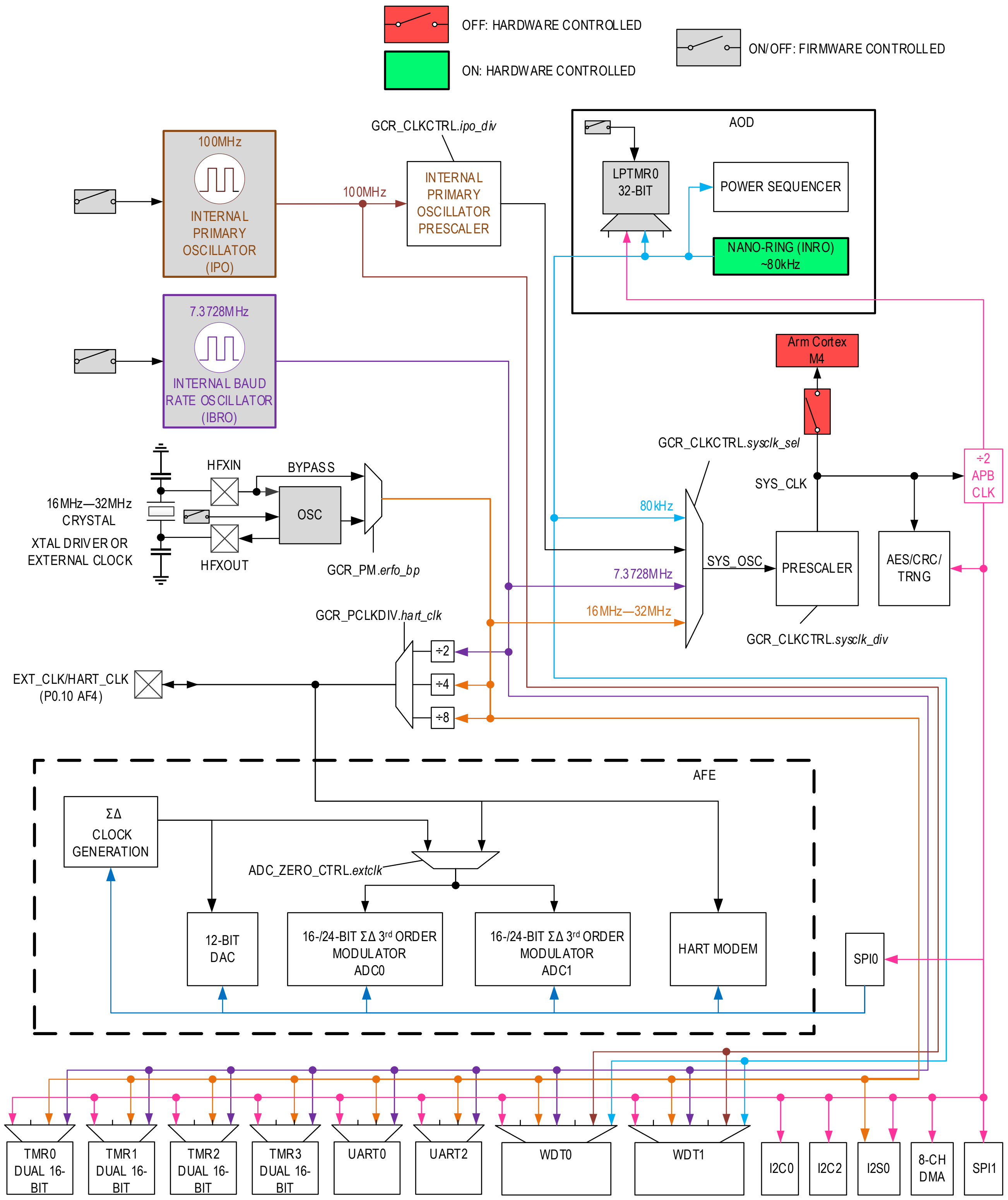
```
SCR.sleepdeep = 0; // SLEEP enabled

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

Refer to the *Arm Cortex M4 Technical Reference* for more information on SCR.

Table 4-12 and *Table 4-14* show the effects that *SLEEP* has on the various clock sources. *Figure 4-1* shows the clocks available and blocks disabled during *SLEEP*.

Figure 4-3: MAX32675 SLEEP Clock Control



4.3.2.2 DEEPSLEEP

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

The low-power peripheral, LPTMR0, can be enabled to operate in this mode. The clock source for this peripheral is selectable, but because the primary bus clocks, PCLK and HCLK, are gated off, the clock source choice is limited. See *Table 10-1* for the clock source table. This low-power peripheral is disabled/enabled before entering *DEEPSLEEP* by setting/clearing the associated bit in the *MCR_CLKDIS* register, as shown in *Table 4-9*.

Table 4-9: DEEPSLEEP Low-Power Peripheral Control Truth Table

Register Settings	**Configuration**
MCR_CLKDIS.lptmr0	
0	LPTMR0 enabled
1	LPTMR0 disabled

The watchdog timers are inactive in *DEEPSLEEP*. All internal register contents and all RAM contents are preserved. The GPIO pins retain their state in this mode.

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

To enter *DEEPSLEEP*,

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

Refer to the *Arm Cortex M4 Technical Reference* for more information on SCR.

Table 4-12 and *Table 4-14* show the effects that *DEEPSLEEP* has on the various clock sources. *Figure 4-4* shows the clock control during *DEEPSLEEP*.

4.3.2.3 BACKUP

This mode places the CPU in a static, low-power state. All oscillators are disabled except for the INRO. SYS_OSC is gated off, so PCLK and HCLK are inactive. The CPU's state is not maintained.

The low-power peripheral, LPTMR0, can be enabled to operate in this mode. The clock source for this peripheral is selectable, but because the main bus clocks PCLK and HCLK are gated off, the clock source choice is limited. See *Table 10-1* for the clock source table. This low-power peripheral is disabled or enabled before entering *BACKUP* by setting or clearing the associated bit in the *MCR_CLKDIS* register, as shown in *Table 4-10*.

Table 4-10: BACKUP Low-Power Peripheral Control Truth Table

Register Settings	**Configuration**
MCR_CLKDIS.lptmr0	
0	LPTMR0 enabled
1	LPTMR0 enabled

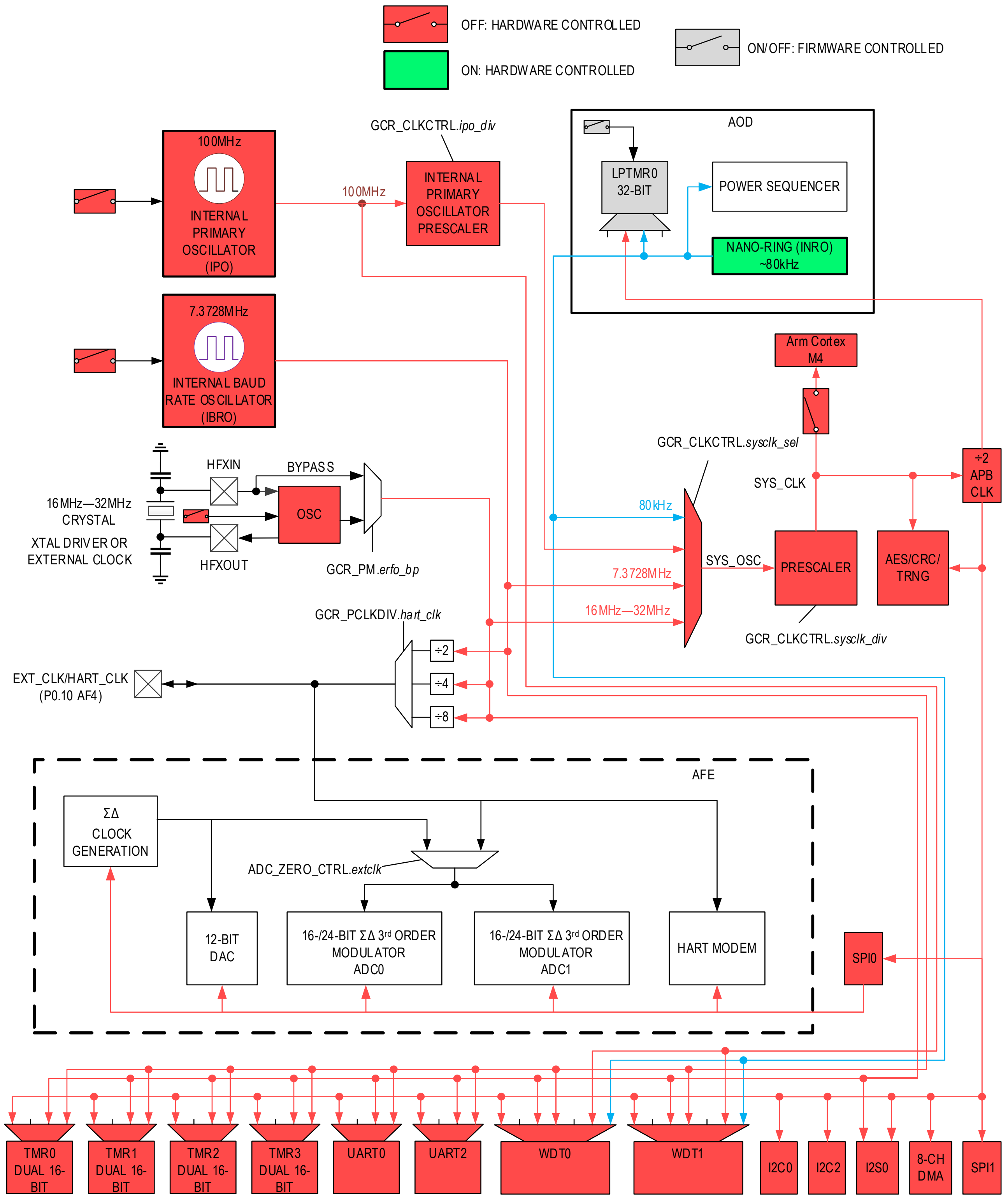
The watchdog timers are inactive in this mode.

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

To enter *BACKUP*, write 6 to *GCR_PM*.*mode*.

Table 4-12 and *Table 4-14* show the effects that *BACKUP* has on the various clock sources. *Figure 4-4* shows the clock control during *BACKUP*.

Figure 4-4: MAX32675 DEEPSLEEP and BACKUP Clock Control



4.3.2.4 STORAGE

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

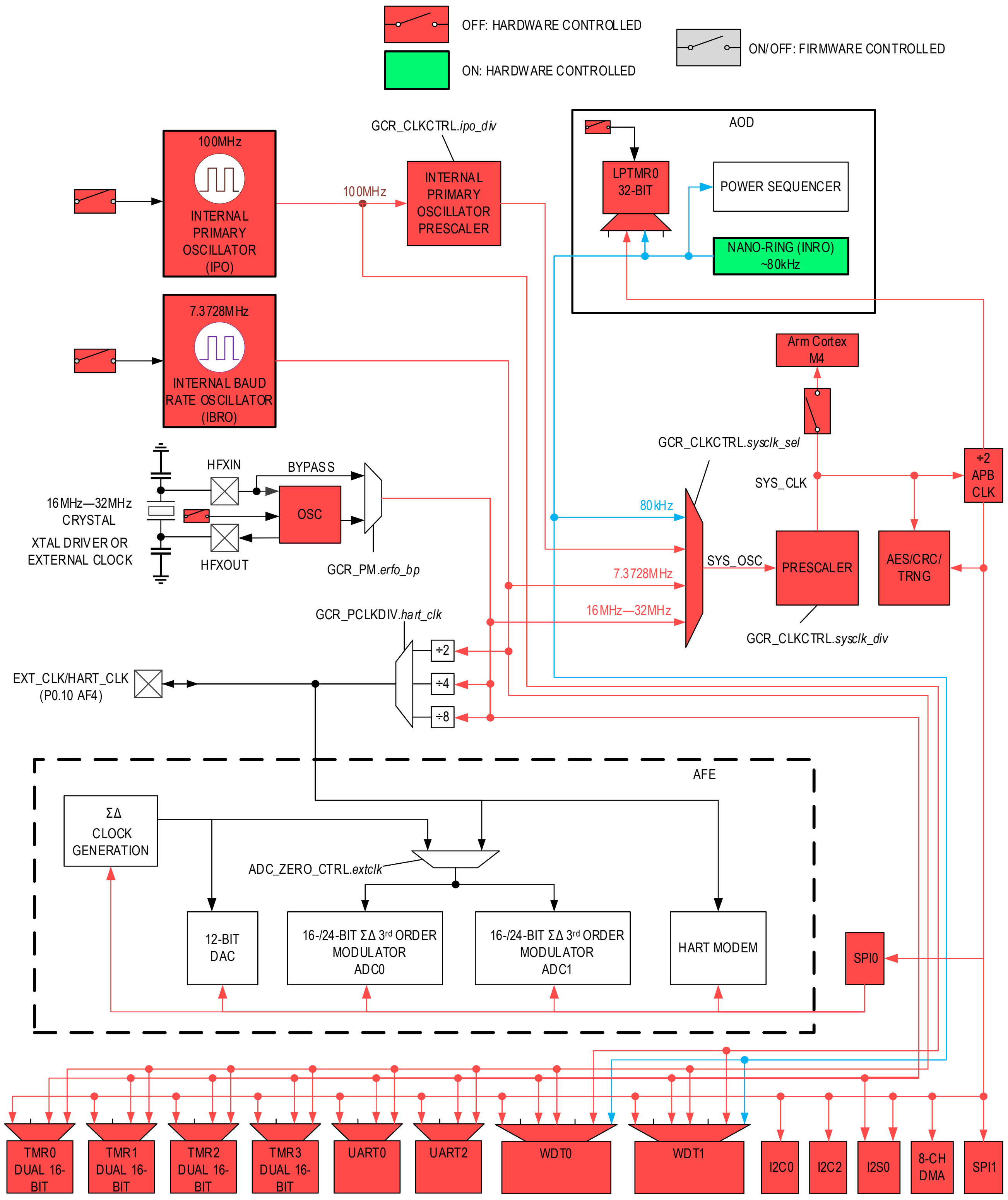
- No system RAM can be retained.
- LPTMR1 is disabled.

To enter *STORAGE*:

1. Write *PWRSEQ_LPCN*.*storage_en* = 1 to set the *STORAGE* bit.
2. Write *GCR_PM*.*mode* = 6, causing the device to enter *STORAGE* through *BACKUP*.

Table 4-12 and *Table 4-14* show the effects that *STORAGE* has on the various clock sources. *Figure 4-5* shows the clock control during *STORAGE*.

Figure 4-5: MAX32675 STORAGE Clock Control



4.4 Shutdown State

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

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

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

4.5 Device Resets

Four device resets are available:

- Peripheral Reset
- Soft Reset
- System Reset
- Power-On Reset (POR)

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

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

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

Table 4-11, *Table 4-12*, *Table 4-13*, and *Table 4-14* show the effects on the system of the four reset types and the five power modes.

Table 4-11: MAX32675 Clock Source and Global Control Register Reset Effects

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

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

1: AoD is only reset upon power cycling V_{DD} and V_{CORE}.
2: On a system reset or POR, the IPO is automatically selected as the SYS_OSC.
3: A system reset occurs when exiting *BACKUP* or *STORAGE*. The system reset does not affect the low-power peripherals.
4: Peripheral, soft, and system resets are initiated by software through the *GCR_RST0* register. System reset can also be triggered by the RSTN device pin or watchdog reset.

Table 4-12: MAX32675 Clock Source and Global Control Register Low-Power Mode Effects

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

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

1: AoD is only reset upon power cycling V_{DD} and V_{CORE}.
2: On a system reset or POR, the IPO is automatically selected as the SYS_OSC.
3: A system reset occurs when exiting *BACKUP* or *STORAGE* low-power mode. The system reset does not affect the low-power peripherals.
4: Peripheral, soft, and system resets are initiated by software through the *GCR_RST0* register. System reset can also be triggered by the RSTN device pin or watchdog reset.

Table 4-13: MAX32675 Peripheral and CPU Reset Effects

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

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

1: AoD is only reset upon power cycling V_{DD} and V_{CORE}.
2: On a system reset or POR, the IPO is automatically selected as SYS_OSC.
3: A system reset occurs when exiting *BACKUP* or *STORAGE* low-power mode. The system reset does not affect the low-power peripherals.
4: Peripheral, soft, and system resets are initiated by software through the *GCR_RST0* register. System reset can also be triggered by the RSTN device pin or watchdog reset.

Table 4-14: MAX32675 Peripheral and CPU Low-Power Mode Effects

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

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

1: AoD is only reset upon power cycling V_{DD} and V_{CORE}.
2: On a system reset or POR, the IPO is automatically selected as SYS_OSC.
3: A system reset occurs when exiting *BACKUP* or *STORAGE* low-power mode. The system reset does not affect the low-power peripherals.
4: Peripheral, soft, and system resets are initiated by software through the *GCR_RST0* register. System reset can also be triggered by the RSTN device pin or watchdog reset.

4.5.1 Peripheral Reset

As shown in *Table 4-11* and *Table 4-13*, a peripheral reset performs a reset for all peripherals. The CPU retains its state. The GPIO, watchdog timers, AoD, RAM retention, and general control registers (GCR), including the clock configuration, are unaffected.

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

4.5.2 *Soft Reset*

As shown in *Table 4-11* and *Table 4-13*, a soft reset is the same as a peripheral reset, except that it resets the GPIO to its POR state.

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

4.5.3 *System Reset*

As shown in *Table 4-11* and *Table 4-13*, a system reset is the same as a soft reset, except it also resets all GCR, resetting the clocks to their default state. In addition, 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_RST0.sys* = 1.

4.5.4 *Power-On Reset (POR)*

As shown in *Table 4-11*, *Table 4-13*, a POR resets everything in the device to its default state.

4.6 Internal Cache Controller

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

4.6.1 *Enabling ICC*

Perform the following steps to enable the ICC:

1. Set *ICC_CTRL.en* to 1.
2. Read *ICC_CTRL.rdy* until it returns 1.

4.6.2 *Disabling ICC*

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

4.6.3 *Invalidating ICC Cache*

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

4.7 ICC Registers

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

Table 4-15: Internal Cache Controller Register Summary

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

4.7.1 Register Details

Table 4-16: ICC Cache Information Register

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

Table 4-17: ICC Memory Size Register

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

Table 4-18: ICC Cache Control Register

ICC Cache Control				**ICC_CTRL**	**[0x0100]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:17	-	R/W	-	**Reserved**	
16	rdy	R	1	**Ready** This field is cleared by hardware anytime the cache as a whole is invalidated (including a POR). Hardware automatically sets this field to 1 when the invalidate operation is complete, and the cache is ready. 0: Cache invalidation in process. 1: Cache is ready. *Note: While this field reads 0, the cache is bypassed, and reads come directly from the line fill buffer.*	
15:1	-	R/W	-	**Reserved**	
0	en	R/W	0	**Cache Enable** Set this field to 1 to enable the cache. Setting this field to 0 invalidates the cache contents, and the line fill buffer handles reads. 0: Disabled. 1: Enabled.	

Table 4-19: ICC Invalidate Register

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

4.8 RAM Memory Management

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

4.8.1 On-Chip Cache Management

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

4.8.2 RAM Zeroization

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

The following RAMs can be zeroized:

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

4.8.3 RAM Low-Power Modes

RAM low-power modes and shutdown are controlled on a bank basis. The system RAM banks are shown with corresponding bank sizes and base addresses in *Table 4-20*.

Table 4-20: RAM Block Size and Base Address

System RAM Block #	Size (Words)	Base Address
sysram0	4K	2000 0000
sysram1	4K	2000 4000
sysram2	8K	2000 8000
sysram3	16K	2001 0000
sysram4	1K	2002 0000
sysram5	1K	2002 1000
sysram6	2K	2002 2000
sysram7	4K	2002 4000

4.8.4 RAM LIGHTSLEEP

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

4.8.5 RAM Shutdown

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

Note: Shutting down a primary system RAM also shuts down the corresponding check RAM even if ECC is not enabled for the system RAM.

4.9 Miscellaneous Control Registers

This control register set provides reset and clock control for the AoD peripheral LPTMR0.

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

Table 4-21: Miscellaneous Control Register Summary

Offset	Register	Name
[0x0004]	*MCR_RST*	*Reset Control Register*
[0x0024]	*MCR_CLKDIS*	*Clock Disable Register*

4.9.1 Registers Details

Table 4-22: Reset Control Register

Reset Control			**MCR_RST**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	lptmr0	W1	0	**LPTMR0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *Note: See the Device Resets section for additional information.*

Table 4-23: Clock Disable Register

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

4.10 Power Sequencer and Always-On Domain Registers

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

Table 4-24: Power Sequencer and Always-On Domain Register Summary

Offset	Register	Name
[0x0000]	*PWRSEQ_LPCN*	*Low-Power Control Register*
[0x0004]	*PWRSEQ_LPWKST0*	*GPIO0 Low-Power Wake-Up Status Flags*
[0x0008]	*PWRSEQ_LPWKEN0*	*GPIO0 Low-Power Wake-Up Enable Register*
[0x000C]	*PWRSEQ_LPWKST1*	*GPIO1 Low-Power Wake-Up Status Flags*
[0x0010]	*PWRSEQ_LPWKEN1*	*GPIO1 Low-Power Wake-Up Enable Register*
[0x0030]	*PWRSEQ_LPPWKST*	*Peripheral Low-Power Wake-Up Status Flags*
[0x0034]	*PWRSEQ_LPPWKEN*	*Peripheral Low-Power Wake-Up Enable Register*
[0x0040]	*PWRSEQ_LPMEMSD*	*RAM Shutdown Control Register*

4.10.1 Register Details

Table 4-25: Low-Power Control Register

Low-Power Control			**PWRSEQ_LPCN**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31	-	RO	0	**Reserved**
30:29	-	DNM	0	**Reserved. Do Not Modify**
28	inro_en	R/W	0	**INRO Low-Power Mode Control** This bit allows control of the INRO for low-power modes. 0: Power Sequencer controls the INRO. 1: The INRO is enabled in *ALL* low-power modes.
27:26	-	RO	0	**Reserved**
25	porvddmon_dis	DNM	0	**V_{DD} GPIO Supply POR Monitor Disable** Reserved. Do not modify this field.
24:23	-	RO	0	**Reserved**
22	vddamon_dis	DNM	0	**V_{DDA} Analog Supply Power Monitor Disable** Reserved. Do not modify this field.
21	-	RO	0	**Reserved**
20	vcoremon_dis	DNM	0	**V_{CORE} Supply Power Monitor Disable** Reserved. Do not modify this field.
19:18	-	RO	0	**Reserved**
17	vcore_ext	R/W	0	**V_{CORE} 1V Supply** Setting this bit allows the V_{CORE} device pin to be used as the 1V supply.

Low-Power Control			**PWRSEQ_LPCN**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
16	ldo_dis	R/W	1	**LDO Disable** This field initializes to 1 on a POR until the hardware determines if an external power source is connected to the V_{CORE} device pin. If no power supply is connected, this bit is set to 0 by the hardware. If a power supply is connected to the V_{CORE} device pin, the bit remains set to 1. Set this field to 1 to manually disable the LDO. 0: Enabled. 1: Disabled.
15:13	-	RO	0	**Reserved**
12	vcorepor_dis	R/W	1	**V_{CORE} POR Disable for *DEEPSLEEP* and *BACKUP*** Setting this bit to 1 blocks the POR signal to the core when the device is in *DEEPSLEEP* or *BACKUP* operation. Disconnecting the POR signal from the core during *DEEPSLEEP* and *BACKUP* prevents the core from detecting a POR event while the device is in *DEEPSLEEP* or *BACKUP*. 0: POR signal is connected to the core during *DEEPSLEEP* and *BACKUP*. 1: POR signal is not connected to the core during *DEEPSLEEP* and *BACKUP*.
11	bg_dis	R/W	1	**Bandgap Disable for *DEEPSLEEP* and *BACKUP*** Setting this field to 1 disables the Bandgap during *DEEPSLEEP* and *BACKUP*. 0: Enabled. 1: Disabled.
10	fastwk_en	R/W	0	**Fast Wake-Up Enable for *DEEPSLEEP*** Set to 1 to enable fast wake-up from *DEEPSLEEP*. When enabled, the system exits *DEEPSLEEP* faster by: • Bypassing the INRO warmup. • Reducing the warmup time for the IPO. • Reducing the warmup time for the LDO. • Code execution resumes at the next instruction after the entry to *DEEPSLEEP*. When fast wakeup is disabled code execution begins at the reset vector as if a reset occurred. 0: Disabled. 1: Enabled.
9	storage_en	R/W	0	***STORAGE* Enable** 0: Disabled. 1: Enabled. *Note: Setting this bit causes the device to enter STORAGE when setting GCR_PM.mode to BACKUP.*

Low-Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
8	retreg_en	R/W	1	**RAM Retention Regulator Enable for BACKUP** This field selects the source used to retain the RAM contents during *BACKUP* operation. Setting this field to 0 sets the V_{DD} supply for RAM retention during *BACKUP* and disables the RAM retention regulator. 0: RAM retention regulator disabled. The V_{DD} supply is used to retain the state of the internal SRAM as configured by the *PWRSEQ_LPCN.ram0ret_en, PWRSEQ_LPCN.ram1ret_en, PWRSEQ_LPCN.ram2ret_en, and PWRSEQ_LPCN.ram3ret_en* fields. 1: RAM retention regulator enabled. RAM retention in *BACKUP* is configured with the *PWRSEQ_LPCN.ram0ret_en, PWRSEQ_LPCN.ram1ret_en, PWRSEQ_LPCN.ram2ret_en, and PWRSEQ_LPCN.ram3ret_en* fields.
7	-	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 wake-up time. 0: Enabled. 1: Disabled.
5:4	ovr	R/W	0b10	**Output Voltage Range for Internal Regulator** Set this field to control the output voltage of the internal regulator, allowing selection of the internal core operating voltage and the frequency of the IPO. On POR, this field defaults to 1.1V output ± 10% with the f_{IPO} = 100MHz. *Note: If V_{CORE} is connected to an external supply voltage, this field should be modified only to set it to match the input voltage on V_{CORE}. The external supply must be equal to or greater than this field setting indication.* **Dual-Supply Operation:** 0b11: Reserved. 0b10: V_{CORE} = 1.1V, f_{IPO} = 100MHz. 0b01: V_{CORE} = 1.0V, f_{IPO} = 50MHz. 0b00: V_{CORE} = 0.9V, f_{IPO} = 12MHz. **Single-Supply Operation (V_{CORE} = GND)** 0b11: Reserved. 0b10: V_{LDO} = 1.1V, f_{IPO} = 100MHz. 0b01: V_{LDO} = 1.0V, f_{IPO} = 50MHz. 0b00: V_{LDO} = 0.9V, f_{IPO} = 12MHz.
3	ram3ret_en	R/W	0	**Sysram3 Data Retention Enable for BACKUP** Set this field to 1 to enable data retention for *sysram3* and *sysram7*. See *Table 4-20* for system RAM configuration. 0: Data retention for *sysram3*/*sysram7* address space disabled in *BACKUP*. 1: Data retention for *sysram3*/*sysram7* address space enabled in *BACKUP*. *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.*

Low-Power Control				PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
2	ram2ret_en	R/W	0	**sysram2/sysram6 Data Retention Enable for BACKUP** Set this field to 1 to enable data retention for *sysram2* and *sysram6*. See *Table 4-20* for system RAM configuration. 0: Data retention for *sysram2/sysram6* address space disabled in *BACKUP*. 1: Data retention for *sysram2/sysram6* address space enabled in *BACKUP*. *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.*	
1	ram1ret_en	R/W	0	**System RAM 1 Data Retention Enable for BACKUP** Set this field to 1 to enable data retention for *sysram1* and *sysram5*. See *Table 4-20* for system RAM configuration. 0: Data retention for *sysram1/sysram5* address space disabled in *BACKUP*. 1: Data retention for *sysram1/sysram5* address space enabled in *BACKUP*. *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.*	
0	ram0ret_en	R/W	0	**Sysram0 Data Retention Enable for BACKUP** Set this field to 1 to enable data retention for *sysram0* and *sysram4*. See *Table 4-20* for system RAM configuration. 0: Data retention for *sysram0/sysram4* address space disabled in *BACKUP*. 1: Data retention for *sysram0/sysram4* address space enabled in *BACKUP*. *Note: This field is used in conjunction with PWRSEQ_LPCN.retreg_en to control RAM retention.*	

Table 4-26: GPIO0 Low-Power Wake-Up Status Flags

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

Table 4-27: GPIO0 Low-Power Wake-Up Enable Registers

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

Table 4-28: GPIO1 Low-Power Wake-Up Status Flags

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

Table 4-29: GPIO1 Low-Power Wake-Up Enable Registers

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

Table 4-30: Peripheral Low-Power Wake-Up Status Flags

Peripheral Low-Power Wake-Up Status Flags			PWRSEQ_LPPWKST	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**

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

Table 4-31: Peripheral Low-Power Wake-Up Enable Register

Peripheral Low-Power Wake-Up Enable				PWRSEQ_LPPWKEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:1	-	RO	0	**Reserved**	
0	lptmr0	R/W	0	**LPTMR0 Wake-Up Enable** Setting this bit enables an interrupt and wake-up the device from any low-power mode when *PWRSEQ_LPPWKST.lptmr0* does not equal 0. 0: Disabled. 1: Enabled.	

Table 4-32: RAM Shutdown Control Register

RAM Shutdown Control				PWRSEQ_LPMEMSD	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:4	-	RO	0	**Reserved**	
3	ram3	R/W	0	***sysram3* and *sysram7* Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. See *Table 4-20* for System RAM configuration. *Note: See GCR_MEMCTRL register for retention mode power settings.*	
2	ram2	R/W	0	***sysram2* and *sysram6* Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. See *Table 4-20* for System RAM configuration. *Note: See GCR_MEMCTRL register for retention mode power settings.*	
1	ram1	R/W	0	***sysram1* and *sysram5* Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. See *Table 4-20* for System RAM configuration. *Note: See GCR_MEMCTRL register for retention mode power settings.*	
0	ram0	R/W	0	***sysram0* and *sysram4* Shut Down** 0: Power enabled. 1: Power shut down. Affected memory is destroyed. See *Table 4-20* for System RAM configuration. *Note: See GCR_MEMCTRL register for retention mode power settings.*	

4.11 Global Control Registers

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* for an explanation of the read and write access of each field.

Note: The Global Control Registers are only reset on a system reset or POR. A soft reset or peripheral reset does not affect these registers.

Table 4-33: Global Control Register Summary

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

4.11.1 Register Details

Table 4-34: System Control Register

System Control			**GCR_SYSCTRL**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	chkres	R	0	**ROM Checksum Calculation Pass/Fail** This bit's value is the result after setting bit *GCR_SYSCTRL.cchk*. This bit is only valid after the ROM checksum is complete and *GCR_SYSCTRL.cchk* is cleared. 0: Pass. 1: Fail.
14	swd_dis	R/W	0	**Serial Wire Debug Disable** This bit is used to disable the serial wire debug interface. 0: SWD disabled. 1: SWD enabled. *Note: This bit is only writeable if (FMV lock word is not programmed) or if (ICE lock word is not programmed and the GCR_SYSCTRL.romdone bit is not set).*

System Control				GCR_SYSCTRL [0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
13	cchk	R/W	0	**Calculate ROM Checksum** This bit is self-clearing when the ROM checksum calculation is complete, and the result is available at bit *GCR_SYSCTRL.chkres*. Writing a 0 has no effect. 0: No operation. 1: Start ROM checksum calculation.
12	romdone	R	1	**ROM Start Code Status** Reserved. Do Not Modify.
11:7	-	RO	0	**Reserved**
6	icc0_flush	R/W	0	**ICC Cache Flush** Write 1 to flush all three caches. This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect. 0: Memory flush not in progress. 1: Memory flush in progress.
5	fpu_dis	R/W	0	**Floating Point Unit Disable** 0: Enabled. 1: Disabled.
4:3	-	RO	0	**Reserved**
2:1	sbusarb	R/W	1	**System Bus Arbitration Scheme** 0: Fixed Burst. 1: Round-Robin. 2: Reserved. 3: Reserved.
0	-	RO	0	**Reserved**

Table 4-35: Reset Register 0

Reset 0				GCR_RST0 [0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31	sys	R/W	0	**System Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *See the Device Resets section for additional information.*
30	periph	R/W	0	**Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *Note: Watchdog timers, GPIO ports, the AoD, RAM retention, and the general control registers (GCR) are unaffected. See the Device Resets section for additional information.*
29	soft	R/W	0	**Soft Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *See the Device Resets section for additional information.*
28	uart2	R/W	0	**UART2 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
27:25	-	RO	0	**Reserved**
24	trng	R/W	0	**TRNG Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.

Reset 0				**GCR_RST0** / **[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
23:17	-	RO	0	**Reserved**
16	i2c0	R/W	0	**I2C0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
15	-	RO	0	**Reserved**
14	spi1	R/W	0	**SPI1 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
13	spi0	R/W	0	**SPI0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
12	-	RO	0	**Reserved**
11	uart0	R/W	0	**UART0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
10:9	-	RO	0	**Reserved**
8	tmr3	R/W	0	**TMR3 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
7	tmr2	R/W	0	**TMR2 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
6	tmr1	R/W	0	**TMR1 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
5	tmr0	R/W	0	**TMR0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
4	-	RO	-	**Reserved**
3	gpio1	R/W	0	**GPIO1 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
2	gpio0	R/W	0	**GPIO0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
1	wdt0	R/W	0	**Watchdog Timer 0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
0	dma	R/W	0	**DMA Access Block Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.

Table 4-36: System Clock Control Register

System Clock Control				**GCR_CLKCTRL** / **[0x0008]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	RO	1	**Reserved**
29	inro_rdy		0	**Internal Nano-Ring Oscillator (INRO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.

System Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
28	ibro_rdy	R	0	**Internal Baud Rate Oscillator (IBRO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
27	ipo_rdy	R	0	**Internal Primary Oscillator (IPO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
26:25	-	RO	0	**Reserved**
24	erfo_rdy	R	0	**ERFO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
23:22	-	RO	0	**Reserved**
21	ibro_vs	R/W	0	**IBRO Voltage Source Select** In *DEEPSLEEP*, the IBRO voltage is sourced by a dedicated internal 1V regulated supply. When exiting *DEEPSLEEP*, the voltage is automatically switched back to this bit's setting. 0: Dedicated internal 1V regulated supply. 1: V_{CORE} supply.
20	ibro_en	R/W	0	**IBRO Enable** 0: Disabled. 1: Enabled. and ready when *GCR_CLKCTRL.ibro_rdy* = 1.
19	ipo_en	R/W	1	**IPO Enable** 0: Disabled. 1: Enabled. and ready when *GCR_CLKCTRL.ipo_rdy* = 1.
18:17	-	DNM	0	**Reserved. Do Not Modify.**
16	erfo_en	R/W	0	**ERFO Enable** 0: Disabled. 1: Enabled. and ready when *GCR_CLKCTRL.erfo_rdy* = 1.
15:14	ipo_div	R/W	0	**IPO Prescaler** Divides the IPO clock before it is selected as SYS_OSC. 0: Divide by 1. 1: Divide by 2. 2: Divide by 4. 3: Divide by 8.
13	sysclk_rdy	R	0	**SYS_OSC Select Ready** When SYS_OSC is changed by modifying *GCR_CLKCTRL.sysclk_sel* there is a delay until the switchover is complete. This bit is cleared until the switchover is complete. 0: Switch to new clock source not yet complete. 1: SYS_OSC is the clock source selected in *GCR_CLKCTRL.sysclk_sel*.
12	-	RO	0	**Reserved**

System Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
11:9	sysclk_sel	R/W	4	**System Oscillator Source Select** Selects the system oscillator (SYS_OSC) source used to generate the system clock (SYS_CLK). Modifying this field clears *GCR_CLKCTRL.sysclk_rdy* immediately. 0: Reserved. 1: Reserved. 2: ERFO. 3: INRO. 4: IPO. 5: IBRO. 6: Reserved. 7: External Clock P0.10.
8:6	sysclk_div	R/W	0	**System Oscillator Prescaler** Sets the divider for generating SYS_CLK from the selected SYS_OSC. See *Equation 4-1* for details.
5:0	-	RO	8	**Reserved**

Table 4-37: Power Management Register

Power Management			GCR_PM	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0	**Reserved**
20	erfo_bp	R/W	0	**ERFO Bypass** This bit is set to 0 on a POR and is not affected by other resets. 0: The clock source is a crystal oscillator, driving the crystal connected between HFXIN and HFXOUT pins. 1: The clock source is a square wave and is driven into the HFXIN pin.
19:18	-	RO	0	**Reserved**
17	ibro_pd	DNM	1	**IBRO Power Down in *DEEPSLEEP*** Do not modify this field.
16	ipo_pd	DNM	1	**IPO Power Down in *DEEPSLEEP*** Do not modify this field.
15:13	-	RO	0	**Reserved**
12	erfo_pd	DNM	1	**ERFO Power Down in *DEEPSLEEP*** Do not modify this field.
11:7	-	RO	0	**Reserved**
6	lptmr0_we	R/W	0	**LPTMR0 Wake-Up Enable** Set this field to 1 to enable LPTMR0 as a wake-up source. LPTMR0 wakes the device from *SLEEP*, *DEEPSLEEP*, and *BACKUP*. 0: Disabled. 1: Enabled.
5	-	RO	0	**Reserved**

Power Management				GCR_PM [0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
4	gpio_we	R/W	0	**GPIO Wake-Up Enable** Set this field to 1 to enable all GPIO pins as potential wake-up sources. Any GPIO configured for wake-up wakes the device from *SLEEP*, *DEEPSLEEP*, *BACKUP,* and *STORAGE*. 0: Disabled. 1: Enabled.
3	-	RO	0	**Reserved**
2:0	mode	R/W	0	**Operating Mode** 0: *ACTIVE*. 1: *ACTIVE*. 2: *ACTIVE*. 3: Shutdown. 4: *BACKUP*. 5: *BACKUP*. 6: *BACKUP*. 7: Shutdown

Table 4-38: Peripheral Clock Divisor Register

Peripheral Clocks Divisor				GCR_PCLKDIV [0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	-	**Reserved**
16	div_clk_out_en	R/W	0	**Hart Clock Output Enable** Set this field to 1 to enable the Hart clock. 0: Disabled. 1: Enabled. *Note: This field enables the Hart clock in combination with the GCR_PCLKDIV.div_clk_out_ctrl field. If the GCR_PCLKDIV.div_clk_out_ctrl field is 0, the value of this field has no effect.*
15:14	div_clk_out_ctrl	R/W	0	**Hart Clock Frequency Select** Set this field to the desired Hart clock source. Setting this field to 0 disables the Hart clock. 0b00: Hart clock output disabled. 0b01: $\frac{IBRO}{2}$ 0b10: $\frac{ERFO}{4}$ 0b11: $\frac{ERFO}{8}$ *Note: When this field is set to a non-zero value the output enable field, GCR_PCLKDIV.div_clk_out_en, must be set to 1 to enable the Hart clock output signal.*
13:2	-	RO	-	**Reserved**
1:0	aon_clkdiv	R/W	0	**AoD Clock Divider** This field configures the frequency of the AoD clock. See the *Oscillator Sources and Clock Switching* section for details.

Table 4-39: Peripheral Clock Disable Register 0

Peripheral Clock Disable 0		**GCR_PCLKDIS0**		**[0x0024]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	1	**Reserved**
28	-	RO	1	**Reserved**
27:19	-	RO	1	**Reserved**
18	tmr3	R/W	1	**TMR3 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
17	tmr2	R/W	1	**TMR2 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
16	tmr1	R/W	1	**TMR1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
15	tmr0	R/W	1	**TMR0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
14	-	RO	1	**Reserved**
13	i2c0	R/W	1	**I2C0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
12:11	-	RO	1	**Reserved**
10	-	RO	1	**Reserved**
9	uart0	R/W	1	**UART0 Clock Disable** Disabling a clock peripheral disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
8	-	RO	1	**Reserved**
7	spi1	R/W	1	**SPI1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.

Peripheral Clock Disable 0		GCR_PCLKDIS0		[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
6	spi0	R/W	1	**SPI0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
5	dma	R/W	1	**DMA Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
4:2	-	RO	1	**Reserved**
1	gpio1	R/W	1	**GPIO1 Port and Pad Logic Clock Disable** Reserved. Not used.
0	gpio0	R/W	1	**GPIO0 Port and Pad Logic Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.

Table 4-40: Memory Clock Control Register

Memory Clock Control		GCR_MEMCTRL		[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:14	-	RO	0	**Reserved**
13	romls_en	R/W	0	**ROM *LIGHTSLEEP* Enable** Data is unavailable for read and write operations in *LIGHTSLEEP* but is retained. 0: *ACTIVE.* 1: *LIGHTSLEEP* enabled.
12	icc0ls_en	R/W	0	**ICC *LIGHTSLEEP* Enable** Data is unavailable for read and write operations in *LIGHTSLEEP* but is retained. 0: Disabled. 1: Enabled.
11	ram3ls_en	R/W	0	***sysram3* and *sysram7 LIGHTSLEEP* Enable** Data is unavailable for read and write operations in *LIGHTSLEEP* but is retained. 0: Disabled. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 4-20 for base address and size information.*

Memory Clock Control			GCR_MEMCTRL	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
10	ram2ls_en	R/W	0	***sysram2* and *sysram6 LIGHTSLEEP* Enable** Data is unavailable for read and write operations in *LIGHTSLEEP* but is retained. 0: Disabled. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 4-20 for base address and size information.*
9	ram1ls_en	R/W	0	***sysram1* and *sysram5 LIGHTSLEEP* Enable** Data is unavailable for read and write operations in *LIGHTSLEEP* but is retained. 0: Disabled. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 4-20 for base address and size information.*
8	ram0ls_en	R/W	0	***sysram0* and *sysram4 LIGHTSLEEP* Enable** Data is unavailable for read and write operations in *LIGHTSLEEP* but is retained. 0: Disabled. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 4-20 for base address and size information.*
7:5	-	RO	0	**Reserved**
4	ramws_en		1	**System RAM Wait State Enable** 0: No wait state. 1: Wait state enabled.
3	-	RO	0	**Reserved**
2:0	fws	R/W	5	**Program Flash Wait States** This field sets the number of wait-state SYS_OSC cycles per flash code read access. See *Flash Wait States* for details of this field's usage. 0-7: Number of flash code access wait states.

Table 4-41: Memory Zeroization Control Register

Memory Zeroization Control			GCR_MEMZ	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	icc0	W1	0	**ICC Data and Tag Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
1	ramcb	W1	0	**System RAM Check Bit Block Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Memory Zeroization Control			GCR_MEMZ	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
0	ram	W1	0	**System RAM Zeroization** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Table 4-42: System Status Flag Register

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

Table 4-43: Reset Register 1

Reset 1			GCR_RST1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23	i2s	W1	0	**I^{2}S Peripheral Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
22:18	-	RO	0	**Reserved**
17	i2c2	W1	0	**I2C2 Peripheral Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
16:15	-	W1	0	**Reserved**
14	ac	W1	0	**Auto Calibration Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
13:11	-	W1	-	**Reserved**
10	aes	W1	0	**AES Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
9	crc	W1	0	**CRC Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Reset 1				GCR_RST1 [0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
8	wdt1	W1	0	**Watchdog Timer 1 Peripheral Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
7:1	-	RO	0	**Reserved**
0	-	RO	0	**Reserved**

Table 4-44: Peripheral Clock Disable Register 1

Peripheral Clock Disable 1				GCR_PCLKDIS1 [0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	1	**Reserved**
23	i2s	R/W	1	**I^{2}S Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
22	-	RO	1	**Reserved**
21	i2c2	R/W	1	**I2C2 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
20:16	-	RO	1	**Reserved**
15	aes	R/W	1	**AES Block Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
14	crc	R/W	1	**CRC Block Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
13:12	-	RO	1	**Reserved**
11	icc0	R/W	0	**ICC Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
10:6	-	RO	1	**Reserved**

Peripheral Clock Disable 1		GCR_PCLKDIS1		[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
5	wwdt1	R/W	1	**Watchdog Timer 1 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
4	wwdt0	R/W	1	**Watchdog Timer 0 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
3	-	RO	1	**Reserved**
2	trng	R/W	1	**TRNG Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
1	uart2	R/W	1	**UART2 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
0	-	RO	1	**Reserved**

Table 4-45: Event Enable Register

Event Enable		GCR_EVENTEN		[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	tx	R/W	0	**Transmit Event (TXEV) On Send Event (SEV) Enable** When set, a SEV instruction causes a TXEV event from the CPU. 0: Disabled. 1: Enabled.
1	rx	R/W	0	**Receive Event (RXEV) Event Enable** Set this field to 1 to enable the generation of an RXEV event to wake the CPU from a WFE sleep state. 0: Disabled. 1: Enabled.
0	dma	R/W	0	**CPU DMA Count-to-Zero (CTZ) Wake-Up Enable** Allows a DMA CTZ event to generate an RXEV to wake up the CPU from a low-power mode entered using a WFE instruction. 0: Disabled. 1: Enabled.

Table 4-46: Revision Register

Revision			GCR_REVISION	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	revision	R	0x00A3	**Device Revision** This field returns the chip revision ID as a packed BCD. 0x00A3: A3 revision.

Table 4-47: System Status Interrupt Enable Register

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

Table 4-48: Error Correction Coding Error Detected Register

ECC Correctable Error Detected			GCR_ECCERR	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	flash	W1C	0	**Flash ECC Error Detected** Write to 1 to clear the flag. 0: No error. 1: Error.
1	icc0	W1C	0	**Internal Cache ECC Error Detected** Write to 1 to clear the flag. 0: No error. 1: Error.
0	ram	W1C	0	**System RAM ECC Error Detected** Write to 1 to clear the flag. 0: No error. 1: Error.

Table 4-49: Error Correction Coding Correctable Error Detected Register

ECC Correctable Error Detected			GCR_ECCCED	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	flash	W1C	0	**Flash Correctable ECC Error Detected** When set, this field indicates that there is a single correctable error in the flash bank. Write to 1 to clear the flag. 0: No error. 1: Correctable error.

ECC Correctable Error Detected			GCR_ECCCED	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
1	icc0	W1C	0	**Internal Cache Correctable ECC Error Detected** When cleared, this indicates that there is a single correctable error in the internal cache. Write to 1 to clear the flag. 0: No error. 1: Correctable error.
0	ram	W1C	0	**System RAM Correctable ECC Error Detected** When set, this field indicates that there is a single correctable error in the RAM block. Write to 1 to clear the flag. 0: No error 1: Correctable error.

Table 4-50: Error Correction Coding Interrupt Enable Register

ECC Interrupt Enable			GCR_ECCIE	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved.**
2	flash	R/W	0	**Flash ECC Error Interrupt Enable** 0: Disabled. 1: Enabled.
1	icc0	R/W	0	**Internal Cache ECC Error Interrupt Enable** 0: Disabled. 1: Enabled.
0	ram	R/W	0	**System RAM ECC Error Interrupt Enable** 0: Disabled. 1: Enabled.

Table 4-51: Error Correction Coding Address Register

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

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

4.12 Error Correction Coding Enable Register

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

Table 4-52: Error Correction Coding Enable Register Summary

Offset	Register Name	Access	Description
[0x0008]	*ECC_EN*	R/W	*Error Correction Coding Enable*

4.12.1 Register Details

Table 4-53: Error Correction Coding Enable Register

ECC Enable			ECC_EN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved**
10	flash	R/W	0	**Flash ECC Enable** 0: Disabled. 1: Enabled.
9	icc0	R/W	0	**Internal Cache ECC Enable** 0: Disabled. 1: Enabled.

ECC Enable		ECC_EN		[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
8	ram	R/W	0	**System RAM ECC Enable** 0: Disabled. 1: Enabled.
7:0	-	RO	0	**Reserved**

4.13 System Initialization Registers

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

Table 4-54: System Initialization Register Summary

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

4.13.1 Register Details

Table 4-55: System Initialization Error Status Register

System Initialization Error Status		SIR_SIR_STATUS		[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
1	cfg_err	R	*	**Configuration Error Flag** This field is set by hardware during reset if an error in the device configuration is detected. 0: Configuration valid. 1: Configuration invalid. *Note: If this field reads 1, a device error has occurred. Contact Analog Devices, Inc. technical support for additional assistance providing the address contained in* SIR_SIR_ADDR*.addr.*
0	cfg_valid	R	*	**Configuration Valid Flag** This field is set to 1 by hardware during reset if the device configuration is valid. 0: Configuration invalid. 1: Configuration valid. *Note: If this field reads 0, the device configuration is invalid, and a device error has occurred. Contact Analog Devices, Inc. technical support for additional assistance.*

Table 4-56: System Initialization Error Address Register

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

4.14 Function Control Registers

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

Table 4-57: Function Control Register Summary

Offset	Register	Description
[0x0000]	*FCR_FCTRL0*	*Function Control Register 0*
[0x0004]	*FCR_AUTOCAL0*	*Automatic Calibration 0 Register*
[0x0008]	*FCR_AUTOCAL1*	*Automatic Calibration 1 Register*
[0x000C]	*FCR_AUTOCAL2*	*Automatic Calibration 2 Register*

4.14.1 Register Details

Table 4-58: Function Control 0 Register

Function Control 0			**FCR_FCTRL0**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
25	i2c2_scl_filter_en	R/W	0	**I2C2 SCL Glitch Filter Enable** 0: Disabled. 1: Enabled.
24	i2c2_sda_filter_en	R/W	0	**I2C2 SDA Glitch Filter Enable** 0: Disabled. 1: Enabled.
23	-	RO	0	**Reserved**
22	-	RO	0	**Reserved**
21	i2c0_scl_filter_en	R/W	0	**I2C0 SCL Glitch Filter Enable** 0: Disabled. 1: Enabled.
20	i2c0_sda_filter_en	R/W	0	**I2C0 SDA Glitch Filter Enable** 0: Disabled. 1: Enabled.
19:3	-	RO	0	**Reserved**
2:0	erfo_range_sel	R/W	0	**ERFO Frequency Range Select** Set these bits to reflect the crystal frequency connected to the HFXOUT and HFXIN device pins. 0: $< 22.5MHz.$ 1: $22.5MHz\ to\ 24.5MHz.$ 2: $24.5MHz\ to\ 26.3MHz.$ 3: $26.3MHz\ to\ 28.0MHz.$ 4: $28.0MHz\ to\ 29.6MHz.$ 5: $29.6MHz\ to.31.1MHz.$ 6: $31.1MHz.to.32.6MHz.$ 7: Reserved.

Table 4-59: Automatic Calibration 0 Register

Function Control 1				**FCR_AUTOCAL0**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:23	trim	R	0	**IPO Trim Value**	
22:20	-	RO	0	**Reserved**	
19:8	gain	R/W	0	**IPO Trim Adaptation Gain**	
7:5	-	RO	0	**Reserved** Do not modify this field.	
4	atomic	R/W1	0	**IPO Trim Atomic Start** Set this bit to start an atomic automatic calibration of the IPO. The calibration runs for *FCR_AUTOCAL2.runtime* milliseconds. This bit is cleared by hardware once the calibration is complete.	
3	invert	RO	0	**IPO Trim Step Invert**	
2	load	R/W1	0	**IPO Initial Trim Load** Set this bit to load the initial trim value for the IPO from *FCR_AUTOCAL1.initial*. This bit is cleared by hardware once the load is complete.	
1	en	R/W	0	**IPO Automatic Calibration Continuous Mode Enable** 0: Disabled. 1: Enabled.	
0	sel	R/W	0	**IPO Trim Select** 0: Use default trim. 1: Use automatic calibration trim values.	

Table 4-60: Automatic Calibration 1 Register

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

Table 4-61: Automatic Calibration 2 Register

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

5. Interrupts and Exceptions

Interrupts and exceptions are managed by the Arm Cortex-M4 with FPU Nested Vector Interrupt Controller (NVIC). The NVIC handles the interrupts, exceptions, priorities and masking. *Table 5-1* details the MAX32675 interrupt vector table and describes each exception and interrupt.

5.1 Features

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

5.2 Interrupt Vector Table

Table 5-1 lists the interrupt and exception table for the MAX32675. There are 115 interrupt entries for the MAX32675, including reserved for future use interrupt placeholders. Including the 15 system exceptions for the Arm Cortex-M4 with FPU, the total number of entries is 130.

Table 5-1: MAX32675 Interrupt Vector Table

Exception/Interrupt Number	Offset	Name	Description
1	[0x0004]	Reset_IRQn	Reset
2	[0x0008]	NonMaskableInt_IRQn	Non-Maskable Interrupt
3	[0x000C]	HardFault_IRQn	Hard Fault
4	[0x0010]	MemoryManagement_IRQn	Memory Management Fault
5	[0x0014]	BusFault_IRQn	Bus Fault
6	[0x0018]	UsageFault_IRQn	Usage Fault
7:10	[0x001C]-[0x0028]	-	Reserved
11	[0x002C]	SVCall_IRQn	Supervisor Call Exception
12	[0x0030]	DebugMonitor_IRQn	Debug Monitor Exception
13	[0x0034]	-	Reserved
14	[0x0038]	PendSV_IRQn	Request Pending for System Service
15	[0x003C]	SysTick_IRQn	System Tick Timer
16	[0x0040]	PF_IRQn	Power Fail Interrupt
17	[0x0044]	WDT0_IRQn	Windowed Watchdog Timer 0 Interrupt
18:19	[0x0048]:[0x004C]	-	Reserved
20	[0x0050]	TRNG_IRQn	True Random Number Generator Interrupt
21	[0x0054]	TMR0_IRQn	Timer 0 Interrupt
22	[0x0058]	TMR1_IRQn	Timer 1 Interrupt
23	[0x005C]	TMR2_IRQn	Timer 2 Interrupt
24	[0x0060]	TMR3_IRQn	Timer 3 Interrupt
25	[0x0064]	TMR4_IRQn	Timer 4 (Low Power Timer 0) Interrupt
26:28	[0x0068]:[0x0070]	-	Reserved
29	[0x0074]	I2C0_IRQn	I^{2}C Port 0 Interrupt

Exception/Interrupt Number	Offset	Name	Description
30	[0x0078]	UART0_IRQn	UART Port 0 Interrupt
31	[0x007C]	-	Reserved
32	[0x0080]	SPI0_IRQn	SPI Port 0 Interrupt
33	[0x0084]	SPI1_IRQn	SPI Port 1 Interrupt
34:38	[0x0088]:[0x0098]	-	Reserved
39	[0x009C]	FLC0_IRQn	Flash Controller 0 Interrupt
40	[0x00A0]	GPIO0_IRQn	GPIO Port 0 Interrupt
41	[0x00A4]	GPIO1_IRQn	GPIO Port 1 Interrupt
42:43	[0x00A8]:[0x00AC]	-	Reserved
44	[0x00B0]	DMA0_IRQn	DMA0 Interrupt
45	[0x00B4]	DMA1_IRQn	DMA1 Interrupt
46	[0x00B8]	DMA2_IRQn	DMA2 Interrupt
47	[0x00BC]	DMA3_IRQn	DMA3 Interrupt
48:49	[0x00C0]:[0x00C4]	-	Reserved
50	[0x00C8]	UART2_IRQn	UART Port 2 Interrupt
51:69	[0x00CC]:[0x0114]	-	Reserved
70	[0x0118]	GPIOWAKE_IRQn	GPIO Wake-Up Interrupt
71:72	[0x011C]:[0x0120]	-	Reserved
73	[0x0124]	WDT1_IRQn	Windowed Watchdog Timer 1 Interrupt
74:77	[0x0128]:[0x0134]	-	Reserved
78	[0x0138]	I2C2_IRQn	I^{2}C Port 2 Interrupt
79:83	[0x013C]:[0x014C]	-	Reserved
84	[0x0150]	DMA4_IRQn	DMA4 Interrupt
85	[0x0154]	DMA5_IRQn	DMA5 Interrupt
86	[0x0158]	DMA6_IRQn	DMA6 Interrupt
87	[0x015C]	DMA7_IRQn	DMA7 Interrupt
88	[0x0160]	DMA8_IRQn	DMA8 Interrupt
89	[0x0164]	DMA9_IRQn	DMA9 Interrupt
90	[0x0168]	DMA10_IRQn	DMA10 Interrupt
91	[0x016C]	DMA11_IRQn	DMA11 Interrupt
92	[0x0170]	DMA12_IRQn	DMA12 Interrupt
93	[0x0174]	DMA13_IRQn	DMA13 Interrupt
94	[0x0178]	DMA14_IRQn	DMA14 Interrupt
95	[0x017C]	DMA15_IRQn	DMA15 Interrupt
96:97	[0x0180]:[0x0184]	-	Reserved
98	[0x0188]	ECC_IRQn	Error Correction Coding Block Interrupt

Exception/Interrupt Number	Offset	Name	Description
99:112	[0x018C]:[0x01C0]	-	Reserved
113	[0x01C4]	AES_IRQn	AES Block Interrupt
114	[0x01C8]	CRC_IRQn	CRC Block Interrupt
115	[0x01CC]	I2S_IRQn	I^2S Interrupt

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

The general-purpose I/O (GPIO) pins share both 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. Dynamic multiplexing between AF1, AF2, AF3, AF4 and I/O mode must be managed by the application software and the application must manage the AF and GPIO to ensure each is set up properly when switching from a peripheral to the I/O function. Refer to the device data sheet electrical characteristics table, 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 are identical across multiple Analog Devices, Inc. microcontrollers, however the behavior of several registers varies depending on the specific device. The behavior of the registers should not be assumed to be the same from one device to a different device. Specifically the registers GPIOn_PADCTRL0, GPIOn_PADCTRL1, GPIOn_HYSEN, GPIOn_SRSEL, GPIOn_DS0, and GPIOn_DS1 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^{2}C as an AF.
- Wake from low-power modes on rising edge, falling edge or both on the I/O pins.

6.1 Instances

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

Table 6-1: GPIO Pin Count

Package	GPIO	Pins
68-TQFN	GPIO1[12:7] GPIO0[31] GPIO0[21] GPIO0[19:13] GPIO0[11:6] GPIO0[5:2] (Internal Only) GPIO0[1:0]	Refer to the device data sheet for details

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

Note: MAX32675 does not support the selectable GPIO voltage supply feature.

6.2 Configuration

6.2.1 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. See the section *Bootloader* for exceptions.

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

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

6.2.2 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_EN0[pin]* = 1.
   b. *GPIOn_EN1[pin]* = 0.
   c. *GPIOn_EN2[pin]* = 0.
2. Configure the pin for pullup, pulldown, or high-impedance mode. See the *GPIOn_PS* register for pullup and pulldown selection.
   a. GPIO pins with I^{2}C as an AF only support high-impedance or a weak pulldown resistor.
3. Set *GPIOn_PADCTRL0[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 6-2*.

Table 6-2: MAX32675 Input Mode Configuration Summary

Input Mode	**Mode Select** ***GPIOn_PADCTRL0[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.

6.2.3 Output Mode Configuration

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

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

6.2.4 Serial Wire Debug Configuration

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

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

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

6.2.5 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_DS0* registers as shown in *Table 6-3*. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO0_DS.str[25], GPIO0_DS1.str[25]* both represent configuration for device pin P0.25. The drive strength currents shown are targets only. Refer to the device data sheet electrical characteristics table for details of the V_{OL_GPIO}, V_{OH_GPIO}, V_{OL_I2C} and V_{OH_I2C} parameters.

Table 6-3: Standard GPIO Drive Strength Selection

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

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

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

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

6.3 Alternate Function Configuration

Table 6-6 shows the AF selection matrix. Write the *GPIOn_EN0* 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, AF3 or AF4 is supported if all the peripheral's required functions are enabled.

Table 6-5: GPIO Mode and AF Selection

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

Table 6-6: GPIO Mode and AF Transition Selection

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

Most GPIO support one or more alternate functions which are selected with the GPIO configuration enable bits shown in *Table 6-6*. The bits that select the AF must only be changed while the pin is in one of the I/O modes (*GPIOn_EN0* = 1). The specific I/O mode must match the desired AF. For example, if a transition to AF1 is desired, first select the setting corresponding to I/O (transition to AF1). Then enable the desired mode by selecting the AF1 mode.

1. Set the GPIO configuration enable bits shown in *Table 6-6* 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 6-2* if the assigned alternate function uses the pin as an input. See *Table 6-3* if the assigned alternate function uses the pin as an output.
3. Set the GPIO configuration enable bits shown in *Table 6-6* to the desired alternate function.

Table 6-7: GPIO AF Configuration Reference

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

6.3.1 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_INEN[pin]* field to 0. This prevents any new interrupts on the pin from triggering but does not clear previously triggered (pending) interrupts. The application can disable all interrupts for GPIO by writing 0 to *GPIOn_INEN[13:0]*. To maintain previously enabled interrupts, read the *GPIOn_INEN* register and save the value to memory before setting the register to 0.
2. Clear pending interrupts by writing 1 to the *GPIOn_INTFL_CLR[pin]* bit.
3. Set *GPIOn_INTMODE[pin]* to select either level (0) or edge triggered (1) interrupts.
   a. For level triggered interrupts, the interrupt triggers on an input high or low.
   b. *GPIOn_INTPOL[pin]* = 1: Input high triggers interrupt.
   c. *GPIOn_INTPOL[pin]* = 0: Input low triggers interrupt.
   d. For edge triggered interrupts, the interrupt triggers on an edge event.
   e. *GPIOn_INTPOL[pin]* = 0: Input rising edge triggers interrupt.
   f. *GPIOn_INTPOL[pin]* = 1: Input falling edge triggers interrupt.
4. Optionally set *GPIOn_DUALEDGE[pin]* to 1 to trigger on both the rising and falling edges of the input signal.
5. Set *GPIOn_INTEN[pin]* to 1 to enable the interrupt for the pin.

6.3.2 Using GPIO for Wake-Up from Low Power Modes

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

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

Table 6-8: GPIO Wake-Up Interrupt Vector

GPIO Wake Interrupt Source	GPIO Wake Interrupt Status Register	Device Specific Interrupt Vector Number	GPIO Wake-Up Interrupt Vector
GPIO0[0:30]	*GPIOn_INTFL*	70	GPIOWAKE_IRQn

Enable low-power mode wake-up (*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_INTPOL[pin]* field. The wake-up 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 wake-up.
2. Clear pending interrupt flags by writing 0xFF to the *GPIOn_INTFL_CLR* register.
3. Activate the GPIO wake-up function by writing 1 to *GPIOn_WKEN[pin]*.
4. Configure the power manager to use the GPIO as a wake-up source by writing a 1 to the *GCR_PM.gpio_we* field.

6.4 GPIO Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers, shown in *Table 6-9*. Register names for a specific instance are defined by replacing "n" with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 6-9: GPIO Register Summary

Offset	Register Name	Description
[0x0000]	*GPIOn_EN0*	*GPIO Port n Configuration Enable Bit 0 Register*
[0x0004]	*GPIOn_EN0_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 0 Register*
[0x0008]	*GPIOn_EN0_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 0 Register*
[0x000C]	*GPIOn_OUTEN*	*GPIO Port n Output Enable Register*
[0x0010]	*GPIOn_OUTEN_SET*	*GPIO Port n Output Enable Atomic Set Register*
[0x0014]	*GPIOn_OUTEN_CLR*	*GPIO Port n Output Enable Atomic Clear Register*
[0x0018]	*GPIOn_OUT*	*GPIO Port n Output Register*
[0x001C]	*GPIOn_OUT_SET*	*GPIO Port n Output Atomic Set Register*
[0x0020]	*GPIOn_OUT_CLR*	*GPIO Port n Output Atomic Clear Register*
[0x0024]	*GPIOn_IN*	*GPIO Port n Input Register*
[0x0028]	*GPIOn_INTMODE*	*GPIO Port n Interrupt Mode Register*
[0x002C]	*GPIOn_INTPOL*	*GPIO Port n Interrupt Polarity Register*
[0x0030]	*GPIOn_INEN*	*GPIO Port n Input Enable Register*
[0x0034]	*GPIOn_INTEN*	*GPIO Port n Interrupt Enable Register*

Offset	Register Name	Description
[0x0038]	*GPIOn_INTEN_SET*	*GPIO Port n Interrupt Enable Atomic Set Register*
[0x003C]	*GPIOn_INTEN_CLR*	*GPIO Port n Interrupt Enable Atomic Clear Register*
[0x0040]	*GPIOn_INTFL*	*GPIO Port n Interrupt Status Register*
[0x0048]	*GPIOn_INTFL_CLR*	*GPIO Port n Interrupt Clear Register*
[0x004C]	*GPIOn_WKEN*	*GPIO Port n Wake-Up Enable Register*
[0x0050]	*GPIOn_WKEN_SET*	*GPIO Port n Wake-Up Enable Atomic Set Register*
[0x0054]	*GPIOn_WKEN_CLR*	*GPIO Port n Wake-Up Enable Atomic Clear Register*
[0x005C]	*GPIOn_DUALEDGE*	*GPIO Port n Interrupt Dual Edge Mode Register*
[0x0060]	*GPIOn_PADCTRL0*	*GPIO Port n Pad Control 0 Register*
[0x0064]	*GPIOn_PADCTRL1*	*GPIO Port n Pad Control 1 Register*
[0x0068]	*GPIOn_EN1*	*GPIO Port n Configuration Enable Bit 1 Register*
[0x006C]	*GPIOn_EN1_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 1 Register*
[0x0070]	*GPIOn_EN1_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 1 Register*
[0x0074]	*GPIOn_EN2*	*GPIO Port n Configuration Enable Bit 2 Register*
[0x0078]	*GPIOn_EN2_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 2 Register*
[0x007C]	*GPIOn_EN2_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 2 Register*
[0x00A8]	*GPIOn_HYSEN*	*GPIO Port n Input Hysteresis Enable Register*
[0x00AC]	*GPIOn_SRSEL*	*GPIO Port n Slew Rate Select Register*
[0x00B0]	*GPIOn_DS0*	*GPIO Port n Drive Strength Select 0 Register*
[0x00B4]	*GPIOn_DS1*	*GPIO Port n Drive Strength Select 1 Register*
[0x00B8]	*GPIOn_PS*	*GPIO Port n Pullup/Pulldown Enable Register*

6.4.1 Register Details

Table 6-10: GPIO AF 0 Select Register

GPIO AF 0 Select			**GPIOn_EN0**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	config	R/W	1	**GPIO Configuration Enable Bit 0** These bits, in conjunction with bits in *Table 6-5: GPIO Mode and AF Selection* configure the corresponding device pin for digital I/O or an AF mode. This field can be modified directly by writing to this register or indirectly through *GPIOn_EN0_SET* or *GPIOn_EN0_CLR*. *Table 6-7: GPIO AF Configuration Reference* depicts a detailed example of how each of these bits applies to each of the GPIO device pins. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect input and interrupt functionality of the associated pin.*

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

GPIO Port n Configuration Enable Atomic Set Bit 0			GPIOn_EN0_SET	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Configuration Enable Atomic Set Bit 0** Setting a bit in this field sets the corresponding bit in the *GPIOn_EN0* register. 0: No effect. 1: Corresponding bit in *GPIOn_EN0* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

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

GPIO Port n Configuration Enable Atomic Clear Bit 0			GPIOn_EN0_CLR	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1	0	**GPIO Configuration Enable Atomic Clear Bit 0** Setting a bit in this field clears the corresponding bits in the *GPIOn_EN0* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN0* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-13: GPIO Port n Output Enable Register

GPIO Port n Output Enable			GPIOn_OUTEN	[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_OUTEN_SET* or *GPIOn_OUTEN_CLR*. 0: Disabled. 1: Enabled. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

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

GPIO Port n Output Enable Atomic Set			GPIOn_OUTEN_SET	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	set	R/W1	0	**GPIO Output Enable Atomic Set** Setting a bit in this field sets the corresponding bit in the *GPIOn_OUTEN* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUTEN* set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

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

GPIO Port n Output Enable Atomic Clear				**GPIOn_OUTEN_CLR**	**[0x0014]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	clr	R/W1	0	**GPIO Output Enable Atomic Clear** Setting a bit in this field sets the corresponding bits in the *GPIOn_OUTEN* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUTEN* cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*	

Table 6-16: GPIO Port n Output Register

GPIO Port n Output				**GPIOn_OUT**	**[0x0018]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	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: This bit is ignored if the corresponding bit position in the GPIOn_OUTEN register is not set or if the pin is configured for an AF.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*	

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

GPIO Port n Output Atomic Set				**GPIOn_OUT_SET**	**[0x001C]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	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_OUTEN* set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*	

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

GPIO Port n Output Atomic Clear				**GPIOn_OUT_CLR**	**[0x0020]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	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_OUTEN* cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*	

Table 6-19: GPIO Port n Input Register

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

Table 6-20: GPIO Port n Interrupt Mode Register

GPIO Port n Interrupt Mode				**GPIOn_INTMODE**	**[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: This bit has no effect unless the corresponding bit in the GPIOn_INEN register is set.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*	

Table 6-21: GPIO Port n Interrupt Polarity Register

GPIO Port n Interrupt Polarity				**GPIOn_INTPOL**	**[0x002C]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	pol	R/W	0	**GPIO Interrupt Polarity** Interrupt polarity selection bit for the corresponding GPIO pin. Level triggered mode (*GPIOn_INTMODE*.[pin]= 0): 0: Input low (logic 0) triggers interrupt. 1: Input high (logic 1) triggers interrupt. Edge triggered mode (*GPIOn_INTMODE*.[pin]= 1): 0: Falling edge triggers interrupt. 1: Rising edge triggers interrupt. *Note: This bit has no effect unless the corresponding bit in the GPIOn_INEN register is set.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*	

Table 6-22: GPIO Port n Input Enable Register

GPIO Port n Input Enable				**GPIOn_INEN**	**[0x0030]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	en	R/W	1	**GPIO Input Enable** Setting a bit in this field connects the corresponding input pad to the specified input pin for reading the pin state using the *GPIOn_IN* register. 0: Input not connected. 1: Input pin connected. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*	

Table 6-23: GPIO Port n Interrupt Enable Registers

GPIO Port n Interrupt Enable				GPIOn_INTEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	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_INTFL_CLR register to clear pending interrupts.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*	

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

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

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

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

Table 6-26: GPIO Interrupt Status Register

GPIO Interrupt Status				GPIOn_INTFL	[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_INTFL_CLR register to clear the interrupt pending status flag.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*	

Table 6-27: GPIO Port n Interrupt Clear Register

GPIO Port Interrupt Clear			**GPIOn_INTFL_CLR**	**[0x0048]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	clr	R/W1C	0	**GPIO Interrupt Clear** Setting a bit in this field clears the associated interrupt status (*GPIOn_INTFL*). 0: No effect on the associated *GPIOn_INTFL* flag. 1: Clear the associated interrupt pending flag in the *GPIOn_INTFL* register. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-28: GPIO Port n Wake-Up Enable Register

GPIO Port n Wake-Up Enable			**GPIOn_WKEN**	**[0x004C]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	we	R/W	0	**GPIO Wake-Up Enable** Setting a bit in this field enables the corresponding GPIO pin as a wake-up from low-power modes (*SLEEP*, *DEEPSLEEP*, and *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 6-29: GPIO Port n Wake-Up Enable Atomic Set Register

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

Table 6-30: GPIO Port n Wake-Up Enable Atomic Clear Register

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

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

GPIO Port n Interrupt Dual Edge Mode				**GPIOn_DUALEDGE**	**[0x005C]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	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_INTMODE* bit must be set to edge triggered. When enabled, the associated polarity (*GPIOn_INTPOL*) setting has no effect. 0: Disabled. 1: Enabled. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*	

Table 6-32: GPIO Port n Pad Control 0 Register

GPIO Port n Pad Control 0				**GPIOn_PADCTRL0**	**[0x0060]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	config	R/W	0	**GPIO0 Pullup/Pulldown 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 corresponding GPIO with I^2C as an AF bit in GPIOn_PS set to 1, setting the same bit in this register has no effect.*	

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

GPIO Port n Pad Control 1				**GPIOn_PADCTRL1**	**[0x0064]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	config	RO	0	**Reserved**	

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

GPIO Port n Configuration Enable Bit 1				**GPIOn_EN1**	**[0x0068]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	config	R/W	0	**GPIO AF 1 Mode Select** These bits, in conjunction with bits in *Table 6-5* configure the corresponding device pin for digital I/O or an AF mode. This field can be modified directly by writing to this register or indirectly through *GPIOn_EN1_SET* or *GPIOn_EN1_CLR*. *Table 6-7* 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 input and interrupt functionality of the associated pin.*	

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

GPIO Port n Configuration Enable Atomic Set Bit 1				GPIOn_EN1_SET	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	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 6-36: GPIO Port n Configuration Enable Atomic Clear Bit 1 Register

GPIO Port n Configuration Enable Atomic Clear Bit 1				GPIOn_EN1_CLR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	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 6-37: GPIO Port n Configuration Enable Bit 2 Register

GPIO Port n Configuration Enable Bit 2				GPIOn_EN2	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	config	R/W	0	**GPIO Configuration Enable Bit 2** These bits, in conjunction with bits in *Table 6-5* configure the corresponding device pin for digital I/O or an AF mode. This field can be modified directly by writing to this register or indirectly through *GPIOn_EN2_SET* or *GPIOn_EN2_CLR*. *Table 6-7* 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 input and interrupt functionality of the associated pin.*	

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

GPIO Port n Configuration Enable Atomic Set Bit 2				GPIOn_EN2_SET	[0x0078]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	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 6-39: GPIO Port n Configuration Enable Atomic Clear Bit 2 Register

GPIO Port n Configuration Enable Atomic Clear Bit 2			**GPIOn_EN2_CLR**	**[0x007C]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	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 6-40: GPIO Port n Input Hysteresis Enable Register

GPIO Port n Input Hysteresis Enable			**GPIOn_HYSEN**	**[0x00A8]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	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 6-41: GPIO Port n Slew Rate Enable Register

GPIO Port n Slew Rate Select			**GPIOn_SRSEL**	**[0x00AC]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	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: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: GPIO with I^2C as an AF do not support Slew Rate Select. Refer to the symbols V_{OL_I2C} and V_{OH_I2C} in the device data sheet electrical characteristics table for details regarding which I/O pins support I^2C functionality.*

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

GPIO Port n Output Drive Strength Bit 0			**GPIOn_DS0**	**[0x00B0]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	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_DS0* bits for the associated GPIO pin. See the *GPIO Drive Strength* section for the selection options on these I/O pins. *Note: GPIO with I^2C as an AF only support two different drive strengths:* *0: Low output drive strength selected.* *1: High output drive strength selected.* *Refer to the symbols V_{OL_I2C} and V_{OH_I2C} in the device data sheet electrical characteristics table for details regarding which I/O pins support I^2C functionality.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-43: GPIO Port n Output Drive Strength Bit 1 Register

GPIO Port n Output Drive Strength Bit 1			**GPIOn_DS1**	**[0x00B4]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	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_DS0* bits for the associated GPIO pin. See the *GPIO Drive Strength* section for details on the selection options. *Refer to the symbols V_{OL_GPIO} and V_{OH_GPIO} in the device data sheet electrical characteristics table for details of the drive strengths for these I/O pins.* *Note: GPIO with I^2C as an AF only support two different drive strengths and do not use any bits in this register for drive strength selection. See GPIOn_DS0 for details of the two different drive strength settings.* *Refer to the symbols V_{OL_I2C} and V_{OH_I2C} in the device data sheet electrical characteristics table for details regarding which I/O pins support I^2C functionality.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 6-44: GPIO Port n Pulldown/Pullup Strength Select Register

GPIO Port n Pulldown/Pullup Strength Select			**GPIOn_PS**	**[0x00B8]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	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_PADCTRL0* the 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_PADCTRL0.*

7. Flash Controller (FLC)

The MAX32675 Flash Controller manages read, write, and erase accesses to the internal flash. It provides the following features:

- Up to 384KB total internal flash memory
- 48 pages
- 8,192 bytes per page
- 2,048 words by 128 bits per page
- 128-bit data reads and writes
- Page erase and mass erase support
- Write protection

7.1 Instances

The device provides one instance of the FLC.

The 384KB of internal flash memory is programmable through serial wire debug interface (in-system) or directly with the software application (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 of internal flash memory.

Table 7-1: MAX32675 Internal Flash Memory Organization

Instance Number	Page Number	Size (per page)	Start Address	End Address
FLC0	1	8,192 Bytes	0x1000 0000	0x1000 1FFF
	2	8,192 Bytes	0x1000 2000	0x1000 3FFF
	3	8,192 Bytes	0x1000 4000	0x1000 5FFF
	4	8,192 Bytes	0x1000 6000	0x1000 7FFF
	…	…	…	…
	47	·	0x1005 C000·	0x1005 DFFF
	48	·	0x1005 E000	0x1005 FFFF

7.2 Usage

The Flash Controller manages write and erase operations for internal flash memory and provides a lock mechanism to prevent unintentional writes to the internal flash. In-application and in-system programming, page erase and mass erase operations are supported. Flash is also sensitive to voltage. See the *Core Operating Voltage Range Selection* for details.

7.2.1 Clock Configuration

The FLC requires a 1MHz APB PCLK clock. See the section *Oscillator Sources and Clock Switching* for details. Use the FLC clock divisor to generate $f_{FLCnCLK} = 1MHz$, as shown in *Equation 7-1*. For the 100MHz Oscillator as the system clock, the *FLCn_CLKDIV*.*clkdiv* should be set to 50 (0x32).

Equation 7-1: FLC Clock Frequency

$$f_{FLCnCLK} = \frac{f_{SYS\_CLK}/2}{FLCn\_CLKDIV.clkdiv} = 1MHz$$

7.2.2 Lock Protection

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

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

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

7.2.3 Flash Write Width

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

Table 7-2: Valid Addresses Flash Writes

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

7.2.4 Flash Write

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

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

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

7.2.5 Page Erase

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

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

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

7.2.6 Mass Erase

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

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

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

7.3 Flash Error Correction Coding

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

7.4 FLC Registers

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

Table 7-3: Flash Controller Register Summary

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

7.4.1 Register Details

Table 7-4: Flash Controller Address Pointer Register

Flash Controller Address Pointer		**FLCn_ADDR**		**[0x0000]**
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 0x0010 0000.

Table 7-5: Flash Controller Clock Divisor Register

Flash Controller Clock Divisor Register		**FLCn_CLKDIV**		**[0x0004]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	-	**Reserved**
7:0	clkdiv	R/W	0x64	**Flash Controller Clock Divisor** The APB clock is divided by the value in this field to generate the FLCn peripheral clock, $f_{FLCnCLK}$. The FLCn peripheral clock must equal 1MHz. The default on all forms of reset is 0x32 (50), resulting in $f_{FLCnCLK}$ = 1MHz. The FLCn peripheral clock is only used during erase and program functions and not during read functions. See section *Clock Configuration* for details.

Table 7-6: Flash Controller Control Register

Flash Controller Control Register		**FLCn_CTRL**		**[0x0008]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:28	unlock	R/W	0	**Flash Unlock** Write the unlock code, 2, before any flash write or erase operation to unlock the Flash. Writing any other value to this field locks the internal flash. 2: Flash unlock code.
27:26	-	RO	-	**Reserved**
25	lve	R/W	0	**Low Voltage Enable** Set this field to 1 to enable low voltage operation for the flash memory. See *Core Operating Voltage Range Selection* for detailed usage information on this setting. 0: Low voltage operation disabled. 1: Low voltage operation enabled. *Note: The PWRSEQ_LPCN.ovr field must be set to 0 or 1 before setting this field to 1.*
24	pend	RO	0	**Flash Busy Flag** When this field is set, writes to all flash registers except the *FLCn_INTR* register are ignored by the Flash Controller. This bit is cleared by hardware once the flash becomes accessible. *Note: If the Flash Controller is busy (FLCn_CTRL.pend = 1), reads, writes and erase operations are not allowed and result in an access failure (FLCn_INTR.af = 1).* 0: Flash idle. 1: Flash busy.
23:16	-	RO	0	**Reserved**
15:8	erase_code	R/W	0	**Erase Code** Prior to an erase operation this field must be set to 0x55 for a page erase or 0xAA for a mass erase. The flash must be unlocked 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.
4	width	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 writes, or 128-bit writes. 0: 128-bit transactions (*FLCn_DATA[3]*, *FLCn_DATA[2]*, *FLCn_DATA[1]*, *FLCn_DATA[0]*). 1: 32-bit transactions (*FLCn_DATA[0]* only).
3	-	RO	0	**Reserved**

Flash Controller Control Register		FLCn_CTRL		[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
2	pge	R/W1	0	**Page Erase** Write a 1 to this field to initiate a page erase at the address in *FLCn_ADDR.addr*. The flash must be unlocked before attempting a page erase, see *FLCn_CTRL.unlock* for details. The flash controller hardware clears this bit when a page erase operation is complete. 0: No page erase operation in process or page erase is complete. 1: Write a 1 to initiate a page erase. If this field reads 1, a page erase operation is in progress.
1	me	R/W1	0	**Mass Erase** Write a 1 to this field to initiate a mass erase of the internal flash memory. The flash must be unlocked before attempting a mass erase, see *FLCn_CTRL.unlock* for details. The flash controller hardware clears this bit when the mass erase operation completes. 0: No operation 1: Initiate mass erase
0	wr	R/W1O	0	**Write** If this field reads 0, no write operation is pending for the flash. To initiate a write operation, set this bit to 1 and the flash controller writes to the address set in the *FLCn_ADDR* register. 0: No write operation in process or write operation complete. 1: Write 1 to initiate a write operation. If this field reads 1, a write operation is in progress. *Note: This field is protected and cannot be set to 0 by application software.*

Table 7-7: Flash Controller Interrupt Register

Flash Controller Interrupt Register		FLCn_INTR		[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	afie	R/W	0	**Flash Access Fail Interrupt Enable** Set this bit to 1 to enable interrupts on flash access failures. 0: Disabled. 1: Enabled.
8	doneie	R/W	0	**Flash Operation Complete Interrupt Enable** Set this bit to 1 to enable interrupts on flash operations complete. 0: Disabled. 1: Enabled.
7:2	-	RO	0	**Reserved**
1	af	R/W0C	0	**Flash Access Fail Interrupt Flag** This bit is set when an attempt is made to write or erase the flash while the flash is busy or locked. Only hardware can set this bit to 1. Writing a 1 to this bit has no effect. This bit is cleared by writing a 0. 0: No access failure has occurred. 1: Access failure occurred.

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

Table 7-8: Flash Controller ECC Data Register

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

Table 7-9: Flash Controller Data 0 Register

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

Table 7-10: Flash Controller Data Register 1

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

Table 7-11: Flash Controller Data Register 2

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

Table 7-12: Flash Controller Data Register 3

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

Table 7-13: Flash Controller Access Control Register

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

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

Flash Controller Write/Erase Lock 0		FLCn_WELR0		[0x0080]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	welr0	R/W1C	0	**Flash Write/Lock Bit** Each bit in this register maps to a page of the internal flash. *FLCn_WELR0*[0] maps to page 1 of the flash and *FLCn_WELR0*[31] maps to page 32. Each flash page is 8,192 bytes. Write a 1 to a bit position in this register and the corresponding page of flash is immediately locked. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding page of flash is write protected. 1: The corresponding page of flash is *not* write protected.

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

Flash Controller Write/Erase Lock 1		FLCn_WELR1		[0x0088]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	DNM	0xFFFF	**Reserved**
15:0	welr1	R/W1C	0	Each bit in this register maps to a page of the internal flash. *FLCn_WELR1*[0] maps to page 33 of the flash and *FLCn_WELR1*[15] maps to page 28. Each flash page is 8,192 bytes. Write a 1 to a bit position in this register and the corresponding page of flash is immediately locked. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding page of flash is write protected. 1: The corresponding page of flash is *not* write protected.

Table 7-16: Flash Controller Read Lock Register 0

Flash Controller Read Lock 0			**FLCn_RLR0**	**[0x0090]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	rlr0	R/W1C	0xFFFF FFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. *FLCn_RLR0*[0] maps to page 1 of the flash and *FLCn_RLR0*[31] maps to page 32 of flash. Each flash page is 8,192 bytes. Write a 1 to a bit position in this register and the corresponding page of flash is immediately read protected. The page's read protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is read protected. 1: The corresponding flash page is *not* read protected.

Table 7-17: Flash Controller Read Lock Register 1

Flash Controller Read Lock 1			**FLCn_RLR1**	**[0x0098]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	DNM	0xFFFF	**Reserved**
15:0	rlr1	R/W1C	0xFFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. *FLCn_RLR1*[0] maps to page 33 of the flash and *FLCn_RLR1*[15] maps to page 48 of flash. Each flash page is 8,192 bytes. Write a 1 to a bit position in this register and the corresponding page of flash is immediately read protected. The page's read protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is read protected. 1: The corresponding flash page is *not* read protected.

8. 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 table 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.

8.1 Instances

The DAP interface communicates through the serial wire debug (SWD), or JTAG interface signals shown in *Table 8-1*.

Table 8-1: MAX32675 DAP Instances

Pin	Alternate Function	SWD Signal	Pullup
P0.0	AF1	SWDIO	100kΩ to V_{DD}
P0.1	AF1	SWDCLK	

8.2 Access Control

8.2.1 Factory Disabled DAP

Device versions that do not provide a DAP interface have the *GCR_SYSST.icelock* field set to 1 at the factory, permanently disabling the DAP interface. No software action is needed to secure these devices.

8.2.2 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_SYSCTRL.swd_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_SYSCTRL.swd_dis* field. The *GCR_SYSCTRL.swd_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_SYSCTRL.swd_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.

8.3 Pin Configuration

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

By default, the pin associated with the bidirectional SWDIO/TMS signal is configured as a GPIO 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 8-1*. 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.

9. Standard DMA (DMA)

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

DMA transfers are one of three types:

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

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

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

9.1 Instances

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

Table 9-1: MAX32675 DMA and Channel Instances

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

9.2 DMA Channel Operation (DMA_CH)

9.2.1 DMA Channel Arbitration and DMA Bursts

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

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

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

A DMA channel is enabled using the *DMA_CHn_CTRL.en* bit.

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

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

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

Only an error condition can interrupt an ongoing data transfer.

9.2.2 DMA Source and Destination Addressing

The source and destination for DMA transfers are dictated by the request select dedicated to the peripheral instance. The *DMA_CHn_CTRL.request* 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 destination memory addresses, depending on the specific operation.

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

Table 9-2: MAX32675 DMA Source and Destination by Peripheral

DMA_CHn_CTRL.*request*	Peripheral	DMA Source	DMA Destination
0x00	Memory-to-Memory	*DMA_CHn_SRC*	*DMA_CHn_DST*
0x01	SPI0	SPI0 Receive FIFO	*DMA_CHn_DST*
0x02	SPI1	SPI1 Receive FIFO	*DMA_CHn_DST*
0x03	Reserved		
0x04	UART0	UART0 Receive FIFO	*DMA_CHn_DST*
0x05	Reserved	-	
0x06	Reserved		
0x07	I2C0	I2C0 Receive FIFO	*DMA_CHn_DST*
0x08	Reserved		
0x09	Reserved		
0x0A	I2C2	I2C2 Receive FIFO	*DMA_CHn_DST*
0x0B:0x0D	Reserved		
0x0E	UART2	UART2 Receive FIFO	*DMA_CHn_DST*
0x0F	Reserved		
0x10	AES	AES Receive FIFO	*DMA_CHn_DST*
0x11:0x1B	Reserved		
0x1C	Reserved		
0x1D	Reserved		
0x1E	I2S	I^2S Receive FIFO	
0x1F:0x20	Reserved		
0x21	SPI0	*DMA_CHn_SRC*	SPI0 Transmit FIFO
0x22	SPI1	*DMA_CHn_SRC*	SPI1 Transmit FIFO
0x23	Reserved		
0x24	UART0	*DMA_CHn_SRC*	UART0 Transmit FIFO
0x25	Reserved		
0x26	Reserved		
0x27	I2C0	*DMA_CHn_SRC*	I2C0 Transmit FIFO
0x28	Reserved		
0x29	Reserved		
0x2A	I2C2	*DMA_CHn_SRC*	I2C2 Transmit FIFO
0x2B	Reserved		
0x2C	CRC	*DMA_CHn_SRC*	CRC
0x2D	Reserved		
0x2E	UART2	*DMA_CHn_SRC*	UART2 Transmit FIFO
0x2F	Reserved		

DMA_CHn_CTRL.**request**	Peripheral	DMA Source	DMA Destination
0x30	AES	*DMA_CHn_SRC*	AES Transmit FIFO
0x31:0x3B	Reserved		
0x3C	Reserved		
0x3D	Reserved		
0x3E	I2S	*DMA_CHn_SRC*	I^{2}S Transmit FIFO
0x3F	Reserved		

9.2.3 Data Movement from Source to DMA

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

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

Register/Field	Description	Comments
DMA_CHn_SRC	Source address	If the increment enable is set, this increments on every read cycle of the burst. This field is ignored when the DMA source is a peripheral.
DMA_CHn_CNT	Number of bytes to transfer before a CTZ condition occurs	This register is decremented on each read of the burst.
DMA_CHn_CTRL.burst_size	Burst size (1-32)	This maximum number of bytes moved during the burst read.
DMA_CHn_CTRL.srcwd	Source width	This determines the maximum data width used during each read of the AHB burst (byte, two bytes, or four bytes). The actual AHB width might be less if *DMA_CHn_CNT* is not great enough to supply all the needed bytes.
DMA_CHn_CTRL.srcinc	Source increment enable	Increments *DMA_CHn_SRC*. This field is ignored when the DMA source is a peripheral.

9.2.4 Data Movement from DMA to Destination

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

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

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

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_CTRL.en, DMA_CHn_CTRL.rlden* = 0, and *DMA_CHn_STATUS.ctz_if* = 0.
2. If using memory for the destination of the DMA transfer, configure *DMA_CHn_DST* register to the starting address of the destination in memory.
3. If using memory for the source of the DMA transfer, configure the *DMA_CHn_SRC* register to the starting address of the source in memory.
4. Write the number of bytes to transfer to the *DMA_CHn_CNT* register.
5. Configure the following *DMA_CHn_CTRL* register fields in one or more instructions. Do not set *DMA_CHn_CTRL.en* to 1 or *DMA_CHn_CTRL.rlden* to 1 in this step:
   a. Configure *DMA_CHn_CTRL.request* to select the transfer operation associated with the DMA channel.
   b. Configure *DMA_CHn_CTRL.burst_size* for the desired burst size.
   c. Configure *DMA_CHn_CTRL.pri* to set the channel priority relative to other DMA channels.
   d. Configure *DMA_CHn_CTRL.dstwd* to dictate the number of bytes written in each transaction.
   e. If desired, set *DMA_CHn_CTRL.dstinc* to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.
   f. Configure *DMA_CHn_CTRL.srcwd* to dictate the number of bytes read in each transaction.
   g. If desired, set *DMA_CHn_CTRL.srcinc* to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.
   h. If desired, set *DMA_CHn_CTRL.dis_ie* = 1 to generate an interrupt when the channel becomes disabled. The channel becomes disabled when the DMA transfer completes or a bus error occurs.
   i. If desired, set *DMA_CHn_CTRL.ctz_ie* 1 to generate an interrupt when the *DMA_CHn_CNT* register is decremented to zero.
   j. If using the reload feature, configure the reload registers to set the destination, source, and count for the following DMA transaction.
      1) Load the *DMA_CHn_SRCRLD* register with the source address reload value.
      2) Load the *DMA_CHn_DSTRLD* register with the destination address reload value.
      3) Load the *DMA_CHn_CNTRLD* register with the count reload value.
   k. If desired, enable the channel timeout feature described in *Channel Timeout Detect*. Clear *DMA_CHn_CTRL.to_per* to 0x0 to disable the channel timeout feature.
6. Set *DMA_CHn_CNTRLD.en* to 1 to enable the reload feature.
7. Set *DMA_CHn_CTRL.en* = 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_CTRL.rlden*:

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

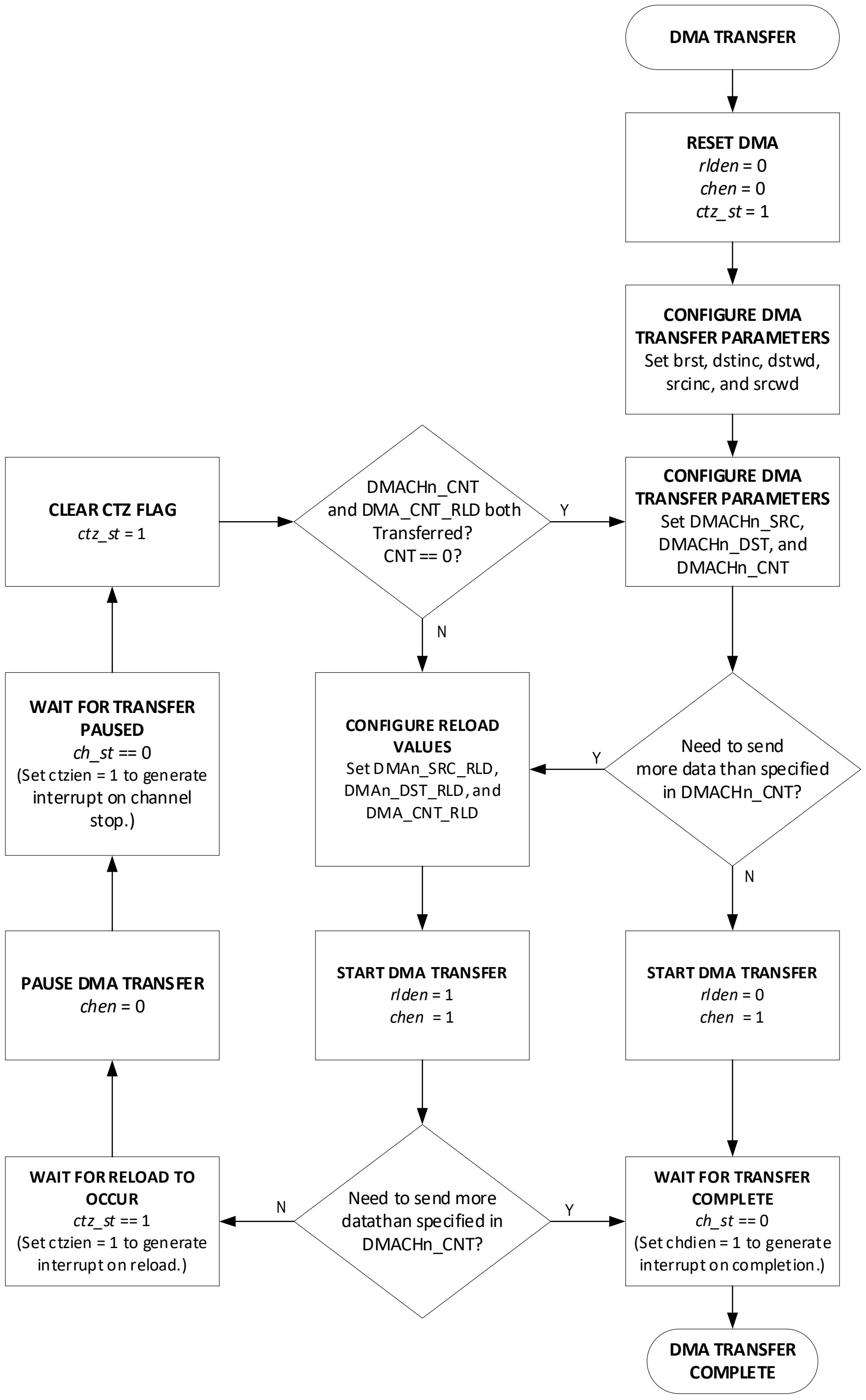
9.5 Chaining Buffers

Chaining buffers reduce the DMA ISR response time and allow 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_CTRL*
  - *DMA_CHn_SRC*
  - *DMA_CHn_DST*
  - *DMA_CHn_CNT*
  - *DMA_CHn_SRCRLD*
  - *DMA_CHn_DSTRLD*
  - *DMA_CHn_CNTRLD*

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

Figure 9-1: DMA Block-Chaining Flowchart


DMA TRANSFER
RESET DMA
rlden = 0
chen = 0
ctz_st = 1
CONFIGURE DMA TRANSFER PARAMETERS
Set brst, dstinc, dstwd, srcinc, and srcwd
CONFIGURE DMA TRANSFER PARAMETERS
Set DMACHn_SRC, DMACHn_DST, and DMACHn_CNT
CLEAR CTZ FLAG
ctz_st = 1
DMACHn_CNT and DMA_CNT_RLD both Transferred? CNT == 0?
Y
N
Need to send more data than specified in DMACHn_CNT?
Y
N
WAIT FOR TRANSFER PAUSED
ch_st == 0
(Set ctzien = 1 to generate interrupt on channel stop.)
CONFIGURE RELOAD VALUES
Set DMAn_SRC_RLD, DMAn_DST_RLD, and DMA_CNT_RLD
PAUSE DMA TRANSFER
chen = 0
START DMA TRANSFER
rlden = 1
chen = 1
START DMA TRANSFER
rlden = 0
chen = 1
WAIT FOR RELOAD TO OCCUR
ctz_st == 1
(Set ctzien = 1 to generate interrupt on reload.)
N
Need to send more datathan specified in DMACHn_CNT?
Y
WAIT FOR TRANSFER COMPLETE
ch_st == 0
(Set chdien = 1 to generate interrupt on completion.)
DMA TRANSFER COMPLETE

9.6 DMA Interrupts

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

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

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

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

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

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

9.7 Channel Timeout Detect

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

Table 9-5: DMA Channel Timeout Configuration

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

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

- If *DMA_CHn_CTRL.to_wait* = 0, the timer begins counting immediately after *DMA_CHn_CTRL.to_per* is configured to a value other than 0x0.
- If *DMA_CHn_CTRL.to_wait* = 1, the timer begins counting when the first DMA request is received from the peripheral.

The timer is reset whenever:

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

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

9.8 Memory-to-Memory DMA

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

9.9 DMA Registers

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

Table 9-6: DMA Register Summary

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

9.9.1 Register Details

Table 9-7: DMA Interrupt Enable Register

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

Table 9-8: DMA Interrupt Enable Register

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

9.10 DMA Channel Register Summary

Table 9-9: Standard DMA Channel 0 to Channel 7 Register Summary

Offset	DMA Channel	Description
[0x0100]	DMA_CH0	DMA Channel 0
[0x0120]	DMA_CH1	DMA Channel 1
[0x0140]	DMA_CH2	DMA Channel 2
[0x0160]	DMA_CH3	DMA Channel 3
[0x0180]	DMA_CH4	DMA Channel 4
[0x0200]	DMA_CH5	DMA Channel 5
[0x0220]	DMA_CH6	DMA Channel 6
[0x0240]	DMA_CH7	DMA Channel 7

9.11 DMA Channel Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 9-10*. Register names for a specific instance are defined by replacing "n" with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 9-10: DMA Channel Registers Summary

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

9.11.1 Register Details

Table 9-11: DMA Channel n Control Register

DMA Channel *n* Control			**DMA_CHn_CTRL**		**[0x0100]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31	ctz_ie	R/W	0	**CTZ Interrupt Enable** 0: Disabled. 1: Enabled. *DMA_INTFL.ch<n>_ipend* is set to 1 whenever a CTZ event occurs.	
30	dis_ie	R/W	0	**Channel Disable Interrupt Enable** 0: Disabled. 1: Enabled. *DMA_INTFL.ch<n>_ipend* bit is set to 1 whenever *DMA_CHn_STATUS.status* changes from 1 to 0.	

DMA Channel *n* Control		**DMA_CHn_CTRL**		**[0x0100]**
Bits	**Field**	**Access**	**Reset**	**Description**
29	-	RO	0	**Reserved**
28:24	burst_size	R/W	0	**Burst Size** The number of bytes transferred into and out of the DMA FIFO in a single burst. 0: 1 byte. 1: 2 bytes. 2: 3 bytes. ... 31: 32 bytes.
23	-	RO	0	**Reserved**
22	dstinc	R/W	0	**Destination Increment Enable** This bit enables the automatic increment of the *DMA_CHn_DST* register upon every AHB transaction. This bit is ignored for a DMA transmit to peripherals. 0: Disabled. 1: Enabled.
21:20	dstwd	R/W	0	**Destination Width** Indicates the width of each AHB transaction to the destination peripheral or memory (the actual width might be less than this if there are insufficient bytes in the DMA FIFO for the full width). 0: 1 byte. 1: 2 bytes. 2: 4 bytes. 3: Reserved.
19	-	RO	0	**Reserved**
18	srcinc	R/W	0	**Source Increment on AHB Transaction Enable** This bit enables the automatic increment of the *DMA_CHn_SRC* register upon every AHB transaction. This bit is ignored for a DMA receive from peripherals. 0: Disabled. 1: Enabled.
17:16	srcwd	R/W	0	**Source Width** Indicates the width of each AHB transaction from the source peripheral or memory. The actual width might be less than this if the *DMA_CHn_CNT* register indicates a smaller value. 0: 1 byte. 1: 2 bytes. 2: 4 bytes. 3: Reserved.
15:14	to_clkdiv	R/W	0	**Timeout Timer Clock Pre-Scale Select** Selects the Pre-Scale divider for the timer clock input. 0: Timer disabled. 1: $\frac{f_{HCIK}}{28}$ 2: $\frac{f_{HCLK}}{216}$f 3: $\frac{f_{HCLK}}{224}$

DMA Channel *n* Control			**DMA_CHn_CTRL**	**[0x0100]**
Bits	**Field**	**Access**	**Reset**	**Description**
13:11	to_per	R/W	0	**Timeout Period Select** Selects the number of pre-scaled clocks seen by the channel timer before a timeout condition is generated. The value is approximate because of synchronization delays between timers 0: 3 – 4. 1: 7 – 8. 2: 15 – 16. 3: 31 – 32. 4: 63 – 64. 5: 127 – 128. 6: 255 – 256. 7: 511 – 512.
10	to_wait	R/W	0	**Request DMA Timeout Timer Wait Enable** 0: Start timer immediately when enabled. 1: Delay timer start until after the first DMA transaction occurs.
9:4	request	R/W	0	**Request Select** Selects the source and destination for the transfer as shown in *Table 9-2*.
3:2	pri	R/W	0	**Channel Priority** Sets the priority of the channel relative to other channels of DMA. Channels of the same priority are serviced in a round-robin fashion. 0: Highest priority. 1: … 2: … 3: Lowest priority.
1	rlden	R/W	0	**Reload Enable** Setting this bit to 1 allows reloading the *DMA_CHn_SRC*, *DMA_CHn_DST*, and *DMA_CHn_CNT* registers with their corresponding reload registers upon CTZ. *Note: This bit is also writeable in the DMA_CHn_CNTRLD register.*
0	en	R/W	0	**Channel Enable** This bit is automatically cleared when *DMA_CHn_STATUS.status* changes from 1 to 0. 0: Disabled. 1: Enabled.

Table 9-12: DMA Status Register

DMA Channel *n* Status			**DMA_CHn_STATUS**	**[0x0104]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	DNM	0	**Reserved, Do Not Modify**
6	to_if	R/W1C	0	**Timeout Interrupt Flag** Timeout. Write 1 to clear. 0: No timeout. 1: A channel timeout has occurred.
5	-	RO	0	**Reserved**

DMA Channel *n* Status			**DMA_CHn_STATUS**	**[0x0104]**
Bits	**Field**	**Access**	**Reset**	**Description**
4	bus_err	R/W1C	0	**Bus Error** If this bit reads 1, an AHB abort occurred and the channel was disabled by hardware. Write 1 to clear. 0: No error found. 1: An AHB bus error occurred.
3	rld_if	R/W1C	0	**Reload Interrupt Flag** Reload. Write 1 to clear. 0: Reload has not occurred. 1: Reload occurred.
2	ctz_if	R/W1C	0	**CTZ Interrupt Flag** Write 1 to clear. 0: CTZ has not occurred. 1: CTZ has occurred.
1	ipend	RO	0	**Channel Interrupt Pending** 0: No interrupt. 1: Interrupt pending.
0	status	RO	0	**Channel Status** This bit indicates when it is safe to change the configuration, address, and count registers for the channel. Whenever this bit is cleared by hardware, the *DMA_CHn_CTRL.en* bit is also cleared. 0: Disabled. 1: Enabled.

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

Table 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_CTRL.dstinc* = 1, then this register is incremented on every AHB transfer cycle by one, two, or four bytes depending on the data width. If a CTZ condition occurs while *DMA_CHn_CTRL.rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_DSTRLD* register.

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

Table 9-16: DMA Channel n Source Reload Register

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

Table 9-17: DMA Channel n Destination Reload Register

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

Table 9-18: DMA Channel n Count Reload Register

DMA Channel *n* Count Reload				**DMA_CHn_CNTRLD**	**[0x011C]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31	ren	R/W	0	**Reload Enable.** Enables automatic loading of the *DMA_CHn_SRC*, *DMA_CHn_DST*, and *DMA_CHn_CNT* registers when a CTZ event occurs. Set this bit after the address reload registers are programmed. *Note: This bit is automatically cleared to 0 when reload occurs.* *Note: This bit is also seen in the DMA_CHn_CTRL register.* 0: Reload disabled. 1: Reload enabled.	
30:24	-	RO	0	**Reserved**	
23:0	cnt	R/W	0	**Count Reload Value.** If *DMA_CHn_CNTRLD*.*en* = 1, then the value of this register is loaded into *DMA_CHn_CNT* upon a CTZ condition.	

10. UART (UART/LPUART)

The universal asynchronous receiver/transmitter (UART) and the low-power universal asynchronous receiver/transmitter (LPUART) interfaces communicate with external devices using industry-standard serial communications protocols. The UARTs are full-duplex serial ports. Each UART instance can be configured independently except for shared external clock sources.

The LPUART is a special version of the peripheral that can receive characters at 9600 baud while in the low-power modes shown in *Table 10-1*. Valid characters are loaded into the receive FIFO and can generate a wake-up to ACTIVE mode if enabled.

DMA transfers are supported, and each instance has separate channels to its receive and transmit FIFOs. DMA capability is not available while in *DEEPSLEEP* or *BACKUP*.

The peripheral provides the following features:

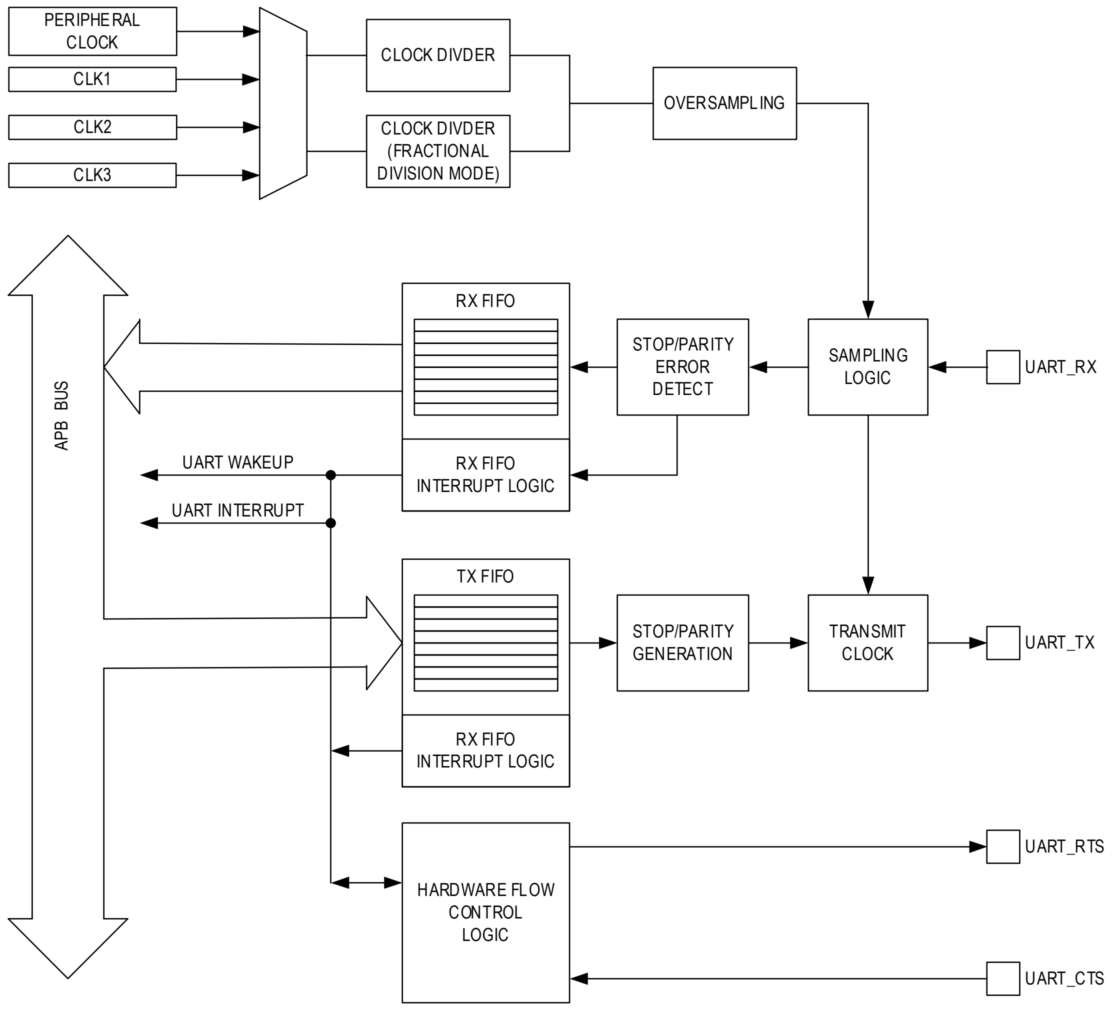
- Flexible baud rate generation up to 4Mbps with ± 2% accuracy (update for highest clock speed
- Programmable character size of 5 bits to 8 bits
- Stop bit settings of 1, 1.5, or 2 bits
- Parity settings of even, odd, mark (always 1), space (always 0), and no parity
- Automatic parity error detection with selectable parity bias
- Automatic framing error detection
- Separate 8-byte transmit and receive FIFOs
- Flexible interrupt conditions
- Hardware flow control for RTS and CTS
- DMA capable

The LPUART, if available, provides these additional features:

- Ability to receive characters in *DEEPSLEEP* and *BACKUP*
- Fractional baud rate divisor settings to allow greater accuracy at slow baud rates
- Wake-up to *ACTIVE* on multiple receive FIFO conditions

Figure 10-1 shows a high-level block diagram.

Figure 10-1: UART/LPUART Block Diagram



10.1 Instances

Instances of the peripheral are shown in *Table 10-1*. Both the UART and LPUART are functionally very similar, so for common functionality they are referred to as just UART. The LPUART instances support the fractional division mode (FDM) and are identified by the phrase "instances which support the FDM function."

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

Table 10-1: MAX32675 UART/LPUART Instances

Instance	FDM Support	Power Modes	Peripheral Clock	Ext Clock Signal Name	CLK2	CLK3	C_TX_FIFO_DEPTH C_RX_FIFO_DEPTH
UART0	NO	*ACTIVE, SLEEP*	PCLK	N/A	IBRO	ERFO	8/8
UART2							

10.2 DMA

Each peripheral instance supports DMA transfers in both the transmit and receive directions with a 32-byte transmit FIFO with a dedicated DMA channel and a 32-byte receive FIFO with a dedicated DMA channel.

The DMA channels are configured using the DMA Configuration register, *UARTn_DMA*. Enable the receive FIFO DMA channel by setting *UARTn_DMA.rx_en* to 1 and enable the transmit FIFO DMA channel by setting *UARTn_DMA.tx_en* to 1. DMA transfers are automatically triggered based on the number of bytes in the receive or transmit FIFO

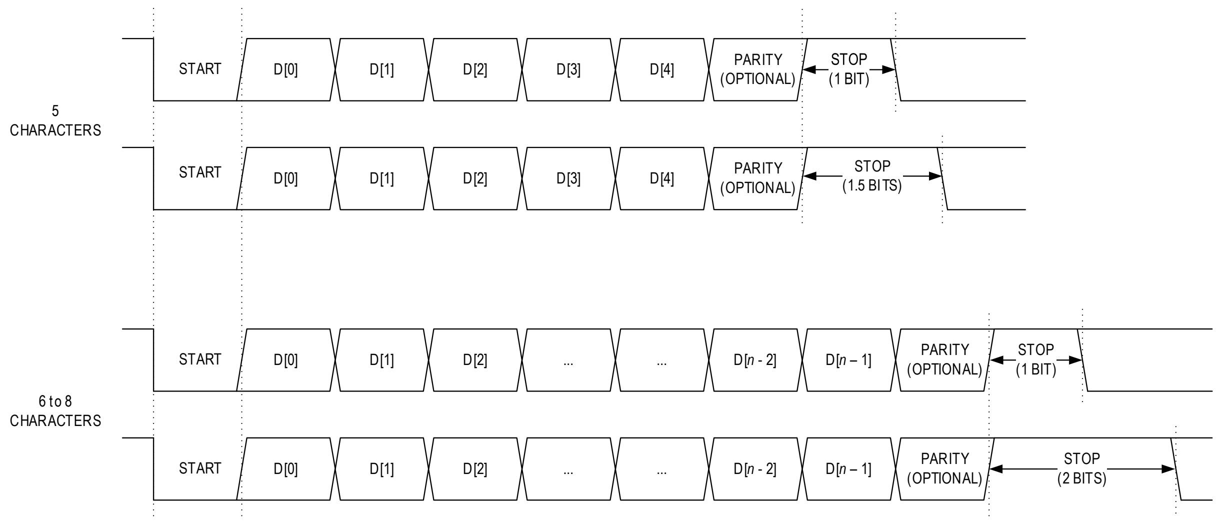
The following describes the behavior of the receive and transmit DMA requests when the DMA function is enabled.

- Receive DMA request is asserted when the number of valid bytes in the receive FIFO is greater than or equal to the receive FIFO threshold.
- Transmit DMA request is asserted when the number of valid bytes in the transmit FIFO is less than the transmit FIFO threshold.

10.3 UART Frame

Figure 10-2 shows a UART frame. Character sizes of 5 to 8 bits are configurable through the *UARTn_CTRL.char_size* field. Stop bits are configurable for as 1 or 1.5 bits for 5-character frames and 1 or 2 stop bits for 6, 7 or 8-character frames. Parity support includes even, odd, mark, space, or none.

Figure 10-2: UART Frame Structure



10.4 FIFOs

Separate read (receive FIFO) and write (transmit FIFO) FIFOs are provided. They are both accessed through the same UARTn_FIFO.*data* field. The current level of the transmit FIFO is read from *UARTn_STATUS.tx_lvl*, and the current level of the receive FIFO is read from *UARTn_STATUS.rx_lvl*.

10.4.1 Transmit FIFO Operation

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

Writes to the transmit FIFO are ignored while *UARTn_STATUS.tx_lvl* = C_TX_FIFO_DEPTH.

10.4.2 Receive FIFO Operation

Reads of *UARTn_FIFO.data* return the character values in the receive FIFO and decrement *UARTn_STATUS.rx_lvl*. Data for character sizes less than 7 bits are right justified. An overrun event occurs if a valid frame, including parity, is detected while *UARTn_STATUS.rx_lvl* = C_RX_FIFO_DEPTH shown in *Table 10-1*. In this case the frame is discarded.

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

10.4.3 Flushing

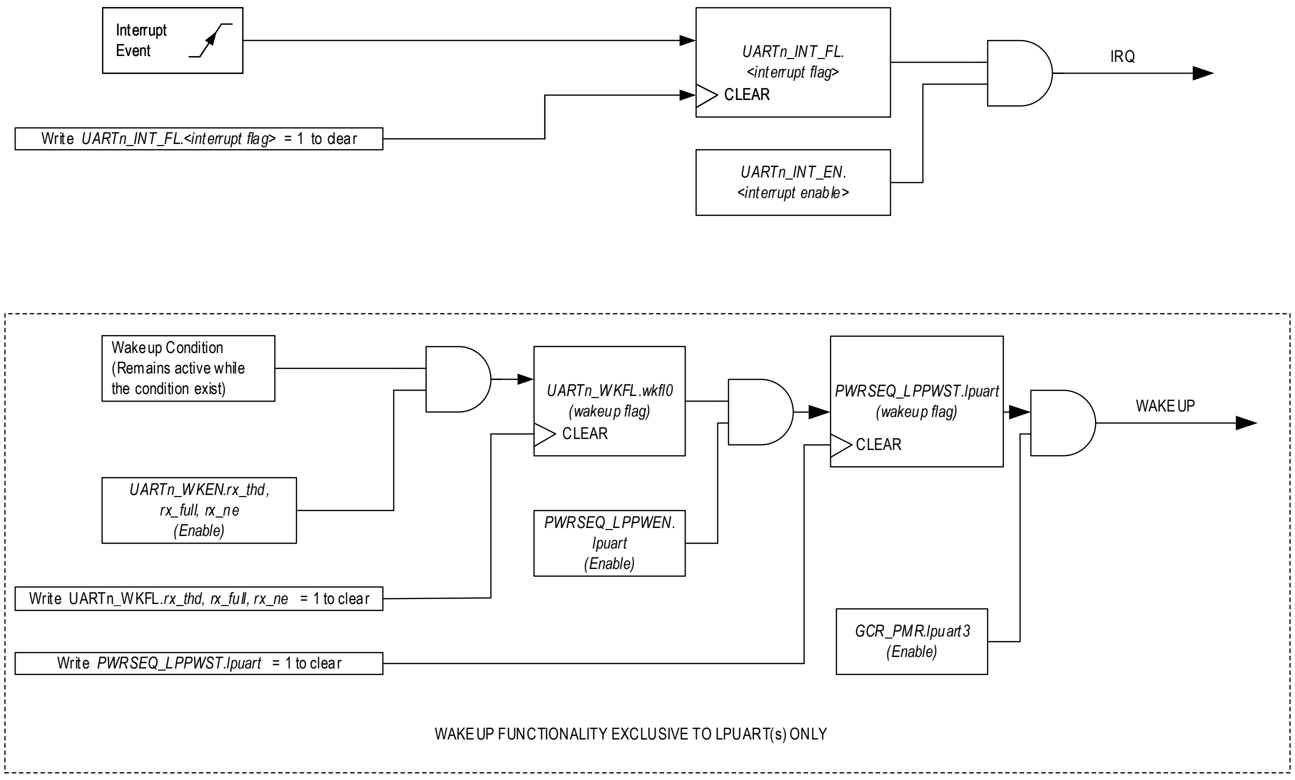
The FIFOs are flushed on the following conditions:

- Setting *UARTn_CTRL.rx_flush* flushes the receive FIFO by setting its pointer to 0.
- Setting *UARTn_CTRL.tx_flush* flushes the transmit FIFO by setting its pointer to 0.
- Setting the corresponding field in the *GCR_RST0* register causes a peripheral reset which flushes both the transmit FIFO and receive FIFO by erasing their contents and setting their pointers to 0.

10.5 Interrupt Events

The peripheral generates interrupts for the events shown in *Figure 10-3*. Unless noted otherwise, each instance has its own independent set of interrupts, and higher-level flag and enable fields as shown in *Table 10-2*.

Figure 10-3: MAX32675 UART Interrupt and Wake-Up Functional Diagram


Interrupt Event
UARTn_INT_FL. <interrupt flag> CLEAR
IRQ
Write UARTn_INT_FL.<interrupt flag> = 1 to clear
UARTn_INT_EN. <interrupt enable>
Wakeup Condition (Remains active while the condition exist)
UARTn_WKFL.wkfl0 (wakeup flag) CLEAR
PWRSEQ_LPPWST.lpuart (wakeup flag) CLEAR
WAKE UP
UARTn_WKEN.rx_thd, rx_full, rx_ne (Enable)
PWRSEQ_LPPWEN. lpuart (Enable)
Write UARTn_WKFL.rx_thd, rx_full, rx_ne = 1 to clear
Write PWRSEQ_LPPWST.lpuart = 1 to clear
GCR_PMR.lpuart3 (Enable)
WAKE UP FUNCTIONALITY EXCLUSIVE TO LPUART(s) ONLY

Some activity may cause more than one event, setting one or more event flags. An event interrupt occurs if the corresponding interrupt enable is set. The flags must be cleared by software.

Table 10-2: MAX32675 Interrupt Events

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

10.5.1 Frame Error

A frame error is generated when the UART sampling circuitry detects an invalid bit. Each bit is sampled three times at the times described earlier and can generate frame errors on the state of the start, stop, and optionally parity and data bits.

The frame error criteria are different depending on whether or not FDM is present in that instance and enabled or not as shown in *Table 10-3* and *Table 10-4*.

Table 10-3: Frame Error Detection for Standard UARTs and LPUART

UARTn_CTRL .par_en	*UARTn_CTRL .par_md*	*UARTn_CTRL .par_eo*	Start Samples	Data Samples	*Parity*	*Stop*
0	N/A	N/A	3 of 3 must be 0	2/3 must match	Not Present	3 of 3 must be 1
1	0	0			3/3 = 1 if even number "1" 3/3 = 0 if odd number "0"	
	0	1			3/3 = 1 if odd number "1" 3/3 = 0 if even number "0"	
	1	0			3/3 = 1 if even number "0" 3/3 = 0 if odd number "1"	
	1	1			3/3 = 1 if odd number "0" 3/3 = 0 if even number "1"	

Table 10-4: Frame Error Detection for LPUARTs with UARTn_CTRL.fdm = 1 and UARTn_CTRL.dpfe_en = 1

UARTn_CTRL .par_en	*UARTn_CTRL .par_md*	*UARTn_CTRL .par_eo*	Start Samples	Data Samples	Parity Samples	Stop Samples
0	N/A	N/A	3 of 3 must be 0	3 of 3 must match	Not Present	3 of 3 must be 1
1	0	0			3 of 3 = 1 if even number of 1s 3 of 3 = 0 if odd number 0s	
	0	1			3 of 3 = 1 if odd number 1s 3 of 3 = 0 if even number 0s	
	1	0			3 of 3 = 1 if even number 0s 3 of 3 = 0 if odd number 1s	
	1	1			3 of 3 = 1 if odd number 0s 3 of 3 = 0 if even number 1s	

10.5.2 Parity Error

Set *UARTn_CTRL.par_en* = 0 to enable parity checking of the received frame. If the calculated parity does not match the parity bit, the corresponding interrupt flag is set.

10.5.3 CTS Signal Change

If hardware flow control is disabled, the CTS pin is a general-purpose input and the RTS pin is a general-purpose output. In that case, a CTS signal change event occurs on each rising or falling edge of the sampling on the CTS input pin.

The CTS sample value is reset to 1 after power-on reset. While the UART baud clock is running, the CTS sampling process continues if *UARTn_CTRL.cts_dis* is not 1. The sampled value is set to 1 when the UART baud clock stops running because *UARTn_CTRL.bclken* has been cleared to 0 again.

10.5.4 Overrun

An overrun condition occurs if a valid frame is received when the receive FIFO is full. The interrupt flag is set at the end of the stop bit and the frame is discarded.

10.5.5 Receive FIFO Threshold

An receive threshold event occurs when the valid frame is received that causes the number of bytes to exceed the configured threshold *UARTn_CTRL.rx_thd_val*.

10.5.6 Transmit FIFO One Byte Remaining

This event occurs when the transmit FIFO transitions from *UARTn_STATUS.tx_lvl* = 2 to *UARTn_STATUS.tx_lvl* = 1.

10.5.7 Transmit FIFO Half-Empty

The transmit FIFO Half-Empty event occurs when *UARTn_STATUS.tx_lvl* transitions from C_TX_FIFO_DEPTH/2+1 to C_TX_FIFO_DEPTH/2.

10.6 Wake-Up Events

Instances that support fractional division mode can receive characters while in the low-power modes listed in *Table 10-1*. If enabled, any of the wake-up sources in *Table 10-5* causes the device to exit the current low-power mode and return to *ACTIVE*.

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

Table 10-5: MAX32675 Wake-Up Events

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

10.6.1 Receive FIFO Threshold

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

10.6.2 Receive FIFO Full

This condition persists while *UARTn_STATUS.rx_lvl* ≥ 8.

10.6.3 Receive Not Empty

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

10.7 Resets

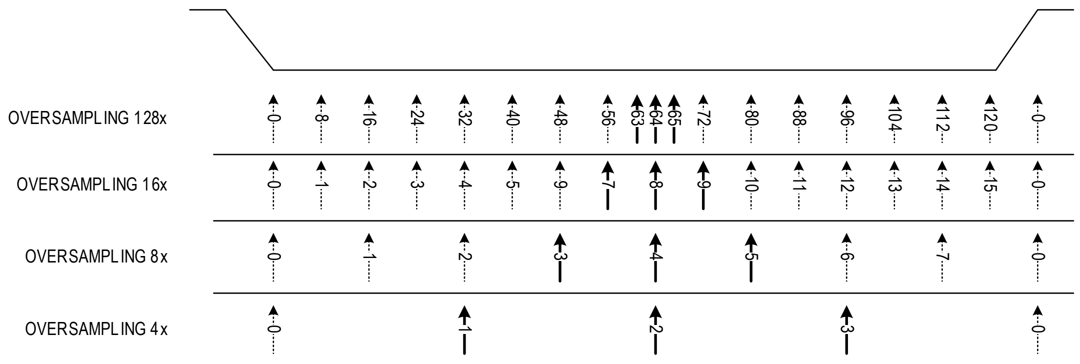
The following stimuli can reset the peripheral.

- The peripheral remains in reset while *UARTn_CTRL.bclken* = 0.
- The peripheral remains in reset while *UARTn_CTRL.bclkrdy* = 0 after setting *UARTn_CTRL.bclken* to 1.
- Any write to *UARTn_CLKDIV.clkdiv* while *UARTn_CTRL.bclken* is 1.
- Any write to *UARTn_OSR.osr* while *UARTn_CTRL.bclken* is 1.

10.8 Receive Sampling

Each bit of a frame is oversampled to improve noise immunity. The oversampling rate (OSR) is configurable with the *UARTn_OSR.osr* field. In most cases, the bit is evaluated based on three samples at the midpoint of each bit time as shown in *Figure 10-4*.

Figure 10-4: Oversampling Example



Whenever *UARTn_CLKDIV.clkdiv* < 0x10 (i.e., a division rate less than 8.0), OSR is not used and oversampling rate is adjusted to full sampling by hardware. That means receive is sampled in every clock cycle, despite of the value of OSR.

For 9600 baud low-power operation, the dual-edge sampling mode must be enabled (*UARTn_CTRL.desm* = 1).

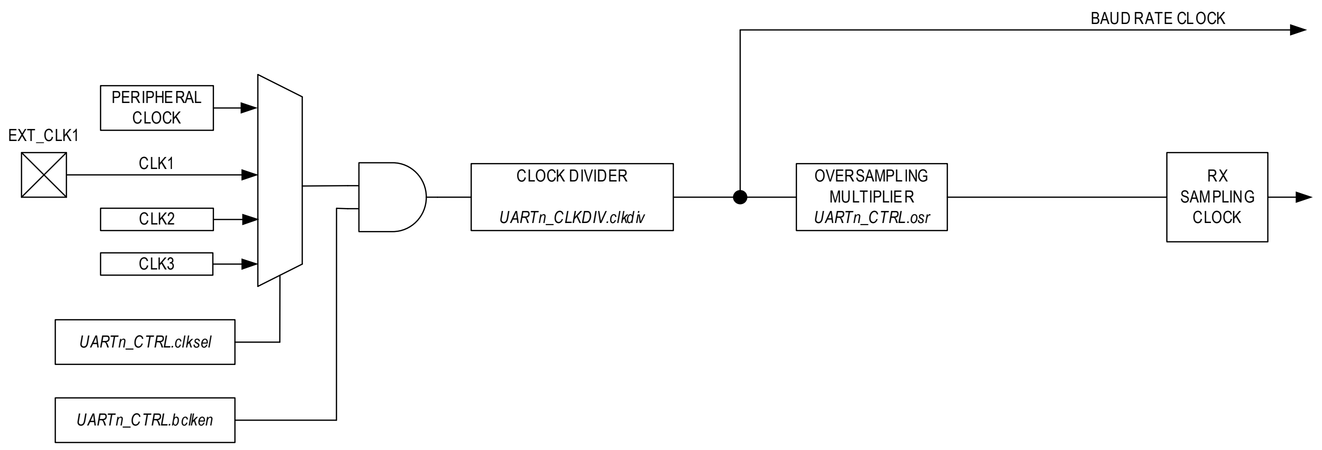
10.9 Baud Rate Generation

The baud rate is determined by the selected UART clock source and value of the clock-divider circuit. Multiple clock sources are available for each instance and can be configured independently for each UART instance as shown in *Table 10-1*.

10.9.1 Clock Sources (FDM Not Supported)

In UART instances, the clock sources are disabled in all low-power modes except *SLEEP*. In this case, the UART is not running and cannot be used as a wake-up source.

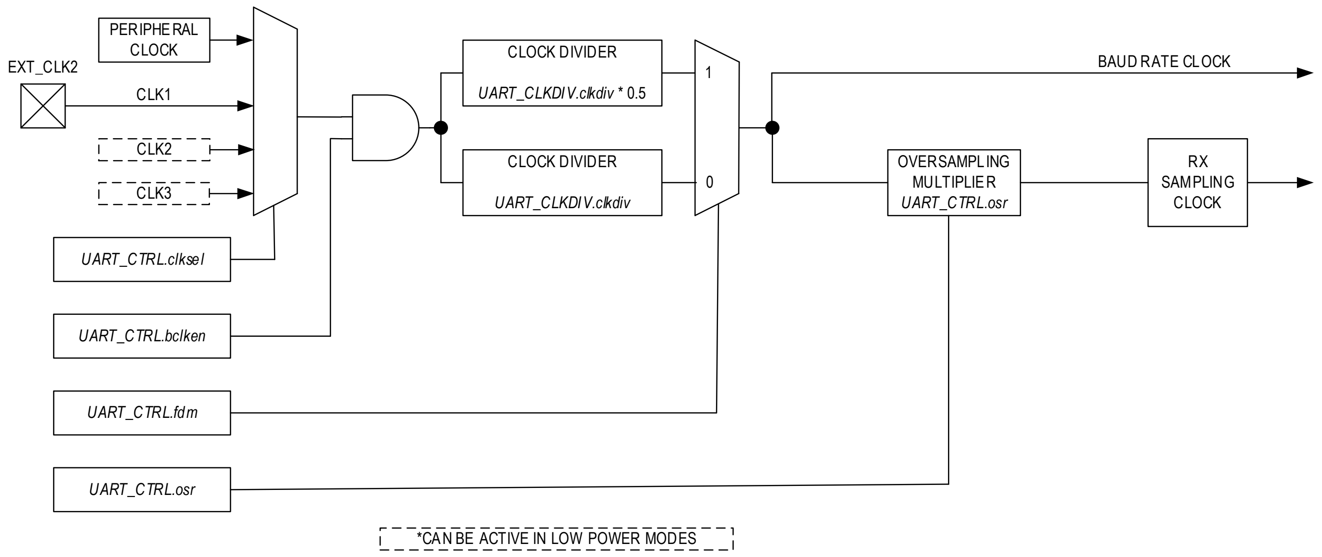
Figure 10-5: UART Timing Generation (FDM Not Supported)



10.9.2 Clock Sources (FDM Supported)

Instances that support FDM can be configured to operate the LPUART receiver at 9600 baud using the 32kHz RTC as shown in *Table 10-1*. This clock source can be configured to remain active in *DEEPSLEEP* and *BACKUP*, allowing the device to receive data and serve as a wake-up source while power consumption is at a minimum.

Figure 10-6: LPUART Timing Generation (FDM Supported)



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

10.10 Baud Rate Calculation

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

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

- *UARTn_CTRL.fdm* = 0: *UARTn_CLKDIV.clkdiv* = INT [(UART clock)/Baud rate]
- *UARTn_CTRL.fdm* = 1: *UARTn_CLKDIV.clkdiv* = INT [(UART clock)/Baud rate x 2]

For example, in a case where the UART clock is 48MHz and targeted baud rate is 115,200bps:

- When FDM = 0, *UARTn_CLKDIV.clkdiv* = 48,000,000/115,200 = 417
- When FDM = 1, *UARTn_CLKDIV.clkdiv* = 48,000,000/115,200 x 2 = 416.5 x 2 = 833

10.11 Low-Power 9600 Baud Receiver Operation

Peripheral instances which support FDM have the option to configure the receiver for 9600 baud and keep it enabled in the low-power modes listed in *Table 10-1*. Receipt of a valid frame loads the receive FIFO and increment *UARTn_STATUS.rx_lvl*. If enabled, any of the conditions in *Table 10-5* causes the device to exit the current low-power mode and return to *ACTIVE*.

Instances that support FDM offer a 0.5 fractional clock division when enabled by setting *UARTn_CTRL.fdm* = 1. This allows for greater accuracy when operating at very low baud rates, as well as finer granularity for the oversampling rate. But whenever *UARTn_CLKDIV.clkdiv* < 16 (division rate less than 8.0) the oversampling feature is OSR is not used and the receive signal is sampled in every clock cycle.

Use the following formula to calculate the *UARTn_CLKDIV.clkdiv* value for low-baud rate operation when *UARTn_CTRL.fdm* = 1.

- *UARTn_CTRL.fdm* = 1: *UARTn_CLKDIV.clkdiv* = INT [(UART clock)/Baud rate x 2]

Table 10-6: Slow Baud Rate Generation Example (FDM = 1)

Clock Source	Baud	Ratio	*UARTn_CLKDIV.clkdiv* (Value)	Error	*UARTn_OSR.osr*
ERTCO	9,600 bit/s	3.413	0x00007 (3.5)	-2.5%	N/A (1x)
	7,200 bit/s	4.551	0x00009 (4)	+1.1%	N/A (1x)
	4,800 bit/s	6.827	0x00014 (7)	-2.5%	N/A (1x)
	2,400 bit/s	13.653	0x00027	+1.1%	0x0: 8x 0x1: 12x
	1,800 bit/s	18.204	0x00036	+1.1%	0x0: 8x 0x1: 12x 0x2: 16x
	1,200 bit/s	27.307	0x00054	+1.1%	0x0: 8x 0x1: 12x 0x2: 16x 0x3: 20x 0x4: 24x

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

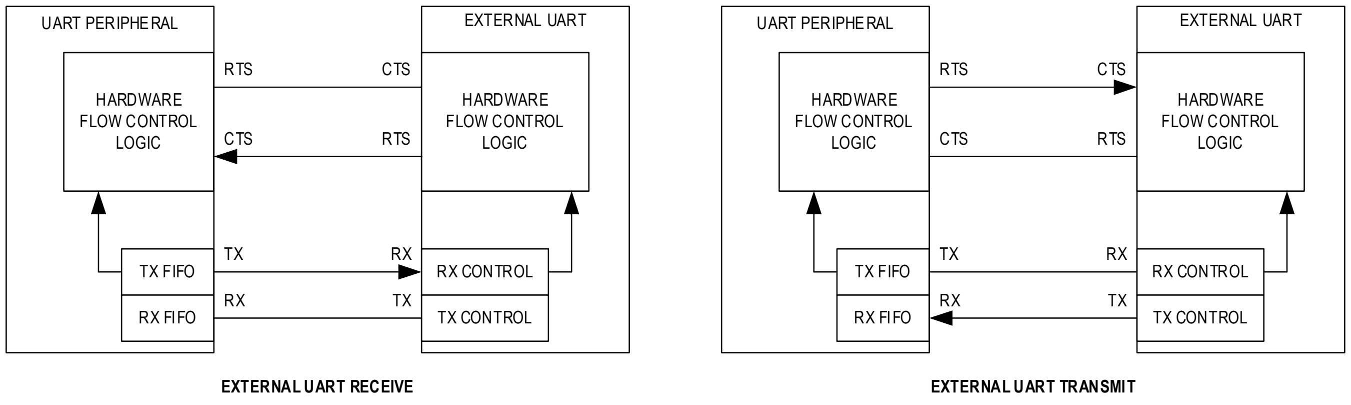
Use the following procedure for the specific use case to receive characters at 9600 baud while in low-power modes:

1. Clear *UARTn_CTRL.bclken* = 0 to disable the baud clock. Hardware immediately clears *UARTn_CTRL.bclkrdy* to 0.
2. Set *PWRSEQ_LPCN.ertco_en* = 1 ensure the 32kHz clock source remains active in *DEEPSLEEP* and *BACKUP*.
3. Ensure *UARTn_CTRL.ucagm* = 1.
4. Configure *UARTn_CTRL.bclksrc* to the 32kHz clock source.
5. Set *UARTn_CTRL.fdm* = 1.
6. Set *UARTn_CLKDIV.clkdiv* = 7.
7. Set *UARTn_CTRL.desm* = 1.
8. Choose the desired wake-up conditions from *Table 10-5* and set the corresponding wake-up enable fields to 1. Ensure the conditions are not active before setting the wake-up enable fields.
9. Set *UARTn_CTRL.bclken* = 1 to re-enable the baud clock.
10. Poll until hardware sets *UARTn_CTRL.bclkrdy* = 1.
11. Enter the desired low-power mode.

10.12 Hardware Flow Control

The optional hardware flow control (HFC) uses two pins, CTS (clear-to-send) and RTS (request-to-send), as a handshaking protocol to manage UART communications. For full duplex operation, the RTS output pin on the peripheral is connected to the CTS input pin on the external UART, and the CTS input pin on the peripheral is connected to the RTS output pin on the external UART as shown in *Figure 10-7*.

Figure 10-7: Hardware Flow Control, Physical Connection



In HFC operation, a UART transmitter that wants to transmit data waits for the other device to assert its CTS input pin. If CTS is asserted, then the UART begins transmitting data from its transmit FIFO to the slave receive FIFO. The receive FIFO keeps CTS asserted until the receive FIFO fills to the specified value. It then de-asserts CTS until the receiver has cleared the buffer below the specified value. CTS is then asserted again and more data is sent.

Hardware flow control can be fully automated by the peripheral, or by software through monitoring of the CTS input signal.

10.12.1 Automated HFC

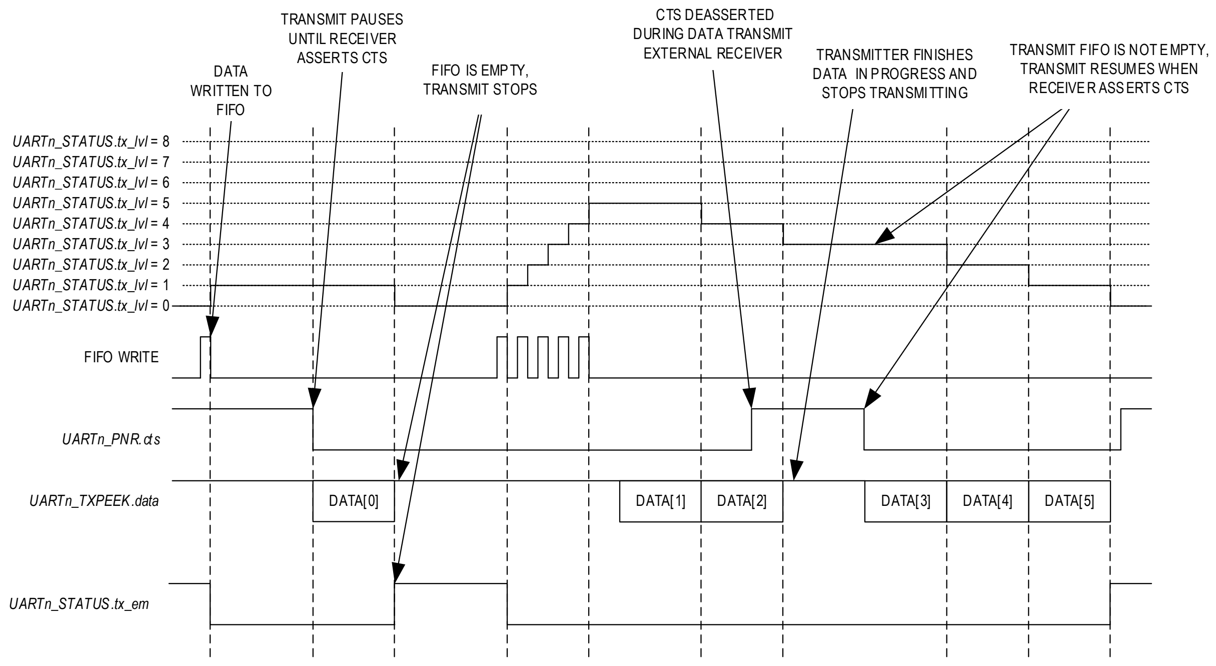
Setting *UARTn_CTRL.hfc_en* = 1 enables hardware flow control. The CTS and RTS external signals are directly managed by hardware without CPU intervention. The assertion of the RTS to CTS signal is dependent on the *UARTn_CTRL.rtsdc* field:

- *UARTn_CTRL.rtsdc* = 0: Deassert RTS while *UARTn_STATUS.rx_lvl* = C_RX_FIFO_DEPTH
- *UARTn_CTRL.rtsdc* = 1: Deassert RTS while *UARTn_STATUS.rx_lvl* ≥ *UARTn_CTRL.rx_thd_val*.

The transmitter continues to send data as long as the CTS signal is asserted and there is data in the transmit FIFO. If the receiver deasserts the transmitter's CTS, the transmitter finishes transmission of the current character and then wait until CTS is asserted again. *Figure 10-8* shows the state of the CTS pin during a transmission under hardware flow control.

HFC by itself does not generate interrupt events. FIFO management must be handled by the software using the user-configurable interrupt event enables.

Figure 10-8: Hardware Flow Control Signaling, Transmit to External Receiver



10.12.2 Software-Controlled HFC

If HFC is disabled (*UARTn_CTRL.hfc_en* = 1), the CTS pin is a general-purpose input and the RTS pin is a general-purpose output. In that case, a signal change interrupt is raised (if enabled) at each rising or falling edge of the sampling on the CTS input pin. The CTS signal change interrupt should be disabled if flow control is not used.

UARTn_CTRL.cts_dis is reset to 1 after POR. When UART baud clock starts and runs, the CTS sampling process continues if *UARTn_CTRL.cts_dis*) is not 1. The sampled value is set to 1 again when UART baud clock stops running because *UARTn_CTRL.bclken* has been cleared to 0 again.

10.13 UART Registers

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

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

All registers and fields apply to both UART and LPUART instances unless specified otherwise.

Table 10-7: UART Register Summary

Offset	Register	Description
0x0000	*UARTn_CTRL*	UART Control Register
0x0004	*UARTn_STATUS*	UART Status Register
0x0008	*UARTn_INT_EN*	UART Interrupt Enable Register
0x000C	*UARTn_INT_FL*	UART Interrupt Flag Register
0x0010	*UARTn_CLKDIV*	UART Clock Divisor Register
0x0014	*UARTn_OSR*	UART Oversampling Control Register
0x0018	*UARTn_TXPEEK*	UART Transmit FIFO
0x001C	*UARTn_PNR*	UART Pin Control Register
0x0020	*UARTn_FIFO*	UART FIFO Data Register
0x0030	*UARTn_DMA*	UART DMA Control Register
0x0034	*UARTn_WKEN*	UART Wake-up Interrupt Enable Register
0x0038	*UARTn_WKFL*	UART Wake-up Interrupt Flag Register

10.13.1 Register Details

Table 10-8: UART Control Register

UART Control		**UARTn_CTRL**		**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:23	-	DNM	0	**Reserved. Do Not Modify.**
22	desm	R/W	0	**Receive Dual-Edge Sampling Mode** 0: Sample receive on clock rising only. 1: Sample receive on both rising and falling edges. This field is undefined if the instance does not support FDM.
21	fdm	R/W	0	**Fractional Division Mode** 0: Baud rate divisor is an integer. 1: Baud rate divisor supports 0.5 division resolution. This field is undefined if the instance does not support FDM.
20	ucagm	R/W	0	**UART Clock Auto Gating Mode** Software must always set this field to 1 for proper operation. 0: No gating. 1: UART clock is paused during transmit/receive idle states.
19	bclkrdy	R	0	**Baud Clock Ready** 0: Not ready. 1: Ready.
18	dpfe_en	R/W	0	**Data/Parity Bit Frame Error Detection Enable** 0: Disable. Do not detect frame errors on receive between start bit and stop bit. 1: Enable. Detect frame errors when receive changes at the center of bit time. *Note: This field is undefined if the instance does not support FDM.*

UART Control				UARTn_CTRL [0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
17:16	bclksrc	R/W	0	**Baud Clock Source** Selects the baud clock source. See *Table 10-1* for specific values for each instance. 0: Peripheral clock. 1: External clock. 2: CLK2. 3: CLK3.
15	bclken	R/W	0	**Baud Clock Enable** 0: Disabled. 1: Enabled.
14	rtsdc	R	0	**Hardware Flow Control RTS Deassert Condition** 0: Deassert RTS when *UARTn_STATUS.rx_lvl* = C_RX_FIFO_DEPTH (FIFO full). 1: Deassert RTS while *UARTn_STATUS.rx_lvl* ≥ *UARTn_CTRL.rx_thd_val.*
13	hfc_en	R/W	0	**Hardware Flow Control Enable** 0: Disabled. 1: Enabled.
12	stopbits	R/W	0	**Number of Stop Bits** 0: 1 stop bit. 1: 1.5 stop bits (for 5 bit mode) or 2 stop bits (for 6/7/8-bit mode).
11:10	char_size	R/W	0	**Character Length** 0: 5 bits. 1: 6 bits. 2: 7 bits. 3: 8 bits.
9	rx_flush	R/W1O	0	**Receive FIFO Flush** Write 1 to flush the FIFO. This bit always reads 0. 0: Normal operation. 1: Flush FIFO.
8	tx_flush	R/W1O	0	**Transmit FIFO Flush** Write 1 to flush the FIFO. This bit always reads 0. 0: Normal operation. 1: Flush FIFO.
7	cts_dis	R/W	0	**CTS Sampling Disable** 0: Enabled. 1: Disabled.
6	par_md	R/W	0	**Parity Value Select** 0: Parity calculation is based on "1" bits. (Mark). 1: Parity calculation is based on "0" bits. (Space).
5	par_eo	R/W	0	**Parity Odd/Even Select** 0: Even parity. 1: Odd parity.
4	par_en	R/W	0	**Transmit Parity Generation Enable** 0: Parity transmission disabled. 1: Parity bit is calculated and transmitted after the last character bit.

UART Control				UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
3:0	rx_thd_val	R/W	0	**Receive FIFO Threshold** Valid settings are from 0x01 to C_RX_FIFO_DEPTH- 1. 0: Reserved. 1: 1 2: 2 3: 3 4: 4 5: 5 6: 6 7: 7 8: 8 9 - 15: Reserved.	

Table 10-9: UART Status Register

UART Status				UARTn_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:16	-	RO	0	**Reserved**	
15:12	tx_lvl	R	0	**Transmit FIFO Level** Number of characters in the transmit FIFO. 0: 0 1: 1 2: 2 3: 3 4: 4 5: 5 6: 6 7: 7 8: 8 9-15: Reserved.	
11:8	rx_lvl	R	0	**Receive FIFO Level** Number of characters in the receive FIFO. 0: 0 1: 1 2: 2 3: 3 4: 4 5: 5 6: 6 7: 7 8: 8 9-15: Reserved.	
7	tx_full	R	0	**Transmit FIFO Full** 0: Not full. 1: Full.	
6	tx_em	R	1	**Transmit FIFO Empty** 0: Not empty. 1: Empty.	
5	rx_full	R	0	**Receive FIFO Full** 0: Not full. 1: Full.	

UART Status				UARTn_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
4	rx_em	R	1	**Receive FIFO Empty** 0: Not empty. 1: Empty.	
3:2	-	RO	0	**Reserved**	
1	rx_busy	R	0	**Receive Busy** 0: UART is not receiving a character. 1: UART is receiving a character.	
0	tx_busy	R	0	**Transmit Busy** 0: UART is not transmitting data. 1: UART is transmitting data.	

Table 10-10: UART Interrupt Enable Register

UART Interrupt Enable				UARTn_INT_EN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:7	-	RO	0	**Reserved**	
6	tx_he	R/W	0	**Transmit FIFO Half-Empty Event Interrupt Enable** 0: Disabled. 1: Enabled.	
5	-	RO	0	**Reserved**	
4	rx_thd	R/W	0	**Receive FIFO Threshold Event Interrupt Enable** 0: Disabled. 1: Enabled.	
3	rx_ov	R/W	0	**Receive FIFO Overrun Event Interrupt Enable** 0: Disabled. 1: Enabled.	
2	cts_ev	R/W	0	**CTS Signal Change Event Interrupt Enable** 0: Disabled. 1: Enabled.	
1	rx_par	R/W	0	**Receive Parity Event Interrupt Enable** 0: Disabled. 1: Enabled.	
0	rx_ferr	R/W	0	**Receive Frame Error Event Interrupt Enable** 0: Disabled. 1: Enabled.	

Table 10-11: UART Interrupt Flag Register

UART Interrupt Flag				UARTn_INT_FL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:7	-	RO	0	**Reserved**	
6	tx_he	R/W1C	0	**Transmit FIFO Half-Empty Event Interrupt Flag** 0: Disabled. 1: Enabled.	
5	-	RO	0	**Reserved**	
4	rx_thd	R/W1C	0	**Receive FIFO Threshold Event Interrupt Flag** 0: Disabled. 1: Enabled.	
3	rx_ov	R/W1C	0	**Receive FIFO Overrun Event Interrupt Flag** 0: Disabled. 1: Enabled.	

UART Interrupt Flag		UARTn_INT_FL		[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
2	cts_ev	R/W1C	0	**CTS Signal Change Event Interrupt Flag** 0: Disabled. 1: Enabled.
1	rx_par	R/W1C	0	**Receive Parity Error Event Interrupt Flag** 0: Disabled. 1: Enabled.
0	rx_ferr	R/W1C	0	**Receive Frame Error Event Interrupt Flag** 0: Disabled. 1: Enabled.

Table 10-12: UART Clock Divisor Register

UART Clock Divisor		UARTn_CLKDIV		[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:0	clkdiv	R/W	0	**Baud Rate Divisor** This field sets the divisor used to generate the baud tick from the baud clock. If *.fdm* = 1, the fractional divisors are in increments of 0.5. The oversampling rate must be no greater than this divisor. When *.fdm* = 0: *.clkdiv* = (UART Clock Frequency/Baud Rate Frequency) When *.fdm* = 1: *.clkdiv* = (UART Clock Frequency/Baud Rate Frequency) x 2

Table 10-13: UART Oversampling Control Register

UART Oversampling Control		UARTn_OSR		[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2:0	osr	R/W	0	**Oversampling Rate** When *.fdm* = 1: 0: 8x. 1: 12x. 2: 16x. 3: 20x. 4: 24x. 5: 28x. 6: 32x. 7: 36x. When *.fdm* = 0: 0: 128x. 1: 64x. 2: 32x. 3: 16x. 4: 8x. 5: 4x. 6 - 7: Reserved.

Table 10-14: UART Transmit FIFO Register

UART Transmit FIFO				UARTn_TXPEEK	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:8	-	RO	0	**Reserved**	
7:0	data	R	0	**Transmit FIFO Data** Read transmit FIFO next data. Reading from this field doesn't affect the contents of transmit FIFO. Note that the parity bit is available from this field. Reading from this field returns the next character available at the output of the transmit FIFO (if one is available, otherwise 0 is returned).	

Table 10-15: UART Pin Control Register

UART Pin Control				UARTn_PNR	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:2	-	RO	0	**Reserved**	
1	rts	R/W	1	**RTS Pin Output State** 0: RTS signal is driven to 0. 1: RTS signal is driven to 1.	
0	cts	R	1	**CTS Pin State** Returns the current sampled state of the GPIO associated with the CTS signal. 0: CTS state is 0. 1: CTS state is 1.	

Table 10-16: UART FIFO Register

UART FIFO				UARTn_FIFO	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:9	-	RO	0	**Reserved**	
8	rx_par	R	0	**Receive FIFO Byte Parity** If parity feature is disabled, this bit always reads 0. If a parity error occurred during the reception of the character at the output end of the receive FIFO, this bit reads 1, otherwise it reads 0.	
7:0	data	R/W	0	**Transmit/Receive FIFO Value** Writing to this field loads the next character into the transmit FIFO (if space is available). Reading from this field returns the next character available at the output of the receive FIFO (if one is available, otherwise 0x00 is returned). For 5/6/7-bit width characters, the unused bit(s) are ignored when the transmit FIFO is loaded, and the unused high bit(s) is read as 0 on characters read from receive FIFO.	

Table 10-17: UART DMA Register

UART DMA				UARTn_DMA	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:10	-	RO	0	**Reserved**	
9	rx_en	0	0	**Receive DMA Channel Enable** 0: Disabled. 1: Enabled.	

UART DMA				UARTn_DMA	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**	
8:5	rx_thd_val	0	0	**Receive FIFO Level DMA Threshold Value** If *UARTn_STATUS.rx_lvl* < *UARTn_DMA. rx_thd_val*, then the receive FIFO DMA interface sends a signal to the system DMA to notify that receive FIFO has characters to transfer to memory.	
4	tx_en	R/W	0	**Transmit DMA Channel Enable** 0: Disabled. 1: Enabled.	
3:0	tx_thd_val	R/W	0	**Transmit FIFO Level DMA Threshold Value** If *UARTn_STATUS.tx_lvl* < *UARTn_DMA.tx_thd_val*, then the transmit FIFO DMA interface sends a signal to system DMA to notify that transmit FIFO is ready to receive data from memory.	

Table 10-18: UART Wake-Up Enable

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

Table 10-19: UART Wake-Up Flag Register

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

11. I^{2}C Master/Slave Serial Communications Peripheral

The I^{2}C peripherals can be configured as either an I^{2}C master or I^{2}C slave at standard data rates. For simplicity, I2Cn is used throughout this section to refer to any of the I^{2}C peripherals.

For detailed information on I^{2}C bus operation, refer to Analog Devices Application Note 4024 "SPI/I²C Bus Lines Control Multiple Peripherals" *https://www.maximintegrated.com/en/app-notes/index.mvp/id/4024*

11.1 I^{2}C Master/Slave Features

Each I^{2}C master/slave is compliant with the I^{2}C Bus Specification and includes the following features:

- Communicates through a serial data bus (SDA) and a serial clock line (SCL)
- Operates as either a master or slave device as a transmitter or receiver
- Supports I^{2}C 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 I^{2}C timing parameters fully controllable through software
- Provides glitch filter and Schmitt trigger hysteresis on SDA and SCL
- Provides control, status, and interrupt events for maximum flexibility.
- Provides independent 8-byte receive FIFO and 8-byte transmit FIFO.
- Provides transmit FIFO preloading
- Provides programmable interrupt threshold levels for the transmit and receive FIFO.

11.2 Instances

The two instances of the peripheral are shown in *Table 11-1*. The table lists the SDA and SCL signals for each of the I^{2}C peripherals per package.

Table 11-1: MAX32675 I^{2}C Peripheral Pins

I^{2}C Instance	Alternate Function	Alternate Function Number
I2C0	I2C0_SCL	AF1
	I2C0_SDA	AF1
I2C2	I2C2_SCL	AF1
	I2C2_SDA	AF1
Note: Refer to the device's data sheet pin description table for alternate function mapping to pin numbers.		

11.3 I^{2}C Overview

11.3.1 I^{2}C Bus Terminology

Table 11-2 contains terms and definitions used in this chapter for the I^{2}C Bus Terminology.

Table 11-2: I^{2}C Bus Terminology

Term	Definition
Transmitter	The device sending data on the bus.
Receiver	The device receiving data from the bus.
Master	The device that initiates a transfer, generates the clock signal, and terminates a transfer.
Slave	The device addressed by a master.
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message.
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one can do so, and the resulting message is not corrupted.
Synchronization	The procedure to synchronize the clock signals of two or more devices.
Clock Stretching	When a slave device holds SCL low to pause a transfer until it is ready. Clock stretching is an optional feature according to the I^{2}C 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.

11.3.2 I^{2}C Transfer Protocol Operation

The I^{2}C protocol operates over a two-wire bus: a clock circuit (SCL) and a data circuit (SDA). I^{2}C 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, followed by the I^{2}C 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.

11.3.3 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.

11.3.4 Master Operation

I^{2}C 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 I^{2}C master generates an interrupt. In addition, the I^{2}C 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 I^{2}C master stops the clock to allow time to read bytes from the receive FIFO or load bytes into the transmit FIFO.

11.3.5 Acknowledge and Not Acknowledge

An acknowledge bit (ACK) is generated by the receiver, whether I^{2}C 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 I^2C master can then either generate a STOP condition to abort the transfer or generate a repeated START condition (i.e., send a START condition without an intervening STOP condition) to start a new transfer.

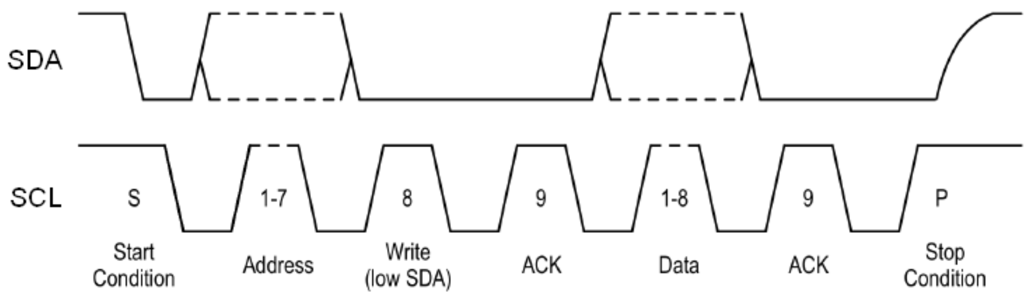
A receiver can generate a NACK after a byte transfer if any of the following conditions occur:

- No receiver is present on the bus with the transmitted address. In that case, no device responds with an acknowledge signal.
- The receiver cannot receive or transmit because it is busy and is not ready to start communication with the 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 I^2C master has requested data from a slave, it signals the slave to stop transmitting by sending a NACK following the last byte it requires.

11.3.6 Bit Transfer Process

Both SDA and SCL circuits are open-drain, bidirectional circuits. Each requires an external pullup resistor that ensures each circuit is high when idle. The I^2C specification states that during data transfer, the SDA line can change state only when SCL is low and that SDA is stable and can be read when SCL is high, as shown in *Figure 11-1*.

Figure 11-1: I^2C Write Data Transfer



An example of an I^2C 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 low to high while SCL is high.

11.4 Configuration and Usage

11.4.1 SCL and SDA Bus Drivers

SCL and SDA are open-drain signals. In this device, once the I^{2}C 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 I^{2}C slave device can hold SCL low, such as clock stretching. Push-pull operation is enabled by setting *I2Cn_CTRL.scl* to 1. SDA, on the other hand, always operates in open-drain mode.

11.4.2 SCL Clock Configurations

The SCL frequency depends on the values of the I^{2}C 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.

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

The master generates the I^{2}C clock on the SCL line. When operating as a master, the software must configure the *I2Cn_CLKHI* and *I2Cn_CLKLO* registers for the desired I^{2}C operating frequency.

The SCL high time is configured in the I^{2}C Clock High Time register field *I2Cn_CLKHI.hi* using *Equation 11-2*. The SCL low time is configured in the I^{2}C Clock Low Time register field *I2Cn_CLKLO.lo* using *Equation 11-3*. Each of these fields is 8-bits. The I^{2}C frequency value is shown in *Equation 11-1*.

Equation 11-1: I^{2}C Clock Frequency

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

Equation 11-2: I^{2}C Clock High Time Calculation

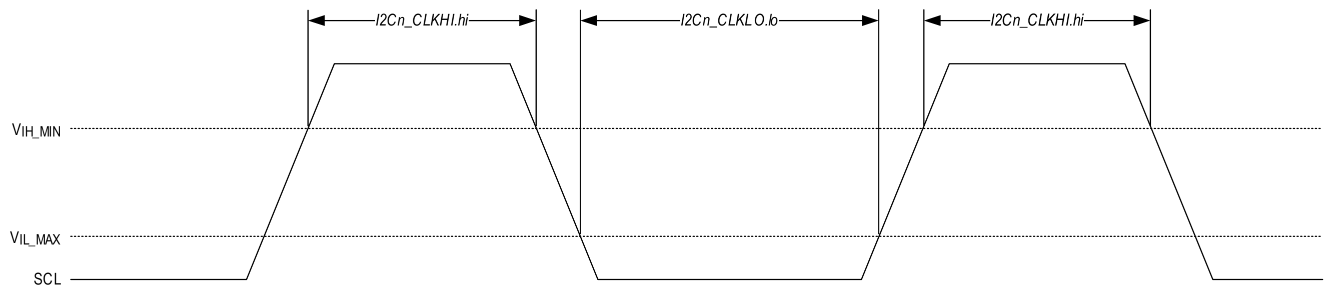
$$t_{SCL\_HI} = t_{I2C\_CLK} \times (I2Cn\_CLKHI.hi + 1)$$

Equation 11-3: I^{2}C Clock Low Time Calculation

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

Figure 11-2 shows the association between the SCL clock low and high times for Standard Mode, Fast Mode, and Fast Mode Plus I^{2}C frequencies.

Figure 11-2: I^{2}C 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 counting the number of SCL high cycles. Similarly, if an external master pulls SCL low before the controller has finished counting SCL high cycles, the controller starts counting SCL low cycles and releases SCL once the time period, *I2Cn_CLKLO.lo*, has expired.

Because the controller does not start counting the high or 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.

11.4.4 SCL Clock Generation for Hs-Mode

The values programmed into the *I2Cn_HSCLK.lo* register and *I2Cn_HSCLK.hi* register must be determined to operate the I^{2}C interface in Hs-Mode at its maximum speed (~3.4MHz). Since the Hs-Mode operation is entered by first using one of the lower speed modes for a preamble, a relevant lower speed mode must also be configured. See *SCL Clock Generation for Standard, Fast and Fast-Plus Modes* for information regarding the configuration of lower speed modes.

11.4.4.1 Hs-Mode Timing

With I^{2}C bus capacitances less than 100pF, the following specifications are extracted from the I^{2}C-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 I^{2}C bus clock.

t_{HIGH_MIN} = 60ns, the minimum high time for the I^{2}C bus clock.

t_{rCL_MAX} = 40ns, the maximum rise time of the I^{2}C bus clock.

t_{fCL_MAX} = 40ns, the maximum fall time of the I^{2}C bus clock.

11.4.4.2 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 11-4: I^{2}C Target SCL Frequency

$$\textit{Desired Target Maximum I}^2\textit{C Frequency: } f_{SCL} = \frac{1}{t_{SCL}}.$$

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

Equation 11-5: Determining the I2Cn_HSCLK.lo Register Value

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

Equation 11-6: Determining the I2Cn_HSCLK.hi Register Value

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

Equation 11-7: The Calculated Frequency of the I^{2}C Bus Clock Using the Results of Equation 11-5 *and* Equation 11-6

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

Table 11-3 shows the I^{2}C bus clock calculated frequencies given different f_{SYS_CLK} frequencies.

Table 11-3: Calculated I^{2}C Bus Clock Frequencies

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

f_{SYS_CLK} (MHz)	I2Cn_HSCLK.hi	I2Cn_HSCLK.lo	Calculated Frequency (MHz)
25	1	2	2.5

11.4.5 Master Mode Addressing

After a START condition, the I^{2}C slave address byte is transmitted by the hardware. The I^{2}C slave address is composed of a slave address followed by a read/write bit.

Table 11-4: I^{2}C 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. First, to address a 7-bit address slave, clear the *I2Cn_MSTCTRL.ex_addr_en* field to 0, then write the address to the transmit FIFO formatted as follows where A*n* is address A6:A0.

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_MSTCTRL.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 I^{2}C then starts receiving data from the slave device.

11.4.6 Master Mode Operation

The peripheral operates in master mode when master mode enable (*I2Cn_CTRL.mst_mode*) is set to 1. To initiate a transfer, the master generates a START condition by setting *I2Cn_MSTCTRL.start* = 1. If the bus is busy, it does not generate a START condition until the bus is available.

A 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_MSTCTRL.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_MSTCTRL.restart* bit is automatically cleared to 0 as soon as the master begins a RESTART condition.

I2Cn_MSTCTRL.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_MSTCTRL.stop* = 1.

If both START and RESTART conditions are enabled simultaneously, a START condition is generated first. Then, at the end of the first transaction, a RESTART condition is generated.

If both RESTART and STOP conditions are enabled simultaneously, a STOP condition is not generated. Instead, a RESTART condition is generated. After the RESTART condition is generated, both bits are cleared.

If START, RESTART, and STOP are all enabled simultaneously, a START condition is first generated. At the end of the first transaction, a RESTART condition is generated. The *I2Cn_MSTCTRL.stop* bit is cleared and ignored.

A slave cannot generate START, RESTART, or STOP conditions. Therefore, when master mode is disabled, the *I2Cn_MSTCTRL.start*, *I2Cn_MSTCTRL.restart*, and *I2Cn_MSTCTRL.stop* bits are all cleared to 0.

For master mode operation, the following registers should only be configured when either:

1. The I^{2}C peripheral is disabled, or
2. the I^{2}C 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_MSTCTRL.start* = 0) satisfies this requirement:

- *I2Cn_CTRL.mst_mode*
- *I2Cn_CTRL.irxm_en*
- *I2Cn_CTRL.one_mst_mode*
- *I2Cn_CTRL.hs_en*
- *I2Cn_RXCTRL1.cnt*
- *I2Cn_MSTCTRL.ex_addr_en*
- *I2Cn_CLKLO.lo*
- *I2Cn_CLKHI.hi*
- *I2Cn_HSCLK.lo*
- *I2Cn_HSCLK.hi*

In contrast to the above set of register fields, the register fields below can be safely (re)programmed at any time:

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

11.4.6.1 I^2C 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 I^2C receive count field (*I2Cn_RXCTRL1.cnt*).
2. Write the I^2C slave address byte to the *I2Cn_FIFO* register with the R/W bit set to 1.
3. Send a START condition by setting *I2Cn_MSTCTRL.start* = 1.
4. The slave address is transmitted by the controller from the *I2Cn_FIFO* register.
5. The I^2C controller receives an ACK from the slave, and the controller sets the address ACK interrupt flag (*I2Cn_INTFL0.addr_ack* = 1).
6. The I^2C 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_RXCTRL1.cnt* data bytes have been received, the I^2C controller sends a NACK to the slave and sets the Transfer Done Interrupt Status Flag (*I2Cn_INTFL0.done* = 1).
8. If *I2Cn_MSTCTRL.restart* or *I2Cn_MSTCTRL.stop* is set, then the I^2C controller sends a repeated START or STOP, respectively.

11.4.6.2 I^2C Master Mode Transmitter Operation

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

1. Write the I^2C 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 before starting the transaction.)
3. Send a START condition by setting *I2Cn_MSTCTRL.start* = 1.
4. The controller transmits the slave address byte written to the *I2Cn_FIFO* register.
5. The I^2C controller receives an ACK from the slave, and the controller sets the address ACK interrupt flag (*I2Cn_INTFL0.addr_ack* = 1).
6. The *I2Cn_FIFO* register data bytes are transmitted on the SDA line.
   a. The I^2C 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_MSTCTRL.restart* or *I2Cn_MSTCTRL.stop*.
8. Once the controller sends all the remaining bytes and empties the transmit FIFO, it sets *I2Cn_INTFL0.done* and proceeds to send out either a RESTART condition if *I2Cn_MSTCTRL.restart* was set or a STOP condition if *I2Cn_MSTCTRL.stop* was set.

11.4.6.3 I^2C Multi-Master Operation

The I^2C protocol supports multiple masters on the same bus. When the bus is free, two (or more) masters might try to initiate communication simultaneously. This is a valid bus condition. If this occurs and the two masters want to transmit different data or address different slaves, only one master can remain in master mode and complete its transaction. The other master must back off the transmission and wait until the bus is idle. This process by which the winning master is determined is called bus arbitration.

The master compares the data being transmitted on SDA to the value observed on SDA to determine which master wins the arbitration for each address or data bit. 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 the arbitration, it stops transmitting on SDA, leaving the following data bits sent on SDA intact.

If the I^{2}C master peripheral detects it has lost the arbitration, it stops generating SCL; sets *I2Cn_INTFL0.arb_err*; sets *I2Cn_INTFL0.tx_lockout*, flushing any remaining data in the transmit FIFO; and clears *I2Cn_MSTCTRL.start*, *I2Cn_MSTCTRL.restart*, and *I2Cn_MSTCTRL.stop* to 0. As long as the peripheral is not addressed by the winning master, the I^{2}C peripheral stays in master mode (*I2Cn_CTRL.mst_mode = 1)*. If, *at any time,* another master addresses this peripheral using the address programmed in I2Cn_SLAVE.*addr*, then the I^{2}C peripheral clears *I2Cn_CTRL.mst_mode* 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_INTFL0.tx_lockout. Therefore, after an arbitration loss, the software needs to clear I2Cn_INTFL0.tx_lockout and reload the transmit FIFO.

Also, in a 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_MSTCTRL.start*). If the bus is free when *I2Cn_MSTCTRL.start* is set to 1, the transaction begins immediately. If, instead, the bus is busy, then the peripheral:

1. Waits for the other 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 11-3), and then*
3. Sends a START condition and begin transmitting the slave address byte(s) in the transmit FIFO, followed by the rest of the transfer.

The I^{2}C master peripheral is compliant with all bus arbitration and clock synchronization requirements of the I^{2}C specification; this operation is automatic, and no additional programming is required.

11.4.7 Slave Mode Operation

When in slave mode, the I^{2}Cn peripheral operates as a slave device on the I^{2}C bus and responds to an external master's requests to transmit or receive data. To configure the I^{2}Cn peripheral as a slave, write the *I2Cn_CTRL.mst_mode* bit to zero. The master drives the I2Cn clock on the bus, so the SCL device pin is driven by the external master, and *I2Cn_STATUS.mst_busy* remains a zero. The desired slave address must be set by writing to the *I2Cn_SLAVE.addr* register.

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

- *I2Cn_CTRL.mst_mode* = 0 for slave operation.
- I^2C slave address
  - *I2Cn_SLAVE.addr* must be set to the desired address for the device on the bus
  - *I2Cn_SLAVE.ext_addr_en* should be set to 1 for 10-bit addressing or 0 for 7-bit addressing
- *I2Cn_CTRL.gc_addr_en*
- *I2Cn_CTRL.irxm_en*
  - The recommended value for this field is 0. *Note that a setting of 1 is incompatible with slave mode operation with clock stretching disabled (I2Cn_CTRL.clkstr_dis = 1).*
- *I2Cn_CTRL.clkstr_dis*
- *I2Cn_CTRL.hs_en*
- *I2Cn_RXCTRL0.dnr*
  - SMBus/PMBus applications should set this to 0, while other applications should set this to 1.
- *I2Cn_TXCTRL0.nack_flush_dis*
- *I2Cn_TXCTRL0.rd_addr_flush_dis*
- *I2Cn_TXCTRL0.wr_addr_flush_dis*
- *I2Cn_TXCTRL0.gc_addr_flush_dis*
- *I2Cn_TXCTRL0.preload_mode*
  - The recommended value is 0 for applications that can tolerate slave clock stretching (*I2Cn_CTRL.clkstr_dis* = 0).
  - The recommended value is 1 for applications that do not allow slave clock stretching (*I2Cn_CTRL.clkstr_dis* = 1).
- *I2Cn_CLKHI.hi*
  - Applies to slave mode when clock stretching is enabled (*I2Cn_CTRL.clkstr_dis* = 0)
    - This is used to satisfy $t_{SU;DAT}$ after clock stretching; program it so that the value defined by *Equation 11-2* is >= $t_{SU;DAT(min)}$.
- *I2Cn_HSCLK.hi*
  - Applies to slave mode in Hs Mode when clock stretching is enabled (*I2Cn_CTRL.clkstr_dis* = 0)
    - This is used to satisfy $t_{SU;DAT}$ after clock stretching during Hs-Mode operation; program it so that the value defined by *Equation 11-6* is >= $t_{SU;DAT(min)}$.

In contrast to the above register fields, the following register fields can be safely (re)programmed at any time:

- All interrupt flags and interrupt enables.
- *I2Cn_TXCTRL0.thd_val* and *I2Cn_RXCTRL0.thd_lvl*
  - Transmit and receive FIFO threshold levels.
- *I2Cn_TXCTRL1.preload_rdy*
  - 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.

11.4.7.1 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.

11.4.7.1.1 Just-in-Time Slave Transmitter

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

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

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers, including:
    a. Set the *I2Cn_SLAVE*.*addr* field with the desired I^2C slave address.
    b. Set the *I2Cn_SLAVE*.*ext_addr_en* for either 7-bit or 10-bit addressing.
    c. Just-in-time mode specific settings:
        i) *I2Cn_CTRL*. *clkstr_dis* = 0
        ii) *I2Cn_TXCTRL0*[5:2] = 0x8
        iii) *I2Cn_TXCTRL0*.*preload_mode* = 0.
    d. Program *I2Cn_CLKHI*.*hi* and *I2Cn_HSCLK*.*hi* with appropriate values satisfying $t_{SU;DAT}$ (and HS $t_{SU;DAT}$).
2. The software sets *I2Cn_CTRL*.*en* = 1.
    a. The controller is now listening for its address. For either a transmit (R/W = 1) or receive (R/W = 0) operation, the peripheral responds to its address with an ACK.
    b. When the address match occurs, the hardware sets *I2Cn_INTFL0*.*addr_match* and *I2Cn_INTFL0*.*tx_lockout*.

3. The software waits for *I2Cn_INTFL0.addr_match* to read 1, either through polling the interrupt flag or setting *I2Cn_INTEN0.addr_match* to interrupt the CPU.
4. After reading *I2Cn_INTFL0.addr_match* =1, the software reads *I2Cn_CTRL.read* to determine whether the transaction is a transmit (read = 1) or receive (read = 0) operation. In this case, assume read = 1, indicating transmit.
    a. The hardware holds SCL low until the software clears *I2Cn_INTFL0.tx_lockout* and loads data into the FIFO.
5. The software clears *I2Cn_INTFL0.addr_match* and *I2Cn_INTFL0.tx_lockout*. Now that *I2Cn_INTFL0.tx_lockout* is 0, the software can begin loading the transmit data into *I2Cn_FIFO*.
6. As soon as there is data in the FIFO, the hardware releases SCL (after counting out *I2Cn_CLKHI.hi*) and sends out the data on the bus.
7. While the master keeps requesting data and sending ACKs, *I2Cn_INTFL0.done* remains 0, and the software should continue to monitor the transmit FIFO and refill it as needed.
    a. The FIFO level can be monitored synchronously through the transmit FIFO status/interrupt flags or asynchronously by setting *I2Cn_TXCTRL0.thd_val* and setting the *I2Cn_INTEN0.tx_thd* interrupt.
    b. If the transmit FIFO ever empties during the transaction, the hardware starts clock stretching and waits for it to be refilled.
8. The master ends the transaction by sending a NACK. Once this happens, the *I2Cn_INTFL0.done* interrupt flag is set, and the software can stop monitoring the transmit FIFO.
    a. If the software needs to know how many data bytes were transmitted to the master, it should check the transmit FIFO level as soon as *I2Cn_INTFL0.done* = 1 and use it to determine how many data bytes were successfully sent.
        1) Note: Any data remaining in the transmit FIFO is discarded before the next transmit operation; it is NOT necessary for the software to manually flush the transmit FIFO.
9. The transaction is complete. The software should clear the *I2Cn_INTFL0.done* interrupt flag and clear the *I2Cn_INTFL0.tx_thd* interrupt flag. Return to step 3, waiting on an address match.

11.4.7.1.2 Preload Mode Slave Transmit

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

To use slave transmit preload mode:

1. With *I2Cn_CTRL.en* = 0, initialize all relevant registers, including:
    a. Set the *I2Cn_SLAVE.addr* field with the desired I^{2}C slave address.
    b. Set the *I2Cn_SLAVE.ext_addr_en* for either 7-bit or 10-bit addressing.
    c. Preload mode specific settings:
        i) *I2Cn_CTRL.clkstr_dis* = 1
        ii) *I2Cn_TXCTRL0*[5:2] = 0xF
        iii) *I2Cn_TXCTRL0.preload_mode* = 1.
2. The software sets *I2Cn_CTRL.en* = 1.
    a. Even though the controller is enabled, it does not ACK an address match with R/W equal to 1 until the software sets the *I2Cn_TXCTRL1.preload_rdy* field to 1.

3. The software prepares for the transmit operation by loading data into the transmit FIFO, enabling DMA, setting *I2Cn_TXCTRL0.thd_val,* and setting *I2Cn_INTEN0.tx_thd* interrupt, etc.
    a. If clock stretching is disabled, an empty transmit FIFO during the transmit operation causes a transmit underrun error. Therefore, the software should take any necessary steps to avoid an underrun *before* setting *I2Cn_TXCTRL1.preload_rdy* = 1.
    b. If clock stretching is enabled, then an empty transmit FIFO does not cause a transmit underrun error. However, it is recommended to follow the same preparation steps to minimize the amount of time spent clock stretching, which lets the transaction complete as quickly as possible.
4. Once the software has prepared for the transmit operation; it sets *I2Cn_TXCTRL1.preload_rdy* = 1.
    a. The controller is now fully enabled and responds with an ACK to an address match.
    b. The hardware sets *I2Cn_INTFL0.addr_match* when an address match occurs. *I2Cn_INTFL0.tx_lockout* is NOT set to 1 and remains 0.
5. The software waits for *I2Cn_INTFL0.addr_match* = 1, either through polling the interrupt flag or setting *I2Cn_INTEN0.addr_match* to interrupt the CPU.
6. After seeing *I2Cn_INTFL0.addr_match* =1, the software reads *I2Cn_CTRL.read* to determine if the transaction is a transmit (read = 1) or receive (read = 0) operation. In this case, assume *I2Cn_CTRL.read*, indicating a transmit.
    a. The hardware begins sending out the data that was preloaded into the transmit FIFO.
    b. Once the first data byte is sent, the hardware automatically clears *I2Cn_TXCTRL1.preload_rdy* to 0.
7. While the master keeps requesting data and sending ACKs, *I2Cn_INTFL0.done* remains 0, and the software should continue to monitor the transmit FIFO and refill it as needed.
    a. The FIFO level can be monitored synchronously through the transmit FIFO status/interrupt flags or asynchronously by setting *I2Cn_TXCTRL0.thd_val* and setting *I2Cn_INTEN0.tx_thd* interrupt.
    b. If clock stretching is disabled and the transmit FIFO empties during the transaction, the hardware sets *I2Cn_INTFL1.tx_un* = 1 and sends 0xFF for all following data bytes requested by the master.
8. The master ends the transaction by sending a NACK, causing the hardware to set the *I2Cn_INTFL0.done* interrupt flag.
    a. If the transmit FIFO empties simultaneously that the master indicates the transaction is complete by sending a NACK, this is not considered an underrun event *I2Cn_INTFL1.tx_un* flag remains 0.
    b. If the software needs to know how many data bytes were transmitted to the master, check the transmit FIFO level when the *I2Cn_INTFL0.done* flag is set to 1.
9. The transaction is complete, the software should "clean up," which should include clearing *I2Cn_INTFL0.done*. Return to step 3 and prepare for the next transaction.
    a. Any data remaining in the transmit FIFO is not discarded; it is reused for the next transmit operation.
        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.en* and the writing 1 to *I2Cn_CTRL.en*.

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

If the transmit FIFO is not ready (*I2Cn_TXCTRL1.preload_rdy* = 0) and the I^{2}C 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 I^{2}C read transaction is after the address byte.

11.4.7.2 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.en* = 0, initialize all relevant registers, including:
   a. Set the *I2Cn_SLAVE.addr* field with the desired I^{2}C slave address.
   b. Set the *I2Cn_SLAVE.ext_addr_en* for either 7-bit or 10-bit addressing.
2. Set *I2Cn_CTRL.en* = 1.
   a. If an address match with the R/W bit equal to zero occurs, and the receive FIFO is empty, the peripheral responds with an ACK, and the *I2Cn_INTFL0.addr_match* flag is set.
   b. If the receive FIFO is not empty, then depending on the value of *I2Cn_RXCTRL0.dnr*, the peripheral NACKs either the address byte (*I2Cn_RXCTRL0.dnr* = 1) or the first data byte (*I2Cn_RXCTRL0.dnr* = 0).
3. Wait for *I2Cn_INTFL0.addr_match* = 1, either by polling or by enabling the *wr_addr_match* interrupt. Once a successful address match occurs, the hardware sets *I2Cn_INTFL0.addr_match* = 1.
4. Read *I2Cn_CTRL.read* to determine if the transaction is a transmit (*I2Cn_CTRL.read* = 1) or a receive (*I2Cn_CTRL.read* = 0) operation. In this case, assume *I2Cn_CTRL.read* = 0, indicating receive. The device begins receiving data into the receive FIFO.
5. Clear *I2Cn_INTFL0.addr_match*, and while the master keeps sending data, *I2Cn_INTFL0.done* remains 0, and the software should continue to monitor the receive FIFO and empty it as needed.
   a. The FIFO level can be monitored synchronously through the receive FIFO status/interrupt flags or asynchronously by setting *I2Cn_RXCTRL0.thd_lvl* and enabling the *I2Cn_INTFL0.rx_thd* interrupt.
   b. If the receive FIFO ever fills up during the transaction, then the hardware sets *I2Cn_INTFL1.rx_ov* and then either:
      i. If *I2Cn_CTRL.clkstr_dis* = 0, start clock stretching and wait until the software reads from the receive FIFO, *or*
      ii. If *I2Cn_CTRL.clkstr_dis* = 1, respond to the 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_INTFL0.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^{2}C bus to the Idle state, and the hardware sets the *I2Cn_INTFL0.start_err* field or *I2Cn_INTFL0.stop_err* field to 1 based on the specific condition.

If the software has not emptied the data in the receive FIFO from the previous transaction by the time a 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 sent at the end of the first data byte. Setting *I2Cn_RXCTRL0.dnr* to 1 chooses the former while setting *I2Cn_RXCTRL0.dnr* to 0 chooses the latter.

11.4.8 Interrupt Sources

The I^{2}C controller has a very flexible interrupt generator that generates an interrupt signal to the interrupt controller on any of several events. On recognizing the I^{2}C interrupt, the software determines the cause of the interrupt by reading the I^{2}C interrupt flags registers *I2Cn_INTFL0* and *I2Cn_INTFL1*. 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, or full to a configurable threshold level (master/slave).
- Transmit FIFO locked out because it is being flushed (master/slave)
- Out of sequence START and STOP conditions (master/slave).
- Sent a NACK to an external master because the transmit or receive FIFO was not ready (slave).
- Address ACK or NACK received (master).
- Incoming address match (slave)
- Transmit underflow or receive overflow (slave).

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

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

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

11.4.9 Transmit FIFO and Receive FIFO

There are separate transmit and receive FIFOs. Both are accessed using the FIFO data register *I2Cn_FIFO*. Writes to this register enqueue data into the transmit FIFO. Writes to a full transmit FIFO has no effect. Reads from *I2Cn_FIFO* dequeue data from the receive FIFO. Writes to a full transmit FIFO has no effect and reads from an empty receive FIFO return 0xFF.

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

To offload work from the CPU, the DMA can read and write to each FIFO. See *DMA Control* for more information on configuring the DMA.

During a receive transaction (which during 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_INTFL1.rx_ov* the *I2Cn_INTFL1.rx_ov* bit, and one of two things occur depending on the value of *I2Cn_CTRL.clkstr_dis*:

- If clock stretching is enabled (*I2Cn_CTRL.clkstr_dis* = 0), then the hardware stretches the clock until the software makes space available in the receive FIFO by reading *I2Cn_FIFO*. Once space is available, the hardware moves the data byte from the shift register into the receive FIFO, the SCL device pin is released, and the master is free to continue the transaction.
- If clock stretching is disabled (*I2Cn_CTRL.clkstr_dis* = 1), 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 the threshold.
- Receive FIFO level greater than or equal to the threshold.
- Transmit FIFO underflow.
- Receive FIFO overflow.
- Transmit FIFO locked for writing.

Both the receive FIFO and transmit FIFO are flushed when the I2Cn port is disabled by clearing *I2Cn_CTRL.en* to 0. While the peripheral is disabled, writes to the transmit FIFO have no effect and reads from the receive FIFO return 0xFF.

The transmit FIFO and receive FIFO can be flushed by setting the transmit FIFO flush bit (*I2Cn_TXCTRL0.flush*=1) or the receive FIFO flush bit (*I2Cn_RXCTRL0.flush*=1), respectively. In addition, under certain conditions, the transmit FIFO is automatically locked by the hardware and flushed so stale data is not unintentionally transmitted. The transmit FIFO is automatically flushed and writes locked out from the software under the following conditions:

- General Call Address Match: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.gc_addr_flush_dis*.
- Slave Address Match Write: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.wr_addr_flush_dis*.
- Slave Address Match Read: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.rd_addr_flush_dis*.
- During operation as a slave transmitter, a NACK is received. Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.nack_flush_dis*.
- Any of the following interrupts:
  - Arbitration error, timeout error, 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_INTFL0.tx_lockout* = 1) and writes to the transmit FIFO are ignored until the software acknowledges the external event by clearing *I2Cn_INTFL0.tx_lockout*.

11.4.10 Transmit FIFO Preloading

There can be situations during slave mode operation where the software wants to preload the transmit FIFO before a transmission, such as when clock stretching is disabled. In this scenario, rather than responding to an external 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_TXCTRL1.preload_rdy*) bit. When *I2Cn_TXCTRL1.preload_rdy* is set to 0, the hardware automatically NACKs all read transactions from the master. Setting *I2Cn_TXCTRL1.preload_rdy* 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 before setting the *I2Cn_TXCTRL1.preload_rdy* field to 1.

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

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

Note: To prevent the preloaded data from being cleared when the master tries to read it, the software must at least set I2Cn_TXCTRL0.rd_addr_flush_dis to 1, disabling auto flush on READ address match. The software determines if the other auto flush disable bits should be set. For example, if a master uses I^{2}C WRITE transactions to determine what data the slave should send in the following READ transactions, the software can clear I2Cn_TXCTRL0.wr_addr_flush_dis to 0. When a WRITE occurs, the transmit FIFO is flushed, giving the software time to load the new data. For the READ transaction, the external master can poll the slave address until the new data has been loaded and I2Cn_TXCTRL1.preload_rdy is set, at which point the peripheral responds with an ACK.

11.4.11 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.irxm_en* = 1. If IRXM is enabled, it must occur before any I^{2}C transfer is initiated.

When IRXM is enabled, after every data byte received, the I2Cn peripheral automatically holds SCL low before the ACK bit. Additionally, after the 8th SCL falling edge, the I2Cn peripheral sets the IRXM interrupt status flag (*I2Cn_INTFL0.irxm* = 1). Software must read the data and generate a response (ACK or NACK) by setting the IRXM Acknowledge (*I2Cn_CTRL.irxm_ack*) bit accordingly. Send an ACK by clearing the *I2Cn_CTRL.irxm_ack* bit to 0. Send a NACK by setting the *I2Cn_CTRL.irxm_ack* bit to 1.

After setting the *I2Cn_CTRL.irxm_ack* bit, clear the IRXM interrupt flag. Write 1 to *I2Cn_INTFL0.irxm* to clear the interrupt flag. When the IRXM interrupt flag is cleared, the I2Cn peripheral hardware releases the SCL line and sends the *I2Cn_CTRL.irxm_ack* on the SDA line.

While the I2Cn peripheral is waiting for the software to clear the *I2Cn_INTFL0.irxm* flag, the software can disable IRXM and, if operating as a 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.clkstr_dis = 0).

11.4.12 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^{2}C 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.irxm_en* = 1), the peripheral can also clock stretch *before* the ACK bit, allowing the software to decide if to send an ACK or NACK.

For a transmit operation (as either master or slave), when the transmit FIFO is empty, SCL is automatically held low after the ACK bit and before the next data byte begins. The software must write data to *I2Cn_FIFO.data* to stop clock stretching and continue the transaction. However, if operating in master mode instead of sending more data, the software can also set either *I2Cn_MSTCTRL.stop* or *I2Cn_MSTCTRL.restart* to send a STOP or RESTART condition, respectively.

For a receive operation (as either master or slave), when both the receive FIFO and the receive shift register are full, SCL is automatically held low until at least one data byte is read from the receive FIFO. The software must read data from *I2Cn_FIFO.data* to stop clock stretching and continue the transaction. If operating in master mode and this is the final byte of the transaction, as determined by *I2Cn_RXCTRL1.cnt*, the software must also set either *I2Cn_MSTCTRL.stop* or *I2Cn_MSTCTRL.restart* to send a STOP or RESTART condition, respectively. This must be done in addition to reading from the receive FIFO since the peripheral cannot start sending the STOP or RESTART until the last data byte has been moved from the receive shift register into the receive FIFO. This automatically occurs once there is space in the receive FIFO.

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

Note: The clock synchronization required to support other I^{2}C master or slave devices stretching the clock is built into the peripheral and requires no intervention from the software to operate correctly.

11.4.13 Bus Timeout

The timeout field, *I2Cn_TIMEOUT.scl_to_val,* is used to detect bus errors. *Equation 11-8* and *Equation 11-9* show equations for calculating the maximum and minimum timeout values based on the value loaded into the *I2Cn_TIMEOUT.scl_to_val* field.

Equation 11-8: I^{2}C 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 11-9*.

Equation 11-9: I²C Timeout Minimum

$$t_{TIMEOUT} \leq \left(\frac{1}{f_{I2C\_CLK}}\right) \times ((I2Cn\_TIMEOUT.scl\_to\_val \times 32) + 2)$$

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_INTFL0.to_err* = 1).

For applications where the device can hold the SCL line low longer than the maximum timeout supported, the timeout can be disabled by setting the timeout field to 0 (*I2Cn_TIMEOUT.scl_to_val* = 0).

11.4.14 DMA Control

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

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

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

Equation 11-10: DMA Burst Size Calculation for I²C Transmit

$$DMA\ Burst\ Size \leq TX\ FIFO\ Depth - I2Cn\_TXCTRL0.thd\_val = 8 - I2Cn\_TXCTRL0.thd\_val$$

$$where\ 0 \leq I2Cn\_TXCTRL0.thd\_val \leq 7$$

Software trying to avoid clock stretching or a transmit underflow should use a smaller burst size and higher *I2Cn_TXCTRL0.thd_val* setting. This fills up the FIFO more frequently but increases internal bus traffic.

If the receive FIFO count (*I2Cn_RXCTRL1.lvl*) is greater than or equal to the receive FIFO threshold level *I2Cn_RXCTRL0.thd_lvl*, the DMA transfers data out of the receive FIFO according to the DMA configuration. The DMA burst size should be set as follows to ensure the DMA does not underflow the receive FIFO:

Equation 11-11: DMA Burst Size Calculation for I²C Receive

$$DMA\ Burst\ Size \leq I2Cn\_RXCTRL0.thd\_lvl$$

$$where\ 1 \leq I2Cn\_RXCTRL0.thd\_lvl \leq 8$$

Software trying to avoid receive overflow or clock stretching should use a smaller burst size and lower *I2Cn_RXCTRL0.thd_lvl*. This results in reading from the Receive FIFO more frequently but increases internal bus traffic.

Note: For receive operations, the length of the DMA transaction (in bytes) must be an integer multiple of I2Cn_RXCTRL0.thd_lvl. Otherwise, the receive transaction ends with some data still in the receive FIFO, but not enough to trigger an interrupt to the DMA, leaving the DMA transaction incomplete. One easy way to ensure this for all transaction lengths is to set burst size to 1 (I2Cn_RXCTRL0.thd_lvl = 1).

Enable the transmit DMA channel (*I2Cn_DMA*.*tx_en*) or the receive DMA channel (*I2Cn_DMA*.*rx_en*) to enable DMA transfers.

11.5 I²C Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers, as shown in *Table 11-5*. 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, a soft reset, a POR, and the peripheral-specific resets.

Table 11-5: Register Summary

Offset	Register	Description
[0x0000]	*I2Cn_CTRL*	Control Register
[0x0004]	*I2Cn_STATUS*	Status Register
[0x0008]	*I2Cn_INTFL0*	Interrupt Flags 0 Register
[0x000C]	*I2Cn_INTEN0*	Interrupt Enable 0 Register
[0x0010]	*I2Cn_INTFL1*	Interrupt Flags 1 Register
[0x0014]	*I2Cn_INTEN1*	Interrupt Enable 1 Register
[0x0018]	*I2Cn_FIFOLEN*	FIFO Length Register
[0x001C]	*I2Cn_RXCTRL0*	Receive Control 0 Register
[0x0020]	*I2Cn_RXCTRL1*	Receive Control 1 Register
[0x0024]	*I2Cn_TXCTRL0*	Transmit Control 0 Register
[0x0028]	*I2Cn_TXCTRL1*	Transmit Control 1 Register
[0x002C]	*I2Cn_FIFO*	Transmit and Receive FIFO Register
[0x0030]	*I2Cn_MSTCTRL*	Master Control Register
[0x0034]	*I2Cn_CLKLO*	Clock Low Time Register
[0x0038]	*I2Cn_CLKHI*	Clock High Time Register
(0x003C)	*I2Cn_HSCLK*	Hs-Mode Clock Control Register
[0x0040]	*I2Cn_TIMEOUT*	Timeout Register
[0x0048]	*I2Cn_DMA*	DMA Enable Register
[0x004C]	*I2Cn_SLAVE*	Slave Address Register

11.5.1 Register Details

Table 11-6: I²C Control Register

I²C Control			**I2Cn_CTRL**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	hs_en	R/W	0	**Hs-Mode Enable** I²C high speed mode operation 0: Disabled. 1: Enabled.
14	-	RO	0	**Reserved**

I²C Control				I2Cn_CTRL [0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
13	one_mst_mode	R/W	0	**Single Master Only** When set to 1, the device MUST ONLY be used in a single master application with slave devices that are NOT going to hold SCL low (i.e., the slave devices never clock stretch)
12	clkstr_dis	R/W	0	**Slave Mode Clock Stretching** 0: Enabled. 1: Disabled.
11	read	R	0	**Slave Read/Write Bit Status** Returns the logic level of the R/W bit on a received address match (*I2Cn_INTFL0.addr_match* = 1) or general call match (*I2Cn_INTFL0.gc_addr_match* = 1). This bit is valid for three system clock cycles after the address match status flag is set.
10	bb_mode	R/W	0	**Software Output Control Enabled** Setting this field to 1 enables software bit-bang control of the I2Cn bus. 0: The I²C controller manages the SDA and SCL pins in the hardware. 1: SDA and SCL are controlled by the software using the *I2Cn_CTRL.sda_out* and *I2Cn_CTRL.scl_out* fields.
9	sda	R	-	**SDA Status** 0: SDA pin is logic low. 1: SDA pin is logic high.
8	scl	R	-	**SCL Status** 0: SCL pin is logic low. 1: SCL pin is logic high.
7	sda_out	R/W	0	**SDA Pin Output Control** Set the state of the SDA hardware pin (actively pull low or float). 0: Pull SDA low. 1: Release SDA. *Note: Only valid when I2Cn_CTRL.bb_mode=1.*
6	scl_out	R/W	0	**SCL Pin Output Control** Set the state of the SCL hardware pin (actively pull low or float). 0: Pull SCL low. 1: Release SCL. *Note: Only valid when I2Cn_CTRL.bb_mode =1.*
5	-	RO	0	**Reserved**
4	irxm_ack	R/W	0	**IRXM Acknowledge** If IRXM is enabled (*I2Cn_CTRL.irxm_en* = 1), this field determines if the hardware sends an ACK or a NACK to an IRXM transaction. 0: Respond to IRXM with ACK. 1: Respond to IRXM with NACK.
3	irxm_en	R/W	0	**IRXM Enable** When receiving data, this field allows for an IRXM interrupt event after each received byte of data. The I2Cn peripheral hardware can be enabled to send either an ACK or NACK for IRXM. See the *Interactive Receive Mode* section for detailed information. 0: Disabled. 1: Enabled. *Note: Only set this field when the I²C bus is inactive.*

I^{2}C Control				I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
2	gc_addr_en	R/W	0	**General Call Address Enable** 0: Ignore general call address. 1: Acknowledge general call address.	
1	mst_mode	R/W	0	**Master Mode Enable** 0: Slave mode enabled. 1: Master mode enabled.	
0	en	R/W	0	**I^{2}C Peripheral Enable** 0: Disabled. 1: Enabled.	

Table 11-7: I^{2}C Status Register

I^{2}C Status				I2Cn_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:6	-	RO	0	**Reserved**	
5	mst_busy	RO	0	**Master Mode I^{2}C Bus Transaction Active** The peripheral is operating in master mode, and a valid transaction beginning with a START command is in progress on the I^{2}C bus. This bit reads 1 until the master ends the transaction with a STOP command. This bit continues to read 1 while a slave performs clock stretching. 0: Device not actively driving SCL clock cycles. 1: Device operating as master and actively driving SCL clock cycles.	
4	tx_full	RO	0	**Transmit FIFO Full** 0: Not full. 1: Full.	
3	tx_em	RO	1	**Transmit FIFO Empty** 0: Not empty. 1: Empty.	
2	rx_full	RO	0	**Receive FIFO Full** 0: Not full. 1: Full.	
1	rx_em	RO	1	**Receive FIFO Empty** 0: Not empty. 1: Empty.	
0	busy	RO	0	**Master or Slave Mode I^{2}C Busy Transaction Active** The peripheral is operating in master or slave mode, and a valid transaction beginning with a START command is in progress on the I^{2}C bus. This bit reads 1 until the peripheral acting as a master or an external master ends the transaction with a STOP command. This bit continues to read 1 while a slave performs clock stretching. 0: I^{2}C bus is idle. 1: I^{2}C bus transaction in progress.	

Table 11-8: I^{2}C Interrupt Flag 0 Register

I^{2}C Interrupt Flag 0				I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:24	-	RO	0	**Reserved**	

I²C Interrupt Flag 0				I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**	
23	wr_addr_match	R/W1C	0	**Slave Write Address Match Interrupt Flag** If set, the device has been accessed for a write (i.e., receive) transaction in slave mode, and the address received matches the device slave address. 0: No address match. 1: Address match.	
22	rd_addr_match	R/W1C	0	**Slave Read Address Match Interrupt Flag** If set, the device has been accessed for a read (i.e., transmit) transaction in slave mode, and the address received matches the device slave address. 0: No address match. 1: Address match.	
21:17	-	RO	0	**Reserved**	
16	-	R/W1C	0	**MAMI Interrupt Flag**	
15	tx_lockout	R/W1C	0	**Transmit FIFO Locked Interrupt Flag** If set, the transmit FIFO is locked and writes to the transmit FIFO are ignored. When set, the transmit FIFO is automatically flushed. Writes to the transmit FIFO are ignored until this flag is cleared. Write 1 to clear. 0: transmit FIFO not locked. 1: transmit FIFO is locked, and all writes to the transmit FIFO are ignored.	
14	stop_err	R/W1C	0	**Out of Sequence STOP Interrupt Flag** This flag is set if a STOP condition occurs out of the expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence STOP condition occurred.	
13	start_err	R/W1C	0	**Out of Sequence START Interrupt Flag** This flag is set if a START condition occurs out of the expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence START condition occurred.	
12	dnr_err	R/W1C	0	**Slave Mode Do Not Respond Interrupt Flag** This flag is set if an address match is made, but the transmit FIFO or receive FIFO is not ready. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: I²C address match has occurred, and either the transmit or receive FIFO is not configured.	
11	data_err	R/W1C	0	**Master Mode Data NACK from External Slave Interrupt Flag** The hardware sets this flag if a NACK is received from a slave. This flag is only valid if the I2Cn peripheral is configured for master mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Data NACK received from a slave.	
10	addr_nack_err	R/W1C	0	**Master Mode Address NACK from Slave Error Flag** The hardware sets this flag if an Address NACK is received from a slave bus. This flag is only valid if the I2Cn peripheral is configured for master mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Address NACK received from a slave.	

I²C Interrupt Flag 0				I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**	
9	to_err	R/ W1C	0	**Timeout Error Interrupt Flag** This flag is set when this device holds SCL low longer than the programmed timeout value. This field's setting applies to both 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_err	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.	
7	addr_ack	R/ W1C	0	**Master Mode Address ACK from External Slave Interrupt Flag** This field is set when a slave address ACK is received. Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: The slave device ACK for the address was received.	
6	stop	R/ W1C	0	**Slave Mode STOP Condition Interrupt Flag** This flag is set by hardware when a STOP condition is detected. Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: Condition occurred.	
5	tx_thd	RO	1	**Transmit FIFO Threshold Level Interrupt Flag** The hardware sets this field if the number of bytes in the Transmit FIFO is less than or equal to the Transmit FIFO threshold level. Write 1 to clear. This field is automatically cleared by the hardware when the transmit FIFO contains fewer bytes than the transmit threshold level. 0: Transmit FIFO contains more bytes than the transmit threshold level. 1: Transmit FIFO contains the transmit threshold level or fewer bytes (Default).	
4	rx_thd	R/W1C	1	**Receive FIFO Threshold Level Interrupt Flag** The hardware sets this field if the number of bytes in the Receive FIFO is greater than or equal to the Receive FIFO threshold level. This field is automatically cleared when the receive FIFO contains fewer bytes than the receive threshold setting. 0: receive FIFO contains fewer bytes than the receive threshold level. 1: receive FIFO contains at least receive threshold level of bytes (Default).	
3	addr_match	R/W1C	0	**Slave Mode Incoming Address Match Status Interrupt Flag** Write 1 to clear. Writing 0 has no effect. 0: Slave address match has not occurred. 1: Slave address match occurred.	
2	gc_addr_match	R/W1C	0	**Slave Mode General Call Address Match Received Interrupt Flag** Write 1 to clear. Writing 0 has no effect. 0: General call address match has not occurred. 1: General call address match occurred.	
1	irxm	R/W1C	0	**Interactive Receive Mode Interrupt Flag** Write 1 to clear. Writing 0 is ignored. 0: Interrupt condition has not occurred. 1: Interrupt condition occurred.	

I²C Interrupt Flag 0				I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**	
0	done	R/W1C	0	**Transfer Complete Interrupt Flag** This flag is set for both master and slave mode once a transaction completes. Write 1 to clear. Writing 0 has no effect. 0: Transfer is not complete. 1: Transfer complete.	

Table 11-9: I²C Interrupt Enable 0 Register

I²C Interrupt Enable 0				I2Cn_INTEN0	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:24	-	RO	0	**Reserved**	
23	wr_addr_match	R/W	0	**Slave Write Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in slave mode, and the address received matches the device slave addressed for a write transaction. 0: Disabled. 1: Enabled.	
22	rd_addr_match	R/W	0	**Slave Read Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in slave mode, and the address received matches the device slave addressed for a read transaction. 0: Disabled. 1: Enabled.	
21:17	-	RO	0	**Reserved**	
16	mami	R/W	0	**MAMI Interrupt Enable**	
15	tx_lockout	R/W	0	**Transmit FIFO Lock Out Interrupt Enable** 0: Disabled. 1: Enabled.	
14	stop_err	R/W	0	**Out of Sequence STOP Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.	
13	start_err	R/W	0	**Out of Sequence START Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.	
12	dnr_err	R/W	0	**Slave Mode Do Not Respond Interrupt Enable** Set this field to enable interrupts in slave mode when the "Do Not Respond" condition occurs. 0: Interrupt disabled. 1: Interrupt enabled.	
11	data_err	R/W	0	**Master Mode Received Data NACK from Slave Interrupt Enable** 0: Disabled. 1: Enabled.	
10	addr_nack_err	R/W	0	**Master Mode Received Address NACK from Slave Interrupt Enable** 0: Disabled. 1: Enabled.	
9	to_err	R/W	0	**Timeout Error Interrupt Enable** 0: Disabled. 1: Enabled.	
8	arb_err	R/W	0	**Master Mode Arbitration Lost Interrupt Enable** 0: Disabled. 1: Enabled.	

I²C Interrupt Enable 0				I2Cn_INTEN0 [0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
7	addr_ack	R/W	0	**Received Address ACK from Slave Interrupt Enable** Set this field to enable interrupts for master mode slave device address ACK events. 0: Interrupt disabled. 1: Interrupt enabled.
6	stop	R/W	0	**STOP Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
5	tx_thd	R/W	0	**Transmit FIFO Threshold Level Interrupt Enable** 0: Disabled. 1: Enabled.
4	rx_thd	R/W	0	**Receive FIFO Threshold Level Interrupt Enable** 0: Disabled. 1: Enabled.
3	addr_match	R/W	0	**Slave Mode Incoming Address Match Interrupt Enable** 0: Disabled. 1: Enabled.
2	gc_addr_match	R/W	0	**Slave Mode General Call Address Match Received Interrupt Enable** 0: Disabled. 1: Enabled.
1	irxm	R/W	0	**Interactive Receive Interrupt Enable** 0: Disabled. 1: Enabled.
0	done	R/W	0	**Transfer Complete Interrupt Enable** 0: Disabled. 1: Enabled.

Table 11-10: I²C Interrupt Flag 1 Register

I²C Interrupt Status Flags 1				I2Cn_INTFL1 [0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	start	R/W1C	0	**START Condition Status Flag** If set, a device START condition has been detected. 0: START condition not detected. 1: START condition detected.
1	tx_un	R/W1C	0	**Slave Mode Transmit FIFO Underflow Status Flag** In slave mode operation, the hardware sets this flag automatically if the transmit FIFO is empty and the master requests more data by sending an ACK after the previous byte is transferred. 0: Slave mode transmit FIFO underflow condition has not occurred. 1: Slave mode transmit FIFO underflow condition occurred.
0	rx_ov	R/W1C	0	**Slave Mode Receive FIFO Overflow Status Flag** In slave mode operation, the hardware sets this flag automatically when a receive FIFO overflow occurs. Write 1 to clear. Writing 0 has no effect. 0: Slave mode receive FIFO overflow event has not occurred. 1: Slave mode receive FIFO overflow condition occurred (data lost).

Table 11-11: I²C Interrupt Enable 1 Register

I²C Interrupt Enable 1			**I2Cn_INTEN1**	**[0x0014]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	start	R/W	0	**START Condition Interrupt Enable** 0: Disabled. 1: Enabled.
1	tx_un	R/W	0	**Slave Mode Transmit FIFO Underflow Interrupt Enable** 0: Disabled. 1: Enabled.
0	rx_ov	R/W	0	**Slave Mode Receive FIFO Overflow Interrupt Enable** 0: Disabled. 1: Enabled.

Table 11-12: I²C FIFO Length Register

I²C FIFO Length			**I2Cn_FIFOLEN**	**[0x0018]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:8	tx_depth	RO	8	**Transmit FIFO Length** This field returns the depth of the transmit FIFO. 8: 8-bytes.
7:0	rx_depth	RO	8	**Receive FIFO Length** This field returns the depth of the receive FIFO. 8: 8-bytes.

Table 11-13: I²C Receive Control 0 Register

I²C Receive Control 0			**I2Cn_RXCTRL0**	**[0x001C]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	thd_lvl	R/W	0	**Receive FIFO Threshold Level** Set this field to the required number of bytes to trigger a receive FIFO threshold event. When the number of bytes in the receive FIFO is equal to or greater than this field, the hardware sets the *I2Cn_INTFL0*.*rx_thd* bit indicating a receive FIFO threshold level event. 0: 0 bytes or more in the receive FIFO causes a threshold event. 1: 1+ bytes in the receive FIFO triggers a receive threshold event (recommended minimum value). … 8: Receive FIFO threshold event only occurs when the receive FIFO is full.
7	flush	R/W1O	0	**Flush Receive FIFO** Write 1 to this field to initiate a receive FIFO flush, clearing all data in the receive FIFO. This field is automatically cleared by the hardware when the receive FIFO flush completes. Writing 0 has no effect. 0: Receive FIFO flush complete or not active. 1: Flush the receive FIFO.
6:1	-	RO	0	**Reserved**

I²C Receive Control 0				I2Cn_RXCTRL0	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**	
0	dnr	R/W	0	**Slave Mode Do Not Respond** Slave mode operation only. If the device has been addressed for a write operation, and there is still data in the receive FIFO, then: 0: Always respond to an address match with an ACK but always respond to data bytes with a NACK. 1: NACK the address.	

Table 11-14: I²C Receive Control 1 Register

I²C Receive Control 1				I2Cn_RXCTRL1	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:12	-	RO	0	**Reserved**	
11:8	lvl	R	0	**Receive FIFO Byte Count Status** This field returns the number of bytes in the receive FIFO. 0: 0 bytes (No data). 1: 1 byte. 2: 2 bytes. 3: 3 bytes. 4: 4 bytes. 5: 5 bytes. 6: 6 bytes. 7: 7 bytes. 8: 8 bytes.	
7:0	cnt	R/W	1	**Receive FIFO Transaction Byte Count Configuration** In master mode, write the number of bytes to be received in a transaction from 1 to 256. 0x00 represents 256. 0: 256 byte receive transaction. 1: 1 byte receive transaction. 2: 2 byte receive transaction. … 255: 255 byte receive transaction. *This field is ignored when I2Cn_CTRL.irxm_en = 1. To receive more than 256 bytes, use I2Cn_CTRL.irxm_en = 1.*	

Table 11-15: I²C Transmit Control 0 Register

I²C Transmit Control 0				I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:12	-	RO	0	**Reserved**	
11:8	thd_val	R/W	0	**Transmit FIFO Threshold Level** This field sets the level for a transmit FIFO threshold event interrupt. If the number of bytes remaining in the transmit FIFO falls to this level or lower, the interrupt flag *I2Cn_INTFL0.tx_thd* is set, indicating a transmit FIFO threshold event occurred. 0: 0 bytes remaining in the transmit FIFO triggers a transmit FIFO threshold event. 1: 1 byte or fewer remaining in the transmit FIFO triggers a transmit FIFO threshold event (recommended minimum value). … 7: 7 or fewer bytes remaining in the transmit FIFO triggers a transmit FIFO threshold event.	

I²C Transmit Control 0				I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**	
7	flush	R/W1O	0	**Transmit FIFO Flush** A transmit FIFO flush clears all remaining data from the transmit FIFO. 0: transmit FIFO flush is complete or not active. 1: Flush the transmit FIFO. *Note: The hardware automatically clears this bit to 0 after it is written to 1 when the flush is completed.* If *I2Cn_INTFL0.tx_lockout* = 1, then *I2Cn_TXCTRL0.flush* = 1.	
6	-	RO	0	**Reserved**	
5	nack_flush_dis	R/W	0	**Transmit FIFO received NACK Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Received NACK at the end of a slave transmit operation enabled. 1: Received NACK at the end of a slave transmit operation disabled. *Note: Upon entering transmit preload mode, the hardware automatically sets this bit to 0. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 0).*	
4	rd_addr_flush_dis	R/W	0	**Transmit FIFO Slave Address Match Read Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled. 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bitfield to 1).*	
3	wr_addr_flush_dis	R/W	0	**Transmit FIFO Slave Address Match Write Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled. 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 1).*	
2	gc_addr_flush_dis	R/W	0	**Transmit FIFO General Call Address Match Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled. 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 1).*	
1	tx_ready_mode	R/W	0	**Transmit FIFO Ready Manual Mode** 0: The hardware controls *I2Cn_TXCTRL1.preload_rdy.* 1: Software control of *I2Cn_TXCTRL1.preload_rdy.*	

I²C Transmit Control 0				I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**	
0	preload_mode	R/W	0	**Transmit FIFO Preload Mode Enable** 0: Normal operation. An address match in slave mode, or a general call address match, flushes and locks the transmit FIFO so it cannot be written and set *I2Cn_INTFL0.tx_lockout.* 1: Transmit FIFO preload mode. An address match in slave mode, or a general call address match, does not lock the transmit FIFO and does not set *I2Cn_INTFL0.tx_lockout*, allowing the software to preload data into the transmit FIFO. The status of the I²C is controllable using *I2Cn_TXCTRL1.preload_rdy.*	

Table 11-16: I²C Transmit Control 1 Register

I²C Transmit Control Register 1				I2Cn_TXCTRL1	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:12	-	RO	0	**Reserved**	
11:8	lvl	R	0	**Transmit FIFO Byte Count Status** 0: 0 bytes (No data). 1: 1 byte. 2: 2 bytes. 3: 3 bytes. 4: 4 bytes. 5: 5 bytes. 6: 6 bytes. 7: 7 bytes. 8: 8 bytes (max value).	
7:1	-	RO	0	**Reserved**	
0	preload_rdy	R/W1O	1	**Transmit FIFO Preload Ready Status** When transmit FIFO preload mode is enabled, *I2Cn_TXCTRL0.preload_mode* = 1, this bit is automatically cleared to 0. While this bit is 0, if the I2Cn hardware receives a 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_TXCTRL0.preload_mode* = 1, this bit is forced to 1, and the I2Cn hardware behaves normally.	

Table 11-17: I²C Data Register

I²C Data				I2Cn_FIFO	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:8	-	RO	0	**Reserved**	
7:0	data	R/W	0xFF	**FIFO Data** Reads from this register pop data off the receive FIFO. Writes to this register push data onto the transmit FIFO. Reading from an empty receive FIFO returns 0xFF. Writes to a full transmit FIFO are ignored.	

Table 11-18: I²C Master Control Register

I²C Master Control				I2Cn_MSTCTRL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:11	-	RO	0	**Reserved**	
10:8	-	RO	0	**Reserved**	

I²C Master Control			I2Cn_MSTCTRL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
7	ex_addr_en	R/W	0	**Slave Extended Addressing Enable** 0: Send a 7-bit address to the slave. 1: Send a 10-bit address to the slave.
6:3	-	RO	0	**Reserved**
2	stop	R/W1O	0	**Send STOP Condition** 1: Send a STOP Condition at the end of the current transaction. *Note: This bit is automatically cleared by the hardware when the STOP condition begins.*
1	restart	R/W1O	0	**Send Repeated START Condition** After sending data to a slave, the master can send another START to retain control of the bus. 1: Send a repeated START condition to the slave instead of sending a STOP condition at the end of the current transaction. *Note: This bit is automatically cleared by the hardware when the repeated START condition begins.*
0	start	R/W1O	0	**Start Master Mode Transfer** 1: Start master mode transfer. *Note: This bit is automatically cleared by the hardware when the transfer is completed or aborted.*

Table 11-19: I²C SCL Low Control Register

I²C Clock Low Control			I2Cn_CLKLO	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	lo	R/W	1	**Clock Low Time** In master mode, this configures the SCL low time. $t_{SCL\_LO} = f_{I2C\_CLK} \times (lo + 1)$ *Note: 0 is not a valid setting for this field.*

Table 11-20: I²C SCL High Control Register

I²C Clock High Control			I2Cn_CLKHI	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	hi	R/W	1	**Clock High Time** In master mode, this configures the SCL high time. $t_{SCL\_HI} = {}^{1}/_{f_{I2C\_CLK}} \times (hi + 1)$ In both master and slave mode, this configures the time SCL is held low after new data is loaded from the transmit FIFO or after the software clears *I2Cn_INTFL0.irxm* during IRXM. *Note: 0 is not a valid setting for this field.*

Table 11-21: I²C Hs-Mode Clock Control Register

I²C Hs-Mode Clock Control			I2Cn_HSCLK	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**

I^2C Hs-Mode Clock Control			I2Cn_HSCLK	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
15:8	hi	R/W	0	**Hs-Mode Clock High Time** This field sets the Hs-Mode clock high count. In slave mode, this is the time SCL is held high after data is output on SDA. *Note: See* SCL Clock Generation for Hs-Mode *for details on the requirements for the Hs-Mode clock high and low times.*
7:0	lo	R/W	0	**Hs-Mode Clock Low Time** This field sets the Hs-Mode clock low count. In slave mode, this is the time SCL is held low after data is output on SDA. *Note: See* SCL Clock Generation for Hs-Mode *for details on the requirements for the Hs-Mode clock high and low times.*

Table 11-22: I^2C Timeout Register

I^2C Timeout			I2Cn_TIMEOUT	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	scl_to_val	R/W	0	**Bus Error SCL Timeout Period** Set this value to the number of I^2C clock cycles desired to cause a bus timeout error. The peripheral timeout timer starts when it pulls SCL low. After the peripheral releases the line, if the line is not pulled high before the timeout number of I^2C clock cycles, a bus error condition is set (*I2Cn_INTFL0.to_err* = 1), and the peripheral releases the SCL and SDA lines. 0: Timeout disabled. All other values result in a timeout calculation of: $t_{BUS\_TIMEOUT} = {}^{1}/_{f_{I2C\_CLK}} \times scl\_to\_val$ *Note: The timeout counter monitors the I2Cn peripheral's driving of the SCL pin, not an external I^2C device driving the SCL pin.*

Table 11-23: I^2C DMA Register

I^2C DMA			I2Cn_DMA	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved**
1	rx_en	R/W	0	**Receive DMA Channel Enable** 0: Disabled. 1: Enabled.
0	tx_en	R/W	0	**Transmit DMA Channel Enable** 0: Disabled. 1: Enabled.

Table 11-24: I^2C Slave Address Register

I^2C Slave Address			I2Cn_SLAVE	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ext_addr_en	R/W	0	**Slave Mode Extended Address Length Select** 0: 7-bit addressing. 1: 10-bit addressing.
14:10	-	RO	0	**Reserved**

I^{2}C Slave Address			I2Cn_SLAVE	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
9:0	addr	R/W	0	**Slave Mode Slave Address** In slave mode operation, (*I2Cn_CTRL.mst_mode* = 0), set this field to the slave address for the I2Cn port. For 7-bit addressing, the address occupies the least significant 7 bits. For 10-bit addressing, the 9-bits of address occupies the most significant 9 bit, and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

12. Inter-Integrated Sound (I^{2}S) Interface (I2S)

I^{2}S is a serial audio interface for communicating pulse-code modulation (PCM) encoded streams between devices. The peripheral supports both master and slave modes.

Key features:

- Stereo (2 channel) and mono (left or right channel option) formats
- Separate DMA channels for transmit and receive
- Flexible timing
  - Configurable sampling rate from $^1/_{65536}$ to 1 of the I^{2}S input clock
- Flexible data format
  - Number of bits per data word can be selected from 1 to 32, typically 8, 16, 24, or 32-bit width
  - Feature enhancement not in the I^{2}S Specification
    - Word/Channel select polarity control
    - First bit position selection
    - Selectable FIFO data alignment to the MSB or the LSB of the sample
    - Sample size less than the word size with adjustment to MSB or LSB of the word
    - Optional sign extension
- Full-duplex serial communication with separate I^{2}S serial data input and serial data output pins

12.1 Instances

Table 12-1: MAX32675 I^{2}S Instances

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

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

12.1.1 *I^{2}S Bus Lines and Definitions*

The I^{2}S peripheral includes support for the following signals:

1. Bit clock line
   - Continuous serial clock (SCK) referred to as bit clock (BCLK) in this document
2. Word clock line
   - Word select (WS) referred to as left right clock (LRCLK) in this document
3. Serial data input (SDI)
4. Serial data output (SDO)
5. ERFO is required for operation in master mode and must be enabled

Detailed pin and alternate function mapping is shown in *Table 12-2*.

Table 12-2: MAX32675 I^{2}S Pin Mapping

Instance	I^{2}S Signal	Pin Description	Notes
I2S	BCLK (SCK)	I^{2}S bit clock	Also referred to as serial clock
	LRCLK (WS)	I^{2}S left/right clock	Also referred to as word select
	SDI	I^{2}S serial data input	
	SDO	I^{2}S serial data output	

12.2 Details

The I^2S supports full-duplex serial communication with separate SDI and SDO pins. *Figure 12-1* shows an interconnect between a peripheral configured in host mode, communicating with an external I^2S slave receiver and an external I^2S transmitter. In master mode, the peripheral hardware generates the BCLK and LRCLK and both are output to each slave device.

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

Figure 12-1: I^2S Master Mode

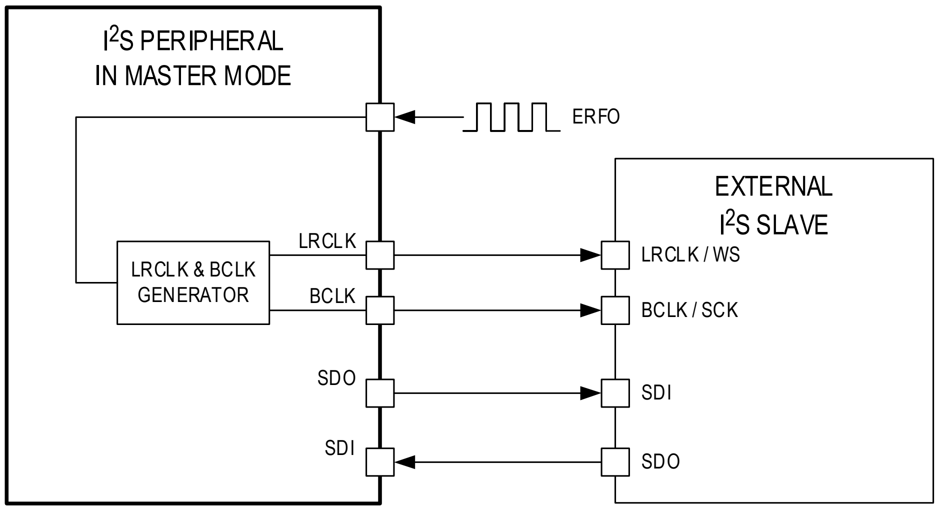
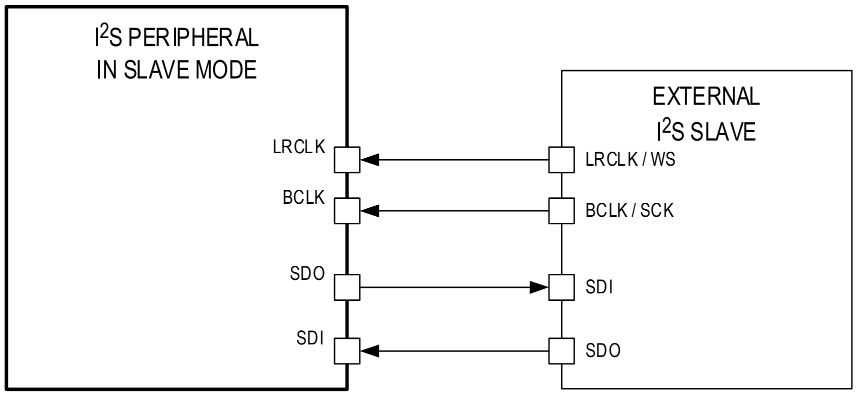


Figure 12-2 shows the I^2S peripheral configured for slave operation. The LRCLK and BCLK signals are generated externally and are inputs to the I^2S peripheral.

Figure 12-2: I^2S Slave Mode



12.3 Master and Slave Mode Configuration

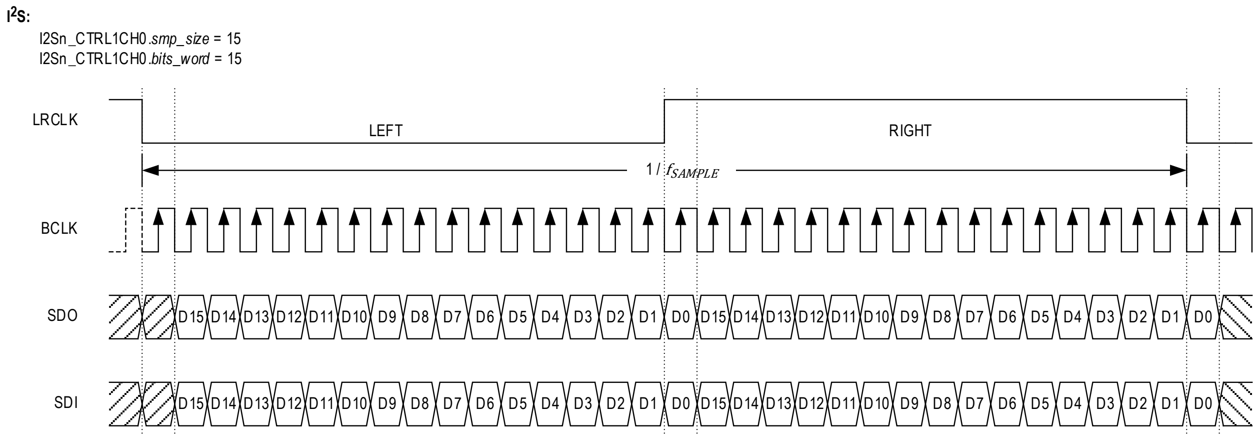
Hardware supports master and slave modes. In master mode, the BCLK and LRCK signals are generated internally and output on the BCLK and LRCLK pins. In slave mode, the BCLK and LRCLK pins are configured as inputs and the peripheral timing is controlled by the external clock source.

Table 12-3: I²S Mode Configuration

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

12.4 Clocking

Figure 12-3: Audio Interface I²S Signal Diagram


I²S:
I2Sn_CTRL1CH0.smp_size = 15
I2Sn_CTRL1CH0.bits_word = 15
LRCLK
LEFT
RIGHT
1 / fSAMPLE
BCLK
SDO
SDI
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0

I²S communication is synchronized using two signals, the LRCLK and the BCLK. When the I²S peripheral is configured as a master, the BCLK and LRCLK signals are generated internally by the peripheral using the ERFO. See *Table 12-2* for details of the I²S pin mapping and alternate function selection. If using the I²S peripheral in master mode, the ERFO must be enabled to generate the BCLK and LRCLK signals.

When the I²S peripheral is configured in slave mode, the BCLK and LRCLK pins must be configured as inputs. An external master generates the BCLK and LRCLK signals which the peripheral uses to synchronize itself to the I²S bus. *Figure 12-3* shows the default I²S signals and timing for I²S communication.

The BCLK frequency is the product of the sample rate, the number of bits per channel (left and right) and the number of channels. For CD audio sampled at a frequency of 44.1kHz, with 16-bit sample width, and stereo audio (left and right) the bit clock frequency, f_{BCLK}, is 1.4112MHz as shown in *Equation 12-1*.

Equation 12-1: CD Audio Bit Frequency Calculation

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

12.4.1 BCLK Generation for Master Mode

As indicated by *Equation 12-1*, the requirements for determining the BCLK frequency are:

1. Audio sample frequency
2. Number of bits per sample (sample width)

Using the above requirements, *Equation 12-2* shows the formula to calculate the bit clock frequency for a given audio file.

Equation 12-2: Calculating the Bit Clock Frequency for Audio

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

In master mode, the I²S external clock input is used to generate the BCLK frequency. The I²S external clock is divided by the *I2S_CTRL1CH0.clkdiv* field to achieve the target BCLK frequency as shown in *Equation 12-3*.

Equation 12-3: Master Mode BLCK Generation Using the I²S External Clock

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

Use *Equation 12-4* to determine the I²S clock divider for a target BCLK frequency.

Equation 12-4: Master Mode Clock Divisor Calculation for a Target Bit Clock Frequency

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

12.4.2 LRCLK Period Calculation

An I²S data stream can carry mono (either left or right channel) or stereo (left and right channel) data. The LRCLK signal indicates which channel is currently being sent, either left or right channel data, as shown in *Figure 12-3*. The LRCLK is a 50% duty cycle signal and is the same frequency as the audio sampling frequency, f_{SAMPLE}.

The I²S peripheral uses the bits per word field, *I2S_CTRL1CH0.bits_word*, to define the sample width of the audio, equivalent to the number of bit clocks per channel. This value should be set to the sample width of the audio minus 1. For example, software should set the *I2S_CTRL1CH0.bits_word* field to 15 for audio sampled using a 16-bit width.

Equation 12-5: Bits Per Word Calculation

$$I2Sn\_CTRL1CH0.bits\_word = Sample\ Width - 1$$

The LRCLK frequency, or word select frequency, is automatically generated by the I²S peripheral hardware when it is set to operate as a master. The LRCLK frequency calculation is shown in *Equation 12-6*.

Equation 12-6: LRCLK Frequency Calculation

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

12.5 Data Formatting

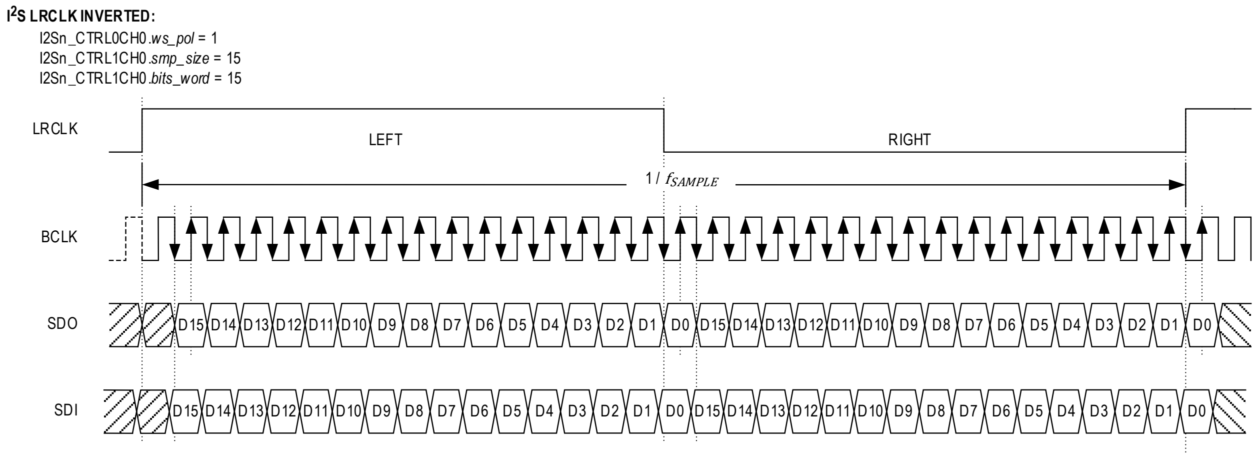
12.5.1 Sample Size

The Sample Size field, *I2S_CTRL1CH0.smp_size*, defines the number of desired samples within each channel, left, right or mono, for the peripheral. This field can be less than or equal to the *I2S_CTRL1CH0.bits_word* field. For example, for 16-bit sample width audio, the *I2S_CTRL1CH0.bits_word* field must be set to 15. However, the Sample Size field can be set from 0 to 15. Setting the Sample Size to 0 is equivalent to setting it to the value of the Bits Per Word field. The Sample Size field determines how many of the Bits per Word are transmitted or saved per channel. The Sample Size field is a 0 based field, therefore, setting *I2S_CTRL1CH0.smp_size* to 15 collects 16 samples. See *Figure 12-6* for an example of the bits per word field's setting compared to the sample size field's setting.

12.5.2 Word Select Polarity

Left channel data, by default, is transferred when the LRCLK signal is low, and right channel data is transferred when the LRCLK signal is high. The polarity of the LRCLK is programmable allowing left and right data to be swapped. The LRCLK polarity is controlled using the word select polarity field, *I2S_CTRL0CH0.ws_pol*. By default, LRCLK low is for the left channel, high is for the right channel as shown in *Figure 12-3*. Setting *I2S_CTRL0CH0.ws_pol* to 1 inverts the LRCLK polarity, using LRCLK high for the left channel and LRCLK low for the right channel as shown in *Figure 12-4*.

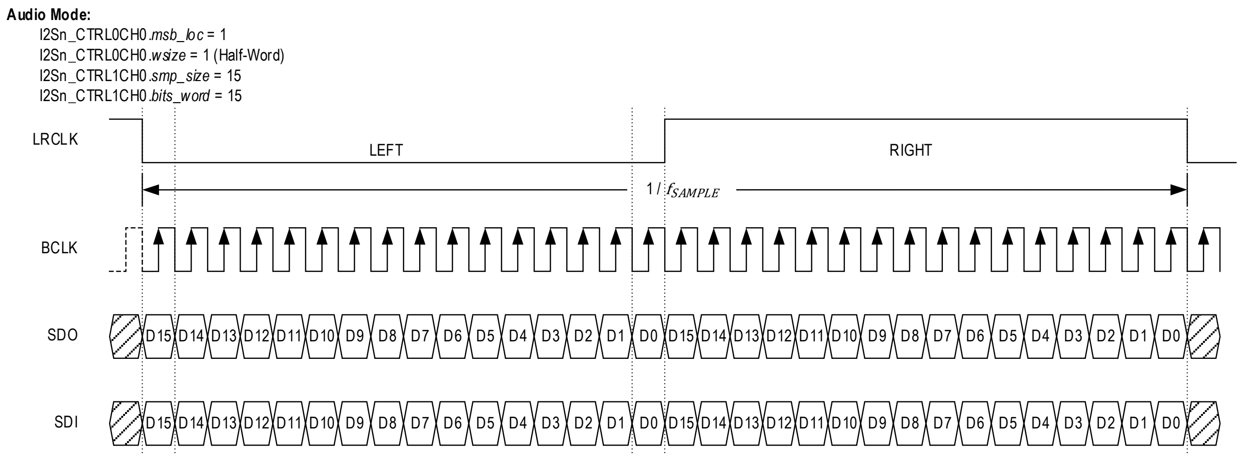
Figure 12-4: Audio Mode with Inverted Word Select Polarity


I2S LRCLK INVERTED:
I2Sn_CTRL0CH0.ws_pol = 1
I2Sn_CTRL1CH0.smp_size = 15
I2Sn_CTRL1CH0.bits_word = 15
LRCLK
LEFT
RIGHT
1 / fSAMPLE
BCLK
SDO
SDI
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0

12.5.3 First Bit Location Control

The default setting is for the first bit of I^{2}S data to be located at the second complete BCLK cycle after the LRCLK transition as required by the I^{2}S specification. See *Figure 12-3* for the standard data sampling configuration. Optionally, the first bit location can be left justified, resulting in the first bit of data being sampled on the first BCLK cycle after the LRCLK signal transitions as shown in *Figure 12-5*. Set *I2S_CTRL0CH0.msb_loc* to 1 to left justify the data with respect to the LRCLK.

Figure 12-5: Audio Master Mode Left-Justified First Bit Location


Audio Mode:
I2Sn_CTRL0CH0.msb_loc = 1
I2Sn_CTRL0CH0.wsize = 1 (Half-Word)
I2Sn_CTRL1CH0.smp_size = 15
I2Sn_CTRL1CH0.bits_word = 15
LRCLK
LEFT
RIGHT
1 / fSAMPLE
BCLK
SDO
SDI
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0

12.5.4 Sample Adjustment

When the sample size field, *I2S_CTRL1CH0.smp_size*, is less than the bits per word field, *I2S_CTRL1CH0.bits_word*, use the *I2S_CTRL1CH0.adjst* field to set which bits are stored in the receive FIFO or transmitted from the transmit FIFO, either from the first sample of the SDI/SDO line or the last sample of the SDI/SDO line for the left and right channels. *Figure 12-6* shows an example of the default adjustment, MSB, where *I2S_CTRL1CH0.smp_size* = 7 and *I2S_CTRL1CH0.bits_word* = 15. *Figure 12-7* shows the adjustment set to the LSB of the SDI/SDO data.

Figure 12-6: MSB Adjustment when Sample Size is Less Than Bits Per Word

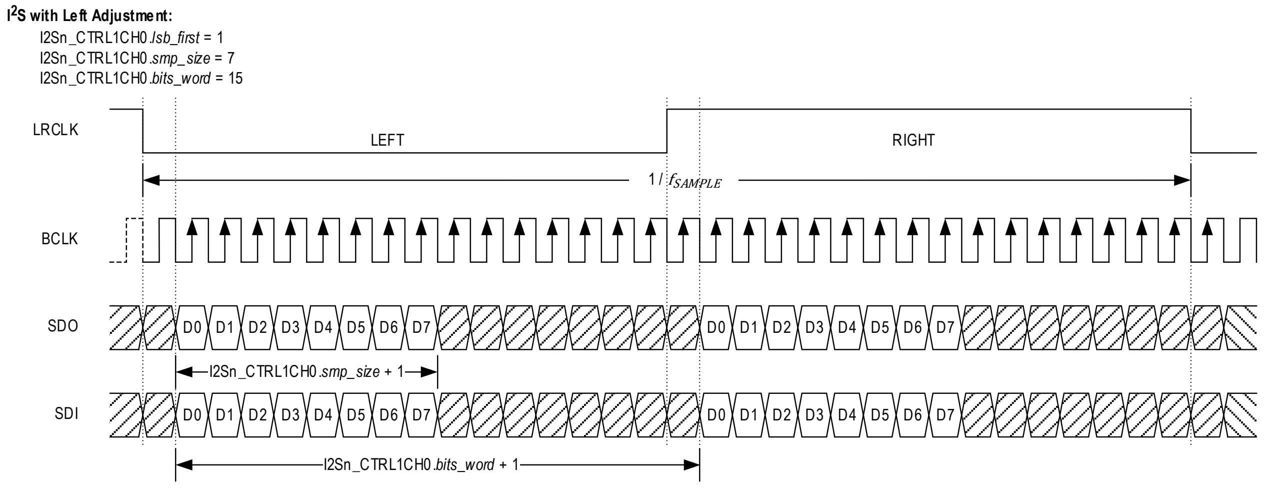
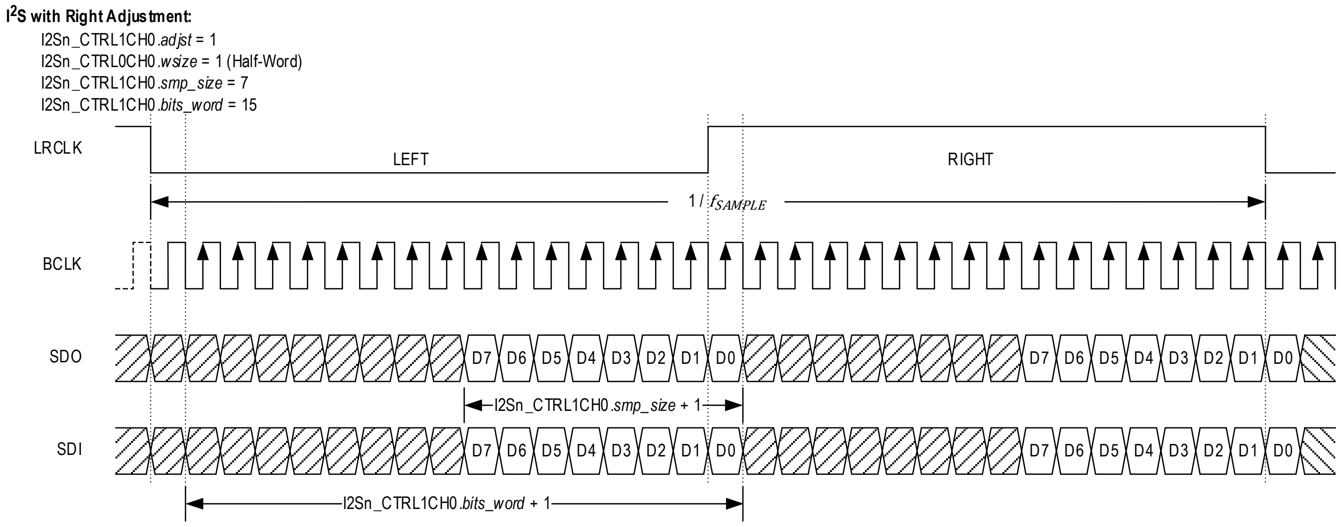

I2S with Left Adjustment:
I2Sn_CTRL1CH0.lsb_first = 1
I2Sn_CTRL1CH0.smp_size = 7
I2Sn_CTRL1CH0.bits_word = 15
LRCLK
LEFT
RIGHT
1 / fSAMPLE
BCLK
SDO
D0 D1 D2 D3 D4 D5 D6 D7
I2Sn_CTRL1CH0.smp_size + 1
SDI
I2Sn_CTRL1CH0.bits_word + 1

Figure 12-7: LSB Adjustment when Sample Size is Less Than Bits Per Word


I2S with Right Adjustment:
I2Sn_CTRL1CH0.adjst = 1
I2Sn_CTRL0CH0.wsize = 1 (Half-Word)
I2Sn_CTRL1CH0.smp_size = 7
I2Sn_CTRL1CH0.bits_word = 15
LRCLK
LEFT
RIGHT
1 / fSAMPLE
BCLK
SDO
D7 D6 D5 D4 D3 D2 D1 D0
I2Sn_CTRL1CH0.smp_size + 1
SDI
I2Sn_CTRL1CH0.bits_word + 1

12.5.5 Stereo/Mono Configuration

The I^{2}S can transfer stereo or mono data based on the *I2S_CTRL0CH0.stereo* field. In stereo mode, the default mode, both the left and right channels hold data. In mono mode, only the left or right channel contain data. For stereo mode, set *I2S_CTRL0CH0.stereo* to 0. Set *I2S_CTRL0CH0.stereo* field to 2 for left channel mono. Set *I2S_CTRL0CH0.stereo* field to 3 for right channel mono.

Figure 12-8: I²S Mono Left Mode

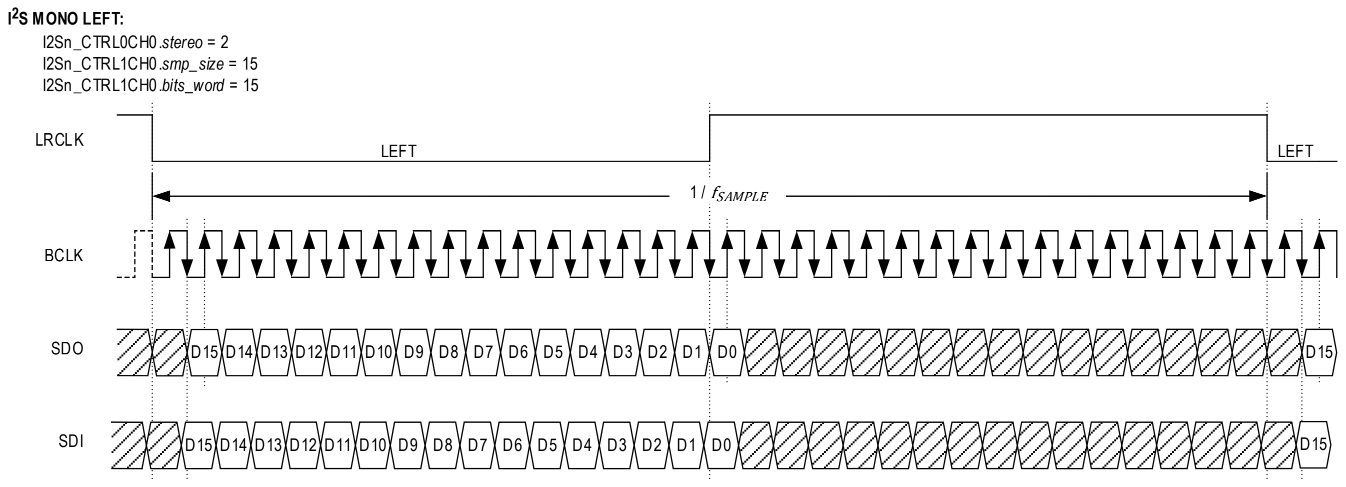
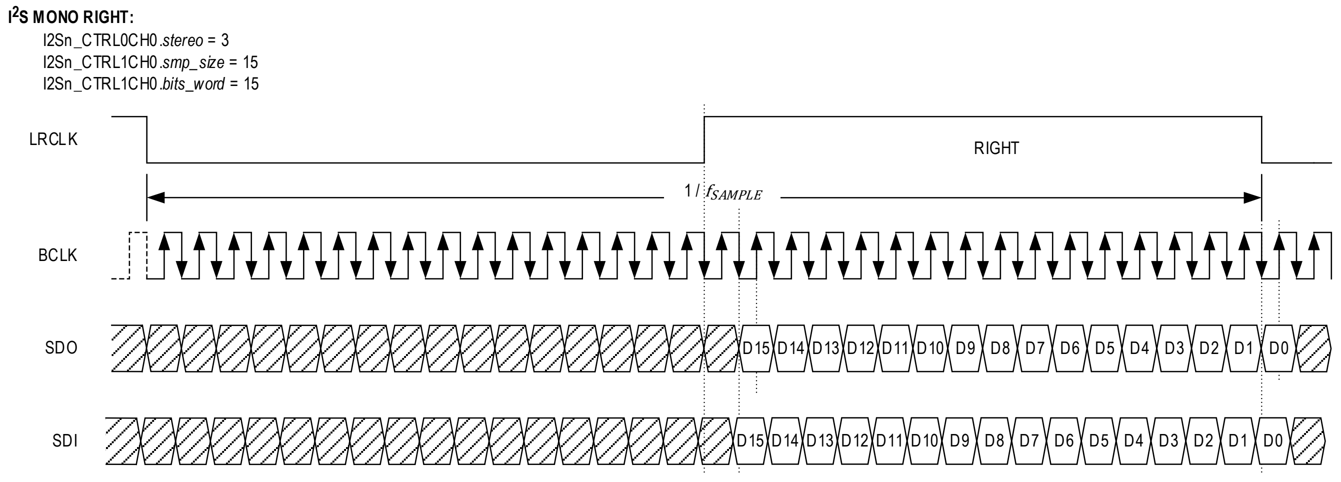

I²S MONO LEFT:
I2Sn_CTRL0CH0.stereo = 2
I2Sn_CTRL1CH0.smp_size = 15
I2Sn_CTRL1CH0.bits_word = 15
LRCLK
LEFT
LEFT
1 / fSAMPLE
BCLK
SDO
SDI
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
D15

Figure 12-9: I²S Mono Right Mode


I²S MONO RIGHT:
I2Sn_CTRL0CH0.stereo = 3
I2Sn_CTRL1CH0.smp_size = 15
I2Sn_CTRL1CH0.bits_word = 15
LRCLK
RIGHT
1 / fSAMPLE
BCLK
SDO
SDI
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0

12.6 Transmit and Receive FIFOs

12.6.1 FIFO Data Width

I²S audio data is programmable from 1 to 32 bits using the *I2S_CTRL1CH0.bits_word* field. Software can set the FIFO width to either 8 bits (byte), 16 bits (half-word) or 32 bits (word). Set the FIFO width using the *I2S_CTRL0CH0.wsize* field. For FIFO word sizes less than 32 bits, the data frame, comprised of a full LRCLK cycle, may still be 64 bits; the unused bits are transmitted as zero by the hardware.

12.6.2 Transmit FIFO

An I²S transaction is started by writing data to the transmit FIFO using the *I2S_FIFOCH0.data* register, either directly or using a DMA channel. The data written is automatically transmitted out by the hardware a FIFO word, as defined using the I2S_CTRL0CH0.*wsize* field, at a time, in the order it was written to the transmit FIFO. Use the I²S interrupt flags to monitor the transmit FIFO status and determine when the transfer cycle(s) have completed.

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

12.6.3 Receive FIFO

Received data is loaded into the receive FIFO and it can then be unloaded by reading from the *I2S_FIFOCH0.data* register. An overrun event occurs if the receive FIFO is full and another word is shifted into the FIFO.

12.6.4 FIFO Word Control

The data width of the transmit and receive FIFOs can be configured using the *I2S_CTRL0CH0.wsize* field. The tables below describe the data ordering based on the *I2S_CTRL0CH0.wsize* setting.

The transmit and receive FIFOs must be flushed and the peripheral reset by software before reconfiguration. Software resets the peripheral by setting the *I2S_CTRL0CH0.rst* field to 1.

Table 12-4: Data Ordering for Byte, Half-Word and Word Data Sizes (Stereo Mode)

Byte Data Width (I2S_CTRL0CH0.*wsize* = 0)				
FIFO Entry	**MSByte**			**LSByte**
FIFO 0	Right Channel Byte 1	Left Channel Byte 1	Right Channel Byte 0	Left Channel Byte 0
FIFO 1	Right Channel Byte 3	Left Channel Byte 3	Right Channel Byte 2	Left Channel Byte 2
…	…	…	…	…
FIFO 7	Right Channel Byte 14	Left Channel Byte 14	Right Channel Byte 13	Left Channel Byte 13

Table 12-5: Data Ordering for Half-Word Data Size (Stereo Mode)

Half-Word Data Width (I2S_CTRL0CH0.*wsize* = 1)		
FIFO Entry	**Most Significant Half-Word**	**Least Significant Half-Word**
FIFO 0	Right Channel Half-Word 0	Left Channel Half-Word 0
FIFO 1	Right Channel Half-Word 1	Left Channel Half-Word 1
…	…	…
FIFO 7	Right Channel Half Word 7	Left Channel Half-Word 7

Table 12-6: Data Ordering for Word Data Size (Stereo Mode)

Word Data Width (I2S_CTRL0CH0.*wsize* = 2 or 3)	
FIFO Entry	**Word**
FIFO 0	Left Channel Word 0
FIFO 1	Right Channel Word 0
FIFO 2	Left Channel Word 1
FIFO 3	Right Channel Word 1
…	…
FIFO 6	Left Channel Word 3
FIFO 7	Right Channel Word 3

12.6.5 FIFO Data Alignment

The I^{2}S data can be left aligned (reset default), or right aligned using the *I2S_CTRL0CH0.align* field. The following conditions apply to each setting:

Left aligned: *I2S_CTRL0CH0.align* = 0

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

Right aligned: *I2S_CTRL0CH0.align* = 1

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

12.6.6 Typical Audio Configurations

Table 12-7: shows the relationship between the bits per word field and the sample size field. *Equation 12-7* shows the required relationship between the sample size field and the bits per word field.

Equation 12-7: Sample Size Relationship Bits per Word

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

The *I2S_CTRL1CH0.bits_word* column in *Table 12-7* is set by the equation $\frac{\#\,BCLK}{Channel} - 1$. The *I2S_CTRL1CH0.smp_size* column is the number of samples per word captured from the I^{2}S bus and is calculated by the equation $\frac{\#\,Samples}{Channel} - 1$. Channel refers to the left and right channels of audio.

Table 12-7: Configuration for Typical Audio Width and Samples per WS Clock Cycle

Audio Sample Width/ Samples per WS Cycle	$\frac{\#\,BCLK}{Channel}$	$\frac{\#\,Samples}{Channel}$	*I2S_CTRL1CH0*			Sign extension (align = 1)†
			bits_word	*smp_size*	*wsize*	
8-bit / 16	8	8	7	7	0	
16-bit / 32	16	16	15	15	1	
20-bit / 40	20	20	19	19	2	sign
24-bit / 48	24	24	23	23	2	sign
24-bit / 64	32	24	31	23	2	sign
32-bit / 64	32	32	31	31	2	

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

12.7 Interrupt Events

The I^{2}S peripheral generates interrupts for the events shown in *Table 12-8*. An interrupt is generated if the corresponding interrupt enable field is set. The interrupt flags stay set until cleared by software by writing 1 to the interrupt flag field.

Table 12-8: I^{2}S Interrupt Events

Event	Interrupt Flag	Interrupt Enable
Receive FIFO overrun	*I2S_INTFL.rx_ov_ch0*	*I2S_INTEN.rx_ov_ch0*
Receive threshold	*I2S_INTFL.rx_thd_ch0*	*I2S_INTEN.rx_thd_ch0*
Transmit FIFO half-empty	*I2S_INTFL.tx_he_ch0*	*I2S_INTEN.tx_he_ch0*
Transmit FIFO one byte remaining	*I2S_INTFL.tx_ob_ch0*	*I2S_INTEN.tx_ob_ch0*

12.7.1 Receive FIFO Overrun

A receive FIFO overrun event occurs if the number of data words in the receive FIFO, *I2S_DMACH0.rx_lvl* is equal to the RX_FIFO_DEPTH and another word has been shifted into the FIFO. Hardware automatically sets the *I2S_INTFL.rx_ov_ch0* field to 1 when this event occurs.

12.7.2 Receive FIFO Threshold

A receive FIFO threshold event occurs when a word is shifted in and the number of words in the receive FIFO, *I2S_DMACH0.rx_lvl*, exceeds the *I2S_CTRL0CH0.rx_thd_val*. The event does not occur if the opposite transition occurs. When this event occurs hardware automatically sets the *I2S_INTFL.rx_thd_ch0* field to 1.

12.7.3 Transmit FIFO Half-Empty

A transmit FIFO half-empty event occurs when the number of words in the transmit FIFO, *I2S_DMACH0.tx_lvl*, is less than ½ of the TX_FIFO_DEPTH as shown in *Equation 12-8*. When this event occurs the *I2S_INTFL.tx_he_ch0* flag is set to 1 by hardware.

Note: The transmit FIFO half empty interrupt flag is set by hardware one BCLK cycle before the actual condition occurring. If the BCLK is much slower than the I^{2}S peripheral clock, software may receive the interrupt while the actual transmit FIFO level is still equal to ½ of the TX_FIFO_DEPTH. Software should always read the transmit FIFO level before filling it to determine the correct number of words to write to the transmit FIFO. Read the level of the transmit FIFO using the I2S_DMACH0.tx_lvl field.

Equation 12-8: Transmit FIFO Half-Empty Condition

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

12.7.4 Transmit FIFO One Entry Remaining

A transmit FIFO one entry remaining event occurs when the number of entries in the transmit FIFO is 1, *I2S_DMACH0*.*tx_lvl* = 1. When this event occurs, the *I2S_INTFL*.*tx_ob_ch0* flag is set to 1 by hardware.

Note: The transmit FIFO one entry remaining interrupt flag is set by hardware one BCLK cycle before the actual condition occurring. If the BCLK is much slower than the I²S peripheral clock, software may receive the interrupt while the actual transmit FIFO level is still equal to 2. Software should always read the transmit FIFO level before filling it to determine the correct number of words to write to the transmit FIFO. Read the level of the transmit FIFO using the I2S_DMACH0.tx_lvl field.

12.8 Direct Memory Access

The I²S supports DMA for both transmit and receive; separate DMA channels can be connected to the receive and transmit FIFOs. The following describe the behavior of the receive and transmit DMA requests.

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

12.9 Block Operation

After exiting a power-on reset, the IP is disabled by default. It must be enabled and configured by software to establish the I²S serial communication. A typical software sequence is shown below.

1. Set *GCR_PCLKDIS1*.*i2s* to 0 to enable the I²S peripheral clock source shown in *Table 12-1*.
2. Disable the I²S clock by setting *I2S_CTRL1CH0*.*en* to 0.
3. Set *I2S_CTRL0CH0*.*rst* to 1 to reset the I²S configuration.
4. Set *I2S_CTRL0CH0*.*flush* to 1 to flush the FIFO buffers.
5. Configure the *I2S_CTRL0CH0*.*ch_mode* to select master or slave configuration.
    a. For master mode, configure the baud rate by programming the *I2S_CTRL1CH0*.*clkdiv* field to achieve the required bit rate, set the *I2S_CTRL1CH0*.*smp_size* field to the desired sample size of the data, and the *I2S_CTRL1CH0*.*adjst* field if the Sample Size is smaller than the number of bits per word.
6. Configure the threshold of the receive FIFO by programming the *I2S_CTRL0CH0*.*rx_thd_val*. The threshold of the transmit FIFO is a fixed value, which is half of the transmit FIFO depth.
7. If desired, configure DMA operation, see section *Direct Memory Access* for details.
8. Enable interrupt functionality by configuring the *I2S_INTEN* register if desired.
9. Program the *clkdiv* bits in *I2S_CTRL1CH0* register for the new bit clock frequency.
10. For master operation, load data in the transmit FIFO for transmit.
11. Re-enable the bit clock by setting *I2S_CTRL1CH0*.*en* to 1.

12.10 I²S Registers

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

Table 12-9: I²S Register Summary

Offset	Register Name	Description
[0x0000]	*I2S_CTRL0CH0*	I²S Global Mode Control 0 Register
[0x0010]	*I2S_CTRL1CH0*	I²S Master Mode Configuration Register
[0x0030]	*I2S_DMACH0*	I²S DMA Control Channel Register
[0x0040]	*I2S_FIFOCH0*	I²S FIFO Register
[0x0050]	*I2S_INTFL*	I²S Interrupt Status Register
[0x0054]	*I2S_INTEN*	I²S Interrupt Enable Register

12.10.1 Register Details

Table 12-10: I²S Control 0 Register

I²S Control 0 Register			**I2S_CTRL0CH0**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	rx_thd_val	R/W	0	**Receive FIFO Interrupt Threshold** This field specifies the level of the receive FIFO for the threshold interrupt generation. Values of 0 or greater than the RX_FIFO_DEPTH are ignored.
23:21	-	RO	0	**Reserved**
20	fifo_lsb	R/W	0	**FIFO Bit Field Control** Only used if the FIFO size is larger than the sample size and *I2S_CTRL0CH0.align* = 0. For transmit, the LSB part is sent from the FIFO. For receive, store the LSB part in the FIFO without sign extension. 0: Disabled. 1: Enabled.
19	rst	R/W1O	0	**Reset** Write 1 to reset the I²S peripheral. Hardware automatically clears this field to 0 when the reset is complete. 0: Reset not in process. 1: Reset peripheral.
18	flush	R/W1O	0	**FIFO Flush** Write 1 to start a flush of the receive and transmit FIFOs. Hardware automatically clears this field when the operation is complete. 0: Flush complete or not in process. 1: Flush receive and transmit FIFOs.
17	rx_en	R/W	0	**Receive Enable** Enable receive mode for the I²S peripheral. 0: Disabled. 1: Enabled.
16	tx_en	R/W	0	**Transmit Enable** Enable transmit mode for the I²S peripheral. 0: Disabled. 1: Enabled.
15:14	wsize	R/W	0x3	**Data Size When Reading/Writing FIFO** Set this field to the desired width for data writes and reads from the FIFO. 0: Byte. 1: Half-word (16 bits). 2-3: Word (32 bits).

I²S Control 0 Register				I2S_CTRL0CH0 [0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
13:12	stereo	R/W	0	**I²S Mode** Select the mode for the I²S to stereo, mono left channel only or mono right channel only. 0-1: Stereo. 2: Mono left channel. 3: Mono right channel.
11	-	RO	0	**Reserved**
10	align	R/W	0	**FIFO Data Alignment** Set this field to control the alignment of the data in the FIFOs. This field is only used if the FIFO data width, *I2S_CTRL0CH0.wsize*, is not equal to the bits per word field. 0: MSB. 1: LSB.
9	msb_loc	R/W	0	**First Bit Location Sampling** This field controls when the first bit is transmitted/received in relation to the LRCLK. By default, the first bit is transmitted/received on SDO/SDI on the second complete LRCLK cycle. Set this field to 1 to transmit/receive the first bit of data on the first complete LRCLK cycle. 0: Second complete LRCLK cycle is the first bit of the data. 1: First complete LRCLK cycle is the first bit of the data.
8	ws_pol	R/W	0	**LRCLK Polarity Select** This field determines the polarity of the LRCLK signal associated with the left channel data. Set this field to 1 to associate the left channel with the LRCLK high state. The default setting is the standard I²S association. 0: LRCLK low for left channel. 1: LRCLK high for left channel.
7:6	ch_mode	R/W	0	**Mode** Set this field to indicate master or slave I²S operation. When using master mode, the ERFO must be used to generate the LRCLK/BCLK signals. 0: Master mode, internal generation of LRCLK/BCLK using the ERFO. 1-2: Reserved. 3: Slave mode, external generation of LRCLK/BCLK.
5:2	-	DNM	0	**Reserved, Do Not Modify** Do not modify this field.
1	lsb_first	R/W	0	**LSB First** Setting this field to 1 indicates the least significant bit of the data is transmitted/received first on the SDI/SDO pins. The default setting, 0, indicates the most significant bit of the data is received first. 0: Disabled. 1: Enabled.
0	-	RO	0	**Reserved**

Table 12-11: I²S Master Mode Configuration Register

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

Table 12-12: I²S DMA Control Register

I²S DMA Control				**I2S_DMACH0**	**[0x0030]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:24	rx_lvl	RO	0	**Receive FIFO Level** This field is the number of data words in the receive FIFO.	
23:16	tx_lvl	RO	0	**Transmit FIFO Level** This field is the number of data words in the transmit FIFO.	
15	dma_rx_en	R/W	0	**DMA Receive Channel Enable** 0: Disabled. 1: Enabled.	
14:8	dma_rx_thd_val	R/W	0	**DMA Receive FIFO Event Threshold** If the receive FIFO level is greater than this value, then the receive FIFO DMA interface sends a signal to the system DMA indicating the receive FIFO has characters to transfer to memory.	

I²S DMA Control				I2S_DMACH0	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**	
7	dma_tx_en	R/W	0	**DMA Transmit Channel Enable** 0: Disabled. 1: Enabled.	
6:0	dma_tx_thd_val	RO	0	**DMA Transmit FIFO Event Threshold** If the transmit FIFO level is less than this value, then the transmit FIFO DMA interface sends a signal to system DMA indicating the transmit FIFO is ready to receive data from memory.	

Table 12-13: I²S FIFO Register

I²S FIFO Register				I2S_FIFOCH0	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	data	R/W	0	**I²S FIFO** Writing to this field loads the next character into the transmit FIFO and increments the *I2S_DMACH0.tx_lvl*. Writes are ignored if the transmit FIFO is full. Reads of this field return the next character available from the receive FIFO and decrements the *I2S_DMACH0.rx_lvl*. The value 0 is returned if *I2S_DMACH0.rx_lvl* = 0.	

Table 12-14: I²S Interrupt Flag Register

I²S Interrupt Flag				I2S_INTFL	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:4	-	DNM	0	**Reserved, Do Not Modify**	
3	tx_he_ch0	W1C	0	**Transmit FIFO Half-Empty Event Interrupt Flag** If this field is set to 1 the event has occurred. Write 1 to clear. 0: No event. 1: Event occurred.	
2	tx_ob_ch0	W1C	0	**Transmit FIFO One Entry Remaining Event Interrupt Flag** If this field is set to 1 the event has occurred. Write 1 to clear. 0: No event. 1: Event occurred.	
1	rx_thd_ch0	W1C	0	**Receive FIFO Threshold Event Interrupt Flag** If this field is set to 1 the event has occurred. Write 1 to clear. 0: No event. 1: Event occurred.	
0	rx_ov_ch0	W1C	0	**Receive FIFO Overrun Event Interrupt Flag** If this field is set to 1 the event has occurred. Write 1 to clear. 0: No event. 1: Event occurred.	

Table 12-15: I²S Interrupt Enable Register

I²S Interrupt Enable				I2S_INTEN	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:4	-	DNM	0	**Reserved, Do Not Modify**	
3	tx_he_ch0	R/W	0	**Transmit FIFO Half-Empty Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.	

I^{2}S Interrupt Enable				I2S_INTEN	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**	
2	tx_ob_ch0	R/W	0	**Transmit FIFO One Entry Remaining Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.	
1	rx_thd_ch0	R/W	0	**Receive FIFO Threshold Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.	
0	rx_ov_ch0	R/W	0	**Receive FIFO Overrun Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.	

13. Serial Peripheral Interface (SPI)

The SPI is a highly configurable, flexible, and efficient synchronous interface between multiple SPI devices on a single bus. The SPI bus uses a single clock signal, single or dual data lines, and one or more slave select lines for communication with external SPI devices.

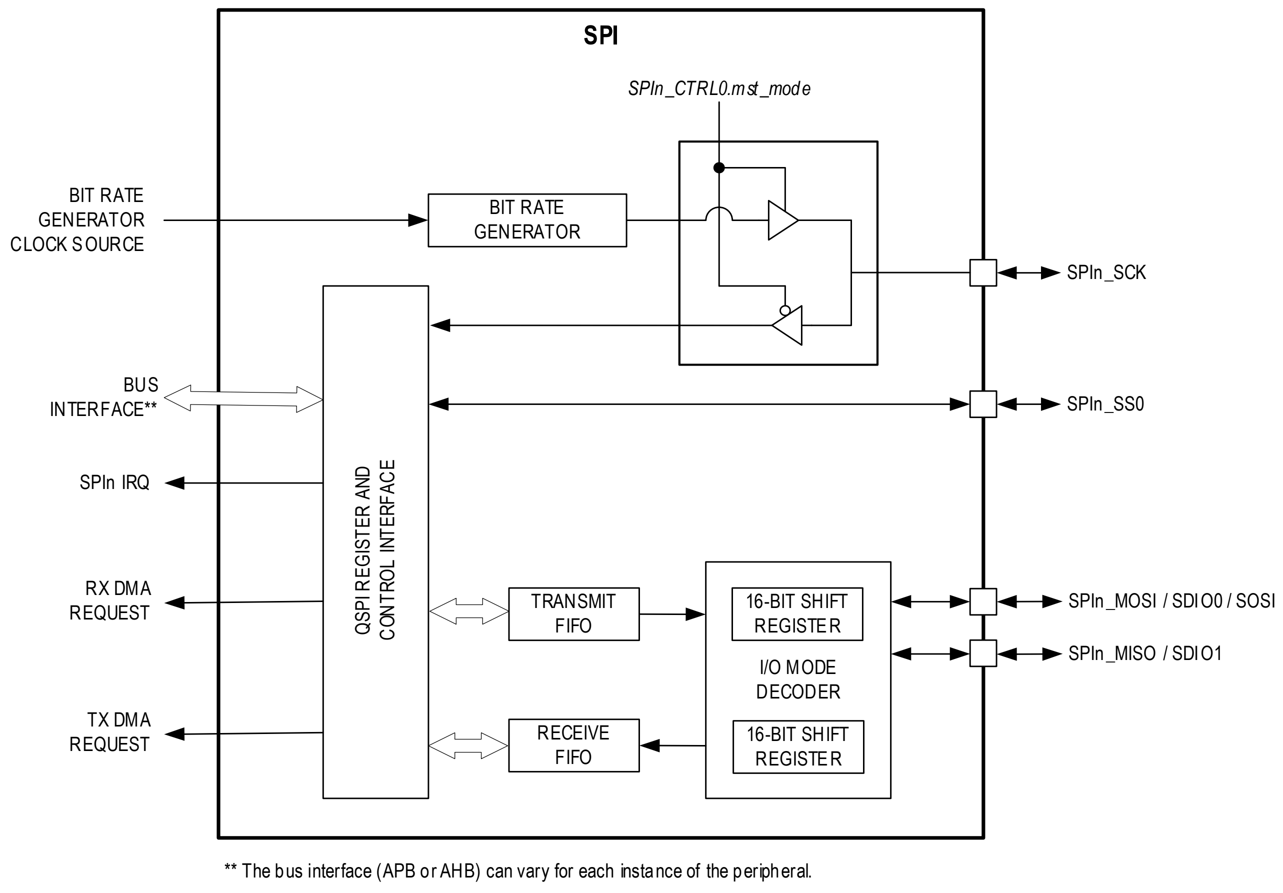
The provided SPI ports support full-duplex, bidirectional I/O, and each SPI includes a bit rate generator (BRG) for generating the clock signal when operating in master mode. Each SPI port operates independently and requires minimal processor overhead. All instances of the SPI peripheral support both 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 mapped on the APB bus.
  - Up to $\frac{f_{HCLK}}{2}$ for instances mapped on the AHB bus.
  - Programmable SCK duty cycle timing.
- Full-duplex, synchronous communication of 2 to 16-bit characters:
  - 1-bit and 9-bit characters are not supported.
  - 2-bit and 10-bit characters do not support maximum clock speed. *SPIn_CLKCTRL.clkdiv* must be > 0.
- Three-wire and four-wire SPI operation for single-bit communication
- Single and dual I/O.
- Byte-wide transmit and receive FIFOs with 32-byte depth:
  - For character sizes greater than 8, each character uses 2 entries per character resulting in 16 entries for the transmit and receive FIFO.
- Transmit and receive DMA support.
- SPI modes 0, 1, 2, 3.
- Configurable slave select lines:
  - Programmable slave select level.
- Programmable slave select timing with respect to the SCK starting edge and ending edge.
- Multi-master mode fault detection.

Figure 13-1 shows the block diagram of the peripheral. See *Table 13-1* for the peripheral-specific bus assignment and bit rate generator clock source.

Figure 13-1: SPI Block Diagram



13.1 Instances

The following instances of the peripheral are provided. SPI0 is only available internally for communication with the AFE. See the *Analog Front-End (AFE)* chapter for the SPI0 internal signal mapping.

Table 13-1: MAX32675 SPI Instances

Name	Formats				Bus Assignment	Bit Rate Generator Clock Source Frequency	Slave Select Signals
	3-Wire	4-Wire	Dual	Quad			68-TQFN
SPI0	No	Yes	No	No	APB	f_{PCLK}	N/A
SPI1	Yes	Yes	No	No	APB	f_{PCLK}	1

Note: For SPI1 signal mapping, refer to the device data sheet's alternate function table.

13.2 SPI Formats

13.2.1 Four-Wire SPI

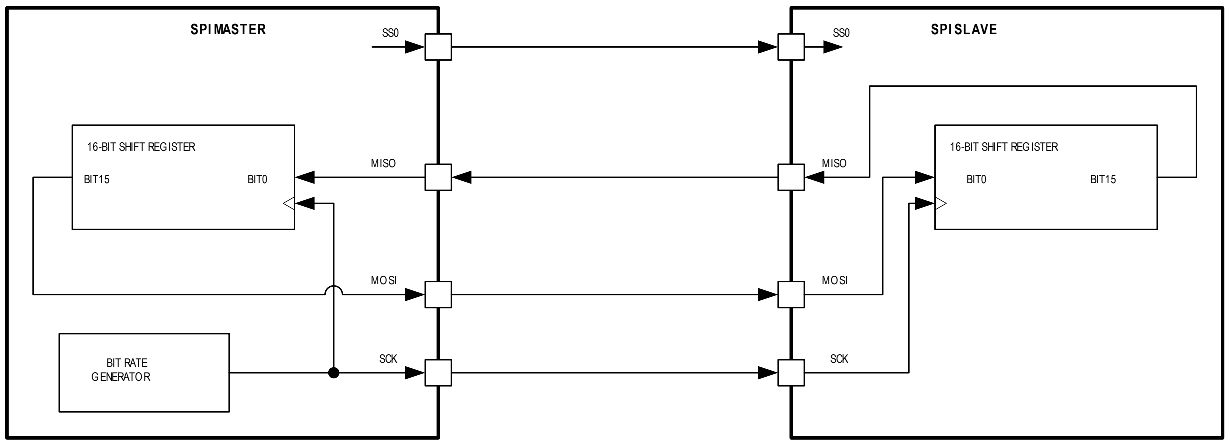
SPI devices operate as either a master or a slave. In four-wire SPI, four signals are required for communication, as shown in *Table 13-2*.

Table 13-2: Four-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The master generates the serial clock signal, 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 before communication. Peripherals can have multiple slave select outputs to communicate with one or more external slave devices. In slave mode, SPIn_SS0 is a dedicated input that indicates an external master must start communication. Other slave select signals into the peripheral are ignored in slave mode.

In a typical SPI network, the master device selects the slave device using the slave select output. The master starts the communication by selecting the slave device by asserting the slave select output. The master then starts the SPI clock through the SCK output pin. When a slave device's slave select pin is deasserted, the device must put the SPI pins in tri-state mode.

Figure 13-2: Four-Wire SPI Connection Diagram



13.2.2 Three-Wire SPI

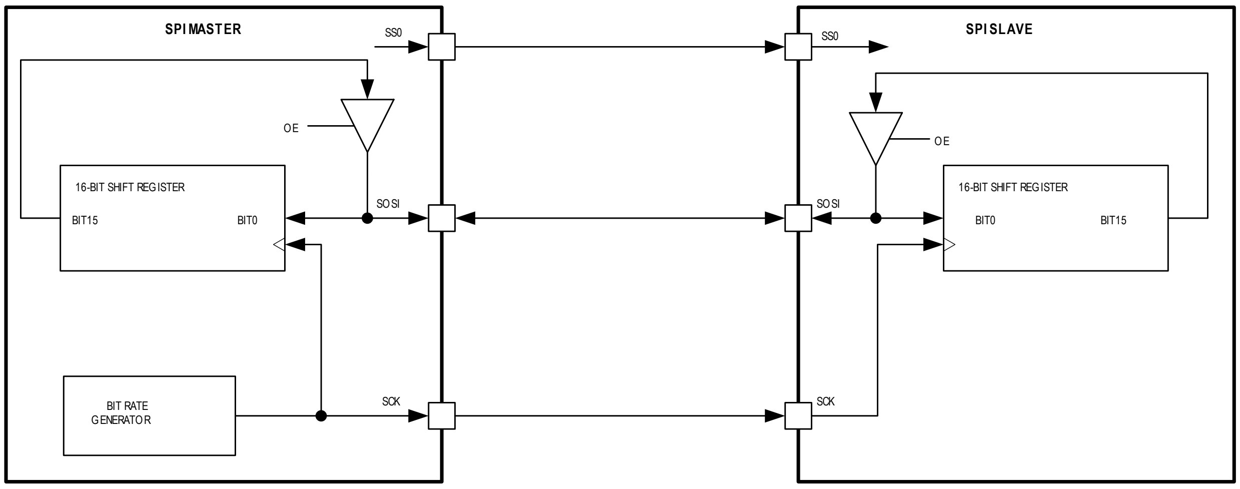
The signals in three-wire SPI operation are shown in *Table 13-3*. The MOSI signal is used as a bidirectional, half-duplex I/O referred to as slave input slave output (SISO). Three-wire SPI also uses a serial clock signal generated by the master and a slave select pin controlled by the master.

Table 13-3: Three-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The master generates the serial clock signal, an output from the master, and an input to the slave.
MOSI	Master Output Slave Input	This signal 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 receive data from the slave by the master.
SS	Slave Select	In master mode, this signal is an output used to select a slave device before communication. In slave mode, SPIn_SS0 is a dedicated input that indicates an external master must start communication. Other slave select signals into the peripheral are ignored in slave mode.

A three-wire SPI network is shown in *Figure 13-3*. The master device selects the slave device using the slave select output. The communication starts with the master asserting the slave select line and then starting the clock (SCK). In three-wire SPI communication, the master and slave must know the data's intended direction to prevent bus contention. For a write, the master drives the data out of the SISO pin. The master must release the SISO line for a read 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 13-3: Three-Wire SPI Master to Slave Connection



13.3 Pin Configuration

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

13.3.1 SPI Alternate Function Mapping

Pin selection and configuration are required to use the SPI port. The following information applies to SPI master and slave operation in three-wire, four-wire, and dual mode communications. Determine the pins required for the SPI type and mode in the application and configure the required GPIO as described in the following sections. Refer to the device data sheet for pin availability for a specific package.

When the SPI port is disabled, *SPIn_CTRL0.en* = 0, the GPIO pins enabled for SPI alternate function are placed in high-impedance input mode.

13.3.2 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 legacy reasons.

Note: Select the pins mapped to the SPI external device in the design and modify the setup accordingly. There is no restriction on which alternate function is used for a specific SPI pin, and each SPI pin can be used independently from the other pins chosen. However, it is recommended that only one set of GPIO port pins are used for any network.

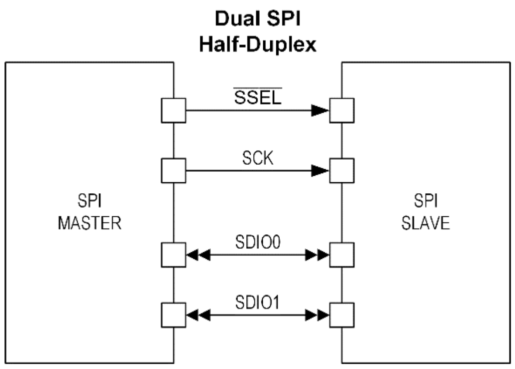
13.3.3 Three-Wire Format Configuration

Three-wire SPI uses SCK, MOSI, and one or more slave select pins for an SPI transaction. Three-wire SPI configuration is identical to the four-wire configuration, except SPIn_MISO does not need to be set up for the SPI alternate function. The direction of communication in three-wire SPI mode is controlled by the SPI transmit and receive FIFO enables. Enabling the receive FIFO and disabling the transmit FIFO indicates a read transaction. Enabling the transmit FIFO and disabling the receive FIFO indicates a write transaction. It is an illegal condition to enable both the transmit and receive FIFOs in three-wire SPI operation.

13.3.4 Dual Mode Format Configuration

In dual mode SPI, two I/O pins transmit 2 bits of data per SCK clock cycle. The communication is half-duplex, and the direction of the data transmission must be known by both the master and slave for a given transaction. Dual mode SPI uses SCK, SDIO0, SDIO1, and one or more slave select lines, as shown in *Figure 13-4*. The configuration of the GPIO pins for dual mode SPI is identical to four-wire SPI. The mode is controlled by setting *SPIn_CTRL2.data_width* to 1, indicating to the SPI hardware to use SDIO0 and SDIO1 for half-duplex communication rather than full-duplex communication.

Figure 13-4: Dual Mode SPI Connection Diagram



13.4 SPI Clock Configuration

13.4.1 Serial Clock

The SCK signal synchronizes data movement in and out of the device. The master drives SCK as an output to the slave's SCK pin. When SPI is set to master mode, the SPI bit rate generator creates the serial clock and outputs it on the configured SPIn_SCK pin. When SPI is configured for slave operation, the SPIn_SCK pin is input from the external master, and the SPI hardware synchronizes communications using the SCK input. Operating as a slave, if a SPI slave select input is not asserted, the SPI 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. The SPI peripheral supports four combinations of SCK phase and polarity referred to as SPI modes 0, 1, 2, and 3. Clock polarity (*SPIn_CTRL2*.*clkpol*) selects an active-low or active-high clock and does not affect the transfer format. The clock phase (*SPIn_CTRL2*.*clkpha*) selects one of two different transfer formats.

The clock phase and polarity must be identical for the SPI master and slave for proper data transmission. The master always places data on the MOSI line a half-cycle before the SCK edge for the slave to latch the data. See section *Clock Phase and Polarity Control* for additional details.

13.4.2 SPI Peripheral Clock

The System Peripheral Clock, PCLK, drives the SPI peripheral clock. The SPI provides an internal clock, SPIn_CLK, used within the SPI peripheral for the base clock to control the module and generate the SCK clock in master mode. Set the SPI internal clock using the field *SPIn_CLKCTRL*.*clkdiv* as shown in *Equation 13-1*. Valid settings for *SPIn_CLKCTRL*.*clkdiv* are 0 to 8, allowing a divisor of 1 to 256.

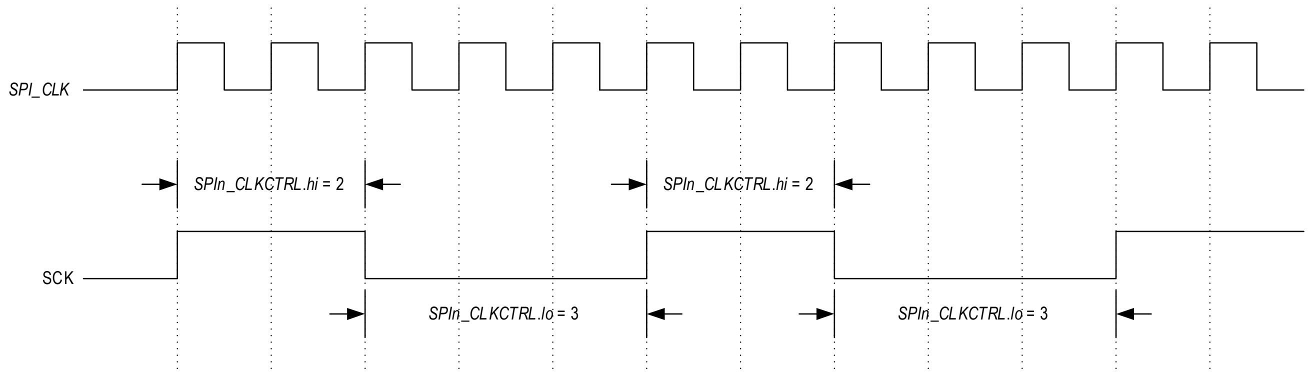
Equation 13-1: SPI Peripheral Clock

$$f_{SPI\_CLK} = \frac{f_{PCLK}}{2^{scale}}$$

13.4.3 Master Mode Serial Clock Generation

In master and multi-master mode, the SCK clock is generated by the master. The SPI provides control for both the high time and low time of the SCK clock. This control allows setting the high and low times for the SCK to duty cycles other than 50% if required. The SCK clock uses the SPI peripheral clock as a base value, and the high and low values are a count of the number of f_{SPI_CLK} clocks. *Figure 13-5* visually represents the use of the *SPIn_CLKCTRL*.*hi* and *SPIn_CLKCTRL*.*lo* fields for a non-50% duty cycle serial clock generation. See *Equation 13-2* and *Equation 13-3* for calculating the SCK high and low time from the *SPIn_CLKCTRL*.*hi* and *SPIn_CLKCTRL*.*lo* field values.

Figure 13-5: SCK Clock Rate Control



Equation 13-2: SCK High Time

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

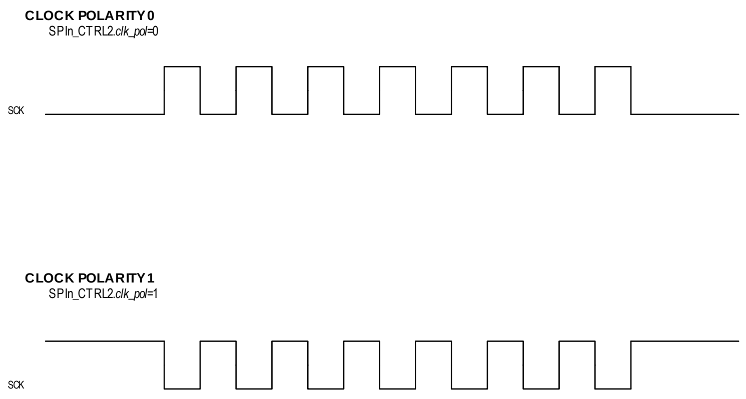
Equation 13-3: SCK Low Time

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

13.4.4 Clock Phase and Polarity Control

SPI supports four combinations of clock and phase polarity, as shown in *Table 13-4*. Clock polarity is controlled using the *SPIn_CTRL2.clkpol* bit, which determines if the clock is active-high or active-low, as shown in *Figure 13-6*. The clock's polarity does not affect the transfer format for SPI. The clock's phase determines when the data must be stable for sampling. Setting the clock phase to 0, *SPIn_CTRL2.clkpha* = 0, dictates the SPI data is sampled on the initial SPI clock edge regardless of clock polarity. Phase 1, *SPIn_CTRL2.clkpha* = 1, results in the data sample occurring on the clock's second edge regardless of clock polarity.

Figure 13-6: 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 13-4: SPI Modes Clock Phase and Polarity Operation

SPI Mode	SPIn_CTRL2 .clkpha	SPIn_CTRL2 .clkpol	SCK Transmit Edge	SCK Receive Edge	SCK Idle State
0	0	0	Falling	Rising	Low
1	0	1	Rising	Falling	High
2	1	0	Rising	Falling	Low
3	1	1	Falling	Rising	High

13.4.5 SPI FIFOs

The transmit FIFO hardware is 32 bytes deep. The write data width can be 8-, 16-, or 32-bits wide. A 16-bit write queues a 16-bit word to the FIFO hardware. A 32-bit write queues two 16-bit words to the FIFO hardware with the least significant word dequeued first. Bytes must be written to two consecutive byte addresses, with the odd byte as the most significant byte and the even byte as the least significant byte. The FIFO logic waits for the odd and even bytes to be written to this register before dequeuing the 16-bit result to the FIFO.

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

13.4.6 SPI Interrupts and Wake-Ups

The SPI supports multiple interrupt sources. Status flags for each interrupt are set regardless of the state of the interrupt enable bit for that event. The event happens once when the condition is satisfied. The software must clear the status flag by writing a 1 to the interrupt flag.

The following FIFO interrupts are supported:

- Transmit FIFO Empty
- Transmit FIFO Threshold
- Receive FIFO Full
- Receive FIFO Threshold
- Transmit FIFO Underrun
    - 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

SPI supports interrupts for the internal state of the SPI as well as external signals. The following transmission interrupts are supported:

- SSn Asserted or Deasserted
- SPI Transaction Complete
    - Master Mode Only
- Slave Mode Transaction Aborted
- Multi-Master Fault

The SPI port can wake up the microcontroller from low-power modes when the wake event is enabled. SPI events that can wake the microcontroller are:

- Receive FIFO Full
- Transmit FIFO Empty
- Receive FIFO Threshold
- Transmit FIFO Threshold

13.5 SPI Registers

See *Table 3-2* for this peripheral/module's base address. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 13-5*. Register names for a specific instance are defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 13-5: SPI Register Summary

Offset	Register Name	Access	Description
[0x0000]	*SPIn_FIFO32*	R/W	*SPI FIFO Data Register*
[0x0004]	*SPIn_CTRL0*	R/W	*SPI Master Signals Control Register*
[0x0008]	*SPIn_CTRL1*	R/W	*SPI Transmit Packet Size Register*
[0x000C]	*SPIn_CTRL2*	R/W	*SPI Static Configuration Register*
[0x0010]	*SPIn_SSTIME*	R/W	*SPI Slave Select Timing Register*
[0x0014]	*SPIn_CLKCTRL*	R/W	*SPI Master Clock Control Register*
[0x001C]	*SPIn_DMA*	R/W	*SPI DMA Control Register*
[0x0020]	*SPIn_INTFL*	R/W1C	*SPI Interrupt Flag Register*
[0x0024]	*SPIn_INTEN*	R/W	*SPI Interrupt Enable Register*
[0x0028]	*SPIn_WKFL*	R/W1C	*SPI Wake-Up Flags Register*
[0x002C]	*SPIn_WKEN*	R/W	*SPI Wake-Up Enable Register*
[0x0030]	*SPIn_STAT*	RO	*SPI Status Register*

13.5.1 Register Details

Table 13-6: SPI FIFO Data Register

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

Table 13-7: SPI Control 0 Register

SPI Control 0 Register			**SPIn_CTRL0**	**[0x0004]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**

SPI Control 0 Register			**SPIn_CTRL0**	**[0x0004]**
Bits	**Name**	**Access**	**Reset**	**Description**
19:16	ss_active	R/W	0	**Master Slave Select** The SPI includes up to four slave select lines for each port. This field selects which slave select pin is active when the next SPI transaction is started (*SPIn_CTRL0.start* = 1). One or more slave select pins can be selected for each SPI transaction by setting the bit for each slave select pin. For example, use SPIn_SS0 and SPIn_SS2 by setting this field to 0b0101 or select all slave selects by setting this field to 0b1111. *Note: This field is only used when the SPI is configured for master mode (SPIn_CTRL0.mst_mode = 1).*
15:9	-	RO	0	**Reserved**
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 the 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	-	RO	0	**Reserved**
5	start	R/W1O	0	**Master Start Data Transmission** Set this field to 1 to start an SPI master mode transaction. 0: No master mode transaction active. 1: Master initiates data transmission. Ensure that all pending transactions are complete before setting this field to 1. *Note: This field is only used when the SPI is configured for master mode (SPIn_CTRL0.mst_mode = 1).*
4	ss_io	R/W	0	**Master Slave Select Signal Direction** Set this field to 1 to use the slave select signal as an input. Setting this field to 0 sets the slave select signal as an output. 0: Slave Select is an output. 1: Slave Select is an input. *Note: This field is only used when the SPI is configured for master mode (SPIn_CTRL0.mst_mode = 1).*
3:2	-	RO	0	**Reserved**
1	mst_mode	R/W	0	**SPI Master Mode Enable** This field selects between slave and master mode operation for the SPI port. Write this field to 0 to operate as an SPI slave. Set this field to 1 to configure the port as an SPI master. 0: Slave mode SPI operation. 1: Master mode SPI operation.

SPI Control 0 Register				SPIn_CTRL0 | [0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
0	en	R/W	0	**SPI Enable/Disable** This field enables and disables the SPI port. Disable the SPI port by setting this field to 0. Disabling the SPI port does not affect the SPI FIFOs or register settings. Access to SPI registers is always available. 0: SPI port is disabled. 1: SPI port is enabled.

Table 13-8: SPI Transmit Packet Size Register

SPI Transmit Packet Size Register				SPIn_CTRL1 | [0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	rx_num_char	R/W	0	**Number of Receive Characters** This field sets the number of characters to receive in receive FIFO. *Note: If the SPI port is set to four-wire mode, this field is ignored, and the SPIn_CTRL1.tx_num_char field is used for both the number of characters to receive and transmit.*
15:0	tx_num_char	R/W	0	**Number of Transmit Characters** This field sets the number of characters to transmit from transmit FIFO. *Note: If the SPI port is set to four-wire mode, this field is used to receive and transmit the number of characters.*

Table 13-9: SPI Control 2 Register

SPI Control 2 Register				SPIn_CTRL2 | [0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:16	ss_pol	R/W	0	**Slave Select Polarity** Controls the polarity of each SS signal where each bit position corresponds to a SS signal. SPIn_SS0 is controlled with bit position 0, and SPIn_SS2 is controlled with bit position 2. For each bit position, 0: SS is active low. 1: SS is active high.
15	three_wire	R/W	0	**Three-Wire SPI Enable** Set this field to 1 to enable three-wire SPI communication. Set this field to 0 for four-wire full-duplex SPI communication. 0: Four-wire full-duplex mode enabled. 1: Three-wire mode enabled. *Note: This field is ignored for dual SPI, SPIn_CTRL2.data_width = 1.*
14	-	RO	0	**Reserved**

SPI Control 2 Register			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
13:12	data_width	R/W	0b00	**SPI Data Width** This field controls the number of data lines used for SPI communications. Three-wire SPI, *data_width* = 0: Set this field to 0, indicating SPIn_MOSI is used for half-duplex communication. Four-wire full-duplex SPI operation, *data_width* = 0: Set this field to 0, indicating SPIn_MOSI and SPIn_MISO are used for the SPI data output and input, respectively. Dual SPI, *data_width* = 1, uses SPIn_SDIO0 and SPIn_SDIO1 for half-duplex communication. 0: 1-bit per SCK cycle (Three-wire half-duplex SPI and four-wire full-duplex SPI). 1: 2-bits per SCK cycle (Dual SPI). 2: Reserved. 3: Reserved. *Note: When this field is set to 0, use SPIn_CTRL2.three_wire to select either three-Wire SPI or four-Wire SPI operation.*
11:8	numbits	R/W	0	**Number of Bits per Character** Set this field to the number of bits per character for the SPI transaction. Setting this field to 0 indicates a character size of 16. 0: 16-bits per character. 1: 1-bit per character. 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. SPIn_CLKCTRL.clkdiv must be > 0.* *Note: For dual mode SPI, the character size should be divisible by the number of bits per SCK cycle.*
7:2	-	RO	0	**Reserved**
1	clkpol	R/W	0	**Clock Polarity** This field controls the SCK polarity. The default clock polarity is for SPI mode 0 and mode 1 operation and is active high. Invert the SCK polarity for SPI mode 2 and mode 3 operation. 0: Standard SCK for use in SPI mode 0 and mode 1. 1: Inverted SCK for use in SPI mode 2 and mode 3.
0	clkpha	R/W	0	**Clock Phase** 0: Data sampled on clock rising edge. Use when in SPI mode 0 and mode 2. 1: Data sampled on clock falling edge. Use when in SPI mode 1 and mode 3.

Table 13-10: SPI Slave Select Timing Register

SPI Slave Select Timing Register			SPIn_SSTIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**

SPI Slave Select Timing Register				SPIn_SSTIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**	
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 following transaction (slave select active). 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255 *Note: The SPIn_SSTIME register bit settings only apply when SPI is operating in master mode (SPIn_CTRL0.mst_mode = 1).*	
15:8	post	R/W	0	**Slave Select Hold Post Last SCK** This field sets the number of system clock cycles that SS remains active after the last SCK edge. 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255 *Note: The SPIn_SSTIME register bit settings only apply when SPI is operating in master mode (SPIn_CTRL0.mst_mode = 1).*	
7:0	pre	R/W	0	**Slave Select Delay to First SCK** This field sets the number of system clock cycles the slave select is held active before the first SCK edge. 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255 *Note: The SPIn_SSTIME register bit settings only apply when SPI is operating in master mode (SPIn_CTRL0.mst_mode = 1).*	

Table 13-11: SPI Master Clock Control Registers

SPI Master Clock Control Register				SPIn_CLKCTRL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:20	-	RO	0	**Reserved**	

SPI Master Clock Control Register				SPIn_CLKCTRL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**	
19:16	clkdiv	R/W	0	**SPI Peripheral Clock Scale** Scales the SPI input clock (PCLK) by 2^{scale} to generate the SPI peripheral clock. $f_{SPInCLK} = \frac{f_{SPIn\_INPUT\_CLK}}{2^{scale}}$ Valid values for scale are 0 to 8 inclusive. Values greater than 8 are reserved. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*	
15:8	hi	R/W	0	**SCK Hi Clock Cycles Control** 0: Hi duty cycle control disabled. Only valid if SPIn_CLKCTRL.*clkdiv* = 0. 1 to 15: The number of SPI peripheral clocks, $f_{SPInCLK}$, that SCK is high. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*	
7:0	lo	R/W	0	**SCK Low Clock Cycles Control** This field controls the SCK low clock time and controls the overall SCK duty cycle in combination with the SPIn_CLKCTRL.*hi* field. 0: Low duty cycle control disabled. Setting this field to 0 is only valid if SPIn_CLKCTRL.*clkdiv* = 0. 1 to 15: The number of SPI peripheral clocks, $f_{SPInCLK}$, that the SCK signal is low. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*	

Table 13-12: SPI DMA Control Registers

SPI DMA Control Register				SPIn_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**	
31	dma_rx_en	R/W	0	**Receive DMA Enable** 0: Disabled. Any pending DMA requests are cleared. 1: Enabled.	
30	-	RO	0	**Reserved**	
29:24	rx_lvl	R	0	**Number of Bytes in the Receive FIFO** Read returns the number of bytes currently in the receive FIFO.	
23	rx_flush	W	-	**Clear the Receive FIFO** 1: Clear the receive FIFO and any pending receive FIFO flags in the SPIn_INTFL register. Write to this field only when the receive FIFO is inactive. *Note: Writing a 0 to this field has no effect.*	
22	rx_fifo_en	R/W	0	**Receive FIFO Enabled** 0: Disabled. 1: Enabled.	
21	-	RO	0	**Reserved**	

SPI DMA Control Register		SPIn_DMA		[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
20:16	rx_thd_val	R/W	0x00	**Receive FIFO Threshold Level** Set this value to the desired receive FIFO threshold level. When the receive FIFO level crosses above this setting, a DMA request is triggered if enabled by setting *SPIn_DMA.dma_rx_en*, and *SPIn_INTFL.rx_thd* becomes set. Valid values are 0 to 30. *Note: 31 is an invalid setting and reserved.*
15	dma_tx_en	R/W	0	**Transmit DMA Enable** 0: Disabled. Any pending DMA requests are cleared. 1: Transmit DMA is enabled.
14	-	RO	0	**Reserved**
13:8	tx_lvl	RO	0	**Number of Bytes in the Transmit FIFO** Read this field to determine the number of bytes currently in the transmit FIFO.
7	tx_flush	R/W	0	**Transmit FIFO Clear** Set this bit to clear the transmit FIFO and all transmit FIFO flags in the *SPIn_INTFL* register. *Note: The transmit FIFO should be disabled (SPIn_DMA.tx_fifo_en = 0) before setting this field.* *Note: Setting this field to 0 has no effect.*
6	tx_fifo_en	R/W	0	**Transmit FIFO Enabled** 0: Disabled. 1: Enabled.
5	-	RO	0	**Reserved**
4:0	tx_thd_val	R/W	0x10	**Transmit FIFO Threshold Level** Set this value to the desired transmit FIFO threshold level. When the transmit FIFO count (*SPIn_DMA.tx_lvl*) falls below this value, a DMA request is triggered if enabled by setting *SPIn_DMA.dma_tx_en,* and *SPIn_INTFL.tx_thd* becomes set.

Table 13-13: SPI Interrupt Status Flags Registers

SPI Interrupt Status Flags Register		SPIn_INTFL		[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	rx_un	R/1	0	**Receive FIFO Underrun Flag** Set when a read is attempted from an empty receive FIFO.
14	rx_ov	R/W1C	0	**Receive FIFO Overrun Flag** Set if SPI is in slave mode, and a write to a full receive FIFO is attempted. If the SPI is in master mode, this bit is not set as the SPI stalls the clock until data is read from the receive FIFO.
13	tx_un	R/W1C	0	**Transmit FIFO Underrun Flag** Set if SPI is in slave mode, and a read from empty transmit FIFO is attempted. If SPI is in master mode, this bit is not set as the SPI stalls the clock until data is written to the empty transmit FIFO.
12	tx_ov	R/W1C	0	**Transmit FIFO Overrun Flag** Set when a write is attempted to a full transmit FIFO.

SPI Interrupt Status Flags Register			SPIn_INTFL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
11	mst_done	R/W1C	0	**Master Data Transmission Done Flag** This field is set if the SPI is in master mode and all transactions have been completed. *SPIn_CTRL1.tx_num_char* has been reached.
10	-	RO	0	**Reserved**
9	abort	R/W1C	0	**Slave Mode Transaction Abort Detected Flag** This field is set if the SPI is in slave mode and the SS signal is deasserted before a complete character is received.
8	fault	R/W1C	0	**Multi-Master Fault Flag** This field is set if the SPI is in master mode, multi-master mode is enabled, and a slave select input is asserted. A collision also sets this flag.
7:6	-	RO	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_thd	R/W1C	0	**Receive FIFO Threshold Level Crossed Flag** Set when the receive FIFO exceeds the value in *SPIn_DMA.rx_thd_val*. Cleared once receive FIFO level drops below *SPIn_DMA.rx_thd_val.*
1	tx_em	R/W1C	1	**Transmit FIFO Empty Flag**
0	tx_thd	R/W1C	0	**Transmit FIFO Threshold Level Crossed Flag** Set when the transmit FIFO is less than the value in *SPIn_DMA.tx_thd_val*. Cleared once transmit FIFO level exceeds *SPIn_DMA.tx_thd_val.*

Table 13-14: SPI Interrupt Enable Registers

SPI Interrupt Enable Register			SPIn_INTEN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	rx_un	R/W	0	**Receive FIFO Underrun Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
14	rx_ov	R/W	0	**Receive FIFO Overrun Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
13	tx_un	R/W	0	**Transmit FIFO Underrun Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
12	tx_ov	R/W	0	**Transmit FIFO Overrun Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
11	mst_done	R/W	0	**Master Data Transmission Done Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
10	-	RO	0	**Reserved**

SPI Interrupt Enable Register			SPIn_INTEN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
9	abort	R/W	0	**Slave Mode Abort Detected Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
8	fault	R/W	0	**Multi-Master Fault Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
7:6	-	RO	0	**Reserved**
5	ssd	R/W	0	**Slave Select Deasserted Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
4	ssa	R/W	0	**Slave Select Asserted Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
3	rx_full	R/W	0	**Receive FIFO Full Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
2	rx_thd	R/W	0	**Receive FIFO Threshold Level Crossed Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
1	tx_em	R/W	0	**Transmit FIFO Empty Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.
0	tx_thd	R/W	0	**Transmit FIFO Threshold Level Crossed Interrupt Enable** 0: Interrupt is disabled. 1: Interrupt is enabled.

Table 13-15: SPI Wake-Up Status Flags Registers

SPI Wake-Up Flags Register			SPIn_WKFL	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	rx_full	R/W1C	0	**Wake On Receive FIFO Full Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
2	rx_thd	R/W1C	0	**Wake On Receive FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
1	tx_em	R/W1C	0	**Wake On Transmit FIFO Empty Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
0	tx_thd	R/W1C	0	**Wake On Transmit FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.

Table 13-16: SPI Wake-Up Enable Registers

SPI Wake-Up Enable Register			SPIn_WKEN	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	rx_full	R/W	0	**Wake On Receive FIFO Full Enable** 0: Wake event is disabled. 1: Wake event is enabled.
2	rx_thd	R/W	0	**Wake On Receive FIFO Threshold Level Crossed Enable** 0: Wake event is disabled. 1: Wake event is enabled.
1	tx_em	R/W	0	**Wake On Transmit FIFO Empty Enable** 0: Wake event is disabled. 1: Wake event is enabled.
0	tx_thd	R/W	0	**Wake On Transmit FIFO Threshold Level Crossed Enable** 0: Wake event is disabled. 1: Wake event is enabled.

Table 13-17: SPI Slave Select Timing Registers

SPI Status Register			SPIn_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**
0	busy	R	0	**SPI Active Status** 0: SPI is not active. In master mode, this flag is cleared when the last character is sent. In slave mode, this flag is cleared when the configured slave select input is deasserted. 1: SPI is active. In master mode, this flag is set when a transaction starts. In slave mode, this flag is set when a configured slave select input is asserted. *Note: If this field is set, SPIn_CTRL0, SPIn_CTRL1, SPIn_CTRL2, SPIn_SSTIME, and SPIn_CLKCTRL should not be configured.*

14. Analog Front-End (AFE)

The MAX32675 communicates internally using the SPI0 interface to provide register and control to the AFE peripherals. See the *Serial Peripheral Interface (SPI)* chapter for details of SPI communications and configuration. The AFE enables access to the following peripherals:

- The dual 16-/24-bit Delta-Sigma ADCs with PGA
- The 12-bit DAC
- The HART modem for control and configuration

14.1 Instances

There is one instance of the AFE. SPI0 is used to communicate to the AFE and must be configured as shown in *Table 14-2*.

Table 14-1: MAX32675 AFE Instance

Instance	SPI Interface	SPI Clock
AFE	SPI0	PCLK

14.2 SPI Communication Interface

The AFE requires the internally connected SPI0 instance for communications to each of the AFE peripherals for configuration and command.

The SPI interface must be set to mode 0. Data is strobed in on the SCLK rising edges. The content of the SPI operation consists of a one-byte register address and read/write command followed by a one, two, or three-byte control or data word. Programming is by a variable cycle (dictated by the peripheral's register byte width) SPI instruction framed by a slave select low interval. To abort a command sequence, the rise of the slave select signal must precede the updating rising edge of SCLK. *Table 14-2* shows the internal pins between the microcontroller and the AFE used for the SPI interface.

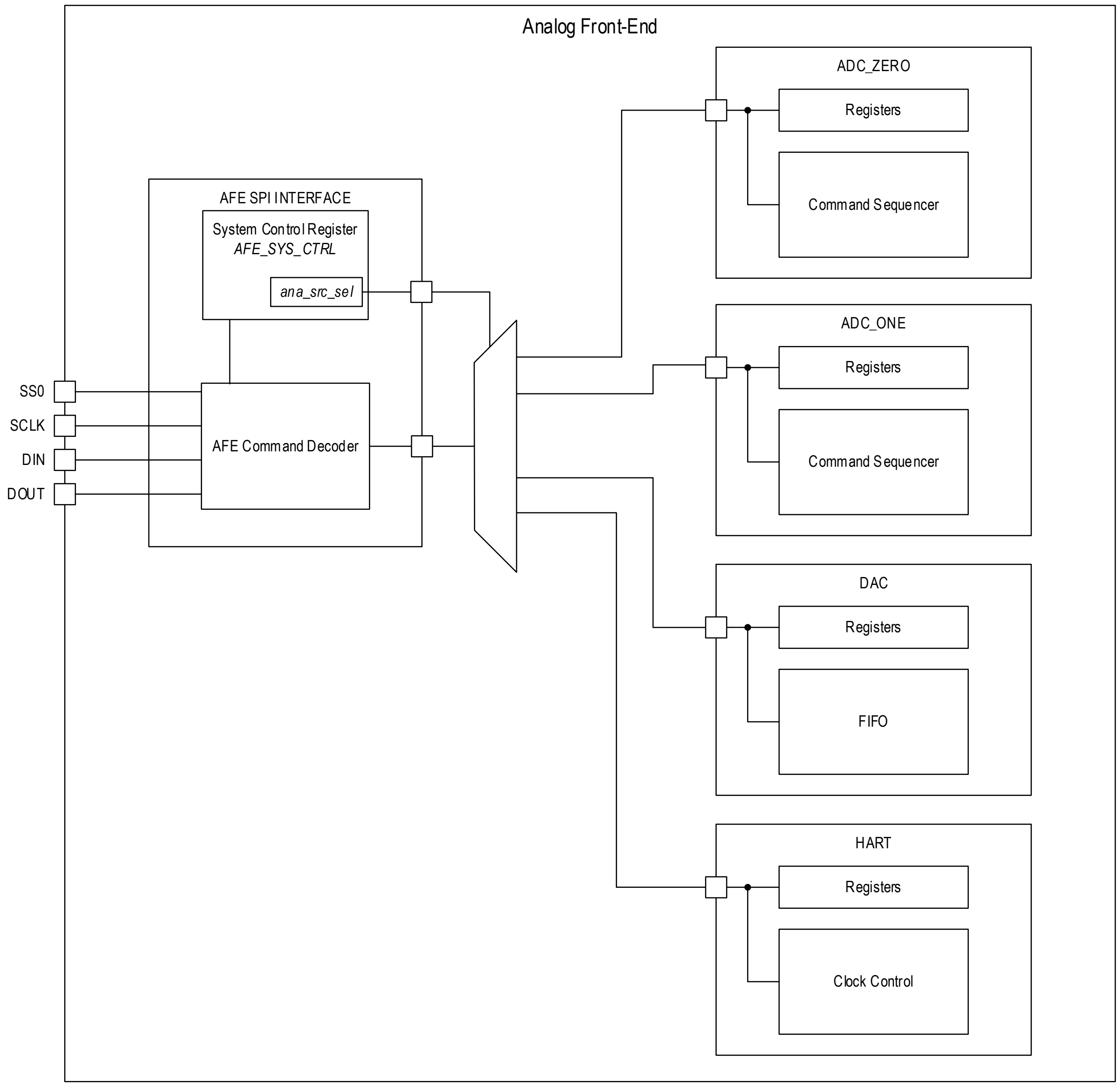
Figure 14-1 shows the AFE functional diagram and interface.

Table 14-2: MAX32675 SPI0 Pins Used for Communication with the AFE

Signal Name	AFE Interface Name	SPI0 Pin	Alternate Function	Direction
SCLK	SCLK	P0.4	AF1	Input from microcontroller
SS	SS0	P0.5	AF1	Input from microcontroller
MOSI	DIN	P0.3	AF1	Input from microcontroller
MISO	DOUT	P0.2	AF1	Output to microcontroller

See the *Serial Peripheral Interface (SPI)* chapter for details on configuring the SPI interface in software.

Figure 14-1: AFE Functional Diagram and Interface



14.2.1 AFE Peripheral Register Byte Width

Each of the individual AFE peripherals contain a list of independent registers. Each of these registers is either 8-bits, 16-bits, or 24-bits wide. See each of the AFE peripheral's register list tables to determine the byte width of each specific register.

Table 14-3 shows the convention used for the AFE peripherals in this document. This convention applies to the following peripherals and their register sets:

- The AFE
- The *16-/24-Bit Delta-Sigma ADC with PGA*
- The *Digital-to-Analog Converter (DAC)*
- The *HART Modem (HART)*

The address column shows the SPI command address when the peripheral is selected using the *AFE_SYS_CTRL.ana_src_sel* field. The width column shows the bit width of the register. This width can be 8-bits, 16-bits, or 24-bits and corresponds to the number of SPI bytes to read or write for each register.

Note: The AFE_SYS_CTRL register is always at SPI address 0x7A regardless of the AFE peripheral selected.

Table 14-3: AFE Peripheral Register Table Convention

Address	Width	Register Name	Description
<SPI Address>	<Register Width>	<Register Name>	<Register Description>

14.2.2 DOUT/INTB

This output from the AFE serves a dual function. In addition to the serial-data output function, DOUT/INTB also indicates the interrupt condition when SS0 is low. To find the interrupt state, assert SS0 low and sample the INTB/DOUT output. When performing a device readback, the DOUT/INTB signal reflects the interrupt states until the 9th SCLK falling edge, at which point it transitions to the DOUT data.

14.2.3 SPI Transactions

All transactions consist of a read/write bit, register address, and register data (returned or written). All registers are either 8, 16, or 24 bits in length. Program word execution happens on either the 16th, 24th, or 32nd edge, depending on the programmed register word length. Paired SPI register reads and writes are not supported. Registers are read and written MSB first. *Figure 14-2* shows the structure of the SPI communications used between SPI0 and the AFE.

If CRC-5-USB is enabled, *AFE_SYS_CTRL.crc5* = 1, all SPI read transactions are extended by one byte. The last byte read from the AFE during the SPI transaction is the CRC-5-USB byte of the data read bytes.

Note: Enabling CRC-5-USB adds an additional byte to the standard SPI read transaction, extending an 8-bit register read to a two byte SPI read, a 16-bit register read to three byte SPI read and a 24-bit register read to a four byte SPI read.

14.2.3.1 SPI Register Address Byte

The write to the SPI register address byte, shown in SCLK cycles 0 through 7 in *Figure 14-2*, begins any read or write transaction to the AFE or the selected AFE peripheral. *Table 14-4* shows the format of the SPI register address byte. The R/WB bits selects whether the transaction is a read of a write. Setting the R/WB bit to 1 indicates a read transaction and setting R/WB to 0 indicates a write transaction. The REG_ADDR bits select the address of the register to be written or read.

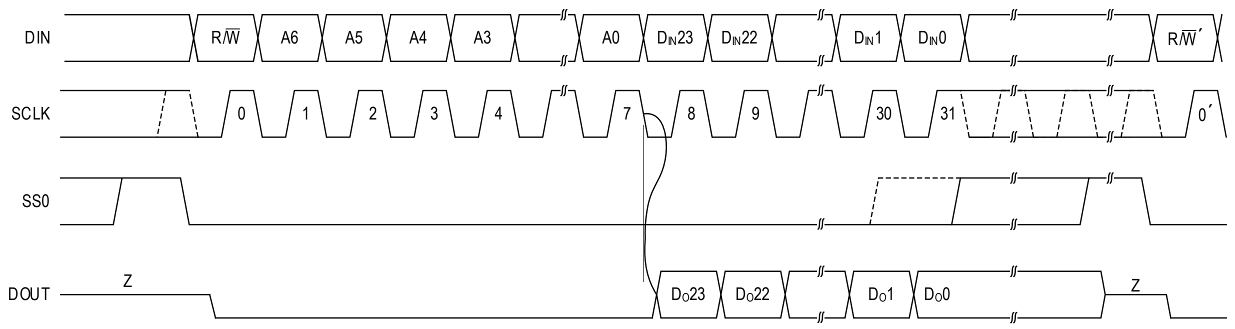
Table 14-4: Register Address Byte

D7	D6	D5	D4	D3	D2	D1	D0
R/WB	REG_ADDR[6:0]						

14.2.4 SPI Transactions and ADC Conversions

When an ADC conversion is started using either of the 16-/24-bit Delta-Sigma ADCs, any SPI register writes to the same ADC results in the conversion being aborted. To allow an ADC conversion to complete, while switching to a different AFE peripheral, it is required to first read the *AFE_SYS_CTRL* register and then modify the *AFE_SYS_CTRL*.*ana_src_sel* field to a different AFE peripheral and then write the *AFE_SYS_CTRL* register.

Figure 14-2: AFE SPI Communications Diagram with CRC-5-USB Disabled



14.3 Selecting an AFE Peripheral

Selecting an AFE peripheral requires writing to the *AFE_SYS_CTRL* register and setting the *AFE_SYS_CTRL*.*ana_src_sel* field to the desired AFE peripheral. To set the *AFE_SYS_CTRL* register, it is recommended to first read the register and modify the desired fields before writing the register. Select one of the four AFE components using the following steps:

1. Configure the SPI0 interface as specified in *Table 14-2*.
2. Perform an 8-bit SPI read using the *AFE_SYS_CTRL* register address.
   a. The data read is the current value of the *AFE_SYS_CTRL* register.
3. Modify the data read and change the *AFE_SYS_CTRL*.*ana_src_sel* field to the desired selection.
   a. Set *AFE_SYS_CTRL*.*ana_src_sel* to 0 to select ADC_ZERO.
   b. Set *AFE_SYS_CTRL*.*ana_src_sel* to 1 to select ADC_ONE.
   c. Set *AFE_SYS_CTRL*.*ana_src_sel* to 2 to select the DAC.
   d. Set *AFE_SYS_CTRL*.*ana_src_sel* to 3 to select the HART modem.
4. Perform an 8-bit SPI write using the *AFE_SYS_CTRL* register address and the modified *AFE_SYS_CTRL* value as the write data.

14.4 AFE Registers

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

Table 14-5: AFE Registers

Address	Width	Register Name	Description
0x7A	8 bits	*AFE_SYS_CTRL*	AFE System Control Register

14.4.1 Register Details

Table 14-6: AFE System Control Register

AFE System Control		**AFE_SYS_CTRL**		**0x7A**
Bits	**Name**	**Access**	**Reset**	**Description**
7	crc_inv	R/W	0	**CRC5 Bit Invert** Set this field to 1 to invert the CRC5 bits. This field has no effect unless the CRC5 is enabled. Enable the CRC5 by setting *AFE_SYS_CTRL.crc5* to 1. 0: CRC5 bits are not inverted for SPI reads. 1: Invert CRC5 bits for SPI reads.
6	por_flag	R/W1O	0	**AFE POR Flag** This field indicates if a POR event has occurred. Setting this field to 1 clears the POR status. Once set to 1, only a POR event can clear this field to 0. Software should write this field to 1 after any form of reset. This field reads 1 until another reset event occurs. 0: POR occurred. 1: POR event has not occurred.
5	spi_abort_dis	See Description	0	**SPI Write Abort ADC Conversion** Reading the *AFE_SYS_CTRL* register automatically sets this field to 1. See *SPI Transactions and ADC Conversions* and *Selecting an AFE Peripheral* for details of how this field is used to allow switching AFE peripherals while allowing an ADC conversion to complete.
4	hart_en	R/W	0	**HART Modem Enable** Setting this field to 1 enables the HART modem. 0: HART Modem disabled. 1: HART Modem enabled.
3	-	RO	0	**Reserved**
2	crc5	R/W	0	**CRC5 Enable for SPI Reads** Setting this field to 1 enables the CRC5-USB protocol on all SPI reads, extending the read by 1 byte. The CRC5 is read as the last byte of the SPI read transaction. 0: CRC5 disabled. 1: CRC5 enabled.
1:0	ana_src_sel	R/W	0	**AFE Source Select** Set this field to the desired analog block's register set. The default setting of this field selects the ADC_ZERO peripheral. 0: ADC_ZERO registers 1: ADC_ONE registers 2: DAC12 registers 3: HART registers

15. HART Modem (HART)

Highway Addressable Remote Transducer (HART) communication is a widely implemented serial communication interface often used in electrically noisy applications. Digital signals are superimposed on the analog signal of a 4–20mA current loop.

The HART modem interfaces directly with a HART communications network.

The HART protocol is based on the phase continuous frequency shift keying (FSK) technique. Bit 0 is modulated to a 2200Hz trapezoidal signal, and bit 1 is modulated to a 1200Hz trapezoidal signal with a baud rate of 1200bps. The hardware superimposes the two frequencies, which can easily be superimposed on the analog current-loop signal. The current-loop signal operates in the range of DC to 10Hz, without affecting either of the FSK signals. The unique methodology of the HART protocol enables simultaneous analog and digital communication on the same wire.

Functionally, the peripheral appears as a UART at the link level and with the 4-20mA current loop at the physical layer.

Software controls the HART mode using the internal SPI port shown in *Figure 15-1*. Data communications use the internal UART shown in the same figure. Software is required to configure the SPI and UART interface for communications between the CPU and the internal HART modem. When the HART modem is in use, the SPI port and UART port are dedicated to the HART modem and are not available for any other use. When software disables the HART modem, the SPI port and UART port are available for general usage.

HART communication is accomplished through a series of commands and responses depending on the specific protocol and network topology. The HART peripheral does not implement any portion of the communication protocol; it only handles the modulation and demodulation of the encoded information. Analog Devices, Inc. provides a complete software solution as an API included with the MAX32675 Software Development Kit.

HART Peripheral Features:

- Integrated design reduces PCB footprint and components.
- Digital signal processing increases reliable signal detection in noisy environments.
- Software access uses SPI for configuration and control. DMA transactions are supported through the standard DMA peripheral.
- The peripheral supports both 500Ω and 30kΩ line impedances.
- The HART transmitter provides the required trapezoidal output signal.
- The HART receiver processes both trapezoidal and sinusoidal input signals.
- Dynamic power gating minimizes current by disabling sub-components when inactive.
- Fast carrier detect output (OCD) assertion after a valid HART signal is received.
- Programmable signal thresholds:
    - Bit detect up
    - Bit detect down
    - Carrier detect up
    - Carrier detect down

15.1 Instances

Instances of the peripheral are listed in *Table 15-1*. The SPI and UART interfaces listed are used when the peripheral is active and should not be used by software while the HART is in use. If desired, the HART peripheral can be disabled, allowing the SPI and UART to be used as external communication interfaces.

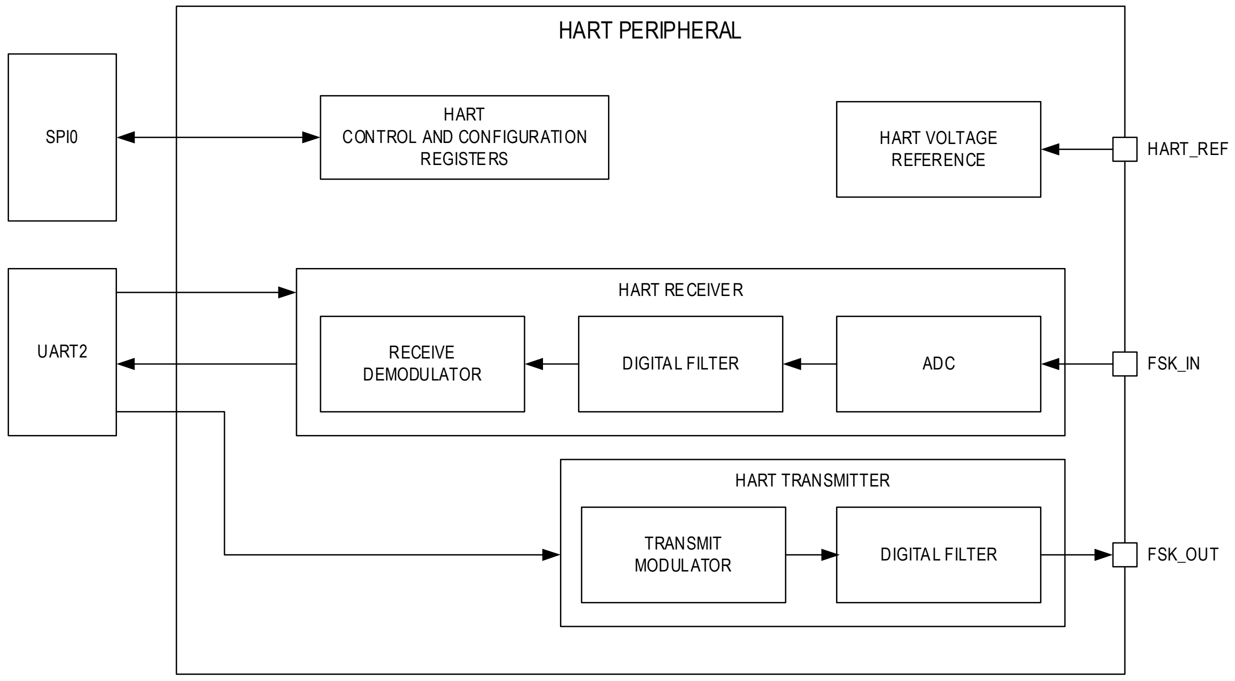
Table 15-1: MAX32675 HART Modem Instances

Instance	SPI Interface	UART Interface	HART Registration Certificate Number
HART	SPI0	UART2	Pending

15.2 Functional Description

The HART peripheral consists of a demodulator, carrier detect, and digital filter as well an ADC for input signal conversion and a modulator and DAC for output signal generation. *Figure 15-1* shows the HART peripheral's block diagram. The SPI interface is used for the configuration of the HART peripheral, and the UART interface provides control and data communications through HART.

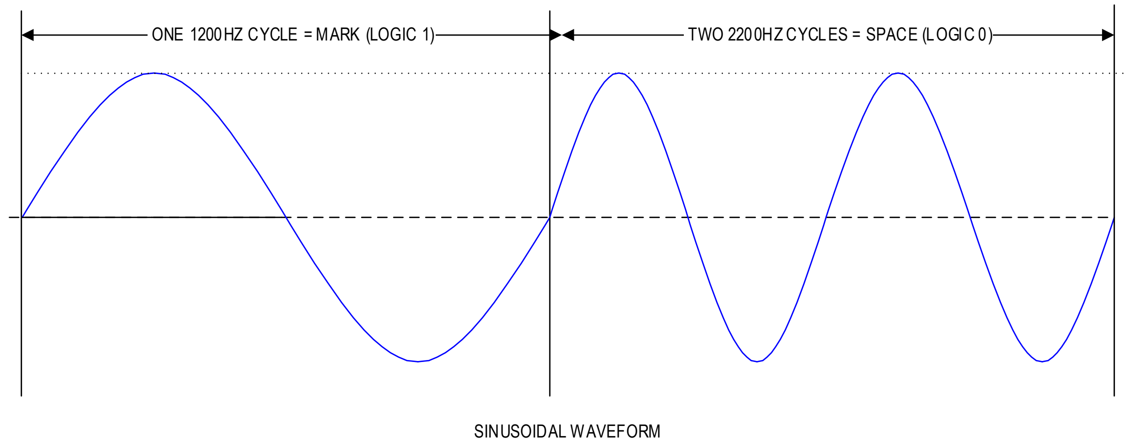
Figure 15-1: MAX32675 HART Block Diagram

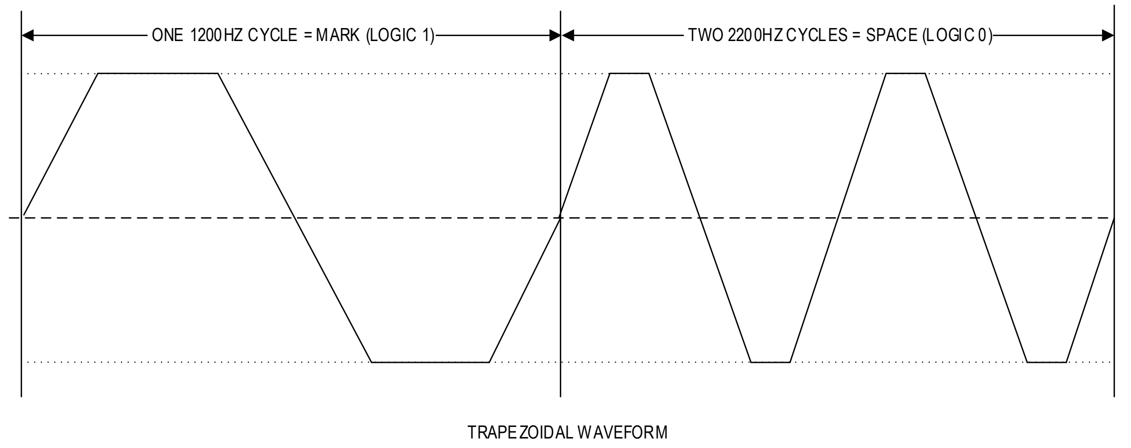


The internal connection between the SPI interface and the SPI register memory is shared with the dual 16-/24-bit Delta-Sigma ADC with PGA and the 12-bit DAC peripherals. See the *Analog Front-End (AFE)* chapter for details on the configuration and details of the communication protocol used to communicate with each of the AFE peripherals.

The HART peripheral transmitter generates a trapezoidal, instead of sinusoidal, signal that is compliant with the relevant HART specifications, as shown in *Figure 15-2*.

Figure 15-2: HART Waveforms Trapezoid and Sinusoid





15.3 Modulator

The FSK-modulated signal is present at the FSK_OUT output pin, as illustrated in *Figure 15-2*. The modulator is enabled by the UART RTS signal set to a logic low (0). The modulator preserves a continuous phase when switching between frequencies to minimize the bandwidth of the transmitted signal.

15.4 Demodulator

The demodulator accepts an FSK signal and reproduces the original modulating signal. The HART signal should be presented as an 11-bit UART character with a start, data, parity, and stop bits for proper operation of the demodulator block. The nominal bit rate of the D_OUT signal is 1200 bits per second. See the schematic in the corresponding evaluation kit for an example of the simple RC filter to condition the raw analog signal and reduce anti-aliasing.

15.5 HART Registration

The peripheral incorporates circuitry, which was previously validated, and received a Modem IC Registration Certificate from the HART Communication Foundation. The use of a HART-registered IC reduces the customer cost and effort associated with achieving HART registration of the end product. This IC has been submitted to the HART Communication

Foundation for evaluation, and a copy of the MAX32675 Registration Certificate will be available at *https://www.fieldcommgroup.org/technologies/hart* when the registration is complete.

15.6 HART Protocol and Interface Management

The implementation of the HART modem protocol for this device involves software interrupts and data handling routines. The details of this software are beyond the scope of this chapter and are demonstrated fully in the SDK.

15.7 Writing and Reading the HART Registers

The communication interface consists of:

- UART Data Interface
- SPI Control Interface

HART modem data transfer is performed through the UART2 FIFO located on the APB bus. Once configured, the transmit and receive data is available in their respective FIFOs. Data management is performed using the FIFO status flags and programmable interrupts

The modem control and configuration registers for this peripheral are not mapped to the APB bus like other peripherals. Instead, they are accessed indirectly through the internal SPI0 port shown in *Figure 15-1*. Once the SPI0 port is configured, the HART registers are accessed using standard SPI read and write operations. See *Table 15-2* for the addresses of each of the HART registers. The procedure for configuring the SPI0 peripheral and accessing these registers is described in *Analog Front-End (AFE)* and fully explained in the SDK example software.

15.8 Configuring the Modem

The basic steps for the internal configuration of the HART modem and UART interface are shown below. The device's SDK provides all the necessary software and configuration information. The customer is encouraged to use the provided software, which is part of the HART certificate registration.

The HART modem must be disabled before writing to any of the SPI-mapped registers and writing to any UART control registers.

The HART modem uses trim values calculated during testing to maximize the accuracy of the HART analog signals. The values are stored in non-volatile memory and must be copied to the appropriate volatile RAM registers following any reset. Refer to the SDK for details on how this process is done.

1. Select the HART peripheral if not already selected. See *Selecting an AFE Peripheral* for details.
2. If not already disabled, clear AFE_SYS_CTRL.*hart_en* to 0 to disable the peripheral.
3. Configure the HART clock source (P0.10. AF4)
    a. Enable the ERFO oscillator as described in *16MHz to 32MHz External Radio Frequency Oscillator (ERFO)*.
    b. Write *GCR_PCLKDIV*.*div_clk_out_ctrl* to the value required to obtain a 4MHz clock rate from the ERFO frequency.
    c. Set *GCR_PCLKDIV*.*div_clk_out_ctrl* to 1.
    d. Write *GPIOn_EN0_CLR* to 0x0000 0400
    e. Write GPIOn_EN1_CLR to 0x0000 0400.
    f. Write GPIOn_EN2_SET to 0x0000 0400.
    g. Write *GPIOn_OUTEN_SET* to 0x0000 0400.

4. Configure internal connections
   a. Configure RTS
      i. Write *GPIOn_EN0_SET* to 0x0000 0100.
      ii. Write GPIOn_EN1_CLR to 0x0000 0100.
      iii. Write GPIOn_EN2_CLR to 0x0000 0100.
      iv. Write GPIOn_OUTEN_SET to 0x0000 0100.
   b. Configure CD
      i. Write GPIOn_EN0_SET to 0x0001 0000.
      ii. Write GPIOn_EN1_CLR to 0x0001 0000.
      iii. Write GPIOn_EN2_CLR to 0x0001 0000.
      iv. Set GPIOn_DUALEDGE[16] to 1.
      v. Set GPIOn_INTEN[16] to 1.
   c. Configure HART_IN
      i. Write GPIOn_EN0_SET to 0x0000 8000.
      ii. Write GPIOn_EN1_CLR to 0x0000 8000.
      iii. Write GPIOn_EN2_CLR to 0x0000 8000.
      iv. Write GPIOn_OUT_CLR to 0x0000 8000.
      v. Write GPIOn_OUTEN_SET to 0x0000 8000.
5. Configure the UART2 interface
   a. Write *UART2_CTRL*.*bclksrc* to 0b10 to select the ERFO as the clock source.
   b. Set *UART2_CTRL*.*bclken* to 1 to enable the clock source.
   c. Write *UART2_CLKDIV*.*clkdiv* to the value required to generate a 1200Hz baud rate from the ERFO frequency.
   d. Set *UART2_CTRL*.*hfc_en* to 1 to enable hardware flow control.
   e. Write *UART2_CTRL*.*char_size* to 0b11 to select 8-bit character length.
   f. Clear *UART2_CTRL*.*par_mode* to 0 to calculate mark parity.
   g. Set *UART2_CTRL*.*par_eo* to 1 to calculate odd parity.
   h. Set *UART2_CTRL*.*par_en* to 1 to enable parity calculations.

15.9 HART Registers

These registers are accessed indirectly through the internal SPI port. See the *Analog Front-End (AFE)* chapter for details. See *Table 1-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

All HART modem registers are 24-bits wide. The register addresses listed are absolute addresses.

Table 15-2: HART Modem Register Summary

Address	Width	Name	Description
0x00	24 bits	*AFE_HART_CTRL*	Control Register
0x01	24 bits	*AFE_HART_RX_TX_CTL*	Receive-Transmit Control Register
0x02	24 bits	*AFE_HART_RX_CTL_EXT1*	Receive Control Extension 1 Register
0x03	24 bits	*AFE_HART_RX_CTL_EXT2*	Receive Control Extension 1 Register
0x04	24 bits	*AFE_HART_RX_DB_THRSHLD*	Receive Bit-Detect/Demodulation Threshold Register
0x05	24 bits	*AFE_HART_RX_CRD_UP_THRSHLD*	Receive Carrier Detect Up Threshold Register
0x06	24 bits	*AFE_HART_RX_CRD_DN_THRSHLD*	Receive Carrier Detect Down Threshold Register
0x07	24 bits	*AFE_HART_RX_CRD_DOUT_THRSHLD*	Receive Carrier Detect DOUT Threshold
0x08	24 bits	*AFE_HART_TX_MARKSPACE_CNT*	Transmit Mark-Space Count Values Register
0x09	24 bits	*AFE_HART_STAT*	Status Register

Address	Width	Name	Description
0x0A	24 bits	*AFE_HART_TRIM*	Trim Register
0x0B	24 bits	*AFE_HART_TM*	Test Mode Register 0

15.9.1 Register Details

Table 15-3: HART Control Register

HART Control			**AFE_HART_CTRL**	**0x00**
Bits	**Field**	**Access**	**Reset**	**Description**
23:1	-	RO	0	**Reserved**
0	adm_tm_en	R/W	0	**ADC Test Mode Enable** When set, this bit enables the HART ADC test mode for INL/DNL measurements. The *AFE_SYS_CTRL*.*hart_en* bit must be set to enable this mode. When this bit is set, the RTS to the HART digital is automatically set and is not taken from the designated GPIO. This then enables the HART receive mode as a functional mode, but the GPIO[6:1] now outputs the HART ADC Data Output, ADC_DOUT[7:2] bits for INL/DNL measurements. 0: Disabled. 1: Enabled.

Table 15-4: HART Receive-Transmit Control Register

HART Receive-Transmit Control			**AFE_HART_RX_TX_CTL**	**0x01**
Bits	**Field**	**Access**	**Reset**	**Description**
23	tx_4mhz_clk_en	R/W	1	**Transmit 4MHz Clock Enable** Selects the clock that generates the TX_FSK_CLK_DIG input for the transmit slew rate AFE. The default 4MHz results in higher accuracy in generating the mark/space outputs on the TX_FSK_CLK_DIG output to the AFE. When selecting the 2MHz clock for CLK_TX, the mark/space count values must also be changed in the *AFE_HART_TX_MARKSPACE_CNT* register. 0: 2MHz. 1: 4MHz.
22	tx_ws_dis_rs	R/W	0	**Transmit Disable Edge** Selects which edge of the TX_FSK_CLK_DIG disables the transmit slew rate AFE. 0: Falling. 1: Rising .
21	tx_bus_dcl_en	R/W	1	**Bus DC Load Select** 0: 30kΩ. 1: 500Ω.
20	tx_buf_en	R/W	1	**Transmit Buffer Enable** 0: Disabled. 1: Enabled.
19:16	rx_adc_pwr_dly_cnt	R/W	0x2	**Receive ADC Power-up Sample Ignore Count** Sets the delay in ADC cycles before the carrier detect, and the bit-detect logic is enabled to reject false assertions/glitches when the receiver is enabled. Note that this has a direct impact on carrier detect assertion time; thus, the value must be programmed appropriately. A large value delays carrier detect assertion delaying the receive traffic.
15:8	rx_bp_settle_cnt	R/W	0x50	**Receive Bandpass Settling Count** This sets the number of ADC samples to delay before enabling the Receive DSP. This ignores Carrier-Detect/Bit-Detect glitches due to the initial power-up of the ADC.

HART Receive-Transmit Control			AFE_HART_RX_TX_CTL	0x01
Bits	**Field**	**Access**	**Reset**	**Description**
7:4	rx_adc_pwr_up_smp_ignr	R/W	0x4	**Receive ADC Ignore Sample Count on Power Up** 0: Disabled. 1: Enabled.
3	rx_dout_uart_en	R/W	0	**Receive Demodulated Bit UART Timing Enable** 0: UART bypassed. 1: UART timing.
2	rx_adc_offset_sel	R/W	0	**Receive ADC Offset Select** 0: Disabled. 1: Enabled.
1	rx_adc_refbuf_en	R/W	1	**Receive ADC Reference Buffer Enable** 0: Disabled. 1: Enabled.
0	rx_adc_ref_en	R/W	1	**Receive ADC Reference Enable** 0: Disabled. 1: Enabled.

Table 15-5: HART Receive Control Extension 1 Register

HART Receive Control Extension 1			AFE_HART_RX_CTL_EXT1	0x02
Bits	**Field**	**Access**	**Reset**	**Description**
23:19	-	RO	0	**Reserved**
18:0	rx_an_init_val	R/W	0	**Receive AN Initialization Value** The AN register can be initialized at the start of receive transaction (before enabling the HART through the MAX32675 system control register) to the value programmed in this register to help quick carrier detect (OCD) activation. Normally the AN average builds up from zero to the carrier detect threshold. Initializing this and the ARN registers (must be done together to see the effect) can shorten this process. This is useful to create a less noisy scenario to ensure fast carrier detection on the HART signal application.

Table 15-6: HART Receive Control Extension 2 Register

HART Receive Control Extension 2			AFE_HART_RX_CTL_EXT2	0x03
Bits	**Field**	**Access**	**Reset**	**Description**
23:22	-	RO	0	**Reserved**
21	rx_uart_timer_fast_cnt_en	R/W	0	**Receive UART Timer Fast Count Enable** 0: Disabled. 1: Enabled.
20	rx_uart_timer_syn_alws_en	R/W	1	**Receive UART Timer Bit Transition Synchronize** This determines which event triggers the UART synchronization. 0: Synchronization occurs on every start bit. 1: Synchronization occurs on every one to zero transition.
19:18	-	RO	0	**Reserved**
17:16	rx_zc_ign_val	R/W	0b01	**Receive Zero Cross Ignore Value** This determines how many zero crossings are ignored after the receiver bandpass settling time counter expires before the carrier detect is enabled. After the HART receive bandpass settling time counter expires. This allows the filters to stabilize and gives control over carrier-detect/bit-detect glitching, particularly in noisy environments. Note that this has a direct impact on carrier detect assertion time; thus, the value must be programmed appropriately. A large value delays carrier detect assertion delaying the received traffic.
15	-	RO	0	**Reserved**

HART Receive Control Extension 2		AFE_HART_RX_CTL_EXT2		0x03
Bits	**Field**	**Access**	**Reset**	**Description**
14:0	rx_arn_init_val	R/W	0	**Receive AN Initialization Value** This value determines the minimum delay for quick carrier detect (OCD) activation in applications that can guarantee relatively low levels of electrical noise. This value should be left at the default value in an end-product and only modified during product development if needed.

Table 15-7: HART Receive Bit-Detect/Demodulation Threshold Register

HART Receive Bit-Detect/Demodulation Threshold		AFE_HART_RX_DB_THRSHLD		0x04
Bits	**Field**	**Access**	**Reset**	**Description**
23:21	-	RO	0	**Reserved**
20:12	rx_bitdtct_up_thrshld	R/W	0	**Receive Bit Detect Up Threshold Value** When in receive mode, this is the bit detect up threshold value used for deciding the demodulated bit.
11:9	-	RO	0	**Reserved**
8:0	rx_bitdtct_dn_thrshld	R/W	0	**Receive Bit Detect Down Threshold Value** When in receive this is the bit detect down threshold value used for deciding the demodulated bit.

Table 15-8: HART Receive Carrier Detect Up Threshold Register

HART Receive Carrier Detect Up Threshold		AFE_HART_RX_CRD_UP_THRSHLD		0x05
Bits	**Field**	**Access**	**Reset**	**Description**
23:19	-	RO	0	**Reserved**
18:0	rx_crd_up_thrshld	R/W	0	**Receive Carrier Detect Up Threshold Value** When in receive this is the carrier detect up threshold value used for deciding the carrier detect.

Table 15-9: HART Receive Carrier Detect Down Threshold Register

HART Receive Carrier Detect Down Threshold		AFE_HART_RX_CRD_DN_THRSHLD		0x06
Bits	**Field**	**Access**	**Reset**	**Description**
23:19	-	RO	0	**Reserved**
18:0	rx_crd_dn_thrshld	R/W	0	**Receive Carrier Detect Down Threshold Value** This is the carrier detect down threshold value used for deciding the carrier detect.

Table 15-10: HART Receive Carrier Detect DOUT Threshold Register

HART Receive Carrier Detect DOUT Threshold		AFE_HART_RX_CRD_DOUT_THRSHLD		0x07
Bits	**Field**	**Access**	**Reset**	**Description**
23:19	-	RO	0	**Reserved**
18:0	rx_crd_dout_thrshld	R/W	0	**Receive Carrier Detect DOUT Threshold Value** This adjusts the threshold used to suppress the DOUT = zero decoding in the receiver demodulation logic.

Table 15-11: HART Transmit Mark-Space Count Values Register

HART Transmit Mark-Space Count Values		AFE_HART_TX_MARKSPACE_CNT		0x08
Bits	**Field**	**Access**	**Reset**	**Description**
23	-	RO	0	**Reserved**
22:12	tx_mark_cnt	R/W	0	**Transmit Mark Count Value**
11	-	RO	0	**Reserved**

HART Transmit Mark-Space Count Values		AFE_HART_TX_MARKSPACE_CNT		0x08
Bits	**Field**	**Access**	**Reset**	**Description**
10:0	tx_space_cnt	R/W	0	**Transmit Space Count Value** This is the number of clocks to be counted for Space frequency on TX_FSK_CLK_DIG output. The value programmed is based on the *AFE_HART_RX_TX_CTL.tx_4mhz_clk_en* value.

Table 15-12: HART Status Register

HART Status		AFE_HART_STAT		0x09
Bits	**Field**	**Access**	**Reset**	**Description**
23:0	-	DNM	0	**Reserved**

Table 15-13: HART Trim Register

HART Trim		AFE_HART_TRIM		0x0A
Bits	**Field**	**Access**	**Reset**	**Description**
23:0	-	R/W	0	**Reserved** The values in this field should be written once after every reset, with the factory calibration values as described in *Configuring the Modem*. The values should not be modified by the user after that.

Table 15-14: HART Test Mode Register

HART Test Mode 0		AFE_HART_TM		0x0B
Bits	**Field**	**Access**	**Reset**	**Description**
23:4	-	RO	0	**Reserved**
3	tm_vref_en	R/W	0	**Test Mode V_{REF} Enable** 0: Disabled. 1: Enabled.
2	tm_bg_en	R/W	0	**Test Mode Bandgap Analog Enable** 0: Disabled. 1: Enabled.
1	tm_bias_en	R/W	0	**Test Mode BIAS Analog Enable** 0: Disabled. 1: Enabled.
0	tm_en	R/W	0	**Test Mode Enable** 0: Disabled. 1: Enabled.

16. 16-/24-Bit Delta-Sigma ADC with PGA

The MAX32675 includes dual multi-channel 16-/24-bit delta-sigma ADCs with features and specifications optimized for precision sensor measurement. The architecture includes a low-noise programmable gain amplifier (PGA), low-power input buffers, programmable matched current sources, differential/single-ended input multiplexer, and integrated on-chip oscillator.

Features include:

- PGA with available gains from 1× to 128×
  - Very high input impedance
  - Optimizes overall dynamic range
- Low-power input buffers
  - Provide input isolation
- Selectable reference
  - Internal differential (V_{REF})
  - External differential
- Programmable current sources
  - Bias for resistive sensors
  - 16 current level options
  - Detection of broken sensor wires
- 12 analog inputs
  - 6 differential or 12 single-ended
- Sample rates up to 15,360 samples per second
- FIR digital filter
  - Allows fast settling time while providing rejection of 50Hz and 60Hz line noise

16.1 Instances

There are two instances of the 16-/24-bit delta-sigma ADC, as shown in *Table 16-1*.

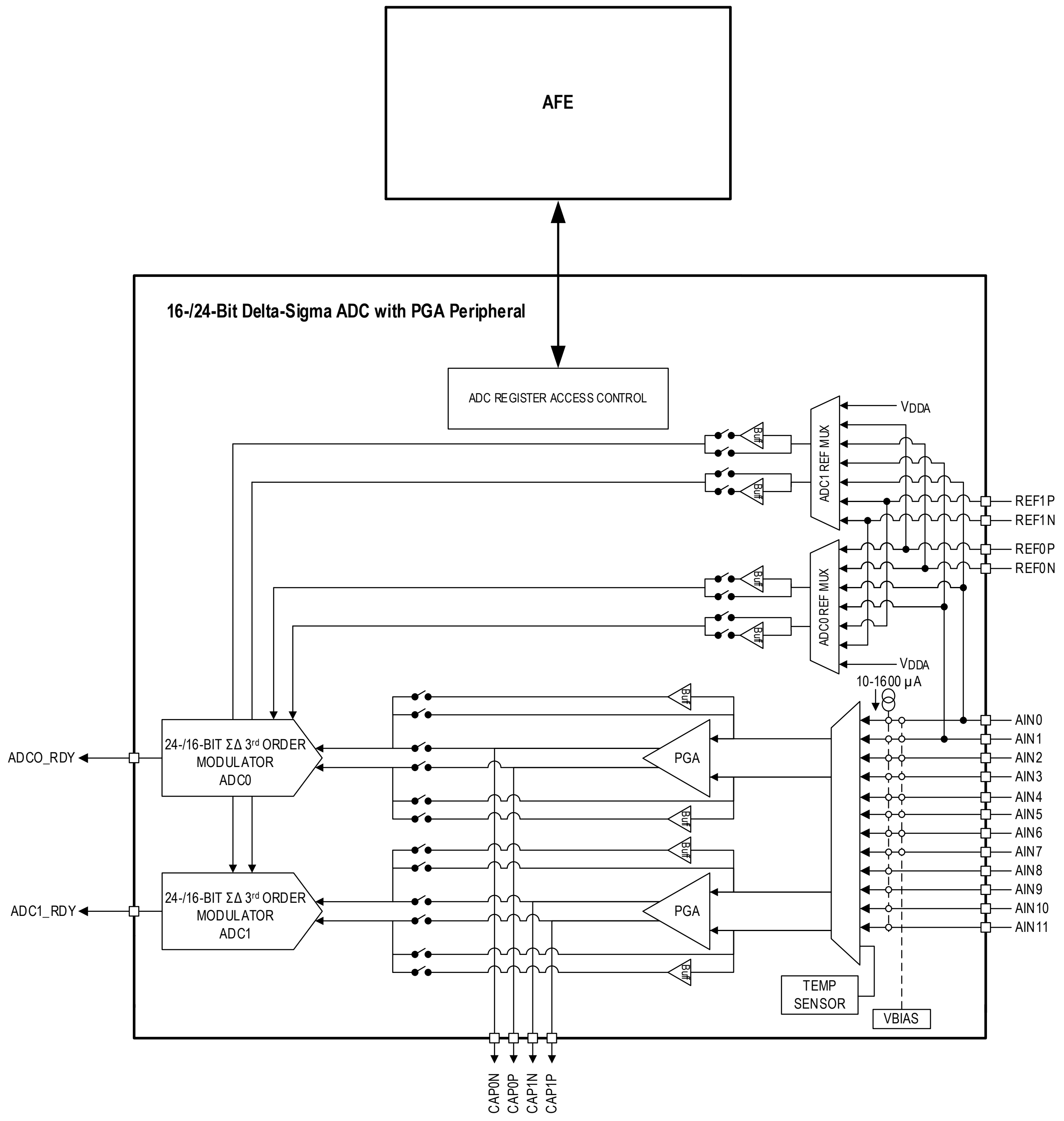
Table 16-1: MAX32675 16-/24-bit ADC with PGA Instances

Instance
AFE_ADC_ZERO
AFE_ADC_ONE

16.2 Functional Description

Figure 16-1 shows the ADC peripheral's block diagram. The SPI interface is used for configuration and reading of each of the 16-/24-bit delta-sigma ADCs registers. Each ADC instance contains an independent set of registers to control and configure the ADC, PGA, reference selection, and analog inputs.

Figure 16-1: MAX32675 ADC Block Diagram



16.3 Detailed Description

This low-power, multi-channel, 24-bit delta-sigma ADC has features and specifications optimized for precision measurement of sensors and other analog signal sources.

The input section includes a low-noise PGA with very high input impedance and available gains from 1× to 128× to optimize the overall dynamic range. In addition, low-power input buffers can be enabled to provide isolation of the signal source from the modulator's switched-capacitor sampling network when the PGA is not in use, reducing the supply current requirements compared to the PGA.

Several integrated features simplify precision sensor applications. The programmable matched current sources provide excitation for resistive sensors; sixteen different current levels are available, allowing sensor full-scale range to be tuned for optimum signal-to-noise ratio. An additional current sink and current source supply small current levels to aid in detecting broken sensor wires. The 6-channel differential/12-channel single-ended multiplexer provides the flexibility needed for complex multi-sensor measurements. Two GPIO, ADC0_RDY and ADC1_RDY, reduce isolation components and ease control of switches or other circuitry.

The ADC can operate in continuous conversion mode at data rates up to 15,360 sps and in single-cycle conversion mode at rates up to 3,840sps. When used in single-cycle mode, the digital filter settles within a single conversion cycle. The available FIR digital filter allows single-cycle settling in 16ms while providing more than 90dB simultaneous rejection of 50Hz and 60Hz line noise. The integrated on-chip oscillator requires no external components. If needed, an external clock source may be used instead. Control registers and conversion data are accessed through the AFE using the SPI0 internal interface.

16.4 Analog Inputs

The twelve analog inputs (AIN0–AIN11) are configurable for differential/single-ended operation. For each conversion, the input multiplexer can be configured such that any of the twelve external analog inputs or V_{DDA} can be used as the positive input. Additionally, any of the twelve external analog inputs or V_{SSA} can be used as the negative input for the differential measurement. Additionally, each input mux also supports an internal temperature sensor input. The multiplexer outputs may either drive the ADC inputs directly or drive low-power buffers. They then drive the ADC or the PGA inputs.

16.5 Signal Path Considerations

Three signal-path options are available to trade power-supply current against input impedance, gain, and input voltage range by enabling the PGA or the input buffers or bypassing both and driving the modulator directly. The PGA control register selects among these options, which are summarized below.

16.5.1 Bypass (Direct Signal Path) Mode

In bypass mode, the multiplexer outputs are directly connected to the ADC modulator inputs. In this mode, the input buffer and the PGA are disabled for minimum power-supply current. This mode allows input voltages from V_{SSA} - 30mV to V_{DDA} + 30mV and adds no amplifier noise to the signal. Input bias current is typically 1µA/V, which is appropriate when driving with a low source resistance.

For smaller signal amplitudes, "digital gains" of 2 and 4 are available when using the direct signal path. See the *Digital Gain* section for more information.

16.5.2 Buffered Mode

The multiplexer outputs drive the inputs to the low-power signal buffers in buffered mode, which then drive the ADC modulator inputs. Selecting buffered mode disables the PGA. Input voltages from V_{SSA} + 100mV to V_{DDA} - 100mV are accepted in this mode, and no amplifier noise is added to the signal. The input bias current, typically 61nA, is significantly less than that in the direct mode, so higher source resistances may be accommodated without causing appreciable errors. Enabling the input buffers increases the power supply current by 35µA (typical) compared to the bypassed (direct signal path) mode.

As with the bypassed mode, digital gains of 2 and 4 are available when using the buffered mode. See the *Digital Gain* section for more information.

16.5.3 PGA Mode

The PGA provides 1, 2, 4, 8, 16, 32, 64, or 128 gain. Selecting PGA mode enables the PGA, connects the PGA inputs to the multiplexer outputs, connects the PGA outputs to the ADC modulator inputs, and disables the low-power input buffers. The PGA accepts input voltages from V_{SSA} + 100mV to V_{DDA} - 100mV for gains up to 16, and V_{SSA} + 200mV to V_{DDA} - 200mV for gains from 32 to 128. When enabled, the PGA supply current is typically 130µA.

Input current in PGA mode is much lower than in the buffered or direct modes, so PGA mode is a good choice for maintaining precision when source resistances are high.

Note: The input current in PGA mode is dominated by multiplexer leakage current and is highest when the input voltage, including that of unused inputs, is nearest V_{DDA} or GND. For applications that are most sensitive to the effects of input current, connect any unused inputs to a voltage near $\frac{V_{DDA}}{2}$.

Note: The maximum usable gain is limited by the reference voltage and input voltage. Ensure that the differential input voltage multiplied by the PGA gain is less than or equal to the reference voltage:

$$V_{IN} \times GAIN \leq V_{REF}$$

Where:

V_{IN} = differential input voltage

GAIN = PGA gain

V_{REF} = reference voltage

Also, ensure that the input common-mode voltage (V_{CM}) falls within the acceptable common-mode voltage range of the PGA:

$$200mV + \frac{V_{IN} \times GAIN}{2} \leq V_{CM} \leq V_{DDA} - 200mV - \frac{V_{IN} \times GAIN}{2}\ for\ gains\ of\ 32\ to\ 128\ or$$

$$100mV + \frac{V_{IN} \times GAIN}{2} \leq V_{CM} \leq V_{DDA} - 100mV - \frac{V_{IN} \times GAIN}{2} for\ gains\ of\ 1\ to\ 16$$

Where:

$$V_{CM} = \frac{AINP + AINN}{2}$$

Enable the PGA and set the gain using the following steps:

1. Set the *AFE_ADC_n_PGA* register fields:
   a. Set the signal path field, *AFE_ADC_n_PGA.sig_path* field to 2.
   b. Set the desired gain by setting the *AFE_ADC_n_PGA.gain* field.
2. Write the *AFE_ADC_n_PGA* register using the AFE SPI interface.

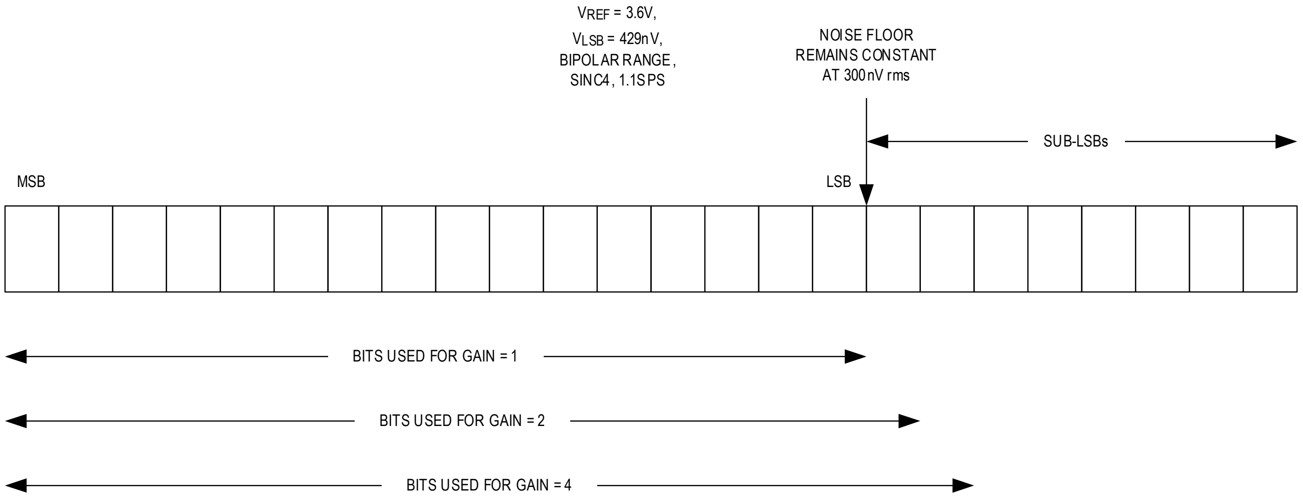
16.6 Digital Gain

Programmable digital gain settings of 2 and 4 are available in the Direct and Buffered modes. Select the desired gain using the *AFE_ADC_n_PGA.gain* field. Gain bits of the PGA register. Digital gain selections greater than or equal to 4 results in digital gain equal to 4. The input range is 0V to $\frac{V_{REF}}{GAIN}$ for unipolar conversions or $\pm \frac{V_{REF}}{GAIN}$ for bipolar conversions.

The modulator produces 32 bits of data, and for unity gain, the 8 LSBs are truncated before the data is stored in the 24-bit conversion data registers. Selecting a digital gain of 2 causes the MSB and the 7 LSBs to be discarded, producing 24 bits of data with an effective "gain" of 2.

Note: For any data rate, the noise floor remains constant, independent of the digital gain setting. Digital gain is useful for systems whose input noise is dominated by the source or systems that can take advantage of averaging multiple readings to improve effective resolution. For cases when the output noise is below an LSB, using digital gain can decrease the input-referred noise at the expense of reduced dynamic range.

Figure 16-2: Digital Programmable Gain Example



16.7 Reference Inputs

There are three selectable differential reference voltage inputs. Select the reference input using the *AFE_ADC_n_CTRL.ref_sel* field. Either V_{REFP}, V_{REFN}, or both may be buffered, as determined by the *AFE_ADC_n_CTRL.refbufp_en* and *AFE_ADC_n_CTRL.refbufn_en* bits. With the reference buffer disabled, the input current is a few microamps (2.1µA/V, typical). Enabling a reference buffer reduces the reference input current to 65nA, typical. With the buffer enabled, the common-mode voltage range for V_{REFP} and V_{REFN} is between 100mV and V_{DDA} - 100mV. With the buffer disabled, the common-mode range is between GND and V_{DDA}.

Selectable buffers allow flexibility in using resistive voltage references. For example, suppose a voltage reference is generated by driving a current through a grounded reference resistor. In that case, V_{REFN} may be unbuffered, allowing it to be connected directly to GND, while V_{REFP} is buffered, helping reduce the effect of input bias current on the reference voltage.

16.8 Low-Power Considerations

Several operating modes help to optimize power and performance. As discussed in the *Signal Path Considerations* section, applications that do not require the gain or low input bias current available in PGA mode can reduce supply current by 130µA by disabling the PGA. For low-impedance sources, the input buffers may be disabled for further power savings. Similarly, the reference buffers may be disabled when the source resistance is low. The modulator has a selectable "duty cycle" mode for low power at lower sampling rates. The IC may be placed into sleep mode between conversions to reduce the average power supply current.

16.9 Modulator Duty Cycle Mode

In addition to its normal operating mode, the modulator can be operated in a ¼ duty cycle mode to reduce power consumption for a given data rate at the expense of noise. The noise performance of a ΣΔ ADC generally improves when increasing the OSR (lowering the output data rate) because more samples of the internal modulator can be averaged to yield one conversion result. However, in applications where power consumption is critical, improved noise performance at low data rates may not be required. The internal duty cycling mode can yield significant power savings for these applications by periodically entering a low-power state between conversions. In principle, the modulator runs in normal mode with a duty cycle of 25%, performing one "normal" conversion and then automatically entering a low-power state for three consecutive conversion cycles. Therefore, the noise performance in duty-cycle mode is comparable to the noise performance in normal mode at four times the data rate. The duty-cycle mode can be selected using Direct, Buffered, or PGA signal paths. Neither the input buffers nor PGA is duty-cycled while in duty cycle mode.

Select duty-cycle mode using the *AFE_ADC_n_CONV_START.conv_type* field. To minimize current consumption in duty-cycle mode, set the signal path for an appropriate low-power mode (see *Signal Path Considerations* section for additional details).

16.10 Sleep Mode

Sleep mode (controlled by the *AFE_ADC_n_PD* register) powers down all analog circuitry, including the internal oscillator, resulting in 0.5µA typical current consumption. Exit sleep mode either by writing to the *AFE_ADC_n_PD* register or (when enabled) by using a GPIO trigger.

16.11 Circuit Settling Time

The input to the ADC requires some time to settle after changing the state of the multiplexer, PGA, current sources, and other analog components. Therefore, when using the sequencer, insert appropriate wait times when changing the state of any of these components.

16.11.1 Input Multiplexer

Settling time for changes to the state of the input multiplexer depends on several factors. These include the delay time of the nonoverlap circuits and the on-resistance of the multiplexer switches but are dominated by the output impedance of the external source, the impedance (cables, protection components) between the external source and the multiplexer, any input filter capacitance, the 10pF capacitance on the input to the PGA and modulator blocks, and whether or not the I_{DAC} current sources or the V_{BIAS} source are being used. To obtain an accurate conversion, wait until the multiplexer is fully settled before starting a new conversion. With no added capacitance at the inputs, the settling time after a multiplexer channel change with a 2kΩ source is typically 2µs.

16.11.2 PGA

The external PGA filter primarily limits PGA settling time. A 100nF external capacitor across CAP0P and CAP0N and/or CAP1P and CAP1N reduces noise by limiting the bandwidth of the PGA. This results in a 2kHz single-pole lowpass filter at the PGA's output. Settling to 22-bit accuracy (0.25ppm) requires 15.25 time constants or 1.21ms for a 2kHz bandwidth. Therefore, the PGA typically dominates the settling time of the input when changing multiplexer settings or changing the PGA's gain.

16.11.3 Reference Multiplexer

Settling time for the reference input multiplexer is similar to that of the input multiplexer but with less complexity. The reference multiplexer has fewer channels and does not have the I_{DAC} current sources or the V_{BIAS} source as possible inputs. The delay is still dependent on the on-resistance of the reference multiplexer switches, the impedance between the reference source and the reference multiplexer, the output impedance of the reference source, and the input capacitance of the modulator. For accurate conversions, it is essential to wait until the reference multiplexer is fully settled before starting a new conversion.

Normally the reference should be located close to the reference inputs, so the resistance between the source and the input should be negligible. If the reference source is an active voltage reference, the source impedance should be low enough to ignore. In some cases, the reference source may be a resistor with a value of a few kilohms. So long as the source resistance is less than around 10kΩ, the settling time contribution from the reference source resistance is less than 1µs and can generally be ignored.

16.11.4 Excitation Current Source

Enabling/disabling the current source(s) requires time for any input capacitance to charge or discharge. This can be especially important when external capacitors have been added at the inputs for noise filtering.

16.12 V_{BIAS} Source

The V_{BIAS} source generates a bias voltage equal to $\frac{V_{DD}}{2}$. There are three V_{BIAS} modes, controlled by the *AFE_ADC_n_SOURCE*.*vbias_mode* field.

The first mode is an active bias generator featuring a class AB output stage with a series 125kΩ resistor to create a nominal output impedance of 125kΩ. The active bias generator mode reduces current and channel to channel crosstalk. In active mode, if the output is not settled to $\frac{V_{DD}}{2}$, the series resistor is bypassed by a separate low-impedance class AB output stage to decrease settling time. When the output settles to $\frac{V_{DD}}{2}$, the resistor is reasserted for improved noise filtering.

The second and third modes create the V_{BIAS} with resistive voltage-dividers to offer fixed output impedance (either 125kΩ or 20kΩ) at the expense of increased current consumption. The 125kΩ mode offers increased supply noise filtering at the expense of increased settling time. The 20kΩ mode offers reduced settling time but is higher in current consumption and offers less supply noise filtering.

The bias voltage can be switched into the input channels using the channel's corresponding *AFE_ADC_n_MUX_CTRL2*.*vbias_sel_n* field.

16.13 Sensor Excitation Current Sources

The matched current sources can be programmed to provide 16 different levels of matched currents from 10µA to 1600µA with ± 10% accuracy, 0.1% matching, and 50ppm/°C temperature drift from -40°C to +85°C. Either current source or both may be enabled, and each current source can be connected to any one of the twelve analog inputs.

Note: Only one current source may be connected to any input, and a current source may not be connected to an input that has V_{BIAS} connected to it.

16.14 Burnout Currents

The internal, selectable 1µA, 5µA, and 10µA burnout current source and sink can be used to detect a sensor fault or wire break.

When enabled, the current source is connected to the selected positive analog input (AINP), and the current sink is connected to the selected negative analog input (AINN).

In an open circuit condition in the sensor input path, these burn-out currents pull the positive input towards V_{DDA} and the negative input towards V_{SSA}, resulting in a full-scale reading.

Note: A full-scale reading may also indicate that the sensor is overdriven or that the reference voltage is absent.

16.15 Calibration

The ADC can, on-demand, automatically calibrate its internal offset and gain errors and system offset and gain errors and store the calibration values in dedicated registers. The calibration register value defaults are zero (offset) and one (gain). Calibration values may be calculated and stored automatically using an *AFE_ADC_n_CAL_START* command or written directly to the registers through the serial interface. The *AFE_ADC_n_CAL_START* command selects the type of calibration to be performed (self-calibration, PGA gain calibration, system calibration) and initiates the calibration cycle. In addition, there is a separate gain calibration register for each PGA gain.

Calibration values are applied to the conversion results stored in the *AFE_ADC_n_DATA0*:*AFE_ADC_n_DATA7* registers according to the following equation:

$$\begin{aligned} AFE\_ADC\_n\_DATA[0:7] &= AFE\_ADC\_n\_SYS\_GAIN\_[A,B] \\ &\times (((Conversion - AFE\_ADC\_n\_SELF\_OFF) \times AFE\_ADC\_n\_SELF\_GAIN[1:128]) \\ &- AFE\_ADC\_n\_SYS\_OFF\_[A,B]) \end{aligned}$$

Where

$AFE\_ADC\_n\_DATA[0:7]$ is the ADC data result destination register, selected by the *AFE_ADC_n_CONV_START.dest* field,

$Conversion$ is the ADC's conversion result before calibration results are applied,

$AFE\_ADC\_n\_SELF\_GAIN[1:128]$ is the internal gain correction value for the selected gain,

$AFE\_ADC\_n\_SELF\_OFF$ is the internal offset correction value,

$AFE\_ADC\_n\_SYS\_GAIN\_[A,B]$ is the selected system gain corrections value, and

$AFE\_ADC\_n\_SYS\_OFF\_[A,B]$ is the selected system offset correction value.

All calibration operations are performed at the filter settings programmed into the *AFE_ADC_n_FILTER.linef* and *AFE_ADC_n_FILTER.rate* fields.

There are two sets of system calibration registers, A and B. Either A, B, or neither set can be applied to the ADC conversion result, selectable by the *AFE_ADC_n_SYSC_SEL* register.

Note: Calibration routines are performed using the conversion rate, PGA gain, and filter settings in the control registers. In general, slower conversion rates exhibit lower noise and therefore produce more accurate calibration.

16.15.1 Self-Calibration

In self-calibration, the required connections to zero and full scale are made internally using the PGA gain setting set in the GAIN register. Self-calibration is typically sufficient to achieve offset and gain accuracy on the order of the noise. When the gain is 1, self-calibration provides 20ppm of typical full-scale accuracy. The self-calibration routine does not include external effects such as source resistance of the signal driving the input pins, which can change the offset and gain of the system. The range of digital gain correction is from 0.5× to 2.0×. The range of offset correction is $\pm\frac{V_{REF}}{4}$. *Table 16-2* and *Table 16-3* show example values for gain and offset calibration codes.

Table 16-2: Gain Calibration Codes

Code Description	Gain	Code
Maximum Gain Correction	1.999999881	0xFFFFFF
1 LSB Greater Than Unity Gain	$1 + {}^{1}/_{2^{23}}$	0x800001
Unity Gain	1.000000	0x800000
1 LSB Less Than Unity Gain	$1 - {}^{1}/_{2^{23}}$	0x7FFFFF
Minimum Recommended Gain Correction	0.5	0x400000
Zero Gain	0	0x000000

Table 16-3: Offset Calibration Codes

Code Description	Offset	Code
Maximum Offset Correction	$0.24 \times V_{REF}$	0x7FFFFF
Positive 0.25LSB (Bipolar) or 0.5LSB (Unipolar)	$0.25 \times \frac{V_{REF}}{2^{23}-1}$	0x000001
Zero Offset Correction	0V	0x000000
Negative 0.25LSB (Bipolar) or 0.5LSB (Unipolar)	$-0.25 \times \frac{V_{REF}}{2^{23}-1}$	0xFFFFFF

Code Description	Offset	Code
Minimum Offset Correction	$-0.25 \times V_{REF} \times (1 + \frac{1}{2^{23} - 1})$	0x800000

16.15.1.1 Self-Calibration Example

Table 16-4 shows an example of self-calibration.

Table 16-4: Self-Calibration Example

Step	Description	Register	Comments
1	Select filter and rate	*AFE_ADC_n_FILTER*	For best results, select a rate no faster than the rate that is used for conversions. A slower rate results in more accurate calibration. This determines the time required to execute a calibration.
2	Select clock source and format	*AFE_ADC_n_CTRL*	For best results, select the clock source (internal or external) that is used for conversions. If the external clock is selected, ensure that the external clock is operating before beginning calibration. Format selection does not affect results.
3	Start calibration	*AFE_ADC_n_CAL_START*	Write XXXXX000 to the *AFE_ADC_n_CAL_START* register. Two conversions execute at the rate controlled by the *AFE_ADC_n_FILTER* register. The *AFE_ADC_n_SELF_OFF* and *AFE_ADC_n_SELF_GAIN_1* registers are updated.

16.15.2 PGA Self-Calibration

To ensure the lowest possible gain error, eight separate self-gain calibration registers store the calibration factors for each PGA gain from 1x to 128x. When performing gain calibration, the register corresponding to the currently selected PGA gain is updated. Perform a PGA gain calibration for each PGA gain setting that is used. Not doing so yields errors for conversions performed using the gains that have not been calibrated. Self-calibration updates the 1× self-gain register (*AFE_ADC_n_SELF_GAIN_1*).

16.15.2.1 PGA Gain Calibration Example

Table 16-5 shows an example of PGA gain calibration.

Table 16-5: Self-Calibration Example

Step	Description	Register	Comments
1	Select filter and rate	*AFE_ADC_n_FILTER*	For best results, select a rate no faster than the rate that is used for conversions. A slower rate results in more accurate calibration. Filter selection does not affect results.
2	Select gain and signal path	*AFE_ADC_n_PGA*	For best results, select the signal path that will be used for conversions. The gain selection causes the calibration value to be saved in the corresponding *AFE_ADC_n_SELF_GAIN_1*:*AFE_ADC_n_SELF_GAIN_128* registers.
3	Select clock source and format	*AFE_ADC_n_CTRL*	For best results, select the clock source (internal or external) that will be used for conversions. If the external clock is selected, ensure that the external clock is operating before beginning calibration. Format selection does not affect results.

Step	Description	Register	Comments
4	Select PGA gain and start calibration	*AFE_ADC_n_CAL_START*	Write XXXXX001 to the *AFE_ADC_n_CAL_START* register. One conversion executes at the rate controlled by the *AFE_ADC_n_FILTER* register. The *AFE_ADC_n_SELF_GAIN_1*:*AFE_ADC_n_SELF_GAIN_128* register for the selected gain is updated.

16.15.3 System Offset and Gain Calibration

A system calibration enables calibration of system zero scale and system full scale by presenting a zero-scale signal or a full-scale signal to the selected input pins and initiating a system zero-scale or system gain calibration command. As an alternative to automatic generation of the system calibration values, values may be directly written to the internal calibration registers to achieve any digital offset or scaling required. The range of digital offset correction is $\pm\frac{V_{REF}}{4}$. The range of digital gain correction is from 0.5× to 2.0×. The resolution of offset correction is 0.5 LSB.

Automatic system calibration requires applying the appropriate external signals to the selected AIN inputs. Therefore, the input multiplexer must be properly configured before system calibration. Two sets of system calibration coefficients can be created and stored (*AFE_ADC_n_SYS_OFF_A* and *AFE_ADC_n_SYS_GAIN_A*, and *AFE_ADC_n_SYS_OFF_B* and *AFE_ADC_n_SYS_GAIN_B*). Conversions may be performed using either or neither of these sets of coefficients.

Request a system offset calibration by presenting a system zero-scale signal level to the input pins and programming the *AFE_ADC_n_CAL_START* register with the appropriate value. The *AFE_ADC_n_SYS_OFF_A* or *AFE_ADC_n_SYS_OFF_B* register then updates with the value that corrects the chip zero scale.

Request a system gain calibration by presenting a system full-scale signal level to the input pins and programming the *AFE_ADC_n_CAL_START* register with the appropriate value. The *AFE_ADC_n_SYS_GAIN_A* or *AFE_ADC_n_SYS_GAIN_B* register then updates with the value that corrects the system full-scale. A system offset calibration is required before system gain calibration to ensure accurate gain calculation.

16.15.3.1 System Offset Calibration Example

Table 16-6 shows an example of system offset calibration.

Table 16-6: System Offset Calibration Example

Step	Description	Register	Comments
1	Apply "System Zero"	-	Apply the input voltage that should result in a conversion result of 0 to the appropriate analog input(s).
2	Select filter and rate	*AFE_ADC_n_FILTER*	For best results, select a rate no faster than the rate that will be used for conversions. A slower rate results in more accurate calibration.
3	Select reference input	*AFE_ADC_n_CTRL*	For best results, select a reference voltage equal to or near the value that will be used for conversions.
4	Set input multiplexer	*AFE_ADC_n_MUX_CTRL0*	Select the inputs to which "system zero" is applied.
5	Select gain and signal path	*AFE_ADC_n_PGA*	For best results, select the signal path that will be used for conversions. Gain selection does not affect results.
6	Select clock source and format	*AFE_ADC_n_CTRL*	For best results, select the clock source (internal or external) that will be used for conversions. If the external clock is selected, ensure that the external clock is operating before beginning calibration. Format selection does not affect results.
7	Select system offset and start calibration	*AFE_ADC_n_CAL_START*	Write XXXXX100 to store the result in the *AFE_ADC_n_SYS_OFF_A* register, or write XXXXX110 to store the result in the *AFE_ADC_n_SYS_OFF_B* register.

16.15.3.2 System Gain Calibration Example

Table 16-7 shows an example of a system gain calibration.

Table 16-7: System Gain Calibration Example

Step	Description	Register	Comments
1	Apply "System Full-Scale"	-	Apply an input voltage that should result in a full-scale conversion result to the appropriate analog input(s).
2	Select filter and rate	*AFE_ADC_n_FILTER*	For best results, select a rate no faster than the rate that will be used for conversions. A slower rate results in more accurate calibration.
3	Select reference input	*AFE_ADC_n_CTRL*	For best results, select a reference voltage equal to or near the value that will be used for conversions.
4	Set input multiplexer	*AFE_ADC_n_MUX_CTRL0*	Select the inputs to which "system full-scale" is applied.
5	Select gain and signal path	*AFE_ADC_n_PGA*	For best results, select the signal path that will be used for conversions. Then, select the gain that, when combined with the applied input voltage, yields a full-scale conversion result.
6	Select clock source and format	*AFE_ADC_n_CTRL*	For best results, select the clock source (internal or external) that will be used for conversions. If the external clock is selected, ensure that the external clock is operating before beginning calibration. Format selection does not affect results.
7	Select system offset and start calibration	*AFE_ADC_n_CAL_START*	Write XXXXX101 to store the result in the *AFE_ADC_n_SYS_GAIN_A* register, or write XXXXX111 to store the result in the *AFE_ADC_n_SYS_GAIN_B* register.

16.15.4 Sensitivity of Calibration Coefficients

Calibration needs to be repeated if external factors change.

- Both offset and gain calibration (PGA GAIN = 1) should be performed if V_{DDA} supply voltage changes.
- Temperature change affects the calibration accuracy to a much lesser extent (10°C change results in 0.2ppm offset error drift and 0.5ppm gain error drift).;
- For gain settings > 1, the PGA has reduced sensitivity to supply changes than the modulator (28ppm over the supply range). However, it is still comparable to the device's Electrical Characteristics table specification. Refer to the device data sheet for the electrical characteristics table.

Therefore, it is a good idea to recalibrate in the unlikely case that the supply voltage changes from the minimum V_{DDA} to the maximum V_{DDA} and vice versa.

Note: Calibration is done at the currently selected data rate, so for best results, set the data rate to a value equal to or lower than the lowest rate used for conversions.

16.16 GPIOs

Two general-purpose digital I/O increase the ADC's flexibility. When used as an output, a GPIO can be used as a microcontroller interrupt if connected externally, a control signal for a multiplexer or multichannel switch, or a modulator clock output. GPIO pins configured as outputs operate on the V_{DDA} rail. Care should be taken when using the GPIO pins in input mode to avoid bringing the signal above V_{DDA} + 0.3V.

A GPIO can be used as an ADC start control or a sequence start control when configured as an input. GPIO pins configured as inputs accept inputs at V_{DDIO} levels (not to exceed V_{DDA}).

The GPIO ports are configurable with the *AFE_ADC_n_GP0_CTRL* and *AFE_ADC_n_GP1_CTRL* registers. The registers select whether a GPIO is used as an input or as an output, and if used as an output, the output configuration (CMOS/open-drain).

16.16.1 Low-Side Power Switch

The GPIO pins can be configured to function as a low-side power switch with less than 35Ω on-resistance (25mA switch current) to reduce system power consumption in bridge sensor applications by powering down a bridge circuit between conversions.

16.16.1.1 Automatic Low-Side Switch Operation

Select automatic low-side switch operation by setting the respective GPIO direction and output select bits as follows.

GPIO0 (ADC0_RDY):

1. Set the GPIO0 direction field to output mode, open-drain (*AFE_ADC_n_GP0_CTRL*.*gp0_dir* = 0b10).
2. Set the GPIO0 output selection type to automatic low-side switch operation (*AFE_ADC_n_GP0_CTRL*.*gp0_osel* = 0b101).

GPIO1 (ADC1_RDY):

1. Set the GPIO1 direction field to output mode, open-drain (*AFE_ADC_n_GP1_CTRL*.*gp1_dir* = 0b10).
2. Set the GPIO1 output selection type to automatic low-side switch operation (*AFE_ADC_n_GP1_CTRL*.*gp1_osel* = 0b101).

16.16.1.2 Manual Low-Side Switch Operation

Manually control the low-side power switch by configuring a GPIO as an open-drain output, and switch between state logic 0 and logic 1:

GPIO0 (ADC0_RDY):

1. Set the GPIO0 direction field to output mode, open-drain (*AFE_ADC_n_GP0_CTRL*.*gp0_dir* = 0b10).
2. Set the GPIO0 output selection type to output state logic 0 (*AFE_ADC_n_GP0_CTRL*.*gp0_osel* = 0b011), switch closed.
3. Set the GPIO0 output selection type to output state logic 1 (*AFE_ADC_n_GP0_CTRL*.*gp0_osel* = 0b100), switch open.

GPIO1 (ADC1_RDY):

1. Set the GPIO1 direction field to output mode, open-drain (*AFE_ADC_n_GP1_CTRL*.*gp1_dir* = 0b10).
2. Set the GPIO1 output selection type to output state logic 0 (*AFE_ADC_n_GP1_CTRL*.*gp1_osel* = 0b011), switch closed.
3. Set the GPIO1 output selection type to output state logic 1 (*AFE_ADC_n_GP1_CTRL*.*gp1_osel* = 0b100), switch open.

16.17 Status

The *AFE_ADC_n_STATUS* register can be read to determine the state of the ADC and determine the cause of DOUT/INTB signal assertion. Therefore, most *AFE_ADC_n_STATUS* register bits are cleared by an *AFE_ADC_n_STATUS* register read.

The *AFE_ADC_n_STATUS*.*tor* bits are threshold register over-range status bits. When one is set, it indicates that the corresponding *AFE_ADC_n_DATA0*:*AFE_ADC_n_DATA7* register value is greater than the value set by the *AFE_ADC_n_UTHRESH0*:*AFE_ADC_n_UTHRESH7* register or the ADC conversion result has created a digital under-range condition. *AFE_ADC_n_STATUS*.*tor* is cleared when the *AFE_ADC_n_STATUS* register is read. *AFE_ADC_n_STATUS*.*tor* register bits do not self-clear.

The *AFE_ADC_n_STATUS.tur* bits are threshold register under-range status bits. When one is set, it indicates that the corresponding *AFE_ADC_n_DATA0*:*AFE_ADC_n_DATA7* register value is less than the value set by the *AFE_ADC_n_UTHRESH0*:*AFE_ADC_n_UTHRESH7* register or the ADC conversion result has created a digital under-range condition. *AFE_ADC_n_STATUS.tur* is cleared when the *AFE_ADC_n_STATUS* register is read. *AFE_ADC_n_STATUS.tur* register bits do not self-clear.

The system gain over-range status bit (*AFE_ADC_n_STATUS.sysgor*) indicates that a system gain calibration was over-range. The *AFE_ADC_n_STATUS.sysgor* calibration coefficient has a maximum value of 1.9999999 (0xFFFFFF). When set to 1, *AFE_ADC_n_STATUS.sysgor* indicates that the full-scale value out of the converter is likely not available. *AFE_ADC_n_STATUS.sysgor* is cleared when the *AFE_ADC_n_STATUS* register is read or if a new system gain calibration yields a valid result.

The *AFE_ADC_n_STATUS.data_rdy* bit indicates that the DATA registers contain unread ADC conversion results. This bit is cleared when all unread *AFE_ADC_n_DATA0*:*AFE_ADC_n_DATA7* registers have been read. Unlike other status bits, *AFE_ADC_n_STATUS.data_rdy* is not cleared by an *AFE_ADC_n_STATUS* register read. The *AFE_ADC_n_STATUS.data_rdy* status bit is a logical OR of 8 internal register status bits that are set when new ADC data is written to an *AFE_ADC_n_DATA0*:*AFE_ADC_n_DATA7* register and are cleared when the corresponding *AFE_ADC_n_DATA0*:*AFE_ADC_n_DATA7* register is read.

16.17.1 Status Interrupt Enable

The *AFE_ADC_n_STATUS_IE* register enables or disables status events from appearing as a logic OR of the INT signal state. For every *AFE_ADC_n_STATUS* register bit, there is a corresponding *AFE_ADC_n_STATUS_IE* bit. This register allows the INT signal to be used as a system interrupt for one or more system status sources. Writing a 1 to a bit causes the corresponding *AFE_ADC_n_STATUS* bit state to assert an interrupt. The specific cause of the interrupt can be discerned by reading the *AFE_ADC_n_STATUS* register. An interrupt can be masked by disabling its corresponding enable bit in this register.

The default value of this register is 0x000001, enabling only *AFE_ADC_n_STATUS.conv_rdy* interrupt by default.

16.18 Conversion Data Formats

The conversion data format is selected by the *AFE_ADC_n_CTRL.format* and *AFE_ADC_n_CTRL.u_bn* bits, as shown in *Table 16-8*. The unipolar/bipolar select (*AFE_ADC_n_CTRL.u_bn*) bit selects whether the input range is bipolar or unipolar. A '1' in this bit location selects unipolar input range, and a '0' selects bipolar input range. The format select (*AFE_ADC_n_CTRL.format*) bit controls the data format when in bipolar mode (*AFE_ADC_n_CTRL.u_bn* =0). Unipolar data is always in straight binary format. The *AFE_ADC_n_CTRL.format* bit has no effect in unipolar mode (*AFE_ADC_n_CTRL.u_bn* = 1). In bipolar mode, if the *AFE_ADC_n_CTRL.format* bit = 1, then the data format is offset binary. If the *AFE_ADC_n_CTRL.format* bit = 0, then the data format is two's complement.

Table 16-8: Conversion Data Formats

Mode	**Bipolar Mode**			**Unipolar Mode**	
AFE_ADC_n_CTRL.format		1	0		x
AFE_ADC_n_CTRL.u_bn		0	0		1
Code Description	Input Voltage ($V_{AINP} – V_{AINN}$)	Offset Binary	2's Complement	Input Voltage (V_{AINP} - V_{AINN})	Straight Binary (Unipolar Mode)
Positive Full Scale	$\geq V_{REF}$	0xFFFFFF	0x7FFFFF	$\geq V_{REF}$	0xFFFFFF
Positive FS - 1 LSB	$V_{REF}(1-1/(2^{23}-1))$	0xFFFFFE	0x7FFFFE	$V_{REF}(1-1/(2^{24}-1))$	0xFFFFFE
Positive Mid-Scale	$V_{REF}(1+1/(2^{23}-1))/2$	0xC00000	0x400000	$V_{REF}(1+1/(2^{24}-1))/2$	0x800000
Positive 1 LSB	$V_{REF}/(2^{23} – 1)$	0x800001	0x000001	$V_{REF}/(2^{24} – 1)$	0x000001
	0V	0x800000	0x000000	0V	0x000000
Negative 1 LSB	$-V_{REF}/(2^{23} – 1)$	0x7FFFFF	0xFFFFFF	< 0V	0x000000
Negative FS + 1 LSB	$-V_{REF}$	0x000001	0x800001	< 0V	0x000000
Negative FS	$-V_{REF}(1+1/(2^{23} – 1))$	0x000000	0x800000	< 0V	0x000000

16.19 Digital Filter

The configurable digital filter has selectable notch frequencies (50Hz and 60Hz, 50Hz, 60Hz, or SINC4) and selectable data rates. The filter rejection and frequency response is determined by the *AFE_ADC_n_FILTER.linef* and *AFE_ADC_n_FILTER.rate* field settings.

The simultaneous 50Hz/60Hz rejection FIR filter provides well over 90dB rejection of 50Hz and 60Hz at 16sps and significant rejection of their harmonics. The 50Hz and 60Hz FIR filter settings provide a lower level of attenuation for those frequencies but at a faster conversion time than available with the simultaneous 50Hz/60Hz FIR filter. The SINC4 setting enables a 4th-order SINC filter that can operate at continuous data rates up to 1920sps, with the first notch at the continuous data rate. The *AFE_ADC_n_FILTER.linef* and the *AFE_ADC_n_FILTER.rate* settings determine the conversion rate.

Note: The data rate for a given RATE setting is determined by the type of conversion selected in the AFE_ADC_n_CONV_START or AFE_ADC_n_GP_CONV register, based on a nominal clock period of 2.456MHz. In continuous conversion mode with AFE_ADC_n_FILTER.linef = 0b11, the digital filter has a settling time of 4× the sample rate. The first sample is not available until the expiration of that settling time. Subsequent samples are available at the listed sample rate. The filter sample rate is determined by the combination of AFE_ADC_n_FILTER.linef and AFE_ADC_n_FILTER.rate settings and the type of conversion launched by the AFE_ADC_n_CONV_START command. Data rates and rejection specifications for all settings are summarized below.

Table 16-9: AFE_ADC_n_FILTER.linef = 0b00 Data Rate and Filter Rejection Settings

AFE_ADC_n_FILTER.rate	**Filter Type**	**Rejection (Hz)**	**Data Rate (SPS)**		
			Single Cycle	**Continuous**	**Duty Cycle**
0b0000	FIR50/60	50/60	1.0	1.1	0.3
0b0001	FIR50/60	50/60	2.0	2.1	0.5
0b0010	FIR50/60	50/60	4.0	4.2	1.1
0b0011	FIR50/60	50/60	8.0	8.4	2.1
0b0100 – 0b1111	FIR50/60	50/60	16.0	16.8	4.2

Table 16-10: AFE_ADC_n_FILTER.linef = 0b01 Data Rate and Filter Rejection Settings

AFE_ADC_n_FILTER.rate	**Filter Type**	**Rejection (Hz)**	**Data Rate (SPS)**		
			Single Cycle	**Continuous**	**Duty Cycle**
0b0000	FIR50	50	1.3	1.3	0.3
0b0001	FIR50	50	2.5	2.7	0.7
0b0010	FIR50	50	5.0	5.3	1.3
0b0011	FIR50	50	10.0	10.7	2.7
0b0100	FIR50	50	20.0	21.3	5.3
0b0101 – 0b1111	FIR50	50	35.6	40.0	10.0

Table 16-11: AFE_ADC_n_FILTER.linef = 0b10 Data Rate and Filter Rejection Settings

AFE_ADC_n_FILTER.rate	**Filter Type**	**Rejection (Hz)**	**Data Rate (SPS)**		
			Single Cycle	**Continuous**	**Duty Cycle**
0b0000	FIR60	60	1.3	1.3	0.3
0b0001	FIR60	60	2.5	2.7	0.7
0b0010	FIR60	60	5.0	5.3	1.3
0b0011	FIR60	60	10.0	10.7	2.7
0b0100	FIR60	60	20.0	21.3	5.3
0b0101 – 0b1111	FIR60	60	35.6	40.0	10.0

Table 16-12: AFE_ADC_n_FILTER.linef = 0b11 Data Rate and Filter Rejection Settings

AFE_ADC_n_FILTER.rate	**Filter Type**	**Rejection (Hz)**	**Data Rate (SPS)**		
			Single Cycle	**Continuous**	**Duty Cycle**
0b0000	SINC4	4	1	4	0.25
0b0001	SINC4	10	2.5	10	0.63
0b0010	SINC4	20	5	20	1.25
0b0011	SINC4	40	10	40	2.5
0b0100	SINC4	60	15	60	5
0b0101	SINC4	120	30	120	10
0b0110	SINC4	240	60	240	15
0b0111	SINC4	480	120	480	30
0b1000	SINC4	960	240	960	60
0b1001	SINC4	1,920	480	1,920	120
0b1010	SINC4	3,840	960	3,840	240
0b1011	SINC4	7,680	3,840	15,360	960
0b1100 – 0b1111	SINC4	N/A	N/A	N/A	N/A

16.20 Temperature Sensor

The ADCs support an internal temperature sensor. The converted value read from the ADC after selecting the temperature sensor input can be converted to temperature using *Equation 16-1*.

Equation 16-1: Temperature Conversion Formula

$$\mathrm{T}(°\mathrm{K}) = \frac{300 \times V_{REF} \times code}{0.5626 \times FS\_CODE}$$

Where FS_CODE is the full-scale output of the ADC.

Perform the following steps to measure the temperature using one of the ADCs:

1. Select the ADC peripheral to use for the temperature conversion if not already selected. See *Selecting an AFE Peripheral* for details.
2. Configure the ADC reference by configuring the *AFE_ADC_n_CTRL* register and perform an 8-bit SPI write to the *AFE_ADC_n_CTRL* address.
3. Set the ADC's power mode to normal by writing 0 to the *AFE_ADC_n_PD* register, performing an 8-bit SPI write to the *AFE_ADC_n_PD* address.
4. Select the temperature sensor inputs for the ADC mux. Set *AFE_ADC_n_MUX_CTRL0.ainp_sel* to 0xC and *AFE_ADC_n_MUX_CTRL0.ainn_sel* to 0xD. Write the register value (0xCD) to the *AFE_ADC_n_MUX_CTRL0*'s address using an 8-bit SPI write.
5. Set the ADC filter to the default value by writing 0 to the *AFE_ADC_n_FILTER* address using an 8-bit SPI write.
6. Enable the temperature sensor by writing 1 to the *AFE_ADC_n_TS_CTRL.ts_en* field using an 8-bit SPI write.
7. Perform an 8-bit SPI read of the *AFE_ADC_n_TS_CTRL* register until the *ts_intg_rdy* field reads 1.
8. Perform an ADC conversion by setting the *AFE_ADC_n_CONV_START.dest* field to the desired data conversion destination and setting the *AFE_ADC_n_CONV_START.conv_type* field to 0. Perform an 8-bit SPI write with the value of the *AFE_ADC_n_CONV_START* register.
9. Perform a 24-bit SPI read of the *AFE_ADC_n_STATUS* register until *AFE_ADC_n_STATUS.data_rdy* reads 1.
10. Read the selected *AFE_ADC_n_DATA* register by performing a 24-bit SPI read.
11. Convert the data read using *Equation 16-1*.

16.21 Sequencer

The sequencer is a powerful feature that allows a sequence of commands to be programmed into the sequence buffer (*AFE_ADC_n_UC_0* - *AFE_ADC_n_UC_52*) registers. When a sequence is initiated by a write to the *AFE_ADC_n_SEQ_START* register or (when configured) a rising edge on an ADC GPIO pin, the sequencer serially executes commands as if it were the application software writing those commands to the control registers. Thus, the initiated sequence begins executing at the address in the *AFE_ADC_n_SEQ_START.seq_address* or *AFE_ADC_n_GP_SEQ_ADDR*, depending on which is selected. Sequences execute until a PD command is encountered or until the sequencer is interrupted by a write from the application software through the AFE. If no PD command is encountered, the sequence executes in a loop and wraps around from *AFE_ADC_n_UC_52* → *AFE_ADC_n_UC_0*.

All *AFE_ADC_n_UC* registers are set to 0 by default, which corresponds to a '*AFE_ADC_n_PD.pd* = Normal' command. *AFE_ADC_n_PD* commands function as a sequence stop. The completion of a sequence can be configured to generate an interrupt through the *AFE_ADC_n_STATUS_IE.seq_rdy_ie* bit. A wraparound, an *AFE_ADC_n_PD* execution, or an *AFE_ADC_n_SEQ_START* inside a sequence causes the assertion of *AFE_ADC_n_STATUS.seq_rdy*. As with a continuous conversion with *AFE_ADC_n_STATUS.conv_rdy*, executing the sequencer in a loop auto-clears *AFE_ADC_n_STATUS.seq_rdy* before re-asserting it.

Using an *AFE_ADC_n_SEQ_START* command within the sequencer microcode functions as a GOTO statement, enabling multiple continuous sequences to be programmed into the sequencer's *AFE_ADC_n_UC* register space. An *AFE_ADC_n_CONV_START*, *AFE_ADC_n_CAL_START*, or *AFE_ADC_n_WAIT_START* command prevents the sequencer from advancing until the command is completed. Likewise, sequence timing can be controlled with an *AFE_ADC_n_WAIT_START* command. Wait durations should be programmed according to the settling time of the associated internal and external circuitry.

The currently executing microcode address and data can be read back through the read-only *AFE_ADC_n_UCADDR* register. The *AFE_ADC_n_UCADDR.ucaddr* can be read back at any time to determine the currently executing microcode address. A read of 0x00 indicates that the sequencer is inactive. Values of 0x3A-0x6E indicate an active sequence.

Active sequences are exited by a write to any register, resetting the *AFE_ADC_n_UCADDR* register to 0x00. Launching a sequence does not reset the control registers. All register states are retained, either as a result of a prior write or a prior sequence execution.

16.21.1 Sequencer Notes

1. Registers with 24-bit data operands, including *AFE_ADC_n_UTHRESH*, *AFE_ADC_n_LTHRESH*, *AFE_ADC_n_SELF_OFF*, *AFE_ADC_n_STATUS_IE*, amongst others, are not supported in sequencer mode. Programming a register with a 24-bit operand into an *AFE_ADC_n_UC* register results in a '0000' or '*AFE_ADC_n_PD*' being written to the register.
2. Write an *AFE_ADC_n_UC* address to an *AFE_ADC_n_UC* register results in a '0000' or '*AFE_ADC_n_PD*' being written to the register.

16.21.2 Sequencer Example

Table 16-13 shows a populated sequence buffer. Three *AFE_ADC_n_SEQ_START* examples are discussed.

Example 1: The interface executes *AFE_ADC_n_SEQ_START* in *AFE_ADC_n_UC_0*

The sequencer executes the commands shown (configure the input multiplexer, select the buffered signal path, wait, convert and store in location 1, configure the input multiplexer, wait, convert and store in data location 2, initiate a power down, issue an *AFE_ADC_n_STATUS.seq_rdy* status, and halt the sequence at register *AFE_ADC_n_UC_7*.

Example 2: The interface executes *AFE_ADC_n_SEQ_START* in *AFE_ADC_n_UC_8*

The sequencer executes the commands starting at *AFE_ADC_n_UC_8* and continuing through *AFE_ADC_n_UC_48* in a loop until the sequencer is interrupted with a write to the interface through the AFE. This sequence configures the input multiplexer for various combinations of AIN0 through AIN7, performing a conversion with a variety of PGA settings, storing the results in *AFE_ADC_n_DATA0* through *AFE_ADC_n_DATA7*. Finally, an *AFE_ADC_n_STATUS.seq_rdy* is asserted at the end of the sequence (*AFE_ADC_n_UC_48*) and deasserted when the sequence starts again (*AFE_ADC_n_UC_8*).

Example 3: The interface executes *AFE_ADC_n_SEQ_START* in *AFE_ADC_n_UC_49*

The sequencer executes the commands shown from address *AFE_ADC_n_UC_49* (program the input multiplexer and filter, wait, perform a self-calibration) and wraparound continuing execution at *AFE_ADC_n_UC_0*. Ultimately, the sequencer initiates a power down, issues an *AFE_ADC_n_STATUS.seq_rdy* status, and halts the sequencer at register *AFE_ADC_n_UC_7*.

Table 16-13: Populated Sequencing Buffer Example

Sequencer Register	Sequencer Address	Command Address Bits 15:8	Command Name	Command Data Bits 7:0	Comments
AFE_ADC_n_UC_0	0x3A	0x0B	*AFE_ADC_n_MUX_CTRL0*	0x01	Select AINP = AIN0, AINN = AIN1.
AFE_ADC_n_UC_1	0x3B	0x0E	*AFE_ADC_n_PGA*	0x00	Select buffered input, gain = 1.
AFE_ADC_n_UC_2	0x3C	0x10	*AFE_ADC_n_WAIT_START*	0xD0	Insert wait time of: *AFE_ADC_n_WAIT_EXT* × 16)× (*AFE_ADC_n_WAIT_EXT* × 16)×407ns. Assuming *AFE_ADC_n_WAIT_EXT* = 0, wait time = 1.3ms.
AFE_ADC_n_UC_3	0x3D	0x01	*AFE_ADC_n_CONV_START*	0x10	Initiate a single conversion and send data to the *AFE_ADC_n_DATA1* register.
AFE_ADC_n_UC_4	0x3E	0x0B	*AFE_ADC_n_MUX_CTRL0*	0x23	Select AINP = AIN2, AINN = AIN3.
AFE_ADC_n_UC_5	0x3F	0x10	*AFE_ADC_n_WAIT_START*	0xD0	Insert wait time of 1.3ms (assuming *AFE_ADC_n_WAIT_EXT* = 0).
AFE_ADC_n_UC_6	0x40	0x01	*AFE_ADC_n_CONV_START*	0x20	Initiate a single conversion and send data to the *AFE_ADC_n_DATA2* register.
AFE_ADC_n_UC_7	0x41	0x00	*AFE_ADC_n_PD*	0x10	Enter Sleep Mode, issue *AFE_ADC_n_STATUS.seq_rdy* status, halt sequence.
AFE_ADC_n_UC_8	0x42	0x0E	*AFE_ADC_n_PGA*	0x21	Select PGA, Gain = 2.
AFE_ADC_n_UC_9	0x43	0x0B	*AFE_ADC_n_MUX_CTRL0*	0x01	Select AINP = AIN0, AINN = AIN1.
AFE_ADC_n_UC_10	0x44	0x10	*AFE_ADC_n_WAIT_START*	0xD0	Insert wait time of 1.3ms (assuming *AFE_ADC_n_WAIT_EXT* = 0).
AFE_ADC_n_UC_11	0x45	0x08	*AFE_ADC_n_FILTER*	0x04	Select 50/60Hz rejection, 16sps.

Sequencer Register	Sequencer Address	Command Address Bits 15:8	Command Name	Command Data Bits 7:0	Comments
AFE_ADC_n_UC_12	0x46	0x01	*AFE_ADC_n_CONV_START*	0x00	Initiate a single conversion and send data to the *AFE_ADC_n_DATA0* register.
AFE_ADC_n_UC_13	0x47	0x0E	*AFE_ADC_n_PGA*	0x22	Select PGA, Gain = 4.
AFE_ADC_n_UC_14	0x48	0x0B	*AFE_ADC_n_MUX_CTRL0*	0x23	Select AINP = AIN2, AINN = AIN3.
AFE_ADC_n_UC_15	0x49	0x10	*AFE_ADC_n_WAIT_START*	0xD0	Insert wait time of 1.3ms (assuming *AFE_ADC_n_WAIT_EXT* = 0).
AFE_ADC_n_UC_16	0x4A	0x08	*AFE_ADC_n_FILTER*	0x04	Select 50/60Hz rejection, 16sps.
AFE_ADC_n_UC_17	0x4B	0x01	*AFE_ADC_n_CONV_START*	0x10	Initiate a single conversion and send data to the *AFE_ADC_n_DATA1* register.
AFE_ADC_n_UC_18	0x4C	0x0E	*AFE_ADC_n_PGA*	0x20	Select PGA, Gain = 1.
AFE_ADC_n_UC_19	0x4D	0x0B	*AFE_ADC_n_MUX_CTRL0*	0x45	Select AINP = AIN4, AINN = AIN5.
AFE_ADC_n_UC_20	0x4E	0x10	*AFE_ADC_n_WAIT_START*	0xD0	Insert wait time of 1.3ms (assuming *AFE_ADC_n_WAIT_EXT* = 0).
AFE_ADC_n_UC_21	0x4F	0x08	*AFE_ADC_n_FILTER*	0x04	Select 50/60Hz rejection, 16sps.
AFE_ADC_n_UC_22	0x50	0x01	*AFE_ADC_n_CONV_START*	0x20	Initiate a single conversion and send data to the *AFE_ADC_n_DATA2* register.
AFE_ADC_n_UC_23	0x51	0x0E	*AFE_ADC_n_PGA*	0x02	Select buffered input, digital gain = 4.
AFE_ADC_n_UC_24	0x52	0x0B	*AFE_ADC_n_MUX_CTRL0*	0x03	Select AINP = AIN0, AINN = AIN3.
AFE_ADC_n_UC_25	0x53	0x10	*AFE_ADC_n_WAIT_START*	0xD0	Insert wait time of 1.3ms (assuming *AFE_ADC_n_WAIT_EXT* = 0).
AFE_ADC_n_UC_26	0x54	0x08	*AFE_ADC_n_FILTER*	0x04	Select 50/60Hz rejection, 16sps.
AFE_ADC_n_UC_27	0x55	0x01	*AFE_ADC_n_CONV_START*	0x30	Initiate a single conversion and send data to the *AFE_ADC_n_DATA3* register.
AFE_ADC_n_UC_28	0x56	0x0E	*AFE_ADC_n_PGA*	0x24	Select PGA, Gain = 16.
AFE_ADC_n_UC_29	0x57	0x0B	*AFE_ADC_n_MUX_CTRL0*	0x78	Select AINP = AIN7, AINN = AIN8.
AFE_ADC_n_UC_30	0x58	0x10	*AFE_ADC_n_WAIT_START*	0xD0	Insert wait time of 1.3ms (assuming *AFE_ADC_n_WAIT_EXT* = 0).
AFE_ADC_n_UC_31	0x59	0x08	*AFE_ADC_n_FILTER*	0x04	Select 50/60Hz rejection, 16sps.
AFE_ADC_n_UC_32	0x5A	0x01	*AFE_ADC_n_CONV_START*	0x40	Initiate a single conversion and send data to the *AFE_ADC_n_DATA4* register.
AFE_ADC_n_UC_33	0x5B	0x0E	*AFE_ADC_n_PGA*	0x22	Select PGA, Gain = 4.
AFE_ADC_n_UC_34	0x5C	0x0B	*AFE_ADC_n_MUX_CTRL0*	0x69	Select AINP = AIN6, AINN = AIN9.
AFE_ADC_n_UC_35	0x5D	0x10	*AFE_ADC_n_WAIT_START*	0xD0	Insert wait time of 1.3ms (assuming *AFE_ADC_n_WAIT_EXT* = 0).
AFE_ADC_n_UC_36	0x5E	0x08	*AFE_ADC_n_FILTER*	0x04	Select 50/60Hz rejection, 16sps.
AFE_ADC_n_UC_37	0x5F	0x01	*AFE_ADC_n_CONV_START*	0x50	Initiate a single conversion and send data to the *AFE_ADC_n_DATA5* register.
AFE_ADC_n_UC_38	0x60	0x0E	*AFE_ADC_n_PGA*	0x22	Select PGA, Gain = 4.
AFE_ADC_n_UC_39	0x61	0x0B	*AFE_ADC_n_MUX_CTRL0*	0x20	Select AINP = AIN2, AINN = AIN0.

Sequencer Register	Sequencer Address	Command Address Bits 15:8	Command Name	Command Data Bits 7:0	Comments
AFE_ADC_n_UC_40	0x62	0x10	*AFE_ADC_n_WAIT_START*	0xD0	Insert wait time of 1.3ms (assuming *AFE_ADC_n_WAIT_EXT* = 0).
AFE_ADC_n_UC_41	0x63	0x08	*AFE_ADC_n_FILTER*	0x04	Select 50/60Hz rejection, 16sps.
AFE_ADC_n_UC_42	0x64	0x01	*AFE_ADC_n_CONV_START*	0x60	Initiate a single conversion and send data to the *AFE_ADC_n_DATA6* register.
AFE_ADC_n_UC_43	0x65	0x0E	*AFE_ADC_n_PGA*	0x27	Select PGA, Gain = 128.
AFE_ADC_n_UC_44	0x66	0x0B	*AFE_ADC_n_MUX_CTRL0*	0x19	Select AINP = AIN1, AINN = AIN9.
AFE_ADC_n_UC_45	0x67	0x10	*AFE_ADC_n_WAIT_START*	0xD0	Insert wait time of 1.3ms (assuming *AFE_ADC_n_WAIT_EXT* = 0).
AFE_ADC_n_UC_46	0x68	0x08	*AFE_ADC_n_FILTER*	0x04	Select 50/60Hz rejection, 16sps.
AFE_ADC_n_UC_47	0x69	0x01	*AFE_ADC_n_CONV_START*	0x70	Initiate a single conversion and send data to the *AFE_ADC_n_DATA7* register.
AFE_ADC_n_UC_48	0x6A	0x02	*AFE_ADC_n_SEQ_START*	0x42	Loop back to sequencer address 0x42 and restart the sequence.
AFE_ADC_n_UC_49	0x6B	0x0B	*AFE_ADC_n_MUX_CTRL0*	0x26	Select AINP = AIN2, AINN = AIN6.
AFE_ADC_n_UC_50	0x6C	0x08	*AFE_ADC_n_FILTER*	0x04	Select 50/60Hz rejection, 16sps.
AFE_ADC_n_UC_51	0x6D	0x10	*AFE_ADC_n_WAIT_START*	0xD0	Insert wait time of 1.3ms (assuming *AFE_ADC_n_WAIT_EXT* = 0).
AFE_ADC_n_UC_52	0x6E	0x03	*AFE_ADC_n_CAL_START*	0x00	Perform a self-calibration. Wrap around to *AFE_ADC_n_UC_0* (0x3A) and continue.

16.22 ADC Registers

These registers are accessed through the *Analog Front-End (AFE)* using the internal SPI0 interface. See *Table 1-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

There are two instances of the ADC peripheral, and each instance has its own independent set of the registers shown in *Table 16-14*. Register names for a specific instance are defined by replacing "n" with the instance number's name. For example, a register AFE_ADC_*n*_PD resolves to AFE_ADC_ZERO_PD and AFE_ADC_ONE_PD for instances 0 and 1, respectively.

The ADC registers range in size from 8-bits to 24-bits wide. See the width column in *Table 16-14* for each register's width.

Table 16-14: 16-24-Bit Delta-Sigma ADC with PGA Registers

Address	Width	Name	Description
0x00	8 bits	*AFE_ADC_n_PD*	Power Down Register
0x01	8 bits	*AFE_ADC_n_CONV_START*	Conversion Start
0x02	8 bits	*AFE_ADC_n_SEQ_START*	Sequence Start
0x03	8 bits	*AFE_ADC_n_CAL_START*	Calibration Start
0x04	8 bits	*AFE_ADC_n_GP0_CTRL*	GPIO0 (ADC0_RDY) Control
0x05	8 bits	*AFE_ADC_n_GP1_CTRL*	GPIO1 (ADC1_RDY) Control
0x06	8 bits	*AFE_ADC_n_GP_CONV*	GPIO Conversion
0x07	8 bits	*AFE_ADC_n_GP_SEQ_ADDR*	GPIO Sequence Address Start
0x08	8 bits	*AFE_ADC_n_FILTER*	Conversion Rate and Filter

Address	Width	Name	Description
0x09	8 bits	*AFE_ADC_n_CTRL*	Control
0x0A	8 bits	*AFE_ADC_n_SOURCE*	Source
0x0B	8 bits	*AFE_ADC_n_MUX_CTRL0*	AINP and AINN Mux Control
0x0C	8 bits	*AFE_ADC_n_MUX_CTRL1*	Matched Excitation Source Select
0x0D	8 bits	*AFE_ADC_n_MUX_CTRL2*	V_{BIAS} Connection
0x0E	8 bits	*AFE_ADC_n_PGA*	PGA Control
0x0F	8 bits	*AFE_ADC_n_WAIT_EXT*	Wait Extension
0x10	8 bits	*AFE_ADC_n_WAIT_START*	Wait Start
0x11	24-bit	*AFE_ADC_n_PART_ID*	Part Identification
0x12	24-bit	*AFE_ADC_n_SYSC_SEL*	System Calibration Registers Select
0x13	24-bit	*AFE_ADC_n_SYS_OFF_A*	System Offset A Calibration
0x14	24-bit	*AFE_ADC_n_SYS_OFF_B*	System Offset B Calibration
0x15	24-bit	*AFE_ADC_n_SYS_GAIN_A*	System Gain A Calibration
0x16	24-bit	*AFE_ADC_n_SYS_GAIN_B*	System Gain B Calibration
0x17	24-bit	*AFE_ADC_n_SELF_OFF*	Self-Offset
0x18	24-bit	*AFE_ADC_n_SELF_GAIN_1*	Self-Gain 1x
0x19	24-bit	*AFE_ADC_n_SELF_GAIN_2*	Self-Gain 2x
0x1A	24-bit	*AFE_ADC_n_SELF_GAIN_4*	Self-Gain 4x
0x1B	24-bit	*AFE_ADC_n_SELF_GAIN_8*	Self-Gain 8x
0x1C	24-bit	*AFE_ADC_n_SELF_GAIN_16*	Self-Gain 16x
0x1D	24-bit	*AFE_ADC_n_SELF_GAIN_32*	Self-Gain 32x
0x1E	24-bit	*AFE_ADC_n_SELF_GAIN_64*	Self-Gain 64x
0x1F	24-bit	*AFE_ADC_n_SELF_GAIN_128*	Self-Gain 128x
0x20	24-bit	*AFE_ADC_n_LTHRESH0*	Lower Threshold 0 Register
0x21	24-bit	*AFE_ADC_n_LTHRESH1*	Lower Threshold 1 Register
0x22	24-bit	*AFE_ADC_n_LTHRESH2*	Lower Threshold 2 Register
0x23	24-bit	*AFE_ADC_n_LTHRESH3*	Lower Threshold 3 Register
0x24	24-bit	*AFE_ADC_n_LTHRESH4*	Lower Threshold 4 Register
0x25	24-bit	*AFE_ADC_n_LTHRESH5*	Lower Threshold 5 Register
0x26	24-bit	*AFE_ADC_n_LTHRESH6*	Lower Threshold 6 Register
0x27	24-bit	*AFE_ADC_n_LTHRESH7*	Lower Threshold 7 Register
0x28	24-bit	*AFE_ADC_n_UTHRESH0*	Upper Threshold 0 Register
0x29	24-bit	*AFE_ADC_n_UTHRESH1*	Upper Threshold 1 Register
0x2A	24-bit	*AFE_ADC_n_UTHRESH2*	Upper Threshold 2 Register
0x2B	24-bit	*AFE_ADC_n_UTHRESH3*	Upper Threshold 3 Register
0x2C	24-bit	*AFE_ADC_n_UTHRESH4*	Upper Threshold 4 Register
0x2D	24-bit	*AFE_ADC_n_UTHRESH5*	Upper Threshold 5 Register
0x2E	24-bit	*AFE_ADC_n_UTHRESH6*	Upper Threshold 6 Register
0x2F	24-bit	*AFE_ADC_n_UTHRESH7*	Upper Threshold 7 Register
0x30	24-bit	*AFE_ADC_n_DATA0*	Data 0 Register
0x31	24-bit	*AFE_ADC_n_DATA1*	Data 1 Register
0x32	24-bit	*AFE_ADC_n_DATA2*	Data 2 Register
0x33	24-bit	*AFE_ADC_n_DATA3*	Data 3 Register
0x34	24-bit	*AFE_ADC_n_DATA4*	Data 4 Register

Address	Width	Name	Description
0x35	24-bit	*AFE_ADC_n_DATA5*	Data 5 Register
0x36	24-bit	*AFE_ADC_n_DATA6*	Data 6 Register
0x37	24-bit	*AFE_ADC_n_DATA7*	Data 7 Register
0x38	24-bit	*AFE_ADC_n_STATUS*	Status Register
0x39	24-bit	*AFE_ADC_n_STATUS_IE*	Status Interrupt Enable Register
0x3A	16 bit	*AFE_ADC_n_UC_0*	Sequencer Register
0x3B	16 bit	*AFE_ADC_n_UC_1*	Sequencer Register
0x3C	16 bit	*AFE_ADC_n_UC_2*	Sequencer Register
0x3D	16 bit	*AFE_ADC_n_UC_3*	Sequencer Register
0x3E	16 bit	*AFE_ADC_n_UC_4*	Sequencer Register
0x3F	16 bit	*AFE_ADC_n_UC_5*	Sequencer Register
0x40	16 bit	*AFE_ADC_n_UC_6*	Sequencer Register
0x41	16 bit	*AFE_ADC_n_UC_7*	Sequencer Register
0x42	16 bit	*AFE_ADC_n_UC_8*	Sequencer Register
0x43	16 bit	*AFE_ADC_n_UC_9*	Sequencer Register
0x44	16 bit	*AFE_ADC_n_UC_10*	Sequencer Register
0x45	16 bit	*AFE_ADC_n_UC_11*	Sequencer Register
0x46	16 bit	*AFE_ADC_n_UC_12*	Sequencer Register
0x47	16 bit	*AFE_ADC_n_UC_13*	Sequencer Register
0x48	16 bit	*AFE_ADC_n_UC_14*	Sequencer Register
0x49	16 bit	*AFE_ADC_n_UC_15*	Sequencer Register
0x4A	16 bit	*AFE_ADC_n_UC_16*	Sequencer Register
0x4B	16 bit	*AFE_ADC_n_UC_17*	Sequencer Register
0x4C	16 bit	*AFE_ADC_n_UC_18*	Sequencer Register
0x4D	16 bit	*AFE_ADC_n_UC_19*	Sequencer Register
0x4E	16 bit	*AFE_ADC_n_UC_20*	Sequencer Register
0x4F	16 bit	*AFE_ADC_n_UC_21*	Sequencer Register
0x50	16 bit	*AFE_ADC_n_UC_22*	Sequencer Register
0x51	16 bit	*AFE_ADC_n_UC_23*	Sequencer Register
0x52	16 bit	*AFE_ADC_n_UC_24*	Sequencer Register
0x53	16 bit	*AFE_ADC_n_UC_25*	Sequencer Register
0x54	16 bit	*AFE_ADC_n_UC_26*	Sequencer Register
0x55	16 bit	*AFE_ADC_n_UC_27*	Sequencer Register
0x56	16 bit	*AFE_ADC_n_UC_28*	Sequencer Register
0x57	16 bit	*AFE_ADC_n_UC_29*	Sequencer Register
0x58	16 bit	*AFE_ADC_n_UC_30*	Sequencer Register
0x59	16 bit	*AFE_ADC_n_UC_31*	Sequencer Register
0x5A	16 bit	*AFE_ADC_n_UC_32*	Sequencer Register
0x5B	16 bit	*AFE_ADC_n_UC_33*	Sequencer Register
0x5C	16 bit	*AFE_ADC_n_UC_34*	Sequencer Register
0x5D	16 bit	*AFE_ADC_n_UC_35*	Sequencer Register
0x5E	16 bit	*AFE_ADC_n_UC_36*	Sequencer Register
0x5F	16 bit	*AFE_ADC_n_UC_37*	Sequencer Register
0x60	16 bit	*AFE_ADC_n_UC_38*	Sequencer Register

Address	Width	Name	Description
0x61	16 bit	*AFE_ADC_n_UC_39*	Sequencer Register
0x62	16 bit	*AFE_ADC_n_UC_40*	Sequencer Register
0x63	16 bit	*AFE_ADC_n_UC_41*	Sequencer Register
0x64	16 bit	*AFE_ADC_n_UC_42*	Sequencer Register
0x65	16 bit	*AFE_ADC_n_UC_43*	Sequencer Register
0x66	16 bit	*AFE_ADC_n_UC_44*	Sequencer Register
0x67	16 bit	*AFE_ADC_n_UC_45*	Sequencer Register
0x68	16 bit	*AFE_ADC_n_UC_46*	Sequencer Register
0x69	16 bit	*AFE_ADC_n_UC_47*	Sequencer Register
0x6A	16 bit	*AFE_ADC_n_UC_48*	Sequencer Register
0x6B	16 bit	*AFE_ADC_n_UC_49*	Sequencer Register
0x6C	16 bit	*AFE_ADC_n_UC_50*	Sequencer Register
0x6D	16 bit	*AFE_ADC_n_UC_51*	Sequencer Register
0x6E	16 bit	*AFE_ADC_n_UC_52*	Sequencer Register
0x6F	8 bits	*AFE_ADC_n_UCADDR*	Sequencer Register
0x77	24-bit	AFE_ADC_*n*_ADC_TRIM0	ADC Trim 0 Register
0x78	16 bit	AFE_ADC_*n*_ADC_TRIM1	ADC Trim 1 Register
0x79	16 bit	AFE_ADC_*n*_ANA_TRIM	Analog Trim 0 Register
0x7C	8 bits	*AFE_ADC_n_TS_CTRL*	Temperature Sensor Control Register

16.22.1 Register Details

Table 16-15: Power-Down Register

Power Down			**AFE_ADC_*n*_PD**	**0x00**
Bits	**Name**	**Access**	**Reset**	**Description**
7:2	-	R	0	**Reserved**
1:0	pd	R/W	1	**Power Down** This field selects the power-down state to be executed. While in a sequence, executing a power-down command causes the sequencer to stop and issue an *AFE_ADC_n_STATUS.seq_rdy* status. In standby or sleep mode, writing an asynchronous start command (*AFE_ADC_n_WAIT_START*, *AFE_ADC_n_CONV_START*, *AFE_ADC_n_SEQ_START*) initiates wake-up, causing the PD state to change to normal mode. During wake-up, the *AFE_ADC_n_PD.pd* field reads 0b10 and transitions to 0b00 after the wake-up timer expires. Asynchronous start operations are delayed until the wake-up timer expires. 0b00: Normal mode. 0b01: Standby mode - Powers down all analog circuitry, but not the internal voltage regulator. 0b10: Sleep mode – Powers down all analog circuitry, including the internal voltage regulator. 0b11: Reset – Resets all registers to the POR state. When complete, the ADC automatically enters standby mode.

Table 16-16: Start Conversion Register

Start Conversion				**AFE_ADC_*n*_CONV_START**	**0x01**
Bits	**Name**	**Access**	**Reset**	**Description**	
7	-	R	0	**Reserved**	
6:4	dest	R/W	0	**ADC Conversion Destination** This field selects the register to store the ADC conversion output. 0: *AFE_ADC_n_DATA0*. 1: *AFE_ADC_n_DATA1*. 2: *AFE_ADC_n_DATA2*. 3: *AFE_ADC_n_DATA3*. 4: *AFE_ADC_n_DATA4*. 5: *AFE_ADC_n_DATA5*. 6: *AFE_ADC_n_DATA6*. 7: *AFE_ADC_n_DATA7*. *Note: Writing to this register when the AFE_ADC_n_PD.pd field is set to sleep or standby automatically results in the AFE_ADC_n_PD.pd field being set to normal operation and initiates a conversion.*	
3:2	-	R	0	**Reserved**	
1:0	conv_type	R/W	0	**ADC Conversion Type** Selects the type of ADC conversion to perform. 0: Single Conversion. 1: Continuous conversions. 2 - 3: ¼ duty cycled conversions.	

Table 16-17: Sequencer Start Register

Sequencer Start				**AFE_ADC_*n*_SEQ_START**	**0x02**
Bits	**Name**	**Access**	**Reset**	**Description**	
7	-	R	0	**Reserved**	
6:0	seq_address	R/W	0	**Sequencer Start Address** Writing a valid address of a sequencer register to this field immediately executes a sequence beginning at the sequencer address written. If the ADC is set to sleep or standby mode, the *AFE_ADC_n_PD.pd* field is automatically set to 0 by hardware, and the sequence immediately executes. 0x3A – 0x6E: Execute sequence starting at the address written. *Note: Writing an address outside the sequencer's microcode address range is ignored.*	

Table 16-18: Calibration Start Register

Calibration Start				**AFE_ADC_*n*_CAL_START**	**0x03**
Bits	**Name**	**Access**	**Reset**	**Description**	
7:3	-	R	0	**Reserved**	

Calibration Start			AFE_ADC_*n*_CAL_START	0x03
Bits	**Name**	**Access**	**Reset**	**Description**
2:0	cal_type	R/W	0	**Calibration Type** Writing this field executes a calibration as selected by the calibration type selected. Successful completion of a calibration results in an update of the corresponding calibration value registers. All calibrations are performed at the filter settings in the *AFE_ADC_n_FILTER.linef* and *AFE_ADC_n_FILTER.rate* fields when the calibration is initiated. 0: Performs a self-calibration. The resulting offset calibration value is stored in the *AFE_ADC_n_SELF_OFF* register, and the 1x gain calibration value is stored in the *AFE_ADC_n_SELF_GAIN_1* register. 1: Performs a PGA gain calibration at the currently programmed PGA gain. A 'No Op' results if PGA gain calibration is executed with the PGA disabled through the *AFE_ADC_n_PGA.sig_path* field or with the *AFE_ADC_n_PGA.gain* set to 1x. The resulting gain calibration value is stored in the *AFE_ADC_n_SELF_GAIN_2*:*AFE_ADC_n_SELF_GAIN_128* register corresponding to the currently programmed PGA gain setting. 2, 3: Reserved. 4: Performs a system offset calibration. The resulting calibration value is storing the *AFE_ADC_n_SYS_OFF_A* register. 5: Performs a system gain calibration. The resulting calibration value is stored in the *AFE_ADC_n_SYS_GAIN_A* register. 6: Performs a system offset calibration. The resulting calibration is stored in the *AFE_ADC_n_SYS_OFF_B* register. 7: Performs a system gain calibration. The resulting calibration value is stored in the *AFE_ADC_n_SYS_GAIN_B* register. *Note: Writing to this register when the AFE_ADC_n_PD.pd field is set to sleep or standby automatically results in the AFE_ADC_n_PD.pd field being set to normal, and the calibration is performed.*

Table 16-19: GPIO0 (ADC0_RDY) Control Register

GPIO0 Control			AFE_ADC_*n*_GP0_CTRL	0x04
Bits	**Name**	**Access**	**Reset**	**Description**
7:6	gp0_dir	R/W	0	**GPIO0 Input/Output Selection** This field controls the ADC's GPIO0 (ADC0_RDY) input/output direction and type. 0b00: Input mode, reference to V_{DDIO}. 0b01: Reserved. 0b10: Output mode, open-drain output. 0b11: Output mode, CMOS output.
5:4	gp0_isel	R/W	0	**GPIO0 Input Select Type** When GPIO0 (ADC0_RDY) is configured as an input, *AFE_ADC_n_GP0_CTRL.gp0_dir* = 0, this field selects the usage of the input signal. 0b00: Input disabled. 0b01: Input configured as a rising-edge triggered conversion start. 0b10: Input configured as a rising-edge triggered sequence start. 0b11: Reserved.
3	-	R	0	**Reserved**

GPIO0 Control				AFE_ADC_*n*_GP0_CTRL	0x04
Bits	**Name**	**Access**	**Reset**	**Description**	
2:0	gp0_osel	R/W	0	**GPIO0 Output Select Type** When GPIO0 (ADC0_RDY) is configured as an output, *AFE_ADC_n_GP0_CTRL.gp0_dir* = 2 or 3, this field selects the usage of the output signal. 0b000: Output disabled, high Z. 0b001: Output is configured as an interrupt (active low). 0b010: Output is configured as an interrupt (active high). 0b011: Output is configured as state logic 0. 0b100: Output is configured as state logic 1. 0b101: Output is configured as an automatic low-side switch operation (CMOS output mode overridden). 0b110: Output is configured as modulator active status. 0b111: Reserved.	

Table 16-20: GPIO1 (ADC1_RDY) Control Register

GPIO1 Control				AFE_ADC_*n*_GP1_CTRL	0x05
Bits	**Name**	**Access**	**Reset**	**Description**	
7:6	gp1_dir	R/W	0	**GPIO1 Input/Output Selection** This field controls the ADC's GPIO1 (ADC1_RDY) input/output direction and type. 0b00: Input mode, reference to V_{DDIO}. 0b01: Reserved. 0b10: Output mode, open-drain output. 0b11: Output mode, CMOS output.	
5:4	gp1_isel	R/W	0	**GPIO1 Input Select Type** When GPIO1 (ADC1_RDY) is configured as an input, *AFE_ADC_n_GP1_CTRL.gp1_dir* = 0, this field selects the usage of the input signal. 0b00: Input disabled. 0b01: Input configured as a rising-edge triggered conversion start. 0b10: Input configured as a rising-edge triggered sequence start. 0b11: Reserved.	
3	-	R	0	**Reserved**	
2:0	gp1_osel	R/W	0	**GPIO1 Output Select Type** When GPIO1 (ADC1_RDY) is configured as an output, *AFE_ADC_n_GP1_CTRL.gp1_dir* = 2 or 3, this field selects the usage of the output signal. 0b000: Output disabled, high Z. 0b001: Output is configured as an interrupt (active low). 0b010: Output is configured as an interrupt (active high). 0b011: Output is configured as state logic 0. 0b100: Output is configured as state logic 1. 0b101: Output is configured as an automatic low-side switch operation (CMOS output mode overridden). 0b110: Output is configured as modulator active status. 0b111: Reserved.	

Table 16-21: GPIO Conversion Register

GPIO Conversion				AFE_ADC_*n*_GP_CONV / 0x06
Bits	**Name**	**Access**	**Reset**	**Description**
7	-	R	0	**Reserved**
6:4	dest	R/W	0	**Destination Register** When the GPIO0 is configured as a conversion start input, *AFE_ADC_n_GP0_CTRL.gp0_isel* = 2 or 3, this field selects the destination register to store the conversion result. 0b000: Store result in *AFE_ADC_n_DATA0*. 0b001: Store result in *AFE_ADC_n_DATA1*. 0b010: Store result in *AFE_ADC_n_DATA2*. 0b011: Store result in *AFE_ADC_n_DATA3*. 0b100: Store result in *AFE_ADC_n_DATA4*. 0b101: Store result in *AFE_ADC_n_DATA5*. 0b110: Store result in *AFE_ADC_n_DATA6*. 0b111: Store result in *AFE_ADC_n_DATA7*.
3:2	-	R	0	**Reserved**
1:0	conv_type	R/W	0	**Conversion Type** When the GPIO0 is configured as a conversion start input, *AFE_ADC_n_GP0_CTRL.gp0_isel* = 2 or 3, this field selects the type of conversion to perform. 0b00: Single conversion. 0b01: Continuous conversion. 0b10, 0b11: 1:4 duty cycled conversions (modulator low-power mode).

Table 16-22: GPIO Sequence Address Register

GPIO Sequence Address				AFE_ADC_*n*_GP_SEQ_ADDR / 0x07
Bits	**Name**	**Access**	**Reset**	**Description**
7	-	R	0	**Reserved**
6:0	gp_seq_addr	R/W	0	**GPIO Sequencer Address** Write the address of the sequencer register at which a sequence should be initiated by a sequencer start GPIO event. Writing to this register does not start a sequence. 0x3A – 0x6E: Execute sequence starting at the address written. *Note: Writing an address outside the sequencer's microcode address range is ignored.*

Table 16-23: Filter Register

Filter				AFE_ADC_*n*_FILTER / 0x08
Bits	**Name**	**Access**	**Reset**	**Description**
7:6	-	R	0	**Reserved**
5:4	linef	R/W	0	**Digital Filter Selection** This field sets the filter type. 0b00: Simultaneous 50/60Hz FIR rejection. 0b01: 50Hz FIR rejection. 0b10: 60Hz FIR rejection. 0b11: SINC4.

Filter				**AFE_ADC_*n*_FILTER**	**0x08**
Bits	**Name**	**Access**	**Reset**	**Description**	
3:0	rate	R/W	0	**Conversion Rate** The valid settings for this field are dependent on the filter selected with the *AFE_ADC_n_FILTER.linef* setting. See the tables in the *Digital Gain* and *Digital Filter* section for details.	

Table 16-24: Control Register

Control				**AFE_ADC_*n*_CTRL**	**0x09**
Bits	**Name**	**Access**	**Reset**	**Description**	
7	extclk	R/W	0	**External Clock Select** Setting this field to 1 enables the external clock input, EXT_CLK, as the ADC clock source. The default clock source is the internal ADC clock. 0: Internal ADC clock. 1: External ADC clock on alternate function EXT_CLK, P0.10.	
6	u_bn	R/W	0	**Unipolar/Bipolar Input Range** 0: Bipolar input range. 1: Unipolar input range.	
5	format	R/W	0	**Format** This field selects the format of the data conversion. 0: Two's complement format. It only applies when bipolar input range is enabled. 1: Offset binary format.	
4	refbufp_en	R/W	0	**Reference P-Side Buffer Select** 0: Power down the reference P-side buffer and bypass, driving the ADC reference input directly from the reference mux. 1: Enable the reference P-side buffer.	
3	refbufn_en	R/W	0	**Reference N-Side Buffer Select** 0: Power down the reference N-side buffer and bypass, driving the ADC reference input directly from the reference mux. 1: Enable the reference N-side buffer.	
2:0	ref_sel	R/W	1	**Reference Select** This field selects the reference for the ADC. 0b000: AIN0(Positive reference)/AIN1(Negative reference). 0b001: REF1P/REF1N. 0b010: REF0P/REF0N. 0b011: V_{DDA}/V_{SSA}. 0b100: AIN0(Positive reference)/V_{SSA} (single-ended mode). 0b101: REF1P/V_{SSA} (single-ended mode). 0b110: REF0P/V_{SSA} (single-ended mode). 0b111: Reserved.	

Table 16-25: Source Register

Source				AFE_ADC_*n*_SOURCE	0x0A
Bits	**Name**	**Access**	**Reset**	**Description**	
7:6	vbias_mode	R/W	0	**V_{BIAS} Mode Select** This field selects the operating mode for the $V_{DDA}/_2$ bias voltage source. The bias voltage may be supplied by an amplifier (Active mode) or a resistive device (either 125kΩ or 20kΩ source resistance). 0b00: Active mode. 0b01: High impedance; 125kΩ output impedance. 0b10: Low impedance; 20kΩ output impedance. 0b11: Low impedance; 20kΩ output impedance. *Note: Enabling the V_{BIAS} and IDAC sources on the same analog input is not supported. Enabling V_{BIAS} on an analog input with an IDAC enabled clears the corresponding IDAC enable.*	
5:4	brn_mode	R/W	0	**Burnout Current Source Select** This field selects the nominal current value for the burnout detection current source and sink. Three current values are available. 0b00: Powered down, burnout sources disabled. 0b01: 0.5µA burnout current sources enabled. 0b10: 1µA burnout current sources enabled. 0b11: 10µA burnout current sources enabled.	
3:0	idac_mode	R/W	0	**IDAC Matched Current Source Select** This field selects the nominal current output for the two matched excitation current sources. 0b0000: 10µA 0b0001: 50µA 0b0010: 75µA 0b0011: 100µA 0b0100: 125µA 0b0101: 150µA 0b0110: 175µA 0b0111: 200µA 0b1000: 225µA 0b1001: 250µA 0b1010: 300µA 0b1011: 400µA 0b1100: 600µA 0b1101: 800µA 0b1110: 1200µA 0b1111: 1600µA *Note: Enabling the V_{BIAS} and IDAC sources on the same analog input is not supported. Enabling IDAC on an analog input with a V_{BIAS} enabled clears the corresponding V_{BIAS} enable.*	

Table 16-26: Mux Control 0 Register

Mux Control 0				**AFE_ADC_*n*_MUX_CTRL0**	**0x0B**
Bits	**Name**	**Access**	**Reset**	**Description**	
7:4	ainp_sel	R/W	0b1111	**Positive Analog Input Connection** This field selects which analog input is connected to AINP. 0b0000: AINP = AIN0. 0b0001: AINP = AIN1. 0b0010: AINP = AIN2. 0b0011: AINP = AIN3. 0b0100: AINP = AIN4. 0b0101: AINP = AIN5. 0b0110: AINP = AIN6. 0b0111: AINP = AIN7. 0b1000: AINP = AIN8. 0b1001: AINP = AIN9. 0b1010: AINP = AIN10. 0b1011: AINP = AIN11. 0b1100: AINP = Temp Sense P. 0b1101: AINP = Temp Sense N. 0b1110: AINP = V_{DDA}. 0b1111: AINP = Unconnected. *Note: AINP defaults to unconnected.*	
3:0	ainn_sel	R/W	0b1111	**Negative Analog Input Connection** This field selects which analog input is connected to AINN. 0b0000: AINN = AIN0. 0b0001: AINN = AIN1. 0b0010: AINN = AIN2. 0b0011: AINN = AIN3. 0b0100: AINN = AIN4. 0b0101: AINN = AIN5. 0b0110: AINN = AIN6. 0b0111: AINN = AIN7. 0b1000: AINN = AIN8. 0b1001: AINN = AIN9. 0b1010: AINN = AIN10. 0b1011: AINN = AIN11. 0b1100: AINN = Temp Sense P. 0b1101: AINN = Temp Sense N. 0b1110: AINN = GND. 0b1111: AINN = Unconnected. *Note: AINN defaults to unconnected.*	

Table 16-27: Mux Control 1 Register

Mux Control 1			**AFE_ADC_*n*_MUX_CTRL1**	**0x0C**
Bits	**Name**	**Access**	**Reset**	**Description**
7:4	idac1_sel	R/W	0b1111	**IDAC1 Matched Current Source Select** This field selects which analog input the IDAC1 matched excitation current source is connected to or if it is powered down. 0b0000: AIN0. 0b0001: AIN1. 0b0010: AIN2. 0b0011: AIN3. 0b0100: AIN4. 0b0101: AIN5. 0b0110: AIN6. 0b0111: AIN7. 0b1000: AIN8. 0b1001: AIN9. 0b1010: AIN10. 0b1011: AIN11. 0b1100: Temp Sense P. 0b1101: Temp Sense N. 0b1110: V_{DDA}. 0b1111: Unconnected; IDAC1 powered down. *Note: IDAC1 defaults to unconnected.*
3:0	idac0_sel	R/W	0b1111	**IDAC0 Select** This field selects which analog input the IDAC0 matched excitation current source is connected to or if it is powered down. 0b0000: AIN0. 0b0001: AIN1. 0b0010: AIN2. 0b0011: AIN3. 0b0100: AIN4. 0b0101: AIN5. 0b0110: AIN6. 0b0111: AIN7. 0b1000: AIN8. 0b1001: AIN9. 0b1010: AIN10. 0b1011: AIN11. 0b1100: Temp Sense P. 0b1101: Temp Sense N. 0b1110: GND. 0b1111: Unconnected; IDAC0 powered down. *Note: IDAC0 defaults to unconnected.*

Table 16-28: Mux Control 2 Register

Mux Control 2				**AFE_ADC_*n*_MUX_CTRL2** / **0x0D**
Bits	**Name**	**Access**	**Reset**	**Description**
7	vbias_sel_7	R/W	0	**V_{BIAS} Connection to AIN7** This field enables the connection of the V_{BIAS} source to the input mux. The V_{BIAS} input source can be connected to multiple inputs simultaneously, but only AIN0 to AIN7 are supported. 0: Unconnected. 1: V_{BIAS} connected to AIN7.
6	vbias_sel_6	R/W	0	**V_{BIAS} Connection to AIN6** This field enables the connection of the V_{BIAS} source to the input mux. The V_{BIAS} input source can be connected to multiple inputs simultaneously, but only AIN0 to AIN7 are supported. 0: Unconnected. 1: V_{BIAS} connected to AIN6.
5	vbias_sel_5	R/W	0	**V_{BIAS} Connection to AIN5** This field enables the connection of the V_{BIAS} source to the input mux. The V_{BIAS} input source can be connected to multiple inputs simultaneously, but only AIN0 to AIN7 are supported. 0: Unconnected. 1: V_{BIAS} connected to AIN5.
4	vbias_sel_4	R/W	0	**V_{BIAS} Connection to AIN4** This field enables the connection of the V_{BIAS} source to the input mux. The V_{BIAS} input source can be connected to multiple inputs simultaneously, but only AIN0 to AIN7 are supported. 0: Unconnected. 1: V_{BIAS} connected to AIN4.
3	vbias_sel_3	R/W	0	**V_{BIAS} Connection to AIN3** This field enables the connection of the V_{BIAS} source to the input mux. The V_{BIAS} input source can be connected to multiple inputs simultaneously, but only AIN0 to AIN7 are supported. 0: Unconnected. 1: V_{BIAS} connected to AIN3.
2	vbias_sel_2	R/W	0	**V_{BIAS} Connection to AIN2** This field enables the connection of the V_{BIAS} source to the input mux. The V_{BIAS} input source can be connected to multiple inputs simultaneously, but only AIN0 to AIN7 are supported. 0: Unconnected. 1: V_{BIAS} connected to AIN2.
1	vbias_sel_1	R/W	0	**V_{BIAS} Connection to AIN1** This field enables the connection of the V_{BIAS} source to the input mux. The V_{BIAS} input source can be connected to multiple inputs simultaneously, but only AIN0 to AIN7 are supported. 0: Unconnected. 1: V_{BIAS} connected to AIN1.

Mux Control 2				AFE_ADC_*n*_MUX_CTRL2	0x0D
Bits	**Name**	**Access**	**Reset**	**Description**	
0	vbias_sel_0	R/W	0	**V_{BIAS} Connection to AIN0** This field enables the connection of the V_{BIAS} source to the input mux. The V_{BIAS} input source can be connected to multiple inputs simultaneously, but only AIN0 to AIN7 are supported. 0: Unconnected. 1: V_{BIAS} connected to AIN0.	

Table 16-29: PGA Register

PGA				AFE_ADC_*n*_PGA	0x0E
Bits	**Name**	**Access**	**Reset**	**Description**	
7:6	-	R	0	**Reserved**	
5:4	sig_path	R/W	0	**Signal Path** 0: Buffered, low-power, unity-gain path (PGA disabled, digital gain). 1: Bypass path (signal buffer disabled, PGA disabled, digital gain). 2: PGA path (signal buffer disabled, analog gain). 3: Reserved.	
3	-	R	0	**Reserved**	
2:0	gain	R/W	0	**Gain Selection** 0b000: 1x. 0b001: 2x. 0b010: 4x. 0b011: 8x. 0b100: 16x. 0b101: 32x. 0b110: 64x. 0b111: 128x.	

Table 16-30: Wait Extend Register

Wait Extend				AFE_ADC_*n*_WAIT_EXT	0x0E
Bits	**Name**	**Access**	**Reset**	**Description**	
7:0	wait_ext	R/W	0	**Wait Extension for Wait Command** This field extends the wait command as shown in the following equation. $Wait\ Clocks = (wait\_ext \times 16) \times (wait \times 16)$ For a 2.456MHz input clock, the minimum wait period is 6.5µs. The maximum wait period using the wait extender is 6.77s. *Note: Reading this register returns the written value. A write to this register does not cause a wait command to execute. When wait_ext = 0x00, no wait extension is applied, and the wait period is equal to wait × 16. In the absence of a reset, the wait_ext selection applies to all subsequence AFE_ADC_n_WAIT_START commands.*	

Table 16-31: Wait Start Register

Wait Start				**AFE_ADC_*n*_WAIT_START**	**0x10**
Bits	**Name**	**Access**	**Reset**	**Description**	
7:0	wait	R/W	0	**Wait Extension for Wait Command** A write to this register executes a wait operation with a clock count equal to: $Wait\ Clocks = (wait\_ext \times 16) \times (wait \times 16)$ For a 2.456MHz input clock, the minimum wait period is 6.5µs. The maximum wait period using the wait extender is 6.77s. *Note: Reading this register returns the current count value, as decremented since the last wait execution. Writing to any register during a wait aborts the count operation but does not reset the register. Writing 0x00 to this register results in a 'No Op' and no AFE_ADC_n_STATUS.wait_done status is issued.*	

Table 16-32: Part ID Register

Part ID				**AFE_ADC_*n*_PART_ID**	**0x11**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:3	-	R	0	**Reserved**	
2:0	rev_id	R	0	**Revision ID** This field returns the revision ID of the ADC.	

Table 16-33: System Calibration Select Register

Self-System Calibration Select				**AFE_ADC_*n*_SYSC_SEL**	**0x12**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:16	-	R	0	**Reserved**	
15:14	sysc_sel_7	R/W	0	**Calibration Select for Data Register 7** 0: *AFE_ADC_n_SYS_OFF_A* and *AFE_ADC_n_SYS_GAIN_A* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA7*. 1: *AFE_ADC_n_SYS_OFF_B* and *AFE_ADC_n_SYS_GAIN_B* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA7*. 2, 3: System calibration disabled for *AFE_ADC_n_DATA7*. (Only self-calibration is applied.)	
13:12	sysc_sel_6	R/W	0	**Calibration Select for Data Register 6** 0: *AFE_ADC_n_SYS_OFF_A* and *AFE_ADC_n_SYS_GAIN_A* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA6*. 1: *AFE_ADC_n_SYS_OFF_B* and *AFE_ADC_n_SYS_GAIN_B* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA6*. 2, 3: System calibration disabled for *AFE_ADC_n_DATA6*. (Only self-calibration is applied.)	
11:10	sysc_sel_5	R/W	0	**Calibration Select for Data Register 5** 0: *AFE_ADC_n_SYS_OFF_A* and *AFE_ADC_n_SYS_GAIN_A* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA5*. 1: *AFE_ADC_n_SYS_OFF_B* and *AFE_ADC_n_SYS_GAIN_B* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA5*. 2, 3: System calibration disabled for *AFE_ADC_n_DATA5*. (Only self-calibration is applied.)	

Self-System Calibration Select				**AFE_ADC_*n*_SYSC_SEL**	**0x12**
Bits	**Name**	**Access**	**Reset**	**Description**	
9:8	sysc_sel_4	R/W	0	**Calibration Select for Data Register 4** 0: *AFE_ADC_n_SYS_OFF_A* and *AFE_ADC_n_SYS_GAIN_A* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA4*. 1: *AFE_ADC_n_SYS_OFF_B* and *AFE_ADC_n_SYS_GAIN_B* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA4*. 2, 3: System calibration disabled for *AFE_ADC_n_DATA4*. (Only self-calibration is applied.)	
7:6	sysc_sel_3	R/W	0	**Calibration Select for Data Register 3** 0: *AFE_ADC_n_SYS_OFF_A* and *AFE_ADC_n_SYS_GAIN_A* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA3*. 1: *AFE_ADC_n_SYS_OFF_B* and *AFE_ADC_n_SYS_GAIN_B* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA3*. 2, 3: System calibration disabled for *AFE_ADC_n_DATA3*. (Only self-calibration is applied.)	
5:4	sysc_sel_2	R/W	0	**Calibration Select for Data Register 2** 0: *AFE_ADC_n_SYS_OFF_A* and *AFE_ADC_n_SYS_GAIN_A* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA2*. 1: *AFE_ADC_n_SYS_OFF_B* and *AFE_ADC_n_SYS_GAIN_B* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA2*. 2, 3: System calibration disabled for *AFE_ADC_n_DATA2*. (Only self-calibration is applied.)	
3:2	sysc_sel_1	R/W	0	**Calibration Select for Data Register 1** 0: *AFE_ADC_n_SYS_OFF_A* and *AFE_ADC_n_SYS_GAIN_A* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA1*. 1: *AFE_ADC_n_SYS_OFF_B* and *AFE_ADC_n_SYS_GAIN_B* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA1*. 2, 3: System calibration disabled for *AFE_ADC_n_DATA1*. (Only self-calibration is applied.)	
1:0	sysc_sel_0	R/W	0	**Calibration Select for Data Register 0** 0: *AFE_ADC_n_SYS_OFF_A* and *AFE_ADC_n_SYS_GAIN_A* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA0*. 1: *AFE_ADC_n_SYS_OFF_B* and *AFE_ADC_n_SYS_GAIN_B* calibration values are applied to the conversion result stored in *AFE_ADC_n_DATA0*. 2, 3: System calibration disabled for *AFE_ADC_n_DATA3*. (Only self-calibration is applied.)	

Table 16-34: System Offset A Register

Self-System Offset A				**AFE_ADC_*n*_SYS_OFF_A**	**0x13**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	sys_off_a	R/W	0	**System Offset A** The system offset A calibration value is subtracted from each conversion result if selected by the *AFE_ADC_n_SYSC_SEL* register. The data is always in 2's complement binary format and is unaffected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. Writes to this field are allowed. A value written to the register remains valid until either a new value is written or until an on-demand system calibration operation is performed, which overwrites the current value. The system offset calibration value applied to the selected destination register is subtracted from the conversion result after self-calibration but before the system gain correction. It is also applied before the 1x or 2x scale factor associated with bipolar and unipolar modes.	

Table 16-35: System Offset B Register

Self-System Offset B				**AFE_ADC_*n*_SYS_OFF_B**	**0x14**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	sys_off_b	R/W	0	**System Offset B** The system offset B calibration value is subtracted from each conversion result if selected by the *AFE_ADC_n_SYSC_SEL* register. The data is always in 2's complement binary format and is unaffected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. Writes to this field are allowed. A value written to the register remains valid until either a new value is written or until an on-demand system calibration operation is performed, which overwrites the current value. The system offset calibration value applied to the selected destination register is subtracted from the conversion result after self-calibration but before the system gain correction. It is also applied before the 1x or 2x scale factor associated with bipolar and unipolar modes.	

Table 16-36: System Gain A Register

Self-System Gain A			AFE_ADC_*n*_SYS_GAIN_A	0x15
Bits	**Name**	**Access**	**Reset**	**Description**
23:0	sys_gain_a	R/W	0	**System Gain A** The System Gain Calibration A value is used to scale the offset-corrected conversion result if selected by the *AFE_ADC_n_SYSC_SEL* register. The format is fixed point, unsigned binary, and is unaffected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. The binary point is located after the MSB. The MSB corresponds to 2^0, and the LSB corresponds to 2^{-23}. Writes to this register are allowed. A value written to the register remains valid until either a new value is written or until an on-demand system calibration operation is performed, which overwrites the current value. The system gain calibration value scales the offset corrected result by up to 1.999999881x or can correct a gain error of –50%. The amount of positive gain error that can be corrected is determined by modulator overload characteristics, which may be as much as +25%.

Table 16-37: System Gain B Register

Self-System Gain B			AFE_ADC_*n*_SYS_GAIN_B	0x16
Bits	**Name**	**Access**	**Reset**	**Description**
23:0	sys_gain_b	R/W	0	**System Gain B** The System Gain Calibration B value is used to scale the offset-corrected conversion result if selected by the *AFE_ADC_n_SYSC_SEL* register. The format is fixed point, unsigned binary, and is unaffected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. The binary point is located after the MSB. The MSB corresponds to 2^0, and the LSB corresponds to 2^{-23}. Writes to this register are allowed. A value written to the register remains valid until either a new value is written or until an on-demand system calibration operation is performed, which overwrites the current value. The system gain calibration value scales the offset corrected result by up to 1.999999881x or can correct a gain error of -50%. The amount of positive gain error that can be corrected is determined by modulator overload characteristics, which may be as much as +25%.

Table 16-38: Self-Calibration Offset Register

Self-Calibration Offset Value				**AFE_ADC_*n*_SELF_OFF**	**0x17**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	self_off	R/W	0	**Self-Calibration Offset** The self-calibration offset value, *self_off*, is subtracted from the conversion result. The format is always 2's complement binary format and is unaffected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. Writing to the self-calibration register is allowed. The value remains valid until either a new write is complete or an on-demand self-calibration operation is performed (*AFE_ADC_n_CAL_START.cal_type* = 0), which overwrites this field. The self-calibration offset value is subtracted from the conversion result before the selected self-calibration gain correction (*AFE_ADC_n_SELF_GAIN_1* to *AFE_ADC_n_SELF_GAIN_128*) and before the system offset (*AFE_ADC_n_SYS_OFF_A*/*AFE_ADC_n_SYS_OFF_B*) and gain (*AFE_ADC_n_SYS_GAIN_A*/*AFE_ADC_n_SYS_GAIN_B*). It is also applied before the 2x scale factor associated with unipolar mode.	

Table 16-39: Self-Gain 1x Register

Self-Gain 1x				**AFE_ADC_*n*_SELF_GAIN_1**	**0x18**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	gain	R/W	0	**Gain 1x**	

Table 16-40: Self-Gain 2x Register

Self-Gain 2x				**AFE_ADC_*n*_SELF_GAIN_2**	**0x19**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	gain	R/W	0	**Gain 2x**	

Table 16-41: Self-Gain 4x Register

Self-Gain 4x				**AFE_ADC_*n*_SELF_GAIN_4**	**0x1A**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	gain	R/W	0	**Gain 4x**	

Table 16-42: Self-Gain 8x Register

Self-Gain 8x				**AFE_ADC_*n*_SELF_GAIN_8**	**0x1B**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	gain	R/W	0	**Gain 8x**	

Table 16-43: Self-Gain 16x Register

Self-Gain 16x				**AFE_ADC_*n*_SELF_GAIN_16**	**0x1C**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	gain	R/W	0	**Gain 16x**	

Table 16-44: Self-Gain 32x Register

Self-Gain 32x		AFE_ADC_*n*_SELF_GAIN_32		0x1D
Bits	**Name**	**Access**	**Reset**	**Description**
23:0	gain	R/W	0	**Gain 32x**

Table 16-45: Self-Gain 64x Register

Self-Gain 64x		AFE_ADC_*n*_SELF_GAIN_64		0x1E
Bits	**Name**	**Access**	**Reset**	**Description**
23:0	gain	R/W	0	**Gain 64x**

Table 16-46: Self-Gain 128x Register

Self-Gain 128x		AFE_ADC_*n*_SELF_GAIN_128		0x1F
Bits	**Name**	**Access**	**Reset**	**Description**
23:0	gain	R/W	0	**Gain 128x**

Table 16-47: Lower Threshold 0 Register

Lower Threshold 0		AFE_ADC_*n*_LTHRESH0		0x20
Bits	**Name**	**Access**	**Reset**	**Description**
23:0	lthresh0	R/W	0	**Lower Threshold Data 0** This field holds the lower comparison threshold for the value in the *AFE_ADC_n_DATA0* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tur_0* field. The comparison result indicated by *AFE_ADC_n_STATUS.tur_0* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.

Table 16-48: Lower Threshold 1 Register

Lower Threshold 1		AFE_ADC_*n*_LTHRESH1		0x21
Bits	**Name**	**Access**	**Reset**	**Description**
23:0	lthresh1	R/W	0	**Lower Threshold Data 1** This field holds the lower comparison threshold for the value in the *AFE_ADC_n_DATA1* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tur_1* field. The comparison result indicated by *AFE_ADC_n_STATUS.tur_1* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.

Table 16-49: Lower Threshold 2 Register

Lower Threshold 2				AFE_ADC_*n*_LTHRESH2	0x22
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	lthresh2	R/W	0	**Lower Threshold Data 1** This field holds the lower comparison threshold for the value in the *AFE_ADC_n_DATA2* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tur_2* field. The comparison result indicated by *AFE_ADC_n_STATUS.tur_2* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-50: Lower Threshold 3 Register

Lower Threshold 3				AFE_ADC_*n*_LTHRESH3	0x23
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	lthresh3	R/W	0	**Lower Threshold Data 3** This field holds the lower comparison threshold for the value in the *AFE_ADC_n_DATA3* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tur_3* field. The comparison result indicated by *AFE_ADC_n_STATUS.tur_3* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-51: Lower Threshold 4 Register

Lower Threshold 4				AFE_ADC_*n*_LTHRESH4	0x24
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	lthresh4	R/W	0	**Lower Threshold Data 4** This field holds the lower comparison threshold for the value in the *AFE_ADC_n_DATA4* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tur_4* field. The comparison result indicated by *AFE_ADC_n_STATUS.tur_4* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-52: Lower Threshold 5 Register

Lower Threshold 5				AFE_ADC_*n*_LTHRESH5	0x25
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	lthresh5	R/W	0	**Lower Threshold Data 5** This field holds the lower comparison threshold for the value in the *AFE_ADC_n_DATA5* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tur_5* field. The comparison result indicated by *AFE_ADC_n_STATUS.tur_5* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-53: Lower Threshold 6 Register

Lower Threshold 6				**AFE_ADC_*n*_LTHRESH6**	**0x26**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	lthresh6	R/W	0	**Lower Threshold Data 6** This field holds the lower comparison threshold for the value in the *AFE_ADC_n_DATA6* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tur_6* field. The comparison result indicated by *AFE_ADC_n_STATUS.tur_6* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-54: Lower Threshold 7 Register

Lower Threshold 7				**AFE_ADC_*n*_LTHRESH7**	**0x27**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	lthresh7	R/W	0	**Lower Threshold Data 7** This field holds the lower comparison threshold for the value in the *AFE_ADC_n_DATA7* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tur_7* field. The comparison result indicated by *AFE_ADC_n_STATUS.tur_7* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-55: Upper Threshold 0 Register

Upper Threshold 0				**AFE_ADC_*n*_UTHRESH0**	**0x28**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	uthresh0	R/W	0	**Lower Threshold Data 0** This field holds the upper comparison threshold for the value in the *AFE_ADC_n_DATA0* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tor_0* field. The comparison result indicated by *AFE_ADC_n_STATUS.tor_0* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-56: Upper Threshold 1 Register

Upper Threshold 1				**AFE_ADC_*n*_UTHRESH1**	**0x29**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	uthresh1	R/W	0	**Lower Threshold Data 2** This field holds the upper comparison threshold for the value in the *AFE_ADC_n_DATA1* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tor_1* field. The comparison result indicated by *AFE_ADC_n_STATUS.tor_1* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-57: Upper Threshold 2 Register

Upper Threshold 2				**AFE_ADC_*n*_UTHRESH2**	**0x2A**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	uthresh2	R/W	0	**Lower Threshold Data 2** This field holds the upper comparison threshold for the value in the *AFE_ADC_n_DATA2* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tor_2* field. The comparison result indicated by *AFE_ADC_n_STATUS.tor_2* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-58: Upper Threshold 3 Register

Upper Threshold 3				**AFE_ADC_*n*_UTHRESH3**	**0x2B**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	uthresh3	R/W	0	**Lower Threshold Data 3** This field holds the upper comparison threshold for the value in the *AFE_ADC_n_DATA3* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tor_3* field. The comparison result indicated by *AFE_ADC_n_STATUS.tor_3* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-59: Upper Threshold 4 Register

Upper Threshold 4				**AFE_ADC_*n*_UTHRESH4**	**0x2C**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	uthresh4	R/W	0	**Lower Threshold Data 4** This field holds the upper comparison threshold for the value in the *AFE_ADC_n_DATA4* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tor_4* field. The comparison result indicated by *AFE_ADC_n_STATUS.tor_4* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-60: Upper Threshold 5 Register

Upper Threshold 5				**AFE_ADC_*n*_UTHRESH5**	**0x2D**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	uthresh5	R/W	0	**Lower Threshold Data 5** This field holds the upper comparison threshold for the value in the *AFE_ADC_n_DATA5* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tor_5* field. The comparison result indicated by *AFE_ADC_n_STATUS.tor_5* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-61: Upper Threshold 6 Register

Upper Threshold 6				**AFE_ADC_*n*_UTHRESH6**	**0x2E**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	uthresh6	R/W	0	**Lower Threshold Data 6** This field holds the upper comparison threshold for the value in the *AFE_ADC_n_DATA6* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tor_6* field. The comparison result indicated by *AFE_ADC_n_STATUS.tor_6* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-62: Upper Threshold 7 Register

Upper Threshold 7				**AFE_ADC_*n*_UTHRESH7**	**0x2F**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	uthresh7	R/W	0	**Lower Threshold Data 7** This field holds the upper comparison threshold for the value in the *AFE_ADC_n_DATA7* register. The comparison result is indicated by the *AFE_ADC_n_STATUS.tor_7* field. The comparison result indicated by *AFE_ADC_n_STATUS.tor_7* is affected by the *AFE_ADC_n_CTRL.u_bn* and *AFE_ADC_n_CTRL.format* fields. If the *AFE_ADC_n_CTRL.u_bn* or *AFE_ADC_n_CTRL.format* fields are changed, the threshold value should be changed accordingly.	

Table 16-63: Data 0 Register

Data 0				**AFE_ADC_*n*_DATA0**	**0x30**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	data	R	0	**ADC Data Conversion** This ADC conversion result is stored in this field if selected by either the *AFE_ADC_n_CONV_START.dest* field or the *AFE_ADC_n_GP_CONV.gp_dest* field.	

Table 16-64: Data 1 Register

Data 1				**AFE_ADC_*n*_DATA1**	**0x31**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	data	R	0	**ADC Data Conversion** This ADC conversion result is stored in this field if selected by either the *AFE_ADC_n_CONV_START.dest* field or the *AFE_ADC_n_GP_CONV.gp_dest* field.	

Table 16-65: Data 2 Register

Data 2				**AFE_ADC_*n*_DATA2**	**0x32**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	data	R	0	**ADC Data Conversion** This ADC conversion result is stored in this field if selected by either the *AFE_ADC_n_CONV_START.dest* field or the *AFE_ADC_n_GP_CONV.gp_dest* field.	

Table 16-66: Data 3 Register

Data 3				**AFE_ADC_*n*_DATA3**	**0x33**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	data	R	0	**ADC Data Conversion** This ADC conversion result is stored in this field if selected by either the *AFE_ADC_n_CONV_START.dest* field or the *AFE_ADC_n_GP_CONV.gp_dest* field.	

Table 16-67: Data 4 Register

Data 4				**AFE_ADC_*n*_DATA4**	**0x34**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	data	R	0	**ADC Data Conversion** This ADC conversion result is stored in this field if selected by either the *AFE_ADC_n_CONV_START.dest* field or the *AFE_ADC_n_GP_CONV.gp_dest* field.	

Table 16-68: Data 5 Register

Data 5				**AFE_ADC_*n*_DATA5**	**0x35**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	data	R	0	**ADC Data Conversion** This ADC conversion result is stored in this field if selected by either the *AFE_ADC_n_CONV_START.dest* field or the *AFE_ADC_n_GP_CONV.gp_dest* field.	

Table 16-69: Data 6 Register

Data 6				**AFE_ADC_*n*_DATA6**	**0x36**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	data	R	0	**ADC Data Conversion** This ADC conversion result is stored in this field if selected by either the *AFE_ADC_n_CONV_START.dest* field or the *AFE_ADC_n_GP_CONV.gp_dest* field.	

Table 16-70: Data 7 Register

Data 7				**AFE_ADC_*n*_DATA7**	**0x37**
Bits	**Name**	**Access**	**Reset**	**Description**	
23:0	data	R	0	**ADC Data Conversion** This ADC conversion result is stored in this field if selected by either the *AFE_ADC_n_CONV_START.dest* field or the *AFE_ADC_n_GP_CONV.gp_dest* field.	

Table 16-71: Status Register

Status				**AFE_ADC_*n*_STATUS**	**0x38**
Bits	**Name**	**Access**	**Reset**	**Description**	
23	tor_7	ROC	0	**Threshold Over-Range/Digital Over-Range Condition on Channel 7** 0: Normal operation. 1: Threshold over-range/digital over-range condition. *Note: This field is automatically cleared by reading the register.*	

Status				AFE_ADC_*n*_STATUS	0x38
Bits	**Name**	**Access**	**Reset**	**Description**	
22	tor_6	ROC	0	**Threshold Over-Range/Digital Over-Range Condition on Channel 6** 0: Normal operation. 1: Threshold over-range/digital over-range condition. *Note: This field is automatically cleared by reading the register.*	
21	tor_5	ROC	0	**Threshold Over-Range/Digital Over-Range Condition on Channel 5** 0: Normal operation. 1: Threshold over-range/digital over-range condition. *Note: This field is automatically cleared by reading the register.*	
20	tor_4	ROC	0	**Threshold Over-Range/Digital Over-Range Condition on Channel 4** 0: Normal operation. 1: Threshold over-range/digital over-range condition. *Note: This field is automatically cleared by reading the register.*	
19	tor_3	ROC	0	**Threshold Over-Range/Digital Over-Range Condition on Channel 3** 0: Normal operation 1: Threshold Overrange /digital over-range condition. *Note: This field is automatically cleared by reading the register.*	
18	tor_2	ROC	0	**Threshold Over-Range/Digital Over-Range Condition on Channel 2** 0: Normal operation. 1: Threshold over-range/digital over-range condition. *Note: This field is automatically cleared by reading the register.*	
17	tor_1	ROC	0	**Threshold Over-Range/Digital Over-Range Condition on Channel 1** 0: Normal operation. 1: Threshold over-range/digital over-range condition. *Note: This field is automatically cleared by reading the register.*	
16	tor_0	ROC	0	**Threshold Over-Range/Digital Over-Range Condition on Channel 0** 0: Normal operation. 1: Threshold over-range/digital over-range condition. *Note: This field is automatically cleared by reading the register.*	
15	tur_7	ROC	0	**Threshold Under-Range/Digital Under-Range Condition on Channel 7** 0: Normal operation. 1: Threshold under-range/digital under-range condition. *Note: This field is automatically cleared by reading this register.*	
14	tur_6	ROC	0	**Threshold Under-Range/Digital Under-Range Condition on Channel 6** 0: Normal operation. 1: Threshold under-range/digital under-range condition. *Note: This field is automatically cleared by reading this register.*	
13	tur_5	ROC	0	**Threshold Under-Range/Digital Under-Range Condition on Channel 5** 0: Normal operation. 1: Threshold under-range/digital under-range condition. *Note: This field is automatically cleared by reading this register.*	

Status				AFE_ADC_*n*_STATUS	0x38
Bits	**Name**	**Access**	**Reset**	**Description**	
12	tur_4	ROC	0	**Threshold Under-Range/Digital Under-Range Condition on Channel 4** 0: Normal operation. 1: Threshold under-range/digital under-range condition. *Note: This field is automatically cleared by reading this register.*	
11	tur_3	ROC	0	**Threshold Under-Range/Digital Under-Range Condition on Channel 3** 0: Normal operation. 1: Threshold under-range/digital under-range condition. *Note: This field is automatically cleared by reading this register.*	
10	tur_2	ROC	0	**Threshold Under-Range/Digital Under-Range Condition on Channel 2** 0: Normal operation. 1: Threshold under-range/digital under-range condition. *Note: This field is automatically cleared by reading this register.*	
9	tur_1	ROC	0	**Threshold Under-Range/Digital Under-Range Condition on Channel 1** 0: Normal operation. 1: Threshold under-range/digital under-range condition. *Note: This field is automatically cleared by reading this register.*	
8	tur_0	ROC	0	**Threshold Under-Range/Digital Under-Range Condition on Channel 0** 0: Normal operation. 1: Threshold under-range/digital under-range condition. *Note: This field is automatically cleared by reading this register.*	
7	sysgor	ROC	0	**System Gain Calibration Over-Range** 0: No fault detected. 1: A system gain calibration was over-range. *Note: This field is automatically cleared by reading this register.*	
6:5	-	RO	0	**Reserved**	
4	data_rdy	R	0	**Data Ready** This field indicates that one of the *AFE_ADC_n_DATA0*:*AFE_ADC_n_DATA7* registers contains an unread ADC conversion result. 0: No change. 1: Unread ADC data conversion data available.	
3	wait_done	ROC	0	**Wait Done** This field is set to 1 when a wait operation is complete. This field is cleared by reading this register or a write to the *AFE_ADC_n_WAIT_START* register.	
2	cal_rdy	ROC	0	**Calibration Complete** 0: No change. 1: Calibration complete. New calibration result(s) available in one of the system or self-calibration registers. *Note: This field is cleared by reading this register or a write to the AFE_ADC_n_CAL_START register.*	

Status				AFE_ADC_*n*_STATUS	0x38
Bits	**Name**	**Access**	**Reset**	**Description**	
1	seq_rdy	ROC	0	**Sequence Ready** This field indicates a sequence has completed at least one iteration. 0: No sequence completed, or status bit has been reset. 1: Sequence has completed at least one iteration. *Note: This field is cleared by reading this register, writing to the AFE_ADC_n_SEQ_START register (including within a sequence), or a sequence wraparound from AFE_ADC_n_UC_52 to AFE_ADC_n_UC_0.*	
0	conv_rdy	ROC	0	**Conversion Ready** This field is set to 1 when a new conversion result is available in the *AFE_ADC_n_DATA0*:*AFE_ADC_n_DATA7* registers.	

Table 16-72: Status Interrupt Enable Register

Status Interrupt Enable				AFE_ADC_*n*_STATUS_IE	0x39
Bits	**Name**	**Access**	**Reset**	**Description**	
23	tor_7	R/W	0	**Threshold Over-Range 7 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tor_7* = 1.	
22	tor_6	R/W	0	**Threshold Over-Range 6 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tor_6* = 1.	
21	tor_5	R/W	0	**Threshold Over-Range 5 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tor_5* = 1.	
20	tor_4	R/W	0	**Threshold Over-Range 4 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tor_4* = 1.	
19	tor_3	R/W	0	**Threshold Over-Range 3 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tor_3* = 1.	
18	tor_2	R/W	0	**Threshold Over-Range 2 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tor_2* = 1.	
17	tor_1	R/W	0	**Threshold Over-Range 1 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tor_1* = 1.	
16	tor_0	R/W	0	**Threshold Over-Range 0 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tor_0* = 1.	
15	tur_7	R/W	0	**Threshold Under-Range 7 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tur_7* = 1.	
14	tur_6	R/W	0	**Threshold Under-Range 6 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tur_6* = 1.	

Status Interrupt Enable			AFE_ADC_*n*_STATUS_IE	0x39
Bits	**Name**	**Access**	**Reset**	**Description**
13	tur_5	R/W	0	**Threshold Under-Range 5 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tur_5* = 1.
12	tur_4	R/W	0	**Threshold Under-Range 4 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tur_4* = 1.
11	tur_3	R/W	0	**Threshold Under-Range 3 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tur_3* = 1.
10	tur_2	R/W	0	**Threshold Under-Range 2 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tur_2* = 1.
9	tur_1	R/W	0	**Threshold Under-Range 1 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tur_1* = 1.
8	tur_0	R/W	0	**Threshold Under-Range 0 Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.tur_0* = 1.
7	sysgor_ie	R/W	0	**System Gain Calibration Over-Range Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.sysgor* = 1.
6:5	-	RO	0	**Reserved**
4	data_rdy_ie	R/W	0	**Data Ready Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.data_rdy* = 1.
3	wait_done_ie	R/W	0	**Wait Done Interrupt Enable** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.wait_done* = 1.
2	cal_rdy_ie	R/W	0	**Calibration Complete** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.cal_rdy* = 1.
1	seq_rdy_ie	R/W	0	**Sequence Ready** 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.seq_rdy* = 1.
0	conv_rdy_ie	R/W	1	Conversion Ready 0: Disabled. 1: Interrupt enabled and asserts when *AFE_ADC_n_STATUS.conv_rdy* = 1.

16.22.1.1 16-Bit Sequencer Registers

Table 16-73: Sequencer 0 to 25 Registers

Sequencer 0 Register	**AFE_ADC_*n*_UC_0**	**0x3A**
Sequencer 1 Register	**AFE_ADC_*n*_UC_1**	**0x3B**
Sequencer 2 Register	**AFE_ADC_*n*_UC_2**	**0x3C**
Sequencer 3 Register	**AFE_ADC_*n*_UC_3**	**0x3D**
Sequencer 4 Register	**AFE_ADC_*n*_UC_4**	**0x3E**
Sequencer 5 Register	**AFE_ADC_*n*_UC_5**	**0x3F**
Sequencer 6 Register	**AFE_ADC_*n*_UC_6**	**0x40**
Sequencer 7 Register	**AFE_ADC_*n*_UC_7**	**0x41**
Sequencer 8 Register	**AFE_ADC_*n*_UC_8**	**0x42**
Sequencer 9 Register	**AFE_ADC_*n*_UC_9**	**0x43**
Sequencer 10 Register	**AFE_ADC_*n*_UC_10**	**0x44**
Sequencer 11 Register	**AFE_ADC_*n*_UC_11**	**0x45**
Sequencer 12 Register	**AFE_ADC_*n*_UC_12**	**0x46**
Sequencer 13 Register	**AFE_ADC_*n*_UC_13**	**0x47**
Sequencer 14 Register	**AFE_ADC_*n*_UC_14**	**0x48**
Sequencer 15 Register	**AFE_ADC_*n*_UC_15**	**0x49**
Sequencer 16 Register	**AFE_ADC_*n*_UC_16**	**0x4A**
Sequencer 17 Register	**AFE_ADC_*n*_UC_17**	**0x4B**
Sequencer 18 Register	**AFE_ADC_*n*_UC_18**	**0x4C**
Sequencer 19 Register	**AFE_ADC_*n*_UC_19**	**0x4D**
Sequencer 20 Register	**AFE_ADC_*n*_UC_20**	**0x4E**
Sequencer 21 Register	**AFE_ADC_*n*_UC_21**	**0x4F**
Sequencer 22 Register	**AFE_ADC_*n*_UC_22**	**0x50**
Sequencer 23 Register	**AFE_ADC_*n*_UC_23**	**0x51**
Sequencer 24 Register	**AFE_ADC_*n*_UC_24**	**0x52**
Sequencer 25 Register	**AFE_ADC_*n*_UC_25**	**0x53**

Bits	Name	Access	Reset	Description
15	-	RO	0	**Reserved**
14:8	reg_addr	R/W	0	**ADC Register Address** Write the address of an 8-bit control register to include it in the sequence.
7:0	reg_data	R/W	0	**ADC Register Command** Write the command that corresponds to the register selected by the *reg_addr* field.

Table 16-74: Sequencer 26 to 52 Registers

Sequencer 26 Register			AFE_ADC_*n*_UC_26	0x54
Sequencer 27 Register			AFE_ADC_*n*_UC_27	0x55
Sequencer 28 Register			AFE_ADC_*n*_UC_28	0x56
Sequencer 29 Register			AFE_ADC_*n*_UC_29	0x57
Sequencer 30 Register			AFE_ADC_*n*_UC_30	0x58
Sequencer 31 Register			AFE_ADC_*n*_UC_31	0x59
Sequencer 32 Register			AFE_ADC_*n*_UC_32	0x5A
Sequencer 33 Register			AFE_ADC_*n*_UC_33	0x5B
Sequencer 34 Register			AFE_ADC_*n*_UC_34	0x5C
Sequencer 35 Register			AFE_ADC_*n*_UC_35	0x5D
Sequencer 36 Register			AFE_ADC_*n*_UC_36	0x5E
Sequencer 37 Register			AFE_ADC_*n*_UC_37	0x5F
Sequencer 38 Register			AFE_ADC_*n*_UC_38	0x60
Sequencer 39 Register			AFE_ADC_*n*_UC_39	0x61
Sequencer 40 Register			AFE_ADC_*n*_UC_40	0x62
Sequencer 41 Register			AFE_ADC_*n*_UC_41	0x63
Sequencer 42 Register			AFE_ADC_*n*_UC_42	0x64
Sequencer 43 Register			AFE_ADC_*n*_UC_43	0x65
Sequencer 44 Register			AFE_ADC_*n*_UC_44	0x66
Sequencer 45 Register			AFE_ADC_*n*_UC_45	0x67
Sequencer 46 Register			AFE_ADC_*n*_UC_46	0x68
Sequencer 47 Register			AFE_ADC_*n*_UC_47	0x69
Sequencer 48 Register			AFE_ADC_*n*_UC_48	0x6A
Sequencer 49 Register			AFE_ADC_*n*_UC_49	0x6B
Sequencer 50 Register			AFE_ADC_*n*_UC_50	0x6C
Sequencer 51 Register			AFE_ADC_*n*_UC_51	0x6D
Sequencer 52 Register			AFE_ADC_*n*_UC_52	0x6E
Bits	**Name**	**Access**	**Reset**	**Description**
15	-	RO	0	**Reserved**
14:8	reg_addr	R/W	0	**ADC Register Address** Write the address of an 8-bit control register to include it in the sequence.
7:0	reg_data	R/W	0	**ADC Register Command** Write the command that corresponds to the register selected by the *reg_addr* field.

Table 16-75: Sequencer Address Register

Sequencer Address			AFE_ADC_*n*_UCADDR	0x6F
Bits	**Name**	**Access**	**Reset**	**Description**
15:7	-	RO	0	**Reserved**
6:0	ucaddr	R	0	**µC Sequencer Address** This field indicates the active address of the sequencer. 0x00: Inactive. 0x01-0x2F: Reserved. 0x3A-0x6E: Sequencer register address. 0x6F-0x7F: Reserved.

Table 16-76: Temperature Sensor Control Register

Temperature Sensor Control			**AFE_ADC_*n*_TS_CTRL**	**0x7C**
Bits	**Name**	**Access**	**Reset**	**Description**
7:3	-	RO	0	**Reserved**
2	ts_intg_rdy	R/W	0	**Temperature Sensor Integration Ready** This field indicates when an ADC conversion should commence. 0: Temperature sensor integration not complete. 1: Temperature sensor integration complete and ready for conversion.
1	ts_conv_en	R/W	0	**Temperature Sensor Automatic Conversion Enable** Set this field to 1 to enable hardware to automatically start an ADC conversion of the temperature sensor when it is enabled (*AFE_ADC_n_TS_CTRL.ts_en* = 1). 0: Disable automatic temperature sensor conversion. 1: Enable automatic temperature sensor conversion.
0	ts_en	R/W	0	**Temperature Sensor Enable** Set this field to 1 to enable temperature sensor integration. 0: Temperature sensor disabled. 1: Temperature sensor enabled.

17. Digital-to-Analog Converter (DAC)

The MAX32675 includes a 12-bit DAC. The DAC can be set independently to generate either a static output voltage or generate a series of preloaded sample outputs at a specified sample rate.

The DAC peripheral support the following features:

- Configurable clock rate and output sample rate.
- Selectable voltage reference.
- The DAC can be set to output a static voltage level, a preset number of samples at a configurable sample rate, or samples continuously at a configurable sample rate.
- Interpolation filter allows for linearly interpolated output samples to be generated between each pair of output samples (2 to 1, 4 to 1, or 8 to 1).
- DAC output samples are pulled from a FIFO allowing continuous sample output generation.

17.1 Instances

There is one instance of the DAC, as shown in *Table 17-1*.

Table 17-1: DAC Instances

Instance Name	FIFO Depth	Internal Interface	DAC Output Pin
AFE_DAC	32 × 16-bits	SPI0	DAC12_OUT

17.2 Operation

The DAC must be configured before use. The following sections describe the required steps for configuring the DAC and enabling operation.

17.2.1 Selecting the DAC Using the AFE

Communication to the DAC is controlled using the AFE interface through the internal SPI0. See the section *Selecting an AFE Peripheral* for the required steps to select the DAC.

17.2.1.1 DAC Reference

Configure the DAC reference by performing the following steps:

1. Select the DAC peripheral if not already selected. See *Selecting an AFE Peripheral* for details.
2. Perform a 32-bit SPI read using the *AFE_DAC_VREF_CTRL* register address.
   a. The data read is the current value of the *AFE_DAC_VREF_CTRL* register.
3. Modify the data read and change the following fields:
   a. Enable the DAC reference block (*AFE_DAC_VREF_CTRL.ref_pu* = 1)
   b. Enable the internal reference output (*AFE_DAC_VREF_CTRL.refdac_outen* = 1)
   c. Set the *AFE_DAC_VREF_CTRL.dacrefsel* field to the desired reference voltage. See *Table 17-2* for details.
4. Perform a 32-bit SPI write using the *AFE_DAC_VREF_CTRL* address and the modified data from step 3.

Table 17-2: DAC Reference Selection

AFE_DAC_VREF_CTRL.dacrefsel	Reference Voltage
0b00	1.024V

AFE_DAC_VREF_CTRL. *dacrefsel*	Reference Voltage
0b01	1.5V
0b01	2.048V
0b11	2.5V

17.2.2 DAC Power Modes

The DAC defaults to power output level 1, 48µA. The DAC power mode is configured using a three bit field which is the concatenation of two fields (*AFE_DAC_CTRL.power_mode_2*:*AFE_DAC_CTRL.power_mode_1_0)*. *Table 17-3* shows the power mode selected using the combined fields.

Table 17-3: DAC Power Settings

AFE_DAC_CTRL. *power_mode_2*	*AFE_DAC_CTRL*. *power_mode_1_0*	DAC Power Level
0b0	0b00	Power Level 0 (48µA)
0b0	0b11	Power Level 1 (130µA)
0b1	0b01	Power Level 2 (210µA)1
0b1	0b11	Power Level 3 (291µA)

17.2.3 Enabling the DAC

After configuring the DAC reference, the DAC can be configured for operation. Initially, the DAC should be powered on and set to a known state before enabling the DAC output.

Configure the DAC by performing the following steps:

1. Select the DAC peripheral if not already selected. See *Selecting an AFE Peripheral* for details.
2. Perform a 32-bit SPI read using the *AFE_DAC_CTRL* register address.
   a. The data read is the current value of the *AFE_DAC_CTRL* register.
3. Modify the data read and change the following fields:
   a. Power on the DAC (*AFE_DAC_CTRL.power_on* = 1).
   b. Enable the DAC clock (*AFE_DAC_CTRL.clock_gate_en* = 1).
   c. Set the desired DAC power mode. See *Table 17-3* for details.
   d. Set the desired DAC operating mode (*AFE_DAC_CTRL.op_mode*).
   e. Set the start mode to 0 (*AFE_DAC_CTRL.cpu_start* = 0).
4. Perform a 32-bit SPI write using the *AFE_DAC_CTRL* address and the modified data from step 3.

17.2.4 FIFO

The DAC includes a 32 x 16-bit internal FIFO. Write data to the FIFO by writing to the *AFE_DAC_FIFO* register. Before writing data to the DAC FIFO, the FIFO almost full and FIFO almost empty fields should be configured. See *Enabling the DAC* for steps to set the levels. Next, write the DAC FIFO using the following steps.

1. Select the DAC peripheral if not already selected. See *Selecting an AFE Peripheral* for details.
2. Perform a 16-bit SPI write using the *AFE_DAC_FIFO* register address.
   a. The 16-bit data to write is added to the DAC FIFO.
3. Repeat step 2 until the DAC FIFO is full or all of the required data is written to the FIFO.

17.3 DAC Registers

These registers are accessed through the *Analog Front-End (AFE)* using the internal SPI0 interface. See *Table 1-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, a soft reset, a POR, and the peripheral-specific resets.

There is one instance of the DAC peripheral, and the DAC registers are shown in *Table 16-14*. The DAC registers are either 16-bits or 32-bits wide. See the width column in *Table 16-14* for details.

Table 17-4: DAC Registers

Address	Width	Name	Description
0x00	32-bits	*AFE_DAC_CTRL*	DAC Control Register
0x01	32-bits	*AFE_DAC_RATE*	DAC Rate Register
0x02	32-bits	*AFE_DAC_INT*	DAC Interrupt Register
0x04	32-bits	*AFE_DAC_TRIM*	DAC Trim Register
0x05	16-bits	*AFE_DAC_VREF_CTRL*	DAC Voltage Reference Control Register
0x06	16-bits	*AFE_DAC_FIFO*	DAC FIFO Register
0x07	16-bits	*AFE_DAC_VREF_TRIM*	DAC Voltage Reference Trim

17.3.1 Register Details

Table 17-5: DAC Control Register

DAC Control Register				**AFE_DAC_CTRL**
Bits	**Name**	**Access**	**Reset**	
31	rst	R/W1O	0	**Reset** Set this field to 1 to reset the DAC. This field is automatically cleared to 0 after a DAC reset is performed. 0: Normal operation. 1: Reset DAC. This field is automatically cleared to 0 when the reset is complete.
30	power_mode_2	R/W	0	**Power Mode 2** See *DAC Power Modes* for details on this field's usage.
29	clock_gate_en	R/W	0	**Clock Gate Enable** 0: Clock gating disabled. 1: Clock gating enabled.
28	power_on	R/W	0	**Power On** 0: DAC powered off. 1: DAC powered on.
27:26	power_mode_1_0	R/W	0	**DAC Power Mode Select 1** See *DAC Power Modes* for details on this field's usage.
25:24	op_mode	R/W	0	**Operating Mode** This field selects the DAC operating mode. 0b00: Output data in FIFO as soon as it is available. 0b01: Output *AFE_DAC_RATE.sample_cnt* data points one time from FIFO at an output rate defined by the *AFE_DAC_RATE.rate_cnt* field. 0b10: Reserved. 0b11: Continuously output *AFE_DAC_RATE.sample_cnt* data points from the FIFO at an output rate defined by the *AFE_DAC_RATE.rate_cnt* field.
23:21	-	RO	0	**Reserved**

DAC Control Register				AFE_DAC_CTRL
Bits	**Name**	**Access**	**Reset**	
20	cpu_start	R/W	0	**Start Bit** This field should be set to 0 for proper DAC operation.
19:18	-	RO	0	**Reserved**
17:16	start_mode	R/W	0	**Start Mode** This field controls the condition that results in a DAC output sequence starting. 0: Start sequence when the FIFO is not empty. 1 - 3: Reserved.
15:12	fifo_af_cnt	R/W	0b0111	**DAC FIFO Almost Full Threshold** Set this field to the level of the FIFO to trigger a FIFO almost full flag, *AFE_DAC_CTRL.fifo_almost_full*, being set to 1. $Threshold = AFE\_DAC\_CTRL.fifo\_af\_cnt + 16$ *Note: Valid values for this field are 0 to 15.*
11	-	R/W	0	**Reserved**
10:8	interp_mode	R/W	0	**DAC Output Interpolation Mode** 0: Disabled. 1: 2 to 1 interpolation. 2: 4 to 1 interpolation. 3: 8 to 1 interpolation. 4-15: Reserved.
7	fifo_almost_empty	R	1	**FIFO Almost Empty Flag** Hardware automatically sets this field to 1 when the FIFO is almost empty as set by the FIFO level falling below the *AFE_DAC_CTRL.fifo_ae_cnt*. 0: FIFO is not in almost empty condition. 1: FIFO threshold is below the *AFE_DAC_CTRL.fifo_ae_cnt* level.
6	fifo_empty	R	1	**FIFO Empty Flag** This field is set to 1 by hardware automatically when the DAC FIFO is empty. 0: FIFO empty. 1: FIFO not empty.
5	fifo_almost_full	R	0	**FIFO Almost Full Flag** This field is set to 1 automatically by hardware when the FIFO level is greater than the FIFO threshold defined by the *AFE_DAC_CTRL.fifo_af_cnt* field. 0: FIFO is not almost full. 1: FIFO is almost full.
4	-	RO	0	**Reserved**
3:0	fifo_ae_cnt	R/W	0b0100	**FIFO Almost Empty Threshold** This field sets the level that triggers hardware to set the *AFE_DAC_CTRL.fifo_almost_empty* field to 1. When the FIFO level falls below the value set in this field, the *AFE_DAC_CTRL.fifo_ae_cnt* field is set to 1. *Note: Valid values for this field are 0 to 15.*

Table 17-6: DAC Rate Register

DAC Rate				AFE_DAC_RATE / 0x01
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	sample_cnt	R/W	0	**Output Sample Count** When the DAC operating mode is set to *AFE_DAC_CTRL.op_mode* is set to 0b01 and interpolation mode is active (*AFE_DAC_CTRL.interp_mode* != 0), this field sets the total number of data points to output using the following equation. $data\ points = (sample\_cnt - 1) \times (2^{interp\_mode}) + 1$ When the DAC operation mode is set to *AFE_DAC_CTRL.op_mode* is set to 0b01 and interpolation mode is disabled (*AFE_DAC_CTRL.interp_mode* = 0), this field sets the total number of data points directly.
15:0	rate_cnt	R/W	0	**Output Rate Control** When the DAC operating mode is set to *AFE_DAC_CTRL.op_mode* is set to 0b01 or 0b11, this field sets the delay between output samples as shown in the following equation. $T_S = (rate\_cnt + 2) \times \frac{1}{2.4} MHz$

Table 17-7: DAC Interrupt Register

DAC Interrupt				AFE_DAC_INT / 0x02
Bits	**Name**	**Access**	**Reset**	**Description**
31:18	-	R	0	**Reserved**
17	underflow_ie	R/W	0	**Underflow Interrupt Enable** Set this field to 1 to enable the interrupt.
16	out_done_ie	R/W	0	**Output Done Interrupt Enable** Set this field to 1 to enable the interrupt.
15:4	-	R	0	**Reserved**
3	underflow	R/W1C	0	**FIFO Underflow** 0: Normal operation. 1: FIFO underflow condition occurred.
2	almost_empty_if	R/W1C	0	**FIFO Almost Empty Interrupt Flag** This field is automatically set by hardware when the FIFO reaches the almost empty level as set using the *AFE_DAC_CTRL.fifo_ae_cnt* field. 0: Normal operation. 1: Interrupt condition occurred.
1	underflow_if	R/W1C	0	**FIFO Underflow Interrupt Flag** This field is automatically set to 1 by hardware when a FIFO underflow condition occurs. 0: Normal operation. 1: Interrupt condition occurred.
0	out_done_if	R/W1C	0	**Output Done Interrupt Flag** 0: Normal operation. 1: Output complete.

Table 17-8: DAC Trim Register

DAC Trim			AFE_DAC_TRIM	0x04
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	trim	RO	*	**DAC Trim Value**

Table 17-9: DAC Voltage Reference Control Register

DAC Voltage Reference Control			AFE_DAC_VREF_CTRL	0x05
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:6	refdac_gain	RO	0	**DAC Reference Gain** 0: Default gain. 3: Highest gain.
5	refdac_cp	R/W	0	**DAC Reference Stability Compensation Pole** 0: No additional pole. 1: Added pole to compensate zero from external (adds 100fF capacitor across 2nd gain stage).
4	ref_pu	R/W	0	**DAC Reference Power Up** 0: DAC reference powered down. 1: DAC reference powered up.
3	refdac_outen	R/W	0	**DAC Reference Output Enable** 0: Internal DAC reference powered down. 1: Internal DAC reference enabled, cannot be driven externally.
2:1	dacrefsel	R/W	0	**DAC Reference Select** This field selects the reference voltage for the DAC. 0: 1.024V. 1: 1.5V. 2: 2.048V. 3: 2.5V.
0	ref_dac_fast_pd	R/W	0	**DAC Reference Fast Power Down** Setting this field to 1 enables a fast DAC reference power down in milliseconds versus seconds in standard power down. In addition, this field can improve the DAC reference slew rate when lowering the DAC reference select from a higher value to a lower value. 0: DAC reference powers down normally. 1: DAC reference powers down in milliseconds rather than seconds. *Note: After setting this field to 1, it must then be set to 0 before powering on the DAC.*

Table 17-10: DAC FIFO Register

DAC FIFO			AFE_DAC_FIFO	0x06
Bits	**Name**	**Access**	**Reset**	**Description**
15:0	fifo_data	R/W	0	**DAC FIFO Data** Write to this register field to load data into the DAC FIFO. See *FIFO* for details.

Table 17-11: DAC Voltage Reference Trim Register

DAC Voltage Reference Trim			AFE_DAC_VREF_TRIM	0x07
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	trim	RO	0	**Reserved**

18. Timers (TMR/LPTMR)

Multiple 32-bit and dual 16-bit reloadable timers are provided.

The features include:

- Operation as a single 32-bit counter or single/dual 16-bit counter(s).
- Programmable clock prescaler with values from 1 to 4096
- Capture, compare, and capture/compare capability.
- Timer input and output signals available mapped as alternate functions.
- Configurable input pin for event triggering, clock gating, or capture signal
- Timer output pin for event output and pulse-width modulated (PWM) signal generation.
- Multiple clock source options.

Instances denoted as LPTMR, shown in *Table 18-1*, are configurable to operate in any of the low-power modes and wake the device from the low-power modes to *ACTIVE*.

Each timer supports multiple operating modes:

- One-shot: the timer counts up to terminal value then halts.
- Continuous: the timer counts up to the terminal value then repeats.
- Counter: the timer counts input edges received on the timer input pin.
- PWM: the timer outputs a signal until a terminal value is reached and then toggles the output signal until a second terminal value is reached and then repeats.
- Capture: the timer captures a snapshot of the current timer count when the timer's input edge transitions.
- Compare: the timer pin toggles when the timer's count exceeds the terminal count.
- Gated: the timer increments only when the timer's input pin is asserted.
- Capture/Compare: the timer counts when the timer input pin is asserted; the timer captures the timer's count when the input pin is deasserted.

18.1 Instances

Instances of the peripheral are listed in *Table 18-1*. Both the TMR and LPTMR are functionally very similar, so for convenience, they are referred to as just TMR. The LPTMR instances can function while the device is in *DEEPSLEEP* and *BACKUP* modes. TMR instances can operate in dual 16-bit mode or cascaded 32-bit mode if supported. LPTMR instances provide a single 32-bit timer and can select clock sources available in *DEEPSLEEP* and *BACKUP* mode.

Table 18-1. MAX32675 TMR/LPTMR

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

Table 18-2: MAX32675 TMR/LPTMR Instances Capture Events

Instance	Capture Event 0	Capture Event 1	Capture Event 2	Capture Event 3
TMR0	Timer Input Pin	-	-	-
TMR1	Timer Input Pin	-	-	-
TMR2	Timer Input Pin	-	-	-
TMR3	Timer Input Pin	-	-	-
LPTMR0	Timer Input Pin	-	-	-

18.2 Basic Timer Operation

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

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

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

18.3 32-Bit Single/32-Bit Cascade/Dual 16-Bit

Most instances contain two 16-bit timers, which may support combinations of single or cascaded 32-bit modes, and single or dual 16-bit modes, as shown in *Table 18-1*. In most cases, the two 16-bit timers have the same functionality.

The terminology TimerA and TimerB are used to differentiate the organization of the 32-bit registers shown in *Table 18-3*. Most of the other registers have the same fields duplicated in the upper and lower 16 bits and are differentiated with the _a and _b suffixes.

In the 32-bit modes, the fields and controls associated with TimerA control the 32-bit timer functionality. In single 16-bit timer mode, the TimerA fields control the single 16-bit timer, and the TimerB fields are ignored. In dual 16-bit timer modes, both TimerA and TimerB fields control the dual timers; TimerB fields control the upper 16-bit timer, and TimerA fields control the lower 16-bit timer. In dual 16-bit timer modes, TimerB can be used as a single 16-bit timer.

Table 18-3: TimerA/TimerB 32-Bit Field Allocations

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

18.4 Timer Clock Sources

Clocking of timer functions is driven by the timer clock frequency, f_{CNT_CLK}, which is a function of the selected clock source shown in *Table 18-1*. Most modes support multiple clock sources and prescaler values, which can be chosen independently for TimerA and TimerB when the peripheral is operating in dual 16-bit mode. The prescaler can be set from 1 to 4096 using the *TMRn_CTRL0.clkdiv* field.

Equation 18-1: Timer Peripheral Clock Equation

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

Software configures and controls the timers by reading and writing to the timer registers. External events on timer pins are asynchronous events to the timer's clock frequency. The external events are latched on the next rising edge of the timer's clock. Since it is not possible to externally synchronize to the timer's internal clock, input events may require up to 50% of the timer's internal clock before the hardware recognizes the event.

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

1. Disable the timer peripheral.
   a. Clear *TMRn_CTRL0.en* to 0 to disable the timer.
   b. Read the *TMRn_CTRL1.clken* field until it returns 0, confirming the timer peripheral is disabled.
2. Set *TMRn_CTRL1.clksel* to the new desired clock source.
3. Configure the timer for the desired operating mode. See the section *Operating Modes* for details on mode configuration.
4. Enable the timer clock source.
   a. Set the *TMRn_CTRL0.clken* field to 1 to enable the timer's clock source.
   b. Read the *TMRn_CTRL1.clkrdy* field until it returns 1, confirming the timer clock source is enabled.
5. Enable the timer.
   a. Set *TMRn_CTRL0.en* to 1 to enable the timer.
   b. Read the *TMRn_CNT.clken* field until it returns 1 to confirm the timer is enabled.

The timer peripheral should be disabled while changing any of the registers in the peripheral.

18.5 Timer Pin Functionality

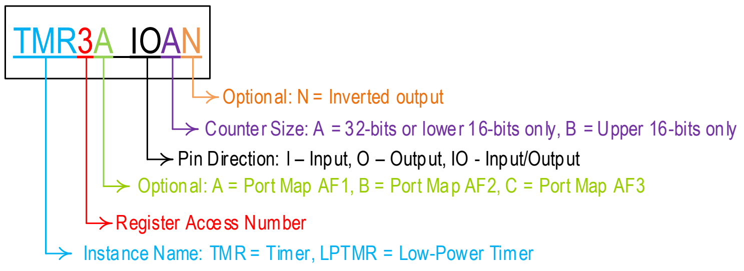
Each timer instance may have an input signal and/or output signal depending on the operating mode. Not all instances of the peripheral are available in all packages. The number of input and output signals per peripheral instance may vary as well. Refer to the data sheet for I/O signal configurations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

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

The timer pin functionality is mapped as an alternate function that is shared with a GPIO. When the timer pin alternate function is enabled, the timer pin has the same electrical characteristics, such as pullup/pulldown strength and drive strength as the GPIO mode settings for that pin. The pin characteristics must be configured before enabling the timer. When configured as an output, the corresponding bit in the *GPIOn_OUT* and *GPIOn_OUTEN* registers should be configured to match the inactive state of the timer pin for that mode. Consult the *General-Purpose I/O and Alternate Function Pins (GPIO)* chapter for details on how to configure the electrical characteristics for the pin.

Figure 18-1 shows the timer pin naming convention of the MAX32675 microcontroller.

Figure 18-1: Timer I/O Signal Naming Conventions


TMR3A_IOAN
Optional: N = Inverted output
Counter Size: A = 32-bits or lower 16-bits only, B = Upper 16-bits only
Pin Direction: I – Input, O – Output, IO - Input/Output
Optional: A = Port Map AF1, B = Port Map AF2, C = Port Map AF3
Register Access Number
Instance Name: TMR = Timer, LPTMR = Low-Power Timer

The TimerA output controls for modes 0, 1, 3, and 5 output signals are shown in *Figure 18-2*. The TimerA input controls for modes 2, 4, 6, 7, 8, and 14 input signals are shown in *Figure 18-3*.

Figure 18-2: MAX32675 TimerA Output Functionality, Modes 0/1/3/5

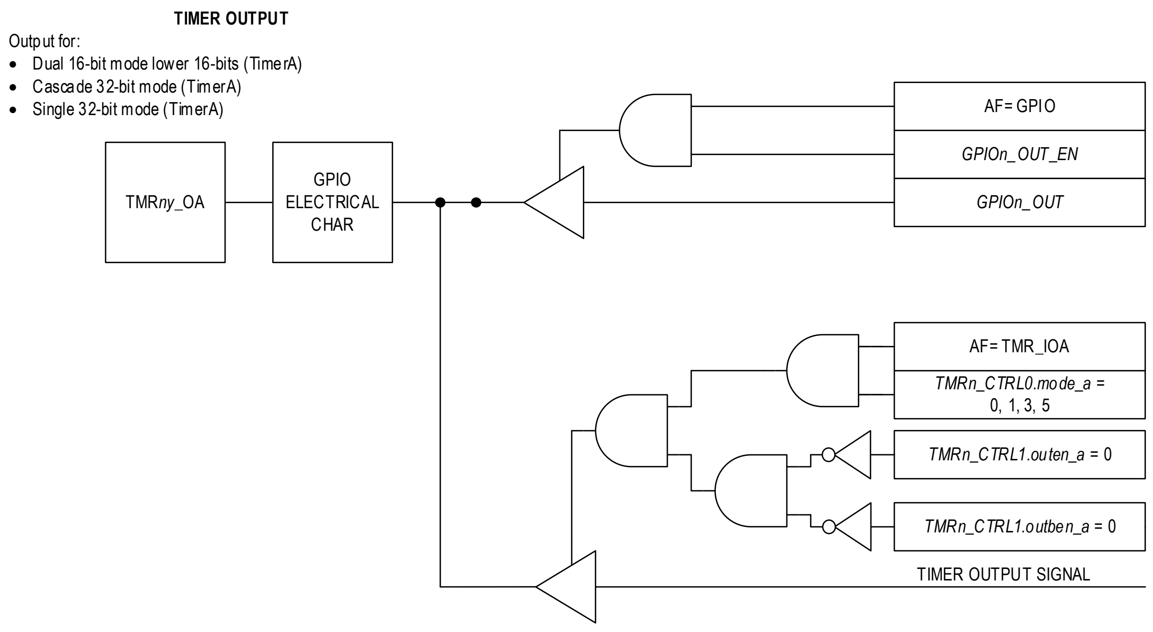
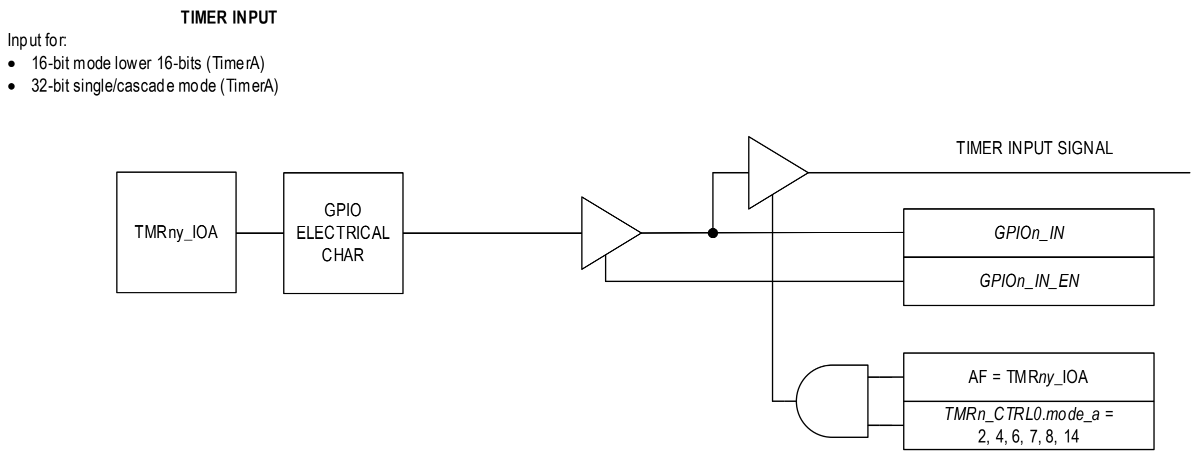


Figure 18-3: MAX32675 TimerA Input Functionality, Modes 2/4/6/7/8/14


TIMER INPUT
Input for:
16-bit mode lower 16-bits (TimerA)
32-bit single/cascade mode (TimerA)
TMRny_IOA
GPIO ELECTRICAL CHAR
TIMER INPUT SIGNAL
GPIOn_IN
GPIOn_IN_EN
AF = TMRny_IOA
TMRn_CTRL0.mode_a = 2, 4, 6, 7, 8, 14

18.6 Wake-Up Events

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

Programming Sequence Example:

1. Disable the timer peripheral and set the timer clock source as described in *Timer Clock Sources*.
2. Configure the timer operating mode as described in the section *Operating Modes*.
3. Enable the timer by setting *TMRn_CTRL0*.*en* to 1.
4. Poll *TMRn_CTRL1*.*clkrdy* until it reads 1.
5. Set the *TMRn_CTRL1*.*we* field to 1 to enable wake-up events for the timer.
6. If desired, enable the timer interrupt and provide an interrupt handler for the timer.
7. Enter a low-power mode as described in the section *Operating Modes*.
8. When the device wakes up from the low-power mode, check the *TMRn_WKFL* register to determine if the timer caused the wake-up event.

18.7 Operating Modes

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

In *Table 18-4* the timer's signal name is generically shown where *n* is the timer number (0, 1, 2, or 3)and *y* is the port mapping alternate function. See *Figure 18-1* for details of the timer's naming convention for I/O signals.

In *Table 18-5* the low power timer's signal name is generically shown where *n* is the timer number (0)and *y* is the port mapping alternate function. See *Figure 18-1* for details of the timer's naming convention for I/O signals.

Table 18-4: MAX32675 Operating Mode Signals for Timer 0 through Timer 4

Timer Mode	**TimerA** *TMRn_CTRL1*.***outen*** **= 0** *TMRn_CTRL1*.***outben*** **= 0**	**I/O Signal Name†**	**Pin Required**
One-Shot Mode (0)	TimerA Output Signal	TMR*ny*_OA	Optional
Continuous Mode (1)	TimerA Output Signal	TMR*ny*_OA	Optional
Counter Mode (2)	TimerA Input Signal	TMR*ny*_IA	Yes
Capture Mode (4)	TimerA Input Signal	TMR*ny*_IA	Yes
Compare Mode (5)	TimerA Output Signal	TMR*ny*_OA	Optional
Gated Mode (6)	TimerA Input Signal	TMR*ny*_IA	Yes
Capture/Compare Mode (7)	TimerA Input Signal	TMR*ny*_IA	Yes
Dual Edge Capture Mode (8)	TimerA Input Signal	TMR*ny*_IA	Yes
Reserved (9 - 13)	-	-	-
Inactive Gated Mode (14)	TimerA Input Signal	TMR*ny*_IA	Yes
Reserved (15)	-	-	-

† See Figure 18-1 for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.

Table 18-5: MAX32675 Operating Mode Signals for Low-Power Timer 0

Timer mode	TMR5 *TMRn_CTRL1.outen* = 0 *TMRn_CTRL1.outben* = 0	I/O Signal Name[†]	Required?
One-Shot Mode (0)	TimerA Output Signal	LPTMR*ny*_OA	Optional
Continuous Mode (1)	TimerA Output Signal	LPTMR*ny*_OA	Optional
Counter Mode (2)	TimerA Input Signal	LPTMR*ny*_IA	Yes
Capture Mode (4)	TimerA Input Signal	LPTMR*ny*_IA	Yes
Compare Mode (5)	TimerA Output Signal	LPTMR*ny*_OA	Optional
Gated Mode (6)	TimerA Input Signal	LPTMR*ny*_IA	Yes
Capture/Compare Mode (7)	TimerA Input Signal	LPTMR*ny*_IA	Yes
Dual Edge Capture Mode (8)	TimerA Input Signal	LPTMR*ny*_IA	Yes
Reserved (9 - 13)	-	-	-
Inactive Gated Mode (14)	TimerA Input Signal	LPTMR*ny*_IA	Yes
Reserved (15)	-	-	-

[†] See Figure 18-1 for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.

18.7.1 One-Shot Mode (0)

In one-shot mode, the timer peripheral increments the timer's *TMRn_CNT.count* field until it reaches the timer's *TMRn_CMP.compare* field, and the timer is then disabled. If the timer's output is enabled, the output signal is driven active for one timer clock cycle. Thus, one-shot mode provides exactly one timer period and is automatically disabled.

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

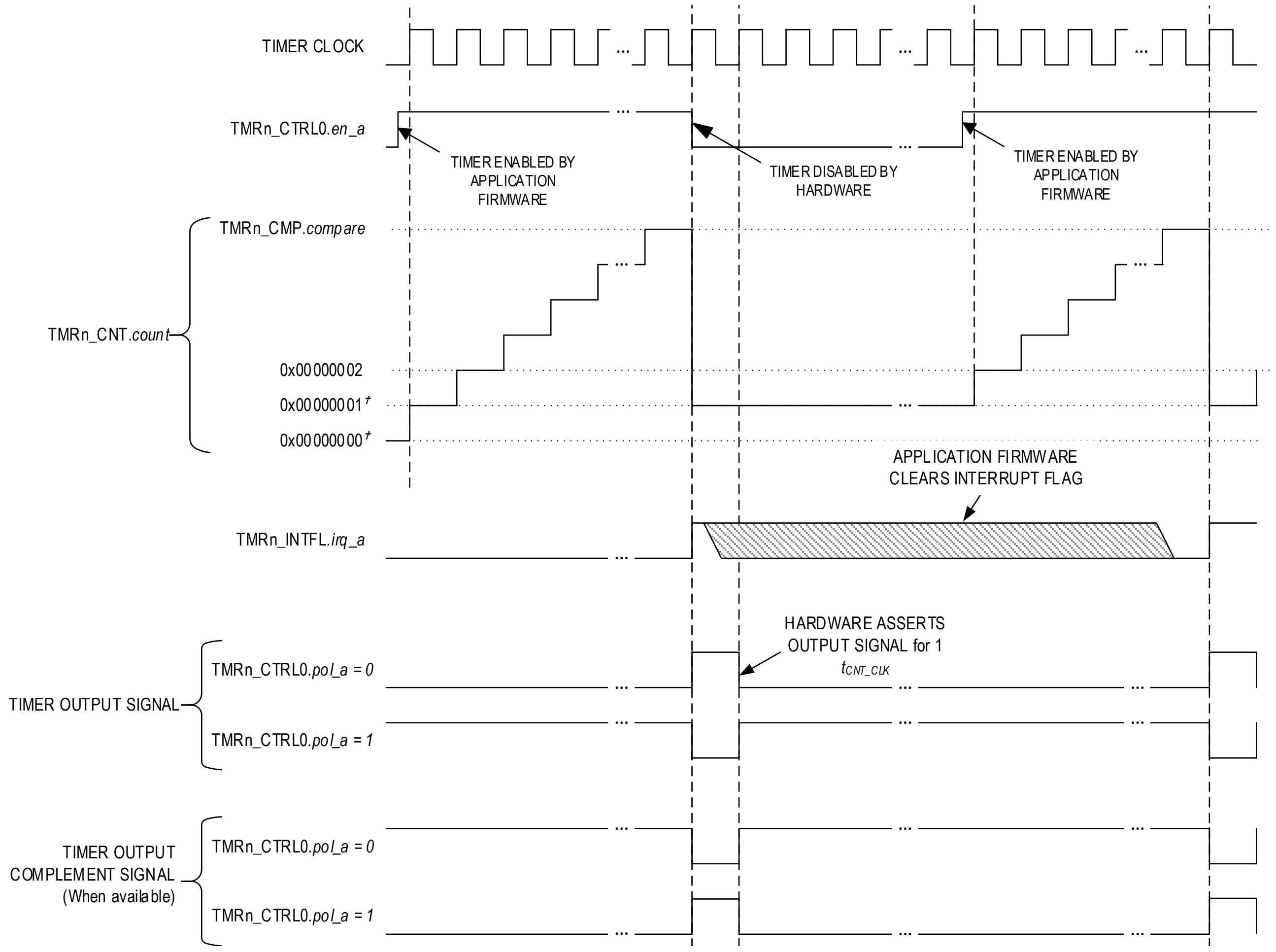
- The *TMRn_CNT.count* field is set to 0x0000 0001.
- The timer is disabled (*TMRn_CTRL0.en* = 0).
- The timer output, if enabled, is driven to its active state for one timer clock period.
- The *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

The timer period is calculated using Equation 18-2.

Equation 18-2: One-Shot Mode Timer Period in Seconds

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

Figure 18-4: One-Shot Mode Diagram


TIMER CLOCK
TMRn_CTRL0.*en_a*
TIMER ENABLED BY APPLICATION FIRMWARE
TIMER DISABLED BY HARDWARE
TIMER ENABLED BY APPLICATION FIRMWARE
TMRn_CMP.*compare*
TMRn_CNT.*count*
0x00000002
0x00000001†
0x00000000†
APPLICATION FIRMWARE CLEARS INTERRUPT FLAG
TMRn_INTFL.*irq_a*
HARDWARE ASSERTS OUTPUT SIGNAL for 1 t_{CNT_CLK}
TIMER OUTPUT SIGNAL
TMRn_CTRL0.*pol_a = 0*
TMRn_CTRL0.*pol_a = 1*
TIMER OUTPUT COMPLEMENT SIGNAL (When available)
TMRn_CTRL0.*pol_a = 0*
TMRn_CTRL0.*pol_a = 1*

This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.*cascade* = 1 (32-bit Cascade Timer)
TMRn_CTRL0.*mode_a* = 0 (One-shot)

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

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

1. Disable the timer peripheral and set the timer clock source as described in *Timer Clock Sources*.
2. Set the *TMRn_CTRL0.mode* field to 0 to select one-shot mode.
3. Set the *TMRn_CTRL0.clkdiv* field to set the prescaler for the required timer frequency.
4. If using the timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
5. Or, if using the inverted timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.
6. If using the timer interrupt, enable the corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to the *TMRn_CMP.compare* field.
   a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
8. If desired, write an initial value to the *TMRn_CNT.count* field.
   a. This affects only the first period; subsequent timer periods always reset the *TMRn_CNT.count* field to 0x0000 0001.
9. Enable the timer peripheral as described in *Timer Clock Sources*.

18.7.2 Continuous Mode (1)

In continuous mode, the *TMRn_CNT.count* field increments until it matches the *TMRn_CMP.compare* field; the *TMRn_CNT.count* field is then set to 0x0000 0001, and the count continues to increment. Optionally, the software can configure continuous mode to toggle the timer output pin at the end of each timer period. A continuous mode timer period ends when the timer count field reaches the timer compare field (*TMRn_CNT.count* = *TMRn_CMP.compare*).

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

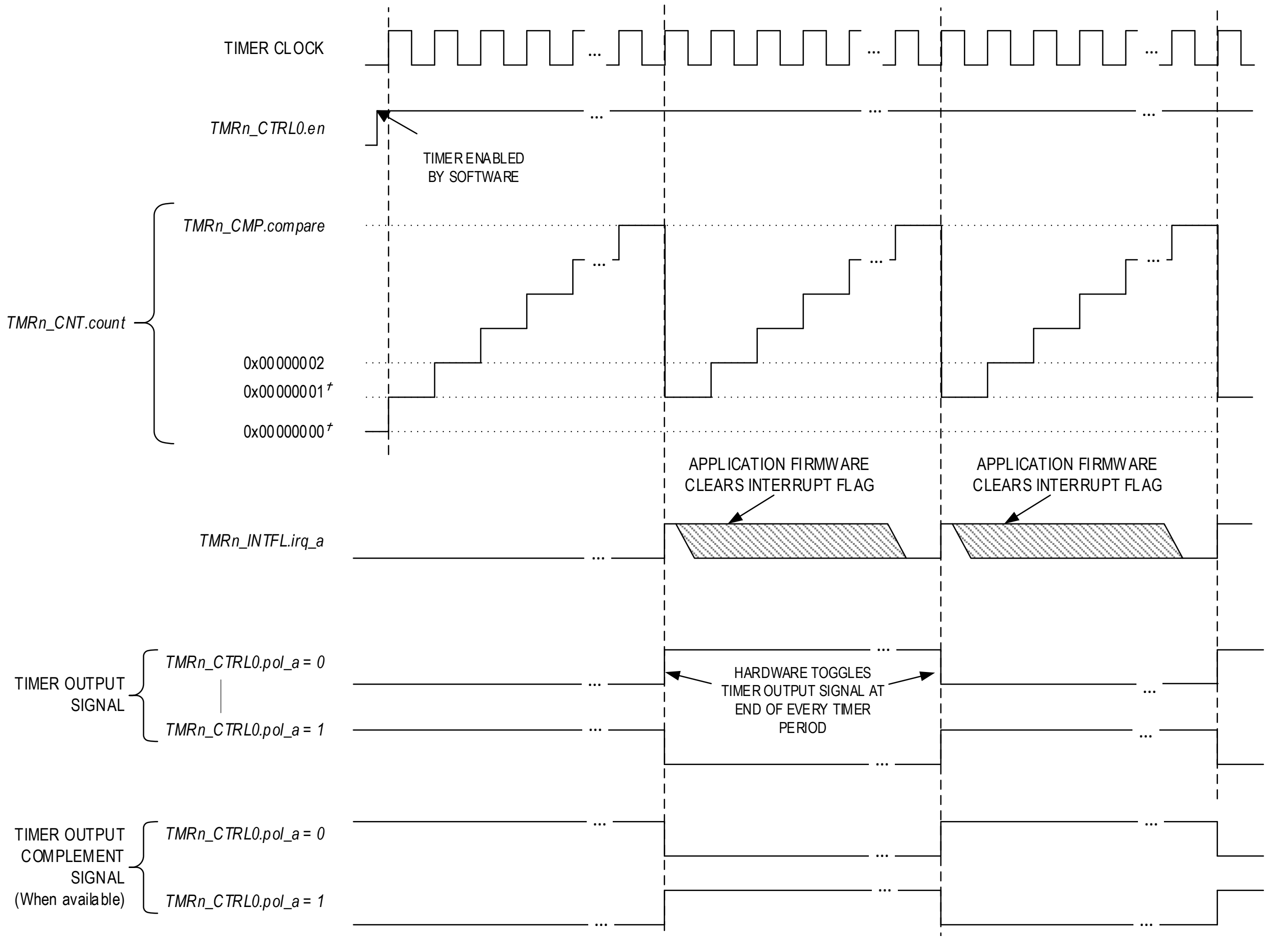
- the *TMRn_CNT.count* field is set to 0x0000 0001,
- if the timer output signal is toggled, the corresponding *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

The continuous mode timer period is calculated using *Equation 18-3*.

Equation 18-3: Continuous Mode Timer Period in Seconds

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

Figure 18-5: Continuous Mode Diagram


TIMER CLOCK
TMRn_CTRL0.en
TIMER ENABLED BY SOFTWARE
TMRn_CMP.compare
TMRn_CNT.count
0x00000002
0x00000001†
0x00000000†
APPLICATION FIRMWARE CLEARS INTERRUPT FLAG
APPLICATION FIRMWARE CLEARS INTERRUPT FLAG
TMRn_INTFL.irq_a
TIMER OUTPUT SIGNAL
TMRn_CTRL0.pol_a = 0
TMRn_CTRL0.pol_a = 1
HARDWARE TOGGLES TIMER OUTPUT SIGNAL AT END OF EVERY TIMER PERIOD
TIMER OUTPUT COMPLEMENT SIGNAL (When available)
TMRn_CTRL0.pol_a = 0
TMRn_CTRL0.pol_a = 1

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

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

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

1. Disable the timer peripheral and set the timer clock as described in *Timer Clock Sources*.
2. Set the *TMRn_CTRL0.mode* field to 1 to select continuous mode.
3. Set the *TMRn_CTRL0.clkdiv* field to set the prescaler that determines the timer frequency.
4. If using the timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
5. If using the timer interrupt, enable the corresponding field in the *TMRn_CTRL1* register.
6. Write the compare value to the *TMRn_CMP.compare* field.
   a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
7. If desired, write an initial value to the *TMRn_CNT*.count field.
   a. This affects only the first period; subsequent timer periods always reset the *TMRn_CNT.count* field to 0x0000 0001.
9. Enable the timer peripheral as described in *Timer Clock Sources*.

18.7.3 Counter Mode (2)

In counter mode, the timer peripheral increments the *TMRn_CNT.count* each time a transition occurs on the timer input signal. The transition must be greater than $4 \times PCLK$ for a count to occur. When the *TMRn_CNT.count* reaches the *TMRn_CMP.compare* field, the hardware automatically sets the interrupt bit to 1 (*TMRn_INTFL.irq)*, sets the *TMRn_CNT.count* field to 0x0000 0001, and continues incrementing. The timer can be configured to increment on either the rising edge or the falling edge of the timer's input signal, but not both. Use the *TMRn_CTRL0.pol_* field to select which edge is used for the timer's input signal count.

The timer prescaler setting does not affect this mode. The timer's input signal frequency(f_{CTR_CLK}) must not exceed 25 percent of the PCLK frequency, as shown in *Equation 18-4*.

Note: If the input signal's frequency is equal to f_{PCLK} *it is possible that the timer hardware can miss the transition due to PCLK being an asynchronous internal clock. A minimum of four PCLK cycles is required for a count to occur. Therefore, the timer input signal must be greater than four PCLK cycles to guarantee a count occurs.*

Equation 18-4: Counter Mode Maximum Clock Frequency

$$f_{CTR\_CLK} \leq \frac{f_{PCLK}\ (Hz)}{4}$$

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

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

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

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

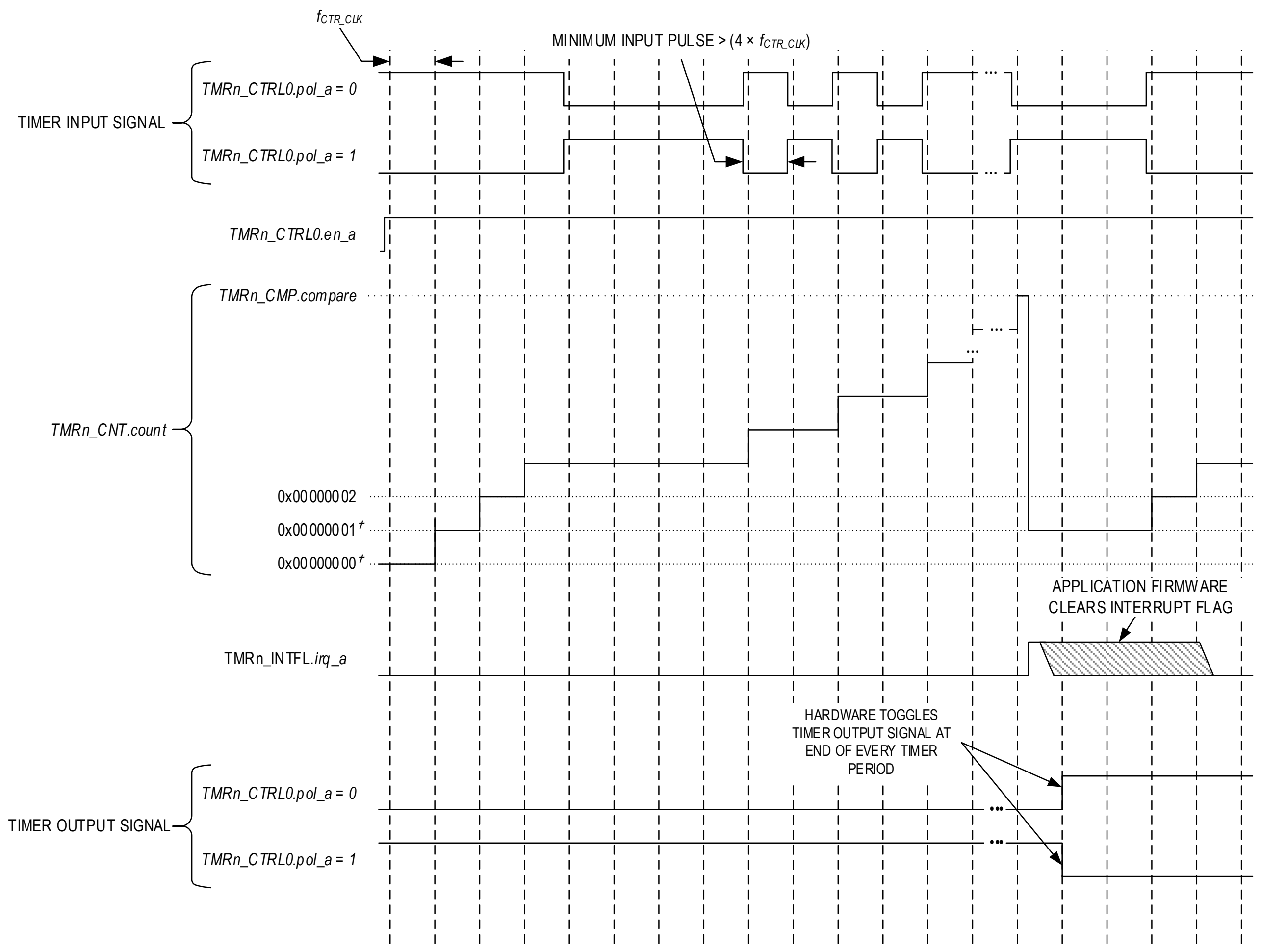
In counter mode, the number of timer input transitions during a period is equal to the *TMRn_CMP.compare* field's setting. Use *Equation 18-5* to determine the number of transitions before the end of the timer's period.

Note: Equation 18-5 is only valid during an active timer count before the end of the timer's period.

Equation 18-5: Counter Mode Timer Input Transitions

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

Figure 18-6: Counter Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 2 (Counter)

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

Configure the timer for counter mode by performing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0.mode* 2 to select counter mode.
4. Configure the timer input function:
    a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
    b. Configure the GPIO electrical characteristics as desired.
    c. Set *TMRn_CTRL1.outen_a* and *TMRn_CTRL1.outben* to the values shown in the section *Operating Modes*.
    d. Select the correct alternate function mode for the timer input pin.
5. Write the compare value to *TMRn_CMP.compare*.
    a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
6. If desired, write an initial value to *TMRn_CNT.count*. The initial value affects only the first period; subsequent timer periods always reset *TMRn_CNT.count* = 0x0000 0001.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

18.7.4 PWM Mode (3)

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

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

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

- The *TMRn_CNT.count* is reset to 0x0000 0001, and the timer resumes counting.
- The timer output signal is toggled.
- The corresponding *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

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

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

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

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set the *TMRn_CTRL0.mode field* to 3 to select PWM mode.
4. Set the *TMRn_CTRL0.clkdiv* field to set the prescaler that determines the timer frequency.
5. Configure the pin as a timer input and configure the electrical characteristics as needed.
6. Set *TMRn_CTRL0.pol* to match the desired initial (inactive) state.
7. Set *TMRn_CTRL0.pol* to select the initial logic level (high or low) and PWM transition state for the timer's output.
8. Set *TMRn_CNT.count* initial value if desired.
    a. The initial *TMRn_CNT.count* value only affects the initial period in PWM mode, with subsequent periods always setting *TMRn_CNT.count* to 0x0000 0001.
9. Set the *TMRn_PWM* value to the transition period count.
    a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
10. Set the *TMRn_CMP.compare* value for the PWM second transition period. Note: *TMRn_CMP.compare* must be greater than the *TMRn_PWM* value.
    a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
11. If using the timer interrupt, set the interrupt priority and enable the interrupt.
12. Enable the timer peripheral as described in *Timer Clock Sources*.

Equation 18-6 shows the formula for calculating the timer PWM period.

Equation 18-6: Timer PWM Period in Seconds

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

If an initial starting value other than 0x0000 0001 is loaded into the *TMRn_CNT.count* register, use the one-shot mode equation, *Equation 18-2*, to determine the initial PWM period.

If *TMRn_CTRL0.pol* is 0, the ratio of the PWM output high time to the total period is calculated using *Equation 18-7*.

Equation 18-7: Timer PWM Output High Time Ratio with Polarity 0

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

If *TMRn_CTRL0.pol* is set to 1, the ratio of the PWM output high time to the total period is calculated using *Equation 18-8*.

Equation 18-8: Timer PWM Output High Time Ratio with Polarity 1

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

18.7.5 Capture Mode (4)

Capture mode is used to measure the time between software determined events. First, the timer starts incrementing the timer's count field until a transition occurs on the timer's input pin or a rollover event occurs. A capture event is triggered by hardware when the timer's input pin transitions state. *Equation 18-9* shows the formula for calculating the capture event's elapsed time.

If a capture event does not occur before the timer's count value reaching the timer's compare value (*TMRn_CNT.count* = *TMRn_CMP.compare*), a rollover event occurs. The capture and rollover events set the timer's interrupt flag, *TMRn_INTFL.irq,* to 1, resulting in an interrupt if the timer's interrupt is enabled.

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

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

18.7.5.1 Capture Event

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

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

Software must check the value of the TMRn_PWM.pwm field to determine the trigger of the timer interrupt.

Equation 18-9: Capture Mode Elapsed Time Calculation in Seconds

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

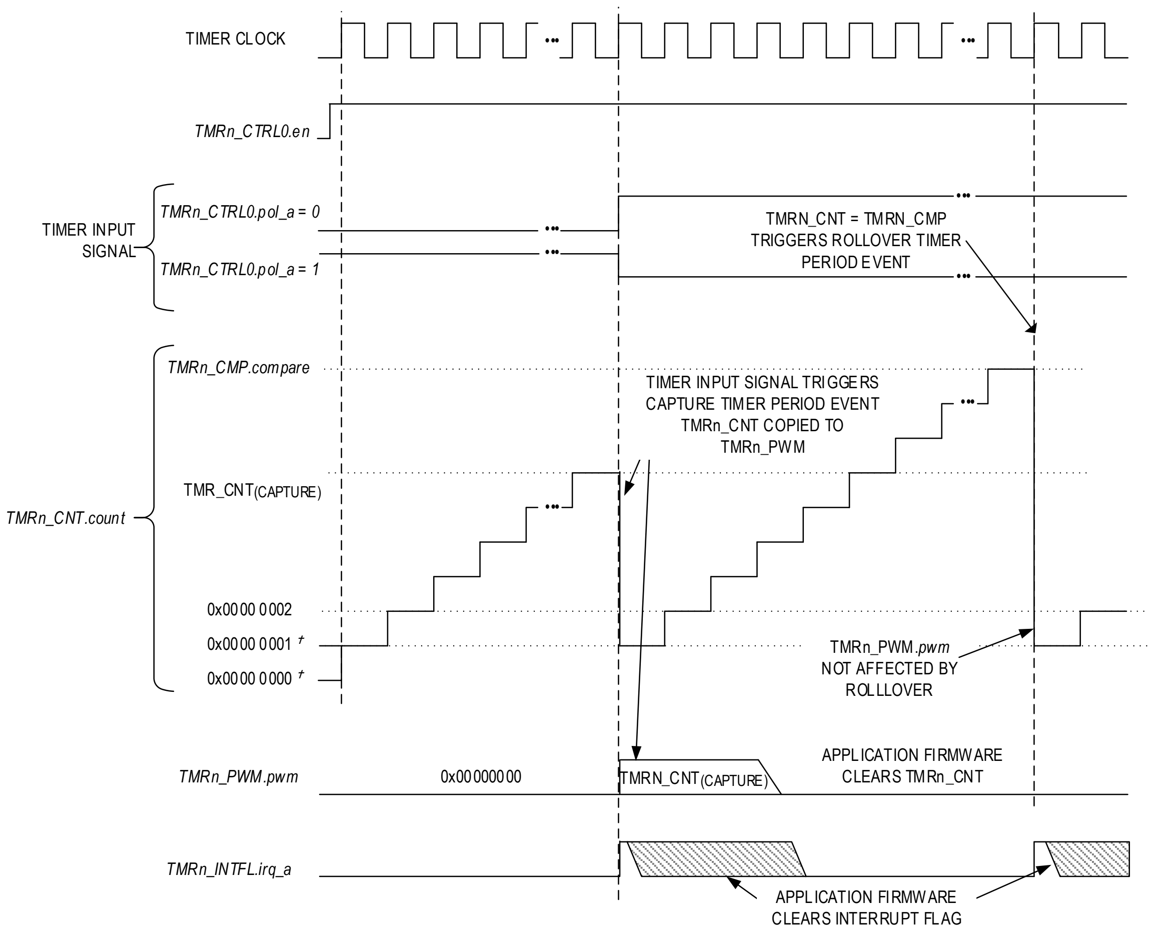
Note: The capture elapsed time calculation is only valid after the capture event occurs, and the timer stores the captured count in the TMRn_PWM register.

18.7.5.2 Rollover Event

A rollover event occurs when the timer's count value reaches the timer's compare value (*TMRn_CNT.count* = *TMRn_CMP.compare*). A rollover event indicates that a capture event did not occur within the set timer period. When a rollover event occurs, the timer hardware automatically performs the following actions during the next timer clock period:

- The *TMRn_CNT.count* field is set to 0x0000 0001.
- The *TMRn_INTFL.irq* field is set to 1.
- The timer remains enabled and continues counting.

Figure 18-7: Capture Mode Diagram


TIMER CLOCK
TMRn_CTRL0.en
TIMER INPUT SIGNAL
TMRn_CTRL0.pol_a = 0
TMRn_CTRL0.pol_a = 1
TMRN_CNT = TMRN_CMP TRIGGERS ROLLOVER TIMER PERIOD EVENT
TMRn_CMP.compare
TIMER INPUT SIGNAL TRIGGERS CAPTURE TIMER PERIOD EVENT TMRn_CNT COPIED TO TMRn_PWM
TMR_CNT(CAPTURE)
TMRn_CNT.count
0x0000 0002
0x0000 0001 †
0x0000 0000 †
TMRn_PWM.pwm NOT AFFECTED BY ROLLLOVER
TMRn_PWM.pwm
0x00000000
TMRN_CNT(CAPTURE)
APPLICATION FIRMWARE CLEARS TMRn_CNT
TMRn_INTFL.irq_a
APPLICATION FIRMWARE CLEARS INTERRUPT FLAG

This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 2 (Counter)

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

Configure the timer for Capture mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0.mode* to 4 to select capture mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0.pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. Write the initial value to *TMRn_CNT.count*, if desired.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT.count* = 0x0000 0001.
6 Write the compare value to the *TMRn_CMP.compare* field.
   a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
7. Select the capture event by setting *TMRn_CTRL1.capevent_sel*.
8. Enable the timer peripheral as described in *Timer Clock Sources*.

The timer period is calculated using the following equation:

Equation 18-10: Capture Mode Elapsed Time Calculation in Seconds

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

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

18.7.6 Compare Mode (5)

In compare mode, the timer peripheral increments continually from 0x0000 0000 (after the first timer period) to the maximum value of the 32- or 16-bit mode, then rolls over to 0x0000 0000 and continues incrementing. The end of timer period event occurs when the timer value matches the compare value, but the timer continues to increment until the count reaches 0xFFFF FFFF. The timer counter then rolls over and continues counting from 0x0000 0000.

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

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

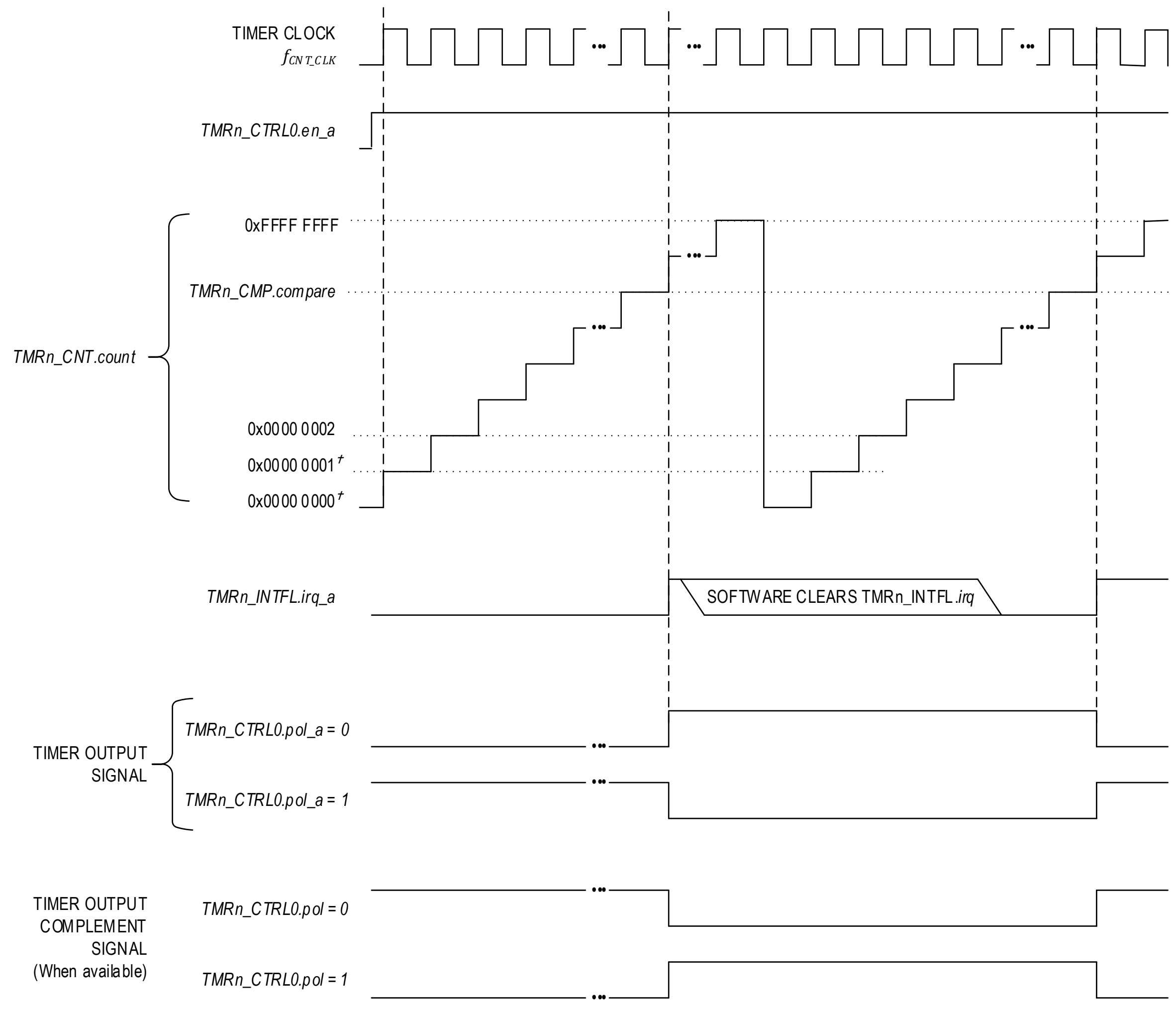
- Unlike other modes, *TMRn_CNT.count* is reset to 0x0000 00000 at the end of the timer period.
- The timer remains enabled and continues incrementing.
- The corresponding *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.
- Hardware toggles the state of the timer output signal.
- The timer output pin changes state if the timer output is enabled.

The compare mode timer period is calculated using *Equation 18-11*.

Equation 18-11: Compare Mode Timer Period in Seconds

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

Figure 18-8: 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 0x0000 0000 on a timer reset. *TMRn_CNT.count* reloads to 0x0000 0001 for all following timer periods.

Configure the timer for compare mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0.mode* to 5 to select Compare mode.
4. Set *TMRn_CTRL0.clkdiv* to set the prescaler that determines the timer frequency.
5. If using the timer output function:
    a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
    b. Configure the GPIO electrical characteristics as desired.
    c. Select the correct alternate function mode for the timer output pin.
6. If using the inverted timer output function:
    a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
    b. Configure the GPIO electrical characteristics as desired.
    c. Select the correct alternate function mode for the inverted timer output pin.
7. If using the timer interrupt, enable the corresponding field in the *TMRn_CTRL1* register.
8. Write the compare value to *TMRn_CMP.compare.*
    a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
9. If desired, write an initial value to *TMRn_CNT.count*.
    a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT.count* = 0x0000 0001.
10. Enable the timer peripheral as described in *Timer Clock Sources*.

18.7.7 Gated Mode (6)

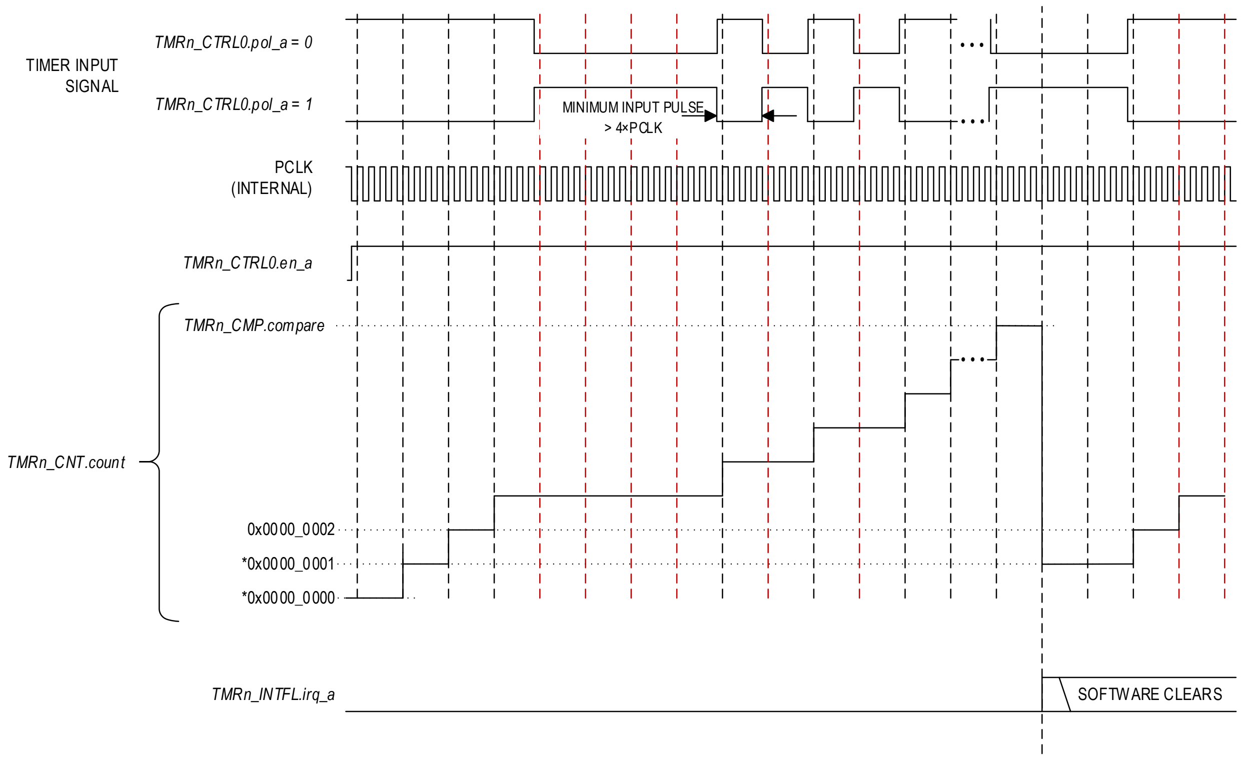
Gated mode is similar to continuous mode, except that *TMRn_CNT.count* only increments when the timer input signal is in its active state.

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

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

- The *TMRn_CNT.count* field is set to 0x0000 0001.
- The timer remains enabled and continues incrementing.
- If the timer output signal toggles state, the timer output pin changes state if the timer output is enabled.
- The corresponding *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

Figure 18-9: Gated Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 6 (Gated)

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

Configure the timer for gated mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0.mode* to 6 to select gated mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0.pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. If desired, write an initial value to the *TMRn_CNT.count* field.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT.count* = 0x0000 0001.
6 Write the compare value to *TMRn_CMP.compare*.
   a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

18.7.8 Capture/Compare Mode (7)

In Capture/Compare mode, the timer starts counting after the first external timer input transition occurs. The transition, a rising edge or falling edge on the timer's input signal, is set using the TMRn_CTRL0.*pol* bit.

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

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

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

If a transition on the timer pin caused the end of the timer period, hardware automatically performs the following:

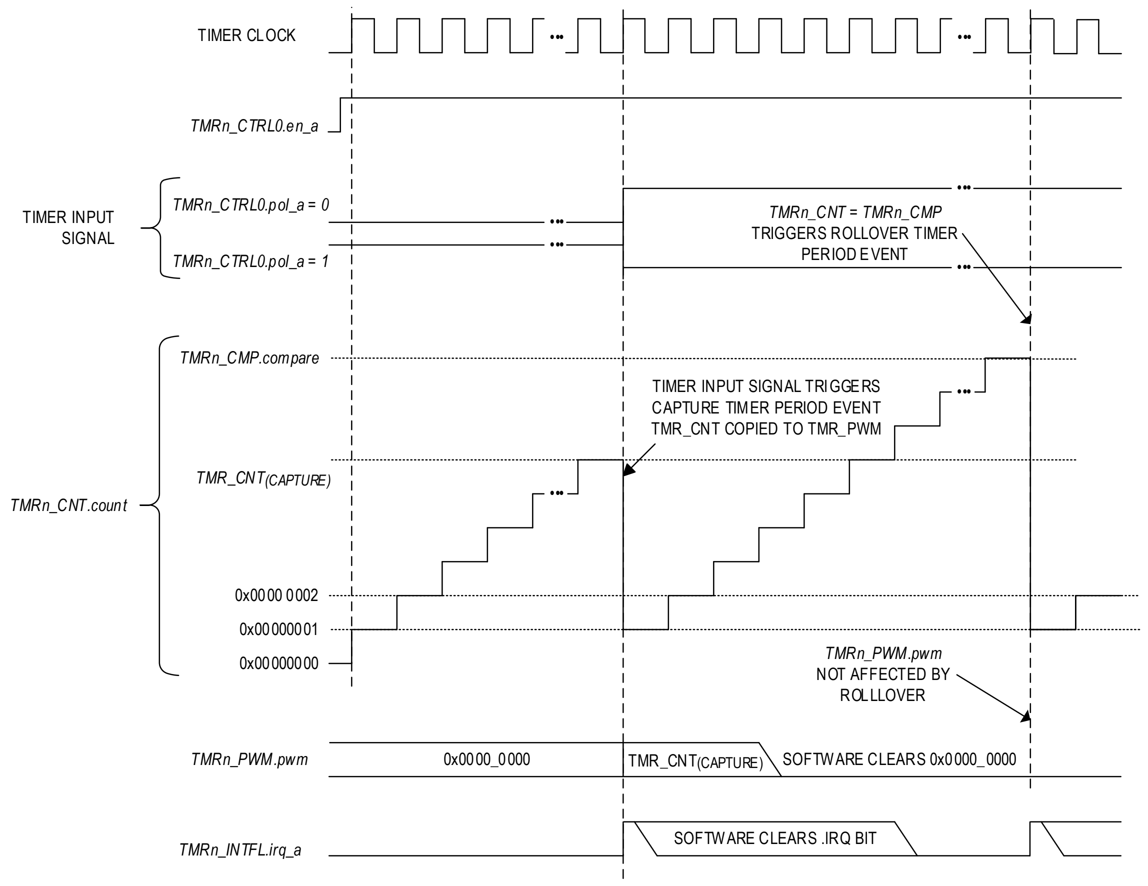
- The value in the TMRn_CNT.*count* field is copied to the TMRn_PWM.*pwm* field.
- The TMRn_CNT.*count* field is set to 0x0000 0001.
- The timer remains enabled and continues incrementing.
- The corresponding TMRn_INTFL.*irq* field is set to 1 to indicate a timer interrupt event occurred.

In capture/compare mode, the elapsed time from the timer start to the capture event is calculated using Equation 18-12.

Equation 18-12: Capture Mode Elapsed Time in Seconds

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

Figure 18-10: Capture/Compare Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 7 (CAPTURE/COMPARE)

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

Configure the timer for Capture/Compare mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0.mode* to 7 to select Capture/Compare mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0.pol* to select the positive edge (*TMRn_CTRL0.pol* = 1) or negative edge (*TMRn_CTRL0.pol* = 0) transition to cause the capture event..
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. If desired, write an initial value to the *TMRn_CNT.count* field.
   a. This effects only the first period; subsequent timer periods always reset *TMRn_CNT.count* = 0x0000 0001.
6 Write the compare value to *TMRn_CMP.compare*.
   a. Poll the *TMRn_INTFL.wrdone* field until it reads 1.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

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

18.7.9 Dual Edge Capture Mode (8)

Dual edge capture mode is similar to capture mode, except the counter can capture on both edges of the timer input pin.

18.7.10 Inactive Gated Mode (14)

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

18.8 Timer Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 18-6*. Register names for a specific instance are defined by replacing "n" with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 18-6: Timer Register Summary

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

18.8.1 Register Details

Table 18-7: Timer Count Register

Timer Count		**TMRn_CNT**		**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	count	R/W	0	**Timer Count** This field increments at a rate dependent on the selected timer operating mode. The function of the bits in this field is dependent on the 32-bit/16-bit configuration. Reads of this register always return the current value.

Table 18-8: Timer Compare Register

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

Table 18-9: Timer PWM Register

Timer PWM		**TMRn_PWM**		**[0x0008]**
Bits	**Field**	**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, when TMRn_CNT.*count* = TMRn_CMP.*compare*, the PWM output transitions to the second period of the PWM cycle. The second PWM period count is stored in TMRn_CMP.*compare*. TMRn_PWM.*pwm* must be less than TMRn_CMP.*compare* for PWM mode operation. **Timer Capture Value** In capture, compare, and capture/compare modes, this field is used to store the TMRn_CNT.*count* value when a Capture, Compare, or Capture/Compare event occurs.

Table 18-10: Timer Interrupt Register

Timer Interrupt		**TMRn_INTFL**		**[0x000C]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
24	wr_dis_b	R/W	0	**TimerB Write Protect in Dual Timer Mode** Set this field to 0 to write protect the TimerB fields in the TMRn_CNT.*count[31:16]* and TMRn_PWM.*pwm[31:16]*. When this field is set to 0, 32-bit writes to the TMRn_CNT and TMRn_PWM registers only modify the lower 16-bits associated with TimerA. 0: Enabled. 1: Disabled. *Note: This field always reads 0 if the timer is configured as a 32-bit cascade timer.*

Timer Interrupt			TMRn_INTFL	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
25	wrdone_b	R	0	**TimerB Write Done** This field is cleared to 0 by hardware when software performs a write to *TMRn_CNT.count[31:16]* or *TMRn_PWM.pwm[31:16]* when in dual timer mode. Wait until the field is set to 1 before proceeding. 0: Operation in progress. 1: Operation complete.
23:17	-	RO	0	**Reserved**
16	irq_b	R/W1C	0	**TimerB Interrupt Event** This field is set when a TimerB interrupt event occurs. Write 1 to clear. 0: No event. 1: Interrupt event occurred.
15:10	-	RO	0	**Reserved**
9	wr_dis_a	R/W	0	**TimerB Dual Timer Mode Write Protect** This field disables write access to the *TMRn_CNT.count[31:16]* and *TMRn_PWM.pwm[31:16]* fields so that only the 16 bits associated with updating TimerA are modified during writes to the *TMRn_CNT* and *TMRn_PWM* registers. 0: Enabled. 1: Disabled. *Note: This field always reads 0 if the timer is configured as a 32-bit cascade timer.*
8	wrdone_a	R	0	**TimerA Write Done** This field is cleared to 0 by hardware when software performs a write to *TMRn_CNT.count[31:16]* or *TMRn_PWM.pwm[31:16]* when in dual 16-bit timer mode. Wait until the field reads 1 before proceeding. 0: Operation in progress. 1: Operation complete.
7:1	-	RO	0	**Reserved**
0	irq_a	W1C	0	**TimerA Interrupt Event** This field is set when a TimerA interrupt event occurs. Write 1 to clear. 0: No event. 1: Interrupt event occurred.

Table 18-11: Timer Control 0 Register

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31	en_b	R/W	0	**TimerB Enable** 0: Disabled. 1: Enabled.
30	clken_b	R/W	0	**TimerB Clock Enable** 0: Disabled. 1: Enabled.
29	rst_b	W1	0	**TimerB Reset** 0: No action. 1: Reset TimerB.
28:24	-	RO	0	**Reserved**

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

Timer Control 0				TMRn_CTRL0 [0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
9	pwmsync_a	RO	0	**Reserved**
8	pol_a	R/W	0	**TimerA Polarity** This field selects the polarity of the timer's input and output signal. This setting is not used if the GPIO is not configured for the timer's alternate function. This field's usage and settings are operating mode specific. See the section *Operating Modes* for details on the mode selected.
7:4	clkdiv_a	R/W	0	**TimerA Prescaler Select** The *clkdiv_a* field selects a prescaler that divides the timer's clock source to set the timer's count clock as follows: $f_{CNT\_CLK} = f_{CLK\_SOURCE} / prescaler$ See the section *Operating Modes* to determine which modes use the prescaler. 0: 1 1: 2 2: 4 3: 8 4: 16 5: 32 6: 64 7: 128 8: 256 9: 512 10: 1024 11: 2048 12: 4096 13-15: Reserved
3:0	mode_a	R/W	0	**TimerA Mode Select** Set this field to the desired operating mode for TimerA. 0: One-shot. 1: Continuous. 2: Counter. 3: PWM. 4: Capture. 5: Compare. 6: Gated. 7: Capture/Compare. 8: Dual-edge capture. 9-11: Reserved. 12: Internally gated. 13-15: Reserved.

Table 18-12: Timer Non-Overlapping Compare Register

Timer Non-Overlapping Compare				TMRn_NOLCMP [0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	hi_b	RO	0	**Reserved**
23:16	lo_b	RO	0	**Reserved**

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

Table 18-13: Timer Control 1 Register

Timer Control 1		TMRn_CTRL1		[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31	cascade	R/W	0	**32-Bit Cascade Timer Enable** This field is only supported by timer instances with support for 32-bit cascade mode. 0: Dual 16-bit timers. 1: 32-bit cascade timer.
30	outben_b	R/W	0	**TimerB Output B Enable** Reserved.
29	outen_b	R/W	0	**TimerB Output Enable** Reserved.
28	we_b	R/W	0	**TimerB Wake-Up Function** 0: Disabled. 1: Enabled.
27	sw_capevent_b	R/W	0	**TimerB Software Event Capture** Write this field to 1 to initiate a software event capture when operating the timer in capture mode to perform a software event capture. 0: No event. 1: Reserved.
26:25	capevent_sel_b	R/W	0	**TimerB Event Capture Selection** Set this field to the desired capture event source. See *Table 18-2* for available capture event 0 and capture event 1 options. 0-3: Reserved.
24	ie_b	R/W	0	**TimerB Interrupt Enable** 0: Disabled. 1: Enabled.
23	negtrig_b	R/W	0	**TimerB Negative Edge Trigger for Event** 0: Rising-edge trigger 1: Falling-edge trigger
22:20	event_sel_b	R/W	0	**TimerB Event Selection** 0: Event disabled. 1-7: Reserved.
19	clkrdy_b	RO	0	**TimerB Clock Ready Status** This field is set to 1 after software enables the TimerA clock by writing 1 to the *TMRn_CTRL1*.*clken_b* field. 0: Timer clock not ready or synchronization in progress. 1: Timer clock is ready.
18	clken_b	RO	0	**TimerB Clock Enable** Write this field to 1 to enable the TimerB clock. 0: Timer not enabled or synchronization in progress. 1: Timer is enabled.

Timer Control 1				TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**	
17:16	clksel_b	R/W	0	**TimerB Clock Source** See *Table 18-1* for the clock sources supported by each instance. 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.	
15	-	RO	0	**Reserved**	
14	outben_a	R/W	0	**Output B Enable** Reserved.	
13	outen_a	R/W	0	**Output Enable** Reserved.	
12	we_a	R/W	0	**TimerA Wake-Up Function** 0: Disabled. 1: Enabled.	
11	sw_capevent_a	R/W	0	**TimerA Software Event Capture** 0: Normal operation. 1: Trigger software capture event.	
10:9	capevent_sel_a	R/W	0	**TimerA Event Capture Selection** Set this field to the desired capture event source. See *Table 18-2* for available capture event 0 and capture event 1 options. 0: Capture event 0. 1: Capture event 1. 2: Capture event 2. 3: Capture event 3.	
8	ie_a	R/W	0	**TimerA Interrupt Enable** 0: Disabled. 1: Enabled.	
7	negtrig_a	R/W	0	**TimerA Edge Trigger Selection for Event** 0: Positive-edge triggered. 1: Negative-edge triggered.	
6:4	event_sel_a	R/W	0	**TimerA Event Selection** 0: Event disabled. 1-7: Reserved.	
3	clkrdy_a	RO	0	**TimerA Clock Ready** This field is set to 1 after software enables the TimerA clock by writing 1 to the *TMRn_CTRL1.clken_a* field. 0: Timer not enabled or synchronization in progress. 1: TimerA clock is ready.	
2	clken_a	R/W	0	**TimerA Clock Enable** Write this field to 1 to enable the TimerA clock. 0: Timer not enabled or synchronization in progress. 1: Timer is enabled.	

Timer Control 1				TMRn_CTRL1 [0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
1:0	clksel_a	R/W	0	**Clock Source TimerA** See *Table 18-2* for the available clock options for each timer instance. 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.

Table 18-14: Timer Wake-Up Status Register

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

19. Watchdog Timer (WDT)

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

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

Each time the application performs a reset, as early as possible in the application software, the peripheral control register should be examined to determine if the reset was cause by at WDT late reset event (or WDT early reset event if the window function is supported). If so, software should take the desired action as part of its restart sequence.

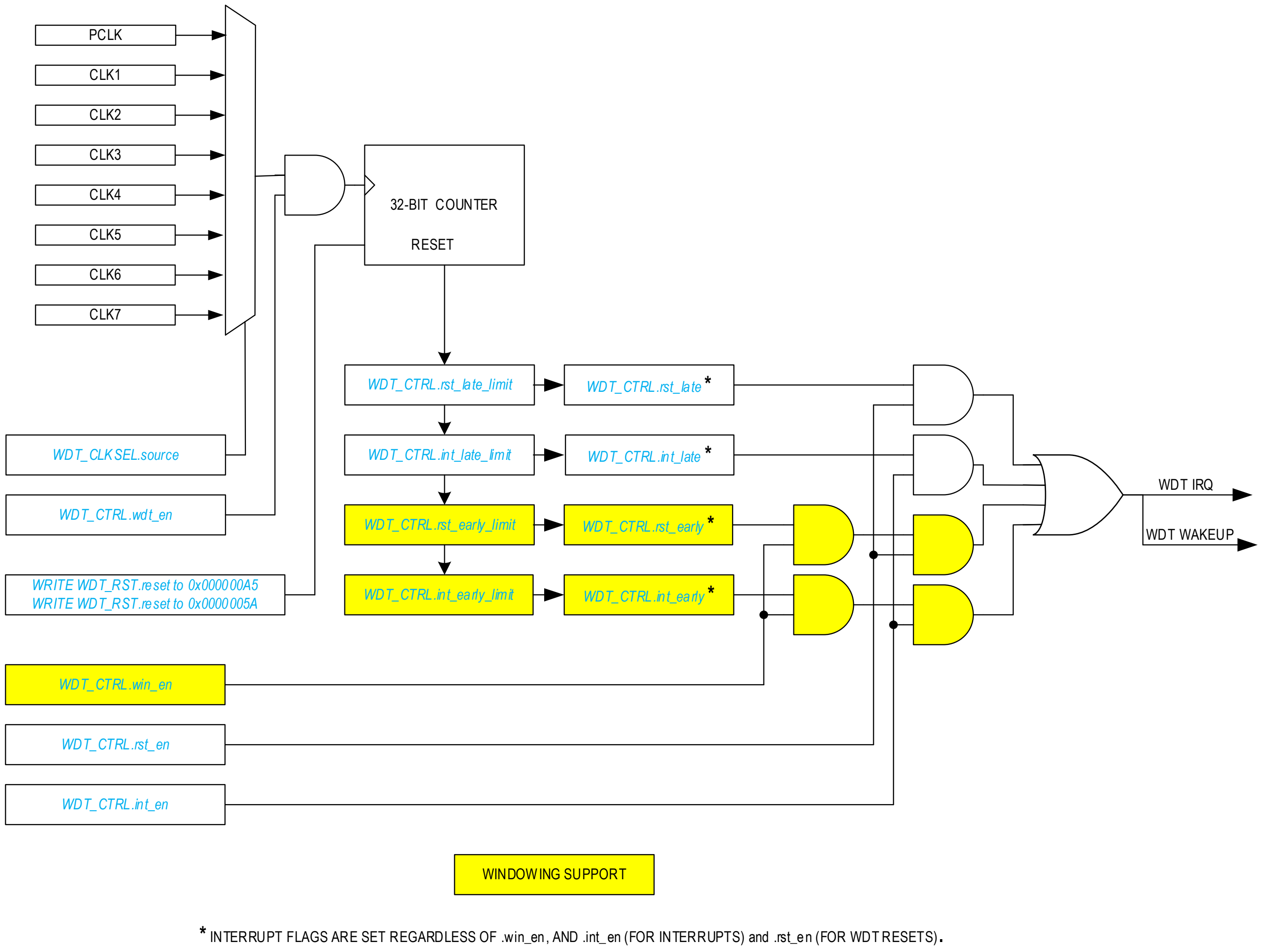
The WDT is a critical safety feature, and most fields are reset on POR or system reset events only.

The WDT includes the following features:

- Single-ended (legacy) watchdog timeout
- Windowed mode adds lower-limit timeout settings to detect loss of control in tight code loops.
- Configurable clock source
- Configurable time base
- Programmable upper and lower limits for reset and interrupts from 2^{16} to 2^{32} time base ticks
- New register to read the WDT counter register, simplifying code development

Figure 19-1 shows the block diagram of the WDT.

Figure 19-1: Windowed WDT Block Diagram



19.1 Instances

Table 19-1 shows the peripheral instance and available clock sources, and also indicates which instances support the windowed watchdog functionality.

Table 19-1: MAX32675 WDT Instances Summary

Instance	Window Support	Peripheral Clock	CLK1	CLK2	CLK3
WDT0	Yes	PCLK	IPO	IBRO	INRO
WDT1	Yes	PCLK	IPO	IBRO	INRO

The pin corresponding to the WDT external clock input only functions when the pin is configured for the required alternate function. Because GPIO pins reset to the I/O (not alternate function) state following a system reset, the WDT does not resume incrementing until the software reconfigures the GPIO for the correct alternate function.

19.2 Usage

When enabled, *WDTn_CNT.count* increments once every t_{WDTCLK} period. During correct operation, the WDT periodically executes the feed sequence and resets *WDTn_CNT.count* to 0x0000 0000 within the target window.

The upper and lower limits of the target window are user-configurable to accommodate different applications and non-deterministic execution times within an application.

The WDT can generate interrupt and reset events in response to the WDT activity. Interrupts are usually configured to respond first to an event outside the target window. The approach is that a minor system event may have temporarily delayed the execution of the feed sequence, so the event can be diagnosed in an interrupt routine and control returned to the system. When the WDT feed sequence occurs much earlier than expected or not at all, a reset event can be generated that forces the system to a known good state before continuing.

Traditional WDTs only detect execution errors that fail to perform the WDT feed sequence. If the counter reaches the WDT late interrupt threshold, the device attempts to regain program control by vectoring to the dedicated WDT interrupt handler. The interrupt handler should reset the WDT counter, perform the desired recovery activity, and then return execution to a known good address.

If the execution error prevents the successful execution of the interrupt handler, the WDT continues to increment until the count reaches the WDT late reset threshold. The WDT generates a late reset event which sets the WDT late reset flag and generates a system interrupt.

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

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

The event flags are set regardless of the state of the corresponding enable fields. This includes the early interrupt and early event flags, even if *WDTn_CTRL*.*win_en* = 0.

19.3 WDT Feed Sequence

The WDT feed sequence protects the system against unintentional altering of the WDT count and unintentional enabling or disabling of the timer itself.

Two consecutive write instructions to the *WDTn_RST*.*reset* field are required to reset *WDTn_CNT*.*count* = 0. Global interrupts should be disabled immediately before, and reenabled afterwards, to ensure both writes complete without interruption.

The feed sequence must also be performed immediately before enabling the WDT to prevent an accidental triggering of the reset or interrupt as soon as the timer is enabled. There is no timed access window for these write operations; the operations can be separated by any length of time as long as they occur in the required sequence.

1. Disable interrupts.
2. In consecutive write operations:
    a. Write *WDTn_RST*.*reset*: 0xA5
    b. Write *WDTn_RST*.*reset*: 0x5A
3. If desired, enable or disable the timer.
4. Re-enable interrupts.

19.4 WDT Events

Multiple events are supported as shown in *Table 19-2*. The corresponding event flag is set when the event occurs.

Typically, the system is configured such that the late interrupt events occur before late reset events, and early interrupts occur when the feed sequence has the least error from the target time, before early reset events.

The event flags are set regardless of the state of the corresponding enable fields. This includes the early interrupt and early event flags, even if *WDTn_CTRL.win_en* = 0.

Software must clear all the event flags before enabling the timers.

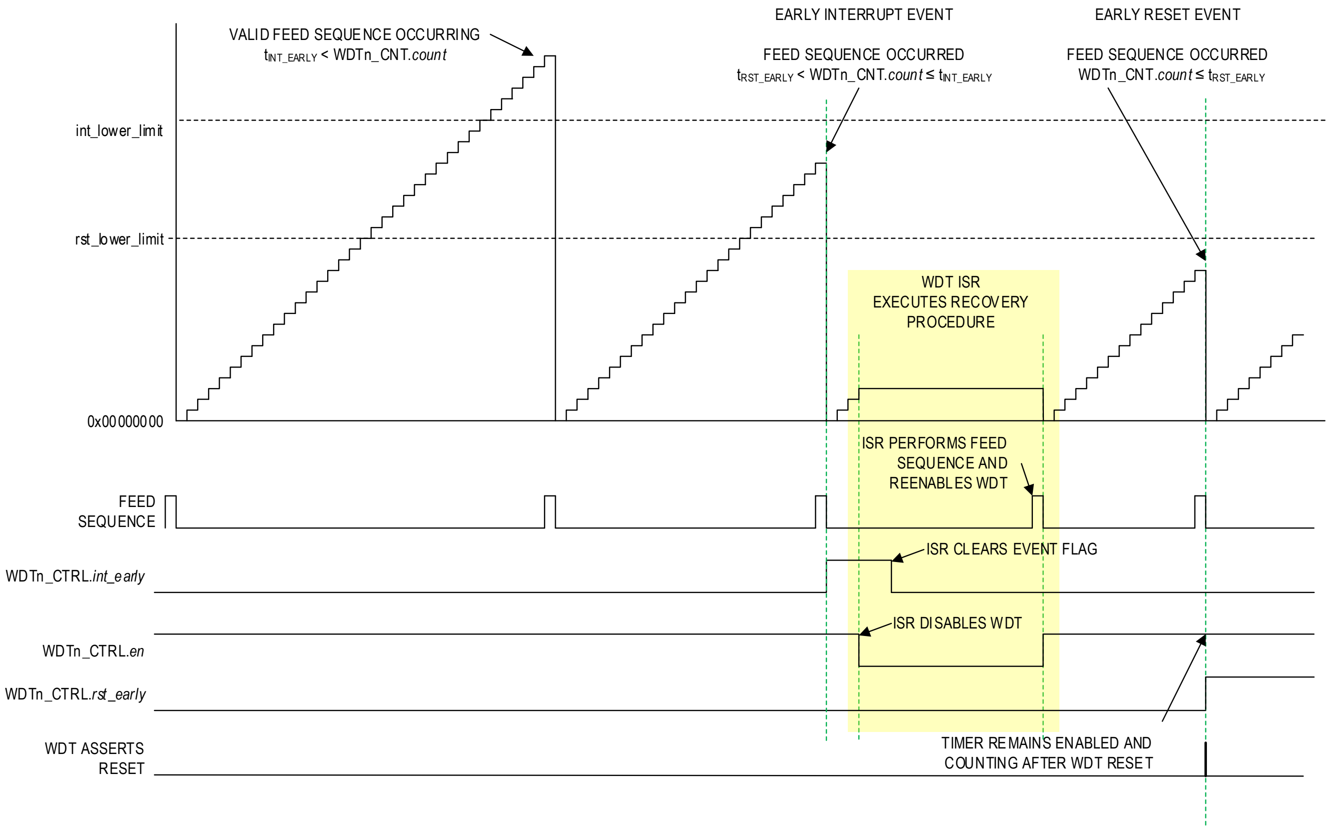
Table 19-2: MAX32675 WDT Event Summary

Event	Condition	Local Interrupt Event Flag	Local Interrupt Event Enable
Early Interrupt	Feed sequence occurs while *WDTn_CTRL.rst_early_val* ≤ *WDTn_CNT.count* < *WDTn_CTRL.int_early_val* *WDTn_CTRL.win_en = 1*	*WDTn_CTRL.int_early*	*WDTn_CTRL.wdt_int_en*
Early Reset	Feed sequence occurs while *WDTn_CNT.count* < *WDTn_CTRL.rst_early_val* *WDTn_CTRL.win_en = 1*	*WDTn_CTRL.rst_early*	*WDTn_CTRL.wdt_rst_en*
Interrupt Late	*WDTn_CNT.count* = *WDTn_CTRL.int_late_val*	*WDTn_CTRL.int_late*	*WDTn_CTRL.wdt_int_en*
Reset Late	*WDTn_CNT.count* = *WDTn_CTRL.rst_late_val*	*WDTn_CTRL.rst_late*	*WDTn_CTRL.wdt_rst_en*
Timer Enabled	*WDTn_CTRL.clkrdy* 0 -> 1	No event flags are set by a timer-enabled event	

19.4.1 WDT Early Reset

The early reset event occurs if software performs the WDT feed sequence while *WDTn_CNT*.count < *WDTn_CTRL.rst_late_val* threshold as shown in *Table 19-2*. *Figure 19-2* shows the sequencing details associated with an early reset event.

Figure 19-2: WDT Early Interrupt and Reset Event Sequencing Details


EARLY INTERRUPT EVENT
EARLY RESET EVENT
VALID FEED SEQUENCE OCCURRING
t_{INT_EARLY} < WDTn_CNT.*count*
FEED SEQUENCE OCCURRED
t_{RST_EARLY} < WDTn_CNT.*count* ≤ t_{INT_EARLY}
FEED SEQUENCE OCCURRED
WDTn_CNT.*count* ≤ t_{RST_EARLY}
int_lower_limit
rst_lower_limit
WDT ISR EXECUTES RECOVERY PROCEDURE
0x00000000
ISR PERFORMS FEED SEQUENCE AND REENABLES WDT
FEED SEQUENCE
ISR CLEARS EVENT FLAG
WDTn_CTRL.*int_early*
ISR DISABLES WDT
WDTn_CTRL.*en*
WDTn_CTRL.*rst_early*
TIMER REMAINS ENABLED AND COUNTING AFTER WDT RESET
WDT ASSERTS RESET

The following occurs when a WDT early reset event occurs:

1. The hardware sets WDTn_CTRL.*rst_early* to 1.
2. The hardware initiates a system reset:
   a. The hardware resets WDTn_CNT.*count* to 0x0000 0000 during the reset event.
3. The WDTn_CTRL.*en* field is unaffected by a system reset. The WDT continues incrementing.
4. The WDTn_CTRL.*rst_early* field is unaffected by a system reset.

19.4.2 WDT Early Interrupt

Figure 19-2 shows the sequencing details associated with an early reset event including the required functions performed by the WDT interrupt handler. The early interrupt event occurs if the software performs the WDT feed sequence while WDTn_CTRL.*rst_early_val* ≤ WDTn_CNT.*count* < WDTn_CTRL.*int_early_val*.

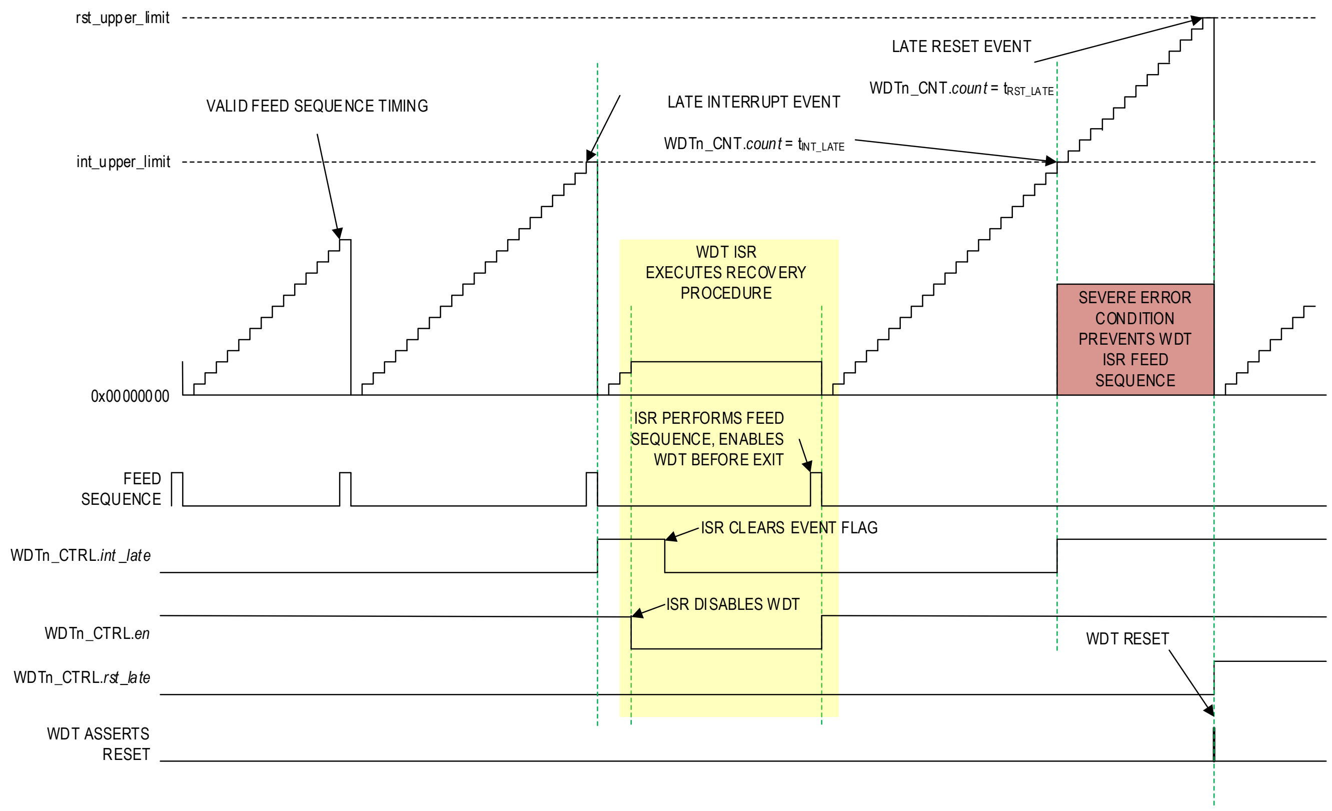
The following occurs when a WDT late interrupt event occurs:

1. The hardware sets WDTn_CTRL.*int_early* to 1.
2. The hardware initiates an interrupt if enabled.

19.4.3 WDT Late Reset

The late reset event occurs if the counter increments to the point where WDTn_CNT.*count* = WDTn_CTRL.*rst_late_val* threshold as shown in *Table 19-2*. *Figure 19-3* shows the sequencing details associated with a late reset event.

Figure 19-3: WDT Late Interrupt and Reset Event Sequencing Details


rst_upper_limit
int_upper_limit
0x00000000
VALID FEED SEQUENCE TIMING
LATE INTERRUPT EVENT
WDTn_CNT.count = tINT_LATE
LATE RESET EVENT
WDTn_CNT.count = tRST_LATE
WDT ISR EXECUTES RECOVERY PROCEDURE
SEVERE ERROR CONDITION PREVENTS WDT ISR FEED SEQUENCE
ISR PERFORMS FEED SEQUENCE, ENABLES WDT BEFORE EXIT
FEED SEQUENCE
ISR CLEARS EVENT FLAG
WDTn_CTRL.int_late
ISR DISABLES WDT
WDTn_CTRL.en
WDT RESET
WDTn_CTRL.rst_late
WDT ASSERTS RESET

The following occurs when a WDT late reset event occurs:

1. The hardware sets *WDTn_CTRL.rst_late* to 1.
2. The hardware initiates a system reset:
   a. The hardware resets *WDTn_CNT.count* to 0x0000 0000 during the reset event.
3. The *WDTn_CTRL.en* field is unaffected by a system reset. The WDT continues incrementing.
4. The *WDTn_CTRL.rst_late* field is unaffected by a system reset.

19.4.4 WDT Late Interrupt

The late reset event occurs if the counter increments to the point where *WDTn_CNT.count* = *WDTn_CTRL.rst_late_val* threshold as shown in *Table 19-2*. *Figure 19-3* shows the sequencing details associated with an late interrupt event, including the required functions performed by the WDT interrupt handler.

The following occurs when WDT late interrupt event occurs:

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

19.5 Initializing the WDT

The full procedure for configuring the WDT is shown as follows:

1. Execute the WDT feed sequence and disable the WDT.
2. Disable global interrupts.
3. Write *WDTn_RST.reset* to 0xA5.
4. Write *WDTn_RST.reset* to 0x5A. The hardware resets *WDTn_CNT.count* to 0x0000 0000.
5. Set *WDTn_CTRL.en* to 0 to disable the WDT.
6. Verify the peripheral is disabled before proceeding:
   a. Poll *WDTn_CTRL.clkrdy* until it reads 1

   or,

   b. Set *WDTn_CTRL.clkrdy_ie* = 1 to generate a WDT enabled interrupt event.
7. Re-enable global interrupts.
8. Configure *WDTn_CLKSEL.source* to select the clock source.
9. Configure the traditional/legacy thresholds:
   a. Configure *WDTn_CTRL.int_late_val* to the desired threshold for the WDT late interrupt event.
   b. Configure *WDTn_CTRL.rst_late_val* to the desired threshold for the WDT late reset event.
10. If using the optional windowed WDT feature:
11. Set *WDTn_CTRL.win_en* = 1 to enable the windowed WDT feature:
    a. Configure *WDTn_CTRL.int_early_val* to the desired threshold for the WDT early interrupt event.
    b. Configure *WDTn_CTRL.rst_early_val* to the desired threshold for the WDT early reset event.
12. Set *WDTn_CTRL.wdt_int_en* to generate an interrupt when a WDT late interrupt event occurs. If *WDTn_CTRL.win_en* = 1, an interrupt is generated by both a WDT late interrupt event and also a WDT early interrupt event.
13. Set *WDTn_CTRL.wdt_rst_en* to generate an interrupt when a WDT late reset event occurs. If *WDTn_CTRL.win_en* = 1, an interrupt is generated by a WDT late reset event and also a WDT early reset event.
14. Execute the WDT feed sequence and enable the WDT:
15. Disable global interrupts.
16. Write *WDTn_RST.reset* to 0xA5.
17. Write *WDTn_RST.reset* to 0x5A. The hardware resets *WDTn_CNT.count* to 0x0000 0000.
18. Set *WDTn_CTRL.en* to 1 to enable the WDT.
19. Verify the peripheral is enabled before proceeding:
    a. Poll *WDTn_CTRL.clkrdy* until it reads 1, or
    b. Set *WDTn_CTRL.clkrdy_ie* = 1 to generate a WDT-enabled event interrupt.
20. Re-enable global interrupts.

19.6 Resets

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

19.7 Using the WDT as a Long-Interval Timer

One application of the WDT is as a very long interval timer in ACTIVE mode. The timer can be configured to generate a WDT late interrupt event for as long as 2^{32} periods of the selected watchdog clock source. The WDT should not be enabled to generate WDT reset events in this application.

19.8 Using the WDT as a Long-Interval Wake-Up Timer

Another application of the WDT is as a very long interval wake-up source from *SLEEP*. The timer can be configured to generate a WDT wake-up event for as long as 2^{32} periods of the selected watchdog clock source.

19.9 WDT Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 19-3*. Register names for a specific instance are defined by replacing "n" with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 19-3: WDT Register Summary

Offset	Register	Name
[0x0000]	*WDTn_CTRL*	*WDT Control Register*
[0x0004]	*WDTn_RST*	*WDT Reset Register*
[0x0008]	*WDTn_CLKSEL*	*WDT Clock Select Register*
[0x000C]	*WDTn_CNT*	*WDT Count Register*

19.9.1 Register Details

Table 19-4: WDT Control Register

WDT Control				**WDTn_CTRL**	**[0x0000]**
Bits	**Name**	**Access**	**Reset**	**Description**	
31	rst_late	R/W	0	**Reset Late Event** A watchdog reset event occurred after the time specified in *WDTn_CTRL.rst_late_val*. This flag is set even if *WDTn_CTRL.win_en* = 0 or *WDTn_CTRL.wdt_rst_en* = 0. The application software has to clear the flag appropriately in case of carried over flags from prior operations. 0: Watchdog did not cause reset event. 1: Watchdog reset occurred after *WDTn_CTRL.rst_early_val*.	
30	rst_early	R/W	0	**Reset Early Event** A watchdog reset event occurred before the time specified in *WDTn_CTRL.rst_early_val*. This flag is set even if *WDTn_CTRL.win_en* = 0 or *WDTn_CTRL.wdt_rst_en* = 0. The application software has to clear the flag appropriately in case of carried over flags from prior operations. 0: Watchdog did not cause reset event. 1: Watchdog reset occurred before *WDTn_CTRL.rst_early_val*.	
29	win_en	R/W	0	**Window Function Enable** 0: Disabled. WDT recognizes interrupt late and reset late events, supporting legacy implementations. 1: Enabled.	
28	clkrdy	R	0	**Clock Status** This field is cleared by hardware when software changes the state of *WDTn_CTRL.en*. Hardware sets the field to 1 when the change to the requested enable/disable is complete. 0: WDT clock is off. 1: WDT clock is on.	

WDT Control				WDTn_CTRL [0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
27	clkrdy_ie	R/W	0	**Clock Switch Ready Interrupt Enable** This interrupt prevents the user from needing to poll the *WDTn_CTRL.clkrdy* all the time. When *WDTn_CTRL.clkrdy* goes from high to low, the interrupt signals the transition is complete. 0: Disabled. 1: Enabled.
26:24	-	RO	0	**Reserved**
23:20	rst_early_val	R/W	0	**Reset Early Event Threshold** 0x0: 2^{31} x t_{WDTCLK} 0x1: 2^{30} x t_{WDTCLK} 0x2: 2^{29} x t_{WDTCLK} 0x3: 2^{28} x t_{WDTCLK} 0x4: 2^{27} x t_{WDTCLK} 0x5: 2^{26} x t_{WDTCLK} 0x6: 2^{25} x t_{WDTCLK} 0x7: 2^{24} x t_{WDTCLK} 0x8: 2^{23} x t_{WDTCLK} 0x9: 2^{22} x t_{WDTCLK} 0xA: 2^{21} x t_{WDTCLK} 0xB: 2^{20} x t_{WDTCLK} 0xC: 2^{19} x t_{WDTCLK} 0xD: 2^{18} x t_{WDTCLK} 0xE: 2^{17} x t_{WDTCLK} 0xF: 2^{16} x t_{WDTCLK} *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*
19:16	int_early_val	R/W	0	**Interrupt Early Event Threshold** 0x0: 2^{31} x t_{WDTCLK} 0x1: 2^{30} x t_{WDTCLK} 0x2: 2^{29} x t_{WDTCLK} 0x3: 2^{28} x t_{WDTCLK} 0x4: 2^{27} x t_{WDTCLK} 0x5: 2^{26} x t_{WDTCLK} 0x6: 2^{25} x t_{WDTCLK} 0x7: 2^{24} x t_{WDTCLK} 0x8: 2^{23} x t_{WDTCLK} 0x9: 2^{22} x t_{WDTCLK} 0xA: 2^{21} x t_{WDTCLK} 0xB: 2^{20} x t_{WDTCLK} 0xC: 2^{19} x t_{WDTCLK} 0xD: 2^{18} x t_{WDTCLK} 0xE: 2^{17} x t_{WDTCLK} 0xF: 2^{16} x t_{WDTCLK} *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*
15:13	-	RO	0	**Reserved**
12	int_early	R/W	0	**Interrupt Early Flag** A feed sequence was performed earlier than the time determined by the *WDTn_CTRL.int_early_val* field. This flag is set even if *WDTn_CTRL.win_en* = 0. 0: No interrupt event. 1: Interrupt event occurred. Generates a WDT interrupt if *WDTn_CTRL.wdt_int_en* = 1.

WDT Control				WDTn_CTRL [0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
11	wdt_rst_en	R/W	0	**WDT Reset Enable** 0: Disabled. 1: Enabled.
10	wdt_int_en	R/W	0	**WDT Interrupt Enable** 0: Disabled. 1: Enabled.
9	int_late	R/W	0	**Interrupt Late Flag** A watchdog feed sequence did not occur before the time determined by the *WDTn_CTRL.int_late_val* field. 0: No interrupt event. 1: Interrupt event occurred. Generates a WDT interrupt if *WDTn_CTRL.wdt_int_en* = 1.
8	en	R/W	0	**WDT Enable** This field enables/disables the watchdog timer clock into the peripheral. The watchdog timer's counter, *WDTn_CNT.count*, holds its value while the watchdog timer is disabled. The watchdog timer feed sequence must be performed immediately before any change to this field. 0: Disabled. 1: Enabled.
7:4	rst_late_val	R/W	0	**Reset Late Event Threshold** 0x0: 2^{31} x t_{WDTCLK} 0x1: 2^{30} x t_{WDTCLK} 0x2: 2^{29} x t_{WDTCLK} 0x3: 2^{28} x t_{WDTCLK} 0x4: 2^{27} x t_{WDTCLK} 0x5: 2^{26} x t_{WDTCLK} 0x6: 2^{25} x t_{WDTCLK} 0x7: 2^{24} x t_{WDTCLK} 0x8: 2^{23} x t_{WDTCLK} 0x9: 2^{22} x t_{WDTCLK} 0xA: 2^{21} x t_{WDTCLK} 0xB: 2^{20} x t_{WDTCLK} 0xC: 2^{19} x t_{WDTCLK} 0xD: 2^{18} x t_{WDTCLK} 0xE: 2^{17} x t_{WDTCLK} 0xF: 2^{16} x t_{WDTCLK} *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*

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

Table 19-5: WDT Reset Register

WDT Reset				WDTn_RST	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:8	-	RO	0	**Reserved**	
7:0	reset	R	0*	**Reset Watchdog Timer Count** Writing the WDT feed sequence in two consecutive write instructions to this register resets the internal counter to 0x0000 0000. 1. Write *WDTn_RST.reset*: 0xA5. 2. Write *WDTn_RST.reset*: 0x5A. Writes to the *WDTn_CTRL*.en field, which enables or disables the WDT; must be the next instruction following the WDT feed sequence. **Note: This field is set to 0 on a POR and is not affected by other resets.*	

Table 19-6: WDT Clock Source Select Register

WDT Clock Source Select				WDTn_CLKSEL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:3	-	RO	0	**Reserved**	
2:0	source	R/W	0*	**Clock Source Select** See *Table 19-1* for available clock options. 0: PCLK. 1: CLK1. 2: CLK2. 3: CLK3. 4: CLK4. 5: CLK5. 6: CLK6. 7: CLK7. **Note: This field is only reset on a POR and unaffected by other resets.* *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*	

Table 19-7: WDT Count Register

WDT Count				**WDTn_CNT**	**[0x000C]**
Bits	**Name**	**Access**	**Reset**	**Description**	
31:0	count	R	0	**WDT Counter** The counter value for debug. This register is reset by system reset, as well as the watchdog feeding sequence. When the WDT clock is off, the feeding sequence generates an asynchronous reset of 1 pclk width. When the WDT clock is on, the feeding sequence generates a synchronous reset that is a handshake to the WDT clock domain. *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*	

20. Cyclic Redundancy Check (CRC)

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

The features include:

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

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

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

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

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

20.1 Instances

Instances of the peripheral are listed in *Table 20-1*.

Table 20-1: MAX32675 CRC Instances

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

20.2 Usage

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

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

Because the receiving end calculates a new CRC on both the data and received CRC, send the received CRC in the correct order, so the highest-order term of the CRC is shifted through the generator first. Because data is typically shifted through the generator LSB first, the CRC is reversed bitwise, with the highest-order term of the remainder in the LSB position. Software CRC algorithms typically manage this by calculating everything backward. The software reverses the polynomial and does right shifts on the data. The resulting CRC is bit swapped and in the correct format.

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

When calculating the CRC on data LSB first, the polynomial should be reversed so that the coefficient of the highest power term is in the LSB position. The largest term, x^n, is implied (always one) and should be omitted when writing to the *CRC_POLY* register. This is necessary because the polynomial is always one bit larger than the resulting CRC, so a 32-bit CRC has a polynomial with 33 terms ($x^0 \dots x^{32}$).

Table 20-2: Organization of Calculated Result in the CRC_VAL.value Field

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

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

20.3 Polynomial Generation

The CRC can be configured for any polynomial up to x^{32} (33 terms) by writing to the *CRC_POLY.poly* field. *Table 20-3* shows common CRC polynomials.

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

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

Table 20-3: Common CRC Polynomials

Algorithm	**Polynomial Expression**	**Order**	**Polynomial**
CRC-32 Ethernet	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x^{1}+x^{0}$	LSB	0xEDB8 8320
CRC-CCITT	$x^{16}+x^{12}+x^{5}+x^{0}$	LSB	0x0000 8408
CRC-16	$x^{16}+x^{15}+x^{2}+x^{0}$	LSB	0x0000 A001
USB Data	$x^{16}+x^{15}+x^{2}+x^{0}$	MSB	0x8005 0000
Parity	$x^{1}+x^{0}$	LSB	0x0000 0001

20.4 Software CRC Calculations

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

Use the following procedure to calculate a CRC:

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

20.5 DMA CRC Calculations

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

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

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

20.6 Registers

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

Table 20-4: CRC Register Summary

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

20.6.1 Register Details

Table 20-5: CRC Control Register

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

Table 20-6: CRC 8-Bit Data Input Register

CRC 8-Bit Data Input			**CRC_DATAIN8**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	data	W	0	**CRC Data Input** Write 8-bit values to this register to calculate 8-bit CRCs. See *Table 20-2* for details on the byte and bit ordering of the data in this register. *Note: Do not write to this register if CRC_CTRL.busy = 1 or CRC_CTRL.en = 0.*

Table 20-7: CRC 16-Bit Data Input Register

CRC Data 16-Bit Input			**CRC_DATAIN16**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	data	W	0	**CRC Data Input** Write 16-bit values to this register to calculate 16-bit CRCs. See *Table 20-2* for details on the byte and bit ordering of the data in this register. *Note: Do not write to this register if CRC_CTRL.busy = 1 or CRC_CTRL.en = 0.*

Table 20-8: CRC 32-Bit Data Input Register

CRC 32-Bit Data Input			**CRC_DATAIN32**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	W	0	**CRC Data Input** Write 32-bit values to this register to calculate 32-bit CRCs. See *Table 20-2* for details on the byte and bit ordering of the data in this register. *Note: Do not write to this register if CRC_CTRL.busy = 1 or CRC_CTRL.en = 0.*

Table 20-9: CRC Polynomial Register

CRC Polynomial				**CRC_POLY**	**[0x0008]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	poly	R/W	0xEDB8 8320	**CRC Polynomial** See *Table 20-2* for details on the byte and bit ordering of the data in this register.	

Table 20-10: CRC Value Register

CRC Value				**CRC_VAL**	**[0x000C]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	value	R/W	0	**Current CRC Value** The software can write to this register to set the initial state of the accelerator. This register should only be read or written when *CRC_CTRL.busy* = 0. See *Table 20-2* for details on the byte and bit ordering of the data in this register.	

21. Advanced Encryption Standard (AES)

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

The features include:

- Supports multiple key sizes:
  - 128-bit
  - 192-bit
  - 256-bit
- DMA support for both input and output channels
- Supports multiple key sources:
  - Encryption using an external AES key
  - Decryption using an external AES key
  - Decryption using an internally generated decryption key for decryption of data previously encrypted

21.1 Instances

Instances of the peripheral are listed in *Table 21-1*. Disable the peripheral by clearing *AES_CTRL.en* to 0 before writing to any register field.

Table 21-1: MAX32675 AES Instances

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

21.2 Encryption of 128-Bit Blocks of Data Using FIFO

AES operations are typically performed on 128 bits of data at a time. The *AES_CTRL.start* field is not used for 128-bit blocks.

1. Generate a key.
2. Wait for hardware to clear *AES_STATUS.busy* to 0.
3. Clear *AES_CTRL.en* to 0 to disable the peripheral.
4. If *AES_STATUS.input_em* is 0, set *AES_CTRL.input_flush* to 1 to flush the input FIFO.
5. If *AES_STATUS.output_em* is 0, set *AES_CTRL.output_flush* to 1 to flush the output FIFO.
6. Set *AES_CTRL.key_size* to the required setting
7. Set *AES_CTRL.type* to 0 (encryption with external key)
8. If interrupts are desired, set *AES_INTEN.done* to 1 so that an interrupt triggers at the end of the AES calculation.
9. Set *AES_CTRL.en* to 1 to enable the peripheral.
10. Write four 32-bit words of data to the *AES_FIFO.data* field.
    a. When complete, the hardware starts the AES calculation.
11. If *AES_INTEN.done* is 1, an interrupt triggers after the AES calculation is complete.
12. If *AES_INTEN.done is* 0, software should poll until *AES_STATUS.busy* reads 0.
13. Read the encrypted data as four 32-bit words from the *AES_FIFO.data register* (least significant word first).
14. Steps 9-13 may be repeated until all 128-bit blocks are processed.

21.3 Encryption of 128-Bit Blocks Using DMA

For this example, it is assumed that the DMA reads and writes data to and from the AES engine. However, the use of DMA is not a requirement. The application could use DMA on one side of the AES process and software on the other if that is desired. It is required that for each DMA_TX_REQ, the DMA must write four 32-bit words of data into the AES engine, and for each DMA_RX_REQ, the DMA must read four 32-bit words of data out of the AES engine.

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

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

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

The procedure to use DMA encryption/decryption is:

1. Generate a key.
2. Initialize the AES receive and transmit channels for the DMA controller.
3. Wait for the hardware to clear *AES_STATUS.busy* to 0.
4. Clear *AES_CTRL.en* to 0 to disable the peripheral.
5. If *AES_STATUS.input_em* is 0, set *AES_CTRL.input_flush* to 1 to flush the input FIFO.
6. If *AES_STATUS.output_em* is 0, set *AES_CTRL.output_flush* to 1 to flush the output FIFO.
7. Set *AES_CTRL.key_size* to the required setting.
8. Configure *AES_CTRL.type* to 0 (encryption with external key).
9. Ensure *AES_INTEN.done* is 0 during DMA operations.
10. Set *AES_CTRL.en* to 1 to enable the peripheral.
11. The DMA fills the FIFO, and the hardware begins the AES calculation.
12. When an AES calculation is completed, the AES signals to the DMA that the data output FIFO is full and that it should be emptied. If the DMA does not empty the FIFO before the next calculation completes, hardware set *AES_STATUS.output_full* to 1.

Steps 11 and 12 are repeated if the DMA has new data to write to the data input FIFO.

Note: The interface from DMA to the AES only works when the amount of data is a multiple of 128 bits. For non-multiples of 128 bits, the remainder after calculating all the 128-bit blocks must be encrypted by software.

21.4 Encryption of Blocks Less Than 128 Bits

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

1. Generate a key.
2. Wait for the hardware to clear *AES_STATUS.busy* to 0.
3. Clear *AES_CTRL.en* to 0 to disable the peripheral.
4. If *AES_STATUS.input_em* is 0, set *AES_CTRL.input_flush* to 1 to flush the input FIFO.
5. If *AES_STATUS.output_em* is 0, set *AES_CTRL.output_flush* to 1 to flush the output FIFO.
6. Set *AES_CTRL.key_size* to the required setting.
7. Configure *AES_CTRL.type* to 0 (encryption with external key).
8. If interrupts are required, set *AES_INTEN.done* to 1 so that an interrupt triggers at the end of the AES calculation.
9. Set *AES_CTRL.en* to 1 to enable the peripheral.
10. Write one to three 32-bit words of data to *AES_FIFO.data* (least significant word first).
11. Manually start the calculation by setting *AES_CTRL.start* to 1.
12. If *AES_INTEN.done* is 1, an interrupt triggers after the AES calculation completes.
13. If *AES_INTEN.done* is 0, software should poll until *AES_STATUS.busy* reads 0.
14. Read four 32-bit words from *AES_FIFO.data* (least significant word first).

21.5 Decryption

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

- Decryption with an external key
- Decryption with an internal decryption key

The internal decryption key is generated during an encryption operation. Therefore, it may be necessary to complete a dummy encryption operation before performing the first decryption operation to ensure that the internal decryption key has been generated.

21.6 Interrupt Events

The peripheral generates interrupts for the events shown in *Table 21-2*.

Multiple events may set an interrupt flag and generate an interrupt if the corresponding interrupt enable is set. The interrupt flags must be cleared by software.

Table 21-2: MAX32675 Interrupt Events

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

21.6.1 Data Output FIFO Overrun

When an AES calculation is completed, the AES engine signals to the DMA that the data output FIFO is full and should be emptied. If the DMA does not empty the FIFO before the following calculation completes a data output FIFO overrun event occurs, and the corresponding interrupt flag is set.

21.6.2 Key Zero

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

21.6.3 Key Change

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

21.6.4 Calculation Done

The transition of *AES_STATUS.busy* from 1 to 0 generates a calculation done event. The calculation done interrupt must be disabled when using DMA.

21.7 Wake-Up Events

The peripheral does not generate wake-up events.

21.8 AES Registers

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

Table 21-3: AES Register Summary

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

21.8.1 Register Details

Table 21-4: AES Control Register

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

AES Control				AES_CTRL [0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
5	output_flush	R/W1O	0	**Flush Data Output FIFO** This field always reads 0. 0: Normal operation. 1: Flush.
4	input_flush	R/W1O	0	**Flush Data Input FIFO** This field always reads 0. 0: Normal operation. 1: Flush.
3	start	R/W1O	0	**Start AES Calculation** This field forces the start of an AES calculation regardless of the state of the data input FIFO, enabling AES calculations on less than 128 bits of data. This field always reads 0. 0: Normal operation. 1: Start calculation.
2	dma_tx_en	R/W	0	**DMA Request To Write Data Input FIFO** 0: Disabled. 1: Enabled. DMA request is generated if the data input FIFO is empty.
1	dma_rx_en	R/W	0	**DMA Request To Read Data Output FIFO** 0: Disabled. 1: Enabled. DMA request is generated if the data output FIFO is full.
0	en	R/W	0	**AES Enable** 0: Disabled. 1: Enabled.

Table 21-5: AES Status Register

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

Table 21-6: AES Interrupt Flag Register

AES Interrupt Flag				AES_INTFL [0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	ov	R/W1C	0	**Data Output FIFO Overrun Event Interrupt** 0: Normal operation. 1: Event occurred.

AES Interrupt Flag				**AES_INTFL**	**[0x0008]**
Bits	**Field**	**Access**	**Reset**	**Description**	
2	key_zero	R/W1C	0	**Key Zero Event Interrupt** 0: Normal operation. 1: Event occurred.	
1	key_change	R/W1C	0	**Key Change Event Interrupt** 0: Normal operation. 1: Event occurred.	
0	done	R/W1C	0	**Calculation Done Event Interrupt** 0: Normal operation. 1: Event occurred.	

Table 21-7: AES Interrupt Enable Register

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

Table 21-8: AES FIFO Register

AES Data				**AES_FIFO**	**[0x0010]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	data	R/W	0	**AES FIFO** Writing this register puts data to the data input FIFO. Hardware automatically starts a calculation after 4 words are written to this FIFO. The data must be written least significant word first. Reading this register removes data from the data output FIFO, least significant word first.	

22. TRNG Engine

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

The general information in this section is provided only for completeness; customers are expected to use the Analog Devices UCL for the generation of random numbers.

The TRNG engine can also be used to generate AES keys.

22.1 TRNG Registers

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

Table 22-1: TRNG Register Summary

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

22.1.1 Register Details

Table 22-2: TRNG Control Register

Cryptographic Control Register		**TRNG_CTRL**		**[0x0000]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	keywipe	R	0	**AES Key Wipe** When this bit is set, the key for securing SRAM is erased. This bit is cleared by hardware once the operation is completed. 0: No effect. 1: Initiate AES key wipe.
14:4	-	RO	0	**Reserved**
3	keygen	W1	0	**AES Key Generate** When this bit is set, the key for securing SRAM is generated and transferred to the secure key register automatically without user visibility or intervention. This bit is cleared by hardware once the key has been transferred to the secure key register. 0: No effect. 1: Initiate AES key generation.
2	-	RO	0	**Reserved**
1	rnd_ie	R/W	0	**Random Number Interrupt Enable** This bit enables an interrupt to be generated when *TRNG_STATUS.rdy* = 1. 0: Disabled. 1: Enabled.
0	-	RO	0	**Reserved**

Table 22-3: TRNG Status Register

TRNG Status Register				**TRNG_STATUS**	**[0x0004]**
Bits	**Name**	**Access**	**Reset**	**Description**	
31:1	-	RO	0	**Reserved**	
0	rdy	R	0	**Random Number Ready** This bit is automatically cleared to 0 and a new random number is generated when *TRNG_DATA.data* is read. 0: Random number generation in progress. The content of *TRNG_DATA.data* is invalid. 1: *TRNG_DATA.data* contains new 32-bit random data. An interrupt is generated if *TRNG_CTRL.rnd_ie* = 1.	

Table 22-4: TRNG Data Register

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

23. Bootloader

The ROM bootloader provides for program loading and verification. The physical interface between the external host and the bootloader is by default the UART.

The secure bootloader (SBL) employs a hash-based message authentication code (HMAC SHA-256) to guarantee both the authenticity of downloaded program files and the integrity of internal program memory after each reset.

All versions of the bootloader provide the ability to block read/write access to program memory.

Bootloader features:

- Common functionality of bootloader and SBL
- Checksum verification of ROM image before further ROM execution
- SWD disabled in LOCKED and PERMLOCKED states
- Programmable through UART or SWD interface
- UART operates at 115200/8-N-1
- LOCKED mode disables SWD and disallows any change to flash through the bootloader
- Unlock erases all flash and secrets before unlocking SWD
- Optional PERMLOCKED state only allows for program validation Lock

Secure Bootloader (SBL) features:

- Secure HMAC SHA-256 with secret key-based transactions.
- Trusted secure boot provides automatic program memory verification and authentication before execution after every reset
- Integrity and authentication verification of program memory downloads
- Optional challenge/response gating entry to bootloader

23.1 Instances

The dedicated pins and features of the bootloader are shown *Table 23-1*.

Table 23-1: MAX32675 Bootloader Instances

Part Number	**Activation Pins**	**Bootloader**	**Secure Bootloader**	**Secure Boot**	**Flash Memory Page Size**
	UART0 RX	**SWDCLK**			
MAX32675	P0.8	P0.1	Yes	No	8KB

23.2 Bootloader Operating States

Each bootloader supports the modes shown in *Table 23-2*. Each bootloader mode has a unique prompt.

Table 23-2: MAX32675 Bootloader Operating States and Prompts

State	**Bootloader**	**Secure Bootloader**	**Recognized Commands**	**Prompt**
UNLOCKED	Yes	Yes	All Commands U/L/P	“ULDR> “ <0x55> <0x4C> <0x44> <0x52> <0x3E> <0x20>
LOCKED	Yes	Yes	Only L/P	“LLDR> “ <0x4C> <0x4C> <0x44> <0x52> <0x3E> <0x20>
PERMLOCK	Yes	Yes	Only P	“PLLDR> “

State	Bootloader	Secure Bootloader	Recognized Commands	Prompt
				<0x50> <0x4C> <0x4C> <0x44> <0x52> <0x3E> <0x20>
CHALLENGE	No	Yes	*GC – Get Challenge* *SR – Send Response*	"<CR> " <0x43> <0x52> <0x3E> <0x20>
APPVERIFY	No	Yes	N/A	N/A

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

23.2.1 UNLOCKED

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

Transitioning from the LOCKED to UNLOCKED states erases all program memory. In the SBL, it also clears the challenge/response and application keys stored by the SBL.

The challenge and application keys can be erased by executing the Unlock command while in the UNLOCKED state and resetting the device. This eliminates the need to transition through the LOCKED state.

23.2.2 LOCKED

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

For the SBL, if the optional challenge key is activated, the bootloader starts in the CHALLENGE state. Successfully completing the challenge/response transitions to the previous PERMLOCKED or LOCKED state.

The application key should be configured before executing the *LOCK – Lock Device* command.

23.2.3 PERMLOCKED

The PERMLOCKED state disables access to program memory other than to verify it with a simple SHA-256 hash. The commands available in the PERMLOCKED state are shown in *Table 23-3*.

Table 23-3: PERMLOCK Command Summary

Command
H – Check Device
I – Get ID

For the SBL, if the optional challenge key is activated, the bootloader starts in the CHALLENGE state. Successfully completing the challenge/response transitions to the previous PERMLOCKED state.

The application key should be configured before executing the *PLOCK – Permanent Lock* command.

23.2.4 CHALLENGE (Secure Bootloader Only)

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

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

23.2.5 APPVERIFY (Secure Bootloader only)

APPVERIFY is an internal state that describes how the SBL verifies the integrity of program memory using a secret-key HMAC SHA-256 hash. It is not directly accessible by the SBL command set.

The SBL performs an APPVERIFY:

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

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

Follow this procedure to initialize the SBL for the APPVERIFY:

1. The host creates the Motorola SREC file as described in *Creating the Motorola SREC File*.
2. The host activates the SBL as described in the *Bootloader Activation* section.
3. Ensure the device is in the UNLOCKED state.
4. Execute the WL command with the length value calculated in step 1.
5. Execute the L command to load the Motorola SREC file.
6. Execute the V command to verify the Motorola SREC file was correctly loaded.
7. Execute the LK command to load the HMAC SHA-256 secret key.
8. Execute the VK command to verify the HMAC SHA-256 secret key was correctly loaded.
9. Execute the AK command. The device automatically verifies program memory on all subsequent resets and attempts to execute the Lock and Plock commands.

23.3 Creating the Motorola SREC File

The Motorola SREC file must include the program bytes and the MAC required for the APPVERIFY process. Address records must be 32-bit aligned and the length of each line must be a multiple of 4 bytes. Any unused memory locations within the program must be defined with a constant value.

The information here is presented for completeness. Analog Devices, Inc. can provide customers with a complete toolset for generating a Motorola SREC file that meets the SBL requirements.

Note: The length of the Motorola SREC file is not the same as code length used for the WL command, as explained below.

The procedure for the generating the SREC file is:

1. Define the 128-bit HMAC secret key.
2. Generate binary image.
3. Pad the binary image with constant value to next 32-byte boundary. The address of the last pad byte is the code length argument for the WL command.
4. Calculate the 32-byte HMAC SHA-256 using the secret key over the length of the padded binary image.
5. Append 32-byte HMAC SHA-256 to the binary image, after the last pad byte.
6. Generate Motorola SREC file.

23.4 Bootloader Activation

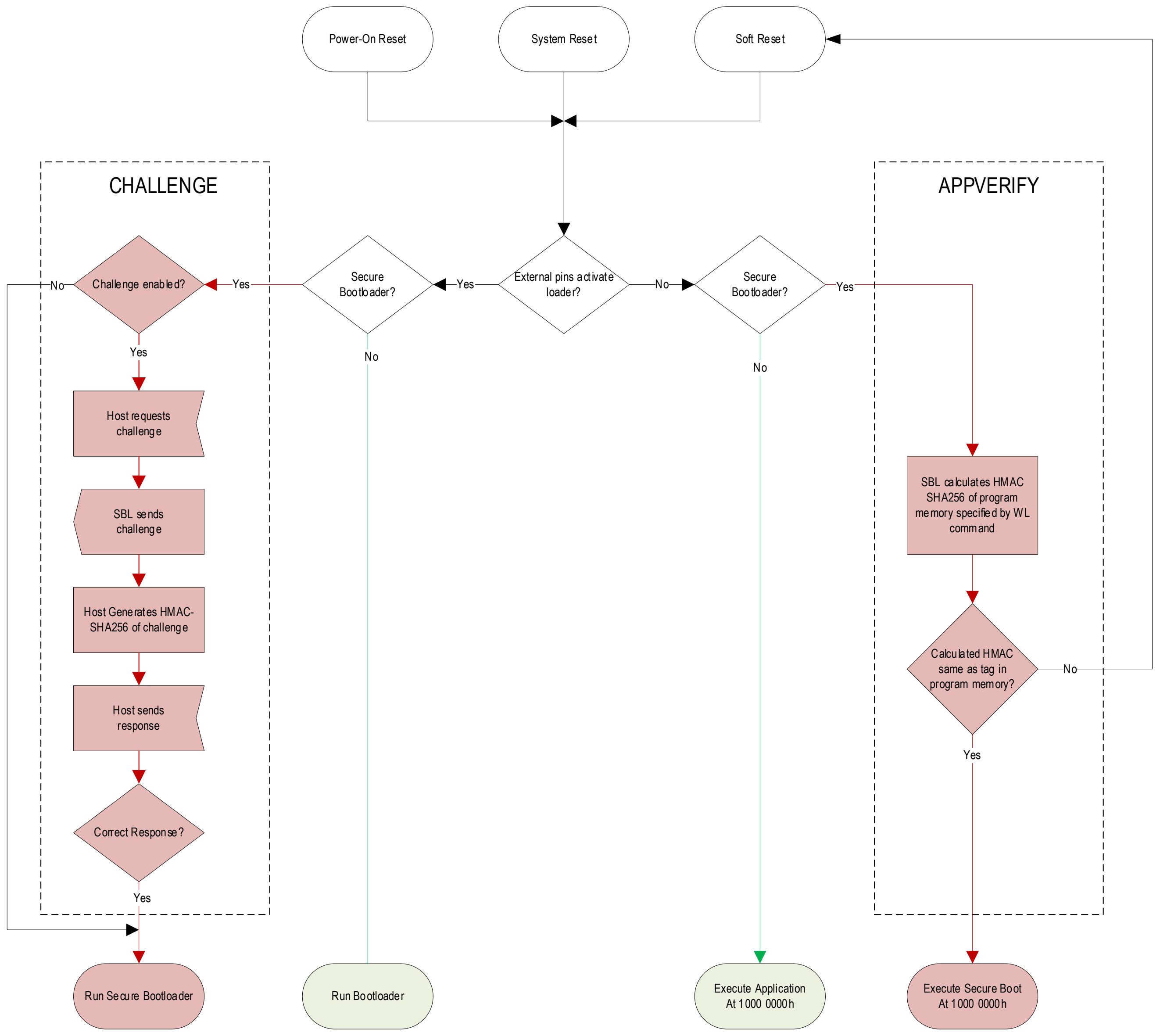
Perform the following sequence to activate the bootloader:

1. The host asserts the UART0 receive pin and SWDCLK pins low as shown in in *Table 23-1*.
2. The host asserts RSTN pin low.
3. The host deasserts the RSTN pin.
4. Bootloader samples the UART0 receive and SWDCLK pins. If they are both low, the hardware activates the bootloader.
5. Bootloader performs its system initialization and configures the bootloader for 115,200bps.
6. The bootloader outputs the status prompt on the UART0 transmit pin. The prompt is unique for each bootloader state as shown in *Table 23-2*.
7. The host releases the UART0 receive and SWDCLK pins once the host confirms the correct bootloader prompt.
8. The host begins bootloader session by sending commands on the UART0 receive pin.

23.5 Bootloader

The bootloader is invoked following a reset when the bootloader activation pin is asserted. The flow chart of the operation following a reset of the device is shown in *Figure 23-1*. Features exclusive to the SBL are highlighted in red.

Figure 23-1: MAX32675 Combined Bootloader Flow


Power-On Reset
System Reset
Soft Reset
CHALLENGE
APPVERIFY
Challenge enabled?
No
Yes
Secure Bootloader?
Yes
External pins activate loader?
No
Secure Bootloader?
Yes
Yes
No
No
Host requests challenge
SBL sends challenge
Host Generates HMAC-SHA256 of challenge
Host sends response
Correct Response?
Yes
SBL calculates HMAC SHA256 of program memory specified by WL command
Calculated HMAC same as tag in program memory?
No
Yes
Run Secure Bootloader
Run Bootloader
Execute Application At 1000 0000h
Execute Secure Boot At 1000 0000h

23.6 Secure Bootloader

The secure version of the bootloader provides additional features for secure and authenticated loading. These features are highlighted in *Figure 23-1*.

23.6.1 Secure Boot

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

Failure of the secure boot forces the device into a continual reset state.

23.6.2 Secure Challenge/Response Authentication

The optional secure challenge/response authentication provides an extra layer of security by requiring the host to authenticate itself using the mutual HMAC SHA-256 key before executing any bootloader commands. If the challenge key is activated, the device enters CHALLENGE mode following a reset if the external bootloader pins are active. The bootloader displays the CHALLENGE mode prompt shown in *Table 23-2*.

Only two commands are available in the CHALLENGE state.

Table 23-4: CHALLENGE Command Summary

Command
GC – Get Challenge
SR – Send Response

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

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

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

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

Follow this procedure to successfully perform the challenge/response:

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

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

Following a successful response, the bootloader returns the prompt appropriate to the last state of the SBL.

23.7 Command Protocol

The bootloader presents a mode-specific prompt based on the current state of the loader as shown as in *Table 23-2*. The general format of commands is the ASCII character(s) of the command, followed by a <CR><LF> which is hexadecimal <0x0D><0x0A>. Commands with arguments always have a space (0x20) between the command mnemonic and the argument.

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

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

Always see the command description for the required format of the command.

23.8 General Commands

Table 23-5: MAX32675 General Command Summary

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

23.8.1 General Command Details

Table 23-6: L - Load

L - Load	Load Motorola SREC File into Program Memory
Description	Load a Motorola SREC formatted file into flash program memory. See *Creating the Motorola SREC File* for the details of the format required for the SBL. After typing the L command, the bootloader responds with "Ready to load SREC", then transmit the file. The end of the file is detected automatically, so there is no need to send <0x0D><0x0A> at the end. The length reported by the success response is the padded image plus the 32-bytes of the HMAC; this is different than the length used for the WL command.
Modes	U

Command	`L<0x0D><0x0A>`
	`        Ready to load SREC<0x0D><0x0A>`
	`[Motorola SREC File]`

Response: Success	`        Load success, image loaded with the following parameters:<0x0D><0x0A>`
	`        Base address: 0xnnnnnnnn<0x0D><0x0A>`
	`        Length: 0xnnnnnnnn<0x0D><0x0A>`

Response: Failure	`        Load failed.<0x0D><0x0A>`

Table 23-7: P – Page Erase

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

Response: Failure	`Bad page address input<0x0D><0x0A>`
	or
	`Erase failed<0x0D><0x0A>`
	or
	`Invalid Page Address: 0xnnnnnnnn<0x0D><0x0A>`

Table 23-8: V – Verify

V – Verify	**Verify Flash Program Memory Against Motorola SREC File**
Description	Verifies contents of flash program memory against a Motorola SREC file.
Modes	U

Command	`V<0x0D><0x0A>`
	`        Ready to verify SREC<0x0D><0x0A>`
	`[Motorola SREC File]`

Response: Success	`        Verify success, image verified with the following parameters: <0x0D><0x0A>`
	`        Base address: 0xnnnnnnnn<0x0D><0x0A>`
	`        Length: 0xnnnnnnnn<0x0D><0x0A>`

Response: Failure	`        Verify failed.<0x0D><0x0A>`

Table 23-9: LOCK – Lock Device

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

Table 23-10: PLOCK – Permanent Lock

PLOCK – Permanent Lock	**Permanently Lock Device**
Description	Permanently locks the device if the argument matches the device ID. The effects of the Plock command do not take effect until the next time the device is reset. The bootloader continues to display the LOCKED or UNLOCKED state prompt but the *S – Status* command shows the LOCKED or UNLOCKED state is active. The Lock command should be followed by the Q command (which generates a device reset) for the Lock command to take effect. The SBL performs an APPVERIFY check before executing the PLock command. Failure of the PLock command means that the APPVERIFY check failed.
Modes	U/L
Command	`PLOCK <USN><0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`
Response: Failure	`Failed<0x0D><0x0A>`

Table 23-11: UNLOCK – Unlock Device

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

Table 23-12: H – Check Device

H – Check Device	**Perform SHA-256 Hash Over Memory Range**
Description	Performs a simple SHA-256 (not HMAC SHA-256) hash of bytes starting at 32-bit address 0xnnnnnnnn to 0xmmmmmmmm. Minimum hash input size is 512 bytes. Function returns 32-byte hash value.
Modes	U/L/P
Command	`H 0xnnnnnnnn 0xmmmmmmmm<0x0D><0x0A>`
Response: Success	`yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>`
Response: Failure	`<0x0D><0x0A>`

Table 23-13: I – Get ID

I – Get ID	**Read Universal Serial Number**
Description	Returns the 13-byte unique USN of the device.
Modes	U/L/P

Command	`I<0x0D><0x0A>`
	`USN: nnnnnnnnnnnnnnnnnnnnnnnnnn<0x0D><0x0A>`

Response: Success	None
Response: Failure	None

Table 23-14: S – Status

S – Status	**Read Device Status**
Description	Returns the state of the loader and the application key and challenge key features. This changes during the same session when the command is executed, unlike the prompt which only changes after reset. The Lock <response> is: "`Inactive`" if the device is in the unlocked state. "`Active`" if the device is in the locked or permanent lock state. The PLock <response> is: "`Inactive`" if the device is in the unlocked or locked state. "`Active`" if the device is in the permanent lock state. In addition, the SBL displays: The Application Length <response> is: "`Not Set`" if the Write Code Length command has not previously loaded a non-zero value. "`0x`nnnnnnnn" which is the previously entered value using the Write Code Length command. The Application Key <response> is: "`None Inactive`" if no application key has been loaded using the LK command. "`Loaded Inactive`" if the application key has been loaded but the application key feature has not been activated by the AK command. "`Loaded Active`" If the application key has been loaded and the application key feature has been activated. The Challenge Key <response> is: "`None Inactive`" if no challenge key has been loaded using the LK command. "`Loaded Inactive`" if the challenge key has been loaded but the challenge key feature has not been activated by the AK command. "`Loaded Active`" if the challenge key has been loaded and the challenge key feature has been activated.
Modes	U

Command	`S<0x0D><0x0A>`
	       `Status<0x0D><0x0A>`
	         `Lock: <response><0x0D><0x0A>`
	         `PLock: <response><0x0D><0x0A>`
	         `Application Length: <response><0x0D><0x0A>`
	         `Application Key: <response><0x0D><0x0A>`
	         `Challenge Key: <response><0x0D><0x0A>`

Response: Success	None.
Response: Failure	None.

Table 23-15: Q – Quit

Q – Quit	Quit Bootloader Session
Description	Terminates the bootloader session and forces a reset of the device.
Modes	U/L/P
Command	`Q<0x0D><0x0A>`
Response: Success	None
Response: Failure	None

23.9 Secure Commands

Table 23-16: MAX32675 Secure Command Summary

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

23.9.1 Secure Command Details

Table 23-17: LK – Load Application Key

LK – Load Application Key	**Load Application HMAC SHA2-56 Key**
Description	Write 128-bit HMAC secret key to non-volatile memory.
Modes	U
Command	`LK yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Bad key input<0x0D><0x0A>`
	 or
	 `Key already loaded<0x0D><0x0A>`

Table 23-18: LK – Load Challenge Key

LC – Load Challenge Key	**Load Challenge Key**
Description	Write 128-bit challenge key to non-volatile memory.
Modes	U
Command	`LC yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Bad key input<0x0D><0x0A>`
	`or`
	`Key already loaded<0x0D><0x0A>`

Table 23-19: VK – Verify Application Key

VK – Verify Application Key	**VK – Verify Application Key**
Description	Verify the application key against a value provided by the host.
Modes	U
Command	`VK yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Bad key input<0x0D><0x0A>`
	or
	`Error, no key loaded<0x0D><0x0A>`
	or
	`Key Mismatch<0x0D><0x0A>`

Table 23-20: VC – Verify Challenge Key

VC – Verify Challenge Key	**VC – Verify Challenge Key**
Description	Verify the challenge key against a value provided by the host.
Modes	U
Command	`VC yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Bad key input<0x0D><0x0A>`
	`or`
	`Error, no key loaded<0x0D><0x0A>`
	`or`
	`Key Mismatch<0x0D><0x0A>`

Table 23-21: AK – Activate Application Key

AK – Activate Application Key	**Activate Application Key**
Description	Activate application key. All application software loads must be encrypted with the application key. The UNLOCK command deactivates the application key.
Modes	U
Command	`AK<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Key activate failed<0x0D><0x0A>`
	 or
	 `Error, no key loaded<0x0D><0x0A>`

Table 23-22: AC – Activate Challenge Key

AC – Activate Challenge Mode	**Activate Challenge Mode**
Description	Activate CHALLENGE mode. All subsequent bootloader sessions in LOCKED or PERMLOCKED states start in CHALLENGE mode. The "Key activate failed" response indicates the device has already activated the challenge/response feature. The host should use the SBL to re-enter the UNLOCKED state to deactivate the challenge mode and reenter the keys and activate the challenge mode again.
Modes	U
Command	`AC<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Key activate failed<0x0D><0x0A>`
	 or
	 `Error, no key loaded<0x0D><0x0A>`

Table 23-23: WL – Write Code Length

WL – Write Code Length	**Write Code Length**
Description	Write the length of the application software in bytes as calculated in *Creating the Motorola SREC File*. The “Write length failed” response indicates the WL command has already been performed. The host should use the SBL to re-enter the UNLOCKED state to clear the WL value and repeat the command.
Modes	U
Command	`WL 0xnnnnnnnn<0x0D><0x0A>`
Response: Success	`Length set to: 0xnnnnnnnn<0x0D><0x0A>`

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

23.10 Challenge/Response Commands

Table 23-24: MAX32675 Challenge/Response Command Summary

Command
GC – Get Challenge
SR – Send Response

23.10.1 Challenge/Response Command Details

Table 23-25: GC – Get Challenge

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

Table 23-26: SR – Send Response

SR – Send Response	Send Response
Description	Host calculates HMAC SHA-256 on the 16-byte challenge and sends the 32-byte hexadecimal ASCII response to SBL. The response is sent MSB first.
Modes	L/P
Command	`SR 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Bad response input<0x0D><0x0A>`
	or
	`Verification failed<0x0D><0x0A>`
	or
	`Error, request challenge<0x0D><0x0A>`

24. Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	01/2022	Initial release	–