MAX32662 User Guide

UG7640; Rev 1; 02/2023

Abstract: This user guide provides application developers information on how to use the memory and peripherals of the MAX32662 microcontroller. It covers detailed information for all the registers and fields in the device. It also provides guidance to manage all the peripherals, clocks, powers, and start-up for the device.

MAX32662 User Guide

Table of Contents

Table of Figures

Table of Tables

Table of Equations

1. Introduction

Refer to the data sheet for ordering information, mechanical and electrical characteristics.

1.1 Related Documentation

The MAX32662 data sheet and errata are available from the Analog Devices website, http://www.maximintegrated.com/MAX32662.

1.2 Document Conventions

1.2.1 Number Notations

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

1.2.2 Register and Field Access Definitions

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

Table 1-1: Field Access Definitions

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

Access Type	Definition
R/W1C	**Read/Write 1 to Clear** Writing 1 to this field clears this field to 0. Writing 0 to the field does not affect device operation.
R/W0S	**Read/Write 0 to Set** Writing 0 to this field sets this field to 1. Writing 1 to the field does not affect device operation.

1.2.3 Register Lists

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

Table 1-2: Example Registers

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

1.2.4 Register Detail Tables

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

Table 1-3: Example Name 0 Register

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

2. Overview

In the DARWIN family, the MAX32662 is an ultra-low-power, cost-effective, highly integrated 32-bit microcontroller designed for battery-powered devices and wireless sensors. It combines a flexible and versatile power management unit with the powerful Arm® Cortex®-M4 processor with floating point unit (FPU) in the industry's smallest form factor.

The MAX32662 enables designs with complex sensor processing without compromising battery life. It also offers legacy designs an easy and cost optimal upgrade path from 8- or 16-bit microcontrollers.

Integrating 256KB of flash memory and 64KB of RAM, the MAX32662 easily accommodates sensor code and algorithms.

The device features five powerful and flexible power modes. It can operate from a single-supply battery or a dual-supply typically provided by a power management integrated circuit (PMIC). The I^2C ports support standard, fast, fast-plus, and high-speed modes, operating up to 3,400kbps. The SPI ports can run up to 50MHz in both controller and peripheral mode. The 12-bit successive approximation analog-to-digital converter (SAR ADC) provides an integrated reference generator and a single-ended input multiplexer. The integrated CAN 2.0B interface is compliant with Bosch CAN 2.0B specification (2.0B Active). Three general-purpose 32-bit timers, one low-power 32-bit timer, one windowed watchdog timer, and a real-time clock (RTC) are also provided. An I^2S interface provides digital audio streaming to a codec.

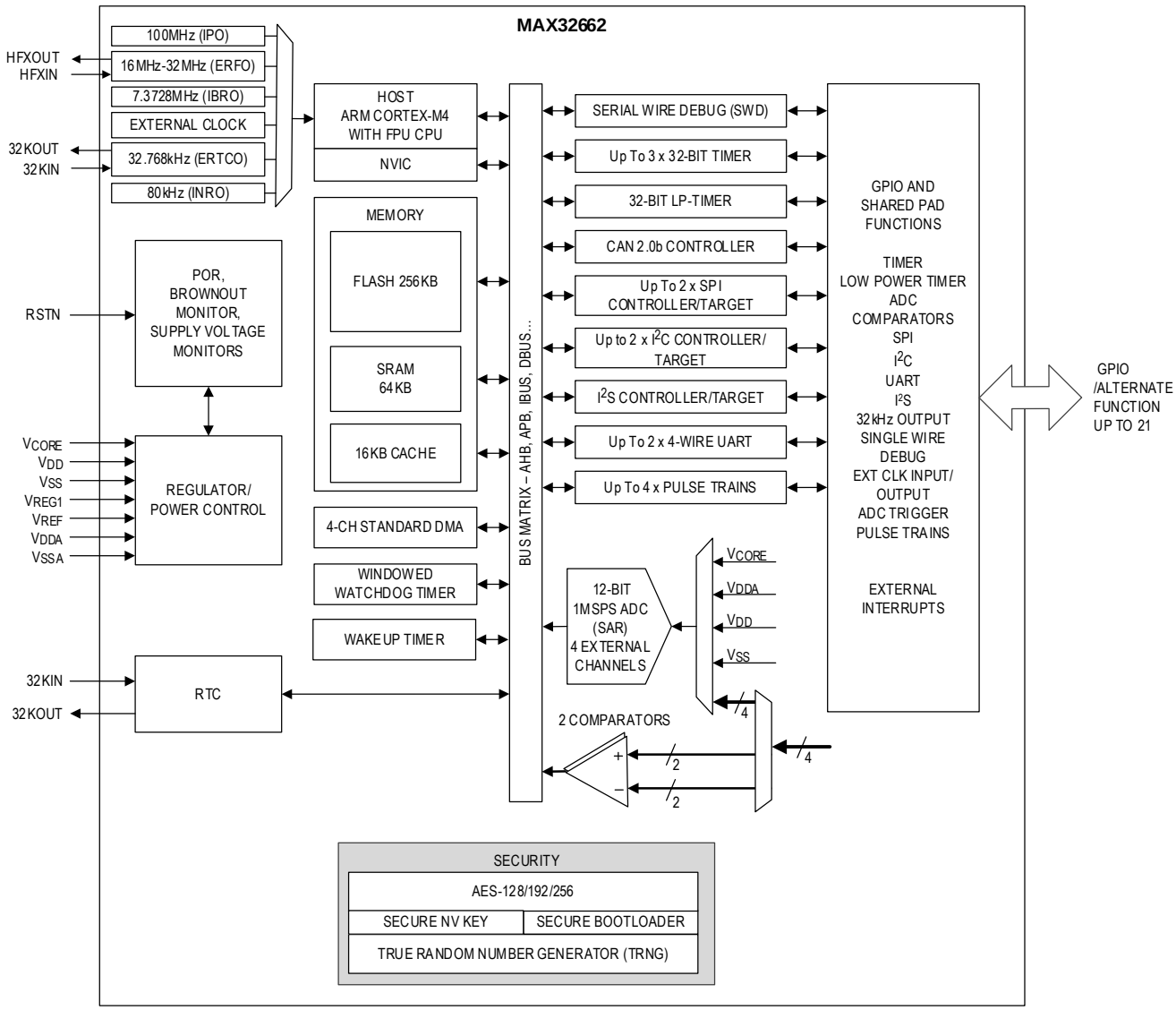
An optional bootloader through I^2C, UART, or SPI is available.

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

The high-level block diagram for the MAX32662 is shown in *Figure 2-1*.

2.1 Block Diagram

Figure 2-1: MAX32662 Block Diagram



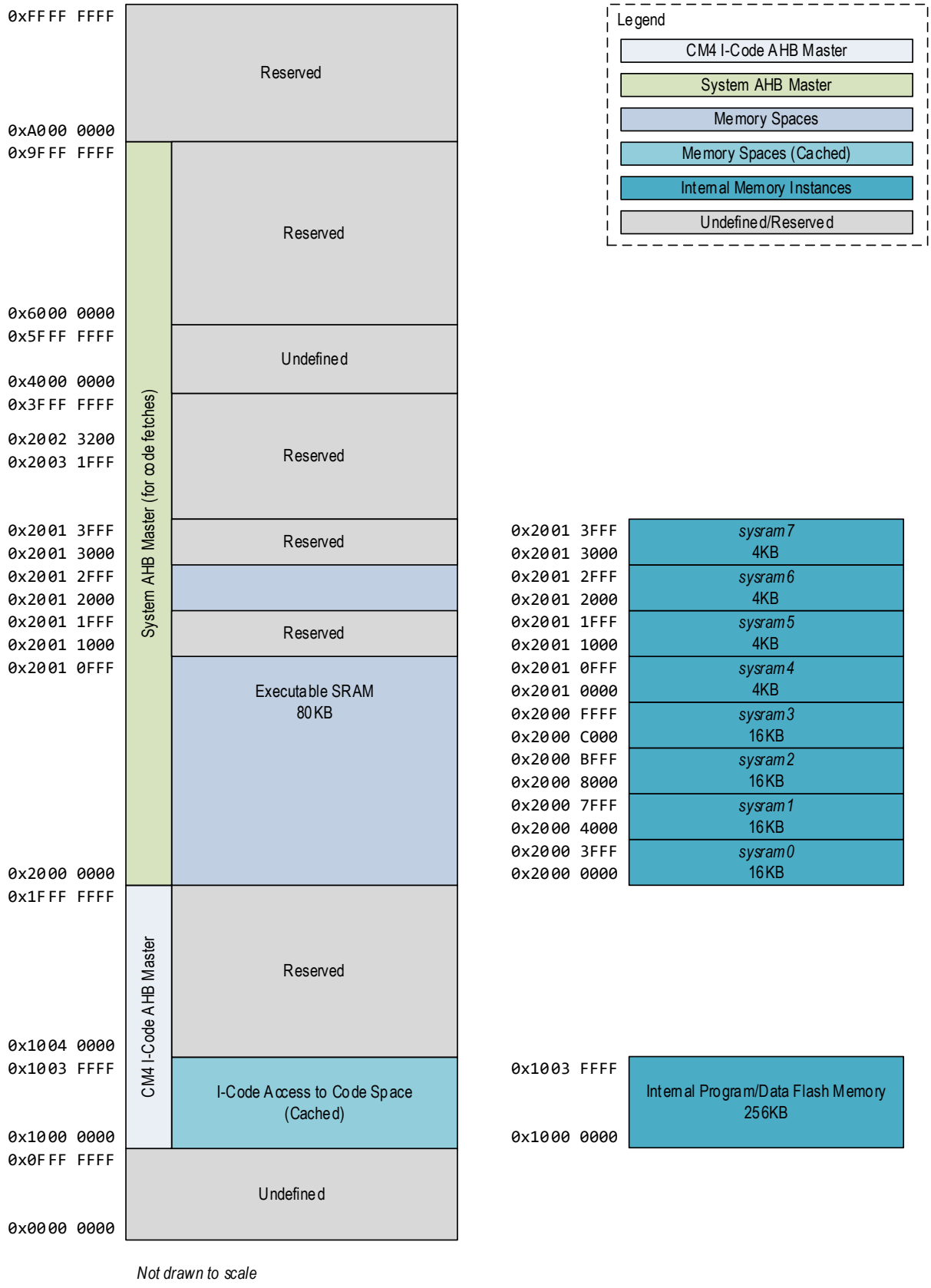
3. Memory, Register Mapping, and Access

3.1 Memory, Register Mapping, and Access Overview

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

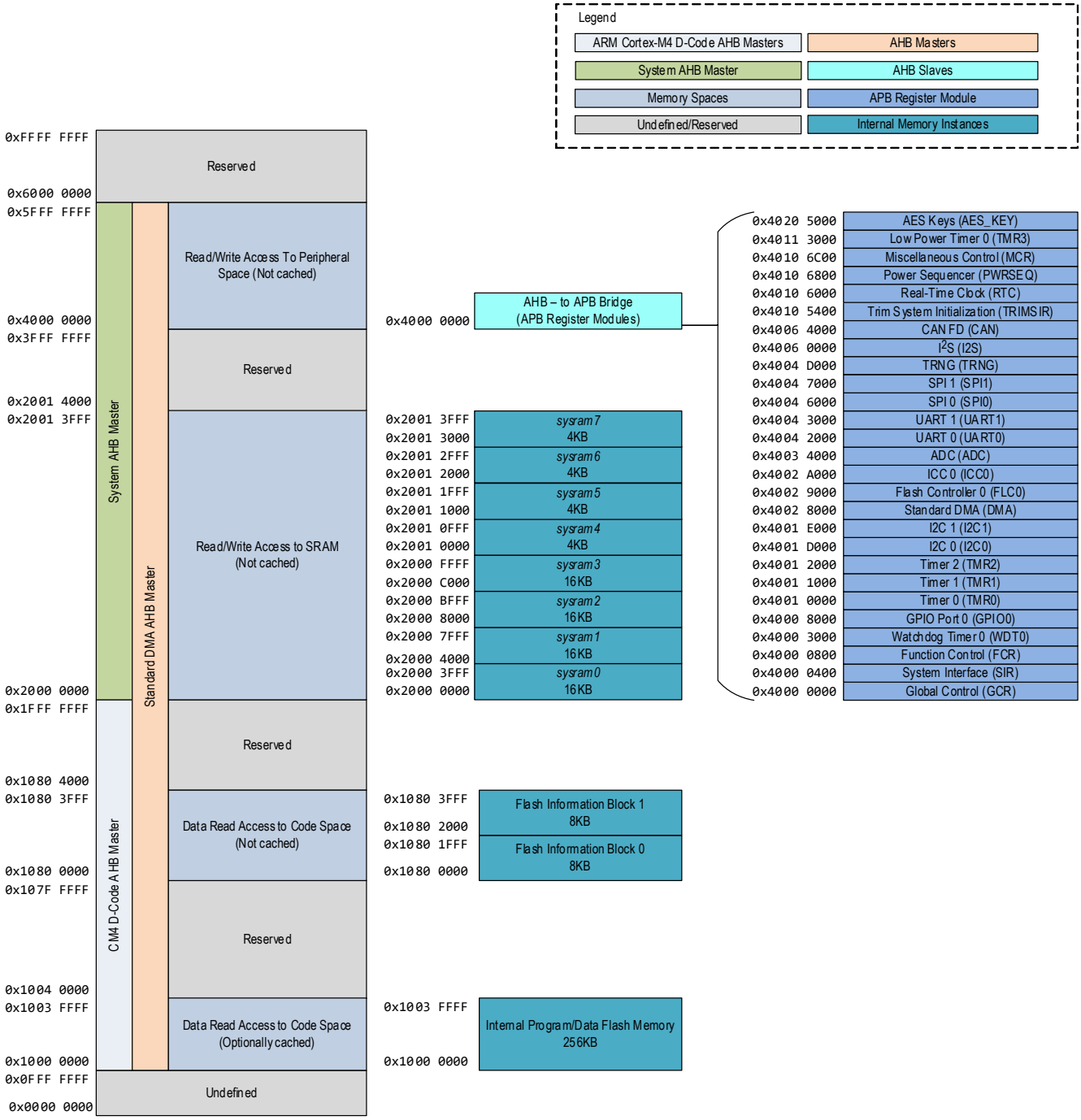
However, it is important to note that the architectural definition does not require the entire 4GB memory range to be populated with addressable memory instances.

Figure 3-1: Code Memory Mapping



Not drawn to scale

Figure 3-2: Data Memory Mapping



3.2 Standard Memory Regions

Several standard memory regions are defined for the Arm Cortex-M4 architecture. The use of many of these is optional for the system integrator. At a minimum, the MAX32662 must contain some code and data memory for the software and variable/stack use, and certain components part of the instantiated core.

3.2.1 Code Space

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

The MAX32662 code memory mapping is illustrated in *Figure 3-1*. The code space memory area contains the main internal flash memory, which holds most of the instruction code executed on the device. The internal flash memory is mapped into both code and data space from 0x1000 0000 to 0x1003 FFFF. It is partitioned as one 256KB block of usable flash.

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 MAX32662 also contains the mapping for the flash information blocks, from 0x1080 0000 to 0x1080 3FFF. Flash information block 0 is read only and mapped from address 0x1080 0000 to 0x1080 1FFF. Flash information block 0 contains the device's unique serial number (USN) as well as trim settings for the device. Flash information block 1 is mapped from address 0x1080 2000 to 0x1080 3FFF.

3.2.1.1 Flash Information Block 0

Flash information block 0 is a read-only page used to store the device's USN, trim settings (option configuration and analog trim), and other nonvolatile device-specific information. The USN is a 104-bit field stored from address 0x1080 0000 through 0x1080 0017. See *Figure 3-3* for the USN mapping. Read the USN using FLC0 and performing a read of addresses 0x1080 0000 through address 0x1080 0017.

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

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

3.2.2 Internal Cache Memory

The MAX32662 includes a dedicated unified internal cache controller (ICC) with 16,384 bytes of cache memory for the CM4 core (ICC0).

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

3.2.3 SRAM Space

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

The MAX32662 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 80KB. Its address range is mapped from 0x2000 0000 to 0x2001 3FFF.

The entirety of the SRAM memory space on the MAX32662 is contained within the dedicated Arm Cortex-M4 SRAM bit-banding region from 0x2000 0000 to 0x200F FFFF (1MB maximum for bit-banding). This means that the CPU can access the entire SRAM using standard byte/word/doubleword access or bit-banding operations. The bit-banding mechanism allows any single bit of any given SRAM byte address location to be set, cleared, or read individually by reading from or writing to a corresponding double word (32-bit wide) location in the bit-banding alias area.

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

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

The SRAM area on the MAX32662 can be used to contain executable code. Code stored in the SRAM is accessed directly for execution (using the system bus) and is not cached. The SRAM is also where the CM4 stack must be located, as it is the only general-purpose SRAM memory on the device. A valid stack location inside the SRAM must be set by the system exception table (which is, by default, stored at the beginning of the internal flash memory).

The MAX32662 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
sysram0	4K	0x2000 0000	0x2000 3FFF
sysram1	4K	0x2000 4000	0x2000 7FFF
sysram2	4K	0x2000 8000	0x2000 BFFF
sysram3	4K	0x2000 C000	0x2000 FFFF
sysram4	1K	0x2001 0000	0x2001 0FFF
sysram5	1K	0x2001 1000	0x2001 1FFF
sysram6	1K	0x2001 2000	0x2001 2FFF
sysram7	1K	0x2001 3000	0x2001 3FFF

3.2.4 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 MAX32662, all device-specific module registers are mapped to this memory area and any local memory buffers or FIFOs required by modules.

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

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

On the MAX32662, access to the region containing most peripheral registers (0x4000 0000 to 0x400F FFFF) goes from the AHB bus through an AHB-to-APB bridge enabling peripheral modules to operate on the lower power APB bus matrix. The AHB-to-APB bridge also ensures that peripherals with slower response times do not tie up bandwidth on the AHB bus, which must have a faster response time since it handles main application instruction and data fetching.

3.2.5 System Area (Private Peripheral Bus)

The system area (private peripheral bus) memory space contains register areas for functions 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 that implement AHB memory masters, such as the direct memory access (DMA) interface.

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

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

3.2.6 System Area (Vendor Defined)

The system area (vendor defined) memory space is reserved for vendor (system integrator) specific functions that are not handled by another memory area. It is defined from byte address range 0xE010 0000 to 0xFFFF FFFF. The MAX32662 does not implement this memory region.

3.3 AHB Interfaces

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

3.3.1 Core AHB Interfaces

3.3.1.1 I-Code

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

3.3.1.2 D-Code

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

3.3.1.3 System

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

3.3.2 AHB Masters

3.3.2.1 Standard DMA

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

3.4 Peripheral Register Map

3.4.1 APB Peripheral Base Address Map

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

Table 3-2: APB Peripheral Base Address Map

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
Global Control	GCR_	0x4000 0000	0x4000 03FF
System Interface	SIR_	0x4000 0400	0x4000 07FF
Function Control	FCR_	0x4000 0800	0x4000 0BFF
Watchdog Timer	WDT_	0x4000 3000	0x4000 33FF
GPIO Port 0	GPIO0_	0x4000 8000	0x4000 8FFF
Timer 0	TMR0_	0x4001 0000	0x4001 0FFF
Timer 1	TMR1_	0x4001 1000	0x4001 1FFF
Timer 2	TMR2_	0x4001 2000	0x4001 2FFF
I^2C 0	I2C0_	0x4001 D000	0x4001 DFFF
I^2C 1	I2C1_	0x4001 E000	0x4001 EFFF
Standard DMA	DMA	0x4002 8000	0x4002 8FFF
Flash Controller	FLC_	0x4002 9000	0x4002 93FF
Internal-Cache Controller	ICC_	0x4002 A000	0x4002 A3FF
ADC	ADC_	0x4003 4000	0x4003 4FFF
Pulse Train	PT_	0x4003 C000	0x4003 CFFF
UART 0	UART0_	0x4004 2000	0x4004 2FFF
UART 1	UART1_	0x4004 3000	0x4004 3FFF
SPI0	SPI0_	0x4004 6000	0x4004 6FFF
SPI1	SPI1_	0x4004 7000	0x4004 7FFF
TRNG	TRNG_	0x4004 D000	0x4004 DFFF
I^2S	I2S_	0x4006 0000	0x4006 0FFF

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
CAN 2.0b	CAN_	0x4006 4000	0x4006 4FFF
Trim System Initialization	TRIMSIR_	0x4010 5400	0x4010 5400
Real-Time Clock	RTC_	0x4010 6000	0x4010 63FF
Power Sequencer	PWRSEQ_	0x4010 6800	0x4010 6BFF
Miscellaneous Control	MCR_	0x4010 6C00	0x4010 6FFF
Timer 3 (Low Power Timer 0)	TMR3_	0x4011 3000	0x4011 4FFF
AES Keys	AES_KEY_	0x4020 5000	0x4020 5FFF
AES	AES_	0x4020 7400	0x4020 7FFF

4. System, Power, Clocks, Reset

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

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

- 100MHz Internal Primary Oscillator (IPO)
- 7.3728MHz Internal Baud Rate Oscillator (IBRO)
- 80kHz Internal Nanoring Oscillator (INRO)
- 32.768kHz External RTC Crystal Oscillator (ERTCO)
  - Clock Source for the RTC
- 16MHz to 32MHz External RF Crystal Oscillator (ERFO)
- HF_EXT_CLK P0.6 AF4

4.1 Selecting the Core Operating Voltage Range

The MAX32662 supports three selections for the core operating voltage range (OVR). In a single-supply operation, changing the OVR sets the output of the internal low dropout (LDO) regulator to the voltage shown in *Table 4-1*. In a dual-supply design, setting the OVR allows an external PMIC to provide the required V_{CORE} voltage dynamically. Changing the OVR also reduces the output frequency of the IPO, further reducing power consumption.

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

Changes to the OVR affect the internal flash memory access time, and the software must set the flash wait states for each OVR setting as outlined in the section *Flash Wait States*. Changing the core operating voltage reduces the output frequency of the 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_LPCTRL.ovr	FLC_CTRL.lve	V_{CORE} Typical (V)	SYS_OSC					
			f_{IPO} (MHz)	f_{IBRO} (MHz)	f_{ERFO} (MHz)	$f_{HF\_EXT\_CLK}$ (MHz)	f_{INRO} (kHz)	f_{ERTCO} (kHz)
0	1	0.9	12	7.3728	20 (Max)	15 (Max)	80	32.768
1	1	1.0	50	7.3728	25 (Max)	30 (Max)	80	32.768
2	0	1.1	100	7.3728	32 (Max)	50 (Max)	80	32.768

4.1.1 Setting the Operating Voltage Range

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

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

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

On each subsequent non-POR reset event:

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

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

4.1.2 Flash Wait States

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

Important: *Flash reads can fail and result in unknown instruction execution if the GCR_MEMCTRL.fws setting is lower than the minimum required for a given PWRSEQ_LPCTRL.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_LPCTRL.ovr	FLC_CTRL.lve					
0	1	0.9	12	0	12	0
				1	6	0
1	1	1.0	50	0	50	1
				1	25	0
2	0	1.1	100	0	100	2
				1	50	1
				2	25	0

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

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

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

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

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

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

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

Core Operating Voltage Range Setting		Core Voltage Range V_{CORE} (V)	f_{ERTCO} (kHz)	System Clock Prescaler GCR_CLKCTRL.sysclk_div	System Clock f_{SYS_CLK} (kHz)	Minimum Flash Wait State Setting GCR_MEMCTRL.fws
PWRSEQ_LPCTRL.ovr	FLC_CTRL.lve					
0	1	0.9	32.768	0	32.768	0
				1	16.384	0
1	1	1.0	32.768	0	32.768	0
				1	16.384	0
2	0	1.1	32.768	0	32.768	0
				1	16.384	0
				2	10.923	0

4.2 Oscillator Sources

4.2.1 100MHz Internal Primary Oscillator (IPO)

The MAX32662 includes a 100MHz internal high-speed oscillator, referred to in this document as the IPO. The IPO is the fastest oscillator and draws the most power.

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

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

4.2.1.1 IPO Calibration

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

1. Enable the ERTCO by setting *MCR_CLKCTRL.ertco_en* to 1.
2. Wait until *GCR_CLKCTRL.ertco_rdy* reads 1. The ERTCO is now operating.
3. Set the *FCR_AUTOCAL2.acdiv* field to 3,051. See the *FCR_AUTOCAL2.acdiv* field for additional information.
4. Set the *FCR_AUTOCAL1.inittrim* field to 0x40.
5. Set the *FCR_AUTOCAL0.acrun*, *FCR_AUTOCAL0.acen*, and *FCR_AUTOCAL0.load* fields to 1 by performing a bitwise OR of the *FCR_AUTOCAL0* register with 0x7.
6. Wait 10ms for the trim to complete.
   a. The calculated trim is loaded to the *FCR_AUTOCAL0.trim* field and is used by the hardware as long as *FCR_AUTOCAL0.acen* is set to 1.
7. Set the *FCR_AUTOCAL0.acrun* field to 0 to stop the calibration.

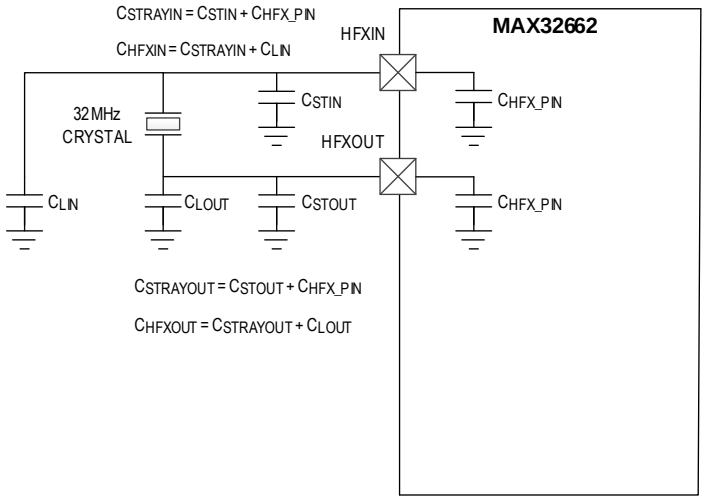
4.2.2 16MHz to 32MHz External RF Oscillator

This oscillator can be selected as SYS_OSC. It is important to use the correct capacitor values on the PCB when connecting the crystal. *Figure 4-1* 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*.
2. Wait until *GCR_CLKCTRL.erfo_rdy* is set. The ERFO is now operating.
3. Set *GCR_CLKCTRL.sysclk_sel* to 2, selecting the ERFO as the SYS_OSC.
4. Wait until *GCR_CLKCTRL.sysclk_rdy* is set. The ERFO is now operating as the SYS_OSC.

Figure 4-1: 32MHz ERFO Crystal Capacitor Determination



Equation 4-1: Determining Load Capacitance for ERFO

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

Calculate the values of C_{LOUT} and C_{LIN}, referring to *Figure 4-1*, using the following steps:

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

4.2.2.1 ERFO Kick Start

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

4.2.2.2 Programmatic Determination of the Optimum Kick Start Settings

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

1. Disable the ERFO by setting *GCR_CLKCTRL.erfo_en* to 0.
2. Disable the kick start circuit by setting the following fields to 0:
   a. *FCR_ERFOKS.ksclksel*
   b. *FCR_ERFOKS.kserfodriver*
   c. *FCR_ERFOKS.kserfo_cnt*
3. Select the IPO to kick start the ERFO by setting *FCR_ERFOKS.ksclksel* field to 3.
4. Enable the IPO by setting *GCR_CLKCTRL.ipo_en* to 1
   a. Read the *GCR_CLKCTRL.ipo_rdy* field until it reads 1.
5. Configure a timer for counter mode using the oscillator selected in step 3 as the timer clock. See *Timers (TMR/LPTMR)* for details of timer configuration and modes.
6. The number of kick start pulses supported, range from 1 to 127. The following steps should be performed by incrementing the kick start count by 1 on each iteration while measuring the amount of time it takes to start up the ERFO. Once the fastest start-up time is determined, save the settings and use them any time the ERFO is started by the software.
   a. Set the kick start pulse count, *FCR_ERFOKS.kserfo_cnt*, to the number of kick start pulses to test. This is the iteration number from 1 to 127.
   b. Enable the kick start circuit by setting *FCR_ERFOKS.kserfo_en* to 1.
   b. Start the configured timer.
   c. Enable the ERFO by setting *GCR_CLKCTRL.erfo_en* to 1.
   d. Read the *GCR_CLKCTRL.erfo_rdy* field until it reads 1.
   e. Stop the timer and determine the elapsed time.
   f. Disable the kick start circuit by setting *FCR_ERFOKS.kserfo_en* to 0.
   g. Repeat steps *a* to *f* until all supported kick start pulses are tested and determine the optimum kick start pulses for the fastest ERFO start-up time.

4.2.2.3 Using the ERFO Kick Start

To use the ERFO kick start circuit, perform the following steps when enabling the ERFO:

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

4.2.3 7.3728MHz Internal Baud Rate Oscillator (IBRO)

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

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

Perform the following steps to enable the IBRO and set it as the system clock:

1. Enable the IBRO by setting *GCR_CLKCTRL.ibro_en* to 1.
2. Wait until the *GCR_CLKCTRL.ibro_rdy* field reads 1.
   a. The IBRO is now operating.
3. If setting the IBRO as the system oscillator, set *GCR_CLKCTRL.sysclk_sel* to 5.
4. Wait until *GCR_CLKCTRL.sysclk_sel* reads 1.
   a. The IBRO is now operating as the SYS_OSC.

4.2.4 32.768kHz External Real-Time Clock Oscillator (ERTCO)

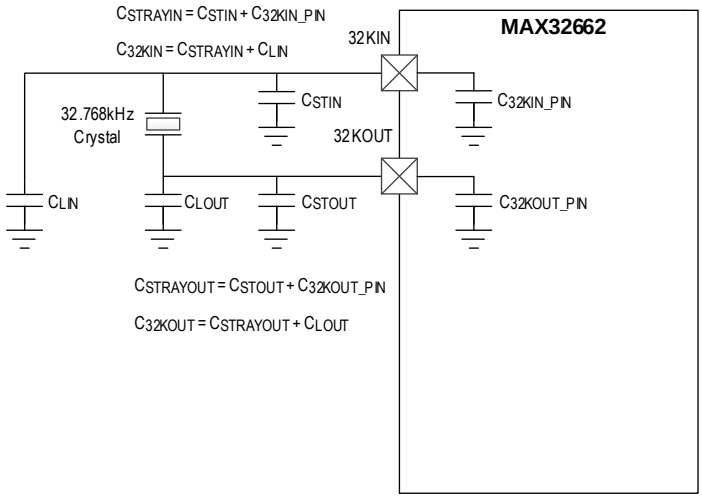
The ERTCO is a very low-power external oscillator that can be selected as SYS_OSC. This oscillator can optionally use a 32.768kHz input clock instead of an external crystal. The ERTCO is available as an output on GPIO as an alternate function (32KCAL (P0.22 AF3)).

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

A POR disables the ERTCO. All other forms of reset do not change the ERTCO enable bit. See *Figure 4-4* and *Figure 4-5* for details on reset sources and the effect on the ERTCO.

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

Figure 4-2: Example 32.768kHz Crystal Capacitor Determination



Equation 4-2: Determining Load Capacitance for ERTCO

$$C_L = C_{32KIN}\ X\ C_{32KOUT}/(C_{32KIN} + C_{32KOUT})$$

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

1. The crystal load, C_L, as specified in the device data sheet electrical characteristics table, is required to be 6pF. Therefore, the total capacitance seen by the crystal must equal C_L.
2. $C_{LIN} = C_{LOUT}$
3. The device pin capacitance of the 32KOUT and 32KIN pins is 6pF each.
4. Assume the circuit board stray capacitance represented in the diagram by C_{STIN} and C_{STOUT} are 0.5pf each.
5. Solve for C_{LOUT}:
   $C_{LOUT} = 5.5pF = C_{LIN}$
6. Choose 6pF as the closest standard value for $C_{LOUT} = C_{LIN}$.

Perform the following steps to enable the ERTCO and set it as the SYS_OSC:

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

4.2.5 80kHz Internal Nano-Ring Oscillator (INRO)

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

4.3 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 4-3*.

Equation 4-3: System Clock Scaling

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

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

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

Equation 4-4: AHB Clock

$$HCLK = SYS\_CLK$$

Equation 4-5: APB Clock

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

Equation 4-6: AoD Clock

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

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

The RTC uses the ERTCO for its clock source.

All oscillators are reset to their POR reset default state during a POR, system reset, or watchdog reset. Oscillator settings are not reset during a soft reset or peripheral reset. *Table 4-8* shows each oscillator's enabled state for each type of reset source in the MAX32662. *Table 4-9* details each reset source's effect on the system clock selection and the system clock prescaler settings.

CAUTION: When switching the SYS_OSC or touching the SYS_OSC prescaler (*GCR_CLKCTRL.sysclk_div*), any device peripherals using SYS_CLK, APB clock, or AHB clock become unstable. The software should understand that all peripherals should be disabled before switching SYS_OSC or touching the SYS_OSC prescaler.

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

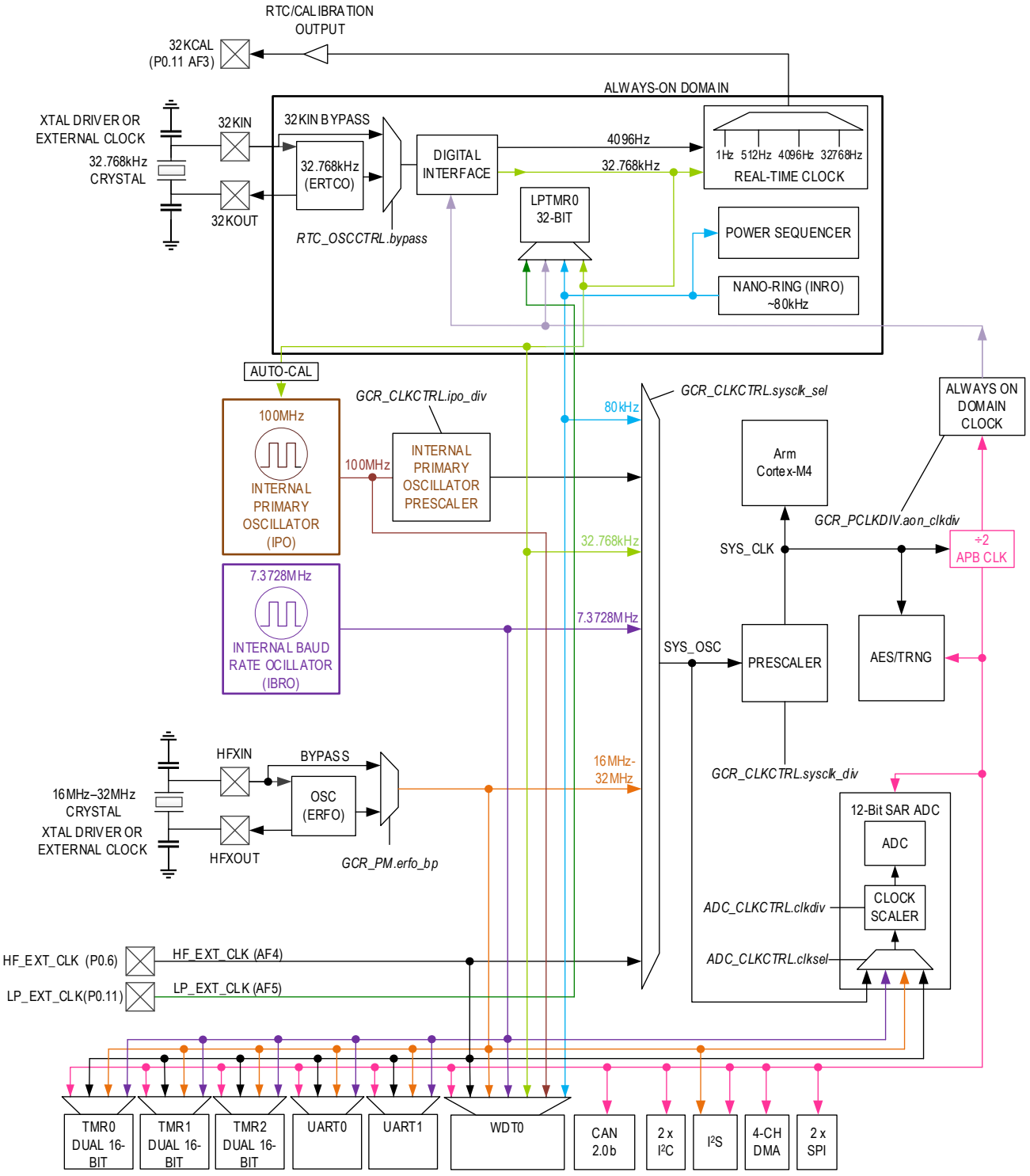
	Reset Source				
Oscillator	**POR**	**System**	**Watchdog**	**Soft**	**Peripheral**
IPO	Enabled	Enabled	Enabled	Retains State	Retains State
IBRO	Disabled	Disabled	Disabled	Retains State	Retains State
INRO	Enabled	Enabled	Enabled	Enabled	Enabled
ERFO	Disabled	Disabled	Disabled	Enabled	Enabled
ERTCO	Disabled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

Table 4-9: Reset Sources and Effect on System Oscillator Selection and Prescaler

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

Figure 4-3 shows a high-level diagram of the MAX32662 clock tree.

Figure 4-3: MAX32662 Clock Block Diagram



4.4 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 4-10*.

Table 4-10: Wake-Up Sources

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

4.4.1 ACTIVE

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

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

4.4.2 SLEEP

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

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

4.4.2.1 Entering SLEEP

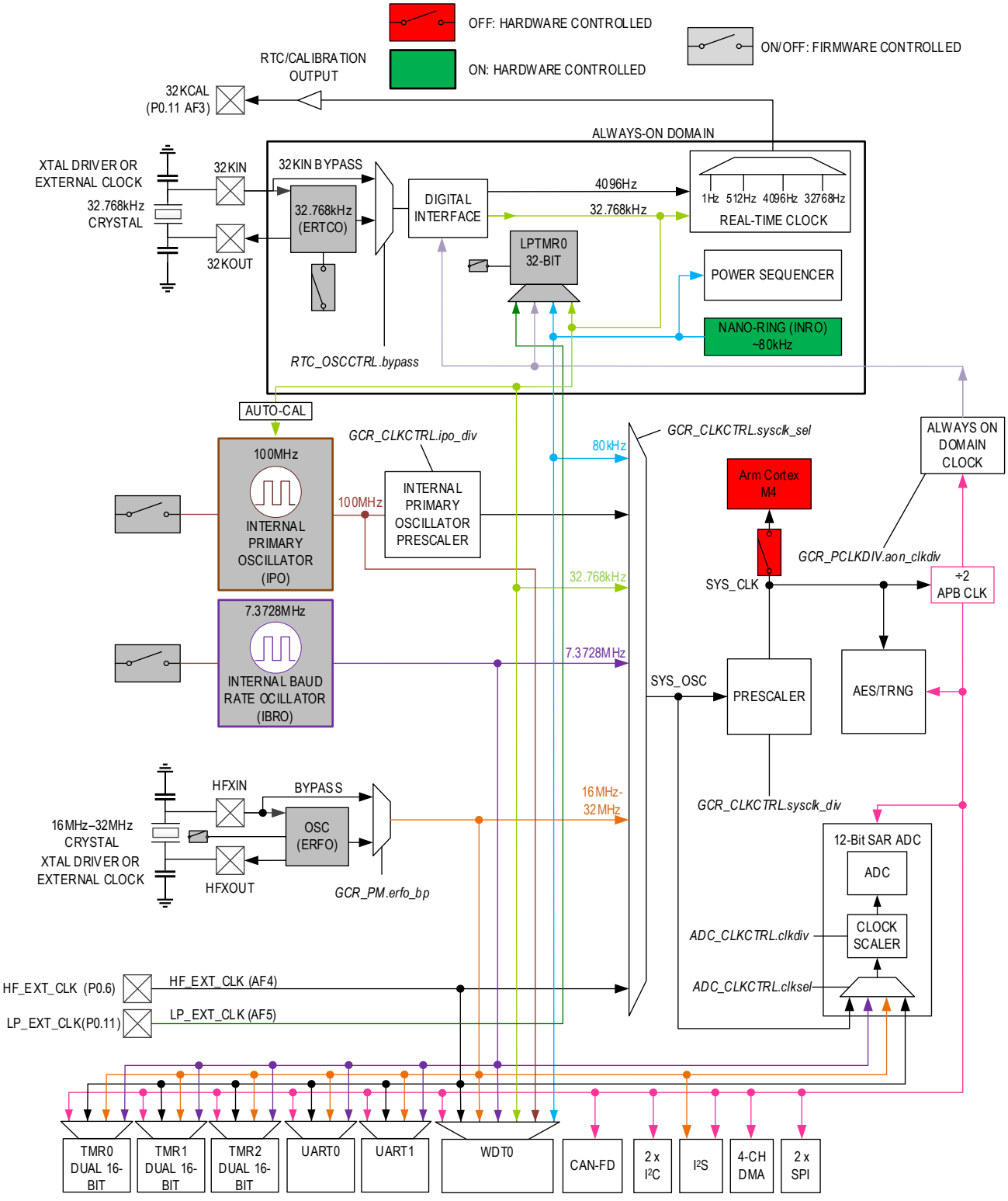
Place the device in SLEEP by performing the following steps:

1. Configure any desired wake-up function.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKFL0*.
   b. *PWRSEQ_LPPWKFL*.
3. Set SCR.sleepdeep to 0.
4. Perform a WFI or WFE instruction.

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

Figure 4-4 shows the clocks available and blocks disabled during *SLEEP*.

Figure 4-4: MAX32662 SLEEP Clock Control



4.4.3 DEEPSLEEP

In this mode, the CPU and critical peripheral configuration settings and all volatile memory is preserved.

The device status is as follows:

- The CPU is powered down.
- System state and all SRAM is retained.
- The GPIO pins retain their state.
- The system oscillators are all disabled to provide additional power savings over *SLEEP*.
- The ERTCO can be enabled by software.
- The LPTMR0 can be active and is an optional wake-up source.

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 11-1* for the clock source table. The LPTMR0 peripheral is disabled/enabled before entering *DEEPSLEEP* by setting/clearing the *MCR_PCLKDIS.tmr3* field to 1.

Note: If LPTMR0 is enabled the outputs associated with this peripheral is no longer available as GPIO.

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

4.4.3.1 Entering DEEPSLEEP

Perform the following steps to enter *DEEPSLEEP*:

1. Configure any desired wake-up function.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKFL0*.
   b. *PWRSEQ_LPPWKFL*.
3. Set SCR.sleepdeep to 1.
4. Perform a WFI or WFE instruction.

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

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

4.4.4 BACKUP

This mode places the CPU in a static, low-power state. *BACKUP* supports the same wake-up sources as *DEEPSLEEP*.

The device status is as follows:

- The CPU is powered down and the state is not maintained.
- The system RAM retention is configurable. See *Table 4-11* for details.
- LPTMR0 can be active and is an optional wake-up source.
- INRO is active.
- The ERTCO can be enabled by software.
- All other oscillators are disabled.

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 11-1* for the clock source table. The LPTMR0 peripheral is disabled/enabled before entering *BACKUP* by setting/clearing the *MCR_PCLKDIS.tmr3* field to 1.

Note: If LPTMR0 is enabled the outputs associated with this peripheral is no longer available as GPIO.

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

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

The boot ROM uses certain ranges of system RAM during a system reset, watchdog timer reset, an external reset, and an exit from *BACKUP*. The boot ROM uses this RAM to perform system checks and to determine if a SCPBL activation pins is asserted. As a result, not all of each RAM can be retained during an exit from *BACKUP*. Table 4-11 details the ranges of RAM that can be retained during an exit from *BACKUP*.

Table 4-11: RAM Retention By Address Range in BACKUP

RAM	RAM Retention Enable Field	Address Range Retention	Amount of RAM Retained
sysram0	*PWRSEQ_LPCTRL.ram0ret_en*	0x2000 0B00 – 0x2000 3FFF	13KB
sysram1	*PWRSEQ_LPCTRL.ram1ret_en*	0x2000 4000 – 0x2000 4DFF 0x2000 5600 – 0x2000 7FFF	14KB
sysram2	*PWRSEQ_LPCTRL.ram2ret_en*	0x2000 8000 – 0x2000 8160 0x2000 81C0 – 0x2000 BFFF	15KB
sysram3	*PWRSEQ_LPCTRL.ram3ret_en*	0x2000 C000 – 0x2000 FFFF	16KB

4.4.4.1 Entering BACKUP

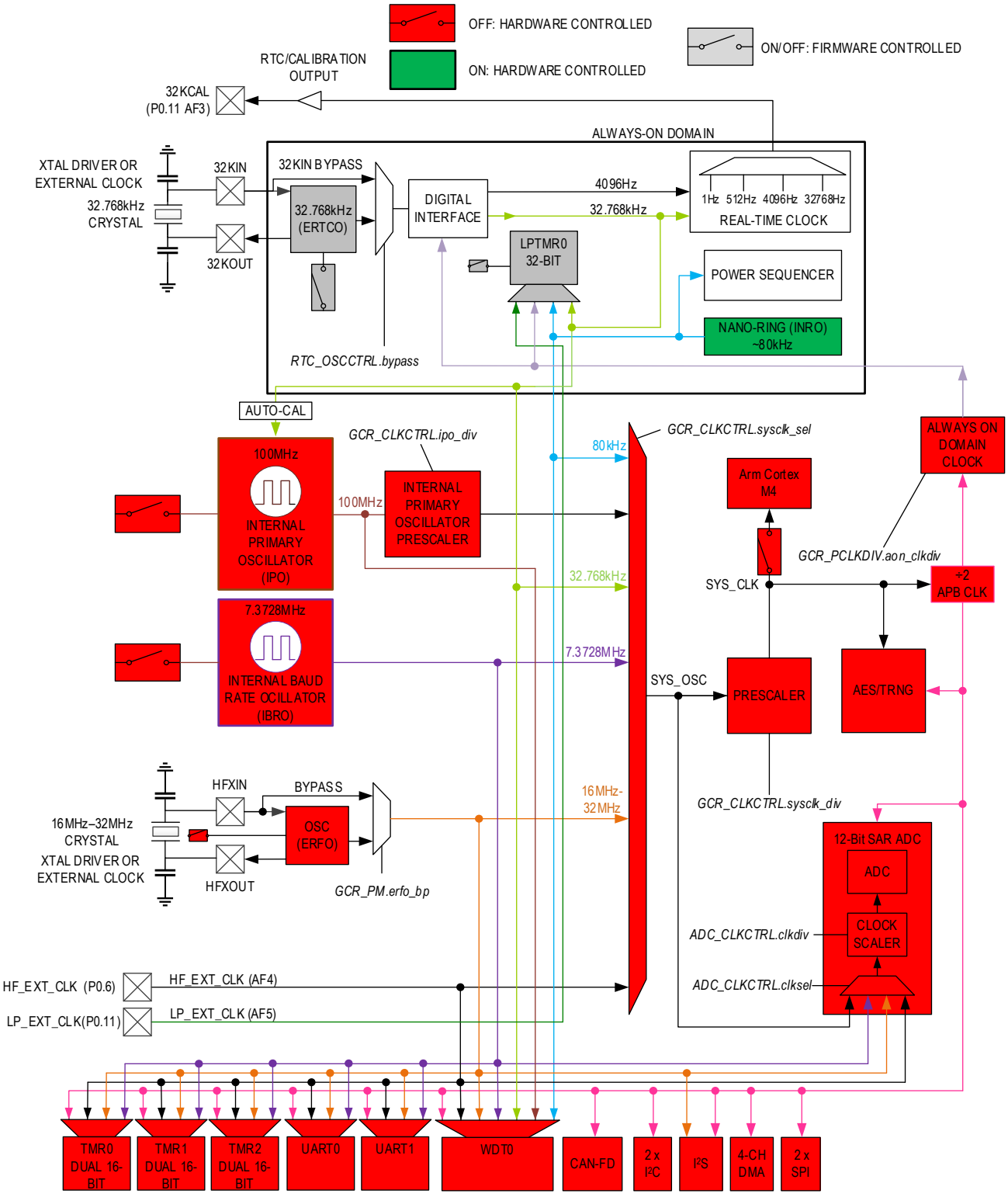
Enter BACKUP by performing the following steps:

1. Configure any desired wake-up functions.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKFL0*.
   b. *PWRSEQ_LPPWKFL*.
3. Set *GCR_PM.mode* = 6.
   a. The device immediately enters *BACKUP*.

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

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

Figure 4-5: MAX32662 DEEPSLEEP and BACKUP Clock Control



4.4.5 STORAGE

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

- No SRAM can be retained.
- LPTMR0 is disabled.

4.4.5.1 Entering STORAGE

Perform the following steps to enter *STORAGE*:

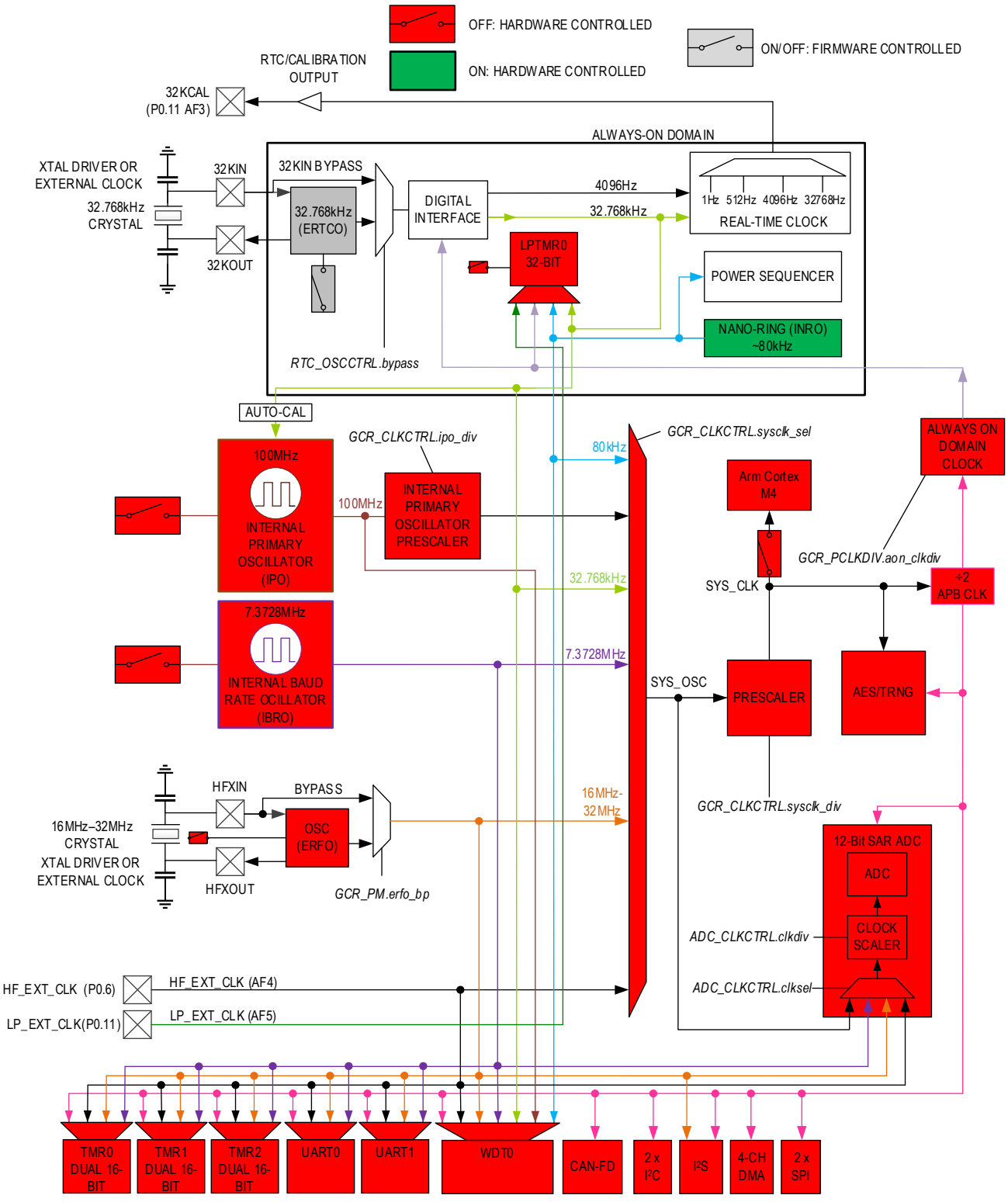
1. Configure any desired wake-up functions.
2. Clear the wake-up status registers by writing 0xFFFF FFFF to each of the following registers:
   a. *PWRSEQ_LPWKFL0*.
   b. *PWRSEQ_LPPWKFL*.
3. Write *PWRSEQ_LPCTRL*.*storage_en* = 1 to set the *STORAGE* bit.
4. Set *GCR_PM*.*mode* = 6.
   a. The device immediately enters *STORAGE*.

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

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

Figure 4-6 shows the clock control during *STORAGE*.

Figure 4-6: MAX32662 STORAGE Clock Control



4.5 Shutdown State

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

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

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

4.6 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}.

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

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

Table 4-12: MAX32662 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
ERTCO	-	-	-	Off
IBRO	-	-	Off	Off
ERFO	-	-	Off	Off
IPO	-	-	On	On
SYS_CLK	On	On	On[2]	On[2]
CPU Clock	On	On	On	On

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

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

Table 4-13: MAX32662 Clock Source and Global Control Register Low-Power Mode Effects

	ACTIVE	SLEEP	DEEPSLEEP	BACKUP[3]	STORAGE
GCR	R	-	-	-	-
INRO	On	On	On	On	On
ERTCO	SW	-	SW	SW	SW
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*. 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: MAX32662 Peripheral and CPU Reset Effects

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

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

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

Table 4-15: MAX32662 Peripheral and CPU Low-Power Mode Effects

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

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

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

4.6.1 Peripheral Reset

As shown in *Table 4-12* and *Table 4-14*, peripheral reset performs a reset for all peripherals. The CPU retains its state. The GPIO, watchdog timers, AoD, RAM retention, and general control registers (GCR), including the clock configuration, are unaffected.

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

4.6.2 Soft Reset

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

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

4.6.3 System Reset

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

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

4.6.4 Power-on Reset (POR)

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

4.6.5 Internal Cache Controller

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

4.6.6 Enabling ICC

Perform the following steps to enable ICC:

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

4.6.7 Disabling ICC

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

4.6.8 Invalidating ICC

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

4.7 ICC Registers

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

Table 4-16: Internal Cache Controller Register Summary

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

4.7.1 Register Details

Table 4-17: ICC Cache Information Register

ICC Cache Information			ICC_INFO	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:10	id	R	*	**Cache ID** 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 4-18: 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 4-19: ICC Cache Control Register

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

Table 4-20: ICC Invalidate Register

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

4.8 RAM Memory Management

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

4.8.1 On-Chip Cache Management

The internal cache controller fetches code from the flash memory. The cache can be enabled, disabled, and zeroized, and the cache clock can be disabled by placing it in *LIGHTSLEEP*. 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 system RAM can be zeroized by setting the respective field to 1 in the *GCR_MEMZ* register.
- ICC 16KB Cache
- Write 1 to *GCR_MEMZ.icc0*

4.8.3 RAM Low-Power Modes

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

4.8.4 RAM LIGHTSLEEP

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

4.8.5 RAM Shutdown

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

4.9 Miscellaneous Control Registers (MCR)

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

Table 4-21: Miscellaneous Control Register Summary

Offset	Register	Description
[0x0004]	*MCR_RST*	*Reset Control Register*
[0x0008]	*MCR_CLKCTRL*	*Clock Control Register*
[0x000C]	*MCR_AINCOMP*	*Analog Input Comparator Register*
[0x0010]	*MCR_LPPIOCTRL*	*Low Power I/O Control Register*
[0x0024]	*MCR_PCLKDIS*	*Clock Disable Register*
[0x0034]	*MCR_AESKEY*	*AES Key Register*
[0x0038]	*MCR_ADCCFG0*	*ADC Config Register 0*
[0x003C]	*MCR_ADCCFG1*	*ADC Config Register 1*
[0x0040]	*MCR_ADCCFG2*	*ADC Config Register 2*

4.9.1 Registers Details

Table 4-22: Reset Control Register

Reset Control			**MCR_RST**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	rtc	R/W1O	0	**RTC Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
2:1	-	RO	0	**Reserved**
0	tmr3	R/W1O	0	**LPTMR0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.

Table 4-23: Clock Control Register

Clock Control			**MCR_CLKCTRL**	**[0x0008]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:18	-	RO	0	**Reserved**
17	ertco_en	R/W	0	**ERTCO Enable** This field enables the ERTCO. 0: Disabled 1: Enabled
16	ertco_pd	R/W	1	**ERTCO Power Down** This field powers down the ERTCO. 0: ERTCO powered on 1: ERTCO powered down
15:0	-	RO	0	**Reserved**

Table 4-24: Analog Comparator Register

Analog Comparator			MCR_AINCOMP	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	RO	-	**Reserved**
29:28	psel_comp1	R/W	0	**Comparator 1 Positive Input Select** This field selects the positive input for comparator 1. Before enabling a comparator input selection, the corresponding GPIO must be configured for the corresponding alternate function for the input. Refer to the device data sheet for the alternate function details. 0: None 1: AIN2 2: AIN3 All other values are reserved. *Note: None is only a valid selection if the comparator is powered down. See MCR_AINCOMP.pd for details on powering down a comparator.*
27:26	-	RO	0	**Reserved**
25:24	nsel_comp1	R/W	0	**Comparator 1 Negative Input Select** This field selects the negative input for comparator 1. Before enabling a comparator input selection, the corresponding GPIO must be configured for the corresponding alternate function for the input. Refer to the device data sheet for the alternate function details. 0: None 1: AIN0 2: AIN1 All other values are reserved. *Note: None is only a valid selection if the comparator is powered down. See MCR_AINCOMP.pd for details on powering down a comparator.*
23:22	-	RO	0	**Reserved**
21:20	psel_comp0	R/W	0	**Comparator 0 Positive Input Select** This field selects the positive input for comparator 0. Before enabling a comparator input selection, the corresponding GPIO must be configured for the corresponding alternate function for the input. Refer to the device data sheet for the alternate function details. 0: None 1: AIN2 2: AIN3 All other values are reserved. *Note: None is only a valid selection if the comparator is powered down. See MCR_AINCOMP.pd for details on powering down a comparator.*
19:18	-	RO	0	**Reserved**
17:16	nsel_comp0	R/W	0	**Comparator 0 Negative Input Select** This field selects the negative input for comparator 0. Before enabling a comparator input selection, the corresponding GPIO must be configured for the corresponding alternate function for the input. Refer to the device data sheet for the alternate function details. 0: None 1: AIN0 2: AIN1 All other values are reserved. *Note: None is only a valid selection if the comparator is powered down. See MCR_AINCOMP.pd for details on powering down a comparator.*

Analog Comparator			MCR_AINCOMP	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
15:4	-	RO	0	**Reserved**
3:2	hyst	RO	0	**Analog Comparator Hysteresis Control** Reserved.
1	pd1	R/W	1	**Analog Comparator 1 Power Down** This field enables analog comparator 1. 0: Enabled 1: Disabled *Note: The analog inputs must be configured before enabling a comparator. See fields MCR_AINCOMP.psel_comp1 and MCR_AINCOMP.nsel_comp1.*
0	pd0	R/W	1	**Analog Comparator 0 Power Down** This field enables analog comparator 0. 0: Enabled 1: Disabled *Note: The analog inputs must be configured before enabling a comparator. See fields MCR_AINCOMP.psel_comp0 and MCR_AINCOMP.nsel_comp0.*

Table 4-25: Low-Power Peripheral I/O Control Register

Low-Power Peripheral I/O Control			MCR_LPPIOCTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
2	tmr3_out_n	R/W	0	**LPTMR0 (TMR3) Inverted Output Control** This field enables peripheral control for the LPTMR0 inverted output signal. When this field is enabled and the LPTMR0 peripheral is enabled (*MCR_PCLKDIS.tmr3* = 0), the LPTMR0 peripheral controls the state of the output signal. Otherwise, the output signal is controlled by GPIO. 0: Disabled 1: LPTMR0 controls the output signal if the LPTMR0 is enabled.
1	tmr3_out	R/W	0	**LPTMR0 (TMR3) Output Control** This field enables peripheral control for the LPTMR0 output signal. When this field is enabled and the LPTMR0 peripheral is enabled (*MCR_PCLKDIS.tmr3* = 0), the LPTMR0 peripheral controls the state of the output signal. Otherwise, the output signal is controlled by GPIO. 0: Disabled 1: LPTMR0 controls the output signal if the LPTMR0 is enabled.
0	tmr3_in	R/W	0	**LPTMR0 (TMR3) Input Control** This field enables peripheral control for the LPTMR0 input signal. When this field is enabled and the LPTMR0 peripheral is enabled (*MCR_PCLKDIS.tmr3* = 0), the LPTMR0 peripheral controls the state of the input signal. Otherwise, the input signal is controlled by GPIO. 0: Disabled 1: LPTMR0 controls the input signal if the LPTMR0 is enabled.

Table 4-26: Clock Disable Register

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

Clock Disable			MCR_PCLKDIS	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
0	tmr3	R/W	1	**LPTMR0 (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. For *DEEPSLEEP* and *BACKUP* operation, see the *DEEPSLEEP* section and the *BACKUP* section. 0: Enabled 1: Disabled

Table 4-27: AES Key Pointer/Status Register

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

Table 4-28: ADC Configuration Register 0

ADC Configuration 0			MCR_ADCCFG0	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
3	int_ref	R/W	0	**ADC Reference Select** This field selects either the 1.25V or 2.048V internal reference when the internal reference is selected (*MCR_ADCCFG0.ext_ref* = 0). 0: 1.25V 1: 2.048V
2	ext_ref	R/W	0	**ADC External Reference Select** This field selects between the internal and external references. The software must configure P0.13 for AF4 to use the external reference. *Note: If the external reference is selected, AIN0 cannot be used as an input to the ADC.* 0: Internal reference 1: External reference
1	-	RO	0	**Reserved**
0	lp_extclk_en	R/W	0	**Low-Power Modes External Clock Input Enable** Set this field to 1 to enable the LP_EXT_CLK (P0.11 AF5) to operate during low-power modes. 0: Disabled. 1: Enabled.

Table 4-29: ADC Configuration Register 1

ADC Configuration 1			MCR_ADCCFG1	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**

ADC Configuration 1			MCR_ADCCFG1	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
11	divsel	R/W	0	**ADC Signal Path Select** This field selects one of the signal divider paths in the analog input buffer path. The setting for this field applies to all analog inputs using the analog input buffer signal path (*MCR_ADCCFG1.thru_pad_sw_en[3]*: *MCR_ADCCFG1.thru_pad_sw_en[0]* = 1). 0b00: Infinite resistance to GND, input divide by 1. 0b01: 50KΩ to GND, input divide by 1. 0b10: 50KΩ to GND, input divide by 2. 0b11: 50KΩ to GND, input divide by 3.
10	amp_rri_en	R/W	0	**Amplifier Rail-to-Rail Input Enable** This field enables rail-to-rail inputs (0.1V to V_{DD} - 0.1V) for the amplifier buffer path. This field applies to all inputs using the analog input buffer signal path(*MCR_ADCCFG1.thru_pad_sw_en[3]*:*MCR_ADCCFG1.thru_pad_sw_en[0]* = 1). 0: Inputs restricted to V_{GS} + V_{DSAT} to V_{DD} - 0.1V. 1: Rail-to-rail input enabled.
9	amp_en	R/W	0	**Amplifier Enable** Set this field to 1 to enable the buffer amplifier using in the buffer path.
8	thru_en	R/W	0	**MUX Switches Enable** Set this field to enable the MUX switches (*MCR_ADCCFG1.thru_pad_sw_en[3]*: *MCR_ADCCFG1.thru_pad_sw_en[0]*) using the analog input buffer path. 0: Disabled. 1: Enabled.
7:4	ain_inp_en	R/W	0	**Analog Input Enable** Set each bit in this field to 1 to enable the ADC input switch for the corresponding input channel in the ADC. This field must be set to 1 to use the input with the ADC. 0: Disabled. 1: Enabled.
3:0	thru_pad_sw_en	R/W	0	**MUX Pad Switch Enable for AIN3:AIN0** Set each bit in this field to 1 to enable the input switch for the corresponding input channel in the analog input buffer path. 0: Disabled. 1: Enabled.

Table 4-30: ADC Configuration Register 2

ADC Configuration 2			MCR_ADCCFG2	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
24	d_iboost	R/W	0	**Extra Drive Current Enable** See the *ADC SFR Interface* for details on this fields usage.
23:22	-	DNM	0	**Reserved, Do Not Modify**
21:20	vcm	R/W	0	**V_{CM} Reference Buffer Output Trim Code** See the *ADC SFR Interface* for details on this fields usage.
19:16	idrv	R/W	0	**Reference Buffer Drive Strength Trim Code** See the *ADC SFR Interface* for details on this fields usage.
15	-	RO	0	**Reserved**

ADC Configuration 2			MCR_ADCCFG2	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
14:8	vrefp	R/W	0	**Trim Code for V_{REFP}** See the *ADC SFR Interface* for details on this fields usage.
7	-	RO	0	**Reserved**
6:0	vrefm	R/W	0	**Trim Code for V_{REFM}** See the *ADC SFR Interface* for details on this fields usage.

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

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* for an explanation of the read and write access of each field. The PWRSEQ registers are only reset on a POR.

Table 4-31: Power Sequencer and Always-On Domain Register Summary

Offset	Register	Description
[0x0000]	*PWRSEQ_LPCTRL*	*Low-Power Control Register*
[0x0004]	*PWRSEQ_LPWKFL0*	*GPIO0 Low-Power Wakeup Status Flags*
[0x0008]	*PWRSEQ_LPWKEN0*	*GPIO0 Low-Power Wakeup Enable Register*
[0x0030]	*PWRSEQ_LPPWKFL*	*Peripheral Low-Power Wakeup Status Flags*
[0x0034]	*PWRSEQ_LPPWKEN*	*Peripheral Low-Power Wakeup Enable Register*
[0x0040]	*PWRSEQ_LPMEMSD*	*RAM Shutdown Control Register*

4.10.1 Register Details

Table 4-32: Low-Power Control Register

Low-Power Control			PWRSEQ_LPCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	DNM	0	**Reserved. Do Not Modify**
29	ertco_en	R/W	0	**ERTCO Low-Power Mode Control** This bit allows control of the ERTCO for low-power modes. 0: Power sequencer controls the ERTCO. 1: The ERTCO is enabled in all low-power modes. *Note: If STORAGE is enabled (PWRSEQ_LPCTRL.storage_en = 1), this field is ignored for STORAGE.*
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. *Note: If STORAGE is enabled (PWRSEQ_LPCTRL.storage_en = 1), this field is ignored for STORAGE.*
27	vbbmon_dis	R/W	0	**ADC Monitor Disable** This field controls the power monitor on the ADC digital supply in all operating modes. 0: Enable if *PWRSEQ_LPCTRL.bg* is enabled. 1: Disabled.
26	-	RO	0	**Reserved**

Low-Power Control			PWRSEQ_LPCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
25	porvddmon_dis	R/W	0	**V_{CORE} Supply POR Monitor Disable** This field controls the POR monitor for V_{CORE} in all operating modes. 0: Enabled. 1: Disabled.
24:23	-	RO	0	**Reserved**
22	vddamon_dis	R/W	0	**V_{DDA} Analog Supply Power Monitor Disable** 0: Enabled. 1: Disabled.
21	-	RO	0	**Reserved**
20	vcoremon_dis	R/W	0	**V_{CORE} Supply Power Monitor Disable** Set this field to 1 to disable the V_{CORE} power monitor in all operating modes. If the V_{CORE} supply is not enabled, this field's setting is ignored. 0: Enabled if V_{CORE} supply is connected externally. 1: Disabled.
19:18	-	DNM	0	**Reserved, Do Not Modify**
17	vcore_ext	R/W	0	**V_{CORE} Supply** Setting this bit allows the V_{CORE} device pin to be used as the internal 1V supply.
16	ldo_dis	R/W	1	**LDO Disable** This field initializes to 1 on a POR until the hardware determines if an external power source is connected to the V_{CORE} device pin. If no power supply is connected, this bit is set to 0 by the hardware. If a power supply is connected to the V_{CORE} device pin, the bit remains set to 1. Set this field to 1 to manually disable the LDO. 0: Enabled 1: Disabled
15:13	-	RO	0	**Reserved**
12	vcorepor_dis	R/W	1	**V_{CORE} POR Disable for *DEEPSLEEP* and *BACKUP*** Setting this bit to 1 blocks the POR signal to the core when the device is in *DEEPSLEEP* or *BACKUP* operation. Disconnecting the POR signal from the core during *DEEPSLEEP* and *BACKUP* prevents the core from detecting a POR event while the device is in *DEEPSLEEP* or *BACKUP*. 0: POR signal is connected to the core during *DEEPSLEEP* and *BACKUP*. 1: POR signal is not connected to the core during *DEEPSLEEP* and *BACKUP*.
11	bg_dis	R/W	1	**Bandgap Disable for *DEEPSLEEP* and *BACKUP*** Setting this field to 1 disables the bandgap during *DEEPSLEEP* and *BACKUP*. 0: Enabled 1: Disabled

Low-Power Control			PWRSEQ_LPCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
10	fastwk_en	R/W	0	**Fast Wake-up Enable for *DEEPSLEEP* Mode** Set to 1 to enable fast wake-up from *DEEPSLEEP*. When enabled, the system exits *DEEPSLEEP* faster by: 1) Bypassing the INRO warmup 2) Reducing the warmup time for the IPO 3) Reducing the warmup time for the LDO 4) Code execution resumes at the next instruction after the entry to *DEEPSLEEP*. When fast wake-up 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* Mode Enable** When this field is set and the device enters BACKUP, STORAGE is actually entered. 0: Disabled. 1: Enabled. *Note: Setting this bit causes the device to enter STORAGE when setting GCR_PM.mode to BACKUP.*
8	retreg_en	R/W	1	**RAM Retention Regulator Enable for *BACKUP*** This field selects the source used to retain the RAM contents during *BACKUP* operation. Setting this field to 0 sets the V_{DD} supply for RAM retention during *BACKUP* and disables the RAM retention regulator. 0: RAM retention regulator disabled. The V_{DD} supply is used to retain the state of the internal SRAM as configured by the *PWRSEQ_LPCTRL.ram0ret_en, PWRSEQ_LPCTRL.ram1ret_en, PWRSEQ_LPCTRL.ram2ret_en, and PWRSEQ_LPCTRL.ram3ret_en* fields. 1: RAM retention regulator enabled. RAM retention in *BACKUP* is configured with the *PWRSEQ_LPCTRL.ram0ret_en, PWRSEQ_LPCTRL.ram1ret_en, PWRSEQ_LPCTRL.ram2ret_en, and PWRSEQ_LPCTRL.ram3ret_en* fields.
7	fvdd_en	R/W	0	**Flash V_{DD} Enable During Low Power Modes** 0: Disabled. 1: Enabled.
6	vcore_det_bypass	R/W	0	**Bypass V_{CORE} External Supply Detection** Set this field to 1 if the system runs from a single supply only and V_{CORE} is not connected to an external supply. Bypassing the hardware detection of an external supply on V_{CORE} enables a faster wake-up time. 0: Enabled. 1: Disabled.

Low-Power Control			PWRSEQ_LPCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
5:4	ovr	R/W	0b10	**Output Voltage Range for Internal Regulator** Set this field to control the output voltage of the internal regulator, allowing selection of the internal core operating voltage and the frequency of the IPO. On POR, this field defaults to 1.1V output ± 10% with f_{IPO} = 100MHz. ***Note**: If V_{CORE} is connected to an external supply voltage, this field should be modified only to set it to match the input voltage on V_{CORE}. The external supply must be equal to or greater than this field setting indication.* **Dual-Supply Operation** 0b00: V_{CORE} = 0.9V, f_{IPO} = 12MHz. 0b01: V_{CORE} = 1.0V, f_{IPO} = 50MHz.. 0b10: V_{CORE} = 1.1V, f_{IPO} = 100MHz 0b11: Reserved. **Single-Supply Operation (V_{CORE} = GND)** 0b00: V_{LDO} = 0.9V, f_{IPO} = 12MHz. 0b01: V_{LDO} = 1.0V, f_{IPO} = 50MHz. 0b10: V_{LDO} = 1.1V, f_{IPO} = 100MHz. 0b11: Reserved.
3	ram3ret_en	R/W	0	***Sysram3* and *Sysram7* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram3* and *sysram7*. See *Table 3-1* for system RAM configuration. 0: Data retention for *sysram3* and *sysram7* address space disabled in *BACKUP*. 1: Data retention for *sysram3* and *sysram7* address space enabled in *BACKUP*. *Note: This field is used in conjunction with PWRSEQ_LPCTRL.retreg_en to control RAM retention.*
2	ram2ret_en	R/W	0	***Sysram2* and *Sysram6* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram2* and *sysram6*. See *Table 3-1* for system RAM configuration. 0: Data retention for *sysram2* and *sysram6* address space disabled in *BACKUP*. 1: Data retention for *sysram2* and *sysram6* address space enabled in *BACKUP*. *Note: This field is used in conjunction with PWRSEQ_LPCTRL.retreg_en to control RAM retention.*
1	ram1ret_en	R/W	0	***Sysram1* and *Sysram5* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram1* and *sysram5*. See *Table 3-1* for system RAM configuration. 0: Data retention for *sysram1* and *sysram5* address space disabled in *BACKUP*. 1: Data retention for *sysram1* and *sysram5* address space enabled in *BACKUP*. *Note: This field is used in conjunction with PWRSEQ_LPCTRL.retreg_en to control RAM retention.*
0	ram0ret_en	R/W	0	***Sysram0* and *Sysram4* Data Retention Enable for *BACKUP*** Set this field to 1 to enable data retention for *sysram0* and *sysram4*. See *Table 3-1* for system RAM configuration. 0: Data retention for *sysram0* and *sysram4* address space disabled in *BACKUP*. 1: Data retention for *sysram0* and *sysram4* address space enabled in *BACKUP*. *Note: This field is used in conjunction with PWRSEQ_LPCTRL.retreg_en to control RAM retention.*

Table 4-33: GPIO0 Low-Power Wakeup Status Flags

GPIO0 Low-Power Wakeup Status Flags			PWRSEQ_LPWKFL0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	wakefl	R/W1C	0	**GPIO0 Pin Wakeup Status Flag** Whenever a GPIO0 pin, in any power mode, transitions from low-to-high or high-to-low, the corresponding bit in this register is set. The device transitions from a low-power to *ACTIVE* if the corresponding interrupt enable bit is set in *PWRSEQ_LPWKEN0*. This register should be cleared before entering any low-power mode. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 4-34: GPIO0 Low-Power Wakeup Enable Registers

GPIO0 Low-Power Wakeup Enable			PWRSEQ_LPWKEN0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	wakeen	R/W	0	**GPIO0 Pin Wakeup Interrupt Enable** Setting a bit in this register causes an interrupt to be generated and wakes up the device from any low-power mode to *ACTIVE* if the corresponding bit in the *PWRSEQ_LPWKFL0* 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 4-35: Peripheral Low-Power Wakeup Status Flags

Peripheral Low-Power Wakeup Status Flags			PWRSEQ_LPPWKFL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**
16	backup	R/W1C	0	**BACKUP Wakeup Flag** This field is set when the device wakes up from *BACKUP*.
15:7	-	RO	0	**Reserved**
6	aincomp1_st	R	*	**Analog Comparator 1 Output Status** This field returns the output status of the analog comparator 1.
5	aincomp0_st	R	*	**Analog Comparator 0 Output Status** This field returns the output status of the analog comparator 0.
4	aincomp1	R/W1C	0	**Analog Comparator 1 Wakeup Flag** This field is set when this peripheral causes the CPU to wake to *ACTIVE*. 0: Normal operation 1: 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_LPCTRL.bg_dis is cleared to 0.*

Peripheral Low-Power Wakeup Status Flags			PWRSEQ_LPPWKFL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
3	aincomp0	R/W1C	0	**Analog Comparator 0 Wakeup Flag** This field is set when this peripheral causes the CPU to wake to *ACTIVE*. 0: Normal operation 1: 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_LPCTRL.bg_dis is cleared to 0.*
2:1	-	RO	0	**Reserved**
0	tmr3	R/W1C	0	**LPTMR0 Wakeup 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_LPCTRL.bg_dis is cleared to 0.*

Table 4-36: Peripheral Low-Power Wakeup Enable Register

Peripheral Low-Power Wakeup Enable			PWRSEQ_LPPWKEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	aincomp1	R/W	0	**Analog Comparator 1 Wakeup Enable** Setting this bit enables an interrupt and wakes up the device from any low-power mode when *PWRSEQ_LPPWKFL.aincomp1* is set to 1.
3	aincomp0	R/W	0	**Analog Comparator 0 Wakeup Enable** Setting this bit enables an interrupt and wakes up the device from any low-power mode when *PWRSEQ_LPPWKFL.aincomp0* is set to 1.
2:1	-	RO	0	**Reserved**
0	lptmr0	R/W	0	**LPTMR0 Wakeup Enable** Setting this bit enables an interrupt and wakes up the device from any low-power mode when *PWRSEQ_LPPWKFL.tmr3* does not equal 0. 0: Disabled 1: Enabled

Table 4-37: RAM Shutdown Control Register

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

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

4.11 Trim System Initialization Registers (TRIMSIR)

See *Table 3-2* for the base address of this peripheral/module. See *Table 1-1* for an explanation of the read and write access of each field. The TRIMSIR is only reset on a POR.

Table 4-38: Trim System Initialization Register Summary

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

4.11.1 Register Details

Table 4-39: TRIMSIR Configuration 2 Register

TRIMSIR Configuration 2			TRIMSIR_BB_SIR2	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:13	trim_ibro_rbias		*	**IBRO Bias Trim**
12:8	-	DNM	0	**Reserved, Do Not Modify**
7:0	trim_ibro	RO	*	**IBRO Trim Value**

Table 4-40: TRIMSIR Configuration 6 Register

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

4.12 Global Control Registers (GCR)

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

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

Table 4-41: 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*

4.12.1 Register Details

Table 4-42: System Control Register

System Control			GCR_SYSCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	chkres	R	0	**ROM Checksum Calculation Pass/Fail** This field is the result after setting bit *GCR_SYSCTRL.cchk*. This bit is only valid after the ROM checksum is complete and *GCR_SYSCTRL.cchk* is cleared. 0: Pass 1: Fail
14	swd_dis	R/W	0	**Serial Wire Debug Disable** This bit is used to disable the serial wire debug interface. 0: SWD Disabled 1: SWD Enabled *Note: This bit is only writeable if the if the GCR_SYSST.icelock word is not programmed.*
13	cchk	R/W	0	**Calculate ROM Checksum** This bit is self-clearing when the ROM checksum calculation is complete, and the result is available at bit *GCR_SYSCTRL.chkres*. Writing a 0 has no effect. 0: No operation 1: Start ROM checksum calculation

System Control			GCR_SYSCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
12:7	-	RO	0	**Reserved**
6	icc0_flush	R/W	0	**ICC Code 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: Normal operation. 1: Flush the memory.
5	fpus_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 4-43: Reset Register 0

Reset 0			GCR_RST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31	sys	R/W	0	**System Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *See the Device Resets section for additional information.*
30	periph	R/W	0	**Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *Note: Watchdog timers, GPIO ports, the AoD, RAM retention, and the GCR are unaffected. See the Device Resets section for additional information.*
29	soft	R/W	0	**Soft Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *See the Device Resets section for additional information.*
28:27	-	RO	0	**Reserved**
26	adc	R/W	0	**ADC Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
25	-	RO	0	**Reserved**
24	trng	R/W	0	**TRNG Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
23:20	-	RO	0	**Reserved**
19	can	R/W	0	**CAN Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
18:17	-	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**

Reset 0				GCR_RST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**	
14	spi1	R/W	0	**SPI1 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
13	spi0	R/W	0	**SPI0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
12	uart1	R/W	0	**UART1 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
11	uart0	R/W	0	**UART0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
10:8	-	RO	0	**Reserved**	
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.	
4:3	-	RO	-	**Reserved**	
2	gpio0	R/W	0	**GPIO0 Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.	
1	wdt	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 4-44: System Clock Control Register

System Clock Control				GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**	
31	extclk_rdy	R	1	**External Clock Ready** This bit field is set when the signal on the P0.6 device pin, driven by an external clock, is ready. 0: Not ready or not enabled. 1: External clock is ready.	
30	-	RO	1	**Reserved**	
29	inro_rdy		0	**INRO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.	
28	ibro_rdy	R	0	**IBRO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.	
27	ipo_rdy	R	0	**IPO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.	
26	-	RO	0	**Reserved**	

System Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
25	ertco_rdy	R	0	**ERTCO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
24	erfo_rdy	R	0	**ERFO Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.
23:22	-	RO	0	**Reserved**
21	ibro_vs	R/W	0	**IBRO Voltage Source Select** In *DEEPSLEEP*, the IBRO voltage is sourced by a dedicated internal 1V regulated supply. When exiting *DEEPSLEEP*, the voltage is automatically switched back to this bit's setting. 0: Dedicated internal 1V regulated supply. 1: V_{CORE} supply
20	ibro_en	R/W	0	**IBRO Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCTRL.ibro_rdy* = 1.
19	ipo_en	R/W	1	**IPO Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCTRL.ipo_rdy* = 1.
18:17	-	DNM	0	**Reserved. Do Not Modify.**
16	erfo_en	R/W	0	**ERFO Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCTRL.erfo_rdy* = 1.
15:14	ipo_div	R/W	0	**IPO Prescaler** Divides the IPO clock before it is selected as SYS_OSC. 0: Divide by 1 1: Divide by 2 2: Divide by 4 3: Divide by 8
13	sysclk_rdy	R	0	**SYS_OSC Select Ready** When SYS_OSC is changed by modifying the *GCR_CLKCTRL.sysclk_sel* field, there is a delay until the switchover is complete. This bit is cleared until the switchover is complete. 0: Switch to new clock source not yet complete. 1: SYS_OSC is the clock source selected in *GCR_CLKCTRL.sysclk_sel*.
12	-	RO	0	**Reserved**
11:9	sysclk_sel	R/W	4	**System Oscillator Source Select** Selects the system oscillator (SYS_OSC) source used to generate the system clock (SYS_CLK). Modifying this field clears *GCR_CLKCTRL.sysclk_rdy* immediately. 0: Reserved 1: Reserved 2: ERFO 3: INRO 4: IPO 5: IBRO 6: ERTCO 7: External Clock

System Clock Control			GCR_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
8:6	sysclk_div	R/W	0	**System Oscillator Prescaler** Sets the divider for generating SYS_CLK from the selected SYS_OSC. See *Equation 4-3* for details.
5:0	-	RO	0b001000	**Reserved**

Table 4-45: Power Management Register

Power Management			GCR_PM	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0	**Reserved**
20	erfo_bp	R/W	0	**ERFO Bypass** This bit is set to 0 on a POR and is not affected by other resets. 0: The clock source is a crystal oscillator, driving the crystal connected between the HFXIN and HFXOUT pins. 1: The clock source is a square wave driven into the HFXIN pin.
19:18	-	RO	0	**Reserved**
17:16	-	DNM	0b11	**Reserved, Do Not Modify** This field must be set to 0b11.
15:13	-	RO	0	**Reserved**
12	-	DNM	1	**Reserved, Do Not Modify**
11:8	-	RO	0	**Reserved**
7	aincomp_we	R/W	0	**AINCOMP Wakeup Enable** Set this field to 1 to enable the comparators as a wake-up source. The analog comparators can wake the device from *SLEEP*, *DEEPSLEEP*, or *BACKUP*.
6	tmr3_we	R/W	0	**LPTMR0 (TMR3) Wakeup Enable** Set this field to 1 to enable LPTMR0 as a wake-up source. LPTMR0 wakes the device from *SLEEP*, *DEEPSLEEP*, or *BACKUP*. 0: Disabled 1: Enabled
5	rtc_we	R/W	0	**RTC Alarm Wakeup Enable** Set this field to 1 to enable an RTC alarm to wake the device. The RTC alarm wakes the device from *SLEEP*, *DEEPSLEEP*, *BACKUP*, or *STORAGE*. 0: Disabled 1: Enabled
4	gpio_we	R/W	0	**GPIO Wakeup Enable** Set this field to 1 to enable all GPIO pins as potential wake-up sources. Any GPIO configured for wake-up can wake the device from *SLEEP*, *DEEPSLEEP*, *BACKUP*, or *STORAGE*. 0: Disabled 1: Enabled
3	-	RO	0	**Reserved**

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

Table 4-46: Peripheral Clock Divisor Register

Peripheral Clocks Divisor		GCR_PCLKDIV	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	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 4-47: Peripheral Clock Disable Register 0

Peripheral Clock Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	RO	1	**Reserved**
29	pt	R/W	1	**Pulse Train Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
28	i2c1	R/W	1	**I²C1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
27:24	-	RO	1	**Reserved**
23	adc	R/W	1	**ADC Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
22:18	-	RO	1	**Reserved**
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.

Peripheral Clock Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
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	uart1	R/W	1	**UART1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
9	uart0	R/W	1	**UART0 Clock Disable** Disabling a clock peripheral disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled. 1: Disabled.
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.
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.
4:1	-	RO	1	**Reserved**

Peripheral Clock Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
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 4-48: Memory Clock Control Register

Memory Clock Control			GCR_MEMCTRL	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:14	-	RO	0	**Reserved**
13	romls_en	R/W	0	**ROM LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled.
12	icc0ls_en	R/W	0	**ICC LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled.
11	ram3ls_en	R/W	0	***Sysram3* and *Sysram7* LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 3-1 for base address and size information.*
10	ram2ls_en	R/W	0	***Sysram2* and *Sysram6* LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 3-1 for base address and size information.*
9	ram1ls_en	R/W	0	***Sysram1* and *Sysram5* LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 3-1 for base address and size information.*
8	ram0ls_en	R/W	0	***Sysram0* and *Sysram4* LIGHTSLEEP Enable** Data is unavailable for read/write operations in *LIGHTSLEEP* mode but is retained. 0: Normal operation. 1: Enabled. *Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the PWRSEQ_LPMEMSD register. See Table 3-1 for base address and size information.*
7:5	-	RO	0	**Reserved**

Memory Clock Control			GCR_MEMCTRL	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
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	0b101	**Flash Wait States** This field sets the number of SYS_OSC wait-state cycles per flash code read access. 0: Reserved 1-7: Number of flash memory wait states.

Table 4-49: Memory Zeroization Control Register

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

Table 4-50: 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 4-51: Reset Register 1

Reset 1			GCR_RST1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23	i2s	R/W1O	0	**I²S Peripheral Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
22:15	-	RO	0	**Reserved**
14	ac	R/W1O	0	**Auto Calibration Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
13:11	-	RO	-	**Reserved**
10	aes	R/W1O	0	**AES Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
9:2	-	RO	0	**Reserved**
1	pt	R/W1O	0	**Pulse Train Peripheral Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
0	i2c1	R/W1O	0	**I²C 1 Peripheral Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Table 4-52: Peripheral Clock Disable Register 1

Peripheral Clock Disable 1			GCR_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	1	**Reserved**
23	i2s	R/W	1	**I²S Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
22:17	-	RO	1	**Reserved**
16	aes_key	R/W	1	**AES Key Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.

Peripheral Clock Disable 1			GCR_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
15	aes	R/W	1	**AES Block Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
14:12	-	RO	1	**Reserved**
11	can	R/W	1	**CAN 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.
10:5	-	RO	1	**Reserved**
4	wdt	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.
1:0	-	RO	1	**Reserved**

Table 4-53: Event Enable Register

Event Enable			GCR_EVENTEN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	tx	R/W	0	**TXEV On SEV Enable** When set, a SEV (Send Event) instruction causes a TXEV event from the CPU. 0: Disabled. 1: Enabled.
1	rx	R/W	0	**RXEV Event Enable** Set this field to 1 to enable the generation of an RXEV event to wake the CPU from a Wait for Event (WFE) sleep state. See the device data sheet for AF details. Not all alternate functions are available on all packages. 0: Disabled. 1: Enabled.
0	dma	R/W	0	**CPU DMA CTZ Wake-Up Enable** Allows a DMA CTZ event to generate an RXEV to wake the CPU from a low-power mode entered with a WFE instruction. See the device data sheet for AF details. Not all alternate functions are available on all packages. 0: Disabled. 1: Enabled.

Table 4-54: Revision Register

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

Table 4-55: System Status Interrupt Enable Register

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

4.13 System Initialization Registers (SIR)

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

Table 4-56: System Initialization Register Summary

Offset	Register	Description
[0x0000]	*SIR_SISTAT*	*System Initialization Error Status Register*
[0x0004]	*SIR_SIADDR*	*System Initialization Error Address Register*

4.13.1 Register Details

Table 4-57: System Initialization Error Status Register

System Initialization Error Status			SIR_SISTAT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	user_magic	RO	*	**User Configuration Valid** 0: Configuration invalid. 1: Configuration valid.
1	crcerr	RO	*	**Configuration Error Flag** This field is set by hardware during reset if an error in the device configuration is detected. 0: Configuration valid. 1: Configuration invalid. *Note: If this field reads 1, a device error has occurred. Contact technical support for additional assistance providing the address contained in SIR_SIADDR.erraddr.*

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

Table 4-58: System Initialization Error Address Register

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

4.14 Function Control Registers (FCR)

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

Table 4-59: 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*
[0x0018]	*FCR_ADCREFTRIM0*	*1.25V Reference Trim Values*
[0x001C]	*FCR_ADCREFTRIM1*	*2.048 Reference Trim Values*
[0x0020]	*FCR_ADCREFTRIM2*	*External Reference Trim Values*
[0x0024]	*FCR_ERFOKS*	*ERFO Kick Start Register*

4.14.1 Register Details

Table 4-60: Function Control 0 Register

Function Control 0			FCR_FCTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23	i2c1dgen1	R/W	0	**I²C1 SCL Glitch Filter Enable** 0: Disabled 1: Enabled
22	i2c1dgen0	R/W	0	**I²C1 SDA Glitch Filter Enable** 0: Disabled 1: Enabled

Function Control 0			FCR_FCTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
21	i2c0dgen1	R/W	0	**I²C0 SCL Glitch Filter Enable** 0: Disabled 1: Enabled
20	i2c0dgen0	R/W	0	**I²C0 SDA Glitch Filter Enable** 0: Disabled 1: Enabled
19:9	-	RO	0	**Reserved**
8	keywipe_sys	R/W	0	**AES Key Wipe** Set this field to 1 to wipe the AES key.
7:3	-	RO	0	**Reserved**
2:0	erfo_range_sel	R/W	0	**ERFO Frequency Range Select** Set these bits to reflect the crystal frequency connected to the HFXOUT and HFXIN device pins. 0: < 22.5MHz 1: 22.5MHz to 24.5MHz 2: 24.5MHz to 26.3MHz 3: 26.3MHz to 28.0MHz 4: 28.0MHz to 29.6MHz 5: 29.6MHz to 31.1MHz 6: 31.1MHz to 32.6MHz 7: Reserved

Table 4-61: Automatic Calibration 0 Register

Function Control 1			FCR_AUTOCAL0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:23	trim	R	0	**IPO Trim Value** This field contains the calculated trim output from an automatic calibration run. See *IPO Calibration* for details.
22:20	-	RO	0	**Reserved**
19:8	mu	R/W	0	**IPO Trim Adaptation Gain** Load this field with the desired number of trim adjustment pulses required before the trim is updated during automatic calibration operation. The recommended value for this field is 4. See *IPO Calibration* for details.
7:5	-	RO	0	**Reserved**
4	atomic	R/W1O	0	**IPO Trim Atomic Start** Set this bit to start an atomic calibration of the IPO. The calibration runs for *FCR_AUTOCAL2.donecnt* milliseconds. This bit is cleared by hardware once the calibration is complete. See *IPO Calibration* for details.
3	gain_inv	RO	0	**IPO Trim Step Invert**
2	load	R/W1O	0	**IPO Initial Trim Load** Set this bit to load the initial trim value for the IPO from *FCR_AUTOCAL1.inittrim*. This bit is cleared by hardware once the load is complete. See *IPO Calibration* for details.
1	acrun	R/W	0	**IPO Automatic Calibration Run** 0: Not running. 1: Running.

Function Control 1			FCR_AUTOCAL0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
0	acen	R/W	0	**IPO Automatic Calibration Enable** Selects the trim value to use for the IPO. The reset default for this field uses the factory trim for the IPO. Setting this field to 1 uses the automatic calibration trim output stored in *FCR_AUTOCAL0.trim*. See *IPO Calibration* for details. 0: Use default trim from the factory. 1: Use automatic calibration trim value calculated and stored in *FCR_AUTOCAL0.trim*.

Table 4-62: Automatic Calibration 1 Register

Function Control 2			FCR_AUTOCAL1	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8:0	inittrim	R/W	0	**IPO Automatic Calibration Initial Trim** This field contains the initial trim setting for the IPO. Set this field to the desired initial trim value to use for an IPO automatic calibration operation. The closer this field is to the target trim value required, the faster the automatic trim operation completes. Set this field to 0x40 when performing auto calibration. See *IPO Calibration* for details. *Note: Valid values for this field are 1 to 0xFF.*

Table 4-63: Automatic Calibration 2 Register

Function Control 2			FCR_AUTOCAL2	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0	**Reserved**
20:8	acdiv	R/W	0	**IPO Trim Automatic Calibration Divide Factor** The target frequency of the calibration is determined by dividing the IPO frequency by 32,768 before comparing it with the ERTCO frequency. For example, to achieve a target IPO frequency of 100MHz, load this field with 0xE4E (3,051). See *IPO Calibration* for details. $acdiv = \frac{f_{IPO\_TARGET}}{32768}$ *Note: Setting acdiv to 0 is equivalent to setting acdiv to 1.*
7:0	donecnt	R/W	0	**IPO Trim Automatic Calibration Run Time** Set this field to the desired number of milliseconds for the atomic calibration run time for the IPO. See *IPO Calibration* for details. $Atomic\ Run\ Time = donecnt\ milliseconds$

Table 4-64: ADC 1.25V Reference Trim Register

ADC 1.25V Reference Trim			FCR_ADCREFTRIM0	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	R	0	**Reserved**
29:24	vx2_tune		*	**Tuning Capacitor In-Line DAC** See *1.25V Internal Reference Trim* for details on this field.
23:18	-	RO	0	**Reserved**

ADC 1.25V Reference Trim			FCR_ADCREFTRIM0	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
17:16	vcm	R	*	**Trim Code for V_{CM} Output of Reference Buffer** See *1.25V Internal Reference Trim* for details on this field.
15	-	RO	0	**Reserved**
14:8	vrefm	R	*	**Trim Code for V_{REFM} Output of Reference Buffer** See *1.25V Internal Reference Trim* for details on this field.
7	-	RO	0	**Reserved**
6:0	vrefp	R	*	**Trim Code for V_{REFP} Output of Reference Buffer** See *1.25V Internal Reference Trim* for details on this field.

Table 4-65: ADC 2.048V Reference Trim Register

ADC 2.048V Reference Trim			FCR_ADCREFTRIM1	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	R	0	**Reserved**
29:24	vx2_tune		*	**Tuning Capacitor In-Line DAC** See *2.048V Internal Reference Trim* for details on this field.
23:18	-	RO	0	**Reserved**
17:16	vcm	R	*	**Trim Code for V_{CM} Output of Reference Buffer** See *2.048V Internal Reference Trim* for details on this field.
15	-	RO	0	**Reserved**
14:8	vrefm	R	*	**Trim Code for V_{REFM} Output of Reference Buffer** See *2.048V Internal Reference Trim* for details on this field.
7	-	RO	0	**Reserved**
6:0	vrefp	R	*	**Trim Code for V_{REFP} Output of Reference Buffer** See *2.048V Internal Reference Trim* for details on this field.

Table 4-66: ADC External Reference Trim Register

ADC External Reference Trim			FCR_ADCREFTRIM2	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	RO	0	**Reserved**
29:24	vx2_tune	R	*	**Tuning Capacitor In-Line DAC** See *External Reference Trim* for details on this field.
23:18	-	RO	0	**Reserved**
17:16	vcm	R	*	**Trim Code for V_{CM} Output of Reference Buffer** See *External Reference Trim* for details on this field.
15	-	RO	0	**Reserved**
12	iboost_2p048	R	0	**Extra Drive Current Enable for 2.048V Reference** See *2.048V Internal Reference Trim* for details on this field.
11:8	idrv_2p048	R	*	**Trim Code for 2.048V Reference Buffer Drive Strength** See *2.048V Internal Reference Trim* for details on this field.
7:5	-	RO	0	**Reserved**

ADC External Reference Trim			FCR_ADCREFTRIM2	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
4	iboost_1p25	R	0	**Extra Drive Current Enable for 1.25V Reference** See *1.25V Internal Reference Trim* for details on this field.
3:0	idrv_1p25	R	*	**Trim Code for 1.25V Reference Buffer Drive Strength** See *1.25V Internal Reference Trim* for details on this field.

Table 4-67: ERFO Kick Start Register

ERFO Kick Start			FCR_ERFOKS	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:14	-	R/W	0	**Reserved**
13:12	ksclksel	R/W	0	**Kick Start Clock Select for ERFO** Set this field to the clock source to use for kick starting the ERFO. 0: No kick start clock 1: Reserved 2: Reserved 3: IPO
11	kserfo2x	R/W	0	**Kick Start ERFO Double Pulse Length** Setting this field to 1 enables double pulse length for the kick start pulses. 0: Disabled 1: Enabled
10:8	kserfodriver	R/W	0	**Kick Start ERFO Drive Strength** This field controls the drive strength of the kick start pulses. 0: Disabled 1 – 7: Drive strength selection.
7	kserfo_en	R/W	0	**Kick Start ERFO Enable** Set this field to 1 to start the kick start of the ERFO. 0: Disabled 1: Enabled
6:0	kserfo_cnt	R/W	0	**Kick Start ERFO Counter** Set this value to the number of kick start counts required to improve the start time for the ERFO. See *ERFO Kick Start* for details.

5. Interrupts and Exceptions

Interrupts and exceptions are managed by the Arm Cortex-M4 with FPU nested vector interrupt controller (NVIC). The NVIC manages the interrupts, exceptions, priorities, and masking. *Table 5-1* details the MAX32662's interrupt vector table for the CM4 and describes each exception and interrupt.

5.1 Interrupt and Exception Features

- Eight 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 MAX32662's CM4 core. There are 108 interrupt entries for the MAX32662, including reserved interrupt placeholders. Including the 15 system exceptions for the Arm Cortex-M4 with FPU, the total number of entries is 127.

Table 5-1: MAX32662 CM4 Interrupt Vector Table

Exception (Interrupt) Number	Offset	Name	Description
1	[0x0004]	Reset_IRQn	Reset
2	[0x0008]	NonMaskableInt_IRQn	Non-Maskable Interrupt
3	[0x000C]	HardFault_IRQn	Hard Fault
4	[0x0010]	MemoryManagement_IRQn	Memory Management Fault
5	[0x0014]	BusFault_IRQn	Bus Fault
6	[0x0018]	UsageFault_IRQn	Usage Fault
7:10	[0x001C]-[0x0028]	-	Reserved
11	[0x002C]	SVCall_IRQn	Supervisor Call Exception
12	[0x0030]	DebugMonitor_IRQn	Debug Monitor Exception
13	[0x0034]	-	Reserved
14	[0x0038]	PendSV_IRQn	Request Pending for System Service
15	[0x003C]	SysTick_IRQn	System Tick Timer
16	[0x0040]	PF_IRQn	Power Fail interrupt
17	[0x0044]	WDT_IRQn	Windowed Watchdog Timer 0 Interrupt
18	[0x0048]	-	Reserved
19	[0x004C]	RTC_IRQn	Real-time Clock Interrupt
20	[0x0050]	TRNG_IRQn	True Random Number Generator Interrupt
21	[0x0054]	TMR0_IRQn	Timer 0 Interrupt
22	[0x0058]	TMR1_IRQn	Timer 1 Interrupt
23	[0x005C]	TMR2_IRQn	Timer 2 Interrupt
24	[0x0060]	TMR3_IRQn	Timer 3 Interrupt
25:28	[0x0064]:[0x0070]	-	Reserved
29	[0x0074]	I2C0_IRQn	I^2C Port 0 Interrupt
30	[0x0078]	UART0_IRQn	UART Port 0 Interrupt

Exception (Interrupt) Number	Offset	Name	Description
31	[0x007C]	UART1_IRQn	UART Port 1 Interrupt
32	[0x0080]	SPI0_IRQn	SPI Port 0 Interrupt
33	[0x0084]	SPI1_IRQn	SPI Port 1 Interrupt
34:35	[0x0088]:[0x008C]	-	Reserved
36	[0x0090]	ADC_IRQn	ADC Interrupt
37:38	[0x0094]:[0x0098]	-	Reserved
39	[0x009C]	FLC_IRQn	Flash Controller 0 Interrupt
40	[0x00A0]	GPIO0_IRQn	GPIO Port 0 Interrupt
41:43	[0x00A4]:[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
48:51	[0x00C0]:[0x00CC]	-	Reserved
52	[0x00D0]	I2C1_IRQn	I^2C Port 1 Interrupt
53:69	[0x00D4]:[0x0114]	-	Reserved
70	[0x0118]	GPIOWAKE_IRQn	GPIO Wake-up Interrupt
71	[0x011C]:[0x0128]	-	Reserved
75	[0x012C]	PT_IRQn	Pulse Train Interrupt
76:97	[0x0130]:[0x0184]	-	Reserved
98	[0x0188]	ECC_IRQn	Error Correction Coding Block Interrupt (Reserved)
99:112	[0x018C]:[0x01C0]	-	Reserved
113	[0x01C4]	AES_IRQn	AES Interrupt
114	[0x01C8]	-	Reserved
115	[0x01CC]	I2S_IRQn	I^2S Interrupt
116:122	[0x01D0]:[0x01E8]	-	Reserved
123	[0x01EC]	CAN_IRQn	CAN 0 Interrupt
124:127	[0x01F0]:[0x01FC]	-	Reserved

6. Debug Access Port (DAP)

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

6.1 Instances

The DAP interface communicates through the serial wire debug (SWD) interface signals, as shown in *Table 6-1*.

Table 6-1: MAX32662 DAP

Pin	Alternate Function	SWD Signal
P0.0	AF1	SWDIO
P0.1	AF1	SWDCLK

6.2 Access Control

The DAP is enabled after every POR to allow debugging during development. The interface can be disabled in software by setting the *GCR_SYSCTRL*.*swd_dis* field to 1. The *GCR_SYSCTRL*.*swd_dis* field clears to 0 again, re-enabling the DAP after a POR. Parts with a customer-accessible DAP should disable the DAP in a final customer product.

6.2.1 Locking the DAP

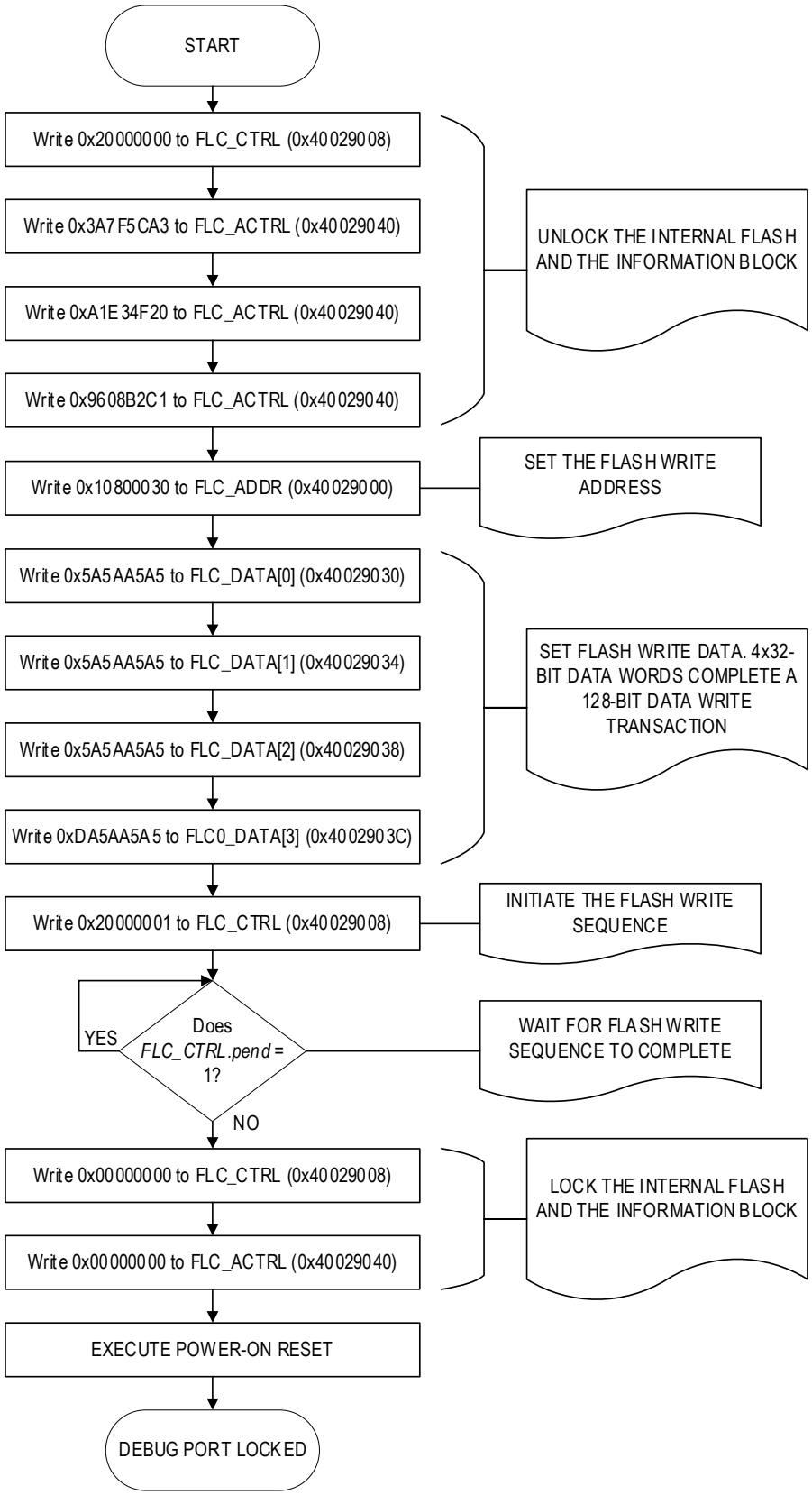
There are two options for locking out the debug access port. Option 1 locks the DAP and makes it available to be unlocked later. This is a one-time-only process. The DAP port cannot be relocked. Option 2 locks the DAP permanently.

6.2.1.1 Option 1

To lock the DAP and make it available to be unlocked later (one time only), follow the flow chart in *Figure 6-1*.

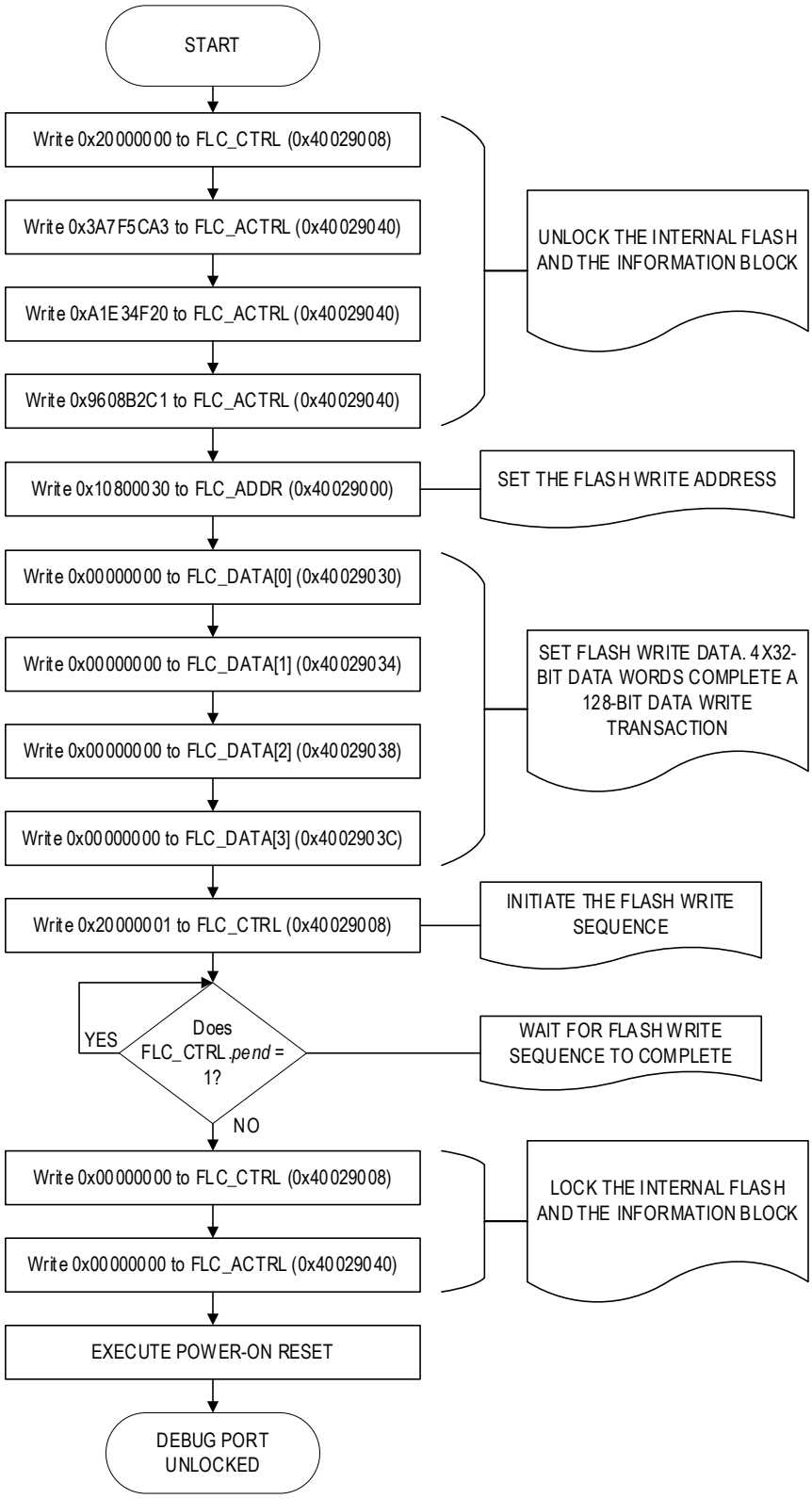
CoreSight is a registered trademark of Arm Limited.

Figure 6-1: Locking the DAP to Make it Available for Unlock Later



To unlock the DAP after it has been locked using the flow chart of *Figure 6-1*, follow the flow chart in *Figure 6-2*.

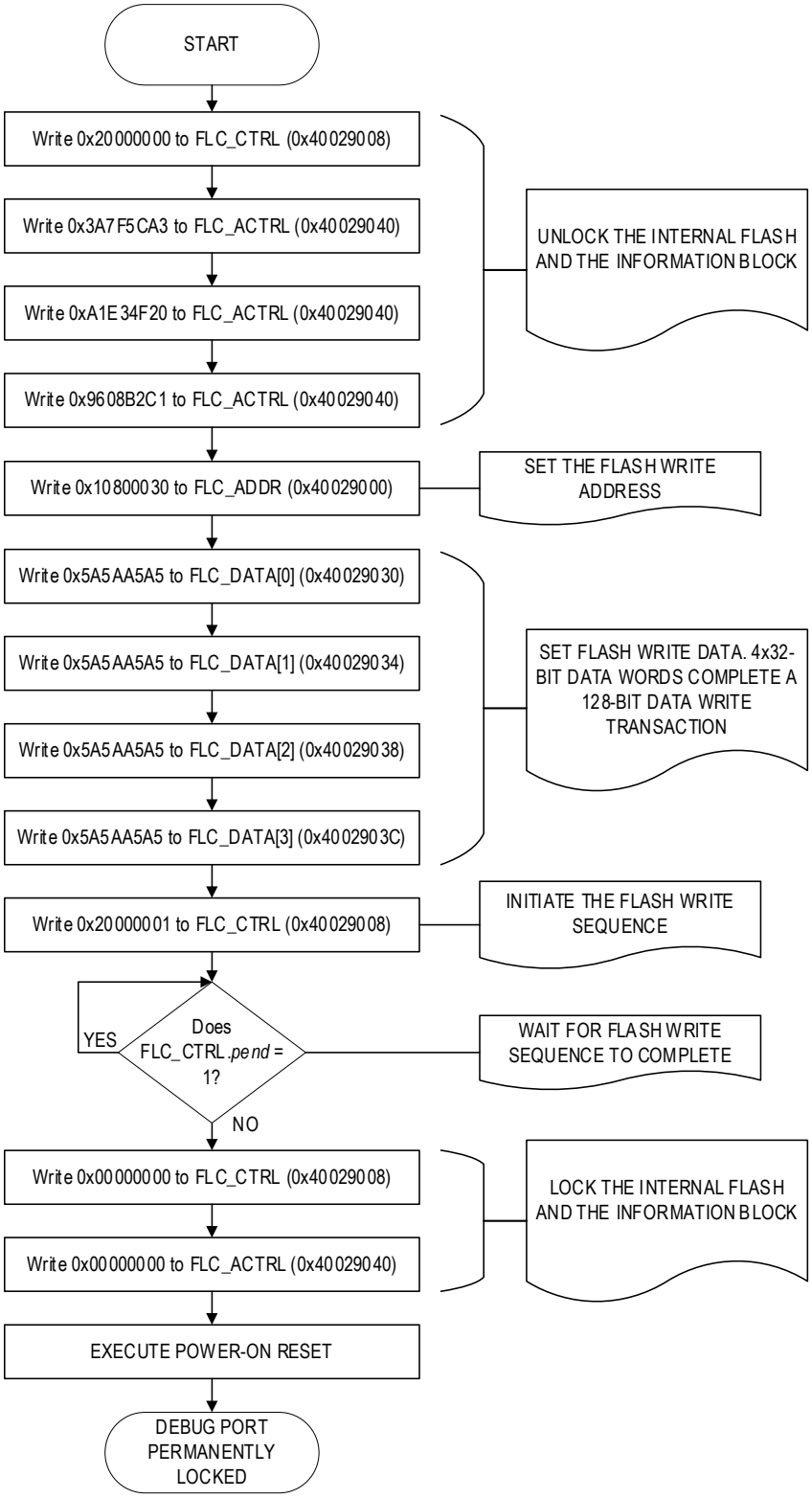
Figure 6-2: Unlocking the DAP After Being Locked as in Figure 6-1



6.2.1.2 Option 2

To lock the DAP permanently, follow the flow chart in *Figure 6-3*.

Figure 6-3: Locking the Debug Access Port Permanently



6.3 Pin Configuration

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

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

7. Flash Controller (FLC)

The MAX32662 flash controller manages read, write, and erase access to the internal flash memory, and provides the following features:

- Up to 256KB internal flash memory.
  - 32 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 and read protection.

7.1 Instances

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

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

Table 7-1: MAX32662 Internal Flash Memory Organization

Instance Number	Page Number	Size (per page)	Start Address	End Address
FLC	0	8,192 bytes	0x1000 0000	0x1000 1FFF
	1	8,192 bytes	0x1000 2000	0x1000 3FFF
	2	8,192 bytes	0x1000 4000	0x1000 5FFF
	3	8,192 bytes	0x1000 6000	0x1000 6FFF
	…	…	…	…
	30	8,192 bytes	0x1003 C000	0x1003 DFFF
	31	8,192 bytes	0x1003 E000	0x1003 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.

7.2.1 Clock Configuration

The flash controller requires a 1MHz internal clock. Use the flash controller clock divisor to generate $f_{FLCn\_CLK} = 1\text{MHz}$, as shown in *Equation 7-1*. If using the IPO as the system clock, set the *FLC_CLKDIV.clkdiv* to 100 (0x64).

Equation 7-1: Flash Controller Clock Frequency

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

7.2.2 Lock Protection

A locking mechanism prevents accidental memory writes and erases. All write and erase operations require the *FLC_CTRL.unlock* field to be set to 2 before starting the operation. Writing any other value to this field, *FLC_CTRL.unlock*, locks the flash controller.

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

7.2.3 Flash Write Width

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

7.2.4 Flash Write

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

1. If desired, enable the flash controller interrupts by setting the *FLC_INTR.af_ie* and *FLC_INTR.done_ie* bits.
2. Read the *FLC_CTRL.pend* bit until it returns 0.
3. Configure the *FLC_CLKDIV.clkdiv* field to achieve a 1MHz frequency based on the selected SYS_CLK frequency.
4. Set the *FLC_ADDR* register to a valid target address.
5. Set *FLC_DATA[3]*, *FLC_DATA[2]*, *FLC_DATA[1]*, and *FLC_DATA[0]* to the data to write.
    a. *FLC_DATA[3]* is the most significant word, and *FLC_DATA[0]* is the least significant word.
        i. Each word of the data to write follows the little-endian format, where the least significant byte of the word is stored at the lowest-numbered byte, and the most significant byte is stored at the highest-numbered byte.
6. Set the *FLC_CTRL.unlock* field to 2 to unlock the flash.
7. Set the *FLC_CTRL.wr* field to 1.
    a. The hardware automatically clears this field when the write operation is complete.
8. The *FLC_INTR.done_if* field is set to 1 by the hardware when the write completes. An interrupt is generated if the *FLC_INTR.done_ie* field is set to 1.
    a. If an error occurred, the *FLC_INTR.af_if* field is set to 1 by the hardware, and an interrupt is generated if the *FLC_INTR.af_ie* field is set to 1.
9. Set the *FLC_CTRL.unlock* field to any value other than 2 to relock the flash.

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

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

7.2.5 Page Erase

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

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

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

7.2.6 Mass Erase

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

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

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

7.3 Flash Controller Registers

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

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

Table 7-2: Flash Controller Register Summary

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

Offset	Register Name	Description
[0x0008]	*FLC_CTRL*	Flash Controller Control Register
[0x0024]	*FLC_INTR*	Flash Controller Interrupt Register
[0x0030]	*FLC_DATA[0]*	Flash Controller Data Register 0
[0x0034]	*FLC_DATA[1]*	Flash Controller Data Register 1
[0x0038]	*FLC_DATA[2]*	Flash Controller Data Register 2
[0x003C]	*FLC_DATA[3]*	Flash Controller Data Register 3
[0x0040]	*FLC_ACTRL*	Flash Controller Access Control Register
[0x0080]	*FLC_WELR0*	Flash Write/Erase Lock 0 Register
[0x0084]	*FLC_RLR0*	Flash Read Lock 0 Register

7.3.1 Register Details

Table 7-3: Flash Controller Address Pointer Register

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

Table 7-4: Flash Controller Clock Divisor Register

Flash Controller Clock Divisor		FLC_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 flash controller peripheral clock, f_{FLC_CLK}. The flash controller peripheral clock must equal 1MHz. The default on POR, system reset, and watchdog reset is 100, resulting in f_{FLC_CLK} = 1MHz when IPO is the system oscillator. The flash controller peripheral clock is only used during erase and program functions, and not during read functions. See *Clock Configuration* for additional details.

Table 7-5: Flash Controller Control Register

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

Flash Controller Control			FLC_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
24	pend	RO	0	**Flash Busy Flag** When this field is set, writes to all of the flash registers except the *FLC_INTR* register are ignored by the flash controller. This bit is cleared by hardware once the flash is accessible. *Note: If the flash controller is busy (FLC_CTRL.pend = 1), reads, writes, and erase operations are not allowed and result in an access failure (FLC_INTR.af = 1).* 0: Flash idle. 1: Flash busy.
23:16	-	RO	0	**Reserved**
15:8	erase_code	R/W	0	**Erase Code** Before an erase operation, this field must be set to 0x55 for a page erase or 0xAA for a mass erase. Unlock the flash controller before setting the erase code. This field is automatically cleared after the erase operation is complete. 0x00: Erase disabled. 0x55: Page erase code. 0xAA: Enable mass erase.
7:5	-	RO	0	**Reserved**
4	width	R/W	0	**Width** Do not modify this field.
3	-	RO	0	**Reserved**
2	pge	R/W1O	0	**Page Erase** Write a 1 to this field to initiate a page erase at the address in *FLC_ADDR.addr*. Unlock the flash before attempting a page erase. See *FLC_CTRL.unlock* for details. The flash controller hardware clears this bit when a page erase operation is complete. 0: Normal operation. 1: Write a 1 to initiate a page erase. If this field reads 1, a page erase operation is in progress.
1	me	R/W1O	0	**Mass Erase** Write a 1 to this field to initiate a mass erase of the internal flash memory. Unlock the flash before attempting a mass erase. See *FLC_CTRL.unlock* for details. The flash controller hardware clears this bit when the mass erase operation completes. 0: Normal 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 *FLC_ADDR* register. 0: Normal operation. 1: Write 1 to initiate a write operation. If this field reads 1, a write operation is in progress. *Note: This field is protected and cannot be set to 0 by application code.*

Table 7-6: Flash Controller Interrupt Register

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

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

Table 7-7: Flash Controller Data 0 Register

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

Table 7-8: Flash Controller Data Register 1

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

Table 7-9: Flash Controller Data Register 2

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

Table 7-10: Flash Controller Data Register 3

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

Table 7-11: Flash Controller Access Control Register

Flash Controller Access Control			FLC_ACTRL	[0x0040]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	actrl	R/W	0	**Access Control** Access the information block by writing the access control sequence.

Table 7-12: Flash Write/Lock 0 Register

Flash Write/Lock 0			FLC_WELR0	[0x0080]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	welr0	R/W1C	0xFFFF FFFF	**Flash Write/Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 0 of the flash, and bit 31 maps to page 31. Write a 1 to a bit position in this register, and the corresponding page of flash is immediately locked. The page protection can only be unlocked by an external reset or a POR. 0: The corresponding page of flash is write protected. 1: The corresponding page of flash is *not* write protected.

Table 7-13: Flash Read Lock 0 Register

Flash Read Lock 0			FLC_RLR0	[0x0084]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	rlr0	R/W1C	0xFFFF FFFF	**Read Lock Bit** Each bit in this register maps to a page of the internal flash. Bit 0 maps to page 0 of the flash, and bit 31 maps to page 31. Write a 1 to a bit position in this register, and the corresponding page of flash is immediately read protected. The page's read protection can only be unlocked by an external reset or a POR. 0: The corresponding flash page is read protected. 1: The corresponding flash page is *not* read protected.

8. 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. The application software must manage the dynamic multiplexing between AF1, AF2, AF3, AF4, AF5, and I/O mode, 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 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 microcontrollers; however, the behavior of several registers varies depending on the specific device. The behavior of the registers should not be assumed to be the same from one device to a different device. Specifically the registers GPIOn_PADCTRL0, GPIOn_PADCTRL1, GPIOn_HYSEN, GPIOn_SRSEL, GPIOn_DS0, GPIOn_DS1, and GPIOn_VSSEL are device-dependent in their usage

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

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

8.1 Instances

Table 8-1 shows the number of GPIO available on each IC package. Some packages and part numbers do not implement all bits of a 32-bit GPIO port. Register fields corresponding to unimplemented GPIO contain indeterminate values and should not be modified.

Table 8-1: GPIO Pin Count

Package	GPIO	Pins
32-TQFN	GPIO0[0:21]	Refer to the device data sheet for details.

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

Note: MAX32662 does not support the selectable GPIO voltage supply feature.

8.2 Configuration

8.2.1 Power-On-Reset Configuration

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

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

8.2.2 Input Mode Configuration

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

1. Set the pin for I/O mode:
   a. *GPIOn_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^2C 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. Enable the pin for input by setting *GPIOn_INEN*.*[pin]* to 1.
5. Read the input state of the pin using the *GPIOn_IN*.*[pin]* field.

A summary of the configuration of the input mode is shown in *Table 8-2*.

Table 8-2: MAX32662 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.

8.2.3 Output Mode Configuration

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

1. Set the pin for I/O mode:
   a. *GPIOn_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.

8.2.4 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 the SWD disable field to 1 (GCR_SYSCTRL.swd_dis = 1).

8.2.5 GPIO Drive Strength

Each I/O pin supports multiple selections for drive strength. Standard GPIO pins are configured for the supported modes using the *GPIOn_DS1* and *GPIOn_DS0* registers, as shown in *Table 8-3*. Each bit 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 8-3: 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 8-4* shows the drive strength setting options. Refer to the device data sheet's alternate function table to determine which GPIO pins support I^2C.

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

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

8.3 Alternate Function Configuration

Table 8-6 shows the AF selection matrix. Write the *GPIOn_EN0*, *GPIOn_EN1*, *GPIOn_EN2*, and *GPIOn_EN3* fields as shown in the table to select the desired AF. Each AF is independently selectable. Mixing functions assigned to AF1, AF2, AF3, or AF4 is supported if all the peripheral's required functions are enabled.

Table 8-5: GPIO Mode and AF Selection

GPIO MODE	*GPIOn_EN3.[pin]*	*GPIOn_EN2.[pin]*	*GPIOn_EN1.[pin]*	*GPIOn_EN0.[pin]*
I/O	0	0	0	1
AF1	0	0	0	0
AF2	0	0	1	0
AF3	0	1	0	0
AF4	0	1	1	0
AF5	1	0	0	0

Table 8-6: GPIO Mode and AF Transition Selection

GPIO MODE	*GPIOn_EN3.[pin]*	*GPIOn_EN2.[pin]*	*GPIOn_EN1.[pin]*	*GPIOn_EN0.[pin]*
I/O (transition to AF1)	0	0	0	1
I/O (transition to AF2)	0	0	1	1
I/O (transition to AF3)	0	1	0	1
I/O (transition to AF4)	0	1	1	1
I/O (transition to AF5)	1	0	0	1

Most GPIO support one or more alternate functions selected with the GPIO configuration enable bits shown in *Table 8-6*. The bits that select the AF must only be changed while the pin is in one of the I/O modes (*GPIOn_EN0* = 1). The specific I/O mode must match the desired AF. For example, if a transition to AF1 is desired, select the setting corresponding to I/O (transition to AF1). Then enable the desired mode by selecting the AF1 mode.

1. Set the GPIO configuration enable bits shown in *Table 8-6* to the I/O mode corresponding to the desired new AF setting. For example, select "I/O (transition to AF1)" if switching to AF1. Switching between different I/O mode settings does not affect the state or electrical characteristics of the pin.
2. Configure the electrical characteristics of the pin. See *Table 8-2* if the assigned alternate function uses the pin as an input. See *Table 8-3* if the assigned alternate function uses the pin as an output.
3. Set the GPIO configuration enable bits shown in *Table 8-5* to the desired alternate function.

Table 8-7: GPIO AF Configuration Reference

Device Pin	AF Configuration Bits			
P0.0	*GPIOn_EN3.[0]*	*GPIOn_EN2.[0]*	*GPIOn_EN1.[0]*	*GPIOn_EN0.[0]*
P0.1	*GPIOn_EN3.[1]*	*GPIOn_EN2.[1]*	*GPIOn_EN1.[1]*	*GPIOn_EN0.[1]*
…	…	…	…	…
P0.29	*GPIOn_EN3.[29]*	*GPIOn_EN2.[29]*	*GPIOn_EN1.[29]*	*GPIOn_EN0.[29]*
P0.30	*GPIOn_EN3.[30]*	*GPIOn_EN2.[30]*	*GPIOn_EN1.[30]*	*GPIOn_EN0.[30]*

8.4 Configuring GPIO (External) Interrupts

Each GPIO supports external interrupt events when the GPIO is configured for I/O mode and the input mode is enabled. The interrupts are peripheral controlled if the GPIO is configured as a peripheral AF. GPIO interrupts can be independently enabled for any number of GPIO on each GPIO port. All implemented pins of a GPIO port have a single assigned/shared interrupt vector.

The following procedure details the steps for enabling GPIO 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.
      i. *GPIOn_INTPOL.[pin]* = 1: Input high triggers interrupt.
      ii. *GPIOn_INTPOL.[pin]* = 0: Input low triggers interrupt.
   b. For edge-triggered interrupts, the interrupt triggers on an edge event.
      i. *GPIOn_INTPOL.[pin]* = 0: Input rising edge triggers interrupt.
      ii. *GPIOn_INTPOL.[pin]* = 1: Input falling edge triggers interrupt.
   c. Optionally set *GPIOn_DUALEDGE.[pin]* to 1 to trigger on both the rising and falling edges of the input signal.
4. Set *GPIOn_WKEN.[pin]* to 1 to enable wake from SLEEP and DEEPSLEEP if desired.
5. Set *GPIOn_INTEN.[pin]* to 1 to enable the interrupt for the pin.

8.4.1 GPIO Interrupt Handling

Each GPIO port is assigned its own dedicated interrupt vector, as shown in *Table 8-9*. Complete the following steps to handle GPIO interrupts using a software interrupt vector handler:

1. Read the *GPIOn_INTFL* register to determine the GPIO pin that triggered the interrupt.
2. Complete interrupt tasks associated with the interrupt source pin as required by the application.
3. Clear the interrupt flag in the *GPIOn_INTFL* register by writing 1 to the *GPIOn_INTFL_CLR* bit position that triggered the interrupt. If multiple bits are set in the *GPIOn_INTFL* register, all of the bits should be cleared.
4. Return from the interrupt vector handler.

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

Table 8-8: MAX32662 GPIO Interrupt Enable Settings for Each Supported Operating Mode

Operating Mode	GPIOn_WKEN	GPIOn_INTEN	Interrupt Handler
ACTIVE	N/A	X	GPIO0_IRQn
SLEEP	X	X	GPIO0_IRQn
DEEPSLEEP	X	X	GPIO0_IRQn
Note: Wake from BACKUP is only supported using the GPIOWAKE interrupt.			

Table 8-9: MAX32662 GPIO Port Interrupt Vector Mapping

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

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

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

8.4.2.1 Using GPIOWAKE for Wake-Up from All Power Modes

For wake-up interrupts on the GPIO, a single interrupt vector, GPIOWAKE_IRQn, is assigned for all the GPIO pins. When the wake-up event occurs, the application software must interrogate the *PWRSEQ_LPWKFL0* register to determine which GPIO0 port pin caused the interrupt.

Table 8-10: GPIO Wakeup Interrupt Vector

GPIO Wake Interrupt Source	GPIO Wake Interrupt Status Register	Device Specific Interrupt Vector Number	GPIO Wake-up Interrupt Vector
GPIO0[31:0]	*GPIOn_INTFL*	40	GPIOWAKE_IRQn

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

1. Clear pending interrupt flags by writing 0xFFFF FFFF to the *PWRSEQ_LPWKFL0* register.
2. Set up a GPIOWAKE_IRQn interrupt handler.
3. Enable the GPIOWAKE for each desired pin by setting *PWRSEQ_LPWKEN0[pin]* to 1.
4. Configure the power manager to use the GPIO as a wake-up source by writing 1 to the *GCR_PM.gpio_we* field.
5. Enter the desired low-power mode. See *Operating Modes* for details.
6. When a wakeup event occurs, if an I/O caused the wake up, the pin's corresponding bit is set in the *PWRSEQ_LPWKFL0* register.

Table 8-11: MAX32662 Operating Mode GPIOWAKE Interrupt Enable Settings

Operating Power Mode	PWRSEQ_LPWKEN0	GCR_PM.gpio_we	Interrupt Handler
ACTIVE	X	X	GPIOWAKE_IRQn
SLEEP	X	X	GPIOWAKE_IRQn
DEEPSLEEP	X	X	GPIOWAKE_IRQn
BACKUP	X	X	GPIOWAKE_IRQn

8.5 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 8-12*. Register names for a specific instance are defined by replacing "n" with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 8-12: GPIO Register Summary

Offset	Register Name	Description
[0x0000]	*GPIOn_EN0*	*GPIO Port n Configuration Enable Bit 0 Register*
[0x0004]	*GPIOn_EN0_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 0 Register*
[0x0008]	*GPIOn_EN0_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 0 Register*
[0x000C]	*GPIOn_OUTEN*	*GPIO Port n Output Enable Register*
[0x0010]	*GPIOn_OUTEN_SET*	*GPIO Port n Output Enable Atomic Set Register*
[0x0014]	*GPIOn_OUTEN_CLR*	*GPIO Port n Output Enable Atomic Clear Register*
[0x0018]	*GPIOn_OUT*	*GPIO Port n Output Register*
[0x001C]	*GPIOn_OUT_SET*	*GPIO Port n Output Atomic Set Register*
[0x0020]	*GPIOn_OUT_CLR*	*GPIO Port n Output Atomic Clear Register*
[0x0024]	*GPIOn_IN*	*GPIO Port n Input Register*
[0x0028]	*GPIOn_INTMODE*	*GPIO Port n Interrupt Mode Register*
[0x002C]	*GPIOn_INTPOL*	*GPIO Port n Interrupt Polarity Register*
[0x0030]	*GPIOn_INEN*	*GPIO Port n Input Enable Register*
[0x0034]	*GPIOn_INTEN*	*GPIO Port n Interrupt Enable Register*
[0x0038]	*GPIOn_INTEN_SET*	*GPIO Port n Interrupt Enable Atomic Set Register*
[0x003C]	*GPIOn_INTEN_CLR*	*GPIO Port n Interrupt Enable Atomic Clear Register*
[0x0040]	*GPIOn_INTFL*	*GPIO Port n Interrupt Status Register*
[0x0048]	*GPIOn_INTFL_CLR*	*GPIO Port n Interrupt Clear Register*
[0x004C]	*GPIOn_WKEN*	*GPIO Port n Wakeup Enable Register*
[0x0050]	*GPIOn_WKEN_SET*	*GPIO Port n Wakeup Enable Atomic Set Register*
[0x0054]	*GPIOn_WKEN_CLR*	*GPIO Port n Wakeup Enable Atomic Clear Register*
[0x005C]	*GPIOn_DUALEDGE*	*GPIO Port n Interrupt Dual Edge Mode Register*
[0x0060]	*GPIOn_PADCTRL0*	*GPIO Port n Pad Control 0 Register*
[0x0064]	*GPIOn_PADCTRL1*	*GPIO Port n Pad Control 1 Register*
[0x0068]	*GPIOn_EN1*	*GPIO Port n Configuration Enable Bit 1 Register*
[0x006C]	*GPIOn_EN1_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 1 Register*
[0x0070]	*GPIOn_EN1_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 1 Register*
[0x0074]	*GPIOn_EN2*	*GPIO Port n Configuration Enable Bit 2 Register*
[0x0078]	*GPIOn_EN2_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 2 Register*
[0x007C]	*GPIOn_EN2_CLR*	*GPIO Port n Configuration Enable Atomic Clear Bit 2 Register*
[0x0080]	*GPIOn_EN3*	*GPIO Port n Configuration Enable Bit 3 Register*
[0x0084]	*GPIOn_EN3_SET*	*GPIO Port n Configuration Enable Atomic Set Bit 3 Register*

Offset	Register Name	Description
[0x0088]	GPIOn_EN3_CLR	GPIO Port n Configuration Enable Atomic Clear Bit 3 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

8.5.1 Register Details

Table 8-13: GPIO AF 0 Select Register

GPIO AF 0 Select			GPIOn_EN0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	1	**GPIO Configuration Enable Bit 0** This field, in conjunction with bits in Table 8-5, configures 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 8-7 depicts a detailed example of how each bit applies to each of the GPIO device pins. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect the input and interrupt functionality of the associated pin.*

Table 8-14: GPIO Port n Configuration Enable Atomic Set Bit 0 Register

GPIO Port n Configuration Enable Atomic Set Bit 0			GPIOn_EN0_SET	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Configuration Enable Atomic Set Bit 0** Setting a bit in this field sets the corresponding bit in the GPIOn_EN0 register. 0: No effect 1: Corresponding bit in GPIOn_EN0 register set to 1 *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-15: GPIO Port n Configuration Enable Atomic Clear Bit 0 Register

GPIO Port n Configuration Enable Atomic Clear Bit 0			GPIOn_EN0_CLR	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Configuration Enable Atomic Clear Bit 0** Setting a bit in this field clears the corresponding bits in the GPIOn_EN0 register. 0: No effect 1: Corresponding bits in GPIOn_EN0 register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-16: GPIO Port n Output Enable Register

GPIO Port n Output Enable			GPIOn_OUTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Output Enable** Setting a bit in this field enables the output driver for the corresponding GPIO pin. A bit can be enabled directly by writing to this register or indirectly through *GPIOn_OUTEN_SET* or *GPIOn_OUTEN_CLR*. 0: Disabled 1: Enabled *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-17: GPIO Port n Output Enable Atomic Set Register

GPIO Port n Output Enable Atomic Set			GPIOn_OUTEN_SET	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Output Enable Atomic Set** Setting a bit in this field sets the corresponding bit in the *GPIOn_OUTEN* register. 0: No effect 1: Corresponding bits in *GPIOn_OUTEN* set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-18: GPIO Port n Output Enable Atomic Clear Register

GPIO Port n Output Enable Atomic Clear			GPIOn_OUTEN_CLR	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Output Enable Atomic Clear** Setting a bit in this field sets the corresponding bits in the *GPIOn_OUTEN* register. 0: No effect 1: Corresponding bits in *GPIOn_OUTEN* cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-19: GPIO Port n Output Register

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

Table 8-20: GPIO Port n Output Atomic Set Register

GPIO Port n Output Atomic Set			GPIOn_OUT_SET	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Output Atomic Set** Setting a bit in this field sets the corresponding bits in the *GPIOn_OUT* register. 0: No effect 1: Corresponding bits in *GPIOn_OUTEN* set to 1 *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-21: GPIO Port n Output Atomic Clear Register

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

Table 8-22: GPIO Port n Input Register

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

Table 8-23: GPIO Port n Interrupt Mode Register

GPIO Port n Interrupt Mode			GPIOn_INTMODE	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Interrupt Mode** Setting a bit in this field sets edge-triggered interrupts for corresponding GPIO pin. Clearing a bit in this field sets level-triggered interrupt for corresponding GPIO pin. 0: Level triggered interrupt. 1: Edge triggered interrupt. *Note: This bit has no effect unless the corresponding bit in the GPIOn_INEN register is set.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-24: GPIO Port n Interrupt Polarity Register

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

Table 8-25: GPIO Port n Input Enable Register

GPIO Port n Input Enable			GPIOn_INEN	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**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 8-26: GPIO Port n Interrupt Enable Registers

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

Table 8-27: GPIO Port n Interrupt Enable Atomic Set Register

GPIO Port Interrupt Enable Atomic Set			GPIOn_INTEN_SET	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Interrupt Enable Atomic Set** Setting a bit in this field sets the corresponding bits in the *GPIOn_INTEN* register. 0: No effect. 1: Corresponding bits in *GPIOn_INTEN* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-28: GPIO Port n Interrupt Enable Atomic Clear Register

GPIO Port Interrupt Enable Atomic Clear			GPIOn_INTEN_CLR	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Interrupt Enable Atomic Clear** Setting a bit in this field clears the corresponding bits in the *GPIOn_INTEN* register. 0: No effect. 1: Corresponding bits in *GPIOn_INTEN* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-29: GPIO Interrupt Status Register

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

Table 8-30: GPIO Port n Interrupt Clear Register

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

Table 8-31: GPIO Port n Wakeup Enable Register

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

Table 8-32: GPIO Port n Wakeup Enable Atomic Set Register

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

Table 8-33: GPIO Port n Wakeup Enable Atomic Clear Register

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

Table 8-34: GPIO Port n Interrupt Dual Edge Mode Register

GPIO Port n Interrupt Dual Edge Mode			GPIOn_DUALEDGE	[0x005C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Interrupt Dual-Edge Mode Select** Setting a bit in this field selects dual-edge mode triggered interrupts (rising and falling edge triggered) on the corresponding GPIO port device pin. The associated *GPIOn_INTMODE* bit must be set to edge-triggered. When enabled, the associated polarity (*GPIOn_INTPOL*) setting has no effect. 0: Disabled. 1: Enabled. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-35: GPIO Port n Pad Control 0 Register

GPIO Port n Pad Control 0			GPIOn_PADCTRL0	[0x0060]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO0 Pull Up/Pull Down Enable** Setting a bit in this field enables either the weak pullup or weak pulldown resistor on the corresponding GPIO port device pin. The selection for pullup or pulldown resistor is set using the *GPIOn_PS* register. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: GPIO with I^2C as an AF do not support a weak pullup resistor. Refer to the symbols V_{OL_I2C} and V_{OH_I2C} in the device data sheet electrical characteristics table for details regarding which I/O pins support I^2C functionality. If the corresponding GPIO with I^2C as an AF bit in GPIOn_PS is set to 1, setting the same bit in this register has no effect.*

Table 8-36: GPIO Port n Pad Control 1 Register

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

Table 8-37: GPIO Port n Configuration Enable Bit 1 Register

GPIO Port n Configuration Enable Bit 1			GPIOn_EN1	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO AF 1 Mode Select** These bits, in conjunction with bits in *Table 8-5*, 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 8-7* depicts a detailed example of how each of these bits applies to each of the GPIO device pins. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect the input and the interrupt functionality of the associated pin.*

Table 8-38: GPIO Port n Configuration Enable Atomic Set Bit 1 Register

GPIO Port n Configuration Enable Atomic Set Bit 1			GPIOn_EN1_SET	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO Configuration Enable Atomic Set Bit 1** Setting a bit in this field sets the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-39: GPIO Port n Configuration Enable Atomic Clear Bit 1 Register

GPIO Port n Configuration Enable Atomic Clear Bit 1			GPIOn_EN1_CLR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1C	0	**GPIO Configuration Enable Atomic Clear Bit 1** Setting a bit in this field clears the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-40: GPIO Port n Configuration Enable Bit 2 Register

GPIO Port n Configuration Enable Bit 2			GPIOn_EN2	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Configuration Enable Bit 2** These bits, in conjunction with bits in *Table 8-5*, 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 8-7* depicts a detailed example of how each of these bits applies to each of the GPIO device pins. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect the input and the interrupt functionality of the associated pin.*

Table 8-41: GPIO Port n Configuration Enable Atomic Set Bit 2 Register

GPIO Port n Configuration Enable Atomic Set Bit 2			GPIOn_EN2_SET	[0x0078]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO AF Select Atomic Set Bit 2** Setting a bit in this field sets the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-42: GPIO Port n Configuration Enable Atomic Clear Bit 2 Register

GPIO Port n Configuration Enable Atomic Clear Bit 2			GPIOn_EN2_CLR	[0x007C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1C	0	**GPIO AF Select Atomic Clear Bit 2** Setting a bit in this field clears the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-43: GPIO Port n Configuration Enable Bit 2 Register

GPIO Port n Configuration Enable Bit 2			GPIOn_EN3	[0x0080]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Configuration Enable Bit 2** These bits, in conjunction with bits in *Table 8-5*, 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_EN3_SET* or *GPIOn_EN3_CLR*. *Table 8-7* depicts a detailed example of how each of these bits applies to each of the GPIO device pins. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect the input and the interrupt functionality of the associated pin.*

Table 8-44: GPIO Port n Configuration Enable Atomic Set Bit 3 Register

GPIO Port n Configuration Enable Atomic Set Bit 3			GPIOn_EN3_SET	[0x0084]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO AF Select Atomic Set Bit 3** Setting a bit in this field sets the corresponding bits in the *GPIOn_EN3* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN3* register set to 1. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-45: GPIO Port n Configuration Enable Atomic Clear Bit 3 Register

GPIO Port n Configuration Enable Atomic Clear Bit 3		GPIOn_EN3_CLR		[0x0088]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1O	0	**GPIO AF Select Atomic Clear Bit 3** Setting a bit in this field clears the corresponding bits in the *GPIOn_EN3* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN3* register cleared to 0. *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-46: GPIO Port n Input Hysteresis Enable Register

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

Table 8-47: GPIO Port n Slew Rate Enable Register

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

Table 8-48: GPIO Port n Output Drive Strength Bit 0 Register

GPIO Port n Output Drive Strength Bit 0			GPIOn_DS0	[0x00B0]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**GPIO Drive Strength 0 Select** The output drive strength supports four modes. The mode selection is set using the combination of the *GPIOn_DS1* and *GPIOn_DS0* bits for the associated GPIO pin. See the *GPIO Drive Strength* section for the selection options on these I/O pins. *Note: GPIO with I^2C as an AF only support two different drive strengths:* *0: Low output drive strength selected.* *1: High output drive strength selected.* *Refer to the symbols V_{OL_I2C} and V_{OH_I2C} in the device data sheet electrical characteristics table for details regarding which I/O pins support I^2C functionality.* *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.*

Table 8-49: GPIO Port n Output Drive Strength Bit 1 Register

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

Table 8-50: GPIO Port n Pulldown/Pullup Strength Select Register

GPIO Port n Pulldown/Pullup Strength Select			GPIOn_PS	[0x00B8]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**Pullup/Pulldown Resistor Select** Setting a bit in this field selects a weak pullup resistor for the corresponding GPIO port device pin. Clearing a bit in this field selects weak pulldown resistor for the corresponding GPIO port device pin. The *GPIOn_PADCTRL0* 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.*

Table 8-51: GPIO Port n Voltage Select Register

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

9. Standard DMA (DMA)

The DMA is a peripheral that provides the ability to perform high-speed, block memory transfers of data independent of a CPU. All DMA transactions consist of a burst read from the source into the internal DMA FIFO, followed by a burst write from the internal DMA FIFO to the destination.

DMA transfers are one of three types:

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

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

- Complete 32-bit source and destination address with 24-bit (16MB) address increment capability
- Ability to chain DMA buffers when a count-to-zero (CTZ) condition occurs
- Up to 16MB 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, referred to as DMA. The DMA provides 16 channels, generically referred to as DMA_CHn. The DMA includes a set of interrupt registers common to all of its channels and a set of registers unique to each channel instance.

Table 9-1: MAX32662 DMA and Channel Instances

DMA Instance	DMA_CH*n* Channel Instance
DMA	DMA_CH0
	DMA_CH1
	DMA_CH2
	DMA_CH3

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 disabled. In general, *DMA_CHn_STATUS.status* follows the setting of the *DMA_CHn_CTRL.en* bit. However, the *DMA_CHn_STATUS.status* bit is automatically cleared under the following conditions:

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

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

Only an error condition can interrupt an ongoing data transfer.

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/or destination memory addresses, depending on the specific operation.

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

Table 9-2: MAX32662 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	UART1	UART1 Receive FIFO	*DMA_CHn_DST*
0x06	CAN	CAN Receive FIFO	*DMA_CHn_DST*
0x07	I2C0	I2C0 Receive FIFO	*DMA_CHn_DST*
0x08	I2C1	I2C1 Receive FIFO	*DMA_CHn_DST*
0x09	ADC	ADC FIFO	*DMA_CHn_DST*
0x0A:0x1D	I2C2	I2C2 Receive FIFO	*DMA_CHn_DST*
0x1E	I^2S	I^2S Receive FIFO	*DMA_CHn_DST*
0x1F:0x20	Reserved	-	-
0x21	SPI0	*DMA_CHn_SRC*	SPI0 Transmit FIFO
0x22	SPI1	*DMA_CHn_SRC*	SPI1 Transmit FIFO
0x23	-	-	-
0x24	UART0	*DMA_CHn_SRC*	UART0 Transmit FIFO
0x25	UART1	*DMA_CHn_SRC*	UART1 Transmit FIFO
0x26	CAN	*DMA_CHn_SRC*	CAN Transmit FIFO
0x27	I2C0	*DMA_CHn_SRC*	I2C0 Transmit FIFO
0x28	I2C1	*DMA_CHn_SRC*	I2C1 Transmit FIFO

DMA_CHn_CTRL.request	Peripheral	DMA Source	DMA Destination
0x29:0x3D	Reserved	-	-
0x3E	I²S	DMA_CHn_SRC	I²S Transmit FIFO
0x3F	-	-	-

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 to 32)	This maximum number of bytes moved during the burst read.
DMA_CHn_CTRL.srcwd	Source width	This field determines the maximum data width used during each read of the AHB burst (byte, two bytes, or four bytes). The actual AHB width might be less if DMA_CHn_CNT is not great enough to supply all the needed bytes.
DMA_CHn_CTRL.srcinc	Source increment enable	Increments DMA_CHn_SRC. This field is ignored when the DMA source is a peripheral.

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 to 32)	The maximum number of bytes moved during a single AHB read/write burst.
DMA_CHn_CTRL.dstwd	Destination width	This field determines the maximum data width used during each write of the AHB burst (one byte, two bytes, or four bytes).
DMA_CHn_CTRL.dstinc	Destination increment enable	Increments DMA_CHn_DST. This field is ignored when the DMA destination is a peripheral.

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

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

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

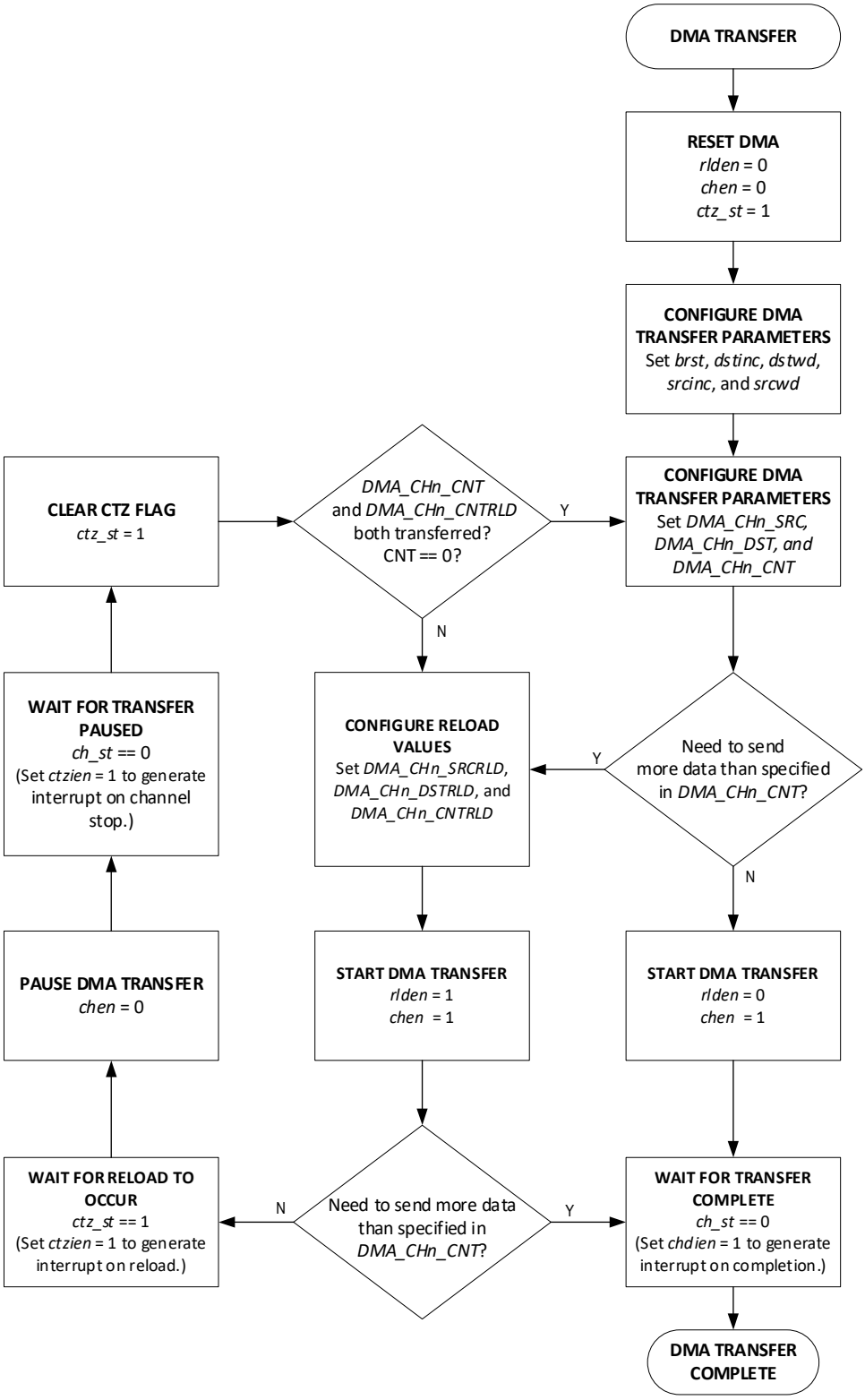
9.5 Chaining Buffers

Chaining buffers reduces the DMA interrupt response time and allows the DMA to service requests without intermediate processing from the CPU. *Figure 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 or write burst, do not write to the *DMA_CHn_SRC*, *DMA_CHn_DST*, or *DMA_CHn_CNT* registers while a channel is active (*DMA_CHn_STATUS.status* = 1). To disable any DMA channel, clear the *DMA_INTEN.ch<n>* bit. Then, poll the *DMA_CHn_STATUS.status* bit to verify that the channel is disabled.

Figure 9-1: DMA Block-Chaining Flowchart



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 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 Detection

Each channel can optionally generate an interrupt when the associated peripheral does not request a transfer in a user-configurable period. When the timeout start conditions are met, an internal 10-bit counter begins incrementing at a frequency determined by the AHB clock, *DMA_CHn_CTRL.to_clkdiv*, and *DMA_CHn_CTRL.to_per* shown in *Table 9-5*

Table 9-5: DMA Channel Timeout Configuration

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

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

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

A channel timeout event is generated if the timer is not reset by one of the events listed below before the timeout period expires. The timer is reset whenever:

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

If the timeout timer period expires, the hardware sets *DMA_CHn_STATUS.to_if* = 1 to indicate a channel timeout event 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 permanently active. The DMA channel generates an almost constant request for the bus until its transfer is complete. For this reason, assign a lower priority to channels executing memory-to-memory transfers to prevent starvation of other DMA channels.

9.9 DMA Registers

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

Table 9-6: DMA Register Summary

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

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

9.10 DMA Channel Register Summary

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

Offset	DMA Channel	Description
[0x0100]	DMA_CH0	DMA Channel 0
[0x0120]	DMA_CH1	DMA Channel 1
[0x0140]	DMA_CH2	DMA Channel 2
[0x0160]	DMA_CH3	DMA Channel 3

9.11 DMA Channel Registers

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

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

Table 9-10: DMA Channel Registers Summary

Offset	Register	Description
[0x0000]	*DMA_CHn_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.
29	-	RO	0	**Reserved**

DMA Channel *n* Control			DMA_CHn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
28:24	burst_size	R/W	0	**Burst Size** The number of bytes transferred into and out of the DMA FIFO in a single burst. 0: 1 byte. 1: 2 bytes. 2: 3 bytes. ... 31: 32 bytes.
23	-	RO	0	**Reserved**
22	dstinc	R/W	0	**Destination Increment Enable** This bit enables the automatic increment of the *DMA_CHn_DST* register upon every AHB transaction. This bit is ignored for a DMA transmit to peripherals. 0: Disabled. 1: Enabled.
21:20	dstwd	R/W	0	**Destination Width** This field selects the width of each AHB transaction to the destination peripheral or memory. The actual width can be less than this field's setting if fewer bytes are in the DMA FIFO than this field's selection. 0: 1 byte. 1: 2 bytes. 2: 4 bytes. 3: Reserved.
19	-	RO	0	**Reserved**
18	srcinc	R/W	0	**Source Increment on AHB Transaction Enable** This bit enables the automatic increment of the *DMA_CHn_SRC* register upon every AHB transaction. This bit is ignored for a DMA receive from peripherals. 0: Disabled. 1: Enabled.
17:16	srcwd	R/W	0	**Source Width** This field selects the width of each AHB transaction from the source peripheral or memory. The actual width can be less than this field's setting if the *DMA_CHn_CNT* register indicates a smaller value than the width setting. 0: 1 byte. 1: 2 bytes. 2: 4 bytes. 3: Reserved.
15:14	to_clkdiv	R/W	0	**Timeout Timer Clock Pre-Scale Select** This field selects the pre-scale divider for the timeout clock input. 0: Timeout timer disabled. 1: $\frac{f_{HCLK}}{2^8}$ 2: $\frac{f_{HCLK}}{2^{16}}$f 3: $\frac{f_{HCLK}}{2^{24}}$

DMA Channel *n* Control			DMA_CHn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
13:11	to_per	R/W	0	**Timeout Period Select** This field selects the number of pre-scaled clocks seen by the channel timer before a timeout condition is generated. The value is approximate because of synchronization delays between timers 0: 3 to 4. 1: 7 to 8. 2: 15 to 16. 3: 31 to 32. 4: 63 to 64. 5: 127 to 128. 6: 255 to 256. 7: 511 to 512.
10	to_wait	R/W	0	**Request DMA Timeout Timer Wait Enable** 0: Start timer immediately when enabled. 1: Delay the timer's start until after the first DMA transaction occurs.
9:4	request	R/W	0	**Request Select** Selects the source and destination for the transfer as shown in *Table 9-2*.
3:2	pri	R/W	0	**Channel Priority** This field sets the priority of the channel relative to other DMA channels. Channels set to the same priority are serviced in a round-robin fashion. 0: Highest priority. 1: … 2: … 3: Lowest priority.
1	rlden	R/W	0	**Reload Enable** Setting this bit to 1 allows reloading the *DMA_CHn_SRC*, *DMA_CHn_DST*, and *DMA_CHn_CNT* registers with their corresponding reload registers upon CTZ. *Note: 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 time out. 1: A channel time out occurred.
5	-	RO	0	**Reserved**
4	bus_err	R/W1C	0	**Bus Error** If this bit reads 1, an AHB abort occurred, and the channel is disabled by hardware. Write 1 to clear. 0: No error found. 1: An AHB bus error occurred.

DMA Channel *n* Status			DMA_CHn_STATUS	[0x0104]
Bits	**Field**	**Access**	**Reset**	**Description**
3	rld_if	R/W1C	0	**Reload Interrupt Flag** Reload. Write 1 to clear. 0: Reload has not occurred. 1: Reload occurred.
2	ctz_if	R/W1C	0	**CTZ Interrupt Flag** Write 1 to clear. 0: CTZ has not occurred. 1: CTZ has occurred.
1	ipend	RO	0	**Channel Interrupt Pending** 0: No interrupt 1: Interrupt pending
0	status	RO	0	**Channel Status** This bit indicates when it is safe to change the channel's configuration, address, and count registers. Whenever this bit is cleared by hardware, the *DMA_CHn_CTRL.en* bit is also cleared. 0: Disabled. 1: Enabled.

Table 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 Address** This field is the source RAM address for memory-to-peripheral and memory-to-memory transfers. This field is ignored for peripheral-to-memory transfers. If *DMA_CHn_CTRL.srcinc* = 1, then this register is incremented on each AHB transfer cycle by one, two, or four bytes depending on the data width selected using *DMA_CHn_CTRL.srcwd*. If *DMA_CHn_CTRL.srcinc* = 0, this register remains constant. If a CTZ condition occurs while *DMA_CHn_CTRL.rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_SRCRLD* register.

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

Table 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 this register's value 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 this register's value 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 this register's value is loaded into *DMA_CHn_CNT* upon a CTZ condition.

10. Analog-to-Digital Converter (ADC)

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

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

- 12-bit successive approximation ADC
- Conversion speed up to 1MSPS without input buffer
- Conversion speed up to 25KSPS with input buffer
- Internal reference without external capacitor
  - 1.25V or 2.048V
- Support for external reference from 2.048V to V_{DDA}.
  - The external reference and AIN0 share a pin. When using the external reference, AIN0 is not available as an input source.
- Capacitor calibration

10.1 Operation

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

Conversion sequences can measure single or multiple channels. The specific channels for a conversion sequence are set using the ADC channel select registers (*ADC_CHSEL1*:*ADC_CHSEL0*) and the *slot0_id* through *slot7_id* fields. A conversion sequence begins with the channel configured for slot 0 and continues sequentially through the software configured number of slots (*ADC_CTRL1*.*num_slots*) up to slot 7.

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

10.2 Input Channels

Each of the input channels is shown in *Table 10-1*.

Table 10-1: MAX32662 Channel Assignments

Channel ID	Source	Mode	Pin Number	Alternate Function Number	Alternate Function Name
0	AIN0	Single-ended	P0.13	AF4	AIN0
1	AIN1	Single-ended	P0.12	AF4	AIN1
2	AIN2	Single-ended	P0.11	AF4	AIN2
3	AIN3	Single-ended	P0.10	AF4	AIN3
4-11	Reserved	-	-	-	-
12	$V_{DDA}/2$	Single-ended	N/A	N/A	N/A
13	Reserved	-	-	-	N/A
14	V_{CORE}	Single-ended	N/A	N/A	N/A

Channel ID	Source	Mode	Pin Number	Alternate Function Number	Alternate Function Name
15	V_{SSA}	Single-ended	N/A	N/A	N/A
16-17	Reserved	-	-	-	-
18	$V_{DDIO}/4$	Single-ended	N/A	N/A	N/A

10.3 Analog Input Buffer Path

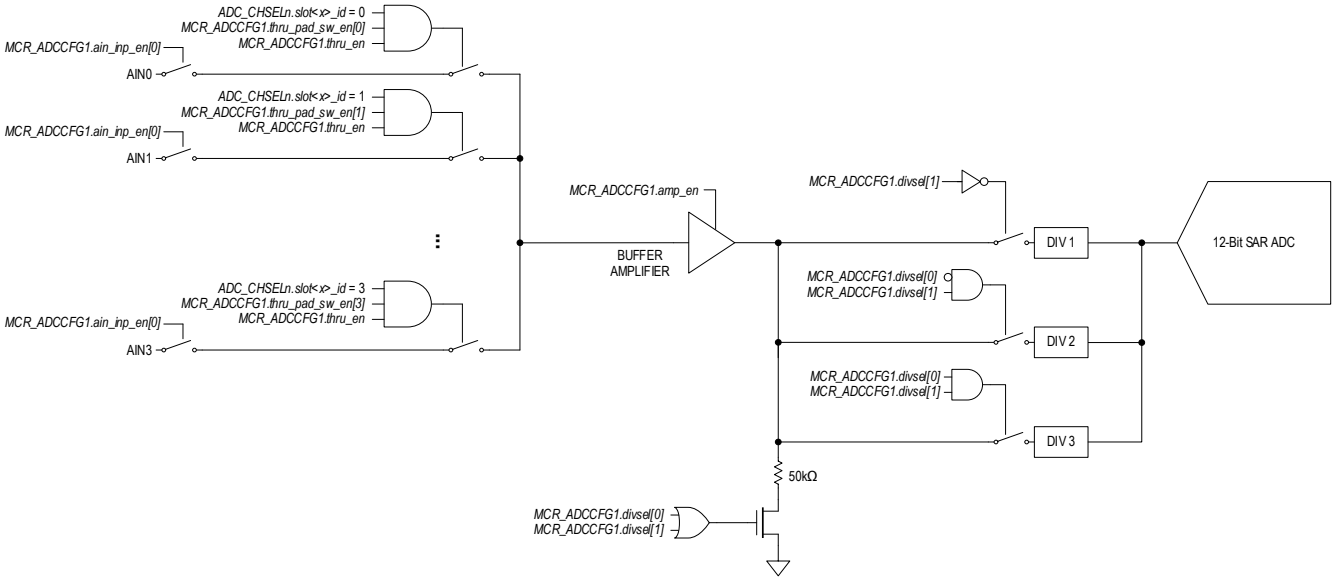
The MAX32662 includes an analog input buffer, optionally selectable for each of the external analog input signals. *Figure 10-1* shows the analog input buffer path. When using the analog input buffer path, the ADC is limited to a maximum of 25KSPS. See *Analog Input Buffer Bypass Path* for details on configuring the sample rate of the ADC. Enable rail-to-rail support for the buffer by setting *MCR_ADCCFG1.amp_rri_en* to 1.

Note: Each individual external analog input can be selected for use with the analog input buffer path. The input signal divider (MCR_ADCCFG1.divsel) applies to all inputs using the analog input buffer path.

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

1. Configure P0.11 for Alternate Function 4. See *Alternate Function Configuration* for details on configuring an I/O pin for alternate functionality.
2. Set the *MCR_ADCCFG1.thru_pad_sw_en[2]* field to enable analog input 2 to connect to the analog input buffer path.
   a. Each analog input can be enabled for the buffer path by setting the corresponding MUX pad switch enable field (*thru_pad_sw_en[3:0]*) in the *MCR_ADCCFG1* register.
3. Set *MCR_ADCCFG1.thru_en* to 1 to enable the analog input buffer path.
4. Set the *MCR_ADCCFG1.ain_inp_en[2]* field to enable analog input 2 to connect to the analog input buffer path.
   a. Each analog input must be enabled for use by the ADC by setting the corresponding analog input enable field in the *MCR_ADCCFG1* register.
5. Select the divide by 3 option by setting *MCR_ADCCFG1.divsel* field to 3.
6. Set *MCR_ADCCFG1.amp_en* to 1 to enable the analog input buffer.
7. Configure the ADC for operation and set the desired slot id field for AIN2.
   a. When the input buffer is enabled, configure the ADC to limit the maximum sampling rate to 25KSPS due to buffer bandwidth limitations.

Figure 10-1: MAX32662 Analog Input Buffer Signal Path



10.4 Analog Input Buffer Bypass Path

The MAX32662 includes a non-buffered input path, optionally selectable for each of the external analog input signals. When using the non-buffered input path, the maximum sample rate is 1MSPS.

The following steps show how to enable the standard input path for AIN1.

1. Configure P0.12 for Alternate Function 4. See *Alternate Function Configuration* for details on configuring an I/O pin for alternate functionality.
2. Bypass the analog input buffer path by setting the following fields to 0.
   a. Set *MCR_ADCCFG1.thru_en* to 0 to bypass the analog input buffer path.
   b. Set *MCR_ADCCFG1.amp_en* to 0 to disable the analog input buffer.
3. Set the *MCR_ADCCFG1.ain_inp_en[1]* field to enable analog input 1 to connect to the ADC.
   a. Each external analog input must be enabled for use by the ADC by setting the corresponding analog input enable field in the *MCR_ADCCFG1* register.
4. The input divider must be set to divide by 1 (infinite resistance) by setting the *MCR_ADCCFG1.divsel* field to 0.
5. Configure the ADC for operation and set the desired slot id field for AIN1.
   a. When the input buffer is bypassed, the ADC is limited to a maximum of 1MSPS.

10.5 Clocks and Timing

Clock and timing configurations are calculated based on the application-specific sampling rate requirements. Several parameters can be adjusted to optimize the ADC power consumption, accuracy, and start-up time. *Table 10-2* shows the ADC clock sources available for the device. The maximum sample rate of the ADC is 1MSPS. If the analog input buffer is enabled, the maximum sample rate of the ADC is 25KSPS.

Table 10-2: MAX32662 ADC Clock Sources

ADC_CLKCTRL.clksel	Source (f_{ADC_SRC})
0	SYS_OSC
1	HF_EXT_CLK (P0.6 / AF4)

ADC_CLKCTRL.clksel	Source (f_{ADC_SRC})
2	IBRO
3	ERFO

The ADC clock frequency (f_{SAR_CLK}) is derived from a selectable clock source (f_{ADC_SRC}) and divided by a selectable clock divider, as shown in *Equation 10-1*. *Table 10-2* lists the available sources for f_{ADC_SRC}. The clock divider is selected using the *ADC_CLKCTRL.clkdiv* field.

Equation 10-1: ADC Clock Generation

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

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

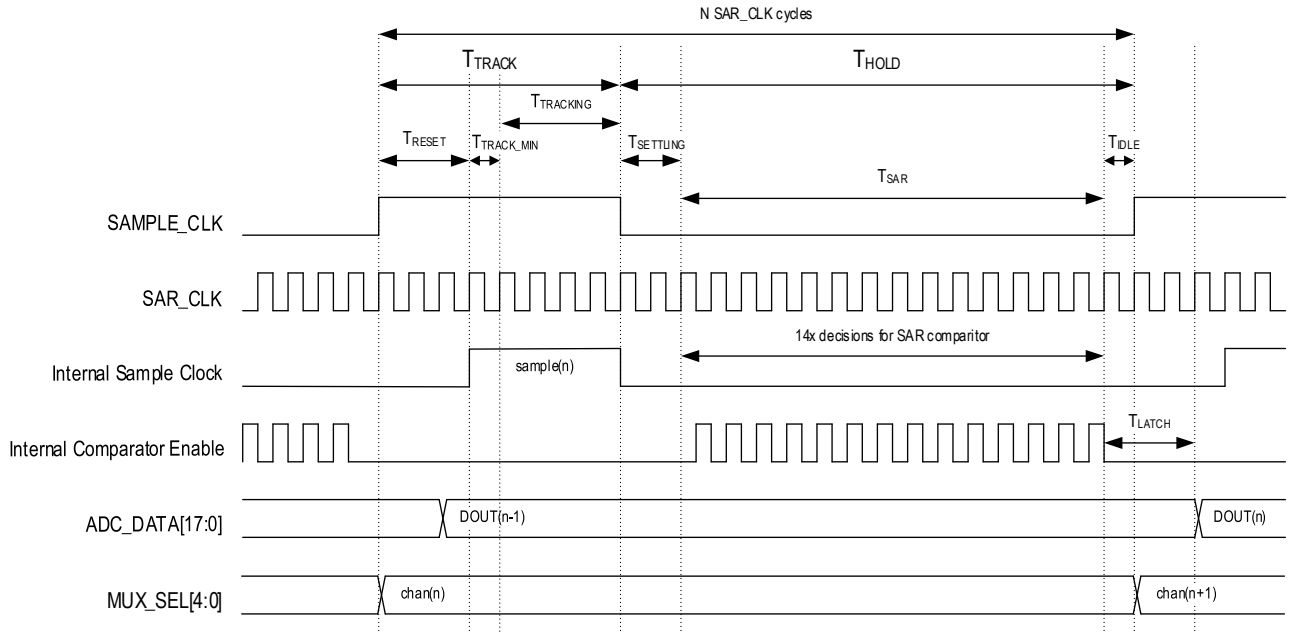
$$f_{SAR\_CLK} \leq 25MHz$$

The SAMPLE_CLK frequency determines the sampling rate, conversion time, and the delay between conversions. It is defined by a high track time and a low hold time of SAR_CLK periods. The sum of the track and hold defines the SAMPLE_CLK frequency (sample rate). *Figure 10-2* shows the SAMPLE_CLK and its relationship to the track and hold values.

Equation 10-2: Sample Clock Frequency Calculation

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

Figure 10-2: ADC Sample Clock



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

Equation 10-3: T_{TRACK} Calculation

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

$$T_{TRACK} \geq 8$$

Equation 10-4: T_{HOLD} Calculation

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

The ADC requires a minimum T_{TRACK} of 8 SAR_CLK cycles and a minimum T_{HOLD} of 17 SAR_CLK cycles. The *ADC_SAMPCLKCTRL.track_cnt* and *ADC_SAMPCLKCTRL.idle_cnt* fields add SAR_CLK cycles to the track and hold, as shown in *Equation 10-3* and *Equation 10-4*.

As an example, the following steps show the settings required to achieve a 1MSPS rate for the ADC using the IPO as the clock source.

1. Select the ADC_SRC clock as the IPO (SYS_OSC = IPO).
   a. Set *ADC_CLKCTRL.clksel* to 0.
2. Select the clock divider to achieve a valid SAR_CLK frequency using *Equation 10-1*.
   a. Set *ADC_CLKCTRL.clkdiv* to 1 (divide by 4, $f_{SAR\_CLK} \leq 25MHz$).
   b. $t_{SAR\_CLK} = 40ns$
3. Determine the SAMPLE_CLK for 1MSPS
   a. $T_{TRACK} + T_{HOLD} = \frac{25MHz}{1MHz} = 25$
4. Determine the *ADC_SAMPCLKCTRL.track_cnt* setting using *Equation 10-3*.
   a. *ADC_SAMPCLKCTRL.track_cnt* = 4 ($T_{TRACK} \geq 8$)
5. Determine the *ADC_SAMPCLKCTRL.idle_cnt* setting using *Equation 10-4*.
   a. $T_{HOLD} = 25 - T_{TRACK} = 25 - 8 = 17$ ($T_{HOLD} \geq 17$)
   b. *ADC_SAMPCLKCTRL.idle_cnt* = 17 - 17 = 0

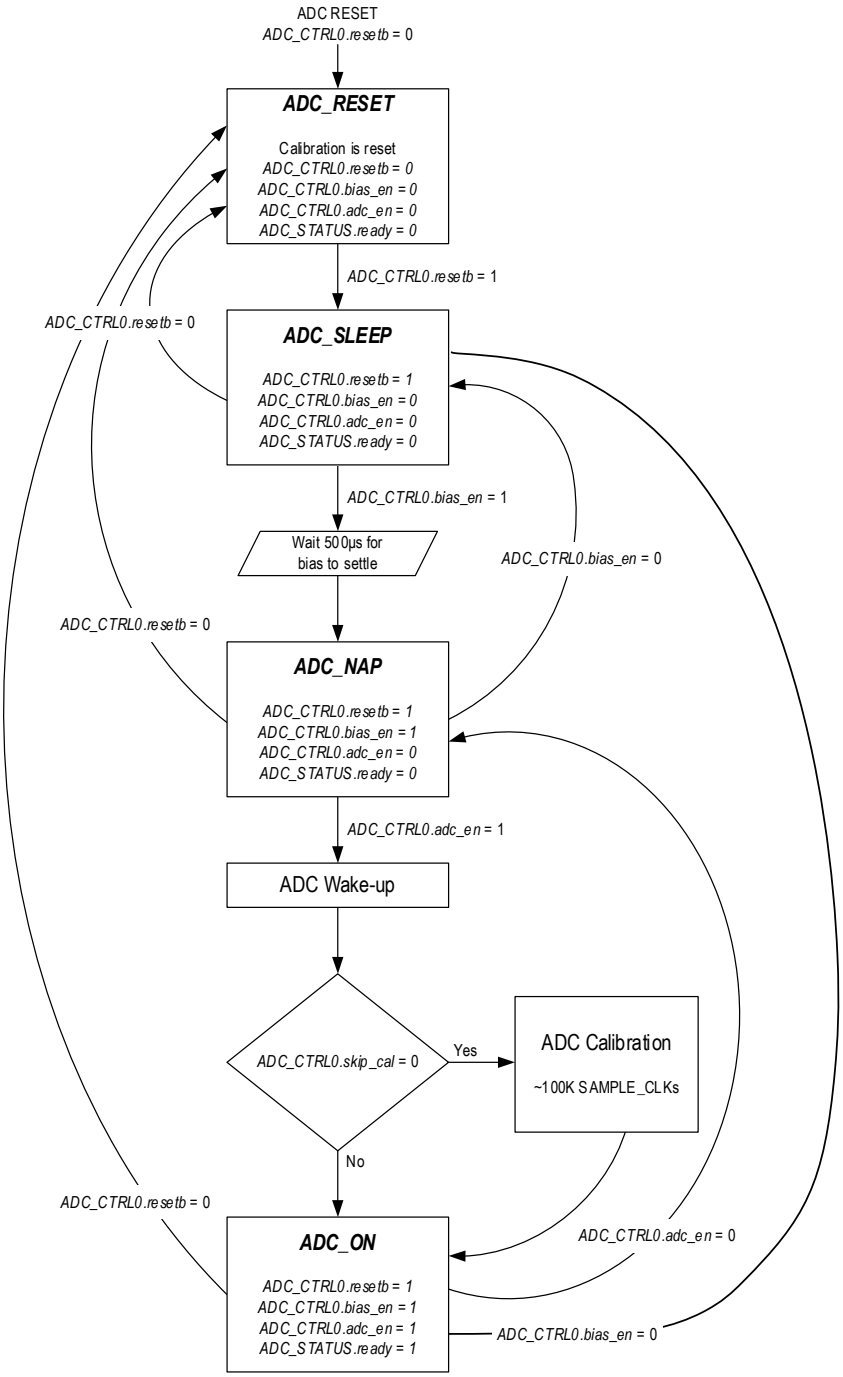
10.6 Operating Modes

Four operating modes allow the ADC to minimize power consumption based on the current needs of the peripheral. After a POR, system reset, peripheral reset (*GCR_RST0.adc* = 1), or software directly resetting the ADC (*ADC_CTRL0.resetb* = 0), the ADC bias regulator is disabled, and the ADC calibration values are reset to 0. The bias regulator must be enabled before performing a measurement. Optionally, a capacitor calibration can be performed. Enabling the bias regulator requires 500µs before performing a conversion. Section *10.6.1* describes initializing the ADC from *ADC_RESET*. Section *10.6.1.2* describes entering the *ADC_NAP* state. Section *10.6.1.3* describes the steps required to enter the *ADC_ON* state and perform an ADC capacitor calibration, and section *10.6.1.4* describes the steps to enter the *ADC_ON* state without performing a calibration. ADC calibration is only necessary after an ADC reset, changing the reference, or changing environmental conditions. *Figure 10-3* shows the ADC operating modes state diagram. *Table 10-3* shows the configuration bits' state and the status bit's state for each operating mode.

Table 10-3: ADC Operating States

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

Figure 10-3: ADC Operating Modes State Diagram



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

Enabling the bias regulator transitions the ADC to *ADC_NAP* state. Setting *ADC_CTRL0.bias_en* to 1 turns on the bias regulator required for ADC conversions. The bias regulator requires approximately 500µs to warm up. There is no dedicated status bit indicating the transition is complete, therefore the software must measure the required time. The ADC's sample rate should be configured with the ADC in the *ADC_NAP* state.

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

10.6.1 Initializing the ADC

10.6.1.1 Entering ADC_SLEEP State

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

To initialize the ADC and enter *ADC_SLEEP*:

1. Clear *GCR_PCLKDIS0.adc* to 0 to enable the ADC peripheral clock.
2. Clear *ADC_CTRL0.resetb* to 0 to enter reset.
3. Select the ADC_SRC clock from *Table 10-2* and set it to the selected clock using the *ADC_CLKCTRL.clksel* field.
4. Select the ADC reference source:
   - External: *MCR_ADCCFG0.ext_ref* to 1.
     - If the external reference is selected, AIN0 cannot be used as an input channel.
   - Internal, 1.25V: Clear *MCR_ADCCFG0.ext_ref* to 0 and clear *MCR_ADCCFG0.int_ref* to 0.
   - Internal, 2.048: Clear *MCR_ADCCFG0.ext_ref* to 0 and set *MCR_ADCCFG0.int_ref* to 1.
5. If using external analog inputs, configure the signal path as described in *Analog Input Buffer* or *Analog Input Buffer Bypass Path* sections.
6. Configure the GPIO associated with the desired external channels as inputs in high-impedance mode. Configure the alternate function mode as indicated in *Table 10-1*.
7. Set the *ADC_CTRL0.resetb* to 1 to enter the *ADC_SLEEP* state.

10.6.1.2 Entering ADC_NAP State

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

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

10.6.1.3 Entering ADC_ON State Using Calibration

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

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

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

10.6.1.4 Entering ADC_ON State Skipping Calibration

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

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

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

10.7 ADC SFR Interface

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

10.7.1 Determination of Bias and Wake-up Counter Settings

The ADC bias and wake-up are configurable to achieve optimum performance based on the sample rate of the ADC. The bias must be at least 500µs and the ADC wake-up timer must be at least 30µs to ensure optimum reference buffer settling requirements. The settings for the bias counter and wake-up counter are dependent on the configured ADC sample rate. *Table 10-4* shows the settings for the bias and wake-up counters, and the resulting number of clock cycles each setting achieves. The following steps show how to determine the settings for the bias counter and wake-up counter for a sample rate of 1MSPS.

1. The bias counter must be 500µs, which results in $1MHz \times 500\mu s = 500$ cycles.
   a. Referring to *Table 10-4*, the closest bias counter setting to achieve at least 500 cycles is 7 (512 clock cycles).
2. The wake-up counter must be 30µs, which results in $1MHz \times 30\mu s = 30$ cycles.
   a. Referring to *Table 10-4*, the closest wake-up counter setting to achieve at least 30 cycles is 11 (32 clock cycles).

Table 10-4: Bias and Wake-up Clock Cycle Selection

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

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

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

10.7.3 1.25V Internal Reference Trim

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

1. Write address 0x01 to *ADC_SFRADDR*.
2. Read the SFR data by reading a byte from the *ADC_SFRADDR* register.
3. Perform a bit-wise OR of the result of step 2 with 0x0F.
4. Write the byte from step 3 to the *ADC_SFRWRDATA* register.
5. Write address 0x0B to *ADC_SFRADDR*.
6. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
7. Mask off the upper two bits of the data read by performing a bit-wise AND of the value read with 0xC0.
8. Perform a bit-wise OR of the result of step 3 with *FCR_ADCREFTRIM0.vx2_tune*.
9. Write the byte from step 4 to the *ADC_SFRWRDATA* register.
10. Write *FCR_ADCREFTRIM0.vcm* to *MCR_ADCCFG2.vcm*.
11. Write *FCR_ADCREFTRIM0.vrefm* to *MCR_ADCCFG2.vrefm*.
12. Write *FCR_ADCREFTRIM0.vrefp* to *MCR_ADCCFG2.vrefp*.
13. Write *FCR_ADCREFTRIM2.iboost_1p25* to *MCR_ADCCFG2.d_iboost*.
14. Write *FCR_ADCREFTRIM2.idrv_1p25* to *MCR_ADCCFG2.idrv*.
15. Write address 0x05 to *ADC_SFRADDR*.
16. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
17. Mask off bits 7:4 of the data read by performing a bit-wise AND of the value read with 0xF0.
18. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of the data from step 17 and the bias counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
19. Write address 0x06 to *ADC_SFRADDR*.
20. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
21. Mask off bits 7:4 of the data read by performing a bit-wise AND of the value read with 0xF0.
22. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of the data from step 21 and the calculated wake-up counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
23. Move the ADC into the ADC_ON state using calibration.

10.7.4 2.048V Internal Reference Trim

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

1. Write address 0x01 to *ADC_SFRADDR*.
2. Read the SFR data by reading a byte from the *ADC_SFRADDR* register.
3. Perform a bit-wise OR of the result of step 2 with 0x0F.
4. Write the byte from step 3 to the *ADC_SFRWRDATA* register.
5. Write address 0x0B to *ADC_SFRADDR*.
6. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
7. Mask off the upper two bits of the data read by performing a bit-wise AND of the value read with 0xC0.
8. Perform a bit-wise OR of the result of step 3 with *FCR_ADCREFTRIM1*.*vx2_tune*.
9. Write the byte from step 4 to the *ADC_SFRWRDATA* register.
10. Write *FCR_ADCREFTRIM1*.*vcm* to *MCR_ADCCFG2*.*vcm*.
11. Write *FCR_ADCREFTRIM1*.*vrefm* to *MCR_ADCCFG2*.*vrefm*.
12. Write *FCR_ADCREFTRIM1*.*vrefp* to *MCR_ADCCFG2*.*vrefp*.
13. Write *FCR_ADCREFTRIM2*.*iboost_2p048* to *MCR_ADCCFG2*.*d_iboost*.
14. Write *FCR_ADCREFTRIM2*.*idrv_2p048* to *MCR_ADCCFG2*.*idrv*.
15. Write address 0x05 to *ADC_SFRADDR*.
16. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
17. Mask off bits 7:4 of the data read by performing a bit-wise AND of the value read with 0xF0.
18. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of the data from step 17 and the bias counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
19. Write address 0x06 to *ADC_SFRADDR*.
20. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
21. Mask off bits 7:4 of the data read by performing a bit-wise AND of the value read with 0xF0.
22. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of the data from step 21 and the calculated wake-up counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
23. Move the ADC into the ADC_ON state using calibration.

10.7.5 External Reference Trim

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

1. Write address 0x01 to *ADC_SFRADDR*.
2. Read the SFR data by reading a byte from the *ADC_SFRADDR* register.
3. Perform a bit-wise OR of the result of step 2 with 0x0F.
4. Write the byte from step 3 to the *ADC_SFRWRDATA* register.
5. Write address 0x0B to *ADC_SFRADDR*.
6. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
7. Mask off the upper bit of the data read by performing a bit-wise AND of the value read with 0x80.
8. Perform a bit-wise OR of the result of step 7 with *FCR_ADCREFTRIM2.vx2_tune*.
9. Write the byte from step 8 to the SFR by writing it to the *ADC_SFRWRDATA* register.
10. Write *FCR_ADCREFTRIM2.vcm* to *MCR_ADCCFG2.vcm*.
11. Write address 0x05 to *ADC_SFRADDR*.
12. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
13. Mask off bits 7:4 of the data read by performing a bit-wise AND of the value read with 0xF0.
14. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of the data from step 13 and the bias counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
15. Write address 0x06 to *ADC_SFRADDR*.
16. Read the SFR data by reading a byte from the *ADC_SFRRDDATA* register.
17. Mask off bits 7:4 of the data read by performing a bit-wise AND of the value read with 0xF0.
18. Write the following byte to the *ADC_SFRWRDATA* register.
    a. Perform a bitwise OR of the data from step 17 and the calculated wake-up counter setting. See *Determination of Bias and Wake-up Counter Settings* for details on calculating this value.
19. Move the ADC into the ADC_ON state using calibration.

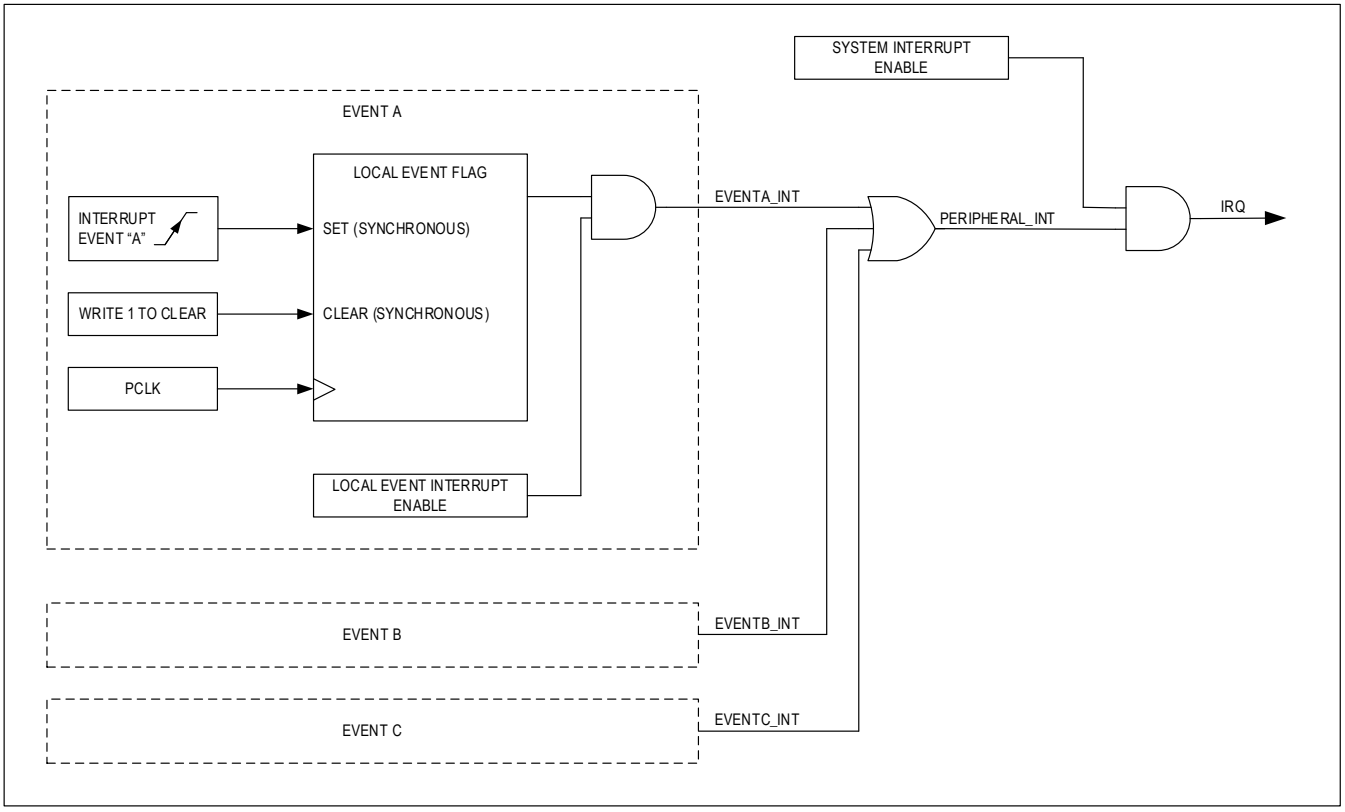
10.8 Interrupts

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

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

Figure 10-4 is a functional diagram showing this relationship.

Figure 10-4: Interrupt Event Signal Generation



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

The interrupt events supported are listed in *Table 10-5*.

Table 10-5: MAX32662 Interrupt Events

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

Event	Description	Interrupt Flag	Interrupt Enable
Conversion Sequence Abort	*ADC_CTRL1*.*start* transitioned from 1 to 0 before a conversion started.	*ADC_INTFL*.*abort*	*ADC_INTEN*.*abort*
ADC Ready	ADC transitioned to the *ADC_ON* state.	*ADC_INTFL*.*ready*	*ADC_INTEN*.*ready*

10.9 FIFO Operation

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

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

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

10.10 Averaging

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

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

10.11 Conversion Results

The results and status information are read from the *ADC_DATA* register. The selection of the format of the *ADC_DATA* register is determined using the *ADC_FIFODMACTRL*.*data_format* field. Each of the format options is shown in *Table 10-6*. A visual representation of the corresponding format of the *ADC_DATA* register is shown in *Figure 10-5*.

In processed modes, the status information includes:

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

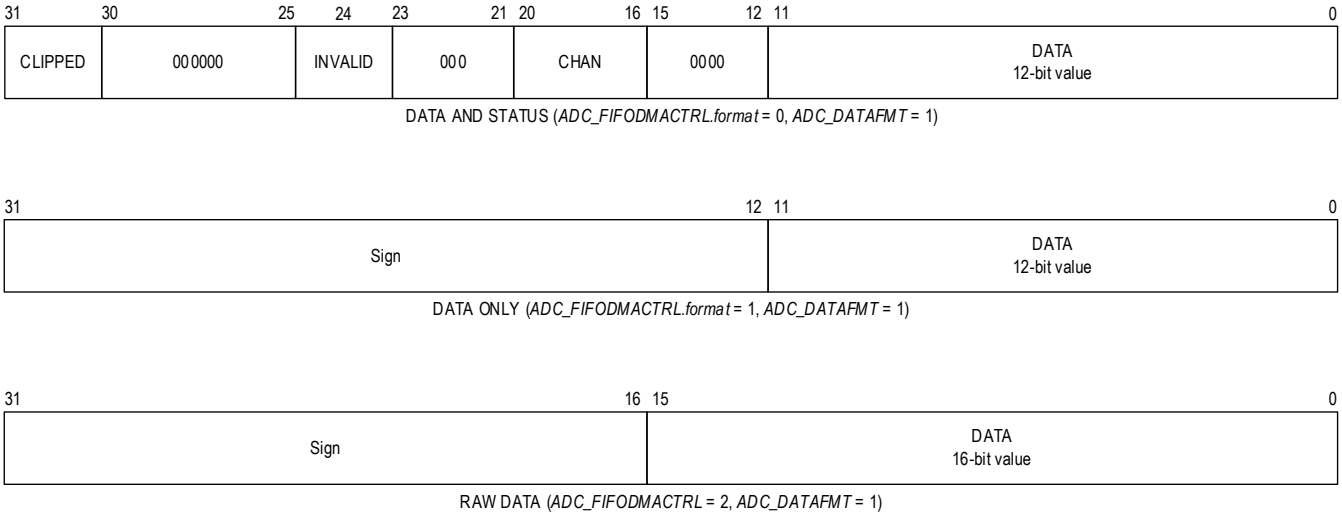
The result formatting options are shown in *Table 10-6*.

Table 10-6: ADC_DATA Register Result Formatting

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

The information structure depends on the channel mode (single-ended or differential) and the selected data format, as shown in *Figure 10-4*.

Figure 10-5: ADC Result Formats



10.12 Conversions

10.12.1 Conversion Sequence Triggers

A conversion sequence is initiated by either a software or hardware trigger. This flexibility allows manual or on-demand measurements using a timer peripheral or an external GPIO. *Table 10-7* lists the hardware triggers available to start a conversion sequence.

Table 10-7: MAX32662 Hardware Conversion Triggers

ADC_CTRL1.trig_sel	Source
0	TMR0 output
1	TMR1 output
2	TMR2 output
3	Reserved
4	ADC_TRIG_D[1]
5	ADC_TRIG_E[1]
6	AIN_TRIG_C[1]
7	Reserved

ADC_CTRL1.trig_sel	Source
1. Refer to the device data sheet's pin description table for alternate function assignments. Not all alternate functions are available on all packages.	

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

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

A hardware-triggered conversion sequence starts when the selected trigger becomes active. Only one of the hardware triggers, shown in *Table 10-7*, can be selected for the conversion sequence.

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

The hardware trigger source must be running and properly configured to generate the trigger signal. Trigger sources are edge-triggered; the event is only recognized if a GPIO pin transitions from low to high or a timer output signal transitions from inactive to active. As a result, software must clear the GPIO or timer output each time before the hardware trigger event occurs for the trigger to be recognized. See *Alternate Function Configuration* in the GPIO chapter for details on configuring port pin alternate functions.

Table 10-8: Conversion Sequence Configurations

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

10.12.2 Single Conversion Sequences

10.12.2.1 Software Triggered

To perform a software-triggered single conversion sequence:

1. Configure the ADC and enter the *ADC_ON* state as described in *Operating Modes*.
2. Clear the *ADC_CTRL1*.*trig_mode* field to 0 to select software triggering.
3. Set *ADC_CTRL1*.*cnv_mode* to 0 to select a single conversion sequence.
4. For external channels, configure the input path, as described in the *Analog Input Buffer Path* or *Analog Input Buffer Bypass Path* sections.
4. Select the number of channels to convert by setting the *ADC_CTRL1*.*num_slots* to the number of channels minus 1.
    a. As an example, set *ADC_CTRL1*.*num_slots* to 4 to perform a single conversion on 5 channels.
5. Set the desired channels for the conversion using the *ADC_CHSEL1*:*ADC_CHSEL0* registers slot fields.
    a. As an example, to perform a single conversion sequence on channels 1, 3, 2, 12, and 14 (*ADC_CTRL1*.*num_slots* = 4), set the channel select fields as follows:
        i.) *ADC_CHSEL0*.*slot0_id* = 1
        ii.) *ADC_CHSEL0*.*slot1_id* = 3
        iii.) *ADC_CHSEL0*.*slot2_id* = 2
        iv.) *ADC_CHSEL0*.*slot3_id* = 12
        v.) *ADC_CHSEL1*.*slot4_id* = 14
6. Configure *ADC_CTRL1*.*avg* to the desired number of samples to average.
    a. Set this field to 0 for a single conversion per channel. See *Averaging* for details on sample averaging.
7. Set the data format for the conversion results using the *ADC_FIFODMACTRL*.*data_format* field. See *Conversion Results* for details.
8. Clear the interrupt flags by writing 0xFFFF FFFF to the *ADC_INTFL* register.
9. Set *ADC_CTRL1*.*start* to 1 to start the conversion sequence. The conversion sequence starts immediately.

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

- Hardware sets *ADC_INTFL*.*conv_done* to 1.
- Hardware sets *ADC_INTFL*.*seq_done* to 1, indicating a sequence done event occurred.
- Software should set *ADC_CTRL1*.*start* to 0 to prevent additional conversions.
- The converted data is available in the *ADC_DATA* register. See *FIFO Operation* for details on the FIFO.

10.12.2.2 Hardware-Triggered

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

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

1. Configure the ADC and enter the *ADC_ON* state as described in *Operating Modes*.
2. Clear the *ADC_CTRL1*.*start* field to 0.
3. Set *ADC_CTRL1*.*trig_mode* to 1 to select hardware triggering.
4. Configure *ADC_CTRL1*.*trig_sel* for the desired hardware trigger. See *Table 10-7* for details of available hardware triggers.
5. Set *ADC_CTRL1*.*cnv_mode* to 0 to select a single conversion sequence.
6. Configure the selected hardware trigger.
7. Select the number of channels to convert by setting the *ADC_CTRL1*.*num_slots* to the number of channels minus 1.
   a. As an example, set *ADC_CTRL1*.*num_slots* to 1 to perform a single conversion on 2 channels.
8. Set the desired channels for the conversion using the *ADC_CHSEL1*:*ADC_CHSEL0* registers slot fields.
   a. As an example, to perform a single conversion sequence on channels 2 and 3 (*ADC_CTRL1*.*num_slots* = 1), set the channel select fields as follows:
      i.) *ADC_CHSEL0*.*slot0_id* = 2
      ii.) *ADC_CHSEL0*.*slot1_id* = 3
9. Configure *ADC_CTRL1*.*avg* to the desired number of samples to average.
   a. Set this field to 0 for a single conversion per channel. See *Averaging* for details on sample averaging.
10. Set the data format for the conversion results using the *ADC_FIFODMACTRL*.*data_format* field. See *Conversion Results* for details.
11. Clear the interrupt flags by writing 0xFFFF FFFF to the *ADC_INTFL* register.
12. Set *ADC_CTRL1*.*start* to 1 to arm the conversion sequence. The conversion sequence begins when the hardware trigger is activated.
13. When the sequence is triggered, hardware sets the *ADC_INTFL*.*seq_started* field to 1.

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

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

10.12.3 Continuous Conversion Sequences

10.12.3.1 Software-Triggered, Continuous Conversion Sequence

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

1. Configure the ADC and enter the *ADC_ON* state as described in *Operating Modes*.
2. Clear the *ADC_CTRL1*.*trig_mode* field to 0 to select software triggering.
3. Set *ADC_CTRL1*.*cnv_mode* to 1 to select continuous conversion mode.
4. Select the number of channels to convert by setting the *ADC_CTRL1*.*num_slots* to the number of channels minus 1.
5. Set the desired channels for the conversion using the *ADC_CHSEL1*:*ADC_CHSEL0* registers slot fields.
6. Configure *ADC_CTRL1*.*avg* to the desired number of samples to average.
   a. Set this field to 0 for a single conversion per channel. See *Averaging* for details on sample averaging.
7. Set the data format for the conversion results using the *ADC_FIFODMACTRL*.*data_format* field. See *Conversion Results* for details.
8. Clear the interrupt flags by writing 0xFFFF FFFF to the *ADC_INTFL* register.
9. If a delay between continuous conversion sequences is desired, set the number of SAMPLE_CLKs to delay using the *ADC_RESTART*.*cnt* field.
10. Set *ADC_CTRL1*.*start* to 1 to activate the conversion sequence. The conversion sequence starts immediately.
11. Software should clear *ADC_CTRL1*.*start* to stop a continuous conversion sequence when desired.

At the end of each continuous conversion sequence:

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

10.12.3.2 Hardware-Triggered, Continuous Conversion Sequence

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

Perform a hardware-triggered continuous conversion sequence:

1. Configure the ADC and enter the *ADC_ON* state as described in *Operating Modes*.
2. Set *ADC_CTRL1.cnv_mode* to 1 to select continuous conversion mode.
3. Set *ADC_CTRL1.trig_mode* to 1 to select hardware triggering.
4. Configure the *ADC_CTRL1.trig_sel* field for the desired hardware trigger. See *Table 10-7* for a list of hardware triggers.
5. Configure the selected hardware trigger. See *Timers (TMR/LPTMR)* for timer configuration and *Alternate Function Configuration* in the GPIO chapter for details on configuring alternate functions.
6. Select the number of channels to convert by setting the *ADC_CTRL1.num_slots* to the number of channels minus 1.
7. Set the desired channels for the conversion using the *ADC_CHSEL1*:*ADC_CHSEL0* registers slot fields.
8. Configure *ADC_CTRL1.avg* to the desired sample averaging.
    a. Setting this field to 0 disables averaging. Set this field to 1 for a single conversion per channel. If this field is set to greater than 1, each channel selected is converted 2^{avg} number of times and averaged before moving to the next channel.
9. Set the data format for the conversion results using the *ADC_FIFODMACTRL.data_format* field. See *Conversion Results* for details.
10. Clear the interrupt flags by writing 0xFFFF FFFF to the *ADC_INTFL* register.
11. If a delay between continuous conversion sequences is desired, set the number of SAMPLE_CLKs to delay using the *ADC_RESTART.cnt* field.
12. Set *ADC_CTRL1.start* to 1 to arm the conversion sequence.
13. When the sequence is triggered, the hardware sets *ADC_INTFL.seq_started* to 1. Continuous sequences can be stopped by clearing the *ADC_CTRL1.start* field to 0.

At the end of a continuous conversion sequence:

- Hardware sets *ADC_INTFL.seq_done* to 1, indicating a sequence done event occurred.
- If *ADC_CTRL1.start* is set to 1:
    - The device idles for the number of sample periods specified in the *ADC_RESTART.cnt* field and then starts another conversion sequence.
- If *ADC_CTRL1.start* is set to 0:
    - The hardware sets the *ADC_INTFL.conv_done* field to 1 and does not perform another conversion sequence.

10.13 Low-Power Analog Wake-Up Comparators

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

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

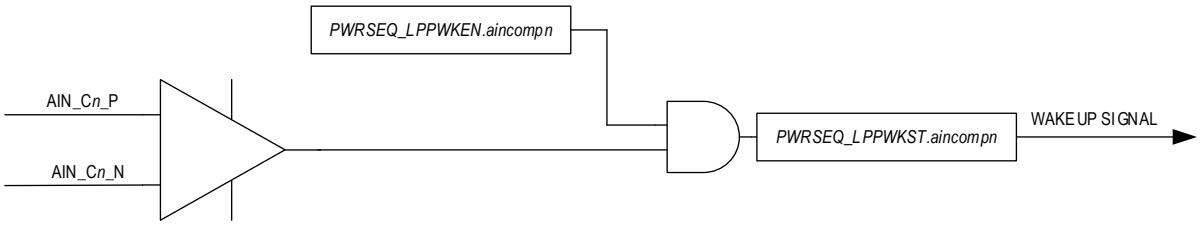
Table 10-9: MAX32662 Analog Comparator 0 Input Selection

Input	Input Signal	Selection
Positive	None	*MCR_AINCOMP.psel_comp0* = 0
	AIN2	*MCR_AINCOMP.psel_comp0* = 1
	AIN3	*MCR_AINCOMP.psel_comp0* = 2
Negative	None	*MCR_AINCOMP.nsel_comp0* = 0
	AIN0	*MCR_AINCOMP.nsel_comp0* = 1
	AIN1	*MCR_AINCOMP.nsel_comp0* = 2

Table 10-10: MAX32662 Analog Comparator 1 Input Selection

Input	Input Signal	Selection
Positive	None	*MCR_AINCOMP.psel_comp1* = 0
	AIN2	*MCR_AINCOMP.psel_comp1* = 1
	AIN3	*MCR_AINCOMP.psel_comp1* = 2
Negative	None	*MCR_AINCOMP.nsel_comp1* = 0
	AIN0	*MCR_AINCOMP.nsel_comp1* = 1
	AIN1	*MCR_AINCOMP.nsel_comp1* = 2

Figure 10-6: Analog Wake-up Comparators



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

The control and status fields are listed in *Table 10-11*.

Table 10-11: MAX32662 Analog Wake-Up Comparator Fields

Comparator	Wake-up Flag	Wake-up Enable	Comparator Output
AINCOMP0	*PWRSEQ_LPPWKFL.aincomp0*	*PWRSEQ_LPPWKEN.aincomp0*	*PWRSEQ_LPPWKFL.aincomp0_st*
AINCOMP1	*PWRSEQ_LPPWKFL.aincomp1*	*PWRSEQ_LPPWKEN.aincomp1*	*PWRSEQ_LPPWKFL.aincomp1_st*

Configure the comparators as follows:

1. Select the comparator's inputs. See *Table 10-9* and *Table 10-10* for details on selecting each comparator's inputs.
2. Enable the analog input by setting the corresponding bits in the *MCR_ADCCFG1.ain_inp_en* field to 1.
3. Enable the selected input's alternate function for the GPIO. See *Alternate Function Configuration* in the GPIO chapter for details. Refer to the device data sheet alternate function table for pin assignments.

10.14 Registers

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

Table 10-12: ADC Register Summary

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

10.14.1 Register Details

Table 10-13: ADC Control 0 Register

ADC Control 0				ADC_CTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:5	-	RO	0	**Reserved**	
4	resetb	R/W	0	**Reset ADC** 0: ADC is in *ADC_RESET*. 1: Not in *ADC_RESET*.	
3	chop_force	R/W	0	**Input Chopping** 0: Disabled. 1: Enabled.	

ADC Control 0				ADC_CTRL0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
2	skip_cal	R/W	0	**Skip Calibration** Set this field to 1 to skip automatic calibration before starting a conversion sequence. 0: Perform automatic calibration. 1: Skip automatic calibration.	
1	bias_en	R/W	0	**Bias Enable** 0: Disabled. 1: Enabled.	
0	adc_en	R/W	0	**ADC Enable** 0: Disabled. 1: Enabled.	

Table 10-14: ADC Control 1 Register

ADC Control 1				ADC_CTRL1	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:21	-	RO	0	**Reserved**	
20:16	num_slots	R/W	0	**Number of Slots Enabled per Conversion Sequence** 0: 1 slot. 1: 2 slots. 2: 3 slots. … : … 7: 8 slots. 8 - 31: Reserved.	
15:11	-	RO	0	**Reserved**	
10:8	avg	R/W	0	**Sample Averaging** 0: No averaging. All other values: Average 2^{avg} samples on each channel before reporting the results.	
7	-	RO	0	**Reserved**	
6:4	trig_sel	R/W	0	**Hardware Trigger Source** See *Table 10-7* for field settings.	
3	samp_ck_off	R/W	0	**Sample Clock Control** 0: Continuous sample clock. 1: Sample clock runs only while a channel is being measured.	
2	cnv_mode	R/W	0	**Conversion Mode** 0: Single conversion sequence. 1: Continuous conversion sequence.	
1	trig_mode	R/W	0	**Trigger Mode Control** 0: Software trigger. 1: Hardware trigger.	

ADC Control 1			ADC_CTRL1	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
0	start	R/W	0	**Conversion Start** In software-triggered mode (*ADC_CTRL1.trig_mode* = 0), a conversion sequence starts immediately when this field is set to 1. After a sequence (*ADC_INTFL.seq_done* = 1) or when a conversion is complete (*ADC_INTFL.conv_done* = 1), software should set this field to 0. In hardware-triggered mode (*ADC_CTRL1.trig_mode* = 1), a conversion sequence is armed when this field is set to 1. A conversion sequence starts when the selected hardware trigger becomes active. Any time after the hardware-triggered conversion is started (*ADC_INTFL.seq_started* = 1), software should set this field to 0 to prevent subsequent conversion sequences from starting if desired. See *Conversions* for details. 0: Conversion complete. 1: Start a conversion sequence or arm the ADC to start a hardware trigger.

Table 10-15: ADC Clock Control Register

ADC Clock Control			ADC_CLKCTRL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved**
6:4	clkdiv	R/W	3	**Clock Divider** See *Analog Input Buffer* for details on determining the required setting for this field. The maximum SAR_CLK frequency is 25MHz. 0: Divide by 2. 1: Divide by 4. 2: Divide by 8. 3: Divide by 16. 4-7: Divide by 1.
3:2	-	RO	0	**Reserved**
1:0	clksel	R/W1C	0	**Clock Source** This field selects the ADC peripheral clock. See *Table 10-2* for available clock sources.

Table 10-16: ADC Sample Clock Control Register

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

Table 10-17: ADC Channel Select 0 Register

ADC Channel Select 0			ADC_CHSEL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	0	**Reserved**

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

Table 10-18: ADC Channel Select 1 Register

ADC Channel Select 1			ADC_CHSEL1	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:29	-	RO	0	**Reserved**
28:24	slot7_id	R/W	0	**Slot 7 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
23:21	-	RO	0	**Reserved**
20:16	slot6_id	R/W	0	**Slot 6 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
15:13	-	RO	0	**Reserved**
12:8	slot5_id	R/W	0	**Slot 5 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.
7:5	-	RO	0	**Reserved**
4:0	slot4_id	R/W	0	**Slot 4 Channel Assignment** Channel number assigned to the slot. Invalid channel numbers are ignored.

Table 10-19: ADC Restart Count Register

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

Table 10-20: ADC Data Format Register

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

Table 10-21: ADC FIFO and DMA Control Register

ADC FIFO and DMA Control			ADC_FIFODMACTRL	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:8	thresh	R/W	0	**FIFO and DMA Threshold** When the number of words in the FIFO exceeds the threshold, a DMA request is triggered and, the *ADC_INTFL.fifo_lvl* interrupt flag is set. Valid settings for this field are 0 to 16.
7:4	-	RO	0	**Reserved**
3:2	data_format	R/W	0	**FIFO Data Format** 0b00: Data and status (12 bits processed plus status fields). 0b01: Data only (12 bits processed). 0b10: Raw Data Only (18-bit raw data). 0b11: Reserved.
1	flush	R/W1O	0	**FIFO Flush** Write 1 to flush the FIFO. This bit always reads 0. 0: Normal operation. 1: Flush FIFO.
0	dma_en	R/W	0	**DMA Enable** 0: Disabled. 1: Enabled.

Table 10-22: ADC Data Register

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

ADC Data			ADC_DATA	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	data	R/W	0xFFFF	**Data** The format of this data is configurable. See *Conversion Results* for details.

Table 10-23: ADC Status Register

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

Table 10-24: ADC Channel Status Register

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

Table 10-25: ADC Interrupt Enable Register

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

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

Table 10-26: ADC Interrupt Flags Register

ADC Interrupt Flags			ADC_INTFL	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved**
10	fifo_ofl	R/W	0	**FIFO Overflow Event** 0: Normal operation. 1: Event occurred.
9	fifo_ufl	R/W	0	**FIFO Underflow Event** 0: Normal operation. 1: Event occurred.
8	fifo_lvl	R/W	0	**FIFO Level Event** 0: Normal operation. 1: Event occurred.
7	clipped	R/W	0	**Data Clipped Event** 0: Normal operation. 1: Event occurred.
6	conv_done	R/W	0	**Conversion Done Event** 0: Normal operation. 1: Event occurred.
5	seq_done	R/W	0	**Sequence Done Event** 0: Normal operation. 1: Event occurred.

ADC Interrupt Flags			ADC_INTFL	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
4	seq_started	R/W	0	**Sequence Started Event** 0: Normal operation. 1: Event occurred.
3	start_det	R/W	0	**Command Start Event** The conversion command started. 0: Normal operation. 1: Event occurred.
2	abort	R/W	0	**Command Aborted Event** The conversion command aborted before conversions completed. 0: Normal operation. 1: Event occurred.
1	-	RO	0	**Reserved**
0	ready	R/W	0	**ADC Ready Event** 0: Normal operation. 1: Event occurred.

Table 10-27: ADC SFR Address Offset Register

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

Table 10-28: ADC SFR Address Register

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

Table 10-29: ADC SFR Write Data Register

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

Table 10-30: ADC SFR Read Data Register

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

Table 10-31: ADC SFR Status Register

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

11. UART (UART)

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 is independently configurable unless using a shared external clock source.

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

The peripheral provides the following features:

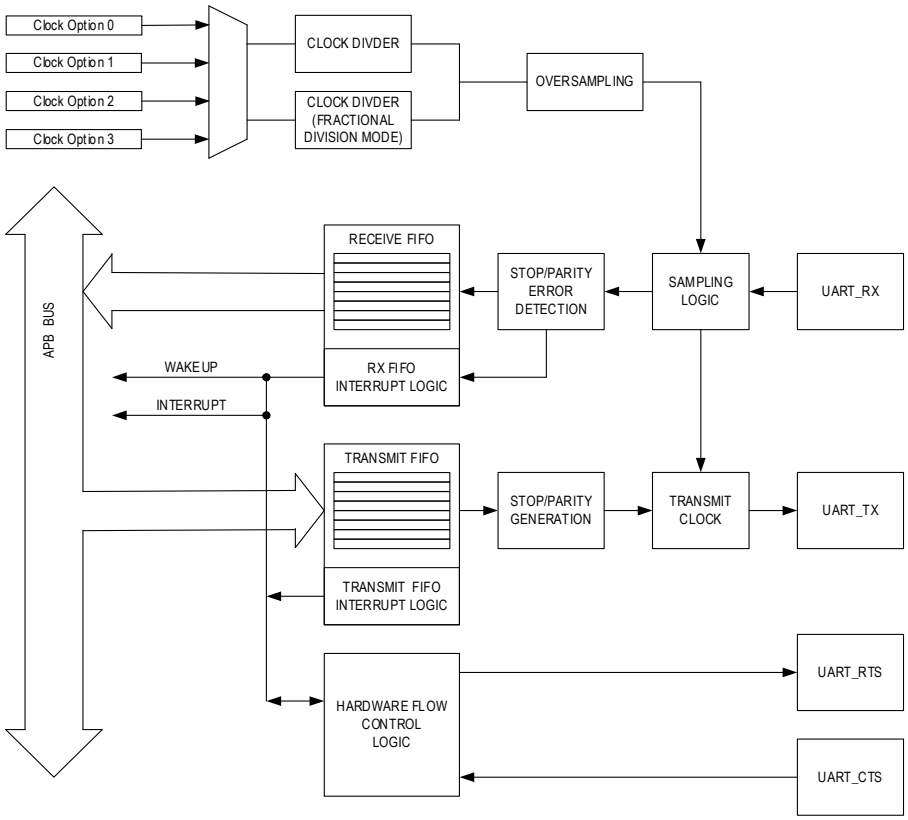
- Flexible baud rate generation for standard UART instances
- Programmable character size of 5 to 8 bits
- Stop bit settings of 1, 1.5, or 2 bits
- Parity settings of even, odd, mark (always 1), space (always 0), and no parity
- Automatic parity error detection with selectable parity bias
- Automatic frame error detection
- Separate 8-byte transmit and receive FIFOs
- Flexible interrupt conditions
- Hardware flow control (HFC) using ready-to-send (RTS) and clear-to-send (CTS) pins
- Separate DMA channels for transmit and receive
  - DMA support is available in *ACTIVE* and *SLEEP*

LPUART instances provides these additional features:

- Baud rate support for up to 1.85Mbps in *ACTIVE*
- Receive characters in *SLEEP*, *DEEPSLEEP*, and *BACKUP* at up to 9600 baud
- Fractional baud rate divisor improves baud rate accuracy for 9600 and lower baud rates
- Wake up from low-power modes to *ACTIVE* on multiple receive FIFO conditions

Figure 11-1 shows a high-level diagram of the UART peripheral.

Figure 11-1: UART Block Diagram



Note: See Table 11-1 for the clock options supported by each UART instance.

11.1 Instances

Instances of the peripheral are shown in *Table 11-1*. The standard UARTs and the LPUARTs are functionally similar; they are referred to as UART for common functionality. The LPUART instance supports fractional division mode (FDM) and is referenced as LPUART for feature-specific options.

Table 11-1: MAX32662 UART/LPUART Instances

Instance	Register Access Name	LPUART	Power Modes	Clock Option				HFC	Transmit FIFO Depth	Receive FIFO Depth
				0	1	2	3			
UART0	UART0	No	*ACTIVE SLEEP*	PCLK	HF_EXT_CLK P0.6 AF4	IBRO	ERFO	Yes	8	8
UART1	UART1									

11.2 DMA

Each UART instance supports DMA for both transmit and receive; separate DMA channels can be connected to the receive and transmit FIFOs.

The UART DMA channels are configured using the UART DMA configuration register, *UARTn_DMA*. Enable the receive FIFO DMA channel by setting *UARTn_DMA.rx_en* to 1 and enable the transmit FIFO DMA channel by setting *UARTn_DMA.tx_en*

to 1. DMA transfers are automatically triggered by the hardware based on the number of bytes in the receive FIFO and transmit FIFO.

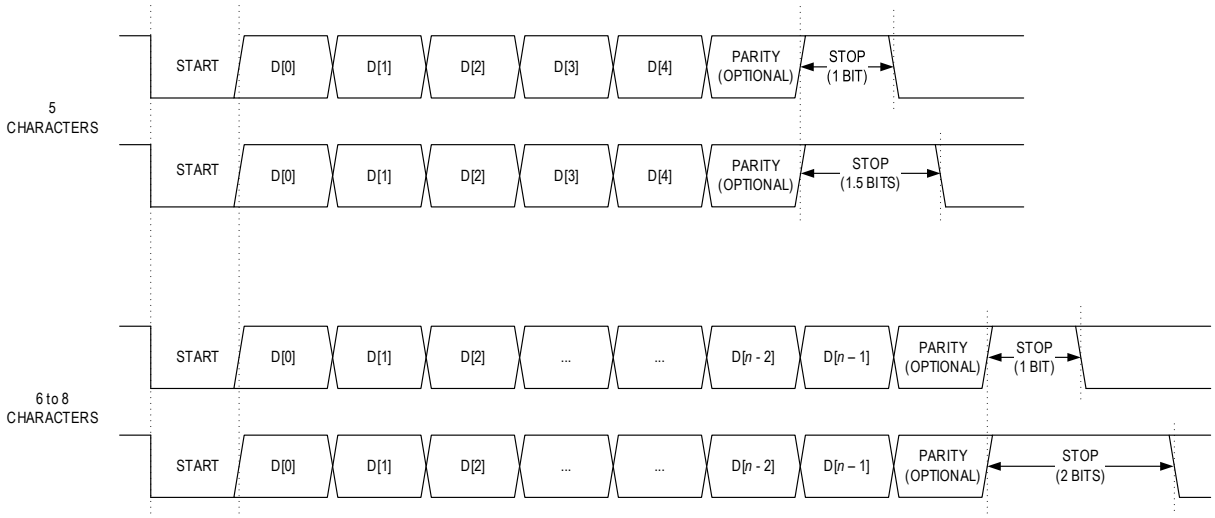
When DMA is enabled, the following describes the behavior of the DMA requests:

- A receive DMA request is asserted when the number of bytes in the receive FIFO transitions to be greater than or equal to the receive FIFO threshold.
- A transmit DMA request is asserted when the number of bytes in the transmit FIFO transitions to be less than the transmit FIFO threshold.

11.3 UART Frame

Figure 11-2 shows the UART frame structure. Character sizes of 5 to 8 bits are configurable through the *UARTn_CTRL.char_size* field. Stop bits are configurable as 1 or 1.5 bits for 5-character frames and 1 or 2 stop bits for 6, 7, or 8-character frames. Parity support includes even, odd, mark, space, and none.

Figure 11-2: UART Frame Structure



11.4 FIFOs

Separate receive and transmit FIFOs are provided. The FIFOs are both accessed through the same *UARTn_FIFO.data* field. The current level of the transmit FIFO is read from *UARTn_STATUS.tx_lvl*, and the receive FIFO current level is read from *UARTn_STATUS.rx_lvl*. Data for character sizes less than 7 bits are right justified.

11.4.1 Transmit FIFO Operation

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

11.4.2 Receive FIFO Operation

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

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

11.4.3 Flushing

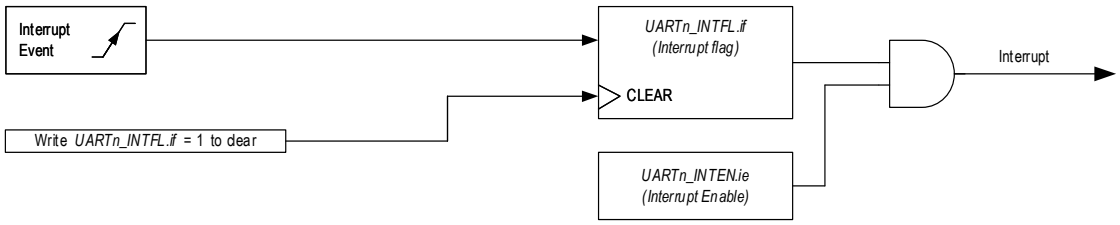
The FIFOs are flushed on the following conditions:

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

11.5 Interrupt Events

The peripheral generates interrupts for the events shown in *Table 11-2*. Unless noted otherwise, each instance has its own set of interrupts and higher-level flag and enable fields, as shown in *Table 11-2*

Figure 11-3: UART Interrupt Functional Diagram



Some activity can set one or more event flags and cause more than one event. An event interrupt occurs if the corresponding interrupt enable is set. The interrupt flags, when set, must be cleared by the software by writing 1 to the corresponding interrupt flag field.

Table 11-2: MAX32662 Interrupt Events

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

11.5.1 Frame Error

A frame error is generated when the UART sampling circuitry detects an invalid bit. Each bit is sampled three times, as shown in *Figure 11-4*, and can generate a frame error on the start bit, stop bit, data bits, and optionally the parity bit. When a frame error occurs, the data is discarded.

The frame error criteria are different based on the following:

- Standard UART and LPUART with FDM disabled.
  - The start bit is sampled 3 times, and all samples must be 0, or a frame error is generated.
  - Each data bit is sampled, and 2 of the 3 samples must match, or a frame error is generated.
  - If parity is enabled, the parity bit is sampled 3 times, and all samples must match, or a frame error is generated.
  - The stop bit is sampled 3 times, and all samples must be 1, or a frame error is generated.
  - See *Table 11-3* for details
- LPUART with FDM enabled (*UARTn_CTRL.fdm* = 1) and data/parity edge detect enabled (*UARTn_CTRL.dpfe_en* = 1).
  - The start bit is sampled 3 times, and all samples must be 0, or a frame error is generated.
  - Each data bit is sampled 3 times, and all samples must match, or a frame error is generated.
  - If parity is enabled, the parity bit is sampled 3 times, and all samples must match, or a frame error is generated.
  - The stop bit is sampled 3 times, and all samples must be 1, or a frame error is generated.
  - See *Table 11-4* for details.

Table 11-3: Frame Error Detection for Standard UARTs and LPUART

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

Table 11-4: 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	

11.5.2 Parity Error

Set *UARTn_CTRL*.*par_en* = 0 to enable parity checking of the received frame. If the calculated parity does not match the parity bit, then the corresponding interrupt flag is set. The data received is saved to the receive FIFO when a parity error occurs.

11.5.3 CTS Signal Change

A CTS signal change condition occurs if HFC is enabled, the UART baud clock is enabled, and the CTS pin changes state.

11.5.4 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.

11.5.5 Receive FIFO Threshold

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

11.5.6 Transmit FIFO Half-Empty

The transmit FIFO half-empty event occurs when *UARTn_STATUS*.*tx_lvl* transitions from more than half-full to half-empty, as shown in *Equation 11-1*.

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

Equation 11-1: UART Transmit FIFO Half-Empty Condition

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

11.5.7 Transmit FIFO Almost Empty

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

11.6 Inactive State

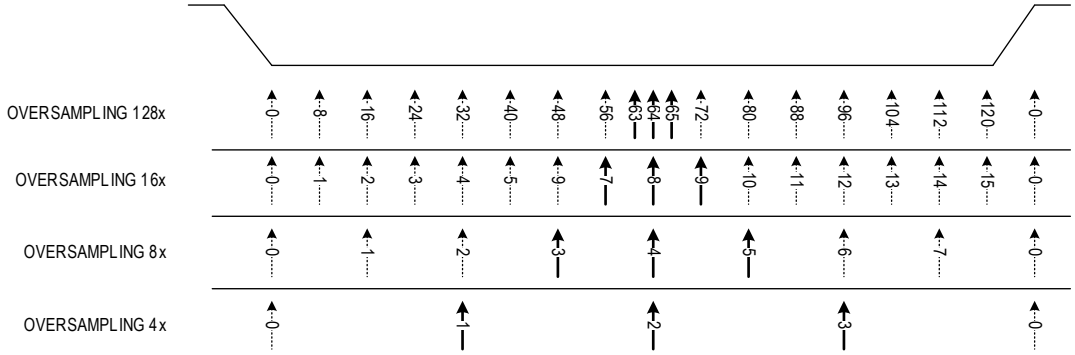
The following conditions result in the UART being inactive:

- When *UARTn_CTRL*.*bclken* = 0
- After setting *UARTn_CTRL*.*bclken* to 1 until *UARTn_CTRL*.*bclkrdy* = 1
- Any write to the *UARTn_CLKDIV*.*clkdiv field* while *UARTn_CTRL*.*bclken* = 1
- Any write to the *UARTn_OSR*.*osr* field when *UARTn_CTRL*.*bclken* = 1

11.7 Receive Sampling

Each bit of a frame is oversampled to improve noise immunity. The oversampling rate (OSR) is configurable with the *UARTn_OSR*.*osr* field. In most cases, the bit is evaluated based on three samples at the midpoint of each bit time, as shown in *Figure 11-4*.

Figure 11-4: Oversampling Example



Whenever *UARTn_CLKDIV.clkdiv* < 0x10 (i.e., division rate less than 8.0), OSR is not used, and the oversampling rate is adjusted to full sampling by the hardware. In full sampling, the receive input is sampled on every clock cycle regardless of the OSR setting.

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

11.8 Baud Rate Generation

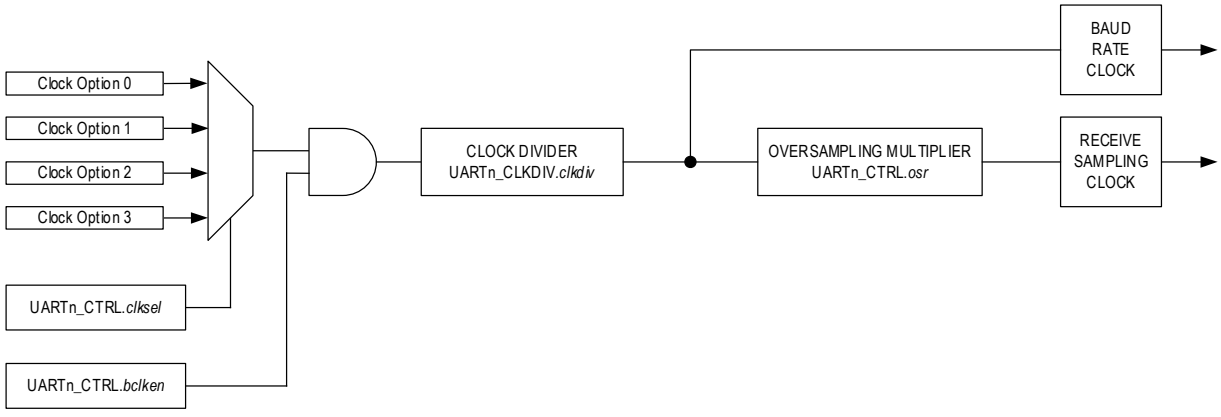
The baud rate is determined by the selected UART clock source and the value of the clock divisor. Multiple clock sources are available for each UART instance. See *Table 11-1* for available clock sources.

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

11.8.1 UART Clock Sources

Standard UART instances operate only in *ACTIVE* and *SLEEP*. Standard UART instances can only wake the device from *SLEEP*. *Figure 11-5* shows the baud rate generation path for standard UARTs.

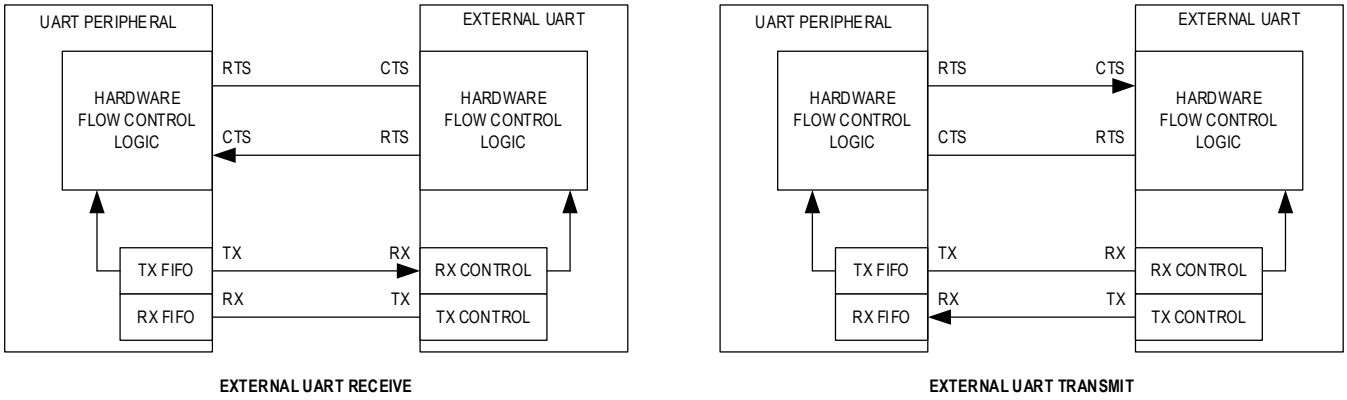
Figure 11-5: UART Baud Rate Generation



11.9 Hardware Flow Control

The optional HFC uses two additional pins, CTS and RTS, as a handshaking protocol to manage UART communications. For full-duplex operation, the RTS output pin on the peripheral is connected to the CTS input pin on the external UART, and the CTS input pin on the peripheral is connected to the RTS output pin on the external UART, as shown in *Figure 11-6*.

Figure 11-6: HFC Physical Connection



In HFC operation, a UART transmitter waits for the external device to assert its CTS pin. When CTS is asserted, the UART transmitter sends data to the external device. The external device keeps CTS asserted until it is unable to receive additional data, typically because the external device's receive FIFO is full. The external device then deasserts CTS until the device can receive more data. The external device then asserts CTS again, allowing additional data to be sent.

HFC can be fully automated by the peripheral hardware or by software through direct monitoring of the CTS input signal and control of the RTS output signal.

11.9.1 Automated HFC

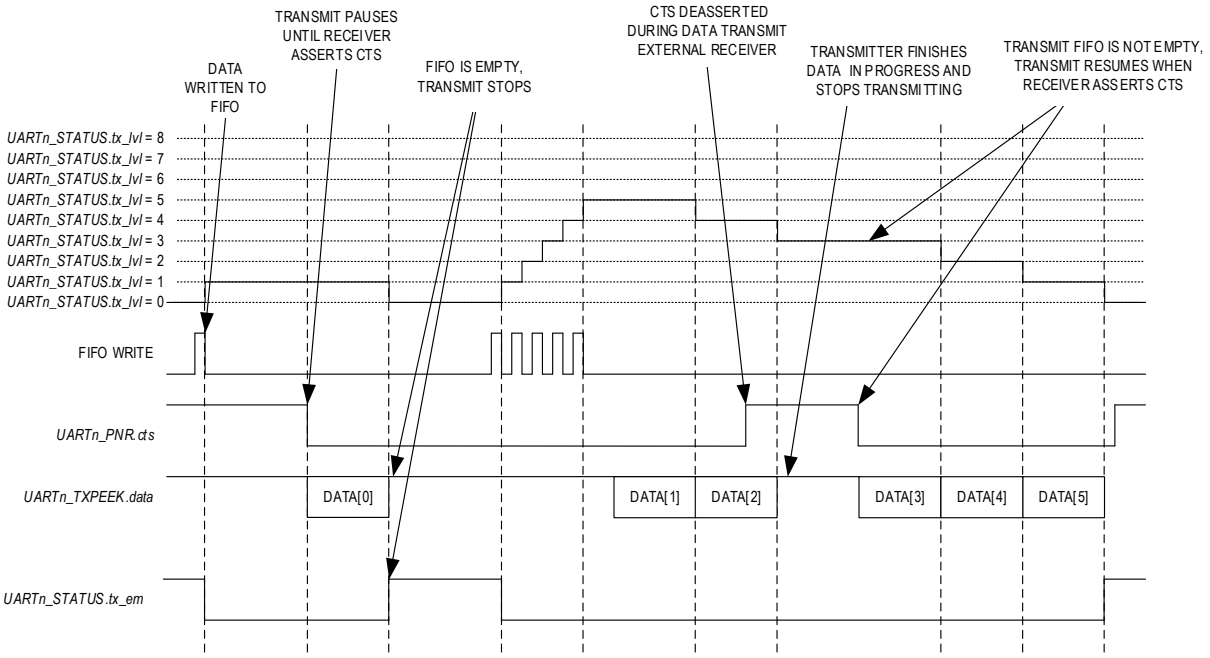
Setting *UARTn_CTRL*.*hfc_en* = 1 enables automated HFC. When automated HFC is enabled, the hardware manages the CTS and RTS signals. The deassertion of the RTS signal is configurable using the *UARTn_CTRL*.*rtsdc* field:

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

The transmitter continues to send data as long as the CTS signal is asserted and there is data in the transmit FIFO. If the receiver deasserts the CTS pin, the transmitter finishes transmitting the current character and then waits until the CTS pin state is asserted before continuing transmission. *Figure 11-7* shows the state of the CTS pin during a transmission under automated HFC.

Automated HFC does not generate interrupt events related to the state of the transmit FIFO or the receive FIFO. The software must handle FIFO management. See *Interrupt Events* for additional information.

Figure 11-7: HFC Signaling for Transmitting to an External Receiver



11.9.2 Software-Controlled HFC

Software-controlled HFC requires the software to manually control the RTS output pin and monitor the CTS input pin. To use the software-controlled HFC, disable the automated HFC by setting the *UARTn_CTRL.hfc_en* field to 1. Additionally, the software should enable CTS sampling (*UARTn_CTRL.cts_dis* = 0) if performing software-controlled HFC.

11.9.2.1 RTC/CTS Handling for Application-Controlled HFC

The software can manually monitor the CTS pin state by reading the field *UARTn_PNR.cts*. The software can manually set the state of the RTS output pin and read the current state of the RTS output pin using the field *UARTn_PNR.rts*. The software must manage the state of the RTS pin when performing software-controlled HFC.

Interrupt support for CTS input signal change events is supported even when automated HFC is disabled. The software can enable the CTS interrupt event by setting the *UARTn_INTEN.cts_ev* field to 1. The CTS signal change interrupt flag is set by the hardware any time the CTS pin state changes. The software must clear this interrupt flag manually by writing 1 to the *UARTn_INTFL.cts_ev* field.

Note: CTS pin state monitoring is disabled any time the UART baud clock is disabled (UARTn_CTRL.bclken = 0). The software must enable CTS pin monitoring by setting the field UARTn_CTRL.cts_dis to 0 after enabling the baud clock if CTS pin state monitoring is required.

11.10 UART Registers

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

defined by replacing "n" with the instance number. For example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

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

Table 11-5: UART/LPUART Register Summary

Offset	Register	Name
[0x0000]	*UARTn_CTRL*	UART Control Register
[0x0004]	*UARTn_STATUS*	UART Status Register
[0x0008]	*UARTn_INTEN*	UART Interrupt Enable Register
[0x000C]	*UARTn_INTFL*	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

11.10.1 Register Details

Table 11-6: UART Control Register

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

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

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

Table 11-7: UART Status Register

UART Status				UARTn_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:16	-	RO	0	**Reserved**	
15:12	tx_lvl	R	0	**Transmit FIFO Level** This field is the number of characters in the transmit FIFO. 0 - 8: Number of bytes in the transmit FIFO. 9 - 15: Reserved.	
11:8	rx_lvl	R	0	**Receive FIFO Level** This field is the number of characters in the receive FIFO. 0 - 8: Number of bytes in the receive FIFO. 9 - 15: Reserved.	
7	tx_full	R	0	**Transmit FIFO Full** 0: Not full. 1: Full.	
6	tx_em	R	1	**Transmit FIFO Empty** 0: Not empty. 1: Empty.	
5	rx_full	R	0	**Receive FIFO Full** 0: Not full. 1: Full.	
4	rx_em	R	1	**Receive FIFO Empty** 0: Not empty. 1: Empty.	
3:2	-	RO	0	**Reserved**	
1	rx_busy	R	0	**Receive Busy** 0: UART is not receiving a character. 1: UART is receiving a character.	
0	tx_busy	R	0	**Transmit Busy** 0: UART is not transmitting data. 1: UART is transmitting data.	

Table 11-8: UART Interrupt Enable Register

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

Table 11-9: UART Interrupt Flag Register

UART Interrupt Flag				UARTn_INTFL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:7	-	RO	0	**Reserved**	
6	tx_he	R/W1C	0	**Transmit FIFO Half-Empty Interrupt Flag**	
5	tx_ae	R/W1C	0	**Transmit FIFO Almost Empty Flag**	
4	rx_thd	R/W1C	0	**Receive FIFO Threshold Interrupt Flag**	
3	rx_ov	R/W1C	0	**Receive FIFO Overrun Interrupt Flag**	
2	cts_ev	R/W1C	0	**CTS Signal Change Interrupt Flag**	
1	rx_par	R/W1C	0	**Receive Parity Error Interrupt Flag**	
0	rx_ferr	R/W1C	0	**Receive Frame Error Interrupt Flag**	

Table 11-10: UART Clock Divisor Register

UART Clock Divisor				UARTn_CLKDIV	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:20	-	RO	0	**Reserved**	
19:0	clkdiv	R/W	0	**Baud Rate Divisor** This field sets the divisor used to generate the baud tick from the baud clock. For LPUART instances, if *UARTn_CTRL.fdm* = 1, the fractional divisors are in increments of 0.5. The over-sampling rate must be no greater than this divisor. See *Baud Rate Generation* for information on how to use this field.	

Table 11-11: UART Oversampling Control Register

UART Oversampling Control				UARTn_OSR	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:3	-	RO	0	**Reserved**	
2:0	osr	R/W	0	**LPUART Over Sampling Rate** For LPUART instances with FDM enabled (*UARTn_CTRL.fdm* = 1): 0: 8 × 1: 12 × 2: 16 × 3: 20 × 4: 24 × 5: 28 × 6: 32 × 7: 36 × For LPUART instances with FDM disabled (*UARTn_CTRL.fdm* = 0): 0: 128 × 1: 64 × 2: 32 × 3: 16 × 4: 8 × 5: 4 × 6 - 7: Reserved	

Table 11-12: UART Transmit FIFO Register

UART Transmit FIFO				UARTn_TXPEEK	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:8	-	RO	0	**Reserved**	
7:0	data	RO	0	**Transmit FIFO Data** Read the transmit FIFO next data without affecting the contents of the transmit FIFO. If there are no entries in the transmit FIFO, this field reads 0. *Note: The parity bit is available from this field.*	

Table 11-13: UART Pin Control Register

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

Table 11-14: UART Data Register

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

UART Data			UARTn_FIFO	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
8	rx_par	R	0	**Receive FIFO Byte Parity** If the parity feature is disabled, this bit always reads 0. If a parity error occurred during the reception of the character at the output end of the receive FIFO (returned by reading the *UARTn_FIFO.data* field), this bit reads 1, otherwise it reads 0.
7:0	data	R/W	0	**Transmit/Receive FIFO Data** Writing to this field loads the next character into the transmit FIFO if the transmit FIFO is not full. Reading from this field returns the next character from the receive FIFO if the receive FIFO is not empty. If the receive FIFO is empty, the hardware returns 0. For character widths less than 8, the unused bit(s) are ignored when the transmit FIFO is loaded, and the unused high bit(s) read 0 on characters read from the receive FIFO.

Table 11-15: UART DMA Register

UART DMA			UARTn_DMA	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	rx_en	0	0	**Receive DMA Channel Enable** 0: Disabled. 1: Enabled.
8:5	rx_thd_val	0	0	**Receive FIFO Level DMA Threshold** If *UARTn_STATUS.rx_lvl* < *UARTn_DMA.rx_thd_val*, then the receive FIFO DMA interface sends a signal to the DMA indicating characters are available in the UART receive FIFO to transfer to memory.
4	tx_en	R/W	0	**Transmit DMA Channel Enable** 0: Disabled. 1: Enabled.
3:0	tx_thd_val	R/W	0	**Transmit FIFO Level DMA Threshold** If *UARTn_STATUS.tx_lvl* < *UARTn_DMA.tx_thd_val*, the transmit DMA channel sends a signal to the DMA indicating the UART transmit FIFO is ready to receive data from memory.

Table 11-16: 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 11-17: 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.

12. Serial Peripheral Interface (SPI)

The SPI peripheral is a configurable, flexible, and efficient synchronous interface between multiple SPI devices on a single bus. The SPI bus uses a single clock signal, single, dual, or quad data lines, and one or more target select lines for communication with external SPI devices.

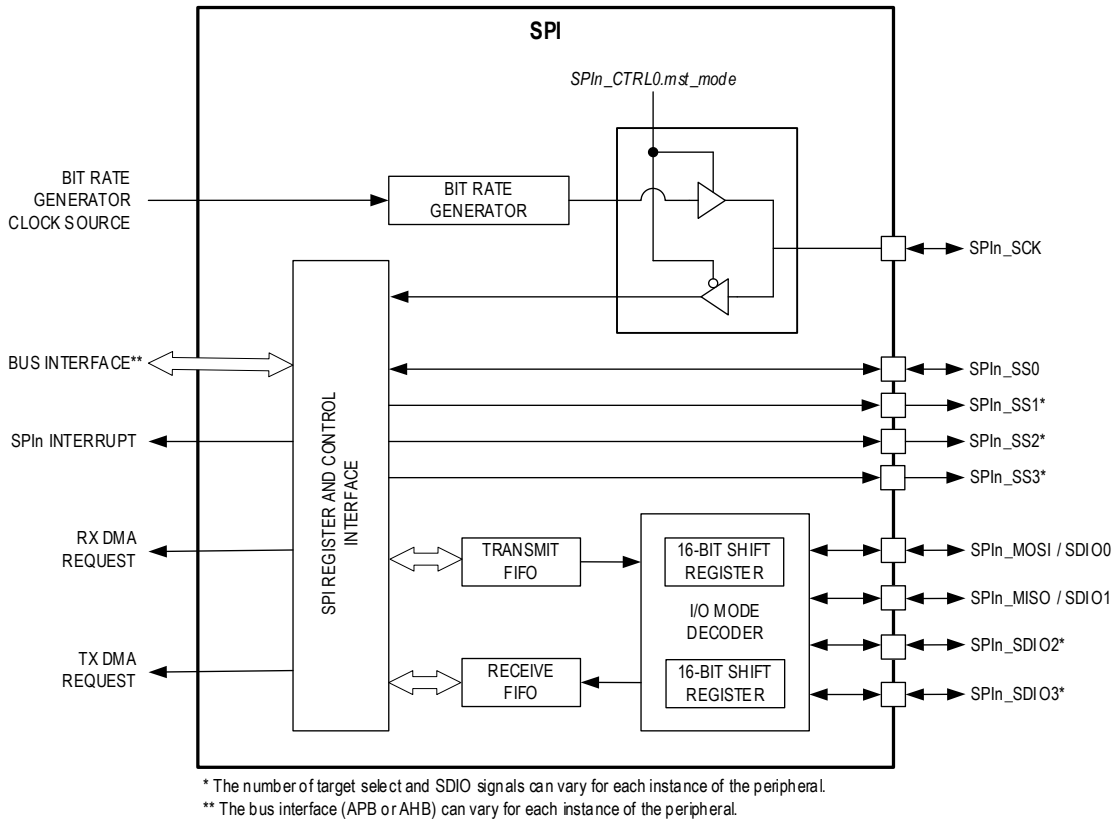
The provided SPI ports support full-duplex, bidirectional I/O, and each SPI includes a bit rate generator (BRG) to generate the clock signal when operating in controller mode. Each SPI port operates independently and requires minimal processor overhead. All instances of the SPI peripheral support both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

Features include:

- Dedicated BRG for precision serial clock generation in controller mode
  - Up to $\frac{f_{PCLK}}{2}$ for instances on the APB bus.
  - Up to $\frac{f_{HCLK}}{2}$ for instances on the AHB bus.
  - Programmable SCK duty cycle timing.
- Full-duplex, synchronous communication of 2 to 16-bit characters
  - 1-bit and 9-bit characters are not supported.
  - 2-bit and 10-bit characters do not support maximum clock speed. *SPIn_CLKCTRL.clkdiv* must be > 0.
- 3-wire and 4-wire SPI operation for single-bit communication.
- Single, Dual, or Quad I/O operation.
- Byte-wide Transmit and Receive FIFOs with 32-byte depth
  - For character sizes greater than 8, each character uses 2 entries per character resulting in 16 entries for the transmit and receive FIFO.
- Transmit and receive DMA support.
- SPI modes 0, 1, 2, 3.
- Configurable target select lines
  - Programmable target select level.
- Programmable target select timing with respect to the SCK starting edge and ending edge.
- Multicontroller mode fault detection.

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

Figure 12-1: SPI Block Diagram



12.1 Instances

There are two instances of the SPI peripheral, as shown in *Table 12-1*. *Table 12-2* lists the locations of the SPI signals for each of the SPI instances.

Table 12-1: MAX32662 SPI Instances

Instance	Formats				Hardware Bus	Bit Rate Generator Clock Source	Target Select Signals
	3-Wire	4-Wire	Dual	Quad			32-TQFN
SPI0	Yes	Yes	No	No	APB	f_{PCLK}	1
SPI1	Yes	Yes	No	No	APB	f_{PCLK}	1

Note: Refer to the MAX32662 data sheet's pin description table for the list of alternate function assignments for each target instance.

Table 12-2: MAX32662 SPI Target Pins

Instance	Signal Description	Alternate Function
SPI0	SPI Clock	SPI0_SCK
	Target Select 0	SPI0_SS0
	MOSI (SDIO0)	SPI0_MOSI
	MISO (SDIO1)	SPI0_MISO
SPI1	SPI Clock	SPI1_SCK
	Target Select 0	SPI1_SS0
	MOSI (SDIO0)	SPI1_MOSI
	MISO (SDIO1)	SPI1_MISO

12.2 Formats

12.2.1 Four-Wire SPI

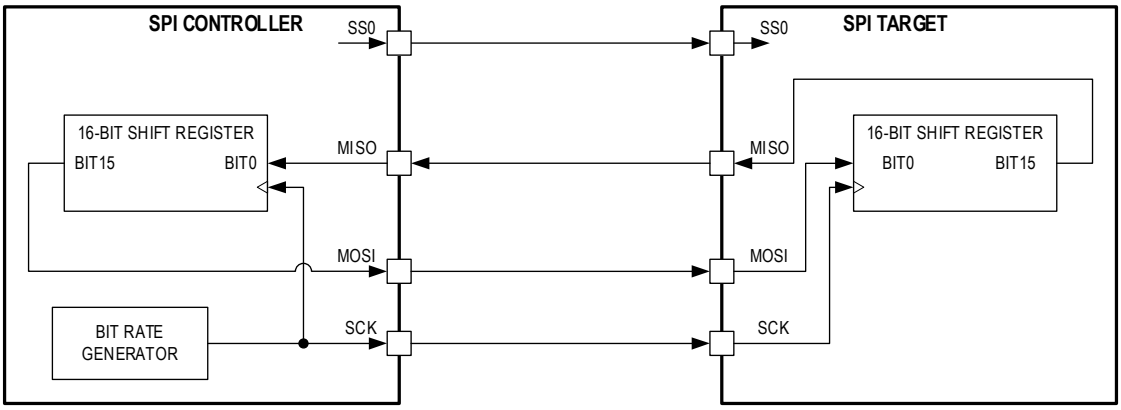
SPI devices operate as either a controller or target device. Four signals are required for communication in four-wire SPI, as shown in *Table 12-3*.

Table 12-3: Four-Wire Format Signals

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

The SPI controller starts communication with a target by asserting the target select output. The controller then starts the SPI clock through the SCK output pin. When a target device's target select pin is deasserted, the target device is required to put the SPI pins in tri-state mode.

Figure 12-2: 4-Wire SPI Connection Diagram



12.2.2 Three-Wire SPI

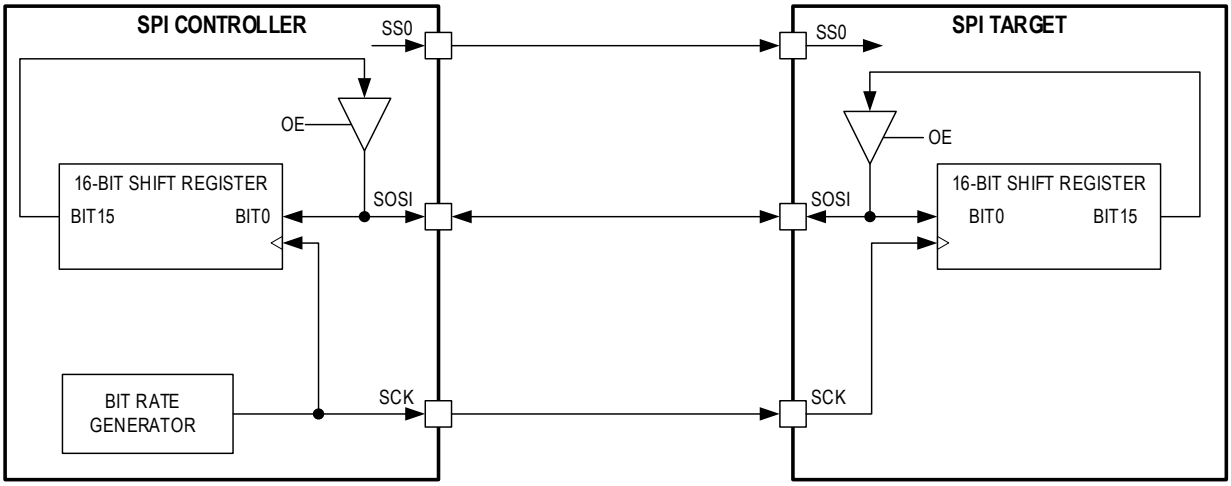
The signals in three-wire SPI operation are shown in *Table 12-4*. The MOSI signal is used as a bidirectional, half-duplex I/O referred to as target input target output (SISO). Three-wire SPI also uses a serial clock signal generated by the controller and a target select pin controlled by the controller.

Table 12-4: Three-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The controller generates the serial clock signal, an output from the controller, and an input to the target.
MOSI	Target Input Target Output	This is a half-duplex, bidirectional I/O pin used for communication between the SPI controller and target. This signal is used to transmit data from the controller to the target and to receive data from the target by the controller.
SS	Target Select	In controller mode, this signal is an output used to select a target device before communication. In target mode, SPIn_SS0 is a dedicated input that indicates an external controller is going to start communication. Other target select signals into the target are ignored in target mode

A three-wire SPI network is shown in *Figure 12-3*. The controller device selects the target device using the target select output. The communication starts with the controller asserting the target select line and then starting the clock (SCK). In three-wire SPI communication, the controller and target must both know the intended direction of the data to prevent bus contention. For a write, the controller drives the data out the SISO pin. For a read, the controller must release the SISO line and let the target drive the SISO line. The direction of transmission is controlled using the FIFO. Writing to the FIFO starts the three-wire SPI write, and reading from the FIFO starts a three-wire SPI read transaction.

Figure 12-3: Generic 3-Wire SPI Controller to Target Connection



12.3 Pin Configuration

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

12.3.1 SPI Alternate Function Mapping

Pin selection and configuration are required to use the SPI port. The following information applies to SPI controller and target operation as well as three-wire, four-wire, dual, and quad mode communications. Determine the pins required for the SPI type and mode in the application, and configure the required GPIO as described in the following sections. Refer to the MAX32662 data sheet for pin availability for a specific package.

When the SPI port is disabled, *SPIn_CTRL0.en* = 0, the GPIO pins enabled for SPI alternate function are placed in high-impedance input mode.

12.3.2 Four-Wire Format Configuration

Four-wire SPI uses SCK, MISO, MOSI, and one or more SS pins. Four-wire SPI may use more than one target select pin for a transaction, resulting in more than four wires total. However, the communication is referred to as four-wire for historical reasons.

Note: Select the pins mapped to the SPI external device in the design and modify the setup accordingly. There is no restriction on which alternate function is used for a specific SPI pin, and each SPI pin can be used independently from the other pins chosen. However, it is recommended that only one set of GPIO port pins be used for any network.

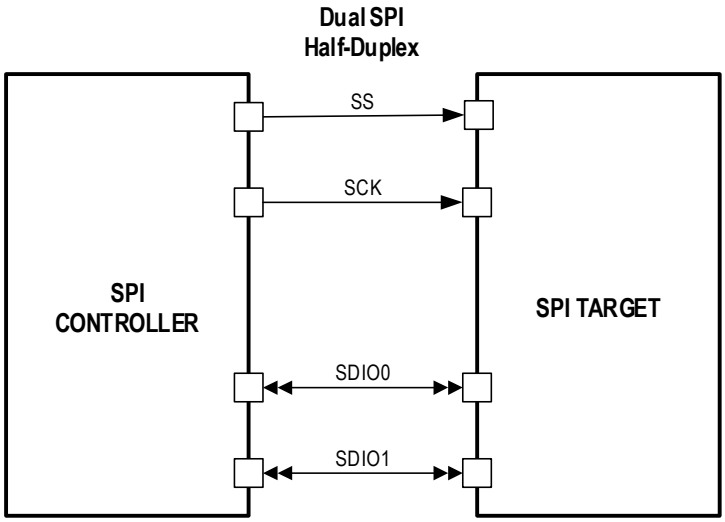
12.3.3 Three-Wire Format Configuration

Three-wire SPI uses SCK, MOSI, and one or more target select pins for an SPI transaction. Three-wire SPI configuration is identical to the four-wire configuration, except SPIn_MISO does not need to be set up for the SPI alternate function. The direction of communication in three-wire SPI mode is controlled by the transmit and receive FIFO enables. Enabling the receive FIFO and disabling the transmit FIFO indicates a read transaction. Enabling the transmit FIFO and disabling the Receive FIFO indicates a write transaction. It is an illegal condition to enable both the transmit and receive FIFOs in three-wire SPI operation.

12.3.4 Dual-Mode Format Configuration

In dual-mode SPI, two I/O pins are used to transmit 2-bits of data per SCK clock cycle. The communication is half-duplex, and the controller and target must know the direction of the data transmission for a given transaction. Dual-mode SPI uses SCK, SDIO0, SDIO1, and one or more target select lines, as shown in *Figure 12-4*. The configuration of the GPIO pins for dual-mode SPI is identical to four-wire SPI, and the mode is controlled by setting *SPIn_CTRL2.data_width* to 1, indicating to the SPI hardware to use SDIO0 and SDIO1 for half-duplex communication rather than full-duplex communication.

Figure 12-4: Dual-Mode SPI Connection Diagram



12.3.5 Quad-Mode Format Pin Configuration

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

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

12.4 Clock Configuration

12.4.1 Serial Clock

The SCK signal synchronizes data movement in and out of the device. The controller drives SCK as an output to the target's SCK pin. When SPI is set to controller mode, the SPI bit rate generator creates the serial clock and outputs it on the configured SPIn_SCK pin. When SPI is configured for target operation, the SPIn_SCK pin is an input from the external controller, and the SPI hardware synchronizes communications using the SCK input. Operating as a target, if an SPI target select input is not asserted, the SPI ignores any signals on the serial clock and serial data lines.

In both the controller and target devices, data is shifted on one edge of the SCK and is sampled on the opposite edge, where data is stable. Data availability and sampling time are controlled using the SPI phase control field, *SPIn_CTRL2.clkpha*. The SCK clock polarity field, *SPIn_CTRL2.clkpol*, controls if the SCK signal is active high or active low.

The SPI peripheral supports four combinations of SCK phase and polarity referred to as SPI modes 0, 1, 2, and 3. Clock Polarity (*SPIn_CTRL2.clkpol*) selects an active low/high clock and has no effect on the transfer format. Clock Phase (*SPIn_CTRL2.clkpha*) selects one of two different transfer formats.

For proper data transmission, the clock phase and polarity must be identical for the SPI controller and target. The controller always places data on the MOSI line a half-cycle before the SCK edge for the target to latch the data. See section *Clock Phase and Polarity Control* for additional details.

12.4.2 Target Clock

See Table 12-1 for the specific input clock, f_{INPUT_CLK}, used for each SPI instance. For SPI instances assigned to the AHB bus, the SPI input clock is the system clock, SYS_CLK. For SPI instances mapped to the APB bus, the SPI input clock is the system peripheral clock, PCLK. The SPI input clock drives the SPI peripheral clock. The SPI provides an internal clock, *SPI_CLK*, used within the SPI peripheral for the base clock to control the module and generate the SCK clock when in controller mode. Set the SPI internal clock using the field *SPIn_CLKCTRL.clkdiv* as shown in Equation 12-1. Valid settings for *SPIn_CLKCTRL.clkdiv* are 0 to 8, allowing a divisor of 1 to 256.

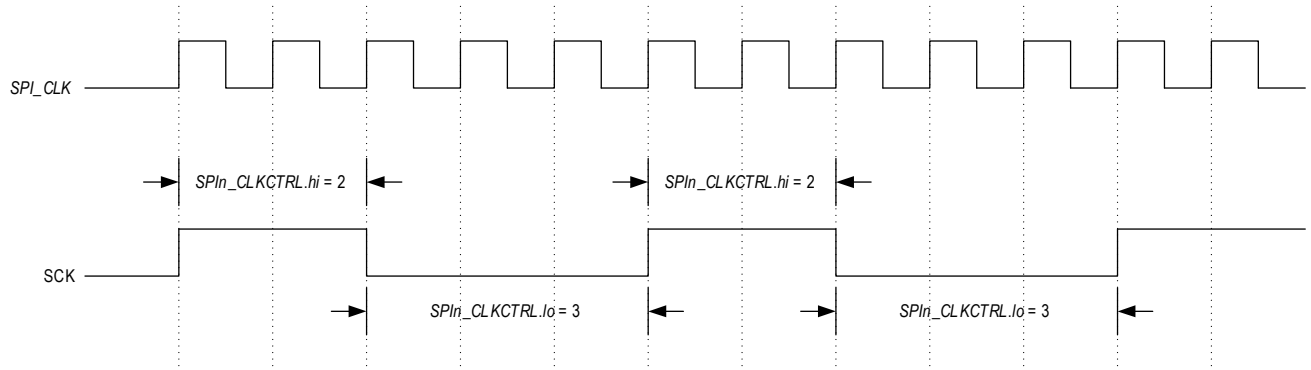
Equation 12-1: SPI Peripheral Clock

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

12.4.3 Controller Mode Serial Clock Generation

In controller and multicontroller mode, the SCK clock is generated by the controller. The SPI peripheral provides control for both the high time and low time of the SCK clock. This control allows setting the high and low times for the SCK to duty cycles other than 50%, if required. The SCK clock uses the SPI target clock as a base value, and the high and low values are a count of the number of f_{SPI_CLK} clocks. Figure 12-5 visually represents the use of the *SPIn_CLKCTRL.hi* and *SPIn_CLKCTRL.lo* fields for a non-50% duty cycle serial clock generation. See Equation 12-2 and Equation 12-3 for calculating the SCK high and low time from the *SPIn_CLKCTRL.hi* and *SPIn_CLKCTRL.lo* field values.

Figure 12-5: SCK Clock Rate Control



Equation 12-2: SCK High Time

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

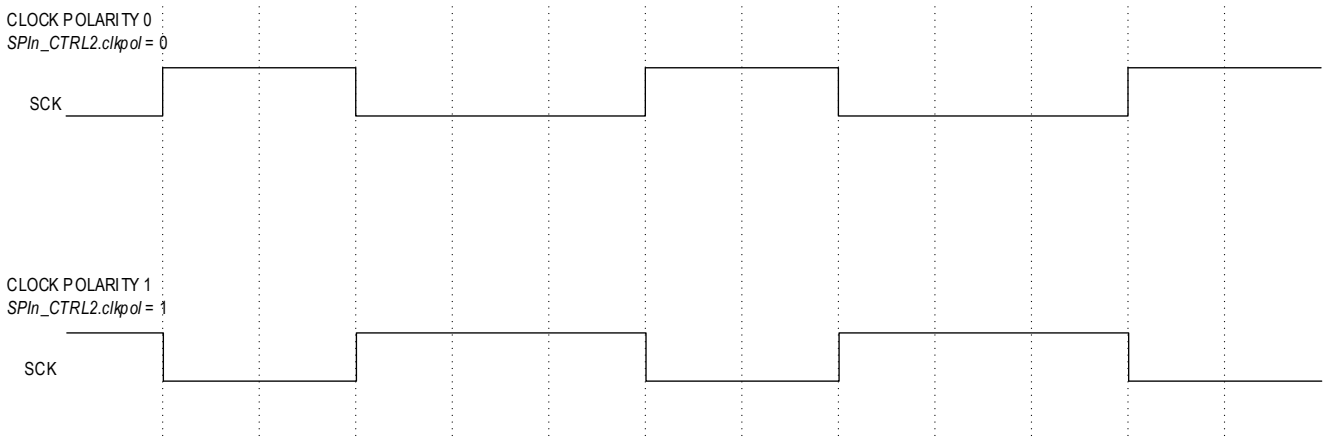
Equation 12-3: SCK Low Time

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

12.4.4 Clock Phase and Polarity Control

SPI supports four combinations of clock and phase polarity, as shown in *Table 12-5*. Clock polarity is controlled using the bit *SPIn_CTRL2.clkpol* and determines if the clock is active high or active low, as shown in *Figure 12-6*. Clock polarity does not affect the transfer format for SPI. The clock phase determines when the data must be stable for sampling. Setting the clock phase to 0, *SPIn_CTRL2.clkpha* = 0, dictates the SPI data is sampled on the initial SPI clock edge regardless of clock polarity. Phase 1, *SPIn_CTRL2.clkpha* = 1, results in data sample occurring on the second edge of the clock regardless of clock polarity.

Figure 12-6: SPI Clock Polarity



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

Table 12-5: SPI Modes Clock Phase and Polarity Operation

SPI Mode	*SPIn_CTRL2 clkpha*	*SPIn_CTRL2 clkpol*	SCK Transmit Edge	SCK Receive Edge	SCK Idle State
0	0	0	Falling	Rising	Low
1	0	1	Rising	Falling	High
2	1	0	Rising	Falling	Low
3	1	1	Falling	Rising	High

12.5 Transmit and Receive FIFOs

The Transmit FIFO hardware is 32 bytes deep. The write data width can be 8-, 16- or 32-bits wide. A 16-bit write queues a 16-bit word to the FIFO hardware. A 32-bit write queues two 16-bit words to the FIFO hardware with the least significant word dequeued first. Bytes must be written to two consecutive byte addresses, with the odd byte as the most significant byte and the even byte as the least significant byte. The FIFO logic waits for both the odd and even bytes to be written to this register space before dequeuing the 16-bit result to the FIFO.

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

12.6 Interrupts and Wakeups

The SPI supports multiple interrupt sources. Status flags for each interrupt are set regardless of the state of the interrupt enable bit for that event. The event happens once when the condition is satisfied. The software must clear the status flag by writing a 1 to the interrupt flag.

The following FIFO interrupts are supported:

- Transmit FIFO Empty.
- Transmit FIFO Threshold.
- Receive FIFO Full.
- Receive FIFO Threshold.
- Transmit FIFO Underrun.
  - Target mode only, controller mode stalls the serial clock.
- Transmit FIFO Overrun.
- Receive FIFO Underrun.
- Receive FIFO Overrun.

  Target mode only, controller mode stalls the serial clock.
- SPI supports interrupts for the internal state of the SPI as well as external signals. The following transmission interrupts are supported:
  - SS asserted or deasserted.
  - SPI transaction complete.
  - Controller mode only.
  - Target mode transaction aborted.
  - Multicontroller fault.

The SPI port can wake up the microcontroller from low-power modes when the wake event is enabled. SPI events that can wake the microcontroller are:

- Receive FIFO full.
- Transmit FIFO empty.
- Receive FIFO threshold.
- Transmit FIFO threshold.

12.7 Registers

See *Table 3-2* for the base address of this target/module. If multiple instances of the target are provided, each instance has its own independent set of registers, shown in *Table 12-6*. Register names for a specific instance are defined by replacing "n" with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 12-6: SPI Register Summary

Offset	Register Name	Description
[0x0000]	*SPIn_FIFO32*	*SPI FIFO Data Register*
[0x0000]	*SPIn_FIFO16*	*SPI 16-bit FIFO Data Register*
[0x0000]	*SPIn_FIFO8*	*SPI 8-bit FIFO Data Register*
[0x0004]	*SPIn_CTRL0*	*SPI Controller Signals Control Register*
[0x0008]	*SPIn_CTRL1*	*SPI Transmit Packet Size Register*
[0x000C]	*SPIn_CTRL2*	*SPI Static Configuration Register*
[0x0010]	*SPIn_SSTIME*	*SPI Target Select Timing Register*

Offset	Register Name	Description
[0x0014]	*SPIn_CLKCTRL*	*SPI Controller Clock Configuration Register*
[0x001C]	*SPIn_DMA*	*SPI DMA Control Register*
[0x0020]	*SPIn_INTFL*	*SPI Interrupt Flag Register*
[0x0024]	*SPIn_INTEN*	*SPI Interrupt Enable Register*
[0x0028]	*SPIn_WKFL*	*SPI Wakeup Flags Register*
[0x002C]	*SPIn_WKEN*	*SPI Wakeup Enable Register*
[0x0030]	*SPIn_STAT*	*SPI Status Register*

12.7.1 Register Details

Table 12-7: SPI FIFO32 Register

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

Table 12-8: SPI 16-bit FIFO Register

SPI FIFO Data			SPIn_FIFO16	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15:0	data	R/W	0	**SPI 16-bit FIFO Data Register** This register is used for the SPI Transmit and Receive FIFO. Reading from this register returns characters from the Receive FIFO, and writing to this register adds characters to the Transmit FIFO. Read and write this register in 2-byte width only for 16-bit FIFO access. Reading from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 12-9: SPI 8-bit FIFO Register

SPI 8-bit FIFO Data			SPIn_FIFO8	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**Reserved**
7:0	data	R/W	0	**SPI 8-bit FIFO Data Register** This register is used for the SPI Transmit and Receive FIFO. Reading from this register returns characters from the Receive FIFO, and writing to this register adds characters to the Transmit FIFO. Read and write this register in 1-byte width only for 8-bit FIFO access. Reading from an empty FIFO or writing to a full FIFO results in undefined behavior.

Table 12-10: SPI Control 0 Register

SPI Control 0			SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved**

SPI Control 0			SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
19:16	ss_active	R/W	0	**Controller Target Select** The SPI includes up to four target select lines for each port. This field selects which target select pin is active when the next SPI transaction is started (*SPIn_CTRL0.start* = 1). One or more target select pins can be selected for each SPI transaction by setting the bit for each target select pin. For example, use SPIn_SS0 and SPIn_SS2 by setting this field to 0b0101 or select all target selects by setting this field to 0b1111. *Note: This field is only used when the SPI is configured for controller mode (SPIn_CTRL0.mst_mode = 1).*
15:9	-	R/W	0	**Reserved**
8	ss_ctrl	R/W	0	**Controller Target Select Control** This field controls the behavior of the target select pins at the completion of a transaction. The default behavior, *ss_ctrl* = 0, deasserts the target select pin at the completion of the transaction. Set this field to 1 to leave the target select pins asserted at the completion of the transaction. If the external device supports this behavior, leaving the target select pins asserted allows multiple transactions without the delay associated with deassertion of the target select pin between transactions. 0: Target Select is deasserted at the end of a transmission. 1: Target Select stays asserted at the end of a transmission.
7:6	-	R/W	0	**Reserved.**
5	start	R/W1O	0	**Controller Start Data Transmission** Set this field to 1 to start an SPI controller mode transaction. 0: No controller mode transaction active. 1: Initiate the data transmission. Ensure that all pending transactions are complete before setting this field to 1. *Note: This field is only used when the SPI is configured for controller mode (SPIn_CTRL0.mst_mode = 1).*
4	ss_io	R/W	0	**Controller Target Select Signal Direction** Set the I/O direction for 0: Target select is an output. 1: Target select is an input. *Note: This field is only used when the SPI is configured for controller mode (SPIn_CTRL0.mst_mode = 1).*
3:2	-	R/W	0	**Reserved**
1	mst_mode	R/W	0	**SPI Controller Mode Enable** This field selects between target mode and controller mode operation for the SPI port. Write this field to 0 to operate as an SPI target. Set this field to 1 to set the port as an SPI controller. 0: Target mode SPI operation. 1: Controller mode SPI operation.
0	en	R/W	0	**SPI Enable/Disable** This field enables and disables the SPI port. Disable the SPI port by setting this field to 0. Disabling the SPI port does not affect the SPI FIFOs or register settings. Access to SPI registers is always available. 0: SPI port is disabled. 1: SPI port is enabled.

Table 12-11: SPI Control 1 Register

SPI Transmit Packet Size			SPIn_CTRL1	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	rx_num_char	R/W	0	**Number of Receive Characters** This field sets the number of characters to receive in the receive FIFO. *Note: If the SPI port is set to four-wire mode, this field is ignored, and the SPIn_CTRL1.tx_num_char field is used for both the number of characters to receive and transmit.*
15:0	tx_num_char	R/W	0	**Number of Transmit Characters** This field sets the number of characters to transmit from transmit FIFO. *Note: If the SPI port is set to four-wire mode, this field is used to receive and transmit the number of characters.*

Table 12-12: SPI Control 2 Register

SPI Control 2			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved**
19:16	ss_pol	R/W	0	**Target Select Polarity** Controls the polarity of each individual SS signal where each bit position corresponds to a SS signal. SPIn_SS0 is controlled with bit position 0, and SPIn_SS2 is controlled with bit position 2. For each bit position: 0: SS is active low. 1: SS is active high.
15	three_wire	R/W	0	**Three-Wire SPI Enable** Set this field to 1 to enable three-wire SPI communication. Set this field to 0 for four-wire full-duplex SPI communication. 0: Four-wire full-duplex mode enabled. 1: Three-wire mode enabled. *Note: This field is ignored for Dual SPI, SPIn_CTRL2.data_width =1, and Quad SPI, SPIn_CTRL2.data_width =2.*
14	-	R/W	0	**Reserved**

SPI Control 2			SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
13:12	data_width	R/W	0b00	**SPI Data Width** This field controls the number of data lines used for SPI communications. *Three-wire SPI*: *data_width* = 0. Set this field to 0, indicating SPIn_MOSI is used for half-duplex communication. *Four-wire full-duplex SPI*: *data_width* = 0. Set this field to 0, indicating SPIn_MOSI and SPIn_MISO are used for the SPI data output and input, respectively. *Dual-mode SPI*: *data_width* = 1. Set this field to 1, indicating SPIn_SDIO0 and SPIn_SDIO1 are used for half-duplex communication. *Quad-mode SPI*: *data_width* = 2. Set this field to 2, indicating SPIn_SDIO0, SPIn_SDIO1, SPIn_SDIO2, and SPIn_SDIO3 are used for half-duplex communication. 0: 1-bit per SCK cycle (Three-wire half-duplex SPI and Four-wire full-duplex SPI). 1: 2-bits per SCK cycle (Dual-mode SPI). 2: 4-bits per SCK cycle (Quad-mode SPI). 3: Reserved. *Note: When this field is set to 0, use the field SPIn_CTRL2.three_wire to select either Three-Wire SPI or Four-Wire SPI operation.*
11:8	numbits	R/W	0	**Number of Bits per Character** Set this field to the number of bits per character for the SPI transaction. Setting this field to 0 indicates a character size of 16. 0: 16-bits per character. 1: 1-bit per character (not supported). 2: 2-bits per character. … 14: 14-bits per character. 15: 15-bits per character. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: 2-bit and 10-bit character lengths do not support maximum SCK speeds in controller mode. SPIn_CLKCTRL.clkdiv must be > 0.* *Note: For Dual and Quad mode SPI, the character size should be divisible by the number of bits per SCK cycle.*
7:2	-	R/W	0	**Reserved**
1	clkpol	R/W	0	**Clock Polarity** This field controls the SCK polarity. The default clock polarity is for SPI mode 0 and mode 1 operation, and is active high. Invert the SCK polarity for SPI mode 2 and mode 3 operation. 0: Standard SCK for use in SPI mode 0 and mode 1. 1: Inverted SCK for use in SPI mode 2 and mode 3.
0	clkpha	R/W	0	**Clock Phase** 0: Data sampled on clock rising edge. Use when in SPI mode 0 and mode 2. 1: Data sampled on clock falling edge. Use when in SPI mode 1 and mode 3.

Table 12-13: SPI Target Select Timing Register

SPI Target Select Timing			SPIn_SSTIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
31:24	-	R/W	0	**Reserved**

SPI Target Select Timing			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 (target select inactive) and the start of the next transaction (target select active). 0: 256. 1: 1. 2: 2. 3:3. … : … 254: 254. 255: 255. *Note: The SPIn_SSTIME register bit settings only apply when SPI is operating in controller mode (SPIn_CTRL0.mst_mode = 1)*
15:8	post	R/W	0	**Target Select Hold Post Last SCK** Set this field to the number of system clock cycles for SS to remain active after the last SCK edge. 0: 256. 1: 1. 2: 2. 3:3. … : … 254: 254. 255: 255. *Note: The SPIn_SSTIME register bit settings only apply when SPI is operating in controller mode (SPIn_CTRL0.mst_mode = 1)*
7:0	pre	R/W	0	**Target Select Delay to First SCK** Set the number of system clock cycles the target select is held active before the first SCK edge. 0: 256. 1: 1. 2: 2. 3:3. … : … 254: 254. 255: 255. *Note: The SPIn_SSTIME register bit settings only apply when SPI is operating in controller mode (SPIn_CTRL0.mst_mode = 1)*

Table 12-14: SPI Controller Clock Configuration Registers

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

SPI Controller Clock Configuration			SPIn_CLKCTRL	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**
19:16	clkdiv	R/W	0	**SPI Target Clock Scale** Scales the SPI input clock (PCLK) by 2^{clkdiv} to generate the SPI target clock. $f_{SPInCLK} = \frac{f_{SPIn\_INPUT\_CLK}}{2^{clkdiv}}$ Valid values for scale are 0 to 8 inclusive. Values greater than 8 are reserved. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*
15:8	hi	R/W	0	**SCK Hi Clock Cycles Control** 0: Hi duty cycle control disabled. Only valid if *SPIn_CLKCTRL.clkdiv* = 0. 1 - 15: The number of SPI target clocks, $f_{SPInCLK}$, that SCK is high. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*
7:0	lo	R/W	0	**SCK Low Clock Cycles Control** This field controls the SCK low clock time and is used to control the overall SCK duty cycle in combination with the *SPIn_CLKCTRL.hi* field. 0: Low duty cycle control disabled. Setting this field to 0 is only valid if *SPIn_CLKCTRL.clkdiv* = 0. 1 to 15: The number of SPI target clocks, $f_{SPInCLK}$, that the SCK signal is low. *Note: 1-bit and 9-bit character lengths are not supported.* *Note: If SPIn_CLKCTRL.clkdiv = 0, SPIn_CLKCTRL.hi = 0, and SPIn_CLKCTRL.lo = 0, character sizes of 2 and 10 bits are not supported.*

Table 12-15: SPI DMA Control Registers

SPI DMA Control			SPIn_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31	dma_rx_en	R/W	0	**Receive DMA Enable** 0: Disabled. Any pending DMA requests are cleared. 1: Enabled.
30:24	dma_rx_en	R	0	**Number of Bytes in the Receive FIFO** Read returns the number of bytes currently in the receive FIFO.
23	rx_flush	R/W1O	-	**Clear the Receive FIFO** 1: Clear the receive FIFO and any pending receive FIFO flags in *SPIn_INTFL*. This should be done when the receive FIFO is inactive. Note: Writing a 0 has no effect.
22	rx_fifo_en	R/W	0	**Receive FIFO Enabled** 0: Disabled. 1: Enabled.
21	-	R/W	0	**Reserved**
20:16	rx_thd_val	R/W	0	**Receive FIFO Threshold Level** Set this value to the desired receive FIFO threshold level. When the receive FIFO level crosses above this setting, a DMA request is triggered if enabled (*SPIn_DMA.tx_en* = 1), and *SPIn_INTFL.rx_thd* is set. Valid values are 0 to 30. *Note: 31 is an invalid setting.*

SPI DMA Control			SPIn_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
15	dma_tx_en	R/W	0	**Transmit DMA Enable** 0: Disabled. Any pending DMA requests are cleared. 1: Transmit DMA is enabled.
14:8	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: Disable the transmit FIFO (SPIn_DMA.tx_fifo_en = 0) before setting this field.* *Note: Setting this field to 0 has no effect.*
6	tx_fifo_en	R/W	0	**Transmit FIFO Enabled** 0: Disabled. 1: Enabled.
5	-	R/W	0	**Reserved**
4:0	tx_thd_val	R/W	0x10	**Transmit FIFO Threshold Level** Set this value to the desired transmit FIFO threshold level. When the transmit FIFO count (*SPIn_DMA.tx_lvl*) falls below this value, a DMA request is triggered if enabled (*SPIn_DMA.tx_en* = 1), and *SPIn_INTFL.tx_thd* becomes set.

Table 12-16: SPI Interrupt Status Flags Registers

SPI Interrupt Status Flags			SPIn_INTFL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15	rx_un	R/1	0	**Receive FIFO Underrun Flag** Set when a read is attempted from an empty receive FIFO.
14	rx_ov	R/W1C	0	**Receive FIFO Overrun Flag** Set if SPI is in target mode, and a write to a full receive FIFO is attempted. If the SPI is in controller mode, this bit is not set as the SPI stalls the clock until data is read from the receive FIFO.
13	tx_un	R/W1C	0	**Transmit FIFO Underrun Flag** Set if SPI is in target mode, and a read from empty transmit FIFO is attempted. If SPI is in controller mode, this bit is not set as the SPI stalls the clock until data is written to the empty transmit FIFO.
12	tx_ov	R/W1C	0	**Transmit FIFO Overrun Flag** Set when a write is attempted, and the transmit FIFO is full.
11	mst_done	R/W1C	0	**Controller Data Transmission Done Flag** Set if SPI is in controller mode and all transactions are complete. *SPIn_CTRL1.tx_num_char* has been reached.
10	-	R/W	0	**Reserved**
9	abort	R/W1C	0	**Target Mode Transaction Abort Detected Flag** Set if the SPI is in target mode, and SS is deasserted before a complete character is received.

SPI Interrupt Status Flags			SPIn_INTFL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
8	fault	R/W1C	0	**Multicontroller Fault Flag** Set if the SPI is in controller mode, multicontroller mode is enabled, and a target select input is asserted. A collision also sets this flag.
7:6	-	R/W	0	**Reserved**
5	ssd	R/W1C	0	**Target Select Deasserted Flag**
4	ssa	R/W1C	0	**Target Select Asserted Flag**
3	rx_full	R/W1C	0	**Receive FIFO Full Flag**
2	rx_thd	R/W1C	0	**Receive FIFO Threshold Level Crossed Flag** Set when the receive FIFO exceeds the value in *SPIn_DMA.rx_lvl*. Cleared once receive FIFO level drops below *SPIn_DMA.rx_lvl.*
1	tx_em	R/W1C	1	**Transmit FIFO Empty Flag** This field is set to 1 by hardware if the transmit FIFO is empty.
0	tx_thd	R/W1C	0	**Transmit FIFO Threshold Level Crossed Flag** This field is set to 1 by hardware when the transmit FIFO is less than the value in *SPIn_DMA.tx_lvl*. This field is cleared by hardware once transmit FIFO level exceeds *SPIn_DMA.tx_lvl.*

Table 12-17: SPI Interrupt Enable Registers

SPI Interrupt Enable			SPIn_INTEN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15	rx_un	R/W	0	**Receive FIFO Underrun Interrupt Enable** 0: Disabled. 1: Enabled.
14	rx_ov	R/W	0	**Receive FIFO Overrun Interrupt Enable** 0: Disabled. 1: Enabled.
13	tx_un	R/W	0	**Transmit FIFO Underrun Interrupt Enable** 0: Disabled. 1: Enabled.
12	tx_ov	R/W	0	**Transmit FIFO Overrun Interrupt Enable** 0: Disabled. 1: Enabled.
11	mst_done	R/W	0	**Controller Data Transmission Done Interrupt Enable** 0: Disabled. 1: Enabled.
10	-	R/W	0	**Reserved**
9	abort	R/W	0	**Target Mode Abort Detected Interrupt Enable** 0: Disabled. 1: Enabled.
8	fault	R/W	0	**Multicontroller Fault Interrupt Enable** 0: Disabled. 1: Enabled.
7:6	-	R/W	0	**Reserved**

SPI Interrupt Enable			SPIn_INTEN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
5	ssd	R/W	0	**Target Select Deasserted Interrupt Enable** 0: Disabled. 1: Enabled.
4	ssa	R/W	0	**Target Select Asserted Interrupt Enable** 0: Disabled. 1: Enabled.
3	rx_full	R/W	0	**Receive FIFO Full Interrupt Enable** 0: Disabled. 1: Enabled.
2	rx_thd	R/W	0	**Receive FIFO Threshold Level Crossed Interrupt Enable** 0: Disabled. 1: Enabled.
1	tx_em	R/W	0	**Transmit FIFO Empty Interrupt Enable** 0: Disabled. 1: Enabled.
0	tx_thd	R/W	0	**Transmit FIFO Threshold Level Crossed Interrupt Enable** 0: Disabled. 1: Enabled.

Table 12-18: SPI Wakeup Status Flags Registers

SPI Wakeup Flags			SPIn_WKFL	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved**
3	rx_full	R/W1C	0	**Wake on Receive FIFO Full Flag** 0: Normal operation. 1: Wake condition occurred.
2	rx_thd	R/W1C	0	**Wake on Receive FIFO Threshold Level Crossed Flag** 0: Normal operation. 1: Wake condition occurred.
1	tx_em	R/W1C	0	**Wake on Transmit FIFO Empty Flag** 0: Normal operation. 1: Wake condition occurred.
0	tx_thd	R/W1C	0	**Wake on Transmit FIFO Threshold Level Crossed Flag** 0: Normal operation. 1: Wake condition occurred.

Table 12-19: SPI Wakeup Enable Registers

SPI Wakeup Enable			SPIn_WKEN	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved**
3	rx_full	R/W	0	**Wake On Receive FIFO Full Enable** 0: Wake event is disabled. 1: Wake event is enabled.
2	rx_thd	R/W	0	**Wake On Receive FIFO Threshold Level Crossed Enable** 0: Wake event is disabled. 1: Wake event is enabled.

SPI Wakeup Enable			SPIn_WKEN	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
1	tx_em	R/W	0	**Wake On Transmit FIFO Empty Enable** 0: Wake event is disabled. 1: Wake event is enabled.
0	tx_thd	R/W	0	**Wake On Transmit FIFO Threshold Level Crossed Enable** 0: Wake event is disabled. 1: Wake event is enabled.

Table 12-20: SPI Target Select Timing Registers

SPI Status			SPIn_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	R/W	0	**Reserved**
0	busy	R	0	**SPI Active Status** This field returns the SPI status. 0: SPI is not active. In controller mode, the *busy* flag is cleared when the last character is sent. In target mode, the *busy* field is cleared when the configured target select input is deasserted. 1: SPI is active. In controller mode, the *busy* flag is set when a transaction starts. In target mode, the *busy* flag is set when a configured target select input is asserted. *Note: SPIn_CTRL0, SPIn_CTRL1, SPIn_CTRL2, SPIn_SSTIME, and* SPIn_CLKCTRL *should not be configured if this bit is set.*

13. I²C Controller/Target Serial Communications Peripheral

The I²C peripherals can be configured as either an I²C controller or an I²C target at standard data rates. For simplicity, I2Cn is used throughout this section to refer to any of the I²C peripherals.

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

13.1 I²C Controller/Target Features

Each I²C controller/target is compliant with the I²C Bus Specification and includes the following features:

- Communicates through a serial data bus (SDA) and a serial clock line (SCL).
- Operates as either a controller or target device as a transmitter or receiver.
- Supports I²C Standard Mode, Fast Mode, Fast Mode Plus, and High Speed (Hs) Mode.
- Transfers data at rates up to:
  - 100kbps in Standard Mode.
  - 400kbps in Fast Mode.
  - 1Mbps in Fast Mode Plus.
  - 3.4Mbps in Hs Mode.
- Supports multicontroller systems, including support for arbitration and clock synchronization for Standard Mode, Fast Mode, and Fast Mode Plus.
- Supports 7- and 10-bit addressing.
- Supports RESTART condition.
- Supports clock stretching.
- Provides transfer status interrupts and flags.
- Provides DMA data transfer support.
- Supports I²C timing parameters fully controllable through software.
- Provides glitch filter and Schmitt trigger hysteresis on SDA and SCL.
- Provides control, status, and interrupt events for maximum flexibility.
- Provides independent 8-byte receive FIFO and 8-byte transmit FIFO.
- Provides transmit FIFO preloading.
- Provides programmable interrupt threshold levels for the transmit and receive FIFO.

13.2 Instances

The two instances of the peripheral are shown in *Table 13-1*. The table lists the alternate function names of the SDA and SCL signals for each of the I²C peripherals.

Table 13-1: MAX32662 I²C Peripheral Pins

I²C Instance	Alternate Function y = Alternate Function Number (A = AF1, B = AF2, C = AF2, D = AF3, E = AF4)*
I2C0	I2C0y_SCL
	I2C0y_SDA
I2C1	I2C1y_SCL
	I2C1y_SDA
** Refer to the device data sheet for alternate function and port pin mapping. Not all peripherals are available in all packages.*	

13.3 I²C Overview

13.3.1 I²C Bus Terminology

Table 13-2 contains terms and definitions used in this chapter for the I²C bus terminology.

Table 13-2: I²C Bus Terminology

Term	Definition
Transmitter	The device sending data on the bus.
Receiver	The device receiving data from the bus.
Controller	The device that initiates a transfer, generates the clock signal, and terminates a transfer.
Target	The device addressed by a controller.
Multicontroller	More than one controller can attempt to control the bus simultaneously without corrupting the message.
Arbitration	Procedure to ensure that, if more than one controller simultaneously tries to control the bus, only one can do so, and the resulting message is not corrupted.
Synchronization	The procedure to synchronize the clock signals of two or more devices.
Clock Stretching	When a target device holds SCL low to pause a transfer until it is ready. Clock stretching is an optional feature according to the I²C specification; thus, a controller does not have to support target clock stretching if none of the targets in the system are capable of clock stretching.

13.3.2 I²C Transfer Protocol Operation

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

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

13.3.3 START and STOP Conditions

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

13.3.4 Controller Operation

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

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

13.3.5 Acknowledge and Not Acknowledge

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

A Not Acknowledge (NACK) occurs if the receiver does not generate an ACK when the transmitter releases SDA. A NACK is generated by allowing SDA to float high during the acknowledge time slot. The I^2C controller can then either generate a STOP condition to abort the transfer or generate a repeated START condition (i.e., send a START condition without an intervening STOP condition) to start a new transfer.

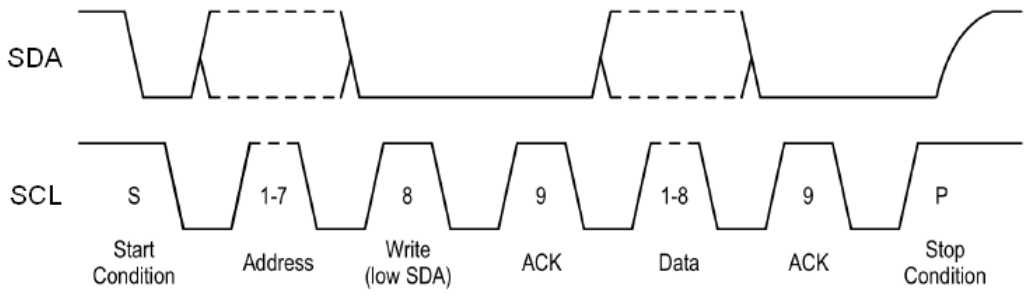
A receiver can generate a NACK after a byte transfer if any of the following conditions occur:

- No receiver is present on the bus with the transmitted address. In that case, no device responds with an acknowledge signal.
- The receiver cannot receive or transmit because it is busy and is not ready to start communication with the controller.
- During the transfer, the receiver receives data or commands it does not understand.
- During the transfer, the receiver is unable to receive any more data.
- If an I^2C controller has requested data from a target, it signals the target to stop transmitting by sending a NACK following the last byte it requires.

13.3.6 Bit Transfer Process

Both SDA and SCL circuits are open-drain, bidirectional circuits. Each requires an external pullup resistor that ensures each circuit is high when idle. The I^2C specification states that during data transfer, the SDA line can change state only when SCL is low, that the SDA is stable, and can be read when SCL is high, as shown in *Figure 13-1*.

Figure 13-1: I^2C Write Data Transfer



An example of an I²C data transfer is as follows:

1. A bus controller indicates a data transfer to a target with a START condition.
2. The controller then transmits one byte with a 7-bit target address and a single read-write bit: a zero for a write or a one for a read.
3. During the next SCL clock following the read-write bit, the controller releases SDA. During this clock period, the addressed target responds with an ACK by pulling SDA low.
4. The controller senses the ACK condition and begins transferring data. If reading from the target, it floats SDA and allows the target to drive SDA to send data. After each byte, the controller drives SDA low to acknowledge the byte. If writing to the target, the controller drives data on the SDA circuit for each of the 8 bits of the byte and then floats SDA during the ninth bit to allow the target to reply with the ACK indication.
5. After the last byte is transferred, the controller indicates the transfer is complete by generating a STOP condition. A STOP condition is generated when the controller pulls SDA from low to high while SCL is high.

13.4 Configuration and Usage

13.4.1 SCL and SDA Bus Drivers

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

13.4.2 SCL Clock Configurations

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

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

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

The controller generates the I²C clock on the SCL line. When operating as a controller, the software must configure the *I2Cn_CLKHI* and *I2Cn_CLKLO* registers for the desired I²C operating frequency.

The SCL high time is configured in the I²C Clock High Time register field *I2Cn_CLKHI.hi* using *Equation 13-2*. The SCL low time is configured in the I²C Clock Low Time register field *I2Cn_CLKLO.lo* using *Equation 13-3*. Each of these fields is 8 bits. The I²C frequency value is shown in *Equation 13-1*.

Equation 13-1: I²C Clock Frequency

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

Equation 13-2: I²C Clock High Time Calculation

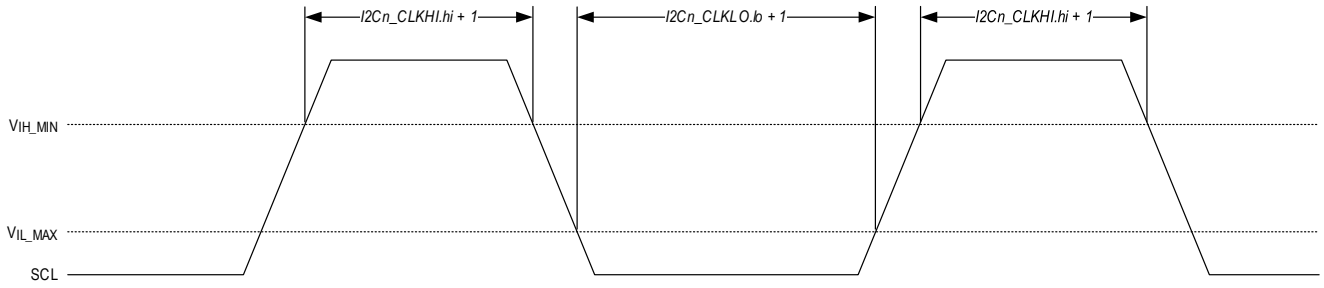
$$t_{SCL\_HI} = t_{I2C\_CLK} \times (I2Cn\_CLKHI.hi + 1)$$

Equation 13-3: I²C Clock Low Time Calculation

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

Figure 13-2 shows the association between the SCL clock low and high times for Standard Mode, Fast Mode, and Fast Mode Plus I²C frequencies.

Figure 13-2: I²C SCL Timing for Standard, Fast and Fast-Plus Modes



During synchronization, external controllers or external targets can drive SCL simultaneously. This affects the SCL duty cycle. By monitoring SCL, the controller determines if an external controller or target is holding SCL low. In either case, the controller waits until SCL is high before starting to count the number of SCL high cycles. Similarly, if an external controller pulls SCL low before the controller has finished counting SCL high cycles, the controller starts counting SCL low cycles and releases SCL once the time period, *I2Cn_CLKLO.lo*, has expired.

Because the controller does not start counting the high/low time until the input buffer detects the new value, the actual clock behavior is based on many factors, including bus loading, other devices on the bus holding SCL low, and the filter delay time of this device.

13.4.4 SCL Clock Generation for Hs-Mode

The values programmed into the *I2Cn_HSCLK.lo* register and *I2Cn_HSCLK.hi* register must be determined to operate the I²C interface in Hs-Mode at its maximum speed (~3.4MHz). Since the Hs-Mode operation is entered by first using one of the lower speed modes for a preamble, configure the lower speed mode as well. See *SCL Clock Generation for Standard, Fast and Fast-Plus Modes* for information regarding the configuration of lower speed modes.

13.4.4.1 Hs-Mode Timing

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

t_{LOW_MIN} = 160ns, the minimum low time for the I²C bus clock.

t_{HIGH_MIN} = 60ns, the minimum high time for the I²C bus clock.

t_{rCL_MAX} = 40ns, the maximum rise time of the I²C bus clock.

t_{fCL_MAX} = 40ns, the maximum fall time of the I²C bus clock.

13.4.4.2 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 13-4: I²C Target SCL Frequency

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

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

Equation 13-5: Determining the I2Cn_HSCLK.lo Register Value

$$I2Cn\_HSCLK.lo = MAX\left\{\left|\left(\frac{t_{LOW\_MIN} + t_{FCL\_MAX} + t_{I2C\_CLK} - t_{AF\_MIN}}{t_{I2C\_CLK}}\right)\right| - 1, \quad \frac{t_{SCL}}{t_{I2C\_CLK}} - 1\right\}$$

Equation 13-6: Determining the I2Cn_HSCLK.hi Register Value

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

Equation 13-7: The Calculated Frequency of the I²C Bus Clock Using the Results of Equation 13-5 *and* Equation 13-6

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

Table 13-3 shows the I²C bus clock calculated frequencies given different f_{SYS_CLK} frequencies.

Table 13-3: Calculated I²C Bus Clock Frequencies

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

13.4.5 Controller Mode Addressing

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

Table 13-4: I²C Target Address Format

Target Address Bits		R/W Bit	Description
0b0000	0b000	0	General Call Address
0b0000	0b000	1	START Condition
0b0000	0b001	*x*	CBUS Address
0b0000	0b010	*x*	Reserved for different bus format
0b0000	0b011	*x*	Reserved for future purposes
0b0000	0b1*xx*	*x*	HS-mode controller code
0b1111	0b1*xx*	*x*	Reserved for future purposes
0b1111	0b0*xx*	*x*	10-bit target addressing

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

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

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

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

13.4.6 Controller Mode Operation

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

A controller can communicate with multiple target devices without relinquishing the bus. Instead of generating a STOP condition after communicating with the first target, the controller generates a Repeated START condition, or RESTART, by setting *I2Cn_MSTCTRL.restart* = 1. If a transaction is in progress, the peripheral finishes the transaction before generating a RESTART. The peripheral then transmits the target address stored in the transmit FIFO. The *I2Cn_MSTCTRL.restart* bit is automatically cleared to 0 as soon as the controller begins a RESTART condition.

I2Cn_MSTCTRL.start is automatically cleared to 0 after the controller has completed a transaction and sent a STOP condition.

The controller can also generate a STOP condition by setting *I2Cn_MSTCTRL.stop* = 1.

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

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

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

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

For controller mode operation, only configure the following registers when either:

1. The I^2C peripheral is disabled,

   or
2. The I^2C bus is guaranteed to be idle/free.

If this peripheral is the only controller on the bus, then changing the registers outside of a transaction (*I2Cn_MSTCTRL.start* = 0) satisfies this requirement:

I2Cn_CTRL.mst_mode

I2Cn_CTRL.irxm_en

I2Cn_CTRL.hs_en

I2Cn_RXCTRL1.cnt

I2Cn_MSTCTRL.ex_addr_en

I2Cn_CLKLO.lo

I2Cn_CLKHI.hi

I2Cn_HSCLK.lo

I2Cn_HSCLK.hi

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

All interrupt flags and interrupt enable bits

I2Cn_TXCTRL0.thd_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

13.4.6.1 I²C Controller Mode Receiver Operation

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

1. Write the number of data bytes to receive to the I²C receive count field (*I2Cn_RXCTRL1.cnt*).
2. Write the I²C target address byte to the *I2Cn_FIFO* register with the R/W bit set to 1.
3. Send a START condition by setting *I2Cn_MSTCTRL.start* = 1.
4. The controller transmits the target address from the *I2Cn_FIFO* register.
5. The I²C controller receives an ACK from the target, and the controller sets the address ACK interrupt flag (*I2Cn_INTFL0.addr_ack* = 1).
6. The I²C controller receives data from the target and automatically ACKs each byte. The software must retrieve this data by reading the *I2Cn_FIFO* register.
7. Once *I2Cn_RXCTRL1.cnt* data bytes are received, the I²C controller sends a NACK to the target and sets the Transfer Done Interrupt Status Flag (*I2Cn_INTFL0.done* = 1).
8. If *I2Cn_MSTCTRL.restart* or *I2Cn_MSTCTRL.stop* is set, then the I²C controller sends a repeated START or STOP, respectively.

13.4.6.2 I²C Controller Mode Transmitter Operation

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

1. Write the I²C target address byte to the *I2Cn_FIFO* register with the R/W bit set to 0.
2. Write the desired data bytes to the *I2Cn_FIFO* register, up to the size of the transmit FIFO. (e.g., If the transmit FIFO size is 8 bytes, the software can write one address byte and seven data bytes before starting the transaction.)
3. Send a START condition by setting *I2Cn_MSTCTRL.start* = 1.
4. The controller transmits the target address byte written to the *I2Cn_FIFO* register.
5. The I²C controller receives an ACK from the target, and the controller sets the address ACK interrupt flag (*I2Cn_INTFL0.addr_ack* = 1).
6. The *I2Cn_FIFO* register data bytes are transmitted on the SDA line.
    a. The I²C controller receives an ACK from the target after each data byte.
    b. As the transfer proceeds, the software should refill the transmit FIFO by writing to the *I2Cn_FIFO* register as needed.
    c. If the transmit FIFO goes empty during this process, the controller pauses at the beginning of the byte and waits for the software to either write more data or instruct the controller to send a RESTART or STOP condition.
7. Once the software writes all the desired bytes to the *I2Cn_FIFO* register; the software should set either *I2Cn_MSTCTRL.restart* or *I2Cn_MSTCTRL.stop*.
8. Once the controller sends all the remaining bytes and empties the transmit FIFO, it sets *I2Cn_INTFL0.done* and proceeds to send out either a RESTART condition if *I2Cn_MSTCTRL.restart* is set or a STOP condition if *I2Cn_MSTCTRL.stop* is set.

13.4.6.3 I²C Multicontroller Operation

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

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

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

If the I²C controller peripheral detects it has lost the arbitration, it stops generating SCL; sets *I2Cn_INTFL0.arb_err*; sets *I2Cn_INTFL0.tx_lockout*, flushing any remaining data in the transmit FIFO; and clears *I2Cn_MSTCTRL.start*, *I2Cn_MSTCTRL.restart*, and *I2Cn_MSTCTRL.stop* to 0. As long as the peripheral is not addressed by the winning controller, the I²C peripheral stays in controller mode (*I2Cn_CTRL.mst_mode = 1)*. If, *at any time,* another controller addresses this peripheral using an address programmed in the target address registers (*I2Cn_SLAVE3*:*I2Cn_SLAVE0*), then the I²C peripheral clears *I2Cn_CTRL.mst_mode* to 0 and begins responding as a target. This can even occur during the same address transmission during which the peripheral lost arbitration.

Note: Arbitration loss is considered an error condition, and like the other error conditions, sets I2Cn_INTFL0.tx_lockout. Therefore, after an arbitration loss, the software needs to clear I2Cn_INTFL0.tx_lockout and reload the transmit FIFO.

Also, in a multicontroller environment, the software does not need to wait for the bus to become free before attempting to start a transaction (writing 1 to *I2Cn_MSTCTRL*.*start*). If the bus is free when *I2Cn_MSTCTRL*.*start* is set to 1, the transaction begins immediately. If, instead, the bus is busy, then the peripheral:

1. Waits for the other controller to complete the transaction(s) by sending a STOP,
2. Counts out the bus free time using $t_{BUF} = t_{SCL\_LO}$ (see *Equation 13-3*), *and then*
3. Sends a START condition and begins transmitting the target address byte(s) in the transmit FIFO, followed by the rest of the transfer.

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

13.4.7 Target Mode Operation

When in target mode, the I²C peripheral operates as a target device on the I²C bus and responds to an external controller's requests to transmit or receive data. To configure the I²C peripheral as a target, write the *I2Cn_CTRL*.*mst_mode* bit to zero. The controller drives the I²C clock on the bus, so the SCL device pin is driven by the external controller, and *I2Cn_STATUS*.*mst_busy* remains a zero. Set the desired target addresses by writing to the target address registers (*I2Cn_SLAVE3*:*I2Cn_SLAVE0*).

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

- *I2Cn_CTRL.mst_mode* = 0 for target operation.
- I²C target address:
  - Set up to four target addresses by programming the *I2Cn_SLAVE3.addr*, *I2Cn_SLAVE2.addr*, *I2Cn_SLAVE1.addr*, and/or the *I2Cn_SLAVE0.addr* fields to the desired address for the device on the bus.
  - For extended addresses, set the corresponding *I2Cn_SLAVE3.ext_addr_en*:*I2Cn_SLAVE0.ext_addr_en* to 1 for 10-bit addressing or 0 for 7-bit addressing.
  - Enable each I²C target address register by clearing the corresponding disable field to 0 (*I2Cn_SLAVE3.dis*:*I2Cn_SLAVE0.dis*).
- *I2Cn_CTRL.gc_addr_en*
- *I2Cn_CTRL.irxm_en*
  - The recommended value for this field is 0. *Note that a setting of 1 is incompatible with target mode operation with clock stretching disabled (I2Cn_CTRL.clkstr_dis = 1).*
- *I2Cn_CTRL.clkstr_dis*
- *I2Cn_CTRL.hs_en*
- *I2Cn_RXCTRL0.dnr*
  - SMBus/PMBus applications should set this to 0, while other applications should set this to 1.
- *I2Cn_TXCTRL0.nack_flush_dis*
- *I2Cn_TXCTRL0.rd_addr_flush_dis*
- *I2Cn_TXCTRL0.wr_addr_flush_dis*
- *I2Cn_TXCTRL0.gc_addr_flush_dis*
- *I2Cn_TXCTRL0.preload_mode*
  - The recommended value is 0 for applications that can tolerate target clock stretching (*I2Cn_CTRL.clkstr_dis* = 0).
  - The recommended value is 1 for applications that do not allow target clock stretching (*I2Cn_CTRL.clkstr_dis* = 1).
- *I2Cn_CLKHI.hi*
  - Applies to target mode when clock stretching is enabled (*I2Cn_CTRL.clkstr_dis* = 0)
    - This is used to satisfy $t_{SU;DAT}$ after clock stretching; program it so that the value defined by *Equation 13-2* is >= $t_{SU;DAT(min)}$.
- *I2Cn_HSCLK.hi*
  - Applies to target mode in Hs Mode when clock stretching is enabled (*I2Cn_CTRL.clkstr_dis* = 0)
    - This is used to satisfy $t_{SU;DAT}$ after clock stretching during Hs-Mode operation; program it so that the value defined by *Equation 13-6* is >= $t_{SU;DAT(min)}$.

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

- All interrupt flags and interrupt enables.
- *I2Cn_TXCTRL0.thd_val* and *I2Cn_RXCTRL0.thd_lvl*
  - Transmit and receive FIFO threshold levels.
- *I2Cn_TXCTRL0.tx_ready_mode*
  - Transmit ready (can only be cleared by hardware).
- *I2Cn_TIMEOUT*.scl_to_val
  - Timeout control.
- *I2Cn_DMA.rx_en* and *I2Cn_DMA.tx_en*
  - Transmit and receive DMA enables.
- *I2Cn_FIFO.data*
  - FIFO access register.

13.4.7.1 Target Transmitter

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

13.4.7.1.1 Just-in-Time Target Transmitter

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

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

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers, including:
   a. Set the *I2Cn_SLAVE3*.*addr*:*I2Cn_SLAVE0*.*addr* fields with the desired I^2C target addresses.
   b. Set the corresponding *I2Cn_SLAVE3*.*ext_addr_en*:*I2Cn_SLAVE0*.*ext_addr_en* fields for either 7-bit or 10-bit addressing.
   c. Enable target address registers by clearing the *I2Cn_SLAVE3*.*dis*:*I2Cn_SLAVE0*.*dis* fields to 0.
   d. Just-in-time mode specific settings:
      i) *I2Cn_CTRL*. *clkstr_dis* = 0
      ii) *I2Cn_TXCTRL0*[5:2] = 0x8
      iii) *I2Cn_TXCTRL0*.*preload_mode* = 0.
   e. Program *I2Cn_CLKHI*.*hi* and *I2Cn_HSCLK*.*hi* with appropriate values satisfying $t_{SU;DAT}$ (and HS $t_{SU;DAT}$).
2. The software sets *I2Cn_CTRL*.*en* = 1.
   a. The controller is now listening for its address. For either a transmit (R/W = 1) or receive (R/W = 0) operation, the peripheral responds to its address with an ACK.
   b. When the address match occurs, the hardware sets *I2Cn_INTFL0*.*addr_match* and *I2Cn_INTFL0*.*tx_lockout*.
3. The software waits for *I2Cn_INTFL0*.*addr_match* to read 1, either through polling the interrupt flag or setting *I2Cn_INTEN0*.*addr_match* to generate an interrupt.
4. After *I2Cn_INTFL0*.*addr_match* =1, the software determines if a read or write address match occurred (*I2Cn_INTFL0*.*wr_addr_match* = 1 for write and *I2Cn_INTFL0*.*rd_addr_match* = 1 for a read. In this case, assume read = 1, indicating transmit.
   a. Software can determine which target address matched by reading the *I2Cn_INTFL0*.*mami* field. Each bit in the field corresponds to each target address.
   b. The hardware holds SCL low until the software clears *I2Cn_INTFL0*.*tx_lockout* and loads data into the FIFO.
5. The software clears *I2Cn_INTFL0*.*addr_match* and *I2Cn_INTFL0*.*tx_lockout*, along with other address match flags in the *I2Cn_INTFL0* register. Now that *I2Cn_INTFL0*.*tx_lockout* is 0, the software can begin loading the transmit data into the *I2Cn_FIFO*.
6. As soon as there is data in the FIFO, the hardware releases SCL (after counting out *I2Cn_CLKHI*.*hi*) and sends out the data on the bus.
7. While the controller keeps requesting data and sending ACKs, *I2Cn_INTFL0*.*done* remains 0, and the software should continue to monitor the transmit FIFO and refill it as needed.
   a. The FIFO level can be monitored synchronously through the transmit FIFO status/interrupt flags or asynchronously by setting *I2Cn_TXCTRL0*.*thd_val* and setting the *I2Cn_INTEN0*.*tx_thd* interrupt.
   b. If the transmit FIFO ever empties during the transaction, the hardware starts clock stretching and waits for it to be refilled.
8. The controller ends the transaction by sending a NACK. Once this happens, the *I2Cn_INTFL0*.*done* interrupt flag is set, and the software can stop monitoring the transmit FIFO.
   a. If the software needs to know how many data bytes are transmitted to the controller, it should check the transmit FIFO level as soon as *I2Cn_INTFL0*.*done* = 1 and use it to determine how many data bytes were successfully sent.

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

13.4.7.1.2 Preload Mode Target Transmit

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

To use target transmit preload mode:

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers, including:
   a. Set the *I2Cn_SLAVE3*.*addr*:*I2Cn_SLAVE0*.*addr* fields with the desired I^2C target addresses.
   b. Set the corresponding *I2Cn_SLAVE3*.*ext_addr_en*:*I2Cn_SLAVE0*.*ext_addr_en* fields for either 7-bit or 10-bit addressing.
   c. Enable target address registers by clearing the *I2Cn_SLAVE3*.*dis*:*I2Cn_SLAVE0*.*dis* fields to 0.
   d. Preload mode specific settings:
      i) *I2Cn_CTRL*.*clkstr_dis* = 1
      ii) *I2Cn_TXCTRL0*[5:2] = 0xF
      iii) *I2Cn_TXCTRL0*.*preload_mode* = 1.
2. The software sets *I2Cn_CTRL*.*en* = 1.
   a. Even though the controller is enabled, it does not ACK an address match with R/W equal to 1 until the software sets the *I2Cn_TXCTRL1*.*preload_rdy* field to 1.
3. The software prepares for the transmit operation by loading data into the transmit FIFO, enabling DMA, setting *I2Cn_TXCTRL0*.*thd_val,* and setting *I2Cn_INTEN0*.*tx_thd* interrupt, etc.
   a. If clock stretching is disabled, an empty transmit FIFO during the transmit operation causes a transmit underrun error. Therefore, the software should take any necessary steps to avoid an underrun *before* setting *I2Cn_TXCTRL1*.*preload_rdy* = 1.
   b. If clock stretching is enabled, then an empty transmit FIFO does not cause a transmit underrun error. However, it is recommended to follow the same preparation steps to minimize the amount of time spent clock stretching, which lets the transaction complete as quickly as possible.
4. Once the software has prepared for the transmit operation; it sets *I2Cn_TXCTRL1*.*preload_rdy* = 1.
   a. The controller is now fully enabled and responds with an ACK to an address match.
   b. The hardware sets *I2Cn_INTFL0*.*addr_match* when an address match occurs. *I2Cn_INTFL0*.*tx_lockout* is NOT set to 1 and remains 0.
5. The software waits for *I2Cn_INTFL0*.*addr_match* = 1, either through polling the interrupt flag or by setting *I2Cn_INTEN0*.*addr_match* to generate an interrupt when the event occurs.
6. After seeing *I2Cn_INTFL0*.*addr_match* =1, the software reads *I2Cn_INTFL0*.*rd_addr_match* to determine if the transaction is a read and reads *I2Cn_INTFL0*.*wr_addr_match* to determine if the transaction is a write. In this case, assume *I2Cn_INTFL0*.*rd_addr_match* = 1, indicating a transmit.
   a. The hardware begins sending out the data preloaded into the transmit FIFO.
   b. Once the first data byte is sent, the hardware automatically clears *I2Cn_TXCTRL1*.*preload_rdy* to 0.
7. While the controller keeps requesting data and sending ACKs, *I2Cn_INTFL0*.*done* remains 0, and the software should continue to monitor the transmit FIFO and refill it as needed.
   a. The FIFO level can be monitored synchronously through the transmit FIFO status/interrupt flags or asynchronously by setting *I2Cn_TXCTRL0*.*thd_val* and setting *I2Cn_INTEN0*.*tx_thd* interrupt.
   b. If clock stretching is disabled and the transmit FIFO empties during the transaction, the hardware sets *I2Cn_INTFL1*.*tx_un* = 1 and sends 0xFF for all following data bytes requested by the controller.

8. The controller ends the transaction by sending a NACK, causing the hardware to set the *I2Cn_INTFL0*.*done* interrupt flag.
    a. If the transmit FIFO empties simultaneously that the controller indicates the transaction is complete by sending a NACK, this is not considered an underrun event *I2Cn_INTFL1*.*tx_un* flag remains 0.
    b. If the software needs to know how many data bytes are transmitted to the controller, check the transmit FIFO level when the *I2Cn_INTFL0*.*done* flag is set to 1.
9. The transaction is complete, the software should "clean up," which includes clearing *I2Cn_INTFL0*.*done*. Return to step 3 and prepare for the next transaction.
    a. Any data remaining in the transmit FIFO is not discarded; it is reused for the next transmit operation.
        i) If this is not desired, the software can flush the transmit FIFO. Flush the transmit and receive FIFOs by writing 0 to *I2Cn_CTRL*.*en* and the writing 1 to *I2Cn_CTRL*.*en*.

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

If the transmit FIFO is not ready (*I2Cn_TXCTRL1*.*preload_rdy* = 0) and the I^2C controller receives a data read request from the controller, the hardware automatically sends a NACK at the end of the first address byte. The setting of the do not respond field is ignored by the hardware in this case because the only opportunity to send a NACK for an I^2C read transaction is after the address byte.

13.4.7.2 Target Receivers

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

Program flow for a receive operation is as follows:

1. With *I2Cn_CTRL*.*en* = 0, initialize all relevant registers, including:
   a. Set the *I2Cn_SLAVE3*.*addr*:*I2Cn_SLAVE0*.*addr* fields with the desired I²C target addresses.
   b. Set the corresponding *I2Cn_SLAVE3*.*ext_addr_en*:*I2Cn_SLAVE0*.*ext_addr_en* fields for either 7-bit or 10-bit addressing.
   c. Enable each target address by clearing the *I2Cn_SLAVE3*.*dis*:*I2Cn_SLAVE0*.*dis* fields to 0.
2. Set *I2Cn_CTRL*.*en* = 1.
   a. If an address match with the R/W bit equal to zero occurs, and the receive FIFO is empty, the peripheral responds with an ACK, and the *I2Cn_INTFL0*.*addr_match* flag is set.
   b. If the receive FIFO is not empty, then depending on the value of *I2Cn_RXCTRL0*.*dnr*, the peripheral NACKs either the address byte (*I2Cn_RXCTRL0*.*dnr* = 1) or the first data byte (*I2Cn_RXCTRL0*.*dnr* = 0).
3. Wait for *I2Cn_INTFL0*.*addr_match* = 1, either by polling or by enabling the *wr_addr_match* interrupt. Once a successful address match occurs, the hardware sets *I2Cn_INTFL0*.*addr_match* = 1.
4. Read *I2Cn_CTRL*.*read* to determine if the transaction is a transmit (*I2Cn_CTRL*.*read* = 1) or a receive (*I2Cn_CTRL*.*read* = 0) operation. In this case, assume *I2Cn_CTRL*.*read* = 0, indicating receive. The device begins receiving data into the receive FIFO.
5. Clear *I2Cn_INTFL0*.*addr_match*, and while the controller keeps sending data, *I2Cn_INTFL0*.*done* remains 0, and the software should continue to monitor the receive FIFO and empty it as needed.
   a. The FIFO level can be monitored synchronously through the receive FIFO status/interrupt flags or asynchronously by setting *I2Cn_RXCTRL0*.*thd_lvl* and enabling the *I2Cn_INTFL0*.*rx_thd* interrupt.
   b. If the receive FIFO ever fills up during the transaction, then the hardware sets *I2Cn_INTFL1*.*rx_ov* and then either:
      i. If *I2Cn_CTRL*.*clkstr_dis* = 0, start clock stretching and wait until the software reads from the receive FIFO, *or*
      ii. If *I2Cn_CTRL*.*clkstr_dis* = 1, respond to the controller with a NACK, and the last byte is discarded.
6. The controller ends the transaction by sending a RESTART or STOP. Once this happens, the *I2Cn_INTFL0*.*done* interrupt flag is set, and the software can stop monitoring the receive FIFO.
7. Once a target starts receiving into its receive FIFO, detection of an out of sequence STOP, START, or RESTART condition releases the I²C bus to the Idle state, and the hardware sets the *I2Cn_INTFL0*.*start_err* field or *I2Cn_INTFL0*.*stop_err* field to 1 based on the specific condition.

If the software has not emptied the data in the receive FIFO from the previous transaction by the time a controller addresses it for another write (i.e., receive) transaction, then the controller does *not* participate in the transaction, and no additional data is written into the FIFO. Although a NACK *is* sent to the controller, the software can control if the NACK is sent with the initial address match or sent at the end of the first data byte. Setting *I2Cn_RXCTRL0*.*dnr* to 1 chooses the former while setting *I2Cn_RXCTRL0*.*dnr* to 0 chooses the latter.

13.4.8 Interrupt Sources

The I²C controller has a very flexible interrupt generator that generates an interrupt signal to the interrupt controller on any of several events. On recognizing the I²C interrupt, the software determines the cause of the interrupt by reading the I²C interrupt flags registers *I2Cn_INTFL0* and *I2Cn_INTFL1*. Interrupts can be generated for the following events:

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

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

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

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

13.4.9 Transmit FIFO and Receive FIFO

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

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

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

During a receive transaction (which during controller operation is a READ, and during target operation is a WRITE), received bytes are automatically written to the receive FIFO. The software should monitor the receive FIFO level and unload data from it as needed by reading *I2Cn_FIFO*. If the receive FIFO becomes full during a controller mode transaction, then the hardware sets the *I2Cn_INTFL1.rx_ov* the *I2Cn_INTFL1.rx_ov* bit, and one of two things occur depending on the value of *I2Cn_CTRL.clkstr_dis*:

- If clock stretching is enabled (*I2Cn_CTRL.clkstr_dis* = 0), then the hardware stretches the clock until the software makes space available in the receive FIFO by reading *I2Cn_FIFO*. Once space is available, the hardware moves the

data byte from the shift register into the receive FIFO, the SCL device pin is released, and the controller is free to continue the transaction.

- If clock stretching is disabled (*I2Cn_CTRL.clkstr_dis* = 1), the hardware responds to the controller with a NACK, and the data byte is lost. The controller can return the bus to idle with a STOP condition or start a new transaction with a RESTART condition.

During a transmit transaction (which during controller operation is a WRITE, and during target operation is a READ), either the software or the DMA can provide data to be transmitted by writing to the transmit FIFO. Once the peripheral finishes transmitting each byte, it removes it from the transmit FIFO and, if available, begins transmitting the next byte.

Interrupts can be generated for the following FIFO status:

- Transmit FIFO level less than or equal to the threshold.
- Receive FIFO level greater than or equal to the threshold.
- Transmit FIFO underflow.
- Receive FIFO overflow.
- Transmit FIFO locked for writing.

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

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

- General Call Address Match: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.gc_addr_flush_dis*.
- Target Address Match Write: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.wr_addr_flush_dis*.
- Target Address Match Read: Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.rd_addr_flush_dis*.
- During operation as a target transmitter, a NACK is received. Automatic flushing and lockout can be disabled by setting *I2Cn_TXCTRL0.nack_flush_dis*.
- Any of the following interrupts:
  - Arbitration error, timeout error, controller mode address NACK error, controller mode data NACK error, start error, and stop error. Automatic flushing cannot be disabled for these conditions.

When the above conditions occur, the transmit FIFO is flushed so that data intended for a previous transaction is not transmitted unintentionally for a new transaction. In addition to flushing the transmit FIFO, the transmit lockout flag is set (*I2Cn_INTFL0.tx_lockout* = 1) and writes to the transmit FIFO are ignored until the software acknowledges the external event by clearing *I2Cn_INTFL0.tx_lockout*.

13.4.10 Transmit FIFO Preloading

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

When transmit FIFO preloading is enabled, the software controls ACKs to the external controller using the transmit ready (*I2Cn_TXCTRL1.preload_rdy*) bit. When *I2Cn_TXCTRL1.preload_rdy* is set to 0, the hardware automatically NACKs all read transactions from the controller. Setting *I2Cn_TXCTRL1.preload_rdy* to 1 sends an ACK to the controller on the next read transaction and transmits the data in the transmit FIFO. Preload the transmit FIFO before setting the *I2Cn_TXCTRL1.preload_rdy* field to 1.

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

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

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

13.4.11 Interactive Receive Mode (IRXM)

In some situations, the I2Cn might want to inspect and respond to each byte of received data. In this case, IRXM can be used. IRXM is enabled by setting *I2Cn_CTRL.irxm_en* = 1. If IRXM is enabled, it must occur before any I²C transfer is initiated.

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

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

While the I2Cn peripheral is waiting for the software to clear the *I2Cn_INTFL0.irxm* flag, the software can disable IRXM and, if operating as a controller, load the remaining number of bytes to be received for the transaction. This allows the software to examine the initial bytes of a transaction, which might be a command, and then disable IRXM to receive the remaining bytes in normal operation.

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

Note: IRXM only applies to data bytes and does not apply to address bytes, general call address responses, or START byte responses.

Note: When enabling IRXM and operating as a target, clock stretching must remain enabled (I2Cn_CTRL.clkstr_dis = 0).

13.4.12 Clock Stretching

When the I2Cn peripheral requires some response or intervention from the software to continue with a transaction, it holds SCL low, preventing the transfer from continuing. This is called 'clock stretching' or 'stretching the clock.' While the I²C Bus Specification defines the term 'clock stretching' to only apply to a target device holding the SCL line low, this section describes situations where the I2Cn peripheral holds the SCL line low in either target *or* controller mode and refers to *both* as clock stretching.

When the I2Cn peripheral stretches the clock, it typically does so in response to either a full receive FIFO during a receive operation or an empty transmit FIFO during a transmit operation. Necessarily, this occurs before the next data byte begins, either between the ACK bit and the first data bit or, if at the beginning of a transaction, immediately after a START or RESTART condition. However, when operating in IRXM (*I2Cn_CTRL.irxm_en* = 1), the peripheral can also clock stretch *before* the ACK bit, allowing the software to decide if to send an ACK or NACK.

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

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

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

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

13.4.13 Bus Timeout

The timeout field, *I2Cn_TIMEOUT.scl_to_val,* is used to detect bus errors. *Equation 13-8* and *Equation 13-9* show equations for calculating the maximum and minimum timeout values based on the value loaded into the *I2Cn_TIMEOUT.scl_to_val* field.

Equation 13-8: I²C Timeout Maximum

$$t_{TIMEOUT} \leq \left(\frac{1}{f_{I2C\_CLK}}\right) \times \left((I2Cn\_TIMEOUT.scl\_to\_val \times 32) + 3\right)$$

Due to clock synchronization, the timeout is guaranteed to meet the following minimum time calculation shown in *Equation 13-9*.

Equation 13-9: I²C Timeout Minimum

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

The timeout feature is disabled when *I2Cn_TIMEOUT.scl_to_val* = 0 and is enabled for any non-zero value. When the timeout is enabled, the timeout timer starts counting when the I2Cn peripheral hardware drives SCL low and is reset by the I2Cn peripheral hardware when the SCL line is released.

The timeout counter only monitors if the I2Cn peripheral hardware is driving the SCL line low. It does not monitor if an external I2Cn device is actively holding the SCL line low. The timeout counter also does not monitor the status of the SDA line.

If the timeout timer expires, a bus error condition occurred. When a timeout error occurs, the I2Cn peripheral hardware releases the SCL and SDA lines, and sets the timeout error interrupt flag to 1 (*I2Cn_INTFL0.to_err* = 1).

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

13.4.14 DMA Control

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

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

Set the DMA burst size as shown in *Equation 13-10* to ensure the DMA does not overflow the transmit FIFO:

Equation 13-10: DMA Burst Size Calculation for I²C Transmit

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

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

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

When the receive FIFO count (*I2Cn_RXCTRL1.lvl*) is greater than or equal to the receive FIFO threshold level *I2Cn_RXCTRL0.thd_lvl*, the DMA transfers data out of the receive FIFO according to the DMA configuration. Set the DMA burst size as shown in *Equation 13-11* to ensure the DMA does not underflow the receive FIFO:

Equation 13-11: DMA Burst Size Calculation for I²C Receive

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

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

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

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

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

13.5 Registers

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

Table 13-5: Register Summary

Offset	Register	Description
[0x0000]	*I2Cn_CTRL*	I²C Control Register
[0x0004]	*I2Cn_STATUS*	I²C Status Register
[0x0008]	*I2Cn_INTFL0*	I²C Interrupt Flags 0 Register
[0x000C]	*I2Cn_INTEN0*	I²C Interrupt Enable 0 Register
[0x0010]	*I2Cn_INTFL1*	I²C Interrupt Flags 1 Register
[0x0014]	*I2Cn_INTEN1*	I²C Interrupt Enable 1 Register
[0x0018]	*I2Cn_FIFOLEN*	I²C FIFO Length Register
[0x001C]	*I2Cn_RXCTRL0*	I²C Receive Control 0 Register
[0x0020]	*I2Cn_RXCTRL1*	I²C Receive Control 1 Register
[0x0024]	*I2Cn_TXCTRL0*	I²C Transmit Control 0 Register
[0x0028]	*I2Cn_TXCTRL1*	I²C Transmit Control 1 Register
[0x002C]	*I2Cn_FIFO*	I²C Transmit and Receive FIFO Register
[0x0030]	*I2Cn_MSTCTRL*	I²C Controller Control Register
[0x0034]	*I2Cn_CLKLO*	I²C Clock Low Time Register
[0x0038]	*I2Cn_CLKHI*	I²C Clock High Time Register
(0x003C)	*I2Cn_HSCLK*	I²C Hs-Mode Clock Control Register
[0x0040]	*I2Cn_TIMEOUT*	I²C Timeout Register
[0x0048]	*I2Cn_DMA*	I²C DMA Enable Register
[0x004C]	*I2Cn_SLAVE0*	I²C Target Address 0 Register
[0x0050]	*I2Cn_SLAVE1*	I²C Target Address 1 Register
[0x0054]	*I2Cn_SLAVE2*	I²C Target Address 2 Register
[0x0058]	*I2Cn_SLAVE3*	I²C Target Address 3 Register

13.5.1 Register Details

Table 13-6: 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	-	RO	0	**Reserved**
12	clkstr_dis	R/W	0	**Target Mode Clock Stretching** 0: Enabled. 1: Disabled.
11	read	R	0	**Target Read/Write Bit Status** Returns the logic level of the R/W bit on a received address match (*I2Cn_INTFL0.addr_match* = 1) or general call match (*I2Cn_INTFL0.gc_addr_match* = 1). This bit is valid for three system clock cycles after the address match status flag is set.
10	bb_mode	R/W	0	**Software Output Control Enabled** Setting this field to 1 enables software bit-bang control of the I2Cn Bus. 0: The I²C controller manages the SDA and SCL pins in the hardware. 1: SDA and SCL are controlled by the software using the *I2Cn_CTRL.sda_out* and *I2Cn_CTRL.scl_out* fields.
9	sda	R	-	**SDA Status** 0: SDA pin is logic low. 1: SDA pin is logic high.
8	scl	R	-	**SCL Status** 0: SCL pin is logic low. 1: SCL pin is logic high.
7	sda_out	R/W	0	**SDA Pin Output Control** Set the state of the SDA hardware pin (actively pull low or float). 0: Pull SDA low. 1: Release SDA. *Note: Only valid when I2Cn_CTRL.bb_mode=1*
6	scl_out	R/W	0	**SCL Pin Output Control** Set the state of the SCL hardware pin (actively pull low or float). 0: Pull SCL low. 1: Release SCL. *Note: Only valid when I2Cn_CTRL.bb_mode =1*
5	-	RO	0	**Reserved**
4	irxm_ack	R/W	0	**IRXM Acknowledge** If IRXM is enabled (*I2Cn_CTRL.irxm_en* = 1), this field determines if the hardware sends an ACK or a NACK to an IRXM transaction. 0: Respond to IRXM with ACK. 1: Respond to IRXM with NACK.
3	irxm_en	R/W	0	**IRXM Enable** When receiving data, this field allows for an IRXM interrupt event after each received byte of data. The I2Cn peripheral hardware can be enabled to send either an ACK or NACK for IRXM. See the *Interactive Receive Mode* section for detailed information. 0: Disabled. 1: Enabled. *Note: Only set this field when the I²C bus is inactive.*
2	gc_addr_en	R/W	0	**General Call Address Enable** 0: Ignore general call address. 1: Acknowledge general call address.

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

Table 13-7: I²C Status Register

I²C Status			I2Cn_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	RO	0	**Reserved**
5	mst_busy	RO	0	**Controller Mode I²C Bus Transaction Active** The peripheral is operating in controller mode, and a valid transaction beginning with a START command is in progress on the I²C bus. This bit reads 1 until the controller ends the transaction with a STOP command. This bit continues to read 1 while a target performs clock stretching. 0: Device not actively driving SCL clock cycles. 1: Device operating as controller and actively driving SCL clock cycles.
4	tx_full	RO	0	**Transmit FIFO Full** 0: Not full. 1: Full.
3	tx_em	RO	1	**Transmit FIFO Empty** 0: Not empty. 1: Empty.
2	rx_full	RO	0	**Receive FIFO Full** 0: Not full. 1: Full.
1	rx_em	RO	1	**Receive FIFO Empty** 0: Not empty. 1: Empty.
0	busy	RO	0	**Controller or Target Mode I²C Busy Transaction Active** The peripheral is operating in controller or target mode, and a valid transaction beginning with a START command is in progress on the I²C bus. This bit reads 1 until the peripheral acting as a controller or an external controller ends the transaction with a STOP command. This bit continues to read 1 while a target performs clock stretching. 0: I²C bus is idle. 1: I²C bus transaction in progress.

Table 13-8: I²C Interrupt Flag 0 Register

I²C Interrupt Flag 0			I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23	wr_addr_match	R/W1C	0	**Target Write Address Match Interrupt Flag** If set, the device is accessed for a write (i.e., receive) transaction in target mode, and the address received matches the device target address. See *I2Cn_INTFL0.mami* to determine which target address match occurred. 0: No address match. 1: Address match.

I²C Interrupt Flag 0			I2Cn_INTFL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
22	rd_addr_match	R/W1C	0	**Target Read Address Match Interrupt Flag** If set, the device is accessed for a read (i.e., transmit) transaction in target mode, and the address received matches the device target address. See *I2Cn_INTFL0.mami* to determine which target address match occurred. 0: No address match. 1: Address match.
21:20	-	RO	0	**Reserved**
19:16	mami	R/W1C	0	**MAMI Interrupt Flag** Each bit in this field corresponds to an address match interrupt with the corresponding target address register. For example, *I2Cn_SLAVE2* corresponds to mami[2] (bit 18).
15	tx_lockout	R/W1C	0	**Transmit FIFO Locked Interrupt Flag** If set, the transmit FIFO is locked, and writes to the transmit FIFO are ignored. When set, the transmit FIFO is automatically flushed. Writes to the transmit FIFO are ignored until this flag is cleared. Write 1 to clear. 0: transmit FIFO not locked. 1: transmit FIFO is locked, and all writes to the transmit FIFO are ignored.
14	stop_err	R/W1C	0	**Out of Sequence STOP Interrupt Flag** This flag is set if a STOP condition occurs out of the expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence STOP condition occurred.
13	start_err	R/W1C	0	**Out of Sequence START Interrupt Flag** This flag is set if a START condition occurs out of the expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence START condition occurred.
12	dnr_err	R/W1C	0	**Target Mode Do Not Respond Interrupt Flag** This occurs if an address match is made, but the transmit FIFO or receive FIFO is not ready. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: I²C address match occurred, and either the transmit or receive FIFO is not configured.
11	data_err	R/W1C	0	**Controller Mode Data NACK from External Target Interrupt Flag** The hardware sets this flag if a NACK is received from a target. This flag is only valid if the I2Cn peripheral is configured for controller mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Data NACK received from a target.
10	addr_nack_err	R/W1C	0	**Controller Mode Address NACK from Target Error Flag** The hardware sets this flag if an Address NACK is received from a target bus. This flag is only valid if the I2Cn peripheral is configured for controller mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Address NACK received from a target.

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

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

Table 13-9: I²C Interrupt Enable 0 Register

I²C Interrupt Enable 0			I2Cn_INTEN0	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved**
23	wr_addr_match	R/W	0	**Target Write Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in target mode, and the address received matches the device target addressed for a write transaction. 0: Disabled. 1: Enabled.
22	rd_addr_match	R/W	0	**Target Read Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in target mode, and the address received matches the device target addressed for a read transaction. 0: Disabled. 1: Enabled.
21:20	-	RO	0	**Reserved**
19:16	mami	R/W	0	**MAMI Interrupt Enable** Each bit in this field enables an interrupt for the corresponding target address register. For example, *I2Cn_SLAVE0* address corresponds to *mami*[0] (bit 16). 0: Disabled. 1: Enabled.
15	tx_lockout	R/W	0	**Transmit FIFO Lock Out Interrupt Enable** 0: Disabled. 1: Enabled.
14	stop_err	R/W	0	**Out of Sequence STOP Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
13	start_err	R/W	0	**Out of Sequence START Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
12	dnr_err	R/W	0	**Target Mode Do Not Respond Interrupt Enable** Set this field to enable interrupts in target mode when the "Do Not Respond" condition occurs. 0: Interrupt disabled. 1: Interrupt enabled.
11	data_err	R/W	0	**Controller Mode Received Data NACK from Target Interrupt Enable** 0: Disabled. 1: Enabled.
10	addr_nack_err	R/W	0	**Controller Mode Received Address NACK from Target Interrupt Enable** 0: Disabled. 1: Enabled.

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

Table 13-10: I²C Interrupt Flag 1 Register

I²C Interrupt Status Flags 1			I2Cn_INTFL1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved**
2	start	R/W1C	0	**START Condition Status Flag** If set, a device START condition is detected. 0: START condition not detected. 1: START condition detected.
1	tx_un	R/W1C	0	**Target Mode Transmit FIFO Underflow Status Flag** In target mode operation, the hardware sets this flag automatically if the transmit FIFO is empty and the controller requests more data by sending an ACK after the previous byte is transferred. 0: Target mode transmit FIFO underflow condition has not occurred. 1: Target mode transmit FIFO underflow condition occurred.

I²C Interrupt Status Flags 1			I2Cn_INTFL1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
0	rx_ov	R/W1C	0	**Target Mode Receive FIFO Overflow Status Flag** In target mode operation, the hardware sets this flag automatically when a receive FIFO overflow occurs. Write 1 to clear. Writing 0 has no effect. 0: Target mode receive FIFO overflow event has not occurred. 1: Target mode receive FIFO overflow condition occurred (data lost).

Table 13-11: I²C Interrupt Enable 1 Register

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

Table 13-12: 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 13-13: I²C Receive Control 0 Register

I²C Receive Control 0			I2Cn_RXCTRL0	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	thd_lvl	R/W	0	**Receive FIFO Threshold Level** Set this field to the required number of bytes to trigger a receive FIFO threshold event. When the number of bytes in the receive FIFO is equal to or greater than this field, the hardware sets the *I2Cn_INTFL0.rx_thd* bit, indicating a receive FIFO threshold level event. 0: 0 bytes or more in the receive FIFO causes a threshold event. 1: 1+ bytes in the receive FIFO triggers a receive threshold event (recommended minimum value). … 8: Receive FIFO threshold event only occurs when the receive FIFO is full.

I²C Receive Control 0			I2Cn_RXCTRL0	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
7	flush	R/W1O	0	**Flush Receive FIFO** Write 1 to this field to initiate a receive FIFO flush, clearing all data in the receive FIFO. This field is automatically cleared by the hardware when the receive FIFO flush completes. Writing 0 has no effect. 0: Receive FIFO flush complete or not active. 1: Flush the receive FIFO
6:1	-	RO	0	**Reserved**
0	dnr	R/W	0	**Target Mode Do Not Respond** Target mode operation only. If the device is addressed for a write operation, and there is still data in the receive FIFO, then: 0: Always respond to an address match with an ACK but always respond to data bytes with a NACK. 1: NACK the address.

Table 13-14: I²C Receive Control 1 Register

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

Table 13-15: I²C Transmit Control 0 Register

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

I²C Transmit Control 0			I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
11:8	thd_val	R/W	0	**Transmit FIFO Threshold Level** This field sets the level for a transmit FIFO threshold event interrupt. If the number of bytes remaining in the transmit FIFO falls to this level or lower, the interrupt flag *I2Cn_INTFL0.tx_thd* is set, indicating a transmit FIFO threshold event occurred. 0: 0 bytes remaining in the transmit FIFO triggers a transmit FIFO threshold event. 1: 1 byte or fewer remaining in the transmit FIFO triggers a transmit FIFO threshold event (recommended minimum value). … 7: 7 or fewer bytes remaining in the transmit FIFO triggers a transmit FIFO threshold event
7	flush	R/W1O	0	**Transmit FIFO Flush** A transmit FIFO flush clears all remaining data from the transmit FIFO. 0: Transmit FIFO flush is complete or not active. 1: Flush the transmit FIFO. *Note: The hardware automatically clears this bit to 0 after it is written to 1 when the flush is completed.* If *I2Cn_INTFL0.tx_lockout* = 1, then *I2Cn_TXCTRL0.flush* = 1.
6	-	RO	0	**Reserved**
5	nack_flush_dis	R/W	0	**Transmit FIFO received NACK Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Received NACK at the end of a target transmit operation enabled. 1: Received NACK at the end of a target transmit operation disabled. *Note: Upon entering transmit preload mode, the hardware automatically sets this bit to 0. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 0).*
4	rd_addr_flush_dis	R/W	0	**Transmit FIFO Target Address Match Read Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled. 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bitfield to 1).*
3	wr_addr_flush_dis	R/W	0	**Transmit FIFO Target Address Match Write Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 1).*

I²C Transmit Control 0			I2Cn_TXCTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
2	gc_addr_flush_dis	R/W	0	**Transmit FIFO General Call Address Match Auto Flush Disable** Various situations or conditions are described in this user guide, leading to the transmit FIFO being flushed and locked out (*I2Cn_INTFL0.tx_lockout* = 1). 0: Enabled. 1: Disabled. *Note: Upon entering transmit preload mode, hardware automatically sets this bit to 1. The software can subsequently set this bit to any value desired (i.e., the hardware does not continuously force the bit to 1).*
1	tx_ready_mode	R/W	0	**Transmit FIFO Ready Manual Mode** 0: The hardware controls *I2Cn_TXCTRL1.preload_rdy.* 1: Software control of *I2Cn_TXCTRL1.preload_rdy.*
0	preload_mode	R/W	0	**Transmit FIFO Preload Mode Enable** 0: Normal operation. An address match in target mode, or a general call address match, flushes and locks the transmit FIFO so it cannot be written and sets *I2Cn_INTFL0.tx_lockout.* 1: Transmit FIFO preload mode. An address match in target mode, or a general call address match, does not lock the transmit FIFO and does not set *I2Cn_INTFL0.tx_lockout*. This allows the software to preload data into the transmit FIFO. The status of the I²C is controllable at *I2Cn_TXCTRL1.preload_rdy.*

Table 13-16: I²C Transmit Control 1 Register

I²C Transmit Control 1			I2Cn_TXCTRL1	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	lvl	R	0	**Transmit FIFO Byte Count Status** 0: 0 bytes (No data). 1: 1 byte. 2: 2 bytes. 3: 3 bytes. 4: 4 bytes. 5: 5 bytes. 6: 6 bytes. 7: 7 bytes. 8: 8 bytes (max value).
7:1	-	RO	0	**Reserved**
0	preload_rdy	R/W1O	1	**Transmit FIFO Preload Ready Status** When transmit FIFO preload mode is enabled, *I2Cn_TXCTRL0.preload_mode* = 1, this bit is automatically cleared to 0. While this bit is 0, if the I2Cn hardware receives a target address match, a NACK is sent. Once the I2Cn hardware is ready (the software has preloaded the transmit FIFO, configured the DMA, etc.), the software must set this bit to 1, so the I2Cn hardware sends an ACK on a target address match. When transmit FIFO preload mode is disabled, *I2Cn_TXCTRL0.preload_mode* = 1, this bit is forced to 1, and the I2Cn hardware behaves normally.

Table 13-17: I²C Data Register

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

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

Table 13-18: I²C Controller Control Register

I²C Controller Control			I2Cn_MSTCTRL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved**
10:8	-	RO	0	**Reserved**
7	ex_addr_en	R/W	0	**Target Extended Addressing Enable** 0: Send a 7-bit address to the target. 1: Send a 10-bit address to the target.
6:3	-	RO	0	**Reserved**
2	stop	R/W1O	0	**Send STOP Condition** 1: Send a STOP Condition at the end of the current transaction. *Note: This bit is automatically cleared by the hardware when the STOP condition begins.*
1	restart	R/W1O	0	**Send Repeated START Condition** After sending data to a target, the controller can send another START to retain control of the bus. 1: Send a repeated START condition to the target instead of sending a STOP condition at the end of the current transaction. *Note: This bit is automatically cleared by the hardware when the repeated START condition begins.*
0	start	R/W1O	0	**Start Controller Mode Transfer** 1: Start controller mode transfer. *Note: This bit is automatically cleared by the hardware when the transfer is completed or aborted.*

Table 13-19: I²C SCL Low Control Register

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

Table 13-20: I²C SCL High Control Register

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

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

Table 13-21: I²C Hs-Mode Clock Control Register

I²C Hs-Mode Clock Control			I2Cn_HSCLK	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved**
15:8	hi	R/W	0	**Hs-Mode Clock High Time** This field sets the Hs-Mode clock high count. In target mode, this is the time SCL is held high after data is output on SDA. *Note: See SCL Clock Generation for Hs-Mode for details on the requirements for the Hs-Mode clock high and low times.*
7:0	lo	R/W	0	**Hs-Mode Clock Low Time** This field sets the Hs-Mode clock low count. In target mode, this is the time SCL is held low after data is output on SDA. *Note: See SCL Clock Generation for Hs-Mode for details on the requirements for the Hs-Mode clock high and low times.*

Table 13-22: I²C Timeout Register

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

Table 13-23: I²C DMA Register

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

I²C DMA			I2Cn_DMA	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
1	rx_en	R/W	0	**Receive DMA Channel Enable** 0: Disabled. 1: Enabled.
0	tx_en	R/W	0	**Transmit DMA Channel Enable** 0: Disabled. 1: Enabled.

Table 13-24: I²C Target Address 0 Register

I²C Target Address			I2Cn_SLAVE0	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ext_addr_en	R/W	0	**Target Mode Extended Address Length Select** 0: 7-bit addressing. 1: 10-bit addressing.
14:11	-	RO	0	**Reserved**
10	dis	R/W	0	**Target Address 0 Disable** Setting this field disables this register's target address.
9:0	addr	R/W	0	**Target Mode Target Address** In target mode operation, (*I2Cn_CTRL.mst_mode* = 0), set this field to the target address for the I²C port. For 7-bit addressing, the address occupies the least significant 7 bits. For 10-bit addressing, the 9-bits of address occupies the most significant 9 bits, and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE0.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

Table 13-25: I²C Target Address 1 Register

I²C Target Address 1			I2Cn_SLAVE1	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ext_addr_en	R/W	0	**Target Mode Extended Address Length Select** 0: 7-bit addressing. 1: 10-bit addressing.
14:11	-	RO	0	**Reserved**
10	dis	R/W	1	**Target Address 1 Disable** Setting this field to 1 disables this register's target address.
9:0	addr	R/W	0	**Target Mode Target Address** In target mode operation, (*I2Cn_CTRL.mst_mode* = 0), set this field to the target address for the I²C port. For 7-bit addressing, the address occupies the least significant 7 bits. For 10-bit addressing, the 9-bits of address occupies the most significant 9 bits, and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE0.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

Table 13-26: I²C Target Address 2 Register

I²C Target Address 2			I2Cn_SLAVE2	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**

I²C Target Address 2			I2Cn_SLAVE2	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
15	ext_addr_en	R/W	0	**Target Mode Extended Address Length Select** 0: 7-bit addressing. 1: 10-bit addressing.
14:11	-	RO	0	**Reserved**
10	dis	R/W	1	**Target Address 2 Disable** Setting this field to 1 disables this register's target address.
9:0	addr	R/W	0	**Target Mode Target Address** In target mode operation, (*I2Cn_CTRL.mst_mode* = 0), set this field to the target address for the I²C port. For 7-bit addressing, the address occupies the least significant 7 bits. For 10-bit addressing, the 9-bits of address occupies the most significant 9 bits, and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE2.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

Table 13-27: I²C Target Address 3 Register

I²C Target Address 3			I2Cn_SLAVE3	[0x0058]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	ext_addr_en	R/W	0	**Target Mode Extended Address Length Select** 0: 7-bit addressing. 1: 10-bit addressing.
14:11	-	RO	0	**Reserved**
10	dis	R/W	1	**Target Address 3 Disable** Setting this field to 1 disables this registers target address.
9:0	addr	R/W	0	**Target Mode Target Address** In target mode operation, (*I2Cn_CTRL.mst_mode* = 0), set this field to the target address for the I²C port. For 7-bit addressing, the address occupies the least significant 7 bits. For 10-bit addressing, the 9-bits of address occupies the most significant 9 bits, and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE3.ext_addr_en controls if this field is a 7-bit or 10-bit address.*

14. Inter-Integrated Sound Interface (I²S)

I²S is a serial audio interface for communicating pulse-code modulation (PCM) encoded streams between devices. The peripheral supports both controller and target modes.

Key features:

- Stereo (2 channel) and mono (left or right channel option) formats.
- Separate DMA channels for transmit and receive.
- Flexible timing
  - Configurable sampling rate from $^{1}/_{65536}$ to 1 of the I²S input clock.
- Flexible data format
  - The number of bits per data word can be selected from 1 to 32, typically 8-, 16-, 24-, or 32-bit width.
  - Feature enhancement not in the I²S specification:
    - Word/Channel select polarity control.
    - First bit position selection.
    - Selectable FIFO data alignment to the MSB or the LSB of the sample.
    - Sample size less than the word size with adjustment to MSB or LSB of the word.
    - Optional sign extension.
- Full-duplex serial communication with separate I²S serial data input and serial data output pins.

14.1 Instances

Table 14-1: MAX32662 I²S Instances

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

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

14.1.1 I²S Bus Lines and Definitions

The I²S peripheral includes support for the following signals:

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

Detailed pin and alternate function mapping are shown in *Table 14-2*.

Table 14-2: MAX32662 I²S Pin Mapping

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

14.2 Details

The I²S supports full-duplex serial communication with separate SDI and SDO pins. *Figure 14-1* shows an interconnect between a peripheral configured in controller mode, communicating with an external I²S target and an external I²S controller. In controller mode, the peripheral hardware generates the BCLK and LRCLK, and each is output to each target device.

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

Figure 14-1: I²S Controller Mode

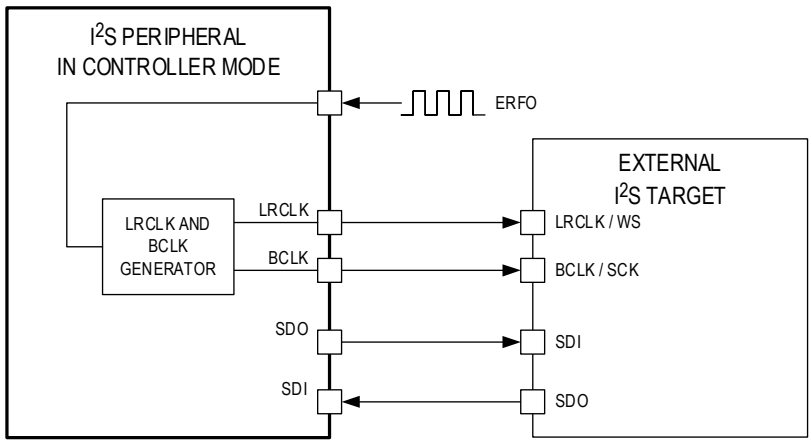
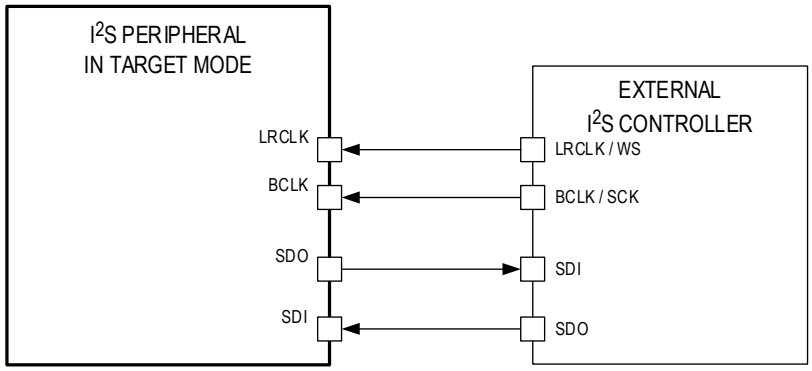


Figure 14-2 shows the I²S peripheral configured for target operation. The LRCLK and BCLK signals are generated externally by the controller, and are inputs to the I²S peripheral.

Figure 14-2: I²S Target Mode



14.3 Controller and Target Mode Configuration

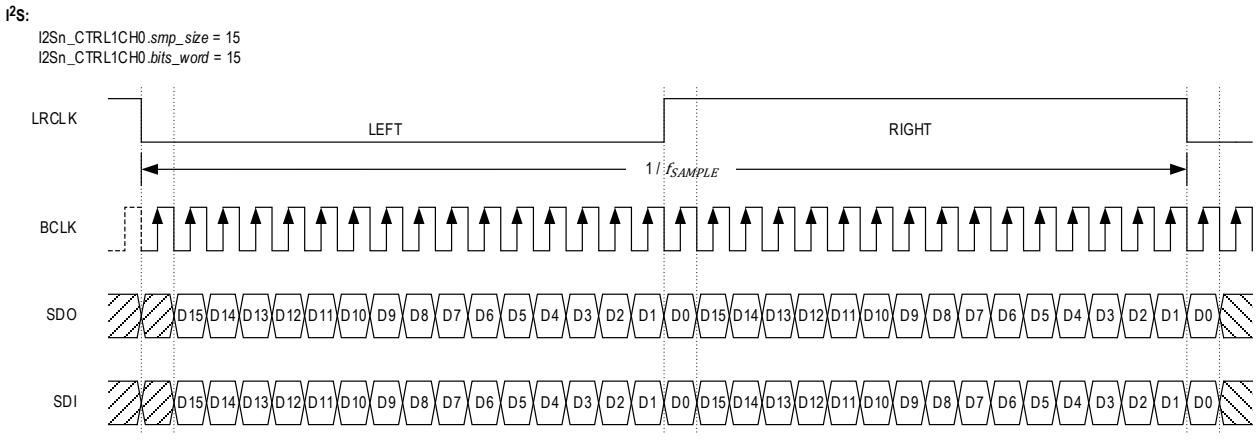
The device supports controller and target modes. In controller mode, the BCLK and LRCLK signals are generated internally, and output on the BCLK and LRCLK pins. In target mode, the BCLK and LRCLK pins are configured as inputs, and the external controller's clock source controls the peripheral timing.

Table 14-3: I²S Mode Configuration

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

14.4 Clocking

Figure 14-3: Audio Interface I²S Signal Diagram



I²S communication is synchronized using two signals, the LRCLK and BCLK. When the I²S peripheral is configured as a controller, the BCLK and LRCLK signals are generated internally by the peripheral using the ERFO. See *Table 14-2* for details of the I²S pin mapping and alternate function selection. If using the I²S peripheral in controller mode, the ERFO must be enabled (*GCR_CLKCTRL.erfo_en* = 1).

When the I²S peripheral is configured in target mode, the BCLK and LRCLK pins must be configured as inputs. An external controller generates the BCLK and LRCLK signals, which the peripheral uses to synchronize itself to the I²S bus. *Figure 14-3* 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 14-1*.

Equation 14-1: CD Audio Bit Frequency Calculation

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

14.4.1 BCLK Generation for Controller Mode

As indicated by *Equation 14-1*, the requirements for determining the BCLK frequency are:

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

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

Equation 14-2: Calculating the Bit Clock Frequency for Audio

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

In controller mode, the I²S external clock input is used to generate the BCLK frequency. The I²S external clock is divided by the *I2S_CTRL1CH0.clkdiv* field to achieve the target BCLK frequency, as shown in *Equation 14-3*.

Equation 14-3: Controller Mode BCLK Generation Using the I²S External Clock

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

Use *Equation 14-4* to determine the I²S clock divider for a target BCLK frequency.

Equation 14-4: Controller Mode Clock Divisor Calculation for a Target Bit Clock Frequency

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

14.4.2 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 14-3*. The LRCLK is a 50% duty cycle signal and is the same frequency as the audio sampling frequency, f_{SAMPLE}.

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

Equation 14-5: Bits Per Word Calculation

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

The LRCLK frequency, or word select frequency, is automatically generated by the I²S peripheral hardware when set to operate as a controller. The LRCLK frequency calculation is shown in *Equation 14-6*.

Equation 14-6: LRCLK Frequency Calculation

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

14.5 Data Formatting

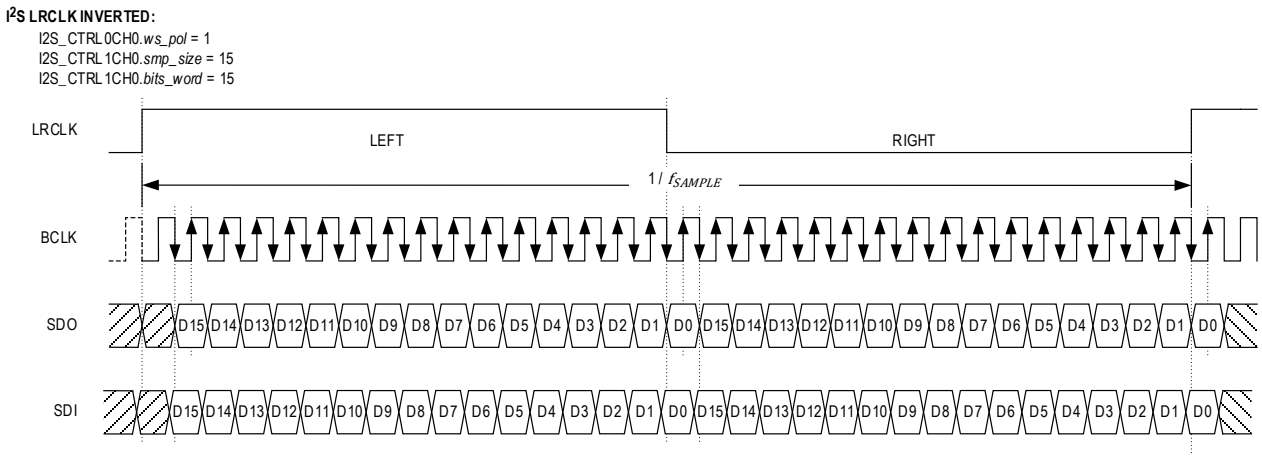
14.5.1 Sample Size

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 14-6* for an example of the bits per word field's setting compared to the sample size field's setting.

14.5.2 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 14-3*. 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 14-4*.

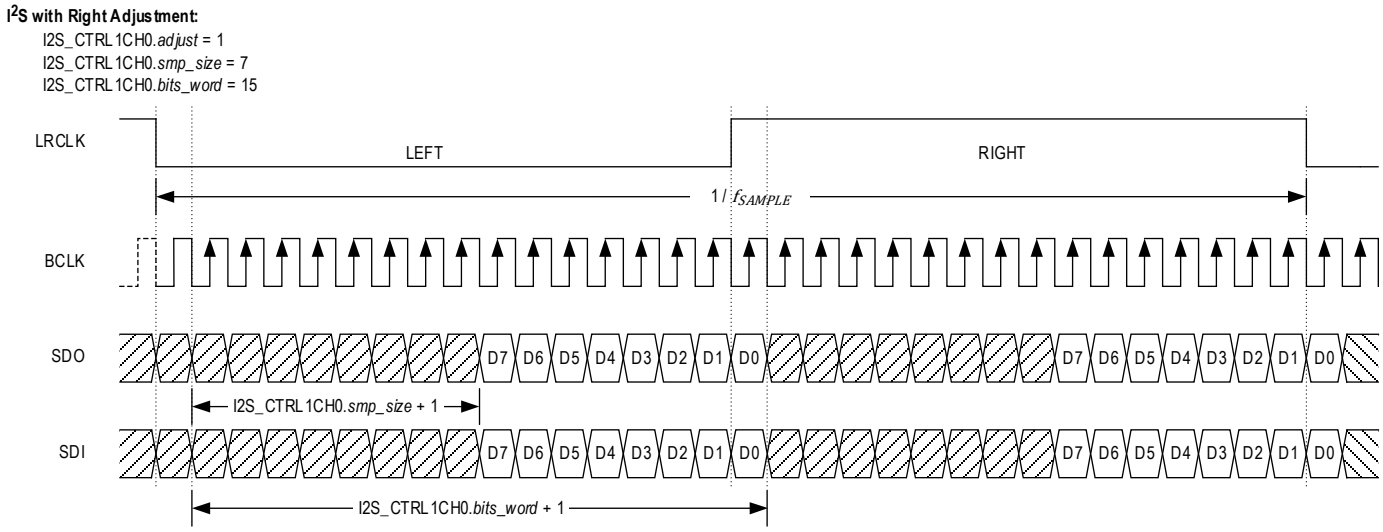
Figure 14-4: Audio Mode with Inverted Word Select Polarity



14.5.3 First Bit Location Control

The default setting is for the first bit of I²S data to be located at the second complete BCLK cycle after the LRCLK transition required by the I²S specification. See *Figure 14-3* 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 14-5*. Set *I2S_CTRL0CH0.msb_loc* to 1 to left justify the data with respect to the LRCLK.

Figure 14-5: Audio Controller Mode Left-Justified First Bit Location



14.5.4 Sample Adjustment

When the sample size field, *I2S_CTRL1CH0.smp_size*, is less than the bits per word field, *I2S_CTRL1CH0.bits_word*, use the *I2S_CTRL1CH0.adjust* field to set which bits are stored in the receive FIFO or transmitted from the transmit FIFO, either from the first sample of the SDI/SDO line or the last sample of the SDI/SDO line for the left and right channels. *Figure 14-6* shows an example of the default adjustment, MSB, where *I2S_CTRL1CH0.smp_size* = 7 and *I2S_CTRL1CH0.bits_word* = 15. *Figure 14-7* shows the adjustment set to the LSB of the SDI/SDO data.

Figure 14-6: MSB Adjustment when Sample Size is Less Than Bits Per Word

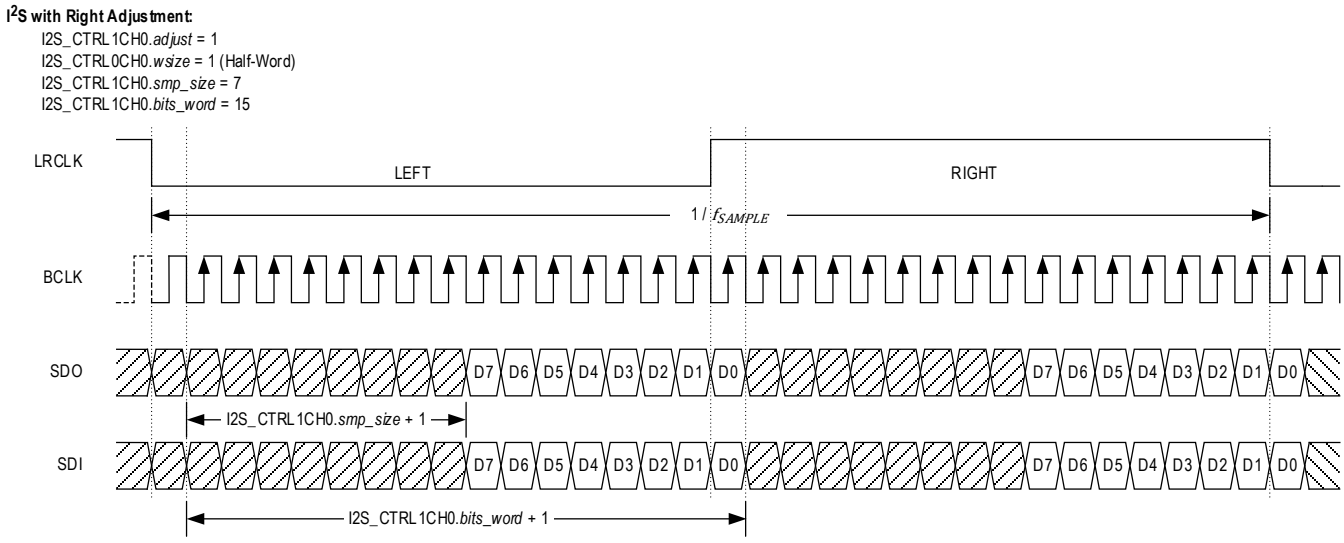
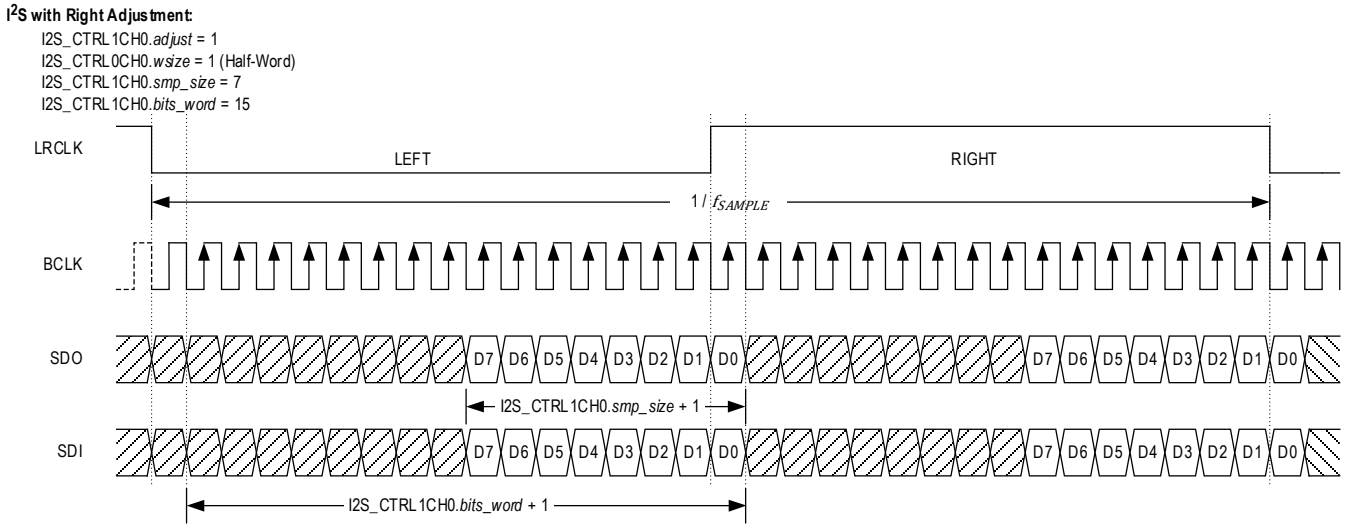


Figure 14-7: LSB Adjustment when Sample Size is Less Than Bits Per Word



14.5.5 Stereo/Mono Configuration

The I²S can transfer stereo or mono data based on the *I2S_CTRL0CH0.stereo* field. In stereo mode, both the left and right channels hold data. In mono mode, only the left or right channel contain data. For stereo mode, set *I2S_CTRL0CH0.stereo* to 0. Set the *I2S_CTRL0CH0.stereo* field to 2 for left channel mono. Set the *I2S_CTRL0CH0.stereo* field to 3 for right channel mono.

Figure 14-8: I²S Mono Left Mode

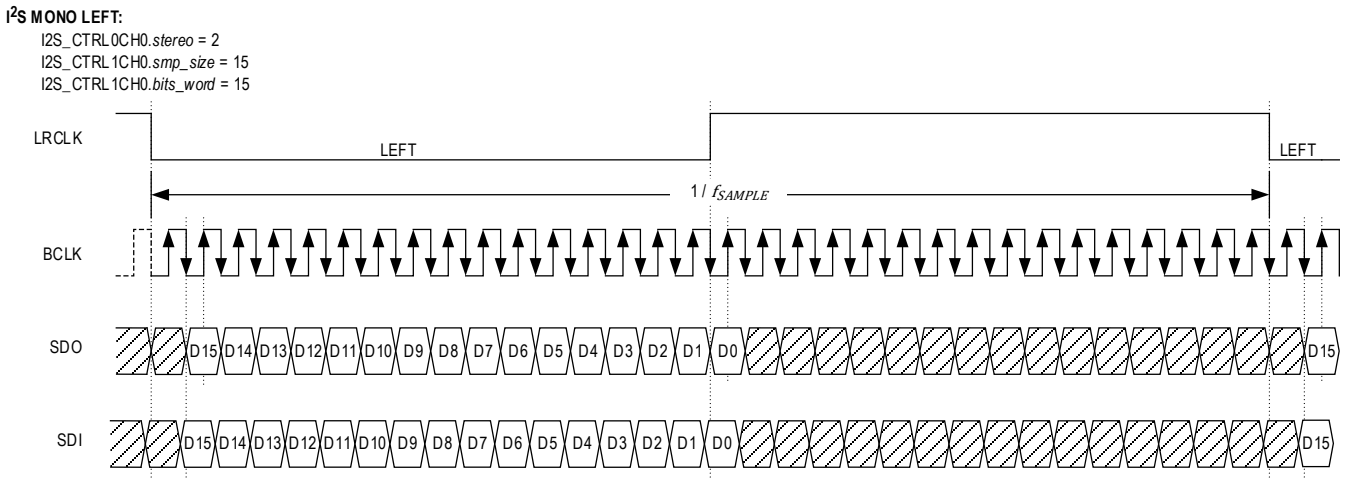
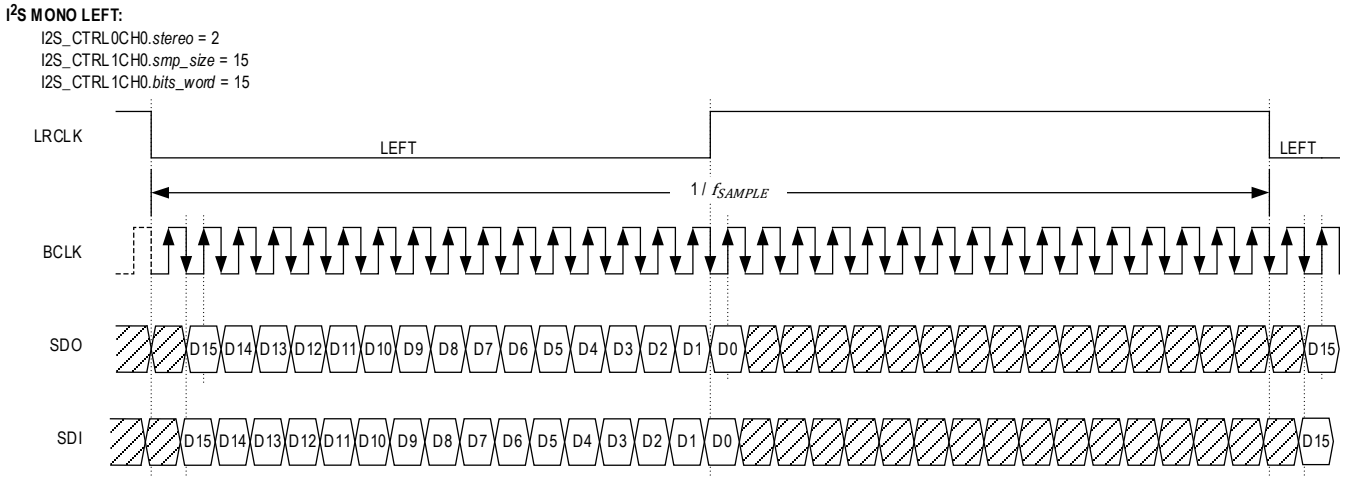


Figure 14-9: I²S Mono Right Mode



14.6 Transmit and Receive FIFOs

14.6.1 FIFO Data Width

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

14.6.2 Transmit FIFO

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

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

14.6.3 Receive FIFO

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

14.6.4 FIFO Word Control

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

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

Table 14-4: Data Ordering for Byte Data Size (Stereo Mode)

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

Table 14-5: Data Ordering for Half-Word Data Size (Stereo Mode)

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

Table 14-6: 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

14.6.5 FIFO Data Alignment

The I²S data can be left aligned or right aligned using the *I2S_CTRL0CH0.align* field. The following conditions apply to each setting:
Left aligned: *I2S_CTRL0CH0.align* = 0

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

Right aligned: *I2S_CTRL0CH0.align* = 1

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

14.6.6 Typical Audio Configurations

Table 14-7 shows the relationship between the bits per word field and the sample size field. *Equation 14-7* shows the required relationship between the sample size field and the bits per word field.

Equation 14-7: Sample Size Relationship Bits per Word

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

The *I2S_CTRL1CH0.bits_word* column in *Table 14-7* is set using the equation $\frac{\#\ BCLK}{Channel} - 1$. The *I2S_CTRL1CH0.smp_size* column is the number of samples per word captured from the I²S bus and is calculated by the equation $\frac{\#\ Samples}{Channel} - 1$. Channel refers to the left and right channels of audio.

Table 14-7: Configuration for Typical Audio Width and Samples per WS Clock Cycle

Audio Sample Width/ Samples per WS Cycle	$\frac{\#\ BCLK}{Channel}$	$\frac{\#\ Samples}{Channel}$	I2S_CTRL1CH0			Sign extension (align = 1)†
			bits_word	smp_size	wsize	
8 bits / 16	8	8	7	7	0	
16 bits / 32	16	16	15	15	1	
20 bits / 40	20	20	19	19	2	sign
24 bits / 48	24	24	23	23	2	sign
24 bits / 64	32	24	31	23	2	sign
32 bits / 64	32	32	31	31	2	

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

14.7 Interrupt Events

The I²S peripheral generates interrupts for the events shown in *Table 14-8*. An interrupt is generated if the corresponding interrupt enable field is set. The interrupt flags stay set until cleared by the software by writing 1 to the interrupt flag field.

Table 14-8: I²S Interrupt Events

Event	Interrupt Flag	Interrupt Enable
Receive FIFO overrun	*I2S_INTFL.rx_ov_ch0*	*I2S_INTEN.rx_ov_ch0*
Receive threshold	*I2S_INTFL.rx_thd_ch0*	*I2S_INTEN.rx_thd_ch0*
Transmit FIFO half-empty	*I2S_INTFL.tx_he_ch0*	*I2S_INTEN.tx_he_ch0*
Transmit FIFO one byte remaining	*I2S_INTFL.tx_ob_ch0*	*I2S_INTEN.tx_ob_ch0*

14.7.1 Receive FIFO Overrun

A receive FIFO overrun event occurs if the number of data words in the receive FIFO, *I2S_DMACH0.rx_lvl* is equal to the RX_FIFO_DEPTH, and another word is shifted into the FIFO. The hardware automatically sets the *I2S_INTFL.rx_ov_ch0* field to 1 when this event occurs.

14.7.2 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.

14.7.3 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 14-8*. When this event occurs, the *I2S_INTFL.tx_he_ch0* flag is set to 1 by hardware.

Note: The transmit FIFO half-empty interrupt flag is set by the hardware one BCLK cycle before the actual condition occurring. If the BCLK is much slower than the I²S peripheral clock, the software can receive the interrupt while the actual transmit FIFO level is still equal to ½ of the TX_FIFO_DEPTH. The software should always read the transmit FIFO level before filling it to determine the correct number of words to write to the transmit FIFO. Read the level of the transmit FIFO using the I2S_DMACH0.tx_lvl field.

Equation 14-8: Transmit FIFO Half-Empty Condition

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

14.7.4 Transmit FIFO One Entry Remaining

A transmit FIFO one entry remaining event occurs when the number of entries in the transmit FIFO is 1, *I2S_DMACH0.tx_lvl* = 1. When this event occurs, the *I2S_INTFL.tx_ob_ch0* flag is set to 1 by the hardware.

Note: The transmit FIFO one entry remaining interrupt flag is set by the hardware one BCLK cycle before the actual condition occurring. If the BCLK is much slower than the I²S peripheral clock, the software can receive the interrupt while the actual transmit FIFO level is still equal to 2. The software should always read the transmit FIFO level before filling it to determine the correct number of words to write to the transmit FIFO. Read the level of the transmit FIFO using the I2S_DMACH0.tx_lvl field.

14.8 Direct Memory Access

The I²S supports DMA for both transmit and receive; separate DMA channels can be assigned to the receive and transmit FIFOs. The following describes the behavior of the receive and transmit DMA requests.

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

14.9 Block Operation

After exiting a power-on reset, the IP is disabled by default. Software must configure and enable the peripheral to establish the I²S serial communication. A typical software sequence is as follows:

1. Set *GCR_PCLKDIS1.i2s* to 0 to enable the I²S peripheral clock source shown in *Table 14-1*.
2. Disable the I²S clock by setting *I2S_CTRL1CH0.en* to 0.
3. Set *I2S_CTRL0CH0.rst* to 1 to reset the I²S configuration.
4. Set *I2S_CTRL0CH0.flush* to 1 to flush the FIFO buffers.
5. Configure the *I2S_CTRL0CH0.ch_mode* to select the controller or target configuration.
    a. For controller mode, configure the baud rate by programming the *I2S_CTRL1CH0.clkdiv* field to achieve the required bit rate, set the *I2S_CTRL1CH0.smp_size* field to the desired sample size of the data, and the *I2S_CTRL1CH0.adjust* field if the Sample Size is smaller than the number of bits per word.
6. Configure the threshold of the receive FIFO by programming the *I2S_CTRL0CH0.rx_thd_val*. The transmit FIFO threshold is a fixed value, which is half of the transmit FIFO depth.
7. If desired, configure DMA operation. See section *Direct Memory Access* for details.
8. Enable interrupt functionality by configuring the *I2S_INTEN* register if desired.
9. Program the *clkdiv* bits in the *I2S_CTRL1CH0* register for the new bit clock frequency.
10. For controller operation, load data in the transmit FIFO for transmit.
11. Re-enable the bit clock by setting *I2S_CTRL1CH0.en* to 1.

14.10 Registers

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

Table 14-9: I²S Register Summary

Offset	Register Name	Description
[0x0000]	*I2S_CTRL0CH0*	I²S Global Mode Control 0 Register
[0x0010]	*I2S_CTRL1CH0*	I²S Controller Mode Configuration Register
[0x0030]	*I2S_DMACH0*	I²S DMA Control Channel Register
[0x0040]	*I2S_FIFOCH0*	I²S FIFO Register
[0x0050]	*I2S_INTFL*	I²S Interrupt Status Register
[0x0054]	*I2S_INTEN*	I²S Interrupt Enable Register

14.10.1 Register Details

Table 14-10: I²S Control 0 Register

I²S Control 0				**I2S_CTRL0CH0**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:24	rx_thd_val	R/W	0	**Receive FIFO Interrupt Threshold** This field specifies the level of the receive FIFO for the threshold interrupt generation. Values of 0 or greater than the RX_FIFO_DEPTH are ignored.	
23:21	-	RO	0	**Reserved**	
20	fifo_lsb	R/W	0	**FIFO Bit Field Control** Only used if the FIFO size is larger than the sample size and *I2S_CTRL0CH0.align* = 0. For transmit, the LSB part is sent from the FIFO. For receive, store the LSB part in the FIFO without sign extension. 0: Disabled. 1: Enabled.	
19	rst	R/W1O	0	**Reset** Write 1 to reset the I²S peripheral. The hardware automatically clears this field to 0 when the reset is complete. 0: Reset not in process. 1: Reset peripheral.	
18	flush	R/W1O	0	**FIFO Flush** Write 1 to start a flush of the receive FIFO and the transmit FIFO. The hardware automatically clears this field when the operation is complete. 0: Flush complete or not in process. 1: Flush receive and transmit FIFOs.	
17	rx_en	R/W	0	**Receive Enable** Enable receive mode for the I²S peripheral. 0: Disabled. 1: Enabled.	
16	tx_en	R/W	0	**Transmit Enable** Enable transmit mode for the I²S peripheral. 0: Disabled. 1: Enabled.	

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

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

Table 14-12: 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 14-13: I²S FIFO Register

I²S FIFO			I2S_FIFOCH0	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**I²S FIFO** Writing to this field loads the next character into the transmit FIFO and increments the *I2S_DMACH0.tx_lvl*. Writes are ignored if the transmit FIFO is full. Reads of this field return the next character available from the receive FIFO and decrement the *I2S_DMACH0.rx_lvl*. The value 0 is returned if *I2S_DMACH0.rx_lvl* = 0.

Table 14-14: 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 14-15: I²S Interrupt Enable Register

I²S Interrupt Enable			I2S_INTEN	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	DNM	0	**Reserved, Do Not Modify**

I²S Interrupt Enable			I2S_INTEN	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**
3	tx_he_ch0	R/W	0	**Transmit FIFO Half-Empty Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.
2	tx_ob_ch0	R/W	0	**Transmit FIFO One Entry Remaining Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.
1	rx_thd_ch0	R/W	0	**Receive FIFO Threshold Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.
0	rx_ov_ch0	R/W	0	**Receive FIFO Overrun Event Interrupt Enable** Set this field to 1 to enable interrupts for this event. 0: Disabled. 1: Enabled.

15. Real-Time Clock (RTC)

15.1 Overview

The RTC is a 32-bit binary timer that keeps the time of day up to 136 years. It provides time-of-day and sub-second alarm functionality in the form of system interrupts.

The RTC operates on an external 32.768kHz time base. It can be generated from the internal crystal oscillator driving an external 32.768kHz crystal between the 32KIN and 32KOUT pins or a 32.768kHz square wave-driven directly into the 32KIN pin. Refer to the device data sheet for the required electrical characteristics of the external crystal.

A user-configurable, digital frequency trim is provided for applications requiring higher accuracy.

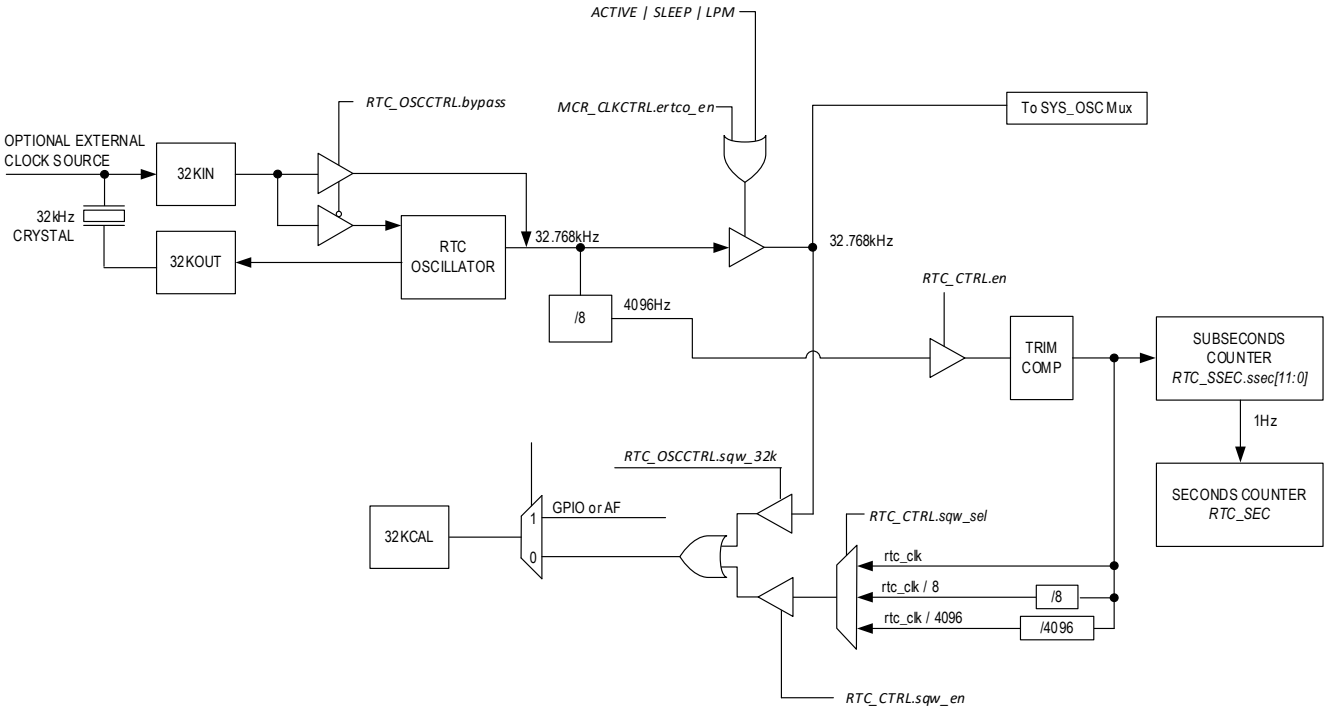
The 32-bit seconds counter register *RTC_SEC* is incremented every time there is a rollover of the *RTC_SSEC.ssec* sub-second counter field.

Two alarm functions are provided:

1. A programmable time-of-day alarm provides a single event alarm timer using the *RTC_TODA* alarm register, *RTC_SEC* register, and the *RTC_CTRL.tod_alarm_ie* field.
2. A programmable sub-second alarm provides a recurring alarm using the RTC sub-second alarm register, *RTC_SSECA*, and the *RTC_CTRL.ssec_alarm* field.

The RTC is powered by the AoD. Disabling the RTC, *RTC_CTRL.en* cleared to 0, stops incrementing the *RTC_SSEC* and *RTC_SEC*, but preserves their current values. The 32kHz oscillator is not affected by the *RTC_CTRL.en* field. While the RTC is enabled (*RTC_CTRL.en* = 1), the *RTC_TRIM.vrtc_tmr* field is also incremented every 32 seconds.

Figure 15-1: MAX32662 RTC Block Diagram



15.2 Instances

One instance of the RTC peripheral is provided. The RTC counter and alarm register details and description are shown in *Table 15-1*.

Table 15-1: RTC Seconds, Sub-Seconds, Time-of-Day Alarm, and Sub-Second Alarm Register Details

Field	Width (bits)	Counter Increment	Minimum	Maximum	Description
RTC_SEC.sec	32	1 second	1 second	136 years	Seconds counter field
RTC_SSEC.ssec	12	244 µs ($\frac{1}{4096Hz}$)	244 µs	1 second	Sub-second counter field
RTC_TODA.tod_alarm	20	1 second	1 second	12 days	Time-of-day alarm field
RTC_SSECA.ssec_alarm	32	244 µs ($\frac{1}{4096Hz}$)	244 µs	12 days	Sub-second alarm field

15.3 Register Access Control

Access protection mechanisms prevent the software from accessing critical registers and fields while the hardware is updating them. Monitoring the *RTC_CTRL.busy* and *RTC_CTRL.rdy* fields allows the software to determine when it is safe to write to registers and when registers return valid results.

Table 15-2: RTC Register Access

Register	Field	Read Access	Write Access	*RTC_CTRL.busy* = 1 during writes	Description
RTC_SEC	*.sec*	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	Y	Seconds counter
RTC_SSEC	*.ssec*	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	Y	Sub-second counter
RTC_TODA	*.tod_alarm*	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.tod_alarm_ie* = 0	Y	Time-of-day alarm
RTC_SSECA	*.ssec_alarm*	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.ssec_alarm_ie* = 0	Y	Sub-second alarm
RTC_TRIM	All	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.wr_en* = 1	Y	Trim
RTC_OSCCTRL	All	Always	*RTC_CTRL.wr_en* = 1	N	Oscillator control
RTC_CTRL	*en*	Always	*RTC_CTRL.busy* = 0 *RTC_CTRL.wr_en* = 1	Y	RTC enable field
	All other fields	Always	*RTC_CTRL.busy* = 0	Y	

† See the *RTC_SEC and RTC_SSEC Read Access Control* section for details.

15.3.1 RTC_SEC and RTC_SSEC Read Access Control

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

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

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

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

15.3.2 RTC Write Access Control

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

15.4 RTC Alarm Functions

The RTC provides time-of-day and sub-second interval alarm functions. The time-of-day alarm is implemented by matching the count values in the counter register with the alarm register's value. The sub-second interval alarm provides an auto-reload timer driven by the trimmed RTC clock source.

15.4.1 Time-of-Day Alarm

Program the RTC time-of-day alarm register (*RTC_TODA*) to configure the time-of-day alarm. The alarm triggers when the value stored in *RTC_TODA.tod_alarm* matches the *RTC_SEC*[19:0] seconds count register. This allows programming the time-of-day alarm to any future value between 1 second and 12 days relative to the current time with a resolution of 1 second. Disable the time-of-day alarm (*RTC_CTRL.tod_alarm_ie* = 0) before changing the *RTC_TODA.tod_alarm* field.

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

Setting the *RTC_CTRL.tod_alarm* bit to 1 in the software results in an interrupt request to the processor if the alarm time-of-day interrupt enable (*RTC_CTRL.tod_alarm_ie*) bit is set to 1, and the RTC's system interrupt enable is set.

15.4.2 Sub-Second Alarm

The *RTC_SSECA.ssec_alarm* and *RTC_CTRL.ssec_alarm_ie* fields control the sub-second alarm. Writing *RTC_SSECA.ssec_alarm* sets the starting value for the sub-second alarm counter. Writing the sub-second alarm enable (*RTC_CTRL.ssec_alarm_ie*) bit to 1 enables the sub-second alarm. Once enabled, an internal alarm counter begins incrementing from the *RTC_SSECA.ssec_alarm* field's value. When the counter rolls over from 0xFFFF FFFF to 0x0000 0000, the hardware sets the *RTC_CTRL.ssec_alarm* bit, triggering the alarm. At the same time, the hardware also reloads the counter with the value previously written to *RTC_SSECA.ssec_alarm*.

Disable the sub-second alarm, *RTC_CTRL.ssec_alarm_ie*, before changing the interval alarm value, *RTC_SSECA.ssec_alarm*.

The delay (uncertainty) associated with enabling the sub-second alarm is up to one sub-second clock period. This uncertainty is propagated to the first interval alarm. After that, if the interval alarm remains enabled, the alarm triggers after each sub-second interval as defined without the first alarm uncertainty because the sub-second alarm is an auto-reload timer. Enabling the sub-second alarm with the sub-second alarm register set to 0 (*RTC_SSECA* = 0) results in the maximum sub-second alarm interval.

15.4.3 RTC Interrupt and Wake-up Configuration

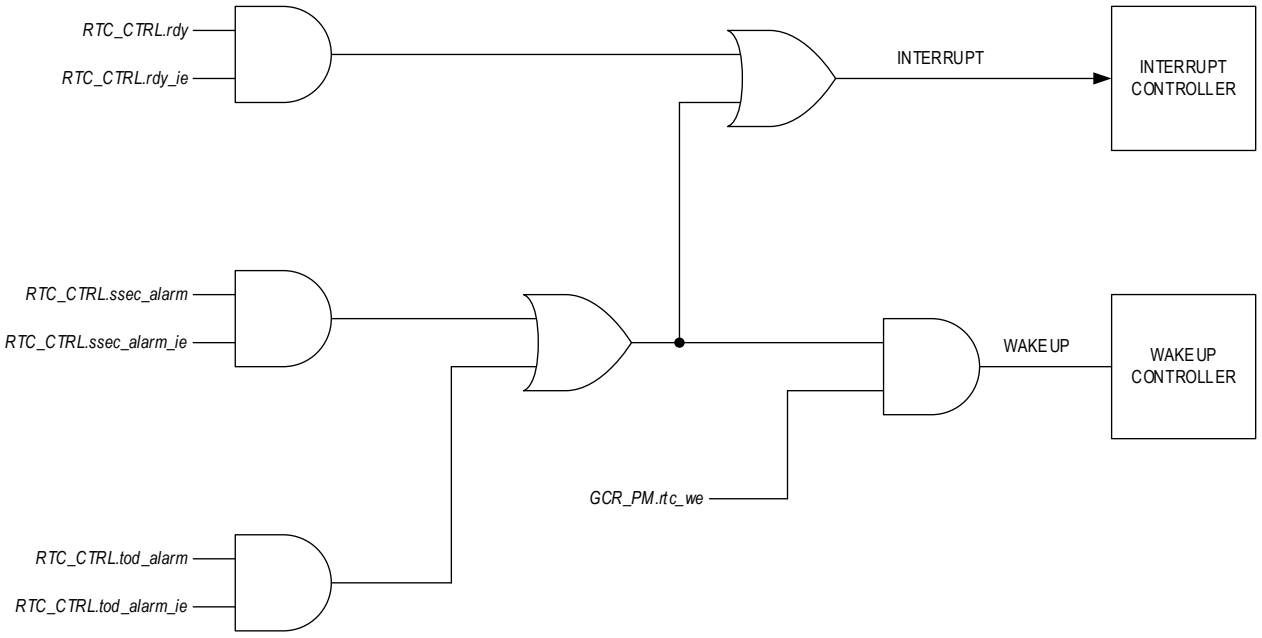
The following is a list of conditions that, when enabled, generate an RTC interrupt:

1. Time-of-day alarm
2. Sub-second alarm
3. *RTC_CTRL.rdy* field asserted high, signaling read access permitted

The RTC can be configured, so the time-of-day and sub-second alarms are a wake-up source for exiting the following low-power modes:

1. *BACKUP*
2. *DEEPSLEEP*
3. *SLEEP*

Figure 15-2: RTC Interrupt/Wake-up Diagram Wake-up Function



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

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

15.5 Square Wave Output

The RTC can output a 50% duty cycle square wave signal derived from the 32kHz oscillator on a selected device pin. See *Table 15-3* for the device pins, frequency options, and control fields specific to this device. Frequencies noted as compensated in *Table 15-3* are used during the RTC frequency calibration procedure because they incorporate the frequency adjustments provided by the digital trim function.

Table 15-3: MAX32662 RTC Square Wave Output Configuration

Function	Option	Control Field
Output Pin	32KCAL	0
Enable Frequency Output	1Hz (Compensated)	*RTC_CTRL.sqw_sel* = 0 *RTC_CTRL.sqw_en* = 1 *RTC_OSCCTRL.sqw_32k* = 0
	512Hz (Compensated)	*RTC_CTRL.sqw_sel* = 1 *RTC_CTRL.sqw_en* = 1 *RTC_OSCCTRL.sqw_32k* = 0
	4kHz	*RTC_CTRL.sqw_sel* = 2 *RTC_CTRL.sqw_en* = 1 *RTC_OSCCTRL.sqw_32k* = 0
	32kHz	*RTC_OSCCTRL.sqw_32k* = 1

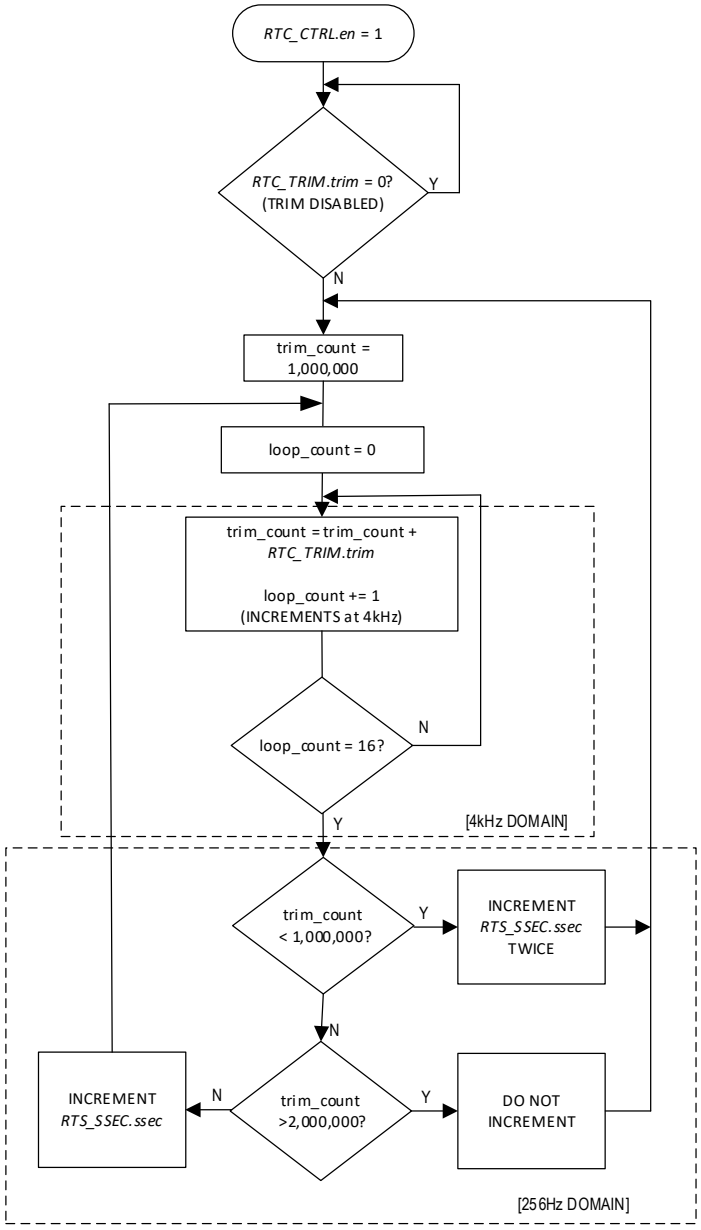
Use the following software procedure to generate and output the square wave:

1. Select the desired output frequency:
   a. Set the field *RTC_CTRL.sqw_sel* to 0 for a 1Hz compensated output frequency, or
   b. set the field *RTC_CTRL.sqw_sel* to 1 for a 512Hz compensated output frequency, or
   c. set the field *RTC_CTRL.sqw_sel* to 2 for a 4kHz output frequency, or
   d. set the field *RTC_OSCCTRL.sqw_32k* to 1 for the 32kHz frequency output.
2. Enable the system level output pin by setting the output pin shown in *Table 15-3*.
3. If the selected frequency is 1Hz, 512Hz, or 4kHz, set the *RTC_CTRL.sqw_en* field to 1 to output the selected output frequency.

15.6 RTC Calibration

A digital trim facility provides the ability to compensate for RTC inaccuracies of up to ± 127ppm when compared against an external reference clock. The trimming function utilizes an independent dedicated timer that increments the sub-second register based on a user-supplied, 2s-complement value in the *RTC_TRIM* register as shown in *Figure 15-3*.

Figure 15-3: Internal Implementation of 4kHz Digital Trim



Complete the following steps to perform an RTC calibration:

1. The software must configure and enable one of the compensated calibration frequencies as described in section *Square Wave Output*.
2. Measure the frequency on the square wave output pin and compute the deviation from an accurate reference clock.
3. Clear the *RTC_CTRL*.*rdy* field to 0.
4. Wait for the *RTC_CTRL*.*rdy* to be set to 1 by the hardware:
   a. Set the *RTC_CTRL*.*rdy_ie* to 1 to generate an interrupt when the *RTC_CTRL*.*rdy* field is set to 1, or
   b. Poll the *RTC_CTRL*.*rdy* field until it reads 1.
5. Poll the *RTC_CTRL*.*busy* field until it reads 0 to allow any active operations to complete.
6. Set the *RTC_CTRL*.*wr_en* field to 1 to allow access to the *RTC_TRIM*.*trim* field.
7. Write a trim value to the *RTC_TRIM*.*trim* field to correct for any measured inaccuracy.
8. Poll the *RTC_CTRL*.*busy* field until it reads 0
9. Clear the *RTC_CTRL*.*wr_en* field to 0.
10. Repeat the process as needed until the desired accuracy is achieved.

15.7 Registers

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

Table 15-4: RTC Register Summary

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

15.7.1 Register Details

Table 15-5: RTC Seconds Counter Register

RTC Seconds Counter			RTC_SEC	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	sec	R/W	0	**Seconds Counter** This register is a binary count of seconds.

Table 15-6: RTC Sub-Second Counter Register

RTC Sub-Seconds Counter			RTC_SSEC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:0	ssec	R/W	0	**Sub-Seconds Counter** *RTC_SEC* increments when this field rolls from 0xFFF to 0x000.

Table 15-7: RTC Time-of-Day Alarm Register

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

Table 15-8: RTC Sub-Second Alarm Register

RTC Sub-Second Alarm			RTC_SSECA	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	ssec_alarm	R/W	0	**Sub-second Alarm (4kHz)** Sets the starting and reload value of the internal sub-second alarm counter. The internal counter increments and generates an alarm when the internal counter rolls from 0xFFFF FFFF to 0x0000 0000.

Table 15-9: RTC Control Register

RTC Control			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	wr_en	R/W	0*	**Write Enable** This field controls access to the *RTC_TRIM* register and the RTC enable (*RTC_CTRL.en*) field. 1: Writes to the *RTC_TRIM* register and the *RTC_CTRL.en* field are allowed. 0: Writes to the *RTC_TRIM* register and the *RTC_CTRL.en* field are ignored. **Note: Reset on POR, system reset, and soft reset.*
14	rd_en	R/W	0	**Asynchronous Counter Read Enable** Set this field to 1 to allow direct read access of the *RTC_SEC* and *RTC_SSEC* registers without waiting for *RTC_CTRL.rdy.* Multiple consecutive reads of *RTC_SEC* and *RTC_SSEC* must be executed until two consecutive reads are identical to ensure data accuracy. 0: *RTC_SEC* and *RTC_SSEC* registers are synchronized and should only be accessed while *RTC_CTRL.rdy*= 1. 1: *RTC_SEC* and *RTC_SSEC* registers are asynchronous and require software interaction to ensure data accuracy.
13:11	-	RO	0	**Reserved**
10:9	sqw_sel	R/W	0*	**Frequency Output Select** This field selects the RTC-derived frequency to output on the square wave output pin. See *Table 15-3* for configuration details. 0: 1Hz (Compensated) 1: 512Hz (Compensated) 2: 4kHz 3: Reserved **Note: Reset on POR only.*
8	sqw_en	R/W	0*	**Square Wave Output Enable** This field enables the square wave output. See *Table 15-3* for configuration details. 0: Disabled. 1: Enabled. **Note: Reset on POR only.*
7	ssec_alarm	R/W	0*	**Sub-second Alarm Interrupt Flag** This interrupt flag is set when a sub-second alarm condition occurs. This flag is a wake-up source for the device. 0: No sub-second alarm pending. 1: Sub-second interrupt pending. **Note: Reset on POR only.*
6	tod_alarm	R/W	0*	**Time-of-Day Alarm Interrupt Flag** This interrupt flag is set by the hardware when a time-of-day alarm occurs. 0: No time-of-day alarm interrupt pending. 1: Time-of-day interrupt pending. **Note: Reset on POR only.*
5	rdy_ie	R/W	0*	**RTC Ready Interrupt Enable** 0: Disabled. 1: Enabled. **Note: Reset on POR, system reset, and soft reset.*

RTC Control			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
4	rdy	R/W0O	0*	**RTC Ready** This bit is set to 1 for 120µs by the hardware once a hardware update of the *RTC_SEC* and *RTC_SSEC* registers occurred. The software should read *RTC_SEC* and *RTC_SSEC* while this hardware bit is set to 1. The software can clear this bit at any time. An RTC interrupt is generated if *RTC_CTRL.rdy_ie* = 1. 0: Software reads of *RTC_SEC* and *RTC_SSEC* are invalid. 1: Software reads of *RTC_SEC* and *RTC_SSEC* are valid. **Note: Reset on POR, system reset, and soft reset.*
3	busy	RO	0*	**RTC Busy Flag** This field is set to 1 by the hardware while a register update is in progress. Software writes to the following registers result in this field being set to 1: • *RTC_SEC* • *RTC_SSEC* • *RTC_TRIM* The following fields cannot be written when this field is set to 1: • *RTC_CTRL.en* • *RTC_CTRL.tod_alarm_ie* • *RTC_CTRL.ssec_alarm_ie* • *RTC_CTRL.rdy_ie* • *RTC_CTRL.tod_alarm* • *RTC_CTRL.ssec_alarm* • *RTC_CTRL.sqw_en* • *RTC_CTRL.rd_en* This field is automatically cleared by the hardware when the update is complete. The software should poll this field until it reads 0 after changing the *RTC_SEC*, *RTC_SSEC*, or *RTC_TRIM* register before making any other RTC register modifications. 0: RTC not busy 1: RTC busy **Note: Reset on POR only.*
2	ssec_alarm_ie	R/W	0*	**Sub-Second Alarm Interrupt Enable** Check the *RTC_CTRL.busy* flag after writing to this field to determine when the RTC synchronization is complete. 0: Disable. 1: Enable. **Note: Reset on POR only.*
1	tod_alarm_ie	R/W	0*	**Time-of-Day Alarm Interrupt Enable** Check the *RTC_CTRL.busy* flag after writing to this field to determine when the RTC synchronization is complete. 0: Disable. 1: Enable. **Note: Reset on POR only.*

RTC Control			RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
0	en	R/W	0*	**Real-Time Clock Enable** The RTC write enable (*RTC_CTRL.wr_en*) bit must be set and RTC busy (*RTC_CTRL.busy*) must read 0 before writing to this field. After writing to this bit, check the *RTC_CTRL.busy* flag for 0 to determine when the RTC synchronization is complete. 0: Disabled. 1: Enabled. **Note: Reset on POR only.*

Table 15-10: RTC 32KHz Oscillator Digital Trim Register

RTC 32KHz Oscillator Digital Trim			RTC_TRIM	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	vrtc_tmr	R/W	0*	**V_{RTC} Time Counter** The hardware increments this field every 32 seconds while the RTC is enabled. **Note: Reset on POR only.*
7:0	trim	R/W	0*	**RTC Trim** This field specifies the 2s complement value of the trim resolution. Each increment or decrement of the field adds or subtracts 1ppm at each 4kHz clock value with a maximum correction of ± 127ppm. **Note: Reset on POR only.*

Table 15-11: RTC 32KHz Oscillator Control Register

RTC Oscillator Control			RTC_OSCCTRL	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:6	-	R/W	0	**Reserved**
5	sqw_32k	R/W	0	**RTC Square Wave Output** 0: Disabled. 1: Enables the 32kHz oscillator output or the external clock source is output on square wave output pin. See *Table 15-3* for configuration details. **Note: Reset on POR only.*
4	bypass	R/W	0	**RTC Crystal Bypass** This field disables the RTC oscillator and allows an external clock source to drive the 32KIN pin. 0: Disable bypass. RTC time base is an external 32kHz crystal. 1: Enable bypass. RTC time base is an external square wave driven on 32KIN. **Note: Reset on POR only.*
3:0	-	DNM	9	**Reserved Do Not Modify**

16. Timers (TMR/LPTMR)

Multiple 32-bit and dual 16-bit reloadable timers are provided.

The features include:

- Operation as a single 32-bit counter or single/dual 16-bit counter(s).
- Programmable clock prescaler with values from 1 to 4096.
- Non-overlapping pulse width modulated (PWM) output generation with configurable off-time.
- Capture, compare, and capture/compare capability.
- Timer input and output signals available, mapped as alternate functions.
- Configurable input pin for event triggering, clock gating, or capture signal.
- Timer output pin for event output and PWM signal generation.
- Multiple clock source options.

Instances denoted as LPTMR, shown in *Table 16-1*, are configurable to operate in any of the low-power modes and wake the device from the low-power modes to *ACTIVE*.

Each timer supports multiple operating modes:

- One-shot: the timer counts up to terminal value then halts.
- Continuous: the timer counts up to terminal value then repeats.
- Counter: the timer counts input edges received on the timer input pin.
- PWM / PWM differential.
- Capture: the timer captures a snapshot of the current timer count when the timer's input edge transitions.
- Compare: the timer pin toggles when the timer's count exceeds the terminal count.
- Gated: the timer increments only when the timer's input pin is asserted.
- Capture/Compare: the timer counts when the timer input pin is asserted; the timer captures the timer's count when the input pin is deasserted.

16.1 Instances

Instances of the peripheral are listed in *Table 16-1*. Both the TMR and LPTMR are functionally similar, so for convenience all timers are referenced as TMR. The LPTMR instances can function while the device is in certain low-power modes.

Refer to the device data sheet for frequency limitations for external clock sources, if available. Refer to the data sheet for I/O signal configurations and alternate functions for each timer instance.

Table 16-1: MAX32662 TMR/LPTMR Instances

Instance	Register Access Name	Cascade 32-Bit Mode	16-Bit Mode	Operating Modes	CLK0	CLK1	CLK2	CLK3
TMR0	TMR0	Yes	Dual	*ACTIVE SLEEP LPM*	PCLK	HF_EXT_CLK	IBRO	ERFO
TMR1	TMR1							
TMR2	TMR2							
LPTMR0	TMR3	No	Single	*ACTIVE SLEEP DEEPSLEEP*	AOD_PCLK	LP_EXT_CLK	ERTCO	INRO
				BACKUP	N/A	N/A	ERTCO	INRO

Table 16-2: MAX32662 TMR/LPTMR Instances Capture Events

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

16.2 Basic Timer Operation

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

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

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

16.3 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 16-1*. 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 16-3*. Most of the other registers have the same fields duplicated in the upper and lower 16 bits, and are differentiated with the _a and _b suffixes.

In the 32-bit modes, the fields and controls associated with TimerA are used to control the 32-bit timer functionality. In single 16-bit timer mode, the TimerA fields are used to control the single 16-bit timer and the TimerB fields are ignored. In dual 16-bit timer modes, both TimerA and TimerB fields are used to control the dual timers; TimerB fields control the upper 16-bit timer and TimerA fields control the lower 16-bit timer. In dual-16 bit timer modes, TimerB can be used as a single 16-bit timer.

Table 16-3: TimerA/TimerB 32-Bit Field Allocations

Register	Cascade 32-Bit Mode	Dual 16-Bit Mode		Single 16-Bit Mode
Timer Counter	TimerA Count = *TMRn_CNT.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]*

16.4 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 16-1*. 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 16-1: Timer Peripheral Clock Equation

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

The software configures and controls the timer by reading and writing to the timer's registers. External events on timer pins are asynchronous events to the timer's clock frequency. The external events are latched on the next rising edge of the timer's clock. Since it is not possible to externally synchronize to the timer's internal clock, input events may require up to 50% of the timer's internal clock cycle before the hardware recognizes the event.

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

1. Disable the timer peripheral.
   a. Clear *TMRn_CTRL0.en* to 0 to disable the timer.
   b. Read the *TMRn_CTRL1.clken* field until it returns 0, confirming the timer peripheral is disabled.
2. Set *TMRn_CTRL1.clksel* to the new desired clock source.
   a. Note: In cascade 32-bit mode, both the *TMRn_CTRL1.clksel_a* and *TMRn_CTRL1.clksel_b* fields must be set to the same clock source for proper operation.
3. Configure the timer for the desired operating mode. See *Operating Modes* for details on mode configuration.
4. Enable the timer clock source:
   a. Set the *TMRn_CTRL0.clken* field to 1 to enable the timer's clock source.
   b. Read the *TMRn_CTRL1.clkrdy* field until it returns 1, confirming the timer clock source is enabled.
5. Enable the timer:
   a. Set *TMRn_CTRL0.en* to 1 to enable the timer.
   b. Read the *TMRn_CTRL1.clken* field until it returns 1 to confirm the timer is enabled.

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

16.5 Timer Pin Functionality

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

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

The timer pin functionality is mapped as an alternate function that is shared with a GPIO. When the timer pin alternate function is enabled, the timer pin has the same electrical characteristics, such as pullup/pulldown strength, drive strength, etc., as the GPIO mode settings for that pin. Configure the pin characteristics before enabling the timer. When configured as an output, configure the corresponding bit in the GPIO_OUT register to match the inactive state of the timer pin for that mode. Consult the GPIO section for details on how to configure the electrical characteristics for the pin.

The TimerA output controls for modes 0, 1, 3, and 5 output signals are shown in *Figure 16-1*. The TimerA input controls for modes 2, 4, 6, 7, 8, and 14 input signals are shown in *Figure 16-2*.

Figure 16-1: MAX32662 TimerA Output Functionality, Modes 0/1/3/5

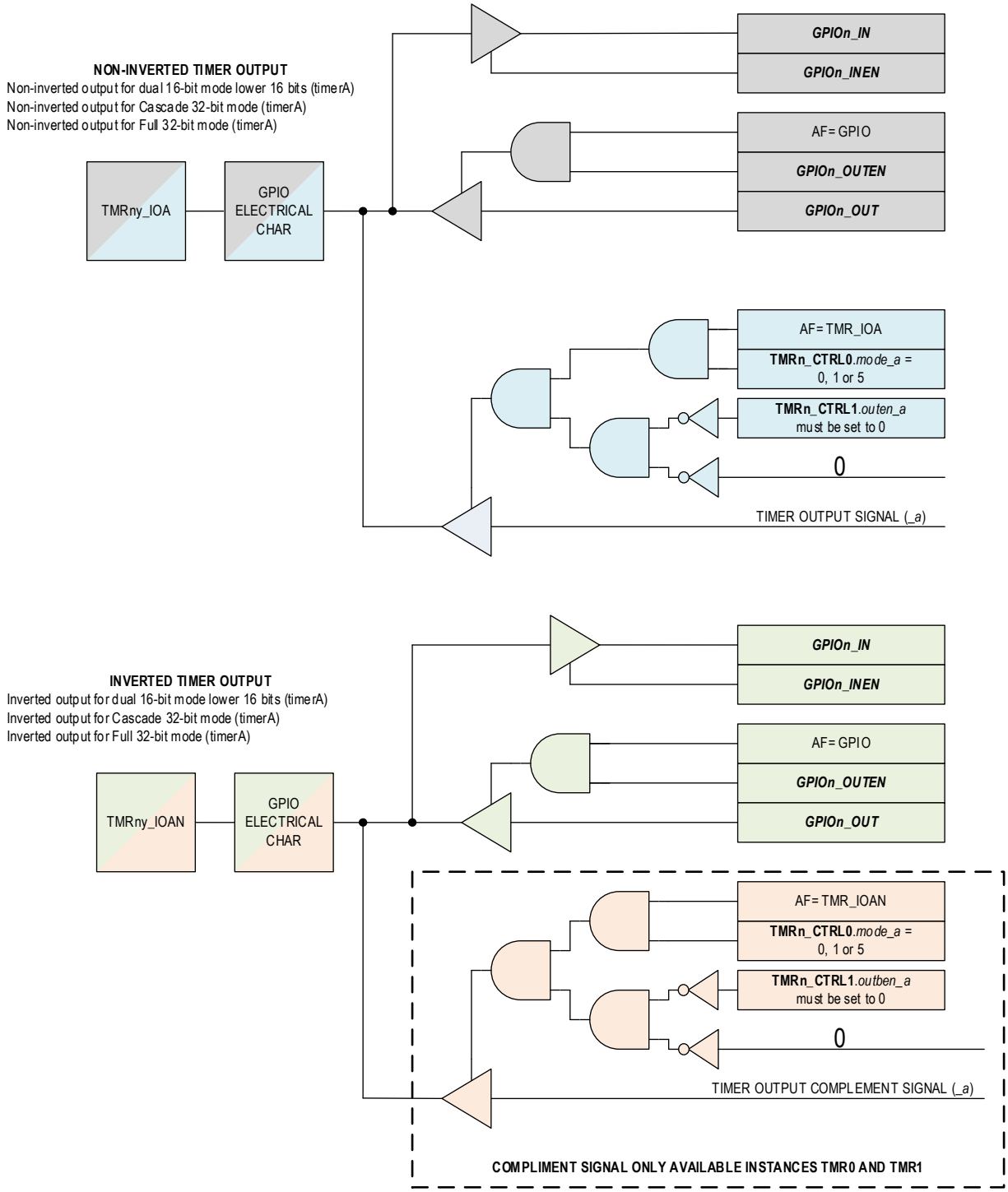
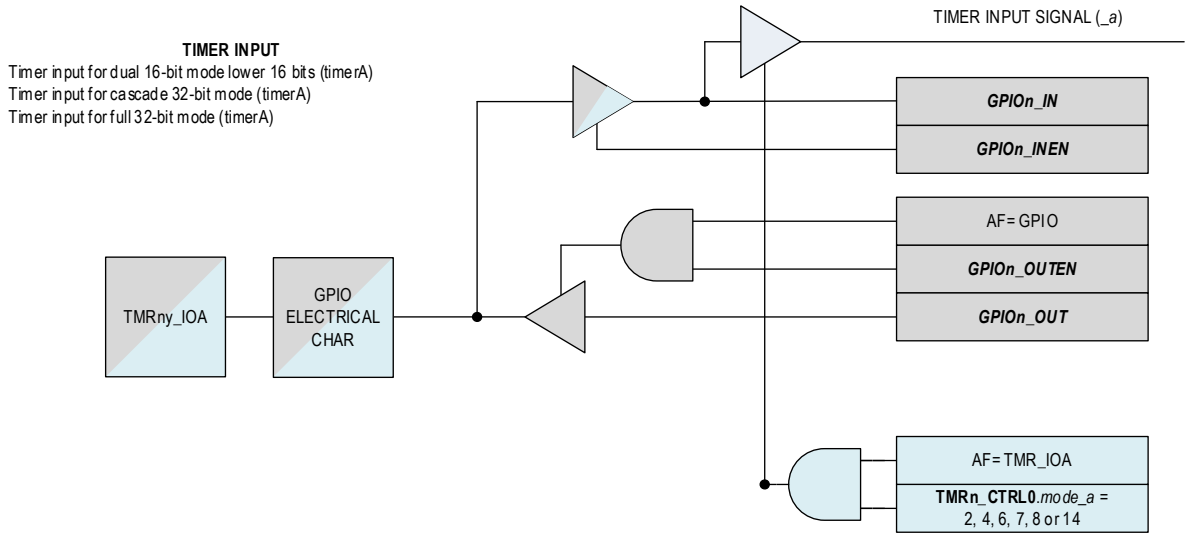


Figure 16-2: MAX32662 TimerA Input Functionality, Modes 2/4/6/7/8/14



16.6 Wake-up Events

In low-power modes, the system clock may be turned off to conserve power. LPTMR instances can continue to run from the clock sources shown in *Table 16-1*. In this case, a wake-up event can be configured to wake up the clock control logic and re-enable the system clock.

Programming sequence example:

1. Disable the timer peripheral and set the timer clock source as described in *Timer Clock Sources*.
2. Configure the timer operating mode as described in the section *Operating Modes*.
3. Enable the timer by setting *TMRn_CTRL0.en* to 1.
4. Poll *TMRn_CTRL1.clkrdy* until it reads 1.
5. Set the *TMRn_CTRL1.we* field to 1 to enable wake-up events for the timer.
6. If desired, enable the timer interrupt and provide a TMRn_IRQn for the timer.
7. Enter a low-power mode as described in the *Operating Modes* section.
8. When the device wakes up from the low-power mode, check the *TMRn_WKFL* register to determine if the timer is the result of the wake-up event.

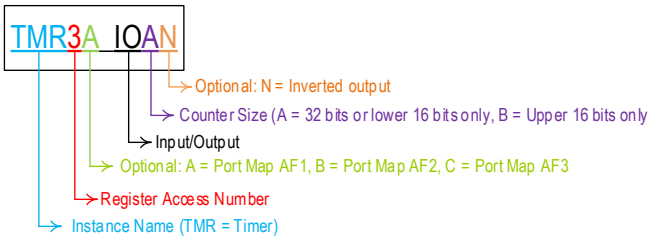
Table 16-4: MAX32662 Wake-up Events

Condition	Peripheral Wake-up Flag *TMRn_INTFL*	Peripheral Wake-up Enable	Low-Power Peripheral Wake-up Flag	Low-Power Peripheral Wake-up Enable	Power Management Wake-up Enable
Any event for LPTMR0	*irq_a*	N/A	*PWRSEQ_LPPWKFL.tmr3*	*PWRSEQ_LPPWKEN.lptmr0*	*GCR_PM.tmr3_we*

16.7 Operating Modes

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

Figure 16-3: Timer I/O Signal Naming Conventions



In *Table 16-5* and *Table 16-6*, the timer's signal name is generically shown where *n* is the timer number (0, 1, 2, 3, etc.) and *y* is the port mapping alternate function. See *Figure 16-3* for details of the timer's naming convention for I/O signals.

Table 16-5: MAX32662 Operating Mode Signals for Timer 0, 1, and 2

Timer Mode	TMR0/TMR1/TMR2 *TMRn_CTRL1.outen_a* = 0 *TMRn_CTRL1.outben_a* = 0	I/O Signal Name[†]	Pin Required
One-Shot Mode (0)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional
Continuous Mode (1)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional
Counter Mode (2)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
PWM Mode (3)	TimerA Output Signal	TMR*ny*_IOA	Yes
	TimerB Output Signal	TMRny_IOB	Yes
Capture Mode (4)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Compare Mode (5)	TimerA Output Signal	TMR*ny*_IOA	Optional
	TimerA Complementary Output Signal	TMR*ny*_IOAN	Optional
	TimerB Output Signal	TMR*ny*_IOB	Optional
	TimerB Complementary Output Signal	TMR*ny*_IOBN	Optional
Gated Mode (6)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Capture/Compare Mode (7)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Dual-Edge Capture Mode (8)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Reserved (9 - 13)	-	-	-

Timer Mode	TMR0/TMR1/TMR2 *TMRn_CTRL1*.*outen_a* = 0 *TMRn_CTRL1*.*outben_a* = 0	I/O Signal Name†	Pin Required
Inactive Gated Mode (14)	TimerA Input Signal	TMR*ny*_IOA	Yes
	TimerB Input Signal	TMR*ny*_IOB	Yes
Reserved (15)	-	-	-

† *See Figure 16-3 for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.*

Table 16-6: MAX32662 Operating Mode Signals for Low-Power Timer 0

Timer mode	LPTMR0 (TMR3) *TMRn_CTRL1*.*outen_a* = 0 *TMRn_CTRL1*.*outben_a* = 0	I/O Signal Name†	Required?
One-Shot Mode (0)	TimerA Output Signal	LPTMR*ny*_IOB	Optional
Continuous Mode (1)	TimerA Output Signal	LPTMR*ny*_IOB	Optional
Counter Mode (2)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
PWM Mode (3)	TimerA Output Signal	LPTMR*ny*_IOB	Yes
Capture Mode (4)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Compare Mode (5)	TimerA Output Signal	LPTMR*ny*_IOB	Optional
Gated Mode (6)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Capture/Compare Mode (7)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Dual-Edge Capture Mode (8)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Reserved (9 - 13)	-	-	-
Inactive Gated Mode (14)	TimerA Input Signal	LPTMR*ny*_IOB	Yes
Reserved (15)	-	-	-

† *See Figure 16-3 for details on the timer I/O signal naming convention and the device data sheet for the alternate functions.*

16.7.1 One-Shot Mode (0)

In one-shot mode, the timer peripheral increments the timer's *TMRn_CNT*.*count* field until it reaches the timer's *TMRn_CMP*.*compare* field and the timer is then disabled. If the timer's output is enabled, the output signal is driven active for one timer clock cycle. One-shot mode provides exactly one timer period and is automatically disabled.

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

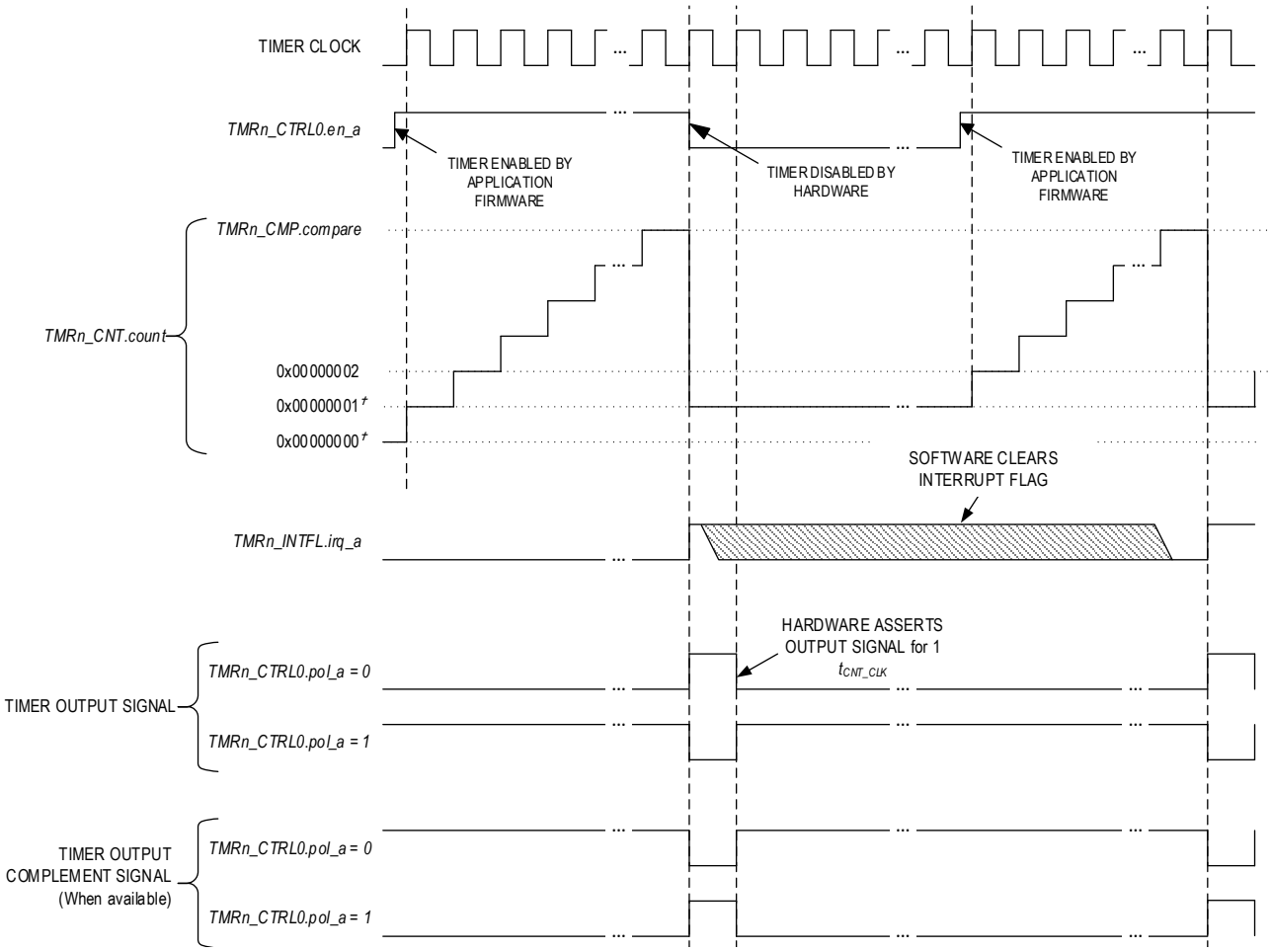
- The *TMRn_CNT*.*count* field is set to 0x0000 0001.
- The timer is disabled (*TMRn_CTRL0*.*en* = 0).
- The timer output, if enabled, is driven to its active state for one timer clock period.
- The *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

The timer period is calculated using *Equation 16-2*.

Equation 16-2: One-Shot Mode Timer Period

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

Figure 16-4: One-Shot Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 0 (One-shot)

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

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

1. Disable the timer peripheral and set the timer clock source as described in *Timer Clock Sources*.
2. Set the *TMRn_CTRL0*.*mode* field to 0 to select one-shot mode.
3. Set the *TMRn_CTRL0*.*clkdiv* field to set the prescaler for the required timer frequency.
4. If using the timer output function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
5. Or, if using the inverted timer output function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.
6. If using the timer interrupt, enable corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to the *TMRn_CMP*.*compare* field.
8. If desired, write an initial value to *TMRn_CNT*.*count* field.
   a. The initial value only affects the first period; subsequent timer periods always reset the *TMRn_CNT*.*count* field to 0x0000 0001.
9. Enable the timer peripheral as described in *Timer Clock Sources*.

16.7.2 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:

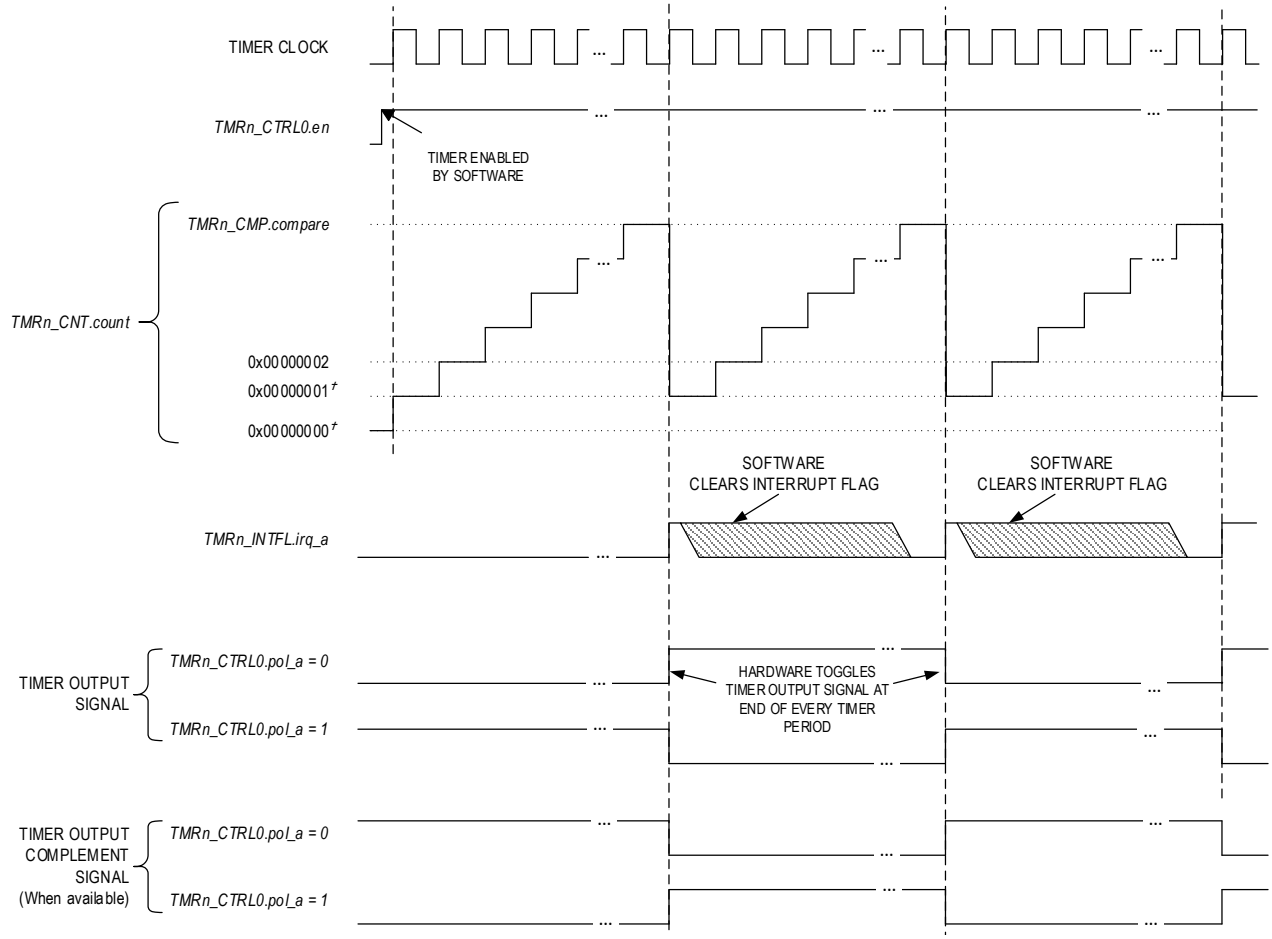
- The *TMRn_CNT*.*count* field is set to 0x0000 0001.
- If the timer output signal is toggled, the corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

The continuous mode timer period is calculated using *Equation 16-3*.

Equation 16-3: Continuous Mode Timer Period

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

Figure 16-5: Continuous Mode Diagram



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

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

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

1. Disable the timer peripheral and set the timer clock as described in *Timer Clock Sources*.
2. Set the *TMRn_CTRL0.mode* field to 1 to select continuous mode.
3. Set the *TMRn_CTRL0.clkdiv* field to set the prescaler that determines the timer frequency.
4. If using the timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
5. Or, if using the inverted timer output function:
   a. Set *TMRn_CTRL0.pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.
6. If using the timer interrupt, enable the corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to the *TMRn_CMP.compare* field.
8. If desired, write an initial value to the *TMRn_CNT*.count field.
   a. The initial value only affects the first period; subsequent timer periods always reset the *TMRn_CNT.count* field to 0x0000 0001.
9. Enable the timer peripheral as described in *Timer Clock Sources*.

16.7.3 Counter Mode (2)

In counter mode, the timer peripheral increments the *TMRn_CNT.count* each time a transition occurs on the timer input signal. The transition must be greater than $4 \times PCLK$ for a count to occur. When the *TMRn_CNT.count* reaches the *TMRn_CMP.compare* field, the hardware automatically sets the interrupt bit to 1 (*TMRn_INTFL.irq)*, sets the *TMRn_CNT.count* field to 0x0000 0001, and continues incrementing. The timer can be configured to increment on either the rising edge or falling edge of the timer's input signal, but not both. Use the *TMRn_CTRL0.pol_* field to select which edge is used for the timer's input signal count.

The timer prescaler setting has no effect in this mode. The frequency of the timer's input signal (f_{CTR_CLK}) must not exceed 25% of the PCLK frequency, as shown in *Equation 16-4*.

Note: If the input signal's frequency is equal to f_{PCLK}, it is possible the transition can be missed by the timer hardware due to PCLK being an asynchronous internal clock. A minimum of 4 PCLK cycles is required for a count to occur. To guarantee a count occurs, the timer input signal should be greater than 4 PCLK cycles.

Equation 16-4: Counter Mode Maximum Clock Frequency

$$f_{CTR\_CLK} \leq \frac{f_{PCLK}\ (Hz)}{4}$$

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

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

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

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

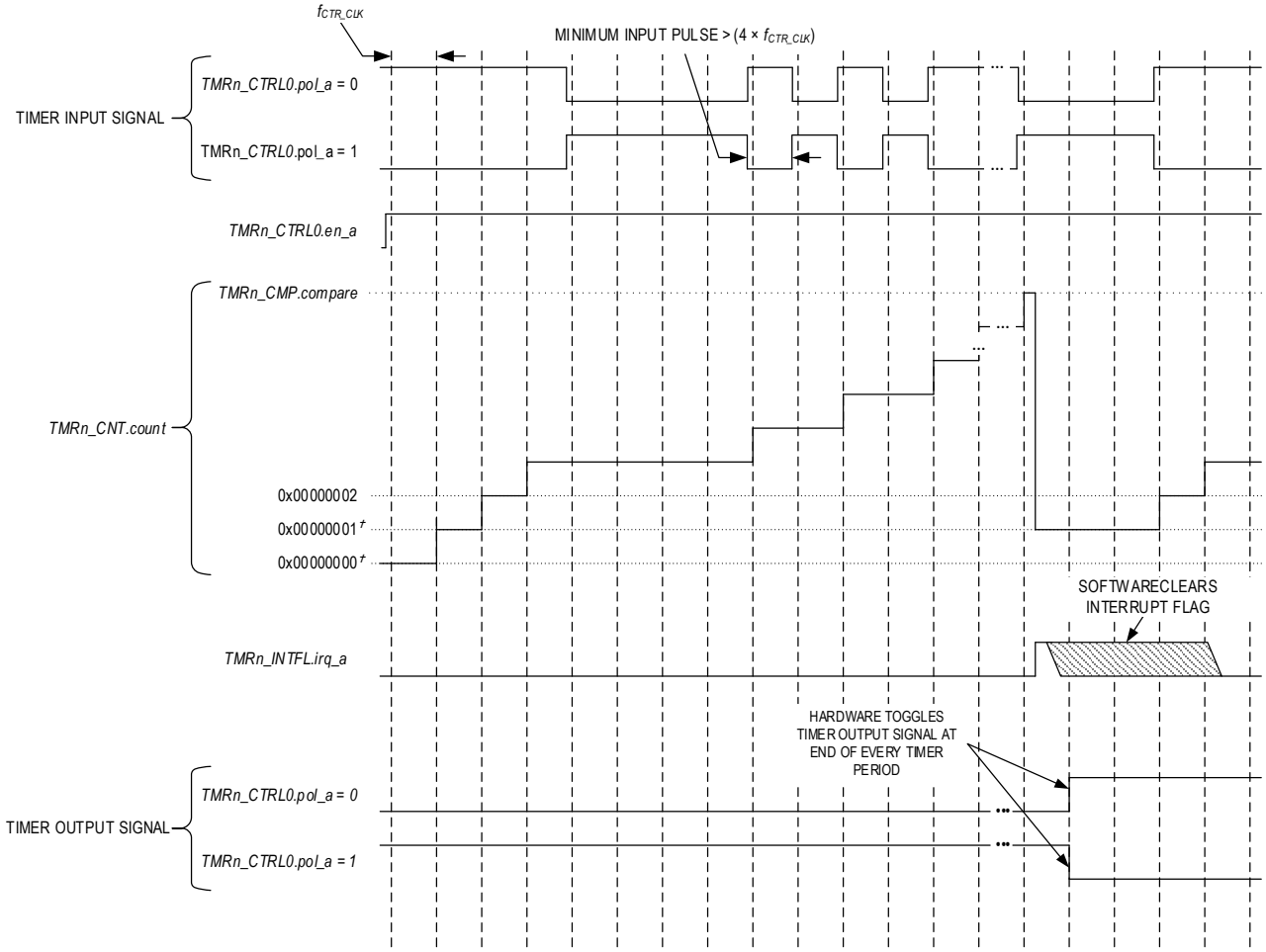
In counter mode, the number of timer input transitions that occurred during a period is equal to the *TMRn_CMP.compare* field's setting. Use *Equation 16-5* to determine the number of transitions that occurred before the end of the timer's period.

Note: Equation 16-5 *is only valid during an active timer count before the end of the timer's period.*

Equation 16-5: Counter Mode Timer Input Transitions

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

Figure 16-6: Counter Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 2 (Counter)

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

Configure the timer for counter mode by performing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* 0x2 to select Counter mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Set *TMRn_CTRL1*.*outen_a* and *TMRn_CTRL1*.*outben* to the values shown in the *Operating Modes* section.
   d. Select the correct alternate function mode for the timer input pin.
5. Write the compare value to *TMRn_CMP*.*compare*.
6. If desired, write an initial value to *TMRn_CNT*.*count*. This affects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0001.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

16.7.4 PWM Mode (3)

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

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

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

- The *TMRn_CNT*.*count* is reset to 0x0000 0001 and the timer resumes counting.
- The timer output signal is toggled.
- The corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

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

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

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

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set the *TMRn_CTRL0.mode field* to 3 to select PWM mode.
4. Set the *TMRn_CTRL0.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. If using the timer in dual 16-bit mode, disable both timers before writing the *TMRn_PWM* register.
10. Set the *TMRn_CMP.compare* value for the PWM second transition period. The *TMRn_CMP.compare* must be greater than the *TMRn_PWM* value.
    a. If using the timer in dual 16-bit mode, disable both timers before writing the *TMRn_CMP* register.
11. If using the timer interrupt, set the interrupt priority and enable the interrupt.
12. Enable the timer peripheral as described in *Timer Clock Sources*.

Equation 16-6 shows the formula for calculating the timer PWM period.

Equation 16-6: Timer PWM Period

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

If an initial starting value other than 0x0000 0001 is loaded into the *TMRn_CNT.count* register, use the one-shot mode equation, *Equation 16-2*, 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 16-7*.

Equation 16-7: Timer PWM Output High Time Ratio with Polarity 0

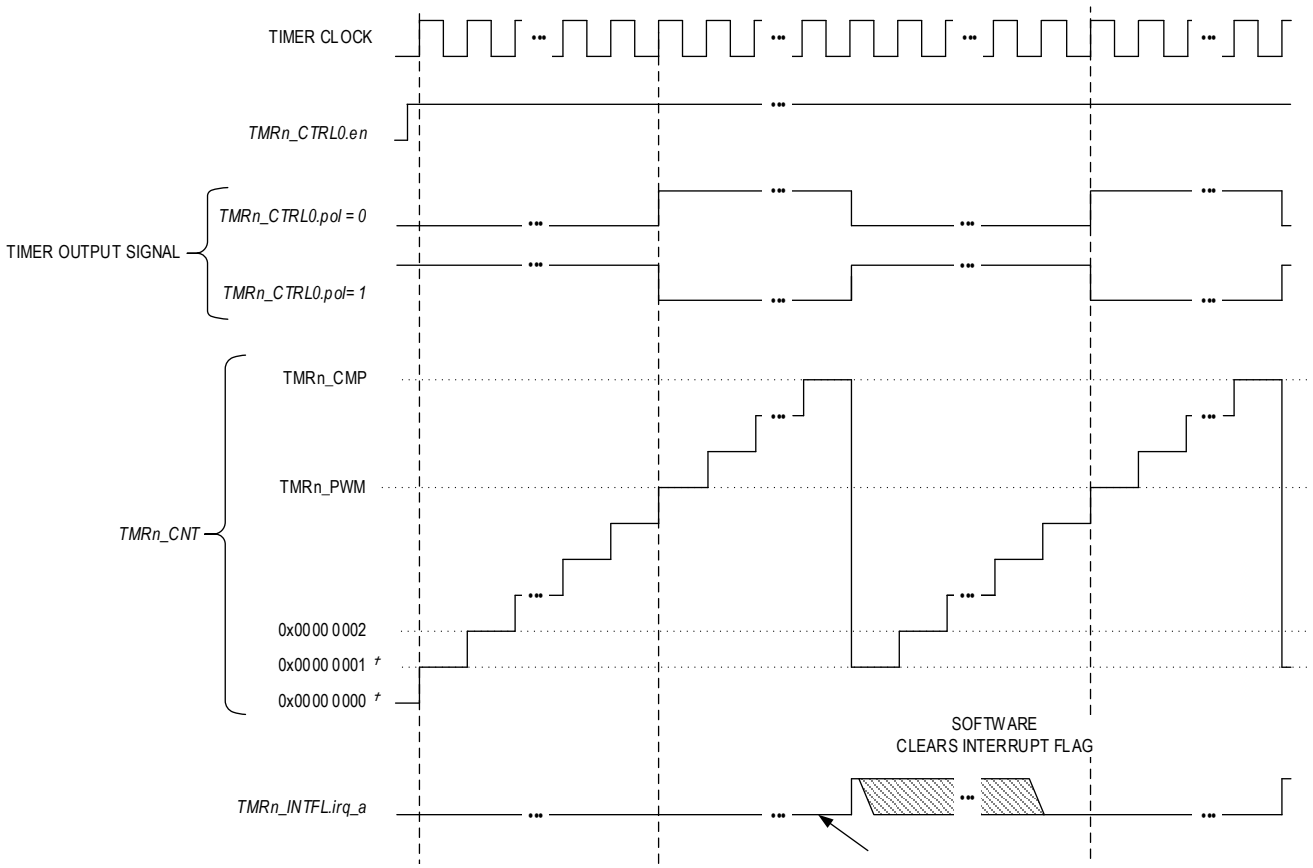
$$PWM\ output\ high\ time\ ratio\ (\%) = \frac{(TMR\_CMP - TMR\_PWM)}{TMR\_CMP} \times 100$$

If *TMRn_CTRL0.pol* is set to 1, the ratio of the PWM output high time to the total period is calculated using *Equation 16-8*.

Equation 16-8: Timer PWM Output High Time Ratio with Polarity 1

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

Figure 16-7: PWM Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 3 (PWM)

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

16.7.5 Capture Mode (4)

Capture mode is used to measure the time between software-determined events. The timer starts incrementing the timer's count field until a transition occurs on the timer's input pin or a rollover event occurs. A capture event is triggered by the hardware when the timer's input pin transitions state. *Equation 16-9* shows the formula for calculating the capture event's elapsed time.

If a capture event does not occur before the timer's count value reaching the timer's compare value (*TMRn_CNT*.*count* = *TMRn_CMP*.*compare*), a rollover event occurs. Both the capture event and the rollover event set the timer's interrupt flag, *TMRn_INTFL*.*irq*, to 1 and result in an interrupt if the timer's interrupt is enabled.

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

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

16.7.5.1 Capture Event

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

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

The software must check the value of the TMRn_PWM.pwm field to determine the trigger of the timer interrupt.

Equation 16-9: Capture Mode Elapsed Time Calculation in Seconds

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

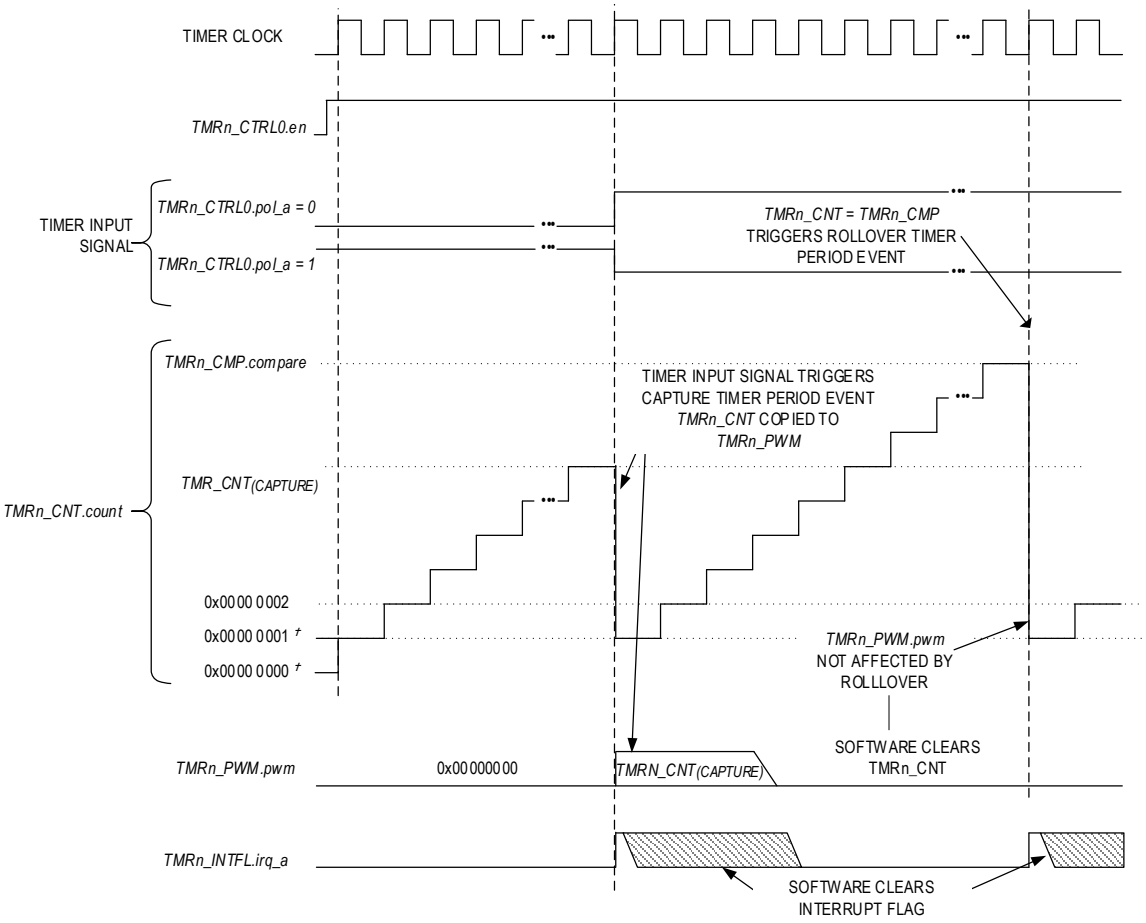
Note: The capture elapsed time calculation is only valid after the capture event occurs and the timer stores the captured count in the TMRn_PWM register.

16.7.5.2 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 16-8: Capture Mode Diagram



THIS EXAMPLES USES THE FOLLOWING CONFIGURATION IN ADDITION TO THE SETTINGS SHOWN ABOVE:
TMRn_CTRL1.cascade = 1 (32-BIT CASCADE TIMER)
TMRn_CTRL0.mode_a = 2 (COUNTER)

† *TMRn*_CNT.*count* DEFAULTS TO 0x00000000 ON A TIMER RESET. *TMRn_CNT.count* RELOADS TO 0x00000001 FOR ALL FOLLOWING TIMER PERIODS.

Configure the timer for Capture mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* to 4 to select capture mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. Write the initial value to *TMRn_CNT*.*count*, if desired.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0001.
6 Write the compare value to the *TMRn_CMP*.*compare* field.
7. Select the capture event by setting *TMRn_CTRL1*.*capevent_sel*.
8. Enable the timer peripheral as described in *Timer Clock Sources*.

The timer period is calculated using the following equation:

Equation 16-10: 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.

16.7.6 Compare Mode (5)

In compare mode, the timer peripheral increments continually from 0x0000 0000 (after the first timer period) to the maximum value of the 32- or 16-bit mode, then rolls over to 0x0000 0000 and continues incrementing. The end of the timer period event occurs when the timer value matches the compare value, but the timer continues to increment until the count reaches 0xFFFF FFFF. The timer counter then rolls over and continues counting from 0x0000 0000.

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

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

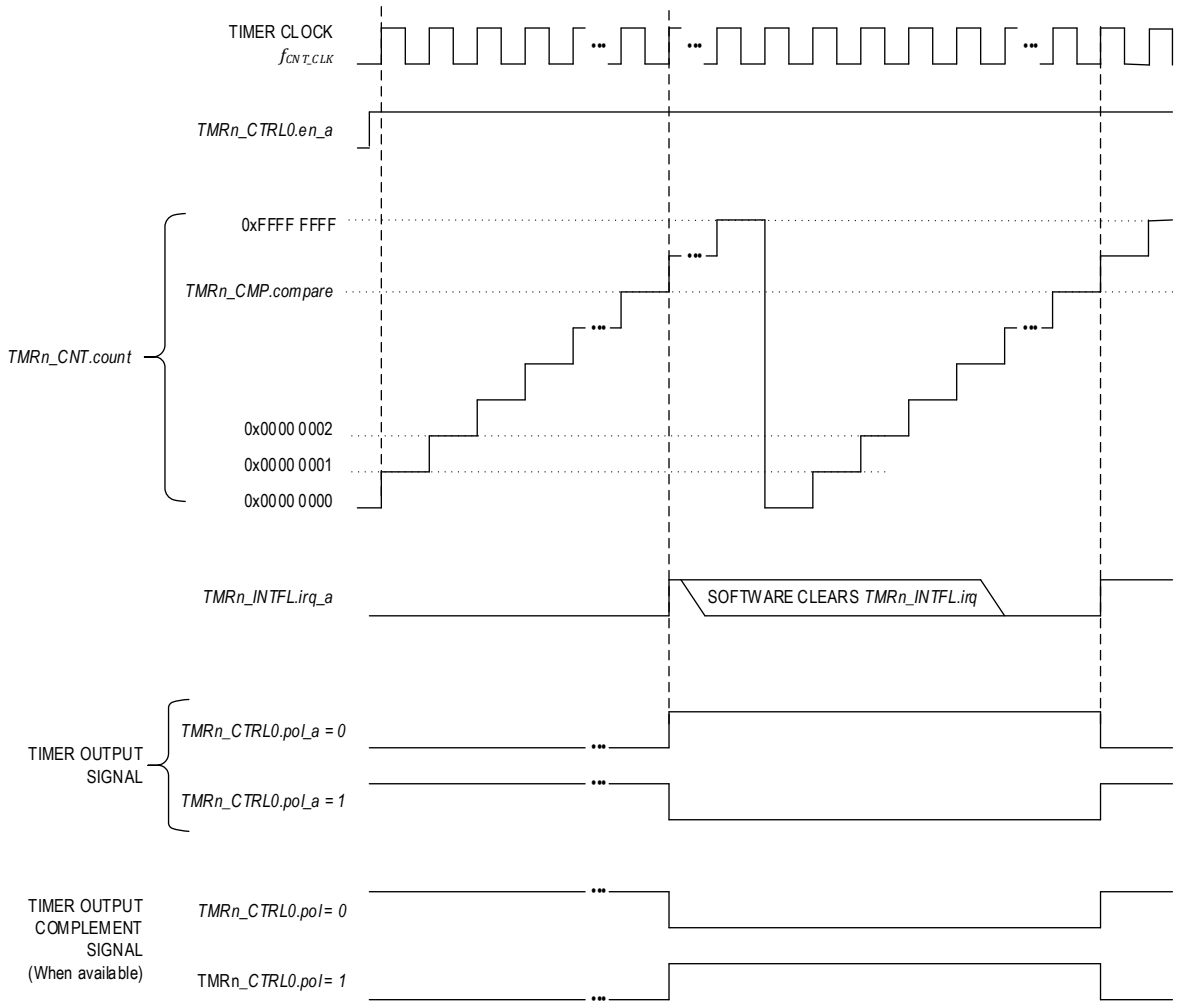
- Unlike other modes, *TMRn_CNT*.*count* is reset to 0x0000 00000, not 0x0000 0001 at the end of the timer period. The timer remains enabled and continues incrementing.
- The corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.
- The hardware toggles the state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

The compare mode timer period is calculated using *Equation 16-11*.

Equation 16-11: Compare Mode Timer Period

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

Figure 16-9: Compare Mode Diagram



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

Configure the timer for compare mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* to 5 to select Compare mode.
4. Set *TMRn_CTRL0*.*clkdiv* to set the prescaler that determines the timer frequency.
5. If using the timer output function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer output pin.
6. If using the inverted timer output function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired (inactive) state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the inverted timer output pin.
7. If using the timer interrupt, enable the corresponding field in the *TMRn_CTRL1* register.
8. Write the compare value to *TMRn_CMP*.*compare.*
9. If desired, write an initial value to *TMRn_CNT*.*count*.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0000.
10. Enable the timer peripheral as described in *Timer Clock Sources*.

16.7.7 Gated Mode (6)

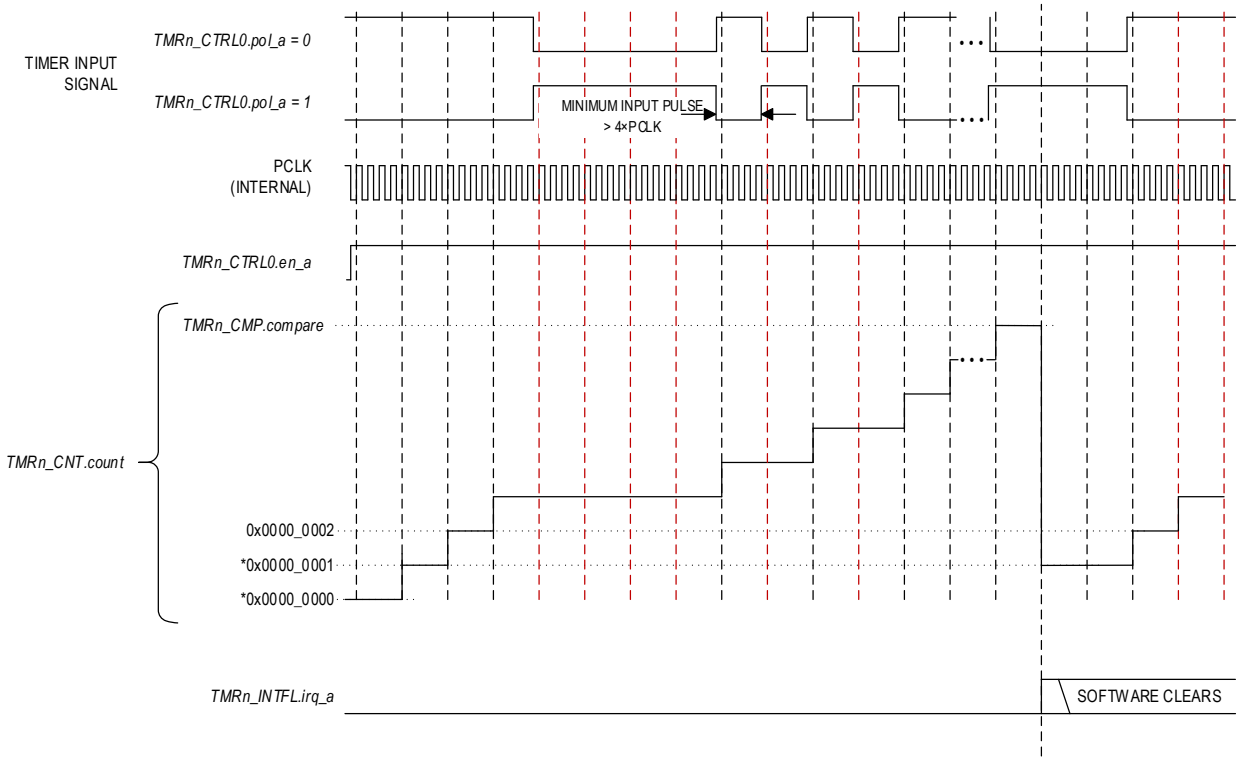
Gated mode is similar to continuous mode, except that *TMRn_CNT*.*count* only increments when the timer input signal is in its active state.

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

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

- The *TMRn_CNT*.*count* field is set to 0x0000 0001.
- The timer remains enabled and continues incrementing.
- The timer output pin changes state if the timer output is enabled.
- The corresponding *TMRn_INTFL*.*irq* field is set to 1 to indicate a timer interrupt event occurred.

Figure 16-10: Gated Mode Diagram



This examples uses the following configuration in addition to the settings shown above:
TMRn_CTRL1.cascade = 1 (32-bit Cascade Timer)
TMRn_CTRL0.mode_a = 6 (Gated)

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

Configure the timer for gated mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0*.*mode* to 6 to select gated mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0*.*pol* to match the desired inactive state.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. If desired, write an initial value to the *TMRn_CNT*.*count* field.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT*.*count* = 0x0000 0001.
6 Write the compare value to *TMRn_CMP*.*compare*.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

16.7.8 Capture/Compare Mode (7)

In Capture/Compare mode, the timer starts counting after the first external timer input transition occurs. The transition, a rising edge or falling edge on the timer's input signal, is set using the *TMRn_CTRL0.pol* bit.

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

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

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

The actions performed at the end of the timer period are dependent on the event that ended the timer period.

If the end of the timer period is caused by a transition on the timer pin, the hardware automatically performs the following:

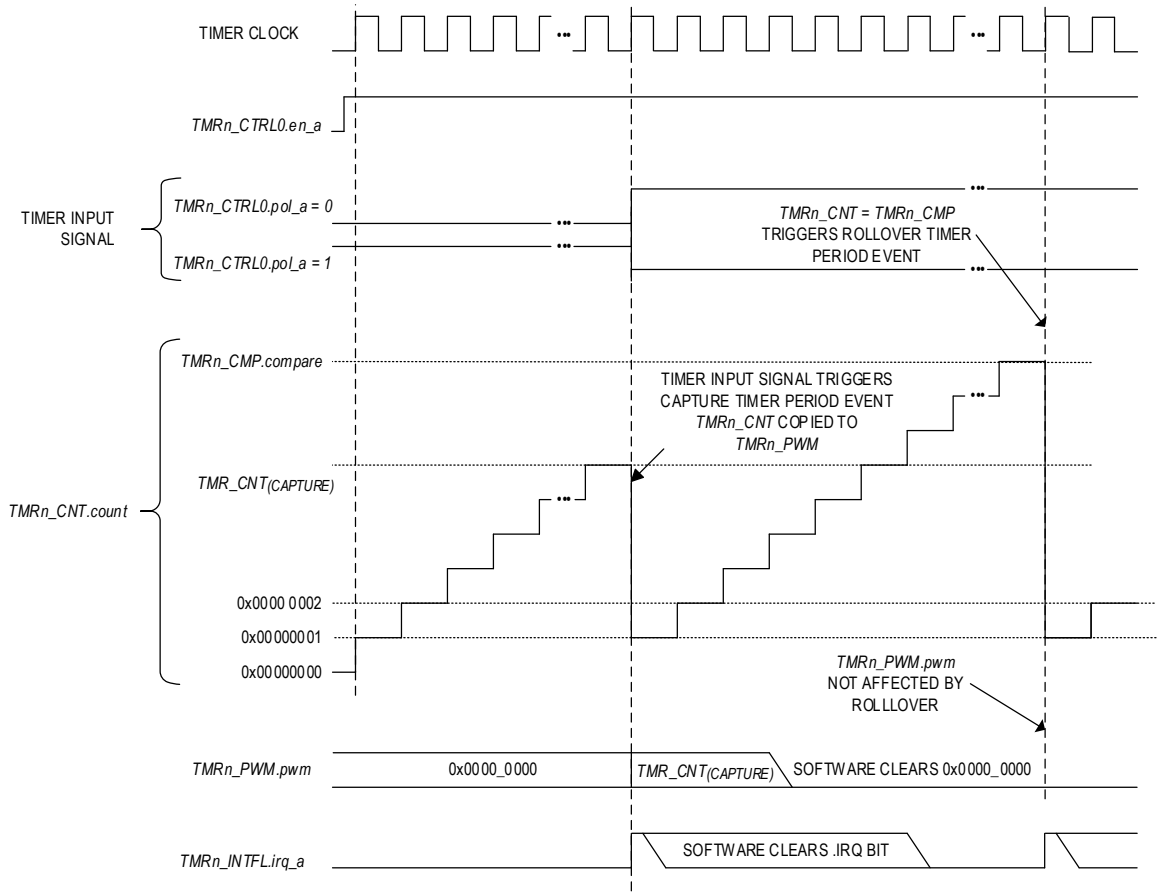
- The value in *TMRn_CNT.count* field is copied to the *TMRn_PWM.pwm* field.
- The *TMRn_CNT.count* field is set to 0x0000 0001.
- The timer remains enabled and continues incrementing.
- The corresponding *TMRn_INTFL.irq* field is set to 1 to indicate a timer interrupt event occurred.

In capture/compare mode, the elapsed time from the timer start to the capture event is calculated using *Equation 16-12*.

Equation 16-12: Capture Mode Elapsed Time

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

Figure 16-11: Capture/Compare Mode Diagram



THIS EXAMPLES USES THE FOLLOWING CONFIGURATION IN ADDITION TO THE SETTINGS SHOWN ABOVE:
TMRn_CTRL1.cascade = 1 (32-BIT CASCADE TIMER)
TMRn_CTRL0.mode_a = 7 (CAPTURE/COMPARE)

† *TMRn_CNT*.count DEFAULTS TO 0x00000000 ON A TIMER RESET. *TMRn_CNT*.count RELOADS TO 0x00000001 FOR ALL FOLLOWING TIMER PERIODS.

Configure the timer for Capture/Compare mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Sources*.
2. If desired, change the timer clock source as described in *Timer Clock Sources*.
3. Set *TMRn_CTRL0.mode* to 7 to select Capture/Compare mode.
4. Configure the timer input function:
   a. Set *TMRn_CTRL0.pol* to select the positive edge (*TMRn_CTRL0.pol* = 1) or negative edge (*TMRn_CTRL0.pol* = 0) transition to cause the capture event.
   b. Configure the GPIO electrical characteristics as desired.
   c. Select the correct alternate function mode for the timer input pin.
5. If desired, write an initial value to the *TMRn_CNT.count* field.
   a. This affects only the first period; subsequent timer periods always reset *TMRn_CNT.count* = 0x0000 0001.
6 Write the compare value to *TMRn_CMP.compare*.
7. Enable the timer peripheral as described in *Timer Clock Sources*.

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

16.7.9 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.

16.7.10 Inactive Gated Mode (14)

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

16.8 Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 16-7*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.
See *Table 1-1* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 16-7: 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

16.8.1 Register Details

Table 16-8: Timer Count Register

Timer Count			TMRn_CNT	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	count	R/W	0	**Timer Count** This field increments at a rate dependent on the selected timer operating mode. The function of the bits in this field are dependent on the 32-bit/16-bit configuration. Reads of this register always return the current value.

Table 16-9: Timer Compare Register

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

Table 16-10: Timer PWM Register

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

Table 16-11: Timer Interrupt Register

Timer Interrupt			TMRn_INTFL	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved**
25	wr_dis_b	R/W	0	**TimerB Write Protect in Dual Timer Mode** Set this field to 0 to write protect the TimerB fields in the *TMRn_CNT.count[31:16]* and *TMRn_PWM.pwm[31:16]*. When this field is set to 0, 32-bit writes to the *TMRn_CNT* and *TMRn_PWM* registers only modify the lower 16 bits associated with TimerA. 0: Enabled. 1: Disabled. *Note: This field always reads 0 if the timer is configured as a 32-bit cascade timer.*

Timer Interrupt			TMRn_INTFL	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
24	wrdone_b	R	0	**TimerB Write Done** This field is cleared to 0 by the hardware when the software performs a write to *TMRn_CNT.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	**TimerA Dual Timer Mode Write Protect** This field disables write access to the *TMRn_CNT.count[15:0]* and *TMRn_PWM.pwm[15:0]* fields so that only the 16 bits associated with updating TimerA are modified during writes to the *TMRn_CNT* and *TMRn_PWM* registers. 0: Enabled. 1: Disabled. *Note: This field always reads 0 if the timer is configured as a 32-bit cascade timer.*
8	wrdone_a	R	0	**TimerA Write Done** This field is cleared to 0 by the hardware when the software performs a write to *TMRn_CNT.count[15:0]* or *TMRn_PWM.pwm[15:0]* when in dual 16-bit timer mode. Wait until the field reads 1 before proceeding. 0: Operation in progress. 1: Operation complete.
7:1	-	RO	0	**Reserved**
0	irq_a	W1C	0	**TimerA Interrupt Event** This field is set when a TimerA interrupt event occurs. Write 1 to clear. 0: No event. 1: Interrupt event occurred.

Table 16-12: Timer Control 0 Register

Timer Control 0			TMRn_CTRL0	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31	en_b	R/W	0	**TimerB Enable** 0: Disabled. 1: Enabled.
30	clken_b	R/W	0	**TimerB Clock Enable** 0: Disabled. 1: Enabled.
29	rst_b	R/W1O	0	**TimerB Reset** 0: Normal operation. 1: Reset Timer B.
28:24	-	RO	0	**Reserved**

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

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

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

Table 16-14: Timer Control 1 Register

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31	cascade	R/W	0	**32-bit Cascade Timer Enable** This field is only supported by Timer instances with support for 32-bit cascade mode. 0: Dual 16-bit timers 1: 32-bit cascade timer
30:29	-	R/W	0	**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 16-2* 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.

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
19	clkrdy_b	RO	0	**TimerB Clock Ready Status** This field indicates if the timer clock is ready. 0: Timer clock not ready or synchronization in progress. 1: Timer clock is ready.
18	clken_b	RO	0	**TimerB Clock Enable Status** This field indicates the status of the timer enable. 0: Timer not enabled or synchronization in progress. 1: Timer is enabled.
17:16	clksel_b	R/W	0	**TimerB Clock Source** See *Table 16-1* for the clock sources supported by each instance. *Note: In cascade 32-bit mode, this field must be set to the same value as the TMRn_CTRL1.clksel_a field.* 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.
15	-	RO	0	**Reserved**
14	outben_a	R/W	0	**Output B Enable** Reserved.
13	outen_a	R/W	0	**Output Enable** Reserved.
12	we_a	R/W	0	**TimerA Wake-up Function** 0: Disabled. 1: Enabled.
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 16-2* 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 the software enables the TimerA clock by writing 1 to the *TMRn_CTRL1.clken_a* field. 0: Timer not enabled or synchronization in progress. 1: TimerA clock is ready.

Timer Control 1			TMRn_CTRL1	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
2	clken_a	R/W	0	**TimerA Clock Enable** Write this field to 1 to enable the TimerA clock. 0: Timer not enabled or synchronization in progress. 1: Timer is enabled.
1:0	clksel_a	R/W	0	**Clock Source TimerA** See *Table 16-1* for the available clock options for each timer instance. *Note: In cascade 32-bit mode, set TMRn_CTRL1.clksel_b to the same value as this field for proper operation.* 0: Clock option 0. 1: Clock option 1. 2: Clock option 2. 3: Clock option 3.

Table 16-15: 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.

17. Watchdog Timer (WDT)

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

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

Each time the application performs a reset as early as possible in the application software, examine the peripheral control register to determine if the reset was caused by a WDT late reset event or a WDT early reset event if the window function is enabled. If so, the software should take the desired action as part of its restart sequence.

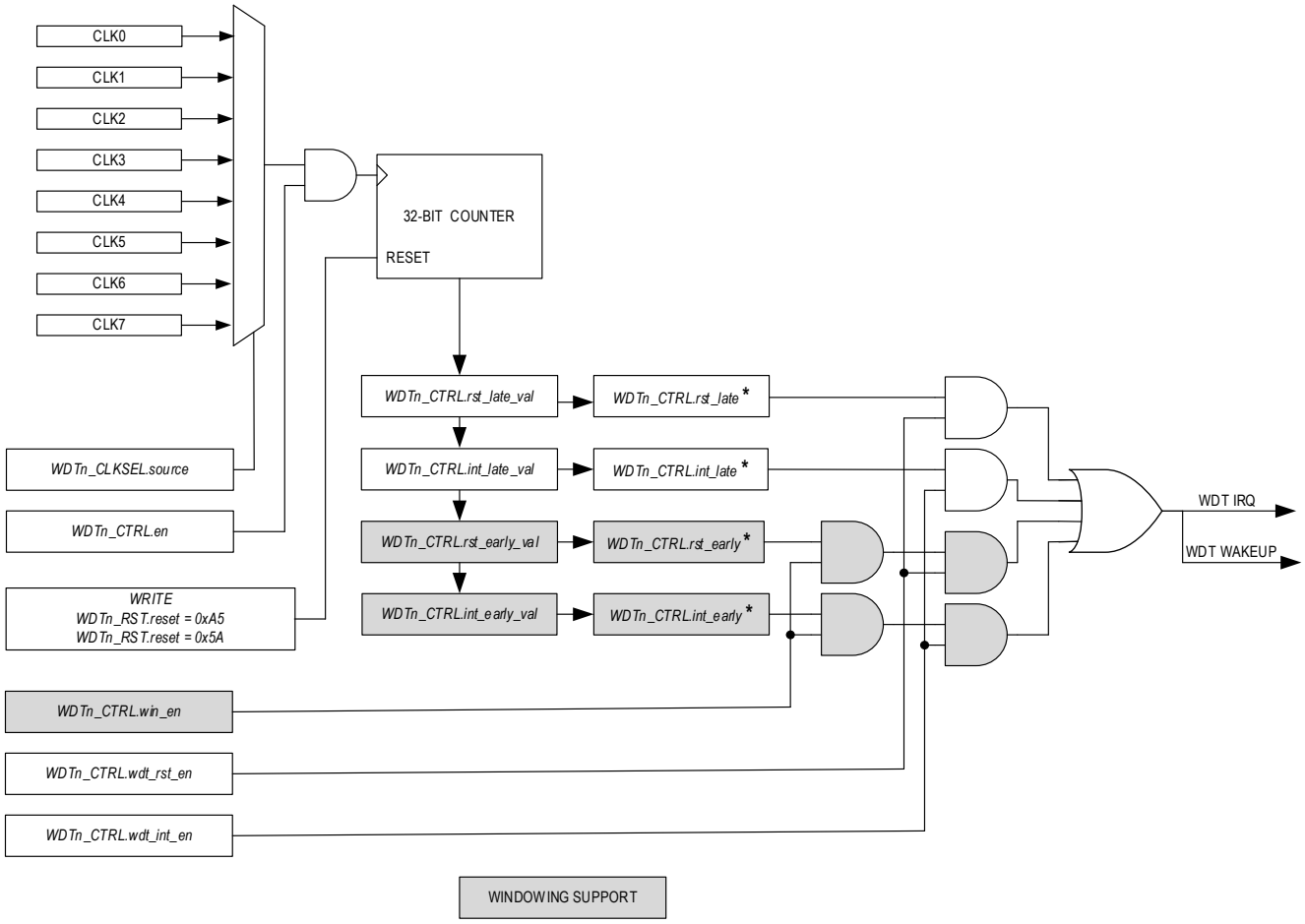
The WDT is a critical safety feature; most fields are reset on POR or system reset events only.

Features:

- Single-ended (legacy) watchdog timeout
- Windowed mode adds lower-limit timeout settings to detect loss of control in tight code loops.
- Configurable clock source
- Configurable time-base
- Programmable upper and lower limits for reset and interrupts from 2^{16} to 2^{32} time-base ticks.

Figure 17-1 shows a high-level block diagram of the WDT.

Figure 17-1: Windowed Watchdog Timer Block Diagram



* INTERRUPT FLAGS ARE SET REGARDLESS OF THE ENABLED STATE OF WDTn_CTRL.win_en, WDTn_CTRL.wdt_int_en and WDTn_CTRL.wdt_rst_en.

17.1 Instances

Table 17-1 shows the peripheral instances, available clock sources, and windowed watchdog support.

Table 17-1: MAX32662 WDT Instances Summary

Instance	Register Access Name	Window Support	CLK0	CLK1	CLK2	CLK3	CLK4	CLK5	CLK6
WDT	WDT	Yes	PCLK	IPO	IBRO	NANO	ERTCO	HF_EXT_CLK P0.6 AF4	ERFO

17.2 Usage

When enabled, *WDTn_CNT.count* is incremented once every t_{WDTCLK} period. The software periodically executes the feed sequence during correct operation, resetting the *WDTn_CNT.count* field to 0x0000 0000 within the target window.

The upper and lower limits of the target window are user-configurable to accommodate different applications and non-deterministic execution times within an application.

The WDT can generate interrupts and/or reset events in response to the WDT activity. Interrupts are typically configured to respond first to an event outside the target window. The approach is that a minor system event can have temporarily delayed the execution of the feed sequence so that the event can be diagnosed in an interrupt routine and control returned

to the system. When the WDT feed sequence occurs much earlier than expected or not at all, a reset event can be generated that forces the system to a known good state before continuing.

Traditional WDTs only detect execution errors that fail to perform the WDT feed sequence. If the counter reaches the WDT late interrupt threshold, the device attempts to regain program control by vectoring to the dedicated WDT interrupt service routine (ISR). The ISR should reset the WDT counter, perform the desired recovery activity, and then return execution to a known good address.

If the execution error prevents the successful execution of the ISR, the WDT continues to increment until the count reaches the WDT late reset threshold. The WDT generates a late reset event that sets the WDT late reset flag and generates a system interrupt.

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

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

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

17.2.1 Using the WDT as a Long-Interval Timer

One application of the WDT is a very long interval timer in *ACTIVE*. 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.

17.2.2 Using the WDT as a Long-Interval Wake-up Timer

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

17.3 WDT Protection Sequence

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

17.3.1 WDT Feed Sequence

Two consecutive write instructions to the *WDTn_RST.reset* field are required to reset the *WDTn_CNT.count* = 0. Disable global interrupts immediately before and re-enable after writing to ensure both writes to the *WDTn_RST.reset* field complete without interruption.

1. Disable interrupts.
2. In consecutive write operations:
   a. Write *WDTn_RST.reset*: 0xA5.
   b. Write *WDTn_RST.reset*: 0x5A.
3. Hardware automatically clears the *WDTn_CNT.count* to 0.
3. Re-enable interrupts.

17.3.2 WDT Enable Sequence

Perform the enable sequence immediately before enabling the WDT to prevent accidental triggering of the reset or interrupt as soon as the timer is enabled. There is no timed access window for these write operations; the operations can be separated by any length of time as long as they occur in the required sequence.

1. Write the *WDTn_RST.reset* field with 0xFE.
2. Write the *WDTn_RST.reset* field with 0xED.
3. The hardware sets *WDTn_CTRL.en* to 1 automatically.

17.3.3 WDT Disable Sequence

Perform the disable sequence immediately before disabling the WDT to prevent accidental disabling of the WDT by software. There is no timed access window for these write operations; the operations can be separated by any length of time as long as they occur in the required sequence.

1. Write the *WDTn_RST.reset* field with 0xDE.
2. Write the *WDTn_RST.reset* field with 0xAD.
3. The hardware clears *WDTn_CTRL.en* to 0 automatically.

17.4 WDT Events

Multiple events are supported, as shown in *Table 17-2*. The corresponding event flag is set when the event occurs.

Typically, the system is configured such that the late interrupt events occur before the late reset events, and early interrupts occur when the feed sequence has the least error from the target time before the early reset events.

The event flags are set even if the corresponding interrupt enable or reset enable are not enabled, and include the early interrupt flag and early event flag even if the window feature is disabled (*WDTn_CTRL.win_en* = 0).

The software must clear the event flags before enabling the WDT.

Table 17-2: WDT Event Summary

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

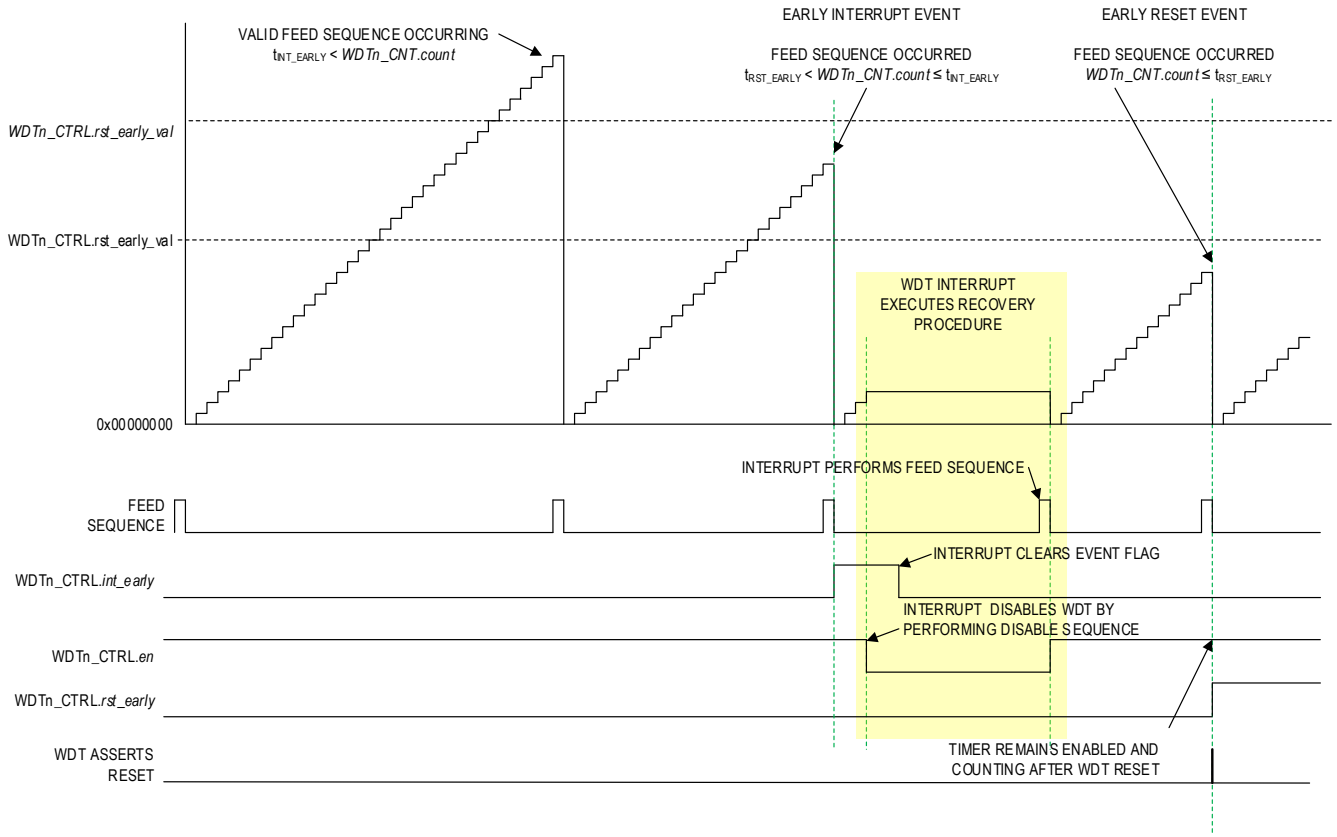
Event	Condition	Peripheral Interrupt Event Flag	Peripheral Interrupt Event Enable
Interrupt Late	*WDTn_CNT.count* = *WDTn_CTRL.int_late_val*	*WDTn_CTRL.int_late*	*WDTn_CTRL.wdt_int_en*
Reset Late	*WDTn_CNT.count* = *WDTn_CTRL.rst_late_val*	*WDTn_CTRL.rst_late*	*WDTn_CTRL.wdt_rst_en*
Timer Enabled	*WDTn_CTRL.clkrdy* 0 → 1	No event flags are set by a timer enabled event	

17.4.1 WDT Early Reset

The early reset event occurs if the software performs the WDT feed sequence while the WDT count is less than the reset late value (*WDTn_CNT.count* < *WDTn_CTRL.rst_late_val*).

Figure 17-2 shows the sequencing details associated with an early reset event.

Figure 17-2: WDT Early Interrupt and Reset Event Sequencing Details



The following occurs when a WDT early reset event occurs:

1. The hardware sets *WDTn_CTRL.rst_early* to 1.
2. The hardware initiates a system reset.
   a. The hardware resets *WDTn_CNT.count* to 0x0000 0000 during the system reset event.
   b. The *WDTn_CTRL.en* and the *WDTn_CTRL.rst_early* fields are unaffected by a system reset.
3. After the system reset is complete, the WDT continues incrementing.

17.4.2 WDT Early Interrupt

The early interrupt event occurs if the software performs the WDT feed sequence while *WDTn_CTRL.rst_early_val* ≤ *WDTn_CNT.count* < *WDTn_CTRL.int_early_val,* as shown in *Table 17-2*. *Figure 17-2* shows the sequencing details associated with an early reset event, including:

- The sequencing details associated with an early interrupt event.
- The required functions performed by the WDT interrupt handler.

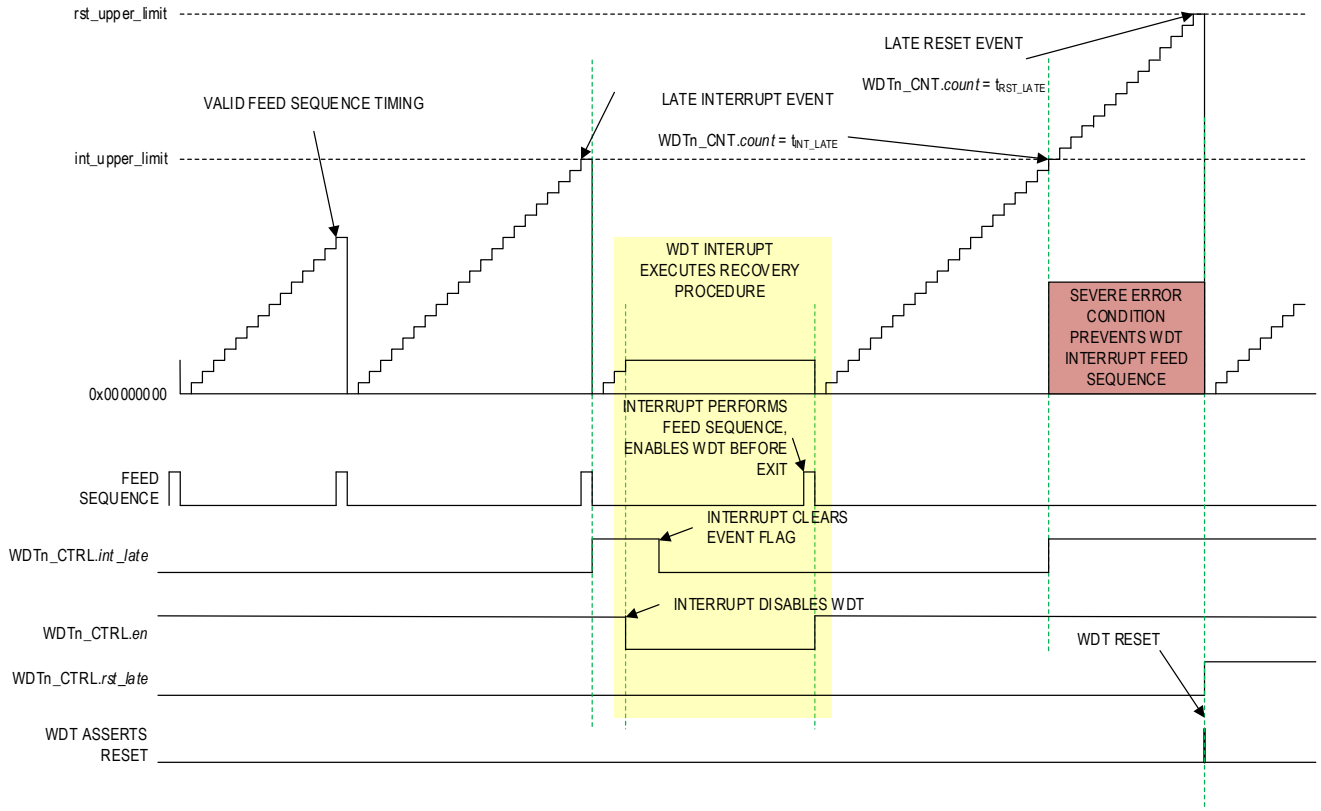
The following occurs when a WDT late interrupt event occurs:

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

17.4.3 WDT Late Reset

The late reset event occurs if the counter increments to the point where *WDTn_CNT.count* = *WDTn_CTRL.rst_late* threshold, as shown in *Table 17-2*. *Figure 17-3* shows the sequencing details associated with a late reset event.

Figure 17-3: WDT Late Interrupt and Reset Event Sequencing Details



The following occurs when a WDT late reset event occurs:

1. The hardware sets *WDTn_CTRL.rst_late* to 1.
2. The hardware initiates a system reset:
   a. The hardware resets *WDTn_CNT.count* to 0x0000 0000 during the reset event.
   b. The *WDTn_CTRL.en* and *WDTn_CTRL.rst_late* fields are unaffected by a system reset.
3. After the hardware exits the system reset, the WDT continues incrementing after the system reset completes.

17.4.4 WDT Late Interrupt

The late reset event occurs if the counter increments to the point where *WDTn_CNT.count* = *WDTn_CTRL.rst_late* threshold as shown in *Table 17-2*. *Figure 17-3* shows the sequencing details associated with a late interrupt event, including the required functions performed by the WDT interrupt handler.

The following occurs when WDT late interrupt event occurs:

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

17.5 Initializing the WDT

The complete procedure for configuring the WDT is as follows:

1. Execute the WDT disable sequence and disable the WDT:
   a. Disable global interrupts.
   b. Write *WDTn_RST.reset* to 0xDE.
   c. Write *WDTn_RST.reset* to 0xAD.
   d. The hardware automatically clears *WDTn_CTRL.en* to 0, disabling the WDT.
   e. Re-enable global interrupts.
2. Verify the peripheral is disabled before proceeding:
   a. Poll *WDTn_CTRL.clkrdy* until it reads 1.
3. Set *WDTn_CTRL.clkrdy_ie* = 1 to generate a WDT enabled interrupt event.
4. Configure *WDTn_CLKSEL.source* to select the clock source.
5. Configure the standard thresholds:
   a. Configure *WDTn_CTRL.int_late* to the desired threshold for the WDT late interrupt event.
   b. Configure *WDTn_CTRL.rst_late_val* to the desired threshold for the WDT late reset event.
6. If using the optional windowed WDT feature:
   a. Set *WDTn_CTRL.win_en* = 1 to enable the windowed WDT feature.
   b. Configure *WDTn_CTRL.int_early_val* to the desired threshold for the WDT early interrupt event.
   c. Configure *WDTn_CTRL.rst_early_val* to the desired threshold for the WDT early reset event.
7. Set *WDTn_CTRL.wdt_int_en* to generate an interrupt when a WDT late interrupt event occurs. If *WDTn_CTRL.win_en* = 1, an interrupt is generated by both a WDT late interrupt event, and a WDT early interrupt event.
8. Set *WDTn_CTRL.wdt_rst_en* to generate an interrupt when a WDT late reset event occurs. If *WDTn_CTRL.win_en* = 1, an interrupt is generated by a WDT late reset event and a WDT early reset event.
9. Execute the WDT feed sequence to reset the WDT counter.
   a. Write *WDTn_RST.reset* to 0xA5.
   b. Write *WDTn_RST.reset* to 0x5A.
10. Execute the WDT enable sequence and enable the WDT:
    a. Disable global interrupts.
    b. Write *WDTn_RST.reset* to 0xFE.
    c. Write *WDTn_RST.reset* to 0xAD.
    d. Set *WDTn_CTRL.en* to 1 to enable the WDT.
    e. Re-enable global interrupts.

11. Verify the peripheral is enabled before proceeding:
    a. Poll *WDTn_CTRL.clkrdy* until it reads 1, or
12. Set *WDTn_CTRL.clkrdy_ie* = 1 to generate a WDT enabled event interrupt.

17.6 Resets

The WDT is a critical safety feature. Most of the fields are reset by a POR or system reset events only; however, the enable field (*WDTn_CTRL.en*) and the interrupt flag fields are not reset by a system reset event.

17.7 Registers

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

Table 17-3: 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

17.7.1 Register Details

Table 17-4: WDT Control Register

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31	rst_late	R/W	0	**Reset Late Event** A watchdog reset event occurred after the time specified in *WDTn_CTRL.rst_late_val*. This flag is set even if *WDTn_CTRL.win_en* = 0 or *WDTn_CTRL.wdt_rst_en* = 0. The software must clear this field to 0. 0: Watchdog did not cause a reset event. 1: Watchdog reset occurred after *WDTn_CTRL.rst_early_val*.
30	rst_early	R/W	0	**Reset Early Event** A watchdog reset event occurred before the time specified in the *WDTn_CTRL.rst_early_val* field. This flag is set even if *WDTn_CTRL.win_en* = 0 or *WDTn_CTRL.wdt_rst_en* = 0. The software must clear this field to 0. 0: Watchdog did not cause a reset event. 1: Watchdog reset occurred before the time specified in the *WDTn_CTRL.rst_early_val* field.
29	win_en	R/W	0	**Window Function Enable** 0: Disabled. The WDT recognizes interrupt late and reset late events, supporting legacy implementations. 1: Enabled.
28	clkrdy	R	0	**Clock Status** This field is cleared to 0 by the hardware when the software changes the state of the *WDTn_CTRL.en* field. The hardware sets this field to 1 when the change to the requested enable or disable is complete. 0: WDT status change in progress. 1: WDT status change complete.

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 software from needing to poll the *WDTn_CTRL.clkrdy* field to determine when the WDT clock is ready. When the *WDTn_CTRL.clkrdy* field transitions from 1 to 0, this interrupt signals the transition is complete. 0: Disabled. 1: Enabled.
26:24	-	RO	0	**Reserved**
23:20	rst_early_val	R/W	0	**Reset Early Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$ *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*
19:16	int_early_val	R/W	0	**Interrupt Early Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$ *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*
15:13	-	RO	0	**Reserved**

WDT Control			WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
12	int_early	R/W	0	**Interrupt Early Flag** A feed sequence is performed earlier than the time determined by the *WDTn_CTRL.int_early* field. This flag is set even if *WDTn_CTRL.win_en* = 0. 0: No interrupt event. 1: Interrupt event occurred. *Note: A WDT interrupt is generated if the WDT interrupt is enabled (WDTn_CTRL.wdt_int_en = 1).*
11	wdt_rst_en	R/W	0	**WDT Reset Enable** 0: Disabled. 1: Enabled.
10	wdt_int_en	R/W	0	**WDT Interrupt Enable** 0: Disabled. 1: Enabled.
9	int_late	R/W	0	**Interrupt Late Flag** A watchdog feed sequence did not occur before the time determined by the *WDTn_CTRL.int_late_val* field. 0: No interrupt event. 1: Interrupt event occurred. *Note: A WDT interrupt is generated if the WDT interrupt is enabled (WDTn_CTRL.wdt_int_en = 1).*
8	en	R/W	0	**WDT Enable** This field enables/disables the WDT clock into the peripheral. *WDTn_CNT.count* holds its value while the WDT is disabled. The WDT disable sequence must be performed immediately before setting this field to 0. The WDT enable sequence must be performed immediately before setting this field to 1. 0: Disabled. 1: Enabled.
7:4	rst_late_val	R/W	0	**Reset Late Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$ *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*

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

Table 17-5: WDT Reset Register

WDT Reset			WDTn_RST	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved**
7:0	reset	R/W	0†	**Reset Watchdog Timer Count** See the *WDT Protection Sequence* section for details on using this field for resetting the counter, enabling and disabling the WDT. †*Note: This field is set to 0 on a POR and is not affected by other resets.*

Table 17-6: 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 17-1* for the available clock options. 0: CLK0. 1: CLK1. 2: CLK2. 3: CLK3. 4: CLK4. 5: CLK5. 6: CLK6. 7: CLK7. †*Note: This field is only reset on a POR and unaffected by other resets.* *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before changing this field.*

Table 17-7: WDT Count Register

WDT Count			WDTn_CNT	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	count	R	0	**WDT Counter** The counter value for debugging. This register is reset by system reset and the watchdog feeding sequence. *Note: The watchdog timer must be disabled (WDTn_CTRL.en = 0) before reading this field.*

18. Pulse Train Engine (PT)

Each independent pulse train engine operates either in square wave mode, which generates a continuous 50% duty-cycle square wave, or pulse train mode, which generates a continuous programmed bit pattern from 2-bits to 32-bits in length. Pulse train engines are used independently or may be synchronized together to generate signals in unison. The frequency of each generated output can be set separately based on a divisor of the peripheral clock.

18.1 Instances

The device provides four instances of the pulse train engine peripheral.

- PT0 to PT3

All peripheral registers share a common register set.

18.2 Features

The pulse train outputs with individually programmable modes, patterns, and output enables. The pulse train engine uses the PCLK, ensuring all pulse train outputs use the same clock source. The pulse trains support the following features:

- Independent or synchronous pulse train output operation
- Atomic enable and atomic disable.
- Synchronous enable or disable of pulse train output(s) without modification to non-intended pulse train outputs.
- Multiple output modes:
  - Square wave output mode generates a repeating square wave (50% duty cycle).
  - Pattern output mode for generating a customizable output wave based on a programmable bit pattern from 2 to 32 output cycles.
- Global clock for all generated outputs
- Individual rate configuration for each pulse train output
- Configuration registers are modifiable while the pulse train engine is running.
- Pulse train outputs can be halted and resumed at the same point.

18.3 Engine

The pulse train engine uses the PCLK as the peripheral input clock. Each pulse train output is individually configurable and independently controlled.

The following sections describe the available configuration options for each individual pulse train output.

18.3.1 Pulse Train Output Modes

Each pulse train output supports the following modes:

- Pulse train mode
- Bit pattern length
- Square wave mode

18.3.1.1 Pulse Train Mode

When pulse train *n* (*PTn*) is configured in pulse train mode, the configuration also includes the bit length (up to 32-bits) of the custom pulse train. This is configured using the 5-bit field *PTn_RATE_LENGTH.mode* as follows:

PTn_RATE_LENGTH.mode = 1:

PTn configured in square wave mode.

PTn_RATE_LENGTH.mode > 1:

PTn is configured in pulse train mode. The value of the *mode* field is the pattern bit length.

PTn_RATE_LENGTH.mode = 0:

PTn configured for pulse train mode (32-bit pattern).

18.3.1.2 In Pulse Train Mode, Set the Bit Pattern

If an output is set to pulse train mode, configure a custom bit pattern from 2- to 32-bits in length in the 32-bit register *PTn_TRAIN*. The pattern is shifted out LSB first. If the output is configured in square wave mode, then the *PTn_TRAIN* register is ignored.

The *PTn_TRAIN* register is shadowed internally and the actual output of the pulse train is dependent upon the internal shadow register. The software must set the*PTn_TRAIN* and other configuration including the loop, delay, rate, and length before setting the enabling the pulse train by setting PTG_ENABLE. Setting PTG_ENABLE will load the PTn_TRAIN content into the shadow registers corresponding to each channel. INT_READY_FLG will set by HW, indicating that the PTn_TRAIN now can be safely replenished with new data without corrupting the ongoing patterns. When the pattern runs out, if there is new pattern in PTn_TRAIN, it will automatically reload. (By new data, it means a new APB write to PTn_TRAIN regardless of content). INT_READY_FLG will assert again informing another opportunity to fill PTn_TRAIN with data. SW must ensure other configurations (loop, delay, rate, length) stays the same during this process. The behavior for changing configurations while loading the new pattern is undetermined.

Equation 18-1: Pulse Train Mode Output Function

$$PTn\_TRAIN = [Bit\ pattern\ for\ PTn]$$

18.3.1.3 Synchronize Two or More Outputs, if Needed

The write-only register *PTG_RESYNC* “PT Global Resync” allows two or more outputs to be reset and synchronized. Write to any bit in *PTG_RESYNC* to simultaneously reset any outputs in pulse train mode to the beginning of the pattern (the LSB) set in the *PTn_TRAIN* bit-pattern register, and reset the output to 0 for outputs in square wave mode.

18.3.1.4 Pulse Train Loop Mode

By default, a pulse train engine runs indefinitely until the software disables it.

A pulse train engine can be configured to repeat its pattern a specified number of times, referred to as loop mode. To select loop mode, write a non-zero value to the 16-bit field *PTn_LOOP.count*. When the pulse train engine is enabled, this field decrements by 1 each time a complete pattern is shifted through the output pin. When the count reaches 0, the output is halted, and the corresponding flag in the *PTG_STOP_INTFL* register is set.

18.3.1.5 Pulse Train Loop Delay

If the pulse train is configured in loop mode, a delay can be inserted after each repeated output pattern. Write the 12-bit field *PTn_LOOP.delay* with the number of PCLK cycles to delay between the MSB of the last pattern to the LSB of the next pattern to enable a delay. During this delay, the output is held at the MSB of the last pattern. If the loop counter has not reached 0, then it is decremented when the next pattern starts.

18.3.1.6 Pulse Train Automatic Restart Mode

When an engine in pulse train mode is in loop mode and stops when the loop count reaches 0, this is called a stop event. A stop event can optionally trigger one or more pulse trains to restart from the beginning. This is called automatic restart mode. While only pulse train engines operating in pulse train mode can operate in loop mode and can optionally restart a pulse train engine, automatic restart mode can trigger pulse train engines operating in pulse train mode or square wave mode.

If another pulse train's stop event triggers a running pulse train engine, automatic restart restarts the running pulse train engine from the beginning of its pattern. If another pulse train's stop event triggers a pulse train engine, and it is not running, automatic restart sets the enable bit to 1 and starts the pulse train engine.

The settings for this mode are contained in the *PTn_RESTART* register for each pulse train engine.

Note: The configuration for automatic restart is set using the pulse train engine(s) triggered by the automatic restart, not the pulse train engine(s) that trigger the automatic restart. For example, the PT2_RESTART register configures which pulse train engine triggers PT2 to restart.

Each pulse train engine can be configured to perform an automatic restart when it detects a stop event from one or two pulse trains.

If *PTn_RESTART.on_pt_x_loop_exit* = 1, then pulse train engine *n* automatically restarts when it detects a stop event from pulse train *x*, where *x* is the value in the 5-bit field *PTn_RESTART.pt_x_select*.

If *PTn_RESTART.on_pt_y_loop_exit* = 1, then pulse train engine *n* automatically restarts when it detects a stop event from pulse train *y*, where *y* is the value in 5-bit field *PTn_RESTART.pt_y_select*.

A pulse train engine can be configured to restart on its stop event, allowing the pulse train to run indefinitely.

Each individual pulse train can be configured for:

- No automatic restart.
- Automatic restart triggered by a stop event from pulse train *x* only.
- Automatic restart triggered by a stop event from pulse train *y* only.
- Automatic restart triggered by a stop event from both pulse train *x* and pulse train *y*

18.4 Enabling and Disabling a Pulse Train Output

The *PTG_ENABLE* register is used to enable each of the individual pulse train outputs. Enable a given pulse train output by setting the respective bit in the *PTG_ENABLE* register. An enabled pulse train output is automatically cleared when the channel runs out of patterns. For a short pulse train pattern, the enable bit may be cleared before software is able to check the bit.

Note: Before changing a pulse train output's configuration, the corresponding pulse train output should be halted to prevent unexpected behavior.

18.5 Atomic Pulse Train Output Enable and Disable

Deterministic enable and disable operations are critical for pulse train outputs that must be synchronized in an application. The *PTG_ENABLE* register does not perform atomic access directly. Atomic operations are supported using the registers *PTG_SAFE_EN*, *PTG_SAFE_DIS*.

For most pulse train peripherals, enabling and disabling individual pulse trains is performed by setting and clearing bits in the global enable/disable register, which for this peripheral is *PTG_ENABLE*. For most Arm Cortex-M microcontrollers, this is usually done by bit banding. Because bit banding performs a read, modify, write (RMW), some pulse trains could start and end during the RMW operation, often with unpredictable results.

Two additional registers are used to enable and disable the outputs to ensure safe and predictable operation.

18.5.1 Pulse Train Atomic Enable

PTG_SAFE_EN “Global Safe Enable” is a write-only register. To safely enable outputs without a read/modify/write, write a 32-bit value to this register with a 1 in the bit positions corresponding to the pulse train engines to be enabled. This immediately sets to 1 the corresponding bits in the *PTG_ENABLE* register to 1, enabling the corresponding pulse train engine. Writing a 0 to any bit position in the *PTG_SAFE_EN* register does not affect the state of the corresponding pulse train enable bit. If the corresponding pulse train engine is already enabled and running, writing a 1 to that bit position in the *PTG_SAFE_EN* register has no effect.

18.5.2 Pulse Train Atomic Disable

PTG_SAFE_DIS “Global Safe Disable” is a write-only register for disabling a pulse train engine without performing a read/modify/write. To safely disable pulse train engines, write a 32-bit value to this register with a 1 in the bit positions corresponding to the pulse train engines to be disabled. This immediately clears to 0 the corresponding bits in *PTG_ENABLE*, which disables the corresponding pulse train engines. Writing a 0 to any bit position in the *PTG_SAFE_DIS* register does not affect the state of the corresponding pulse train enable bit.

Bit banding is not supported for the *PTG_ENABLE*, *PTG_SAFE_EN*, and *PTG_SAFE_DIS* registers and can have unpredictable results.

18.6 Halt and Disable

Once a pulse train engine is enabled and running, it continues to run until one of the following events stops the output:

- The corresponding enable bit in the *PTG_ENABLE* register is cleared to 0 to halt the output.
- A 1 is written to the corresponding disable bit in the *PTG_SAFE_DIS* register to halt the output.
- The corresponding resync bit in the *PTG_RESYNC* register is cleared to 0 to halt and reset the output.
- *PTn_LOOP* was initialized to a non-zero value, and the loop count has reached 0 (this does not affect square wave mode; it only applies to pulse train mode).

When a pulse train is halted, the corresponding enable bit in *PTG_ENABLE* is automatically cleared to 0.

18.7 Interrupts

Each pulse train can generate an interrupt only if it is configured in pulse train mode, and the loop counter *PTn_LOOP* is initialized to a non-zero number. When *PTn_LOOP* counts down to 0, the corresponding status flag in the *PTG_STOP_INTFL* register is set. If the corresponding interrupt enable bit in the *PTG_STOP_INTEN* register is set, the event also generates an interrupt.

18.8 Registers

See *Table 3-2* for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own independent set of the registers shown in *Table 18-1*. Register names for a specific instance are defined by replacing “n” with the instance number. As an example, a register PERIPHERALn_CTRL resolves to PERIPHERAL0_CTRL and PERIPHERAL1_CTRL for instances 0 and 1, respectively.

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

Table 18-1: Pulse Train Engine Register Summary

Offset	Register	Description
[0x0000]	*PTG_ENABLE*	PT Global Enable/Disable Control
[0x0004]	*PTG_RESYNC*	PT Global Resync
[0x0008]	*PTG_STOP_INTFL*	PT Stopped Global Status Flags

Offset	Register	Description
[0x000C]	*PTG_STOP_INTEN*	PT Global Interrupt Enable
[0x0010]	*PTG_SAFE_EN*	PT Global Safe Enable
[0x0014]	*PTG_SAFE_DIS*	PT Global Safe Disable
[0x0018]	*PTG_READY_INTFL*	PT Global Ready Interrupt Flag
[0x001C]	*PTG_READY_INTEN*	PT Global Ready Interrupt Enable
[0x0020]	*PTn_RATE_LENGTH*	PTn Configuration
[0x0024]	*PTn_TRAIN*	PTn Pulse Train Mode Bit Pattern
[0x0028]	*PTn_LOOP*	PTn Loop Control
[0x002C]	*PTn_RESTART*	PTn Automatic Restart
[0x0040]	*PTn_RATE_LENGTH*	PTn Configuration
[0x0044]	*PTn_TRAIN*	PT1 Pulse Train Mode Bit Pattern
[0x0048]	*PTn_LOOP*	PT1 Loop Control
[0x004C]	*PTn_RESTART*	PT1 Automatic Restart
[0x0050]	*PTn_RATE_LENGTH*	PT2 Configuration
[0x0054]	*PTn_TRAIN*	PT2 Pulse Train Mode Bit Pattern
[0x0058]	*PTn_LOOP*	PT2 Loop Control
[0x005C]	*PTn_RESTART*	PT2 Automatic Restart
[0x0060]	*PTn_RATE_LENGTH*	PT3 Configuration
[0x0064]	*PTn_TRAIN*	PT3 Pulse Train Mode Bit Pattern
[0x0068]	*PTn_LOOP*	PT3 Loop Control
[0x006C]	*PTn_RESTART*	PT3 Automatic Restart

18.8.1 Register Details

18.8.1.1 Pulse Train Engine Global Enable/Disable Register

This register enables each of the individual pulse trains. Write a 1 to the individual pulse train enable bits to enable the corresponding pulse train. When, for a given pulse train, the *PTn_LOOP.count* loop counter is set to a non-zero number, when the loop counter reaches zero, then the given pulse train engine stops, and the corresponding enable bit in this register is cleared by hardware.

Table 18-2: Pulse Train Engine Global Enable/Disable Register

PT Global Enable/Disable Control			PTG_ENABLE	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	R/W	0	**Enable PT3** 0: Disabled 1: Enabled *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
2	pt2	R/W	0	**Enable PT2** 0: Disabled 1: Enabled *Note: Disabling an active pulse train halts the output and does not generate a stop event.*

PT Global Enable/Disable Control			PTG_ENABLE	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
1	pt1	R/W	0	**Enable PT1** 0: Disabled 1: Enabled *Note: Disabling an active pulse train halts the output and does not generate a stop event.*
0	pt0	R/W	0	**Enable PT0** 0: Disabled 1: Enabled *Note: Disabling an active pulse train halts the output and does not generate a stop event.*

Table 18-3: Pulse Train Engine Resync Register

PT Resync			PTG_RESYNC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	WO	-	**Resync Control for PT3** Write 1 to reset the output of the pulse train. For pulse train mode, the output is restarted to the beginning of the output pattern. For square wave mode, the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
2	pt2	WO	-	**Resync Control for PT2** Write 1 to reset the output of the pulse train. For pulse train mode, the output is restarted to the beginning of the output pattern. For square wave mode, the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
1	pt1	WO	-	**Resync Control for PT1** Write 1 to reset the output of the pulse train. For pulse train mode, the output is restarted to the beginning of the output pattern. For square wave mode, the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*
0	pt0	WO	-	**Resync Control for PT0** Write 1 to reset the output of the pulse train. For pulse train mode, the output is restarted to the beginning of the output pattern. For square wave mode, the output is reset to 0. Setting multiple bits simultaneously in this register synchronizes the set outputs. 1: Reset/restart the pulse train 0: No effect *Note: Writing 1 has no effect if the corresponding pulse train is disabled.*

Table 18-4: Pulse Train Engine Stop Interrupt Flag Register

PT Stop Interrupt Flag			PTG_STOP_INTFL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	R/W1C	0	**PT3 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in pulse train mode and the loop counter reaches 0. In square wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
2	pt2	R/W1C	0	**PT2 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in pulse train mode and the loop counter reaches 0. In square wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
1	pt1	R/W1C	0	**PT1 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in pulse train mode and the loop counter reaches 0. In square wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.
0	pt0	R/W1C	0	**PT0 Stopped Status Flag** This bit is set to 1 by hardware when the corresponding pulse train is in pulse train mode and the loop counter reaches 0. In square wave mode, this field is not used. Write 1 to clear. 1: Pulse Train is stopped.

Table 18-5: Pulse Train Engine Interrupt Enable Register

PT Stop Interrupt Enable			PTG_STOP_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	R/W	0	**PT3 Stop Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.
2	pt2	R/W	0	**PT2 Stop Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.
1	pt1	R/W	0	**PT1 Stop Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.
0	pt0	R/W	0	**PT0 Stop Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.

18.8.1.2 Pulse Train Engine Safe Enable Register

A 32-bit value written to this register performs an immediate binary OR with the contents of *PTG_ENABLE*. The result is immediately stored in the *PTG_ENABLE*.

Table 18-6: Pulse Train Engine Safe Enable Register

Pulse Train Engine Safe Enable			**PTG_SAFE_EN**	**[0x0010]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	WO	-	**Safe Enable Control for PT3** Writing a 1 sets *PTG_ENABLE.pt3*. 1: Enable corresponding pulse train. 0: No effect.
2	pt2	WO	-	**Safe Enable Control for PT2** Writing a 1 sets *PTG_ENABLE.pt2*. 1: Enable corresponding pulse train. 0: No effect.
1	pt1	WO	-	**Safe Enable Control for PT1** Writing a 1 sets *PTG_ENABLE.pt1*. 1: Enable corresponding pulse train 0: No effect
0	pt0	WO	-	**Safe Enable Control for PT0** Writing a 1 sets *PTG_ENABLE.pt0*. 1: Enable corresponding pulse train. 0: No effect.

18.8.1.3 Pulse Train Engine Safe Disable Register

A 32-bit value written to this register performs an immediate binary OR with the contents of *PTG_ENABLE*. The result is immediately stored in the *PTG_ENABLE*.

Table 18-7: Pulse Train Engine Safe Disable Register

Pulse Train Engine Safe Disable			**PTG_SAFE_DIS**	**[0x0014]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	WO	-	**Safe Disable Control for PT3** Writing a 1 clears *PTG_ENABLE.pt3*. 1: Disable corresponding pulse train. 0: No effect.
2	pt2	WO	-	**Safe Disable Control for PT2** Writing a 1 clears *PTG_ENABLE. pt2*. 1: Disable corresponding pulse train. 0: No effect.
1	pt1	WO	-	**Safe Disable Control for PT1** Writing a 1 clears *PTG_ENABLE.pt1*. 1: Disable corresponding pulse train. 0: No effect.

Pulse Train Engine Safe Disable			PTG_SAFE_DIS	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
0	pt0	WO	-	**Safe Disable Control for PT0** Writing a 1 clears *PTG_ENABLE.pt0*. 1: Disable corresponding pulse train. 0: No effect.

Table 18-8: Pulse Train Engine Ready Interrupt Flag Register

Pulse Train Engine Ready Interrupt Flag			PTG_READY_INTFL	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	R/W	0	**PT3 Ready Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.
2	pt2	R/W	0	**PT2 Ready Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.
1	pt1	R/W	0	**PT1 Ready Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.
0	pt0	R/W	0	**PT0 Ready Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.

Table 18-9: Pulse Train Engine Ready Interrupt Enable Register

Pulse Train Ready Interrupt Enable			PTG_READY_INTEN	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved**
3	pt3	R/W	0	**PT3 Ready Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.
2	pt2	R/W	0	**PT2 Ready Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.

Pulse Train Ready Interrupt Enable			PTG_READY_INTEN	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
1	pt1	R/W	0	**PT1 Ready Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.
0	pt0	R/W	0	**PT0 Ready Interrupt Enable** Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_STOP*_INTFL register. 0: Disabled. 1: Enabled.

18.8.1.4 Pulse Train Registers

Table 18-10: Pulse Train Engine Configuration Register

Pulse Train *n* Configuration			PTn_RATE_LENGTH	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:27	mode	R/W	1	**Square Wave or Pulse Train Output Mode** This field selects either pulse train mode with length or square wave mode. 0: Pulse train mode, 32-bits long. 1: Square wave mode. 2: Pulse train mode, 2-bits long. 3: Pulse train mode, 3-bits long. …: … 31: Pulse train mode, 31-bits long. *Note: If this field is set to 1 square wave mode, the PTn_TRAIN register is not used.*
26:0	rate_control	R/W	0	**Pulse Train Enable and Rate Control** Defines the rate at which the output for PT*n* changes state by setting the divisor of the PT clock. Setting this field to 0 disables the PT*n*. For all other values, the following equation is used to calculate the rate.: $f_{PTn} = \frac{f_{PTE\_CLK}}{rate\_control}$ 0: Output halted. 1: $f_{PTn} = f_{PTE\_CLK}$ 2: $f_{PTn} = {f_{PTE\_CLK}}/{2}$ 3: $f_{PTn} = {f_{PTE\_CLK}}/{3}$ …

Table 18-11: Pulse Train Mode Bit Pattern Register

Pulse Train Mode Bit Pattern			PTn_TRAIN	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**Pulse Train Mode Bit Pattern** Write the repeating bit pattern that is shifted out, LSB first, when configured in pulse train mode. Set the bit pattern length with the *PTn_RATE_LENGTH.mode* field. *Note: This register is ignored in square wave mode.* *Note: 0x0000 0000 and 0x0001 0000 are invalid values for this register.*

Table 18-12: Pulse Train n Loop Configuration Register

Pulse Train Loop Configuration			PTn_LOOP	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:28	-	RO	0	**Reserved**
27:16	delay	R/W	0	**Pulse Train Delay Between Loops** Sets the delay in the number of PCLK cycles, that the output pauses between loops. The *PTn_LOOP.count* is decremented after the delay. *Note: This field is ignored if software writes 0 to the PTn_LOOP.count field.*
15:0	count	R/W	0	**Pulse Train Loop Countdown** Sets the number of times a pulse train pattern is repeated until it automatically stops. Reading this field returns the number of loops remaining. When this field counts down to zero, the corresponding *PTG_STOP*_INTFL flag is set. Writing this field to 0 to repeat the pulse train pattern indefinitely. *Note: This field is ignored in square wave mode.*

Table 18-13: Pulse Train n Automatic Restart Configuration Register

Pulse Train Automatic Restart Configuration			PTn_RESTART	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	on_pt_y_loop_exit	R/W	0	**Enable Automatic Restart for This Pulse Train on PTy Stop Event** 0: Disable automatic restart. 1: When PTy has a stop event, automatically restart this pulse train from the beginning of its pattern.
14:11	-	RO	0	**Reserved**
12:8	pt_y_select	R/W	0	**Select PTy** Select the pulse train number to be associated with PTy. This engine must be in pulse train mode. 0: PT0. 1: PT1. 2: PT2. 3: PT3. 4 - 31: Reserved.
7	on_pt_x_loop_exit	R/W	0	**Enable Automatic Restart for this Pulse Train on a *PTn* Stop Event** 0: Disable automatic restart. 1: When *PTn* has a stop event, automatically restart the pulse train from the beginning of its pattern.
6:5	-	RO	0	**Reserved**

Pulse Train Automatic Restart Configuration			PTn_RESTART	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
4:0	pt_x_select	R/W	0	**Select *PTn*** Select the pulse train number to be associated with *PTn*. This engine must be in pulse train mode. 0: PT0. 1: PT1. 2: PT2. 3: PT3. 4 - 31: Reserved.

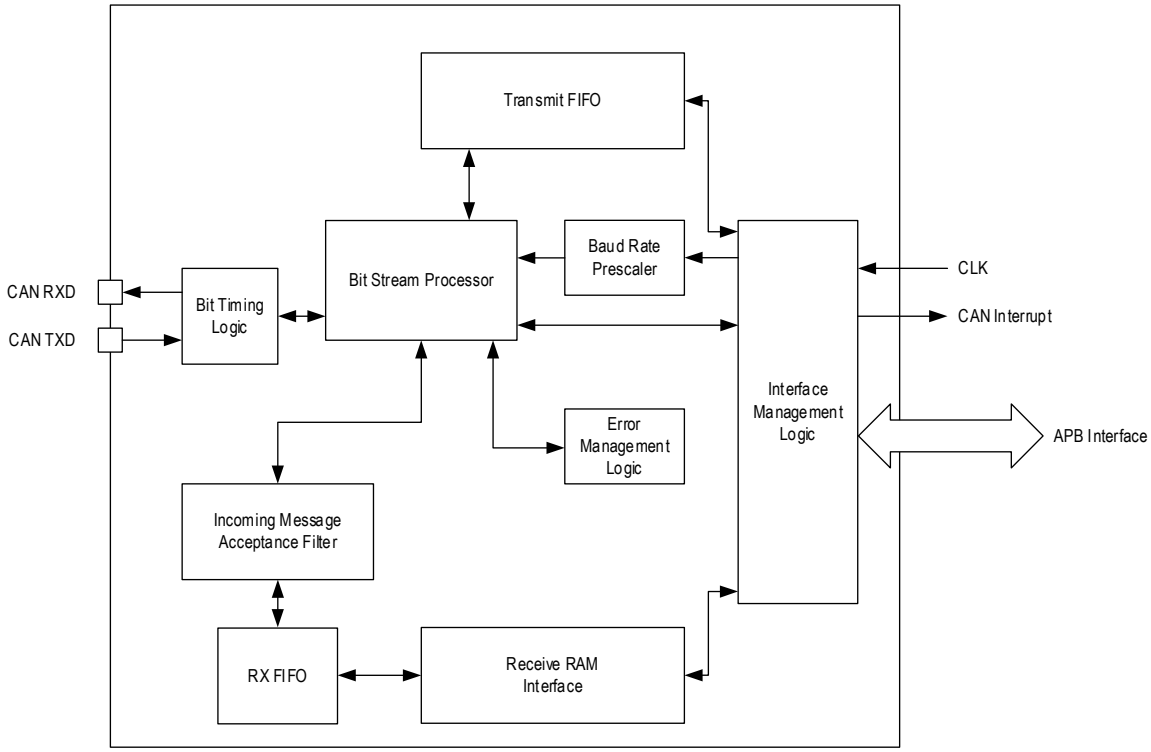
19. Controller Area Network (CAN)

Key features:

- Supports CAN 2.0b frames.
- Selectable ID type:
  - 11-bit standard ID.
  - 11-bit Standard ID + 18-bit extended ID.
- Selectable frame type:
  - Data frame (remote transmission request (RTR) = 0).
  - Remote frame (RTR = 1) for CAN frames.
- Flexible data transfer rate:
  - Up to 1Mbps.
- Hardware message filtering (dual/single filters).
- DMA support for transmit and receive.
- 128-byte transmit FIFO and 256-byte receive FIFO.
- Overload frame is generated on a FIFO overflow.
- Protocol Exception Event detection.
- Normal and Listen Only modes supported.
- Transmitter Delay Compensation up to three data bits long.
- Single Shot transmission.
- Readable error counters.
- Last error code.
- Sleep mode and wake up unit.

Figure 19-1 shows the block diagram of the peripheral. See *Table 19-1* for the peripheral-specific bus assignment and bit rate generator clock source.

Figure 19-1: CAN Block Diagram



19.1 Instances

Table 19-1: MAX32662 CAN Instances

Instance	CAN Clock Options
CAN	PCLK

19.2 Bit Timing

The bit rate (f_{BIT}) of a system characterizes the amount of data bits transmitted per time unit. *Equation 19-1* shows the bit rate of a CAN system. *Figure 19-2* shows the nominal bit time (t_{BIT}). The nominal bit time consists of three non-overlapping segments SYNCSEG, TSEG1, and TSEG2 with the corresponding time durations $t_{SYNCSEG}$, t_{TSEG1}, and t_{SEG2}. The time duration of the nominal bit time is the sum of the segment durations as shown in *Equation 19-2*. Each of these segments is specified as an integer number of the basic unit of time in a bit period, referred to as the time quantum (t_q). The time quantum is one period of the CAN system clock, t_{SCL}, which is derived from selected peripheral clock, t_{CLK}.

The CAN system clock in standard CAN mode can be adjusted by the application software through a programable prescaler selected with the field *CAN_BUSTIM0.br_clkdiv*, as shown in *Equation 19-3*.

The TSEG1 and TSEG2 segments are controlled using the *CAN_BUSTIM1.tseg1* and *CAN_BUSTIM1.tseg2* fields. TSEG1 is adjustable from a minimum of 1 t_q to 16 t_q. TSEG2 is adjustable from 1 t_q to 8 t_q.

Another time segment of importance to the CAN bit timing calculation is the synchronization jump width (SJW). The SJW with time duration t_{SJW} is not a segment of the bit period but defines the maximum number of time quanta by which a bit period can be lengthened or shortened in the event of resynchronization. The SJW is controlled using the *CAN_BUSTIM0.sjw* field and can be set from 1 t_q to 4 t_q.

Equation 19-1: Bit Rate

$$f_{BIT} = \frac{1}{t_{BIT}}$$

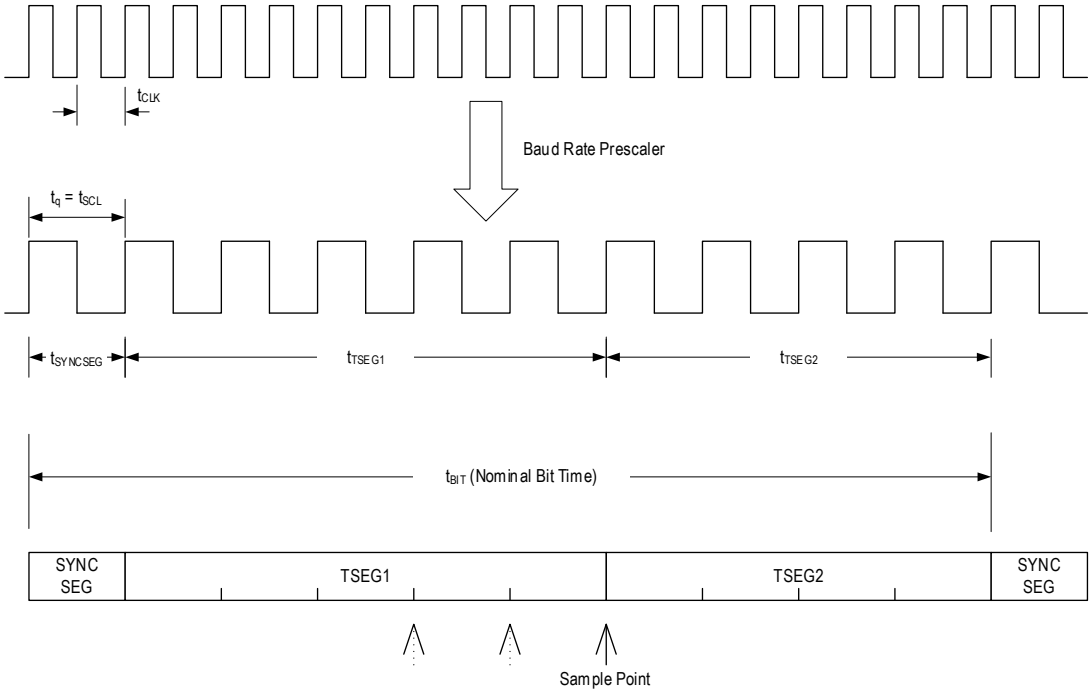
Equation 19-2: Nominal Bit Time Duration

$$t_{BIT} = t_{SYNCSEG} + t_{TSEG1} + t_{TSEG2}$$

Equation 19-3: CAN System Clock Selection for Standard CAN

$$t_{SCL} = 2 \times t_{CLK} \times CAN\_BUSTIM0.br\_clkdiv$$

Figure 19-2: CAN Clock Configuration and Nominal Bit Time



19.3 Operating Modes

The CAN controller supports multiple operating modes and test modes. After a POR, system reset, or peripheral reset, the CAN controller is in the reset state. Reset state is used for configuration of the CAN controller by software.

19.3.1 Reset Mode

When the CAN controller is in reset mode, frame reception and transmission are disabled. Configuration of the CAN controller is performed while in reset mode. After configuration, the CAN controller can be placed into normal, sleep, or one of the supported test modes. The CAN controller is in reset mode when the *CAN_MODE.rst* field is set to 1.

19.3.2 Normal Mode

In normal mode, the CAN controller performs frame reception and transmission. Enter normal mode from reset mode by setting the *CAN_MODE.lom* to 0 and the *CAN_MODE.rst* field to 0 simultaneously.

19.3.3 Listen-Only Mode

Listen-only mode can be used to analyze traffic on the CAN bus without affecting the bus. In listen-only mode, the CAN controller gives no acknowledgement for any message whether received successfully or for an error. The error counters are stopped at their current values. Listen-only mode can also be used for automatic bit rate detection without disturbing traffic on the network.

Enter listen-only mode by performing the following steps:

1. Place the CAN controller into the reset state by setting the *CAN_MODE.rst* field to 1.
2. Perform any other CAN controller configuration required.
3. Set the *CAN_MODE* register to 2 (*CAN_MODE.rst* = 0 and *CAN_MODE.lom* = 1) to enter listen-only mode.

Note: The CAN_MODE.rst and CAN_MODE.lom fields must be written simultaneously to enter listen-only mode.

19.3.4 Test Mode

Two test modes are supported by the CAN controller, loop back and loop back with the transmit pin disconnected.

19.3.4.1 Loop Back Mode

The CAN controller can be put into loop back mode by setting the *CAN_TEST.lben* field to 1. This field can only be set to 1 while the CAN controller is in the reset state (*CAN_MODE.rst* = 1). Once loop back mode is enabled, set *CAN_MODE.rst* to 0 to enter normal mode.

19.3.4.2 Loop Back Mode with Transmit Disconnected

The CAN controller supports loop back mode with the transmit pin disconnected. In this mode, all messages the device transmits are looped back to the receive interface and the CAN TX pin is left in a recessive state. Enter loop back mode with the transmit pin disconnected by performing the following steps:

1. Place the CAN controller into the reset state by setting *CAN_MODE.rst* to 1.
2. Set the *CAN_TEST* register to 3 (*CAN_TEST.txc* = 1 and *CAN_TEST.lben* = 1) to enable both loop back mode and disconnect the transmit pin.
3. Perform any other controller configuration and then enter normal mode by setting *CAN_MODE.rst* to 0.

19.4 Arbitration Lost Capture

The CAN controller is able to identify the exact CAN bit stream position where the arbitration is lost. When arbitration is lost, an arbitration lost interrupt occurs (*CAN_STAT.al* = 1). The bit position where the arbitration is lost is stored in the *CAN_ALC.alc* field. Software must clear the CAN_INTFL.al interrupt to initiate the arbitration lost capture function for the next CAN communication. *Table 19-2* shows the mapping of the *CAN_ALC.alc* field to the identifier of where the arbitration is lost. For example, when arbitration is lost at identifier bit 20 in an extended frame, the *CAN_ALC.alc* field contains the value 8.

Table 19-2: CAN_ALC.alc Field Mapped to Arbitration Lost Identifier

***CAN_ALC.alc* bits**					**Decimal Value**	**Description**
***alc*[4]**	***alc*[3]**	***alc*[2]**	***alc*[1]**	***alc*[0]**		
0	0	0	0	0	0	Arbitration lost in ID28 / 10
0	0	0	0	1	1	Arbitration lost in ID27 / 9
0	0	0	1	0	2	Arbitration lost in ID26 / 8
0	0	0	1	1	3	Arbitration lost in ID25 / 7
0	0	1	0	0	4	Arbitration lost in ID24 / 6
0	0	1	0	1	5	Arbitration lost in ID23 / 5

CAN_ALC.alc bits					Decimal Value	Description
alc[4]	alc[3]	alc[2]	alc[1]	alc[0]		
0	0	1	1	0	6	Arbitration lost in ID22 / 4
0	0	1	1	1	7	Arbitration lost in ID21 / 3
0	1	0	0	0	8	Arbitration lost in ID20 / 2
0	1	0	0	1	9	Arbitration lost in ID19 / 1
0	1	0	1	0	10	Arbitration lost in ID18 / 0
0	1	0	1	1	11	Arbitration lost in SRTR / RTR
0	1	1	0	0	12	Arbitration lost in IDE bit
0	1	1	0	1	13	Arbitration lost in ID17[1]
0	1	1	1	0	14	Arbitration lost in ID16[1]
0	1	1	1	1	15	Arbitration lost in ID15[1]
1	0	0	0	0	16	Arbitration lost in ID14[1]
1	0	0	0	1	17	Arbitration lost in ID13[1]
1	0	0	1	0	18	Arbitration lost in ID12[1]
1	0	0	1	1	19	Arbitration lost in ID11[1]
1	0	1	0	0	20	Arbitration lost in ID10[1]
1	0	1	0	1	21	Arbitration lost in ID9[1]
1	0	1	1	0	22	Arbitration lost in ID8[1]
1	0	1	1	1	23	Arbitration lost in ID7[1]
1	1	0	0	0	24	Arbitration lost in ID6[1]
1	1	0	0	1	25	Arbitration lost in ID5[1]
1	1	0	1	0	26	Arbitration lost in ID4[1]
1	1	0	1	1	27	Arbitration lost in ID3[1]
1	1	1	0	0	28	Arbitration lost in ID2[1]
1	1	1	0	1	29	Arbitration lost in ID1[1]
1	1	1	1	0	30	Arbitration lost in ID0[1]
1	1	1	1	1	31	Arbitration lost in RTR
1: Extended frame messages only.						

19.5 Acceptance Codes and Filtering

Acceptance code filter enables the CAN controller to only receive messages when the identifier bits of the incoming messages are equal to the predefined bits within the acceptance filter registers. The acceptance filters are defined using the acceptance code registers () and the acceptance mask registers (). The acceptance code registers define the bit patterns of the messages to be received, and the corresponding acceptance mask registers are used to enable or disable each of the bits in the pattern. The CAN controller supports either a single acceptance filter of 4 bytes or dual acceptance filters of 2 bytes.

19.5.1 Single Acceptance Filter

In single acceptance filter mode (*CAN_MODE.afm* = 1), a single 4 byte filter is used based on the CAN acceptance filter registers (*CAN_ACR8[0]*:*CAN_ACR8[3]*) and the CAN acceptance mask filter registers (*CAN_AMR8[0]*:*CAN_AMR8[3]*). *Figure 19-3* shows how the single acceptance filter applies to standard frame message and *Figure 19-4* shows how the single acceptance filter applies to extended frame format message. If a standard frame format message is received, the complete identifier including the RTR bit and the first two data bytes (if received) are used for acceptance filtering. Messages are also

accepted if there is no data byte as long as the identifier matches and messages with one data byte are accepted if they match up to the first data byte. If an extended frame format message is received in single filter mode, the complete identifier including the RTR bit is used for acceptance filtering. All enabled single bit comparisons must match to signal acceptance for both standard frame format and extended frame format messages.

Configure the filter by setting the desired bits in the acceptance filter registers (*CAN_ACR8[0]*:*CAN_ACR8[3]*). Enable each desired bit using the corresponding acceptance mask register bits (*CAN_AMR8[0]*:*CAN_AMR8[3]*). Care must be taken when the filter registers are accessed in 16-bit or 32-bit wide accesses to ensure the mapping of the bits are correct. See *Figure 19-3* and *Figure 19-4* for the mapping of the 8-bit registers to the CAN identifier and data bytes.

Figure 19-3: Single Acceptance Filter for Standard Frame Message

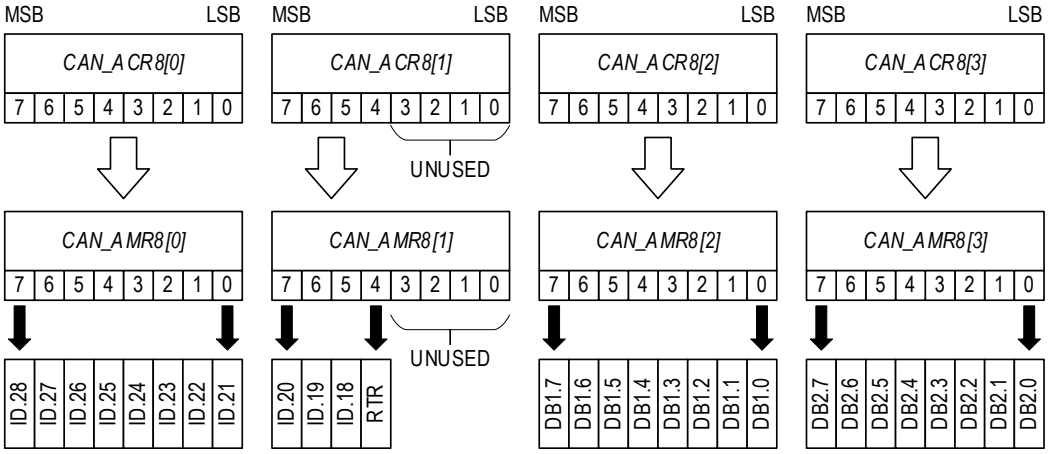
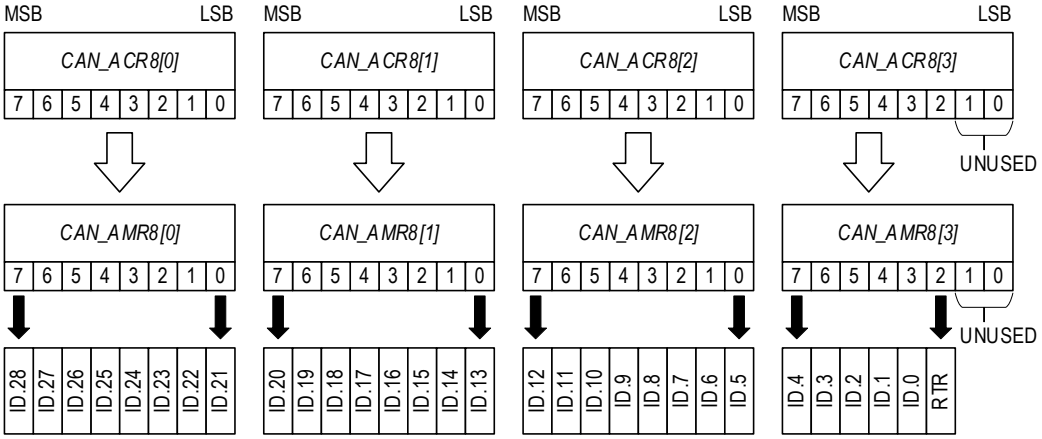


Figure 19-4: Single Acceptance Filter for Extended Frame Messages



19.5.2 Dual Acceptance Filter

In dual acceptance filter mode (*CAN_MODE.afm* = 0), two short filters can be defined. Any received message is compared with both filters to determine if the message should be received. If one or both of the message filters signal acceptance, the message is received.

For a standard frame message, the first filter compares the complete standard identifier including the RTR bit and the first data byte of the message. The second filter compares only the complete standard identifier including the RTR bit. See *Figure 19-5* for the acceptance code register mapping to the identifier and data bits received for a standard frame message. For successful reception, all single bit comparisons of at least one filter must match.

If an extended frame message is received in dual filter mode, both filters compare the first two bytes of the extended identifier range only. For successful reception, all single bit comparisons of at least one filter must match.

Figure 19-5: Dual Filter for Standard Frame Messages

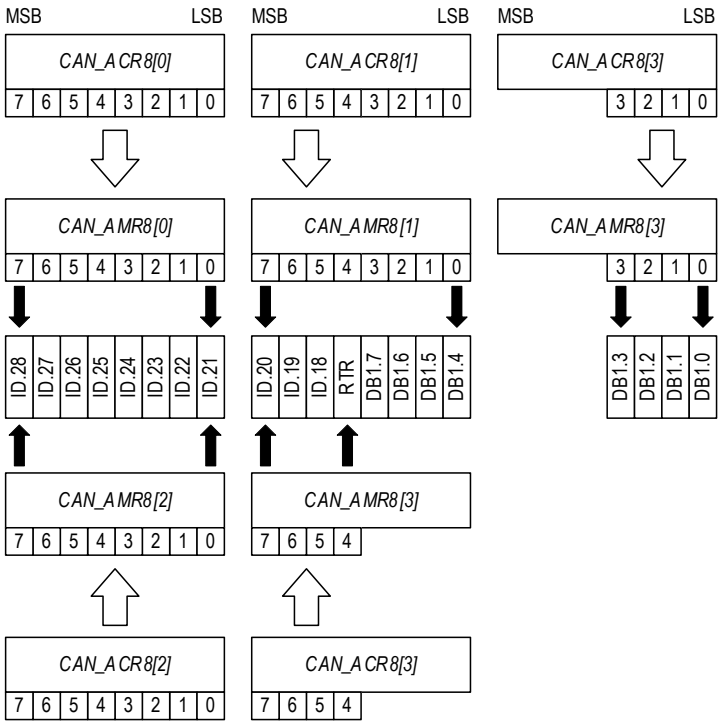
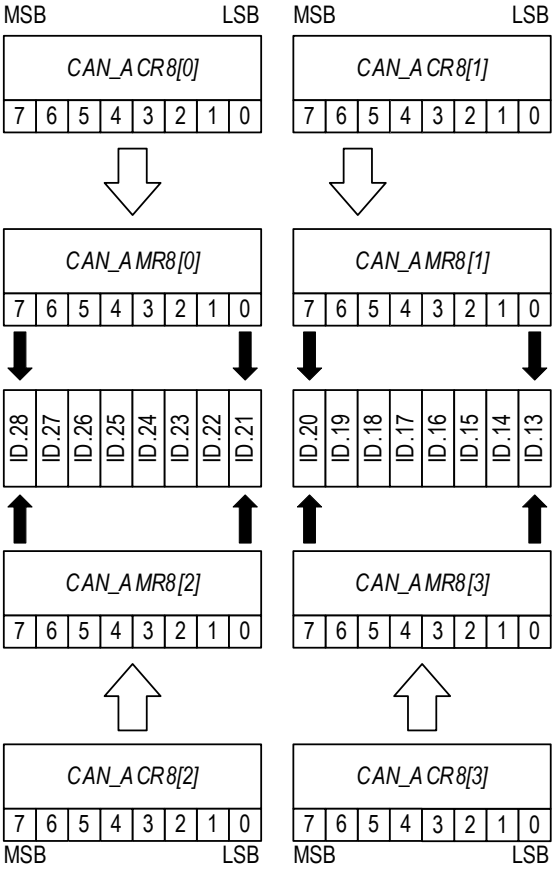


Figure 19-6: Dual Acceptance Filter for Extended Frame Messages



19.6 FIFO Interface

The CAN peripheral supports a 128 byte transmit FIFO and a 256 byte receive FIFO. The FIFOs are accessible in 8-bit, 16-bit, and 32-bit widths. Write to the transmit FIFO using the *CAN_TXFIFO32*, CAN_TXFIFO16[1]:CAN_TXFIFO16[0], and the *CAN_TXFIFO8[3]*:*CAN_TXFIFO8[0]* registers. Read from the receive FIFO in 32-bit, 16-bit, and 8-bit width by reading from the corresponding *CAN_RXFIFO32*, *CAN_RXFIFO16[1]*:*CAN_RXFIFO16[0]*, and CAN_RXFIFO8[3]:CAN_RXFIFO8[0] registers.

Table 19-3 shows the maximum number of identical standard and extended messages that can be stored in the receive FIFO based on the data length code (DLC).

Table 19-3: Maximum Number of Identical Messages in the Receive FIFO

DLC	Number of Standard Messages	Number of Extended Messages
0	64	32
1	64	32
2	32	32
3	32	32
4	32	21
5	32	21
6	21	21
7	21	21

DLC	Number of Standard Messages	Number of Extended Messages
8	21	16
9	16	12
10	12	10
11	10	9
12	9	8
13	7	6
14	4	4
15	3	3

19.6.1 Memory Buffer FIFO Layout for Base Frames

Messages to send and receive are stored in the transmit and receive FIFO. The transmit FIFO can hold one message a time. The receive FIFO buffer can hold many messages at a time. *Table 19-4* shows the transmit and receive FIFO buffer layout for Standard Frame messages.

Table 19-4: Transmit and Receive FIFO Buffer Layout for Standard Frames

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	DATA 1 (DLC<1 then free)								ID[2:0]			Reserved	ESI	Reserved			ID[10:3]								IDE	RRS/RTR	FDF	BRS	DLC			
0x01	DATA 5 (DLC<5 then free)								DATA 4 (DLC<4 then free)								DATA 3 (DLC<3 then free)								DATA 2 (DLC<2 then free)							
0x02	DATA 9 (DLC<9 then free)								DATA 8 (DLC<8 then free)								DATA 7 (DLC<7 then free)								DATA 6 (DLC<6 then free)							
0x03	DATA 13 (DLC<10 then free)								DATA 12 (DLC<9 then free)								DATA 11 (DLC<9 then free)								DATA 10 (DLC<9 then free)							
0x04	DATA 17 (DLC<11 then free)								DATA 16 (DLC<10 then free)								DATA 15 (DLC<10 then free)								DATA 14 (DLC<10 then free)							
0x05	DATA 21 (DLC<11 then free)								DATA 20 (DLC<11 then free)								DATA 19 (DLC<11 then free)								DATA 18 (DLC<11 then free)							
0x06	DATA 25 (DLC<12 then free)								DATA 24 (DLC<11 then free)								DATA 23 (DLC<11 then free)								DATA 22 (DLC<11 then free)							
…																																
0x0F	DATA 61 (DLC<15 then free)								DATA 60 (DLC<15 then free)								DATA 59 (DLC<15 then free)								DATA 58 (DLC<15 then free)							
0x10	free								DATA 64 (DLC<15 then free)								DATA 63 (DLC<15 then free)								DATA 62 (DLC<15 then free)							

Table 19-5: Standard Frame Fields

Field	Description
IDE	**Identifier Extension Flag** This bit specifies the type of frame and is valid in classic CAN and CAN FD frames. 0: Standard Message Identifier. 1: Extended Message Identifier.
RTR/RRS	**Remote Transmission Request** This bit specifies the frame is standard CAN data frame or standard CAN remote frame. For CAN FD format this bit indicates bus state at RRS position of the frame. 0: Standard CAN data frame. 1: Standard CAN remote frame.
FDF	**FD Frame Format Indicator** This field indicates a CAN FD frame. 0: Classic CAN frame format. 1: CAN FD frame format.

Field	Description
BRS	**Baud Rate Switch Bit** 0: Frame without baud rate switching. 1: Frame with baud rate switching. *Note: This field is only available with CAN FD mode.*
ESI	**Error State Indicator** 0: Transmitter of this message is Active Error node. 1: Transmitter of this message is Passive Error node. *Note: This field is only available with CAN FD mode.*
DLC[3:0]	**Data Length Code** Frame data length code field of classic CAN and CAN FD frame.
ID[10:0]	**Standard Message Identifier**
DATA N	**Data Bytes**

19.6.1.1 Message Identifier

The identifier consists of 11 bits (ID[10] to ID[0]). ID[10] is the most significant bit, which is transmitted first on the bus during the arbitration process. The identifier acts as the message's name. It is used in a receiver for acceptance filtering and also determining the bus access priority during the arbitration process. The lower binary value of the identifier means higher priority for the message. This is due to a larger number of leading dominant bits during arbitration.

19.6.1.2 Remote Transmission Request (RTR/RRS)

If this bit is set, a remote frame is transmitted on the CAN bus. This means that no data bytes are included within this frame. Nevertheless, it is necessary to specify the correct data length code, which depends on the corresponding data frame with the same identifier coding. If the RTR bit is not set, a data frame is sent including the number of data bytes as specified by the data length code. In CAN FD mode remote frames do not exist.

19.6.1.3 FD Frame (FDF)

This bit distinguishes between a classic CAN frame and a CAN FD frame. When this bit is set in the transmit data buffer in FD mode (*CAN_FDCTRL.fden* = 1), a CAN FD frame is sent. If FD mode is disabled (*CAN_FDCTRL.fden* = 0), this field is ignored and a classic can frame is sent. If the FDF bit is set to 1 in received data, this indicates the received frame was sent in FD mode, when the CAN peripheral is in FD mode (*CAN_FDCTRL.fden* = 1). If the CAN peripheral is set to classic can (*CAN_FDCTRL.fden* = 0), this field is always set to 0 by the receive hardware.

19.6.1.4 Bit Rate Switch (BRS)

In transmit data this bit determines that the FD frame is sent with bit rate switching in the data phase. In receive mode, the data received where this bit is set means that the received frame included bit time switching if the CAN peripheral is in FD mode (*CAN_FDCTRL.fden* = 1).

19.6.1.5 Error State Indicator (ESI)

This field is only valid when the CAN peripheral is in FD mode (*CAN_FDCTRL.fden* = 1) and the received/transmitted frame is an FD frame. This bit indicates that the transmitter of the message is in passive error node if 1 or active error node if 0.

19.6.1.6 Data Length Code (DLC)

The number of bytes in the data field of a message is coded by the data length code. At the start of a remote frame transmission, the data length code is not considered, due to the RTR bit being a logic 1 (remote). It denotes the number of transmitted/received data bytes to be logic 0. Nevertheless, the data length code must be specified correctly to avoid bus errors, if two CAN controllers start a remote frame transmission with the same identifier simultaneously. The range of the data byte count is 0 to 64 bytes and is coded, as shown in *Table 19-6*.

Table 19-6: DLC Coding to Data Byte Count

Number of Bytes	DLC[3]	DLC[2]	DLC[1]	DLC[0]
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
12	1	0	0	1
16	1	0	1	0
20	1	0	1	1
24	1	1	0	0
32	1	1	0	1
48	1	1	1	0
64	1	1	1	1

In classic CAN, if the selected value is greater than 8, the maximum number of 8 bytes is transmitted in the data frame with the data length code specified in the DLC.

19.6.1.7 Data Field

The number of transferred data bytes is determined by the data length code. The first bit transmitted is the most significant bit of the data byte 1.

Note: The number of transferred data bytes is defined by the DLC. The first data bit transmitted is the MSB of data byte 1. The received data length code located in the frame information byte represents the real data length code, which may be greater than 8.

Note: For transmission in classic CAN frame format, when the DLC is specified greater than 8, only 8 bytes of data are transmitted. When the RTR bit is set to 1, no data bytes are transmitted, regardless of the value of the DLC field.

Note: ID(10) is the first bit transmitted after SOF.

19.6.2 Memory Buffer FIFO Layout for Extended Frame Messages

Messages to send and receive are stored in the transmit and receive FIFO. The transmit FIFO can hold one message a time. The receive FIFO buffer can hold many messages at a time. *Table 19-7* shows the transmit and receive FIFO buffer layout for Extended Frame messages.

Table 19-7: Transmit and Receive FIFO Buffer Layout for Extended Frames

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	ID[12:5]								ID[20:13]								ID[28:21]								IDE	SRR	FDF	BRS	DLC			
0x01	DATA 3 (DLC<3 then free)								DATA 2 (DLC<2 then free)								DATA 1 (DLC<1 then free)								ID[4:0]					RRS\RTR	ESI	Reserved
0x02	DATA 7 (DLC<9 then free)								DATA 6 (DLC<8 then free)								DATA 5 (DLC<7 then free)								DATA 4 (DLC<6 then free)							

Offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x03	DATA 11 (DLC<10 then free)								DATA 10 (DLC<9 then free)								DATA 9 (DLC<9 then free)								DATA 8 (DLC<9 then free)							
0x04	DATA 15 (DLC<11 then free)								DATA 14 (DLC<10 then free)								DATA 13 (DLC<10 then free)								DATA 12 (DLC<10 then free)							
0x05	DATA 19 (DLC<11 then free)								DATA 18 (DLC<11 then free)								DATA 17 (DLC<11 then free)								DATA 16 (DLC<11 then free)							
…																																
0x0F	DATA 59 (DLC<15 then free)								DATA 58 (DLC<15 then free)								DATA 57 (DLC<15 then free)								DATA 56 (DLC<15 then free)							
0x10	DATA 64 (DLC<15 then free)								DATA 62 (DLC<15 then free)								DATA 61 (DLC<15 then free)								DATA 60 (DLC<15 then free)							
0x11	free								free								free								DATA 64 (DLC<15 then free)							

Table 19-8: Extended Frame Fields

Field	Description
IDE	**Identifier Extension Flag** This bit specifies that the frame is using Standard Identifier or Extended Identifier. Valid in both classic CAN and FD CAN frames. 0: Standard Message Identifier. 1: Extended Message Identifier.
SRR	**Substitute Remote Request** This bit is transmitted in Extended formats (classic and FD) at the position of the RTR bit in base formats. This bit is always transmitted as recessive in extended formats (classic CAN or CAN FD). 0: Dominant state on bus. 1: Recessive state on bus.
FDF	**FD Frame Format Indicator** This field indicates a CAN FD frame. 0: Classic CAN frame format. 1: CAN FD frame format.
BRS	**Baud Rate Switch Bit** 0: Frame without baud rate switching. 1: Frame with baud rate switching. *Note: This field is only available with CAN FD mode.*
ESI	**Error State Indicator** 0: Transmitter of this message is Active Error node. 1: Transmitter of this message is Passive Error node. *Note: This field is only available with CAN FD mode.*
RTR/RRS	**Remote Transmission Request** This bit specifies the frame is standard CAN data frame or standard CAN remote frame. For CAN FD format, this bit indicates bus state at RRS position of the frame. 0: Standard CAN data frame. 1: Standard CAN remote frame. *Note: CAN FD format does not support remote frames. In a CAN FD frame at this position, the transmitter always sends the dominant state.*
DLC[3:0]	**Data Length Code** Frame data length code field of classic CAN and CAN FD frame.
ID[10:0]	**Standard Message Identifier**
DATA N	**Data Bytes**

19.6.2.1 Identifier

In the extended frame, the message identifier is divided into two parts:

- Base identifier – 11 bits wide (ID[28] – ID[18]).
- Extended identifier – 18 bits wide (ID[17] – ID[0])

The base identifier consists of 11 bits (ID[28] to ID[18]) and is equivalent to the standard frame identifier. It defines the extended frame's base priority.

19.6.2.2 Substitute Remote Request (SRR)

When the CAN peripheral is a transmitter, it always sends recessive at this position in extended CAN frame format and extended CAN FD frame format. The receiver accepts recessive or dominant at this position.

19.6.2.3 Remote Transmission Request (RTR/RRS)

A remote frame is transmitted on the CAN bus if this bit is set. This means that no data bytes are included within this frame. Nevertheless, it is necessary to specify the correct DLC, which depends on the corresponding data frame with the same identifier coding. If the RTR bit is not set, a data frame is sent, including the number of data bytes as specified by the data length code. In CAN FD mode, remote frames are not supported.

19.6.2.4 FD Frame (FDF)

This bit distinguishes between a classic CAN frame and a CAN FD frame. When this bit is set in the transmit data buffer in FD mode (*CAN_FDCTRL.fden* = 1), a CAN FD frame is sent. If the *CAN_FDCTRL.fden* bit is cleared, the setting for FDF is ignored, and a classic CAN frame is sent. If the FDF bit is a logical one in the received data, the received frame was sent in FD mode if the CAN peripheral is in FD mode (*CAN_FDCTRL.fden* = 1). When the CAN peripheral is set to classic CAN mode, the FDF bit is always equal to 0.

19.6.2.5 Bit Rate Switch (BRS)

When transmitting, if this bit is set to 1, the FD frame is sent with bit rate switching in the data phase. When receiving data, if this bit is set to 1, the received frame has bit rate switching if the peripheral is in FD mode (*CAN_FDCTRL.fden* = 1).

19.6.2.6 Error State Indicator (ESI)

This bit is only valid if the CAN peripheral is in FD mode (*CAN_FDCTRL.fden* = 1) and the received/transmitted frame is an FD frame. If this bit is a 1, the transmitter of the message is in passive error node. If this bit is a 0, the transmitter of the message is an active error node.

19.6.2.7 Data Length Code (DLC)

The number of bytes in the data field of a message is coded by the data length code. At the start of a remote frame transmission, the data length code is not considered, due to the RTR bit being a logic 1 (remote). It denotes the number of transmitted/received data bytes to be logic 0. Nevertheless, the data length code must be specified correctly to avoid bus errors, if two CAN controllers start a remote frame transmission with the same identifier simultaneously. The range of the data byte count is 0 to 64 bytes and is coded, as shown in *Table 19-6*.

Table 19-9: DLC Coding to Data Byte Count

Number of Bytes	DLC[3]	DLC[2]	DLC[1]	DLC[0]
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0

Number of Bytes	DLC[3]	DLC[2]	DLC[1]	DLC[0]
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
12	1	0	0	1
16	1	0	1	0
20	1	0	1	1
24	1	1	0	0
32	1	1	0	1
48	1	1	1	0
64	1	1	1	1

In classic CAN, if the selected value is greater than 8, the maximum number of 8 bytes is transmitted in the data frame with the data length code specified in the DLC.

19.6.2.8 Data Field

The DLC determines the number of transferred data bytes. The first bit transmitted is the most significant bit of data byte 1.

Note:. The DLC defines the number of transferred data bytes. The first data bit transmitted is the MSB of data byte 1. The received data length code located in the frame information byte represents the real data sent length code, which may be greater than 8.

Note: The DLC defines the number of transferred data bytes. The first data bit transmitted is the MSB of data byte 1. The received data length code located in the frame information byte represents the real sent data length code, which may be greater than 8.

Note: For transmission in classic CAN frame format, when DLC is specified as greater than 8, only 8 bytes of data are transmitted. When the RTR bit is set to 1, no data bytes are transmitted, regardless of the DLC value.

Note: ID[28] is the first bit transmitted after the SOF.

Hra

19.7 CAN Registers

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

Table 19-10: CAN Registers

Offset	Register Name	Register Width	Description
[0x0000]	*CAN_MODE*	8-bit	Mode Register
[0x0001]	*CAN_CMD*	8-bit	Command Register
[0x0002]	*CAN_STAT*	8-bit	Status Register
[0x0003]	*CAN_INTFL*	8-bit	Interrupt Flag Register
[0x0004]	*CAN_INTEN*	8-bit	Interrupt Enable Register
[0x0005]	*CAN_RMC*	8-bit	Receive Message Counter Register r
[0x0006]	*CAN_BUSTIM0*	8-bit	Bus Timing 0 Register
[0x0007]	*CAN_BUSTIM1*	8-bit	Bus Timing 1 Register

Offset	Register Name	Register Width	Description
[0x000B]:[0x0008]	*CAN_TXFIFO8[3]*:*CAN_TXFIFO8[0]*	8-bit	8-Bit Transmit FIFO 3:0 Registers
[0x000A]:[0x0008]	*CAN_TXFIFO16[1]*:*CAN_TXFIFO16[0]*	16-bit	16-Bit Transmit FIFO 1:0 Registers
[0x0008]	*CAN_TXFIFO32*	32-bit	32-Bit Transmit FIFO Register
[0x000F]:[0x000C]	*CAN_RXFIFO8[3]*:*CAN_RXFIFO8[0]*	8-bit	8-Bit Receive FIFO 3:0 Registers
[0x000E]:[0x000C]	*CAN_RXFIFO16[1]*:*CAN_RXFIFO16[0]*	16-bit	16-Bit Receive FIFO 1:0 Registers
[0x000C]	*CAN_RXFIFO32*	32-bit	32-Bit Receive FIFO Register
[0x0013]:[0x0010]	*CAN_ACR8[3]*:*CAN_ACR8[0]*	8-bit	8-Bit Acceptance Code 3:0 Response Registers
[0x0012]:[0x0010]	*CAN_ACR16[1]*:*CAN_ACR16[0]*	16-bit	16-Bit Acceptance Code 1:0 Response Registers
[0x0010]	*CAN_ACR32*	32-bit	32-Bit Acceptance Code Response Register
[0x0017]:[0x0014]	*CAN_AMR8[3]*:*CAN_AMR8[0]*	8-bit	8-Bit Acceptance Mask 3:0 Response Registers
[0x0016]:[0x0014]	*CAN_AMR16[1]*:*CAN_AMR16[0]*	16-bit	16-Bit Acceptance Mask 1:0 Response Registers
[0x0014]	*CAN_AMR32*	32-bit	32-Bit Acceptance Mask Response Register
[0x0018]	*CAN_ECC*	8-bit	Error Code Capture Register
[0x0019]	*CAN_RXERR*	8-bit	Receive Error Counter Register
[0x001A]	*CAN_TXERR*	8-bit	Transmit Error Counter Register
[0x001B]	*CAN_ALC*	8-bit	Arbitration Lost Code Capture Register
[0x001C]	*CAN_NBT*	32-bit	Nominal Bit Time Extended Register
[0x0020]	*CAN_DBT_SSPP*	32-bit	Data Bit Time and Second Sample Point Position Register
[0x0024]	*CAN_FDCTRL*	8-bit	FD Control Register
[0x0025]	*CAN_FDSTAT*	8-bit	FD Status Register
[0x0026]	*CAN_DPERR*	8-bit	Data Phase Error Register
[0x0027]	*CAN_APERR*	8-bit	Arbitration Phase Error Counter Register
[0x0028]	*CAN_TEST*	8-bit	Test Register
[0x0029]	*CAN_WUPCLKDIV*	8-bit	Wake-up Timer Clock Divisor Register
[0x002A]	*CAN_WUPFT*	16-bit	Wake-up Filter Time Low Register
[0x002C]	*CAN_WUPET*	16-bit	Wake-up Expire Time Low Register
[0x0030]	*CAN_RXDCNT*	16-bit	Receive FIFO Data Count Register
[0x0032]	*CAN_TXSCNT*	8-bit	Transmit Buffer Space Counter Register
[0x0033]	*CAN_TXDECMP*	8-bit	Transmitter Delay Compensation Register
[0x0034]	*CAN_EINTFL*	8-bit	Extended Interrupt Flag Register
[0x0035]	*CAN_EINTEN*	8-bit	Extended Interrupt Mask Register
[0x0036]	*CAN_RXTO*	16-bit	Receive Timeout Register

19.7.1 Register Details

Table 19-11: Mode Register

Mode			CAN_MODE	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
7	slp	R/W	0	**Sleep Mode** Setting this field to 1 triggers a sleep request to the CAN controller. The CAN controller waits for the CAN bus to be inactive and then enters sleep mode. After writing a 1 to this field, hardware does not set the field to 1 until the CAN bus is inactive and the controller enters sleep. 0: CAN is in active mode. 1: CAN is in sleep mode. *Note: This field cannot be set to 1 if CAN_MODE.rst is set to 1. It is an invalid state to set both this field and CAN_MODE.rst to 1 simultaneously.*
6	dma	R/W	0	**DMA Mode Enable** 0: DMA mode disabled 1: DMA mode enabled
5:3	rxtrig	R/W	0	**Receive FIFO Trigger Level** Set this field to the number of 32-bit words in the receive FIFO to trigger a DMA operation. 0b000: 1 0b001: 4 0b010: 8 0b011: 16 0b100: 32 0b101: 64 0b111: Reserved *Note: This field is only used if DMA mode is enabled (CAN_MODE.dma = 1).*
2	rst	R/W	1	**Reset Mode** Set this field to 1 to enable reset mode of the CAN controller. The CAN controller must be in this mode to configure operation. In reset mode, frame reception and transmission are disabled. See *Operating Modes* for details of the operating modes. When this field is 0, the CAN controller operating mode is controlled using the *CAN_MODE.lom* field. 0: CAN controller not in reset mode. 1: CAN controller in reset mode.
1	lom	R/W	0	**Listen Only Mode Enable** Set this field to 1 to enable listen-only mode or 0 to enable normal operation. See *Operating Modes* for details. 0: Normal mode 1: Listen only mode *Note: This field is only used if CAN_MODE.rst is set to 0.*
0	afm	R/W	0	**Hardware Acceptance Filter Scheme** This field selects either dual or single filter for the hardware acceptance filter scheme. 0: Dual filter 1: Single filter *Note: This field can only be modified if CAN_MODE.rst is set to 0.*

Table 19-12: Command Register

Command			CAN_CMD	[0x0001]
Bits	**Field**	**Access**	**Reset**	**Description**
7:3	-	W	0	**Reserved**
2	txreq	W	0	**Transmit Request** Set this field to 1 to initiate a frame transmission. Setting this field to 1 simultaneously with *CAN_CMD.abort* to 1 initiates a single shot transmission. A single shot transmission does not perform a lost frame retransmission if a bus error or arbitration occur.
1	abort	W	0	**Abort Transmission** If a transmission is started by setting *CAN_CMD.txreq* to 1 and the transmission has not yet started, setting this field to 1 aborts the transmission. Simultaneously setting this field to 1 and *CAN_CMD.txreq* to 1 initiates a single shot transmission.
0	-	W	0	**Reserved**

Table 19-13: Status Register

Status			CAN_STAT	[0x0002]
Bits	**Field**	**Access**	**Reset**	**Description**
7	rxbuf	R	0	**Receive Buffer Status** This field indicates if at least one message is in the receive FIFO. When set, at least one message is in the receive FIFO, when clear, no messages are in the receive FIFO.
6	dor	R	0	**Data Overrun Status** This field indicates if a receive overrun occurred. If set, a receive overrun error occurred.
5	txbuf	R	1	**Transmit Buffer Status** This field indicates if the transmit buffer is available for software to write to it. If this field reads 0, the transmit buffer is locked and cannot be written. The transmit buffer is locked by hardware when a message is being transmitted or a transmission is pending. 0: Transmit buffer is locked. 1: Transmit buffer unlocked.
4	-	RO	0	**Reserved**
3	rx	R	0	**Receive Status** This field is set to 1 by the hardware when a message is being received.
2	tx	R	0	**Transmit Status** This field is set to 1 by hardware when the CAN controller is transmitting a message.
1	err	R	0	**Error Status** This field is set by hardware when at least one of the CAN error counters reaches the warning limit of 96.

Status			CAN_STAT	[0x0002]
Bits	**Field**	**Access**	**Reset**	**Description**
0	bus_off	R	0	**Bus Off Status** If this field is set, the CAN controller is in the bus off state and cannot transmit or receive frames. This field is set to 1 by hardware when the transmit error counter exceeds 255. When this field is set by hardware, the CAN controller automatically is placed into reset mode, and if enabled, an error warning interrupt is generated. The transmit error counter is then set to 127 and the receive error counter is cleared. The CAN controller remains in reset until the software clears the reset bit. Once the reset state is cleared, the CAN controller waits for 128 occurrences of the bus free signal (11 consecutive recessive bits) counting down the transmit error counter. When the transmit error counter reaches 0, the bus_off field is cleared and the bus is in the bus on state, the error counters are reset, and an error warning interrupt is generated if enabled. 0: Bus is in the bus on state 1: Bus in in the bus off state

Table 19-14: Interrupt Flag Register

Interrupt Flag			CAN_INTFL	[0x0003]
Bits	**Field**	**Access**	**Reset**	**Description**
7	wu	R/W1C	0	**Wake-up Interrupt** This field is set to 1 when the CAN controller is woken up from sleep by either a wake-up pattern detection or by software clearing the *CAN_MODE.slp* bit. Write 1 to clear.
6	al	R/W1C	0	**Arbitration Lost Interrupt** This field is set to 1 when the device is transmitting a message and loses arbitration, thus becoming a receiver. Read the *CAN_ALC* register to determine the location of the bit in the frame where arbitration lost occurred. Write 1 to clear.
5	erwarn	R/W1C	0	**Error Warning Interrupt** This field is set to 1 when there is a change of state of the error status (*CAN_STAT.err*) or bus status (*CAN_STAT.bus_off*). Write 1 to clear.
4	erpsv	R/W1C	0	**Error Passive Interrupt** This field is set to 1 when a state change to or from passive occurs (i.e., a state change of active-to-passive or passive-to-active). Write 1 to clear.
3	rx	R/W1C	0	**Receive Interrupt** This field is set to 1 when at least 1 message is in the receive FIFO. After reading a message from the receive FIFO, the software must write this field to 1 to decrement the receive message counter (RMC). The receive message counter is not automatically decremented by reading a message from the FIFO.
2	tx	R/W1C	0	**Transmission Interrupt** This field is set to 1 after a successful transmission occurs. Write 1 to clear.
1	berr	R/W1C	0	**Bus Error Interrupt** This field is set when a bus error occurs while transmitting or receiving a message.
0	dor	R/W1C	0	**Data Overrun Interrupt** This field is set when a receive FIFO overrun occurs. Write 1 to clear.

Table 19-15: Interrupt Enable Register

Interrupt Enable			CAN_INTEN	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
7	wu	R/W	0	**Wake-up Interrupt Enable** 0: Disabled 1: Enabled
6	al	R/W	0	**Arbitration Lost Interrupt Enable** 0: Disabled 1: Enabled
5	erwarn	R/W	0	**Error Warning Interrupt Enable** 0: Disabled 1: Enabled
4	erpsv	R/W	0	**Error Passive Interrupt Enable** 0: Disabled 1: Enabled
3	rx	R/W	0	**Receive Interrupt Enable** 0: Disabled 1: Enabled
2	tx	R/W	0	**Transmit Interrupt Enable** 0: Disabled 1: Enabled
1	berr	R/W	0	**Bus Error Interrupt Enable** 0: Disabled 1: Enabled
0	dor	R/W	0	**Data Overrun Interrupt Enable** 0: Disabled 1: Enabled

Table 19-16: Receive Message Counter Register

Receive Message Counter			CAN_RMC	[0x0005]
Bits	**Field**	**Access**	**Reset**	**Description**
7:5	-	RO	0	**Reserved**
4:0	num_msgs	R	0	**Number of Frames Stored in the Receive FIFO** This field is the number of fields stored in the receive FIFO. This field is incremented by hardware on each successful frame reception. This field is decremented by clearing the receive interrupt (*CAN_INTFL.rx*) field.

Table 19-17: Bus Timing 0 Register

Bus Timing 0			CAN_BUSTIM0	[0x0006]
Bits	**Field**	**Access**	**Reset**	**Description**
7:6	sjw	R/W	0	**Synchronization Jump Width** Set this field to the maximum number of time quanta (t_q) for the SJW used during resynchronization. The following equation defines the maximum number of clock cycles a bit period may be changed by one resynchronization. $t_{SJW} = t_{SCLK} \times (CANn\_BUSTIM0.sjw + 1)$
5:0	br_clkdiv	R/W	0	**Baud Rate Prescaler** Set this field to the desired BRP to achieve the nominal bit rate. See *Bit Timing* for details on setting the nominal bit rate.

Table 19-18: Bus Timing 1 Register

Bus Timing 1			CAN_BUSTIM1	[0x0007]
Bits	**Field**	**Access**	**Reset**	**Description**
7	sam	R/W	0	**Number of Bus Level Samples** This field selects either a single bus sample or three bus samples per bit. When set to 0, the single sample occurs at the end of TSEG1. When this field is set to 1, the three samples are used in a voting scheme and occur on the last three time quanta of TSEG1. See *Figure 19-2* for more information. A single sample is recommended for high-speed CAN buses. Three samples are recommended for low/medium speed CAN buses where there is a benefit from filtering spikes. 0: Single sample 1: 3 samples
6:4	tseg2	R/W	0	**TSEG2 Time Quanta Select** Set this field to the number of time quanta for TSEG2. See *Bit Timing* for details of the nominal bit timing. 0: 1 t_q 1: 2 1 t_q 2: 3 t_q 3: 4 t_q 4: 5 t_q 5: 6 t_q 6: 7 t_q 7: 8 t_q
3:0	tseg1	R/W	0	**TSEG1 Time Quanta Select** Set this field to the number of time quanta for TSEG1. See *Bit Timing* for details of the nominal bit timing. 0: 1 t_q 1: 2 t_q 2: 3 t_q …: … 14: 15 t_q 15: 16 t_q

19.7.1.1 Transmit FIFO Layout

The transmit FIFO is accessible in 8-bit, 16-bit, and 32-bit widths. The FIFO is mapped as four consecutive bytes from address offset [0x0008] through [0x000B]. When writing 32-bit words to the transmit FIFO, the write pointer is incremented after each write. For 16-bit writes, the write pointer is incremented after writing to address offset [0x000A], and when writing 8-bits, the write pointer is incremented after 4 writes. When a successful transmission occurs, the *CAN_INTFL.tx* field is set to 1 by hardware. Clearing the *CAN_INTFL.tx* field to 0 resets the transmit FIFO's internal write pointer to 0.

Table 19-19: 32-Bit Transmit FIFO Register

8-Bit Transmit FIFO 0			CAN_TXFIFO8[0]	[0x0008]
16-Bit Transmit FIFO 0			CAN_TXFIFO16[0]	[0x0008]
32-Bit Transmit FIFO			CAN_TXFIFO32	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	W	0	**8-Bit Transmit FIFO Least Significant Byte Data** Write CAN frame data to this register in 8, 16, or 32-bit widths. A 32-bit write to this address writes to *CAN_TXFIFO8[3]*:*CAN_TXFIFO8[2]*:*CAN_TXFIFO8[1]*:*CAN_TXFIFO16[0]* and automatically increments the internal write pointer. A 16-bit write to this address writes *CAN_TXFIFO8[1]*:*CAN_TXFIFO8[0]*. An 8-bit write to this address writes *CAN_TXFIFO8[0]*.

Table 19-20: 8-Bit Transmit FIFO 1 Register

8-Bit Transmit FIFO 1			CAN_TXFIFO8[1]	[0x0009]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	W	0	**8-Bit Transmit FIFO Data** Write CAN frame data to this register in 8-bit width.

Table 19-21: 8-Bit Transmit FIFO 2 Register

8-Bit Transmit FIFO 2			CAN_TXFIFO8[2]	[0x000A]
16-Bit Transmit FIFO 1			CAN_TXFIFO16[1]	[0x000A]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	W	0	**8-Bit Transmit FIFO Data** Write CAN frame data to this register in 8, or 16-bit widths. A 16-bit write to this address writes to *CAN_TXFIFO8[3]*:*CAN_TXFIFO8[2]* and automatically increments the internal write pointer. An 8-bit write to this address writes *CAN_TXFIFO8[2]*.

Table 19-22: 8-Bit Transmit FIFO 3 Register

8-Bit Transmit FIFO 3			CAN_TXFIFO8[3]	[0x000B]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	W	0	**8-Bit Transmit FIFO Most Significant Byte Data** Write CAN frame data to this register in 8-bit width. An 8-bit write to this address writes to *CAN_TXFIFO8[3]* and automatically increments the internal write pointer.

Table 19-23: 32-Bit Receive FIFO Register

8-Bit Receive FIFO			CAN_RXFIFO8[0]	[0x000C]
16-Bit Receive FIFO			CAN_RXFIFO16[0]	[0x000C]
32-Bit Receive FIFO			CAN_RXFIFO32	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R	0	**32-Bit Receive FIFO Data**

Table 19-24: 8-Bit Receive FIFO 1 Register

8-Bit Receive FIFO 1			CAN_RXFIFO8[1]	[0x000D]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R	0	**8-Bit Receive FIFO Data**

Table 19-25: 8-Bit Receive FIFO 2 Register

8-Bit Receive FIFO 2			CAN_RXFIFO8[2]	[0x000E]
16-Bit Receive FIFO 1			CAN_RXFIFO16[1]	[0x000E]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	-	R	0	**16-Bit Receive FIFO Data**

Table 19-26: 8-Bit Receive FIFO 3 Register

8-Bit Receive FIFO 3			CAN_RXFIFO8[3]	[0x000F]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R	0	**8-Bit Receive FIFO Data**

Table 19-27: 32-Bit Acceptance Code Register

32-Bit Acceptance Code			CAN_ACR32	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**32-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-28: 16-Bit Acceptance Code 0 Register

16-Bit Acceptance Code 0			CAN_ACR16[0]	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	-	R/W	0	**16-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-29: 16-Bit Acceptance Code 1 Register

16-Bit Acceptance Code 1			CAN_ACR16[1]	[0x0012]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	-	R/W	0	**16-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-30: 8-Bit Acceptance Code 0 Register

8-Bit Acceptance Code 0			CAN_ACR8[0]	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-31: 8-Bit Acceptance Code 1 Register

8-Bit Acceptance Code 1			CAN_ACR8[1]	[0x0011]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-32: 8-Bit Acceptance Code 2 Register

8-Bit Acceptance Code 2			CAN_ACR8[2]	[0x0012]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-33: 8-Bit Acceptance Code 3 Register

8-Bit Acceptance Code 3			CAN_ACR8[3]	[0x0013]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit ACR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-34: 32-Bit Acceptance Mask Register

32-Bit Acceptance Mask			CAN_AMR32	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W	0	**32-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-35: 16-Bit Acceptance Mask 0 Register

16-Bit Acceptance Mask 0			CAN_AMR16[0]	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	-	R/W	0	**16-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-36: 16-Bit Acceptance Mask 0 Register

16-Bit Acceptance Mask 1			CAN_AMR16[1]	[0x0016]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	-	R/W	0	**16-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-37: 8-Bit Acceptance Mask 0 Register

8-Bit Acceptance Mask 0			CAN_AMR8[0]	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-38: 8-Bit Acceptance Mask 1 Register

8-Bit Acceptance Mask 1			CAN_AMR8[1]	[0x0015]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-39: 8-Bit Acceptance Mask 2 Register

8-Bit Acceptance Mask 2			CAN_AMR8[2]	[0x0016]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-40: 8-Bit Acceptance Mask 3 Register

8-Bit Acceptance Mask 3			CAN_AMR8[3]	[0x0017]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R/W	0	**8-Bit AMR Data** See *Acceptance Codes and Filtering* for details on the usage of this register.

Table 19-41: Error Code Capture Register

Error Code Capture			CAN_ECC	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
7	rxwrn	R	0	**Receive Error Counter Warning** This field is set when the receive error counter (*CAN_RXERR*) is greater than or equal to 96. *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CAN_INTFL.berr).*
6	txwrn	R	0	**Transmit Error Counter Warning** This field is set when the transmit error counter (*CAN_TXERR*) is greater than or equal to 96. *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CAN_INTFL.berr).*
5	edir	R	0	**The Direction of Transfer When Error Occurred** This field indicates the direction of the error when it occurred and applies to the error codes in bits 4:0 of this register. 0: Transmission 1: Reception *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CAN_INTFL.berr).*
4	acker	R	0	**ACK Error** 0: Normal operation 1: Error occurred *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CAN_INTFL.berr).*

Error Code Capture			CAN_ECC	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
3	frmer	R	0	**Form Error** 0: Normal operation 1: Error occurred *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CAN_INTFL.berr).*
2	crcer	R	0	**CRC Error** 0: Normal operation 1: Error occurred *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CAN_INTFL.berr).*
1	stfer	R	0	**Stuff Error**] 0: Normal operation 1: Error occurred *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CAN_INTFL.berr).*
0	ber	R	0	**Bit Error** 0: Normal operation 1: Error occurred *Note: This register is not updated by hardware until the error is acknowledged by software clearing the bus error interrupt (CAN_INTFL.berr).*

Table 19-42: Receive Error Counter Register

Receive Error Counter			CAN_RXERR	[0x0019]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R	0	**Receive Error Counter** This field reflects the current value of the receive error counter. If a bus off event occurs (*CAN_STAT.bus_off* = 1), this field is reset to 0 by the hardware.

Table 19-43: Transmit Error Counter Register

Transmit Error Counter			CAN_TXERR	[0x001A]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	-	R	0	**Transmit Error Counter** This field reflects the current value of the transmit error counter. If a bus off event (*CAN_STAT.bus_off* = 1) occurs, this field is set to 127 by the hardware and is decremented the protocol defined time (128 occurrences of the bus free signal). Reading this register during this time gives information about the status of the bus off recovery. *Note: The internal transmit error counter is 9-bits, whereas this register is only 8-bits, so only bits 7:0 of the transmit error counter are readable by software.*

Table 19-44: Arbitration Lost Code Capture Register

Arbitration Lost Code Capture			CAN_ALC	[0x001B]
Bits	**Field**	**Access**	**Reset**	**Description**
7:6	-	RO	0	**Reserved**

Arbitration Lost Code Capture			CAN_ALC	[0x001B]
Bits	**Field**	**Access**	**Reset**	**Description**
5:0	alc	R	0	**Arbitration Lost Bit Position** This field contains the bit position where arbitration is lost. See *Table 19-2* for detailed mapping of this field to the identifier of the arbitration lost. *Note: This field is not updated until software acknowledges the arbitration lost interrupt by clearing the CAN_INTFL.al field.*

Table 19-45: Arbitration Nominal Bit Time Register

Arbitration Nominal Bit Time			CAN_NBT	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:25	nsjw	RO	0	**Synchronization Jump Width in Arbitration** This field defines the synchronization jump width in CAN FD frames when extended bit time (*CAN_FDCTRL.extbt* = 1) is selected. *Note: This field is reserved and should not be used.*
24:18	nseg2	RO	0	**Nominal Segment 2 Time in Arbitration** This field defines the time after the sample point in CAN FD frames when extended bit time (*CAN_FDCTRL.extbt* = 1) is selected. *Note: This field is reserved and should not be used.*
17:10	nseg1	RO	0	**Nominal Segment 1 Time in Arbitration** This field defines the time before the sample point in CAN FD frames when extended bit time (*CAN_FDCTRL.extbt* = 1) is selected. *Note: This field is reserved and should not be used.*
9:0	nbrp	RO	0	**Baud Rate Prescaler in Arbitration Phase** This field selects the time quantum in the arbitration phase in the range of 1 to 1024 clock periods according to the equation: $t_q = (CAN\_NBT.nbrp + 1) \times t_{clk}$ *Note: This field is only used in CAN FD frames when extended bit time (CAN_FDCTRL.extbt = 1) is selected.* *Note: This field is reserved and should not be used.*

Table 19-46: Data Bit Time and Second Sample Point Position Register

Data Bit Time and Second Sample Point Position			CAN_DBT_SSPP	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31	-	RO	0	**Reserved**
30:24	sspp	RO	0	**Secondary Sample Point Position** This field contains information about the secondary sample point. The secondary sample is defined as the sum of the measured transceiver loop delay and the value of the transceiver delay compensation (*CAN_TXDECMP*.). The transceiver loop delay is measured from the transmitted FDF to the r0 transition edge to the respective received one in every transmitted CAN FD frame. *Note: This field is reserved and should not be used.*

Data Bit Time and Second Sample Point Position		CAN_DBT_SSPP		[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
23:20	dsjw	RO	0	**Synchronization Jump Width in Data Phase** The bit time in the data phase must be less than or equal to the bit time in the arbitration phase. *Note: This field is only used when the CAN controller is in FD mode (CAN_FDCTRL.fden = 1).* *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*
19:16	dseg2	RO	0	**DSEG2** This field is the time segment after the sample in the data phase. The bit time in the data phase must be less than or equal to the bit time in the arbitration phase. *Note: This field is only used when the CAN controller is in FD mode (CAN_FDCTRL.fden = 1).* *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*
15:10	dseg1	RO	0	**DSEG1** This field is the time segment before the sample point in the data phase. The bit time in the data phase must be less than or equal to the bit time in the arbitration phase. *Note: This field is only used when the CAN controller is in FD mode (CAN_FDCTRL.fden = 1).* *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*
9:0	dbrp	RO	0	**Baud Rate Prescaler in Data Phase** This field selects the time quantum in the data phase in the range of 1 to 1024 clock periods according to the equation: $t_q = (CAN\_DBT\_SSPP.dbrp + 1) \times t_{clk}$ The bit time in the data phase must be less than or equal to the bit time in the arbitration phase. *Note: This field is only used when the CAN controller is in FD mode (CAN_FDCTRL.fden = 1).* *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*

Table 19-47: FD Control Register

FD Control		CAN_FDCTRL		[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
7	-	RO	0	**Reserved**
6	ped	RO	0	**Protocol Exception Disable** 0: A recessive state on the CAN bus at "res" position results in the CAN controller entering in bus integration state. 1: A recessive state on the CAN bus at "res" position results in a Form Error (*CAN_FDSTAT.frmerr* = 1). *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*

FD Control			CAN_FDCTRL	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
5	reom	RO	0	**Restricted Operating Mode** This field selects restricted operating mode. When the CAN controller is in restricted operating mode, the node does not send error frames (active or passive) or overload frames. If an error or overload occurs, the CAN controller waits for bus idle before sending dominant bits. 0: Normal mode (error frames and overload frames are sent when they occur) 1: Restricted operating mode. *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*
4	dar	RO	0	**Disable Auto Retransmission** Set this field to 1 to disable automatic retransmission. 0: Automatic retransmission enabled 1: Automatic retransmission disabled *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*
3	iso	RO	0	**ISO CAN FD Format Selection** 0: Frame format according to Bosch CAN FD specification 1: Frame format according to ISO 11898-1:2015 *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*
2	extbt	RO	0	**Bit Time Prescaler Select in Arbitration Phase** This field configures the bit time prescaler in the arbitration phase. 0: Use the bus timing registers (*CAN_BUSTIM0* and *CAN_BUSTIM1*) to configure the bit timing during the arbitration phase. 1: Use the nominal bit timing register (*CAN_NBT*) to configure the bit timing during the arbitration phase. *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*
1	brsen	RO	0	**Alternative Bit Rate Select** This field selects flexible data rate during the data phase. When this field is set to 1, the data rate is set using the *CAN_DBT_SSPP* register. 0: Bit rate is not switched inside a CAN FD frame 1: Bit rate is switched from the nominal bit rate during the arbitration phase to the alternate bit rate during the data phase. *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*
0	fden	RO	0	**CAN FD Enable** Set this field to 1 to enable CAN FD frames. 0: Classic CAN frame format 1: CAN FD frame format *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*

Table 19-48: FD Status Register

FD Status			CAN_FDSTAT	[0x0025]
Bits	**Field**	**Access**	**Reset**	**Description**
7:6	state	RO	0	**CAN Operation State** This field indicates the CAN controller operating state. 0b00: Integrating – waiting for 11 recessive bit after reset or bus off. 0b01: Idle – waiting for start of frame 0b10: Receiver – node operating as receiver 0b11: Transmitter – node operating as transmitter *Note: This field is reserved and should not be used.*
5	-	RO	0	**Reserved**
4	pee	RO	0	**Protocol Exception Event Indicator** This field indicates that the CAN controller detects recessive state on "res" position and entry to bus integration state. This field is only updated when the CAN controller is in FD mode (*CAN_FDCTRL.fden* = 1). *Note: This field is reserved and should not be used.*
3	stferr	RO	0	**Stuff Error Indicator** This field indicates stuff error occurred in data phase in CAN FD frame with the BRS bit set. This field is only updated when the CAN controller is in FD mode (*CAN_FDCTRL.fden* = 1). *Note: This field is reserved and should not be used.*
2	frmerr	RO	0	**Form Error Indicator** This field indicates that a fixed form bit field contains at least one illegal bit in data phase of CAN FD frame with the BRS bit set. This field is only updated when the CAN controller is in FD mode (*CAN_FDCTRL.fden* = 1). Note: This field is reserved and should not be used.
1	crcerr	RO	0	**CRC Error Indicator** This field indicates that the calculated CRC is different from the received CRC in the CAN FD frame. This field is only updated when the CAN controller is in FD mode (*CAN_FDCTRL.fden* = 1). *Note: This field is reserved and should not be used.*
0	biterr	RO	0	**Bit Error Indicator** When this bit is set to 1, an inconsistency occurred between the transmitted bit and the received bit in the CAN FD frame. This field is only updated when the CAN controller is in FD mode (*CAN_FDCTRL.fden* = 1). *Note: This field is reserved and should not be used.*

Table 19-49: Data Phase Error Register

Data Phase Error			CAN_DPERR	[0x0026]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	dperr	RO	0	**Data Phase Error Counter** This field returns the error count, which is incremented each time the transmitter and receiver detect an error during the data phase of the CAN FD frame. The counter stops counting at 255. Reset this field by placing the CAN controller into reset (*CAN_MODE.rst* = 1). *Note: This field is reserved and should not be used.*

Table 19-50: Arbitration Phase Error Register

Arbitration Phase Error			CAN_APERR	[0x0027]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	aperr	R	0	**Arbitration Phase Error Counter** This field returns the error count, which is incremented each time the transmitter and receiver detect an error during the arbitration phase of the frame. The counter stops counting at 255. Reset this field by placing the CAN controller into reset (*CAN_MODE.rst* = 1).

Table 19-51: Test Register

Test			CAN_TEST	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
7:2	-	RO	0	**Reserved**
1	txc	R/W	0	**Transmit Pin Disconnect** Set this field to 1 to leave the transmit pin disconnected while in test mode. When the transmit pin is disconnected, it remains in recessive state (CAN TX = 1). 0: Transmit pin is connected and any messages transmitted in loop back mode are visible on the CAN TX pin. 1: Transmit pin disconnected and held in the recessive state (CAN TX = 1). *Note: This field can only be modified when the CAN is in reset mode (CAN_MODE.rst = 1).*
0	lben	R/W	0	**Loop Back Mode Enable** Set this field to 1 to enable loop back mode. When the CAN is in loop back mode, the CAN RX pin is disconnected from the CAN peripheral. 0: Disabled. 1: Enabled. *Note: This field can only be modified when the CAN is in reset mode (CAN_MODE.rst = 1).*

Table 19-52: Wake-up Clock Divisor Register

Wake-up Clock Divisor			CAN_WUPCLKDIV	[0x0029]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	wupdiv	R/W	0	**Wake-up Time Prescaler** This field programs the wake-up counter clock used for the wake-up filter time and wake-up expire time fields. The following equation determines the clock cycle time counter. $t = (CAN\_WUPCLKDIV + 1)/f_{CLK}$

Table 19-53: Wake-up Filter Time Register

Wake-up Filter Time			CAN_WUPFT	[0x002A]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	wupft	R/W	0	**Wake-up Filter Time** Set this field to the desired wake-up filter time. For the wake-up to work, this field must be greater than 0. When the wake-up counter is greater than or equal to this field, the measured state in the CAN bus is valid. This field can only be written when the CAN controller is in reset (*CAN_MODE.rst* = 1).

Table 19-54: Wake-up Expire Time Register

Wake-up Expire Time			CAN_WUPET	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:20	-	RO	0	**Reserved**
19:0	wupet	R/W	0	**Wake-up Expire Time** Set this field to the desired wake-up expire time. When the internal wake-up counter is less than this field, the detected wake-up pattern is valid. This field can only be written when the CAN controller is in reset (*CAN_MODE.rst* = 1).

Table 19-55: Receive FIFO Data Count Register

Receive FIFO Data Count			CAN_RXDCNT	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	rxdcnt	R	0	**Receive FIFO Count** This field returns the number of bytes (aligned to 4) of the data in the receive FIFO. This field is set to 0 after a hardware reset or when the CAN controller is in reset state (*CAN_MODE.rst* = 1). This field is decremented after every read from the receive FIFO (*CAN_RXFIFO32*, *CAN_RXFIFO16*[0], *CAN_RXFIFO8*[0]).

Table 19-56: Transmit Buffer Space Count Register

Transmit Buffer Space Count			CAN_TXSCNT	[0x0032]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	txscnt	R	0x80	**Transmit FIFO Count** This field returns the number of bytes available in the transmit FIFO. When the transmit FIFO is empty (after reset or when a message is successfully sent), this field returns 0x80.

Table 19-57: Transmit Delay Compensation Register

Transmit Delay Compensation			CAN_TXDECMP	[0x0033]
Bits	**Field**	**Access**	**Reset**	**Description**
7	tdcen	RO	0	**Transceiver Delay Compensation Enable** 0: Disabled 1: Enabled *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*
6:0	tdco	RO	0	**Transceiver Delay Compensation Offset** This field sets the transceiver delay compensation offset added to the measured transceiver loop delay during frame transmission in CAN FD mode. *Note: This field can only be modified when the CAN controller is in reset (CAN_MODE.rst = 1).* *Note: This field is reserved and should not be used.*

Table 19-58: Extended Interrupt Status Flag Register

Extended Interrupt Status Flag			CAN_EINTFL	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
7:2	-	RO	0	**Reserved**

Extended Interrupt Status Flag			CAN_EINTFL	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
1	rx_to	R/W1C	0	**Receive FIFO Timeout Interrupt** This field indicates if the interrupt condition occurred. Write 1 to clear this field. 0: Normal operation 1: Interrupt occurred
0	rx_thd	R/W1C	0	**Receive FIFO Threshold Interrupt** This field is set to 1 if the receive FIFO level reaches the set receive FIFO threshold (*CAN_MODE.rxtrig*). Write 1 to clear. 0: Normal operation 1: Interrupt occurred

Table 19-59: Extended Interrupt Enable Register

Extended Interrupt Enable			CAN_EINTEN	[0x0035]
Bits	**Field**	**Access**	**Reset**	**Description**
7:2	-	RO	0	**Reserved**
1	rx_to	R/W	0	**Receive FIFO Timeout Interrupt Enable** Set this field to 1 to enable the receive FIFO timeout interrupt. 0: Disabled 1: Enabled
0	rx_thd	R/W	0	**Receive FIFO Threshold Interrupt Enable** Set this field to 1 to enable the interrupt. 0: Disabled 1: Enabled

Table 19-60: Receive FIFO Timeout Register

Receive FIFO Timeout			CAN_RXTO	[0x0036]
Bits	**Field**	**Access**	**Reset**	**Description**
15:0	rx_to	R/W	0	**Receive Timeout** This field specifies the receive timeout when at least 1 entry is in the receive FIFO. Set this field to the number of arbitration phase bit time sots to generate a timeout. When the receive timeout occurs, the *CAN_EINTFL.rx_to* field is set to 1 and an interrupt is generated if enabled (*CAN_EINTEN.rx_to* = 1). Note: This register is reset to 0 after a hardware reset and when the CAN controller is in the reset state (*CAN_MODE.rst* = 1).

20. Advanced Encryption Standard (AES)

The device provides a hardware AES accelerator to perform calculations on blocks up to 128 bits.

The features include:

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

20.1 Instances

Instances of the peripheral are listed in *Table 20-1*. Disable the peripheral by clearing *AES_CTRL*.*en* = 0 before writing to any register field.

Table 20-1: MAX32662 AES Instances

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

20.2 AES Key Generation

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

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

20.3 AES Key Storage

The MAX32662 includes a dedicated memory location to store AES keys loaded on a POR to the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers. The following steps describe the process of saving software generated keys to the internal non-volatile memory for loading after a POR to the AES_KEY registers.

1. Generate an AES key of the desired length.
2. Write the AES key starting at address 0x1080 2008.
   a. For example, to write a 128-bit AES key, write the key to KEY[15] - KEY[0], as shown in *Figure 20-1*.
3. Software must write the AES key magic values. Once written, the AES keys are automatically loaded to the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers after each POR. Write the following values to FMV1[3]:FMV1[0] and FMV0[3]: FMV0[0].
   a. FMV0[0] = 0x79
   b. FMV0[1] = 0xD4
   c. FMV0[2] = 0x86
   d. FMV1[3] = 0x2B
   e. FMV1[0] = 0x79
   f. FMV1[1] = 0xD4
   g. FMV1[2] = 0x86
   h. FMV0[3] = 0x2B

Figure 20-1: AES KEY Storage

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

20.4 Encryption of 128-Bit Blocks of Data Using FIFO

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

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

20.5 Encryption and Decryption of 128-Bit Blocks Using DMA

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

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

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

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

The procedure to use DMA encryption/decryption is:

1. Enable the AES and TRNG peripheral clocks:
   a. Clear *GCR_PCLKDIS1*.*aes* to 0.
   b. Clear *GCR_PCLKDIS1*.*trng* to 0.
2. Set *AES_CTRL*.*en* to 1 to enable the peripheral.
3. If using the POR key, ensure the key is correctly stored, as described in the *AES Key Storage* section.
4. If using a software generated key, write the key to the key registers, otherwise follow the steps in the *AES Key Generation* section.
   a. For a 128-bit key, write the key to the *AES_KEY_AES_KEY3*:*AES_KEY_AES_KEY0* registers.
   b. For a 192-bit key, write the key to the *AES_KEY_AES_KEY5*:*AES_KEY_AES_KEY0* registers.
   c. For a 256-bit key, write the key to the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers.
5. Initialize the AES receive and transmit channels for the DMA controller.
6. Verify the *AES_STATUS*.*busy* field reads 0.
7. Set *AES_CTRL*.*input_flush* to 1 to flush the input FIFO.
8. Set *AES_CTRL*.*output_flush* to 1 to flush the output FIFO.
9. Set *AES_CTRL*.*key_size* to the size of the loaded key.
10. Select encryption or decryption:
    a. Set *AES_CTRL*.*type* to 0 (encryption using the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers).
    b. Set *AES_CTRL*.*type* to 1 (decryption using the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0* registers).
    c. Set *AES_CTRL*.*type* to 2 (decryption using the locally generated decryption key from a previous encryption).
11. Set *AES_INTEN*.*done* to 0 for DMA operations.
12. Set *AES_CTRL*.*start* to 1 to start the AES DMA operation.
13. The DMA fills the FIFO, and the hardware begins the AES calculation.
14. When an AES calculation is completed, the AES hardware signals to the DMA that the data output FIFO is full and it should be emptied. If the DMA does not empty the FIFO before the next calculation is complete, the hardware sets *AES_STATUS*.*output_full* to 1.

Repeat steps 13 and step 14 if the DMA has new data to write to the data input FIFO.

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

20.6 Encryption of Blocks Less Than 128-Bits

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

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

20.7 Decryption

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

- Decrypt with external key.
- Decrypt with internal decryption key.

The internal decryption key is generated during an encryption operation. It may be necessary to complete a dummy encryption before doing the first decryption to ensure that it is generated.

20.8 Interrupt Events

The peripheral generates interrupts for the events shown in *Table 20-2*. Unless noted otherwise, each instance has its own independent set of interrupts, and higher-level flag and enable fields.

Multiple events may set an interrupt flag and generate an interrupt if the corresponding interrupt enable field is set. The flags must be cleared by the software, typically in the interrupt handler.

Table 20-2: Interrupt Events

Event	Local Interrupt Flag	Local Interrupt Enable
Data Output FIFO Overrun	*AES_INTFL*.ov	*AES_INTEN*.*ov*

Event	Local Interrupt Flag	Local Interrupt Enable
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*

20.8.1 Data Output FIFO Overrun

When an AES calculation is completed, the AES hardware signals to the DMA that the data output FIFO is full and that it should be emptied. If the DMA does not empty the FIFO before the next calculation is complete, a data output FIFO overrun event occurs, and the corresponding local interrupt flag is set.

20.8.2 Key Zero

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

20.8.3 Key Change

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

20.8.4 Calculation Done

The transition of *AES_STATUS.busy* = 1 to *AES_STATUS.busy* = 0 generates a calculation done event. The calculation done event interrupt must be disabled when using the DMA.

20.9 AES Registers

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

Table 20-3: 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

20.9.1 Register Details

Table 20-4: 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** 0b00: Encryption using the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0*. 0b01: Decryption using the *AES_KEY_AES_KEY7*:*AES_KEY_AES_KEY0*. 0b10: Decryption using the locally generated decryption key. 0b11: Reserved.

AES Control			AES_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
7:6	key_size	R/W	0	**Encryption Key Size** 0b00: 128-bits. 0b01: 192-bits. 0b10: 256-bits. 0b11: Reserved.
5	output_flush	R/W1O	0	**Flush Data Output FIFO** This field always read 0. 0: No action. 1: Flush.
4	input_flush	R/W1O	0	**Flush Data Input FIFO** This field always read 0. 0: No action. 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. This allows an AES calculation on less than 128-bits of data since an AES calculation normally starts when the data input FIFO is full. This field always read 0. 0: No action. 1: Start calculation.
2	dma_tx_en	R/W	0	**DMA Request To Write Data Input FIFO** When enabled, a DMA request is generated when the data input FIFO is empty. 0: Disabled. 1: Enabled.
1	dma_rx_en	R/W	0	**DMA Request To Read Data Output FIFO** When enabled, a DMA request is generated when the data output FIFO is full. 0: Disabled. 1: Enabled.
0	en	R/W	0	**AES Enable** 0: Disabled. 1: Enabled.

Table 20-5: 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: Normal operation. 1: Full.
3	output_em	R	0	**Output FIFO Empty** 0: Normal operation. 1: Empty.
2	input_full	R	0	**Input FIFO Full** 0: Normal operation. 1: Full.

AES Status			AES_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
1	input_em	R	0	**Input FIFO Empty** 0: Normal operation. 1: Empty.
0	busy	R	0	**AES Busy** 0: Normal operation. 1: Busy.

Table 20-6: AES Interrupt Flag Register

AES Interrupt Flag			AES_INTFL	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	key_one	R/W1C	0	**Key One Event Interrupt** If this interrupt occurs, the AES key is all 1s. 0: Normal operation. 1: Event occurred.
3	ov	R/W1C	0	**Data Output FIFO Overrun Event Interrupt** 0: Normal operation. 1: Event occurred.
2	key_zero	R/W1C	0	**Key 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 20-7: AES Interrupt Enable Register

AES Interrupt Enable			AES_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved**
4	key_one	R/W1C	0	**Key One Event Interrupt Enable** 0: Disabled. 1: Enabled.
3	ov	R/W1C	0	**Data Output FIFO Overrun Event Interrupt Enable** 0: Disabled. 1: Enabled.
2	key_zero	R/W1C	0	**Key Zero Event Interrupt Enable** 0: Disabled. 1: Enabled.
1	key_change	R/W1C	0	**Key Change Event Interrupt Enable** 0: Disabled. 1: Enabled.

AES Interrupt Enable			AES_INTEN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
0	done	R/W1C	0	**Calculation Done Event Interrupt Enable** This interrupt must be disabled when using the DMA. 0: Disabled. 1: Enabled.

Table 20-8: AES FIFO Register

AES Data			AES_FIFO	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**AES FIFO** Writing this register puts data to the data input FIFO. The hardware automatically starts a calculation after 4 words are written to this FIFO. Write the data with the least significant word first. Reading this register pulls data from the data output FIFO. The least significant word is read first.

20.10 AES Key Registers

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

Table 20-9: User AES Key Register Summary

Offset	Name	Description
[0x0000]	*AES_KEY_AES_KEY0*	*User AES Key 0 Register*
[0x0004]	*AES_KEY_AES_KEY1*	*User AES Key 1 Register*
[0x0008]	*AES_KEY_AES_KEY2*	*User AES Key 2 Register*
[0x000C]	*AES_KEY_AES_KEY3*	*User AES Key 3 Register*
[0x0010]	*AES_KEY_AES_KEY4*	*User AES Key 4 Register*
[0x0014]	*AES_KEY_AES_KEY5*	*User AES Key 5 Register*
[0x0018]	*AES_KEY_AES_KEY6*	*User AES Key 6 Register*
[0x001C]	*AES_KEY_AES_KEY7*	*User AES Key 7 Register*

20.10.1 Register Details

Table 20-10: User AES Key 0 Register

User AES Key 0			AES_KEY_AES_KEY0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 31:0**

Table 20-11: User AES Key 1 Register

User AES Key 1			AES_KEY_AES_KEY1	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 63:32**

Table 20-12: User AES Key 2 Register

User AES Key 2			AES_KEY_AES_KEY2	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 95:64**

Table 20-13: System AES Key 3 Register

User AES Key 3			AES_KEY_AES_KEY3	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 127:96**

Table 20-14: User AES Key 4 Register

User AES Key 4			AES_KEY_AES_KEY4	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 159:128**

Table 20-15: User AES Key 5 Register

User AES Key 5			AES_KEY_AES_KEY5	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 191:160**

Table 20-16: User AES Key 6 Register

User AES Key 6			AES_KEY_AES_KEY6	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 223:192**

Table 20-17: User AES Key 7 Register

User AES Key 7			AES_KEY_AES_KEY7	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	key	W	*	**User AES Key bits 255:224**

21. TRNG Engine

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

The general information in this section is provided only for completeness; customers are expected to use the Analog Devices UCL to generate random numbers.

Software can use the TRNG engine to generate AES keys using a hardware key derivation function (HKDF) and using the TRNG as input to the HKDF.

21.1 Registers

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

Table 21-1: TRNG Register Summary

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

21.1.1 Register Details

Table 21-2: TRNG Control Register

Control			TRNG_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15	keywipe	R/W	0	**Wipe Key** Write this field to 1 to wipe the TRNG key.
14:4	-	RO	0	**Reserved**
3	keygen	R/W	0	**Generate Key** Write this field to 1 to generate a key using the TRNG.
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 21-3: TRNG Status Register

Status			TRNG_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved**

Status			TRNG_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
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 21-4: TRNG Data Register

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

22. Secure Communication Protocol Bootloader (SCPBL)

The security of the secure boot and SCP communication relies on public-key infrastructure (PKI) using a manufacturer root key (MRK) authority, a customer root key (CRK), and a certificate. The certificate is a signed CRK using the MRK. The SCPBL stores the signed CRK and MRK values in OTP.

The SCPBL authenticates SCP transactions and verifies the integrity of internal program memory using digital signatures based on the MRK and CRK. SCP packets are signed at the session level to authenticate the sender and verify the integrity of the communication.

The device tests for a bootloader stimulus following every reset and wake up from all low-power modes except SLEEP and DEEPSLEEP. The stimulus includes receipt of the synchronization pattern and may or may not involve a physical pin, as shown in *Table 22-1*. If the stimulus is not present, the device executes the secure boot verifying the digital signature to ensure program memory is unaltered before beginning code execution.

22.1 Development Tools

The information in this chapter explains the details of the SCP protocol and the secure boot process. A software development kit (SDK) provides tools that build a complete application image, digitally signs it, and creates the packets used by the SCP. These tools automatically implement most of the low-level functions described in this document. Refer to the Secure Boot Tool User Guide for the MAX32662 for more information:
https://pdfserv.maximintegrated.com/en/an/ug7637-secure-boot-tools.pdf

The devices provided in the evaluation kit come preloaded with a test CRK (TCRK) and demonstration software.

22.2 Instances

There is only one instance of the SCPBL.

Table 22-1 shows the interface pins associated with the SCPBL. The hardware must ensure access to the default interface/stimulus pins (and a secondary interface and stimulus pin if implemented) if the application uses the SCPBL. There is no way to start an SCPBL session without setting the stimulus pin to its active state.

Table 22-1: MAX32662 Bootloader Physical Interface

Interface	Interface ID	Interface Pins Required For SCPBL	Default Stimulus Condition	Stimulus Pin Reassignment Options	Default SCPBL Activation Interval
UART (115200 bps) 8-N-1 (SCPBL) 8-N-2 (Host)	0x00	UART0A_RX (P0.11) UART0A_TX (P0.10) RSTN	P0.6 (active low)	All GPIO	6s

Multiple methods and algorithms ensure the integrity and authority of SCPBL communication, as shown in *Table 22-2*.

Table 22-2: MAX32662 Data Security and Integrity Methods

Item	Location	Range	Option	Length (Bytes)
Header ID	Start of transport layer header	N/A	0x48, 0x49, 0x53, 0x57, 0x45, 0x44, 0x47, 0x44	8
Application Image Digital Signature	End of the application image	Entire application image, excluding the signature itself	ECDSA-256	64
Header Checksum	End of header	The first 7 bytes of the transport header, then padded with 9 bytes of 0x00	The header checksum is AES-128 encryption with a 16-byte key of 0x00. The value is the MSB of the 16-byte digest.	1

Item	Location	Range	Option	Length (Bytes)
Data Checksum	End of transport layer	Session layer data	The data checksum is AES-128 encryption with a 16-byte key of 0x00. The value is the 4 MSB of the 16-byte digest.	4
Session Layer Protection Profile	The second nibble of the session layer	Session layer	0xA: ECDSA-256	4 bits
MRK/CRK/TCRK	Bootloader authentication keys	N/A	ECDSA-256	64

Detection of a packet without a valid signature may disable the SCPBL and prevents code execution. This condition clears when all power supply and battery inputs, including V_{RTC}, are taken to 0V and then returned to valid levels. The device then resets and operates normally. Removing the power supply and battery inputs clears all battery-backed state information and all SRAM, including volatile AES keys. Still, it preserves the nonvolatile SCPBL certificate, configuration, and flash program memory.

22.3 Secure Boot

Following any reset, the device executes a secure boot from the ROM to verify the integrity of the code in flash memory. The secure boot establishes the chain of trust essential for a secure application.

- The platform knows the software has not been accidentally or maliciously modified.
- The platform knows the software is from a trusted, authorized source.
- The software ensures it is running on a trusted platform.

A device with the secure boot feature must be initialized through the SCPBL before it executes the software:

- The application image must be created and signed with the CRK.
- The CRK must be loaded through the SCPBL.
- The application image must be loaded through the SCPBL.

Before executing any code in flash memory, calculate the digital signature of the application image in the flash memory using the loaded CRK. If the calculated value matches the digital signature stored at the end of the application image, the device begins the execution of the software.

The device does not execute software if the calculated value does not match the digital signature.

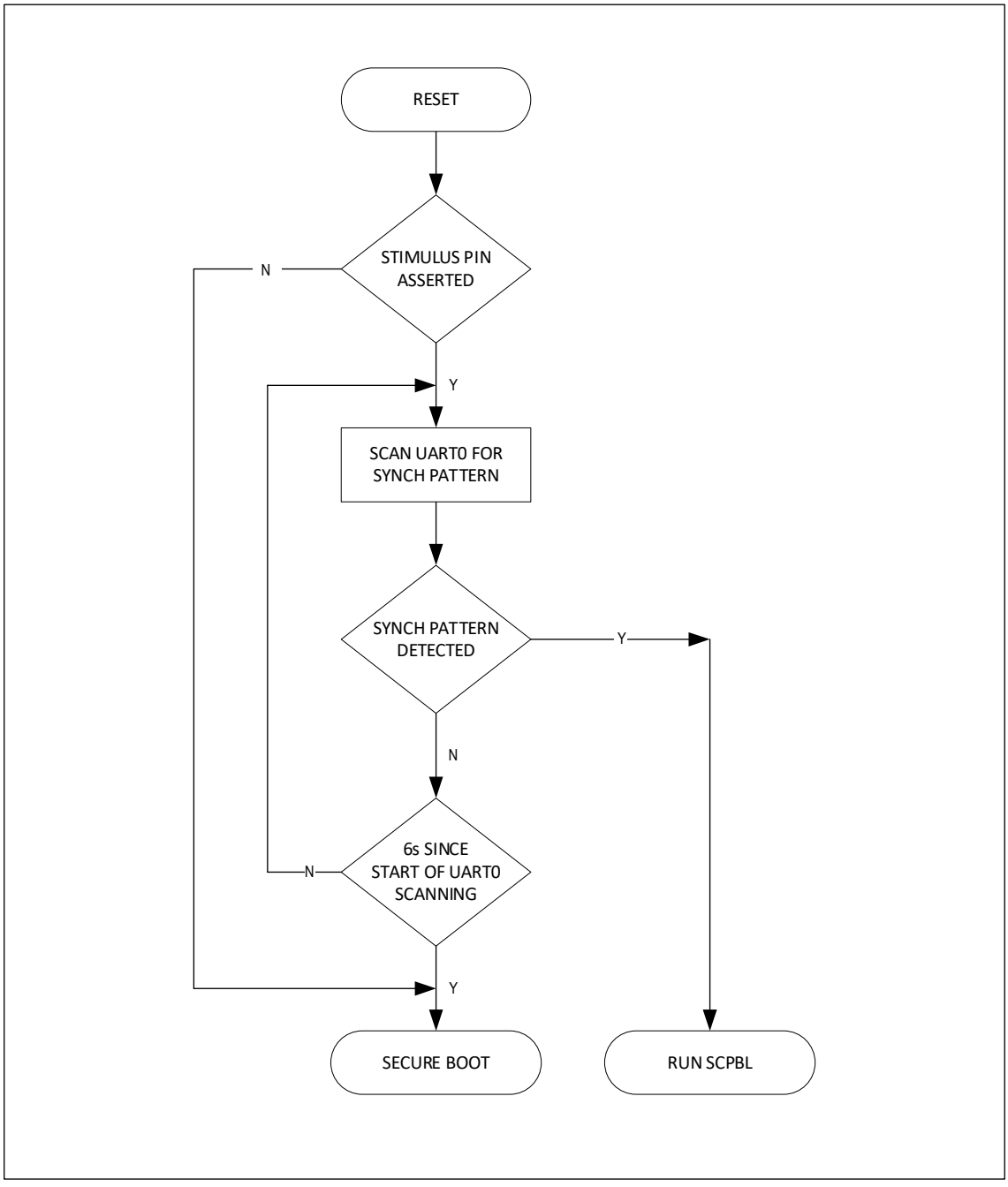
22.4 MAX32662 Bootloader Activation

Two conditions are required following a reset or wake-up from any low-power mode except SLEEP and DEEPSLEEP to activate the bootloader:

- Detection of the stimulus, as described in *Figure 22-1.*
- Detection of the SYNCH pattern on the active interface.

If a stimulus condition is present, the device searches the active interface for the synchronization pattern, as shown in *Figure 22-1*. The device executes the secure boot process if the stimulus is not active or the pattern is not seen.

Figure 22-1: MAX32662 Bootloader Flow



The device tests for the stimulus following each reset or wake-up event unless specified otherwise. It may test for the synchronization pattern multiple times during the timeout period. Once an SCPBL session begins, the stimulus pin must remain asserted until the host terminates the session.

Most devices can permanently reassign the stimulus pin to another GPIO once an SCPBL session is established. This allows access to the alternate functions of the default stimulus pin, if desired.

Following most stimulus events, some GPIO default to a high-impedance input. If a GPIO pin is the stimulus, it must be actively driven high or low by the hardware to the desired state before exiting any wake-up event or reset, including a POR.

Activation of the bootloader may overwrite some or all portions of SRAM. SRAM contents loaded before activation of the bootloader are not guaranteed to be preserved during or after SCPBL operation.

22.5 Root Key Management

The Root of Trust is a public-key infrastructure involving several key pairs.

- The manufacturer root key (MRK) pair.
- The customer root key (CRK) pair.
- The test customer rook key (TCRK) pair.

22.5.1 Manufacturer Root Key (MRK)

The factory owns the MRK pair. The public key is stored in the device and authenticates the CRK. The private key is used to sign the CRK. This certificate is used to authenticate and identify the source.

The MRK private key is stored in a restricted-access hardware security module (HSM) compliant with most security requirements.

22.5.2 Customer Root Key (CRK)

The CRK is used to sign production-ready end products. The customer generates their own CRK keypair, typically using PCI/PTS-compliant tools. The public key used for the digital signature is securely sent to the factory and is signed by the MRK. Refer to Application Note 7494 Secure Information Exchange and CRK Certification Guide for details of the key exchange and signing procedures.

The customer must protect the private part of its key pair. The chain of trust cannot be guaranteed if this key is compromised. The customer should use an HSM or equivalent hardware device to provide physical protection to the ECDSA and strong key protection requirements.

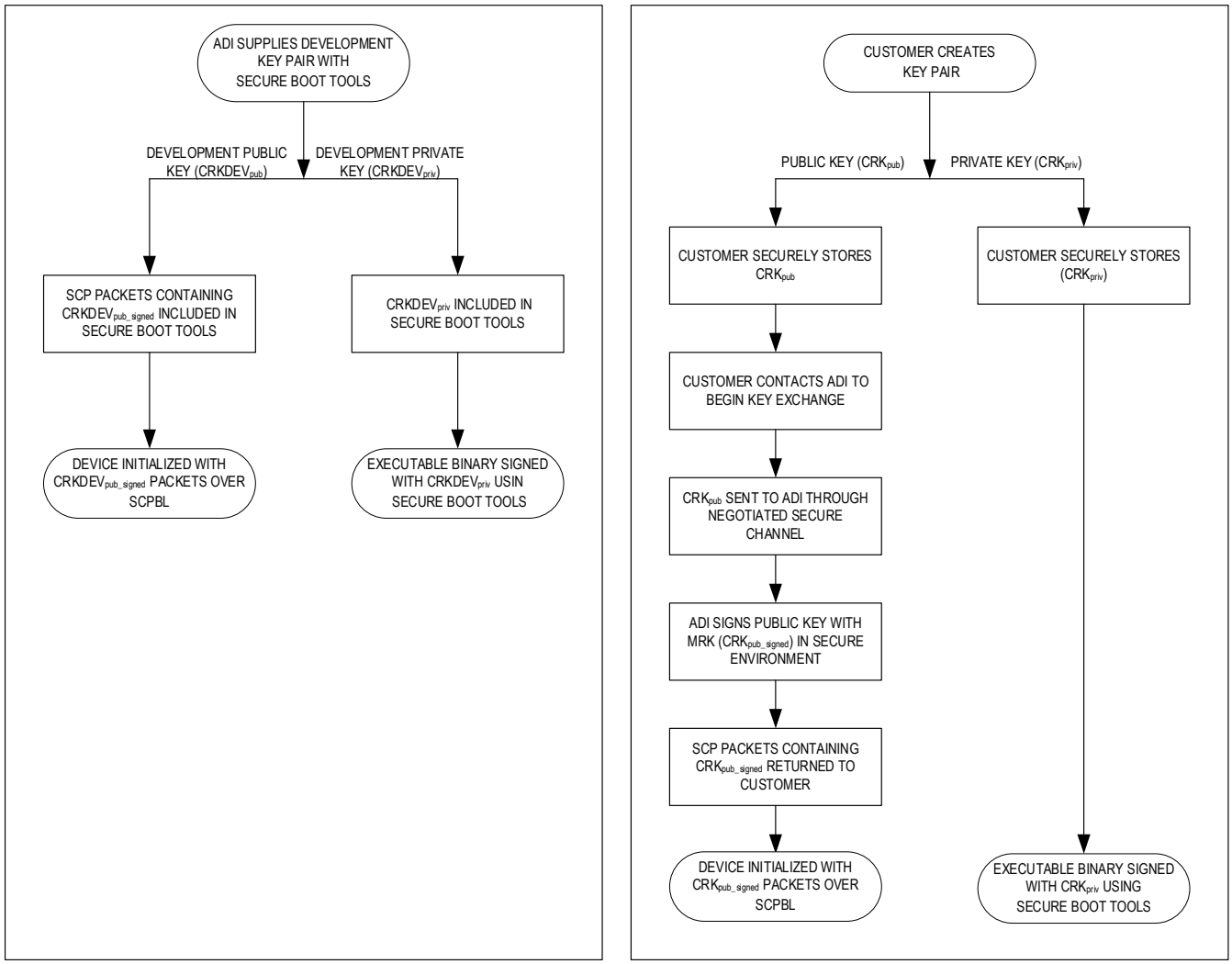
22.5.3 Test CRK (TCRK)

The SDK provides a particular version of the CRK, called the test root key, which is used during development. The use of this key eliminates the need to generate a custom CRK and have it signed by the manufacturer during the development phase. The development key is common to all customers and is not secure. A customer must load devices with a signed CRK before deployment to ensure the security of the end-customer product.

The evaluation kits for the device come preloaded with the test root key.

Figure 22-2 shows a high-level overview of the key generation and development described in the application note.

Figure 22-2: Customer Root and Development Key Generation and Usage



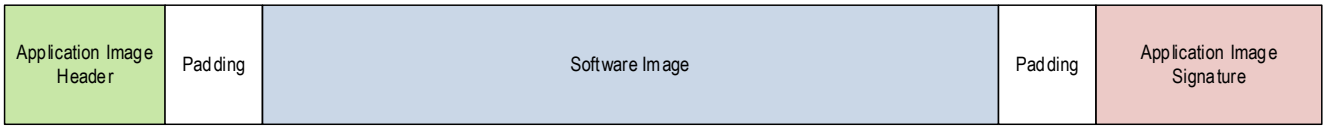
22.6 Building the Application Image

The application image is physically programmed into the program memory, regardless of the transmission protocol. The SBT provides applications to break the application image into smaller "chunks" and converts them to SCP packets.

The application image shown in *Figure 22-3* contains multiple parts:

- The application image header.
- The software image that is the compiled customer code.
- A digital signature to verify the integrity and authenticate the application image header and the software image.
- Padding as needed to ensure the start and end of the software image are aligned with the required boundaries.

Figure 22-3: Application Image Structure



The elements of the image are shown graphically in *Table 22-3*.

Table 22-3: Application Image Structure

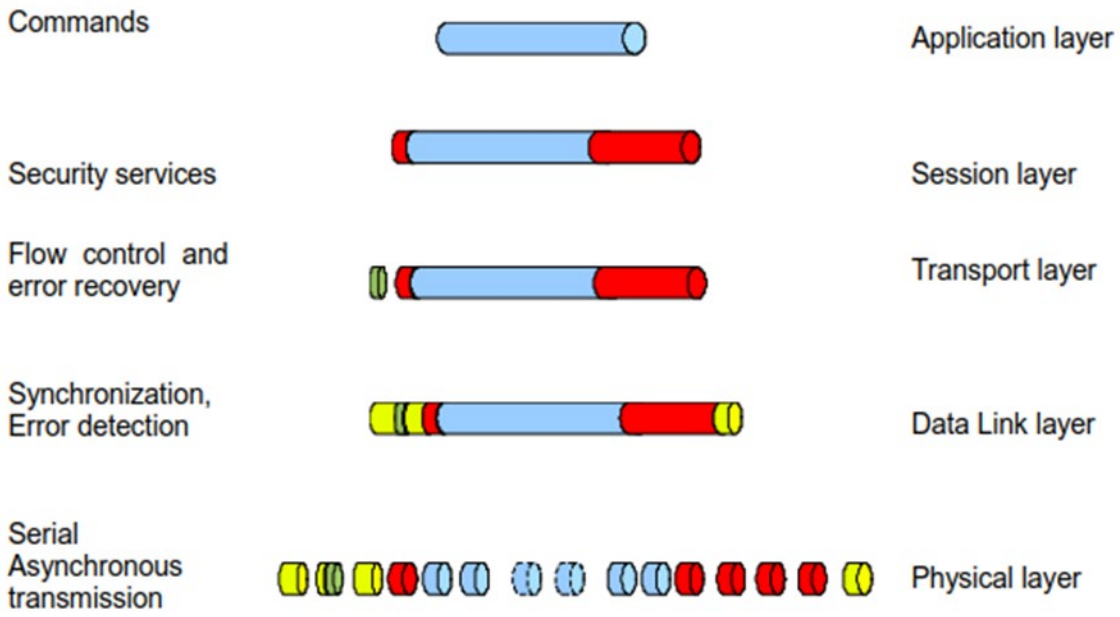
Element	Absolute Start Address Within Application Image	Length (Bytes)	Value
Header ID	0x0000 0000	8	See *Table 22-2*.
Format Version (ROM_REF)	0x0000 0008	4	0x0100 0000 (default defined by device version)
Application Image Start Address	0x0000 000C	4	0x1000 0000
Application Image Signature Offset	0x0000 0010	4	This is the start address of the digital signature located at the end of the software image plus any terminal padding (if necessary).
Executable Start Address	0x0000 0014	4	0x0000 0200 This is a pointer to the first instruction when code execution begins.
Argument Size	0x0000 0018	4	0x0000 0000
Application Version	0x0000 001C	4	User-defined (default convention used by the Secure Boot Tools is 0x0100 0000 corresponding to Version 1.0.0)
Padding	0x0000 0020	480	Padded with 0xFF out to 0x0000 01FF
Software Image	0x0000 0200	Variable	Software image
Terminal padding to 4B boundary, if necessary	End of the software image	0 1 2 3	N/A 0xFF 0xFFFF 0xFFFFFF
Application Image Signature	End of software image + terminal padding	256 bits	ECDSA-256 signature

22.7 SCP Session

The SCP is a session-based protocol that securely exchanges data packets between the host and SCPBL.

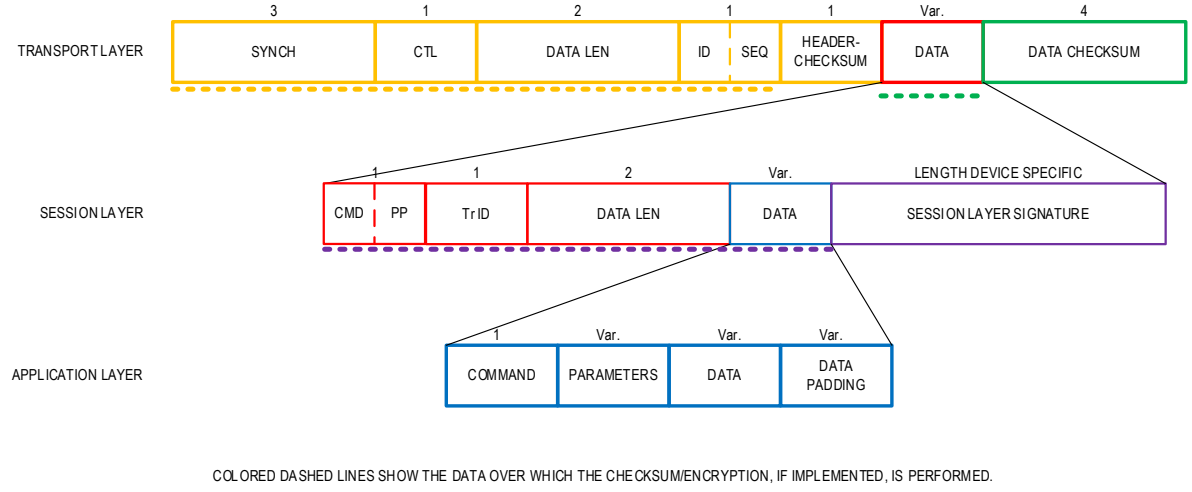
The SCP consists of a stack of layers based on the OSI model shown in *Figure 22-4*. The application layer contains basic commands to configure the SCPBL. It also includes the signed customer application and a set of WRITE data commands used for the SCPBL, including the write command that loads the code (the executable software) into the program memory.

Figure 22-4: SCPBL Implementation of OSI Model



The general structure of a packet is shown in *Figure 22-5*. Simpler commands only require a subset of these elements. Multiple integrity and authentication steps are used throughout an SCP packet.

Figure 22-5: SCP Packet Structure



22.7.1 Physical Layer

At this layer, the data between the SCPBL and the host is transmitted over the active communication interface. The stream of data is subdivided into Protocol Data Units (PDU) called bytes. When a framing error occurs, the data byte is discarded.

The UART interface is fixed at the 8-N-1 format operating at 115,200 bps. Hardware flow control is not implemented.

22.7.2 Data Link Layer

The Data Link Layer accumulates bytes from the Physical Layer into a Protocol Data Unit (PDU) called a frame (logical, structured packets for data).

Each frame contains a header identifying the type of packet and is shown in yellow in *Figure 22-5*, followed by an optional data portion.

Table 22-4: Transport/Session Layer Header Structure

Segment	Offset	Length (Bytes)	Value
Synchronization Pattern (SYNCH)	0x0000 0000	3	0xBEEFED
Segment type (CTL)	0x0000 0003	1	Transport layer packet type
Data Length (DATA LEN)	0x0000 0004	1	Data link segment length
Channel ID (ID)	0x0000 0005	Upper 4 bits	The "channel identifier" is provided by the host to identify an active connection and must be the same for SCPBL and host segments. This "channel identifier" is fixed for the whole session.
Sequence Number (SEQ)		Lower 4 bits	The lower 4 bits of this byte serve as an acknowledgment from the SCPBL that the last segment is received as described in the appropriate section. This field is incremented by one modulo 16 by the sender for each new data segment (a segment represents each data exchange one way, i.e., an ACK is not a new segment). The "sequence number" initial value is 0, used for the first data exchange after opening the session (i.e., after the HELLO-REPLY).
Header Checksum (HEADER CHECKSUM)	0x0000 0006	1	The calculation of the header checksum is described in *Table 22-2.*

22.7.3 Transport Layer

The Transport Layer provides security, data recovery, and flow control. To allow reliable communication, it establishes, manages, and terminates connections with clear phases of link establishment, information transfer, and link termination. This layer implements a reliable message service through several mechanisms:

- A sequential numbering of the segments
- A positive acknowledgment (ACK command) of command receipt
- A response timeout limit of approximately 10 seconds between the SYNCH pattern of one packet and the next. The SCPBL session is terminated if the timeout expires.

22.7.4 Session Layer

The Session Layer manages security issues by encrypting and signing the data. At this layer, a Protocol Data Unit (PDU) is called Data.

The Data has a variable length.

22.8 Sequence and Transaction ID

The SCP incorporates two indexing schemes as an error-detection mechanism:

- The session ID in the header of the transport layer
- Transaction ID in the header of the session layer

The SCPBL automatically increments its internal counter following the ACK of certain data transfer commands. The host software, following the same rules, must increment its internal counter at the same time to remain synchronized. An

unexpected sequence number indicates an error in the SCP protocol, and the SCPBL terminates the correction to prevent the communication of corrupted or compromised data.

The transaction ID is incremented after each successful data transfer at the session level from the host to the SCPBL.

These incrementing of the SEQ and TrID values are summarized in *Table 22-6*.

22.9 Opening an SCP Session Example

The host initiates a new session request by sending a COM_REQ to the SCPBL. The procedure is shown in *Table 22-5*. An error is generated if the host attempts to start a session with the CON_REQ command when a session is already in progress.

Table 22-5: Session Opening Protocol

Host		SCPBL	Session Sequence	Transaction ID
Session Closed				
Connection Request			0	N/A
	CON_REQ ->		0	N/A
		Connection Accepted	0	N/A
	<- CON_REP		0	N/A
Valid Command			0	N/A
	ACK - >		0	N/A
Connection Confirmed			0	N/A
HELLO			0	N/A
	HELLO ->		0	N/A
		Valid Command	0	N/A
	<- ACK		0	N/A
			1	N/A
		HELLO Accepted SEQ = 1	1	N/A
	<- HELLO_REPLY		1	N/A
Valid Command			1	N/A
	ACK - >		2	N/A
Session Open				

22.10 SCPBL Command Summary

Table 22-6: SCPBL Command and Sequencing Summary

Command	CTL	CMD	Command Value	Command Generates an ACK Response	SEQ increments After Command ACK	TrID increments After COMMAND_RSP ACK	Description
Transport Layer Commands							
CON_REQ	0x01	N/A	N/A	No	-	-	Connection request from the host
CON_REP	0x02	N/A	N/A	Yes	No	No	Connection acceptance from the SCPBL

Command	CTL	CMD	Command Value	Command Generates an ACK Response	SEQ increments After Command ACK	TrID increments After COMMAND_RSP ACK	Description
DISC_REQ	0x03	N/A	N/A	No	-	-	Disconnection request from the host
DISC_REP	0x04	N/A	N/A	No	-	-	Disconnection acceptance from the SCPBL
ACK	0x06	N/A	N/A	n/a	-	-	Command acknowledgment between the host and SCPBL.
Data Transfer Status Commands							
HELLO	0x05	0x1	N/A	Yes	Yes	No	Status request from the host
HELLO_REPLY	0x05	0x2	N/A	Yes	Yes	No	Status reply from the SCPBL
COMMAND_RSP	0x05	0x5*	N/A	Yes	n/a	n/a	Command success/failure response
Data Transfer Application Layer Commands							
WRITE_CRK	0x05	0x5	0x470A	Yes	Yes	Yes	Write CRK to Device
REWRITE_CRK/RENEW_CRK	0x05	0x5	0x461A	Yes	Yes	Yes	Write a Second Write CRK to the Device
WRITE_OTP	0x05	0x5	0x4714	Yes	Yes	Yes	Write to OTP
WRITE_TIMEOUT	0x05	0x5	0x4426	Yes	Yes	Yes	Set SCPBL Activation Timeout Interval
WRITE_PARAMS	0x05	0x5	0x4427	Yes	Yes	Yes	Write Parameters to Configure Communication Link
WRITE_STIM	0x05	0x5	0x4428	Yes	Yes	Yes	Configure SCPBL Stimulus Pin
WRITE_DEACTIVATE	0x05	0x5	0x4429	Yes	Yes	Yes	Permanently Deactivate SCPBL Interface
WRITE_DATA	0x05	0x5	0x2402	Yes	Yes	Yes	Write Data to Memory
COMPARE_DATA	0x05	0x5	0x2403	Yes	Yes	Yes	Compare Data Against Internal Memory
ERASE_DATA	0x05	0x5	0x4401	Yes	Yes	Yes	Erase the Range of Flash Memory
EXECUTE_CODE	0x05	0x5	0x2101	Yes	Yes	Yes	Execute Code from Flash Memory

** These commands have the same CTL and CMD number but can be distinguished by context.*

22.11 Transport Layer Command Details

22.11.1 CON_REQ

The host sends a CON_REQ to the SCPBL to initiate a connection.

The sequence number is always 0b0000 for this command.

The session ID encoded in this command becomes the session ID for all SCP transactions.

Figure 22-6: CON_REQ Command Structure

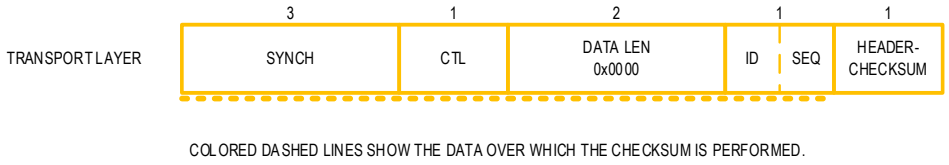


Table 22-7: CON_REQ

CON_REQ	0x01	Connection Request	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x01
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	User-defined. The channel ID value used for this command is used by all subsequent commands in the session.
SEQ		Lower 4 bits	0x0
HEADER CHECKSUM	0x07	1	Header checksum

22.11.2 CON_REP

The SCPBL sends a confirmation to the host in response to a CON_REQ command.

The host responds with an ACK. The SEQ field is not incremented by the ACK response to this command.

The sequence number is always 0b0000 for this command.

Figure 22-7: CON_REP Command Structure

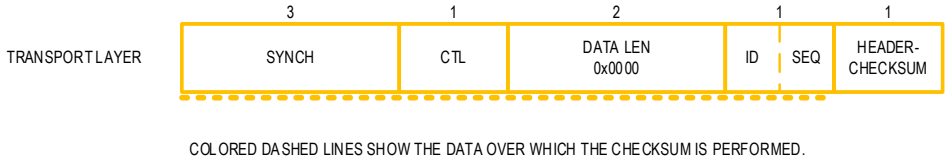


Table 22-8: CON_REP

CON_REQ	0x02	Connection Reply	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x02
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	0x0
HEADER CHECKSUM	0x07	1	Header checksum

22.11.3 DISC_REQ

The host sends a DISC_REQ to the SCPBL to terminate the SCP session. The SCPBL responds with an ACK.

The sequence number is not changed by this command.

Figure 22-8: DISC_REQ Command Structure

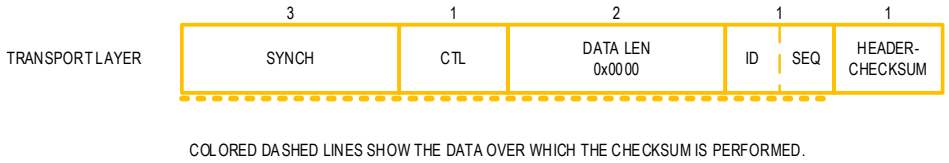


Table 22-9: DISC_REQ

DISC_REQ	0x03	Disconnection Request	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x03
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	Varies
HEADER CHECKSUM	0x07	1	Header checksum

22.11.4 DISC_REP

The SCPBL sends a confirmation response to the host in response to a DISC_REQ command. The host responds with an ACK. The bootloader session terminates, and the device performs a reset. The device reenters the SCPBL if the stimulus conditions are still present.

The sequence number is not changed by this command.

Figure 22-9: DISC_REP Command Structure

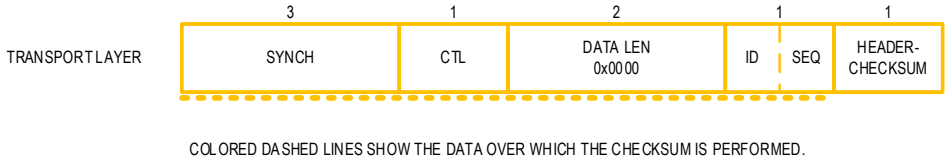


Table 22-10: DISC_REP

DISC_REQ	0x04	Disconnection Reply	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x04
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	Same as the preceding *DISC_REQ*.
HEADER CHECKSUM	0x07	1	Header checksum

22.11.5 ACK

The SCPBL and host each send out an acknowledgment packet to confirm the receipt of any valid command. The ACK command has the same ID and sequence number as the command it is acknowledging. The sequence number is incremented following the ACK command for the next command.

Figure 22-10: ACK Command Structure

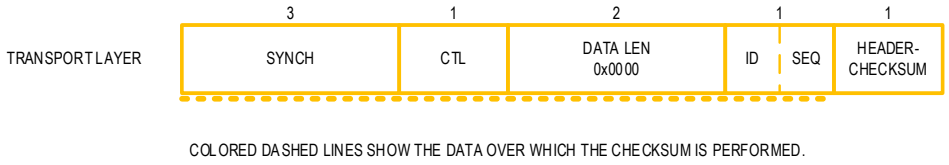


Table 22-11: ACK

ACK	0x06	Command Acknowledgement	
Command Format			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x06
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	Same as the sequence number of the preceding command
HEADER CHECKSUM	0x07	1	Header checksum

22.11.6 HELLO

The HELLO command is sent by the host to the SCPBL as part of the sequence to establish a new session. The command is a unique version of the DATA_TRANSFER command with a fixed payload. The SCPBL responds with an ACK followed by HELLO_REPLY.

The HELLO command does not include a signature following the session layer data like other DATA_TRANSFER commands.

Figure 22-11: HELLO Command Structure

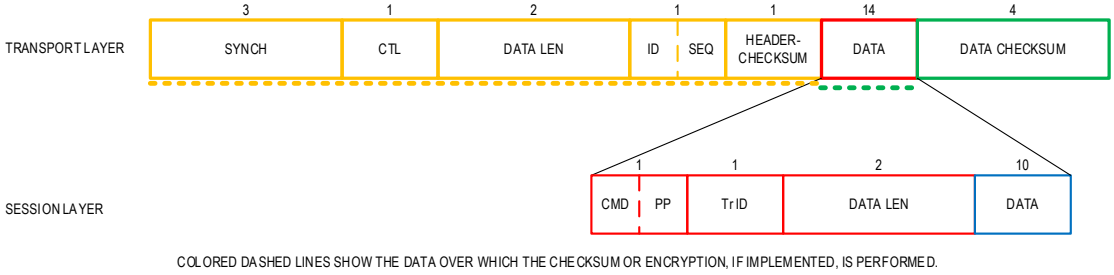


Table 22-12: HELLO Command

HELLO	0x05	Request Configuration Information	
Command Format (Transport Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x05
DATA LEN	0x04	2	0x000E
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	Same as the sequence number of the preceding command
HEADER CHECKSUM	0x07	1	Header checksum
Command Format (Session Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
Command	0x00	4 bits	0x1
PP		4 bits	0b0000 (there is no session layer signature for this command.)
TrID	0x01	1	Variable
DATA LEN (SESSION)	0x02	2	0x000A
DATA	0x04	10	0x00: 'H' 0x01: 'E' 0x02: 'L' 0x03: 'L' 0x04: 'O' 0x05: 0x20 0x06: 'B' 0x07: 'L' 0x08: 0x03 0x09: 0x02

22.11.7 HELLO_REPLY

A HELLO_REPLY is sent by the SCPBL to the host in response to the HELLO command. The command provides information about the device configuration, including:

- ROM version
- JTAG status
- USN

Figure 22-12: HELLO_REPLY Structure

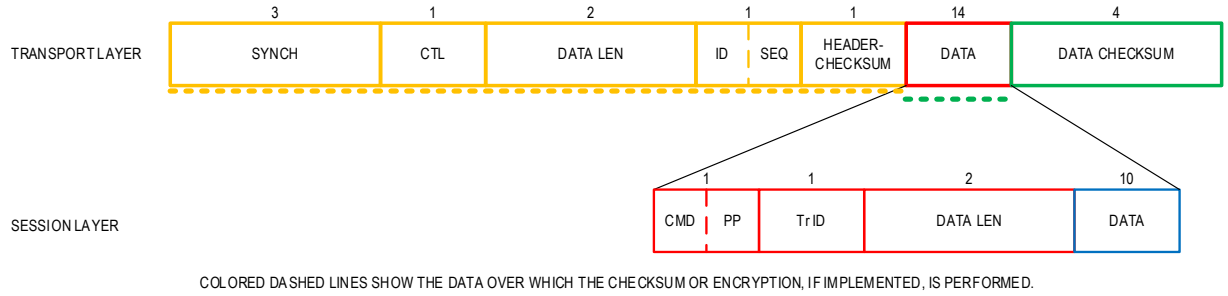


Table 22-13: HELLO_REPLY Command

HELLO_REPLY	0x05	Return Configuration Information	
Command Format (Transport Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x05
DATA LEN	0x04	2	0x0036
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	Varies
HEADER CHECKSUM	0x07	1	Header checksum
Command Format (Session Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
CMD	0x00	Upper 4 bits	0x2
PP		Lower 4 bits	0b0000 (there is no session layer signature for this command.)
TrID	0x01	1	Variable
DATA LEN (SESSION)	0x02	2	0x0032
DATA	0x04	10	0x00 – 0x9: 0x72, 0x69, 0x76, 0x76, 0x79, 0x32, 0x72, 0x79, 0x83, 0x84 0x0A - 0x0D: ROM Version 0x0E: Lifecycle 0x0F: 0x00 0x10: 0x00 0x11: Configuration 0b0xxxxxxx: JTAG Enabled 0b1xxxxxxx: JTAG Disabled 0bxxx1xxxx: No CRK loaded 0bxxx0xxxx: CRK loaded 0x12 - 0x1F: USN 0x20 - 0x31: 0x00

22.11.8 DATA TRANSFER COMMANDS

This type of command is how the application layer commands (which include the loading of program memory) are passed through the SCP.

Most of these commands implement the session-level signature to verify and authenticate the host.

Figure 22-13: DATA TRANSFER Command Structure

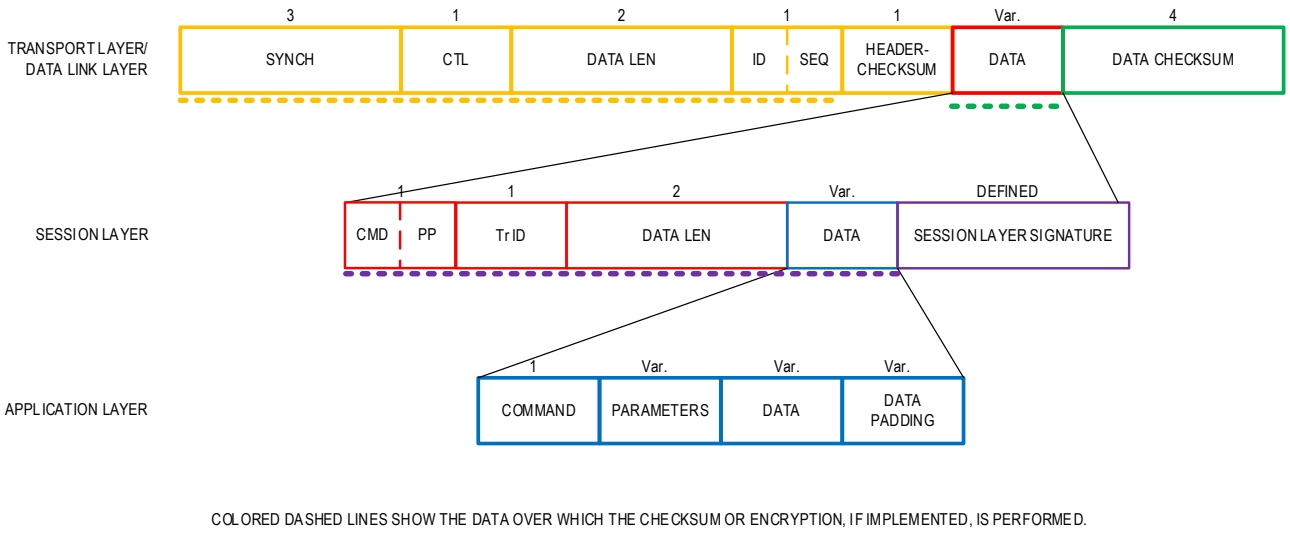


Table 22-14: DATA TRANSFER Command Example

WRITE_DATA	0x06	Write Data To Memory or Configuration Command	
Command Format (Transport Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x05
DATA LEN	0x04	2	0x0000
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	Same as the sequence number of the preceding command
HEADER CHECKSUM	0x07	1	Header checksum
Command Format (Session Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
CMD	0x00	Upper 4 bits	0x5
PP		Lower 4 bits	See *Table 22-2* for the protection profile value.
TrID	0x01	1	Varies
DATA LEN (SESSION)	0x02	2	User-defined
DATA	0x04	10	User-defined
SESSION LAYER SIGNATURE	0x0E	1	Defined by the device.

22.11.9 COMMAND_RSP

Most application layer commands generate a 4-byte response indicating the success or failure of the command. This packet is sent from the SCPBL to the host after the SCPBL ACKs the command and after the application layer command is complete.

A response value of 0x0000 indicates the successful execution of the command. Any other value indicates an error, which may or may not be specific to the command.

This is a DATA TRANSFER command.

Figure 22-14: COMMAND_RSP Command Structure

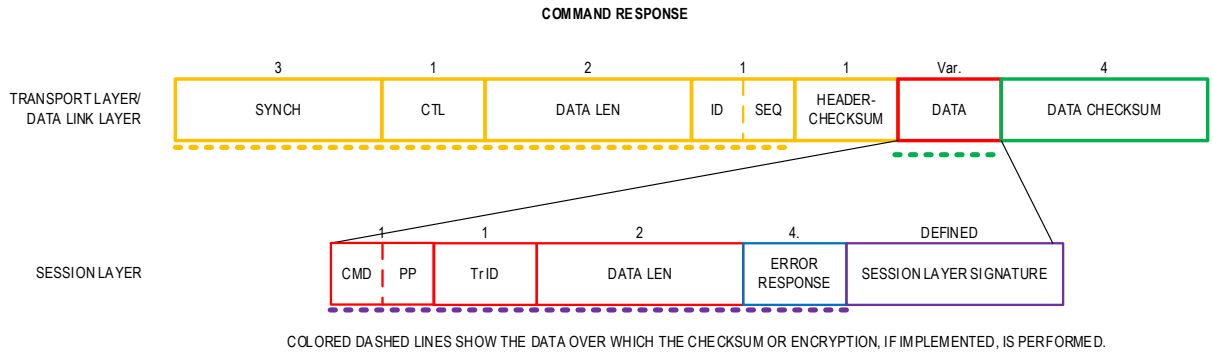


Table 22-15: COMMAND_RSP

COMMAND_RSP	0x05	Success/Failure Response from Application Layer Command	
Command Format (Transport Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
SYNCH	0x00	3	0xBEEFED
CTL	0x03	1	0x05
DATA LEN	0x04	2	0x0008
ID	0x06	Upper 4 bits	Same as the channel ID of the CON_REQ command.
SEQ		Lower 4 bits	Same as the sequence number of the preceding command.
HEADER CHECKSUM	0x07	1	Header checksum
Command Format (Session Layer)			
Element	**Offset**	**Length (Bytes)**	**Values**
CMD	0x00	Upper 4 bits	0x5
PP		Lower 4 bits	See *Table 22-2* for the protection profile value.
TrID	0x01	1	Varies
DATA LEN (SESSION)	0x02	2	0x0004
RESPONSE	0x04	4	0x00000000: Success Any other value: Failure

22.12 Application Layer Command Details

The application layer deals with the commands used to directly modify the memory and configure various functions. The application layer is shown in *Figure 22-5* and is shown in blue.

The memory commands WRITE_DATA, ERASE_DATA, and COMPARE_DATA directly address the internal flash and RAM based on the target address.

The execute command is used indirectly as part of the secondary-level applets that perform WRITE DATA, ERASE DATA, and COMPARE DATA commands to the external memory. This makes the memory commands fully flexible so that these commands can operate on any external memory within the security context of the SCP framework.

Other application layer commands are associated with device configuration.

22.12.1 WRITE_CRK

Table 22-16: WRITE_CRK

WRITE_CRK	0x4701	Write CRK to the Device	
Description	This command writes the CRK to the nonvolatile memory of the SCPBL. This command can only be executed once per device. Use the REWRITE_CRK command to write a second CRK, allowing a maximum of two CRK values. The CRK value is the same as the MRK until this command is executed. The new CRK value is used when the current session terminates, and a new one is started.		
Command Format			
Element	**Start Address**	**Length**	**Values**
CMD	0x0000	2	0x4701
Key Length	0x0002	2	0x0204
CRK	0x0004	64 Bytes	Public Key (ECDSA-256)
MRK Signature	0x0044	64 Bytes	MRK Signature of the CRK using the public key (ECDSA-256)

22.12.2 REWRITE_CRK/RENEW_CRK

Table 22-17: REWRITE_CRK/RENEW_CRK

REWRITE_CRK RENEW_CRK	0x461A	Write a Second Write CRK to the Device	
Description	This command writes a second CRK to the nonvolatile memory of the SCPBL. It is the only way to change the CRK after the WRITE_CRK command. This command can only be executed once per device, so a maximum of two CRK values can be written. This command requires the previous CRK as one of the arguments to prevent accidental changing of the CRK value.		
Command Format			
Element	**Start Address**	**Length**	**Values**
CMD	0x0000	2	0x461A
Key Length	0x0002	2	0x0204
Previous CRK	0x0004	64	Previous CRK (ECDSA-256)
New CRK	0x0044	64	New CRK (ECDSA-256)
MRK Signature	0x0084	64	MRK signature (ECDSA-256)

22.12.3 WRITE_OTP

Table 22-18: WRITE_OTP

WRITE_OTP	0x4714	Write Data to OTP Memory	
Description	This reserved command may only be used when specifically instructed by the factory.		
Command Format			
Element	**Start Address**	**Length**	**Values**
CMD	0x0000	2	0x4714
Arguments	0x0002	Variable	

22.12.4 WRITE_TIMEOUT

Table 22-19: WRITE_TIMEOUT

WRITE_TIMEOUT	0x4426	Set SCPBL Activation Timeout Interval	
Description	This command specifies the amount of time in milliseconds that the device waits for a valid SYNCH pattern following the detection of an active SCPBL activation pin. If no SYNCH pattern is received within the timeout interval, then the SCPBL does not, and instead performs a secure boot. If the secure boot option is not available, the device begins program execution. The device uses the default timeout value shown in *Table 22-1* until changed with this command. This command is ignored if a device does not have a default activation pin. This command can only be executed once for each interface.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
CMD	0x0000	2	0x4426
Interface ID	0x0002	1	See *Table 22-1*.
Value	0x0003	2	This is the timeout value in milliseconds, from 0x0001 to 0xFFFE. An interval of 0x0000 03E8 corresponds to 1s. The timeout value is sent LSB first.

22.12.5 WRITE_PARAMS

Table 22-20: WRITE_PARAMS

WRITE_PARAMS	0x4427	Write Parameters to Configure Communication Link	
Description	This reserved command is for factory configuration only.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x4427
Instance ID	0x0002	1	See *Table 22-1*.
Data	0x0003	Variable	The length of the field is device-specific.

22.12.6 WRITE_STIM

Table 22-21: WRITE_STIM

<table>
<tr><th>WRITE_STIM</th><th>0x4428</th><th colspan="10">Configure SCPBL Stimulus Pin (GPIO0-GPIO3)</th></tr>
<tr><td rowspan="4">Description</td><td colspan="11">This command allows the SCPBL to permanently change the location and polarity of the SCPBL stimulus pin. Valid pin locations are shown in Table 22-1.
Regardless of specific settings, the value of this byte must not be 0xFF.
This command can only be executed once.</td></tr>
<tr><td>Bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td></td><td colspan="2">GPIO Port</td><td>Active State</td><td colspan="6">GPIO Pin of Selected Port</td></tr>
<tr><td>Function</td><td colspan="2">0b00: Port 0
0b01: Port 1
0b10: Port 2
0b11: Port 3</td><td>0: Low
1: High</td><td colspan="6">0x00: Pin 0
0x01: Pin 1
0x02: Pin 2
0x03: Pin 3
0x04: Pin 4
0x05: Pin 5
0x06: Pin 6
0x07: Pin 7
0x08: Pin 8
0x09: Pin 9
0x0A: Pin 10
0x0B: Pin 11
0x0C: Pin 12
0x0D: Pin 13
0x0E: Pin 14
0x0F: Pin 15
0x10: Pin 16
0x11: Pin 17
0x12: Pin 18
0x13: Pin 19
0x14: Pin 20
0x15: Pin 21
0x16: Pin 22
0x17: Pin 23
0x18: Pin 24
0x19: Pin 25
0x1A: Pin 26
0x1B: Pin 27
0x1C: Pin 28
0x1D: Pin 29
0x1E: Pin 30
0x1F: Pin 31</td></tr>
<tr><td colspan="12">Command Format</td></tr>
<tr><td>Segment</td><td>Start Address</td><td colspan="2">Byte Length</td><td colspan="8">Values</td></tr>
<tr><td>Command</td><td>0x0000</td><td colspan="2">2</td><td colspan="8">0x4428</td></tr>
<tr><td>Stimulus Location</td><td>0x0002</td><td colspan="2">1</td><td colspan="8">See Table 22-1 for valid pin options.</td></tr>
</table>

22.12.7 WRITE_SLA_VERSION

Table 22-22: WRITE_SLA_VERSION

WRITE_SLA_VERSION	0x470B	Set Minimum Revision Value	
Description	This command sets up the minimum required revision level in the SLA header of a command. Only commands with a revision level equal to or greater than the one set by this command is allowed to run. This command can only change the revision value six times. The HELLO_RSP command returns the last value written with this command. The HELLO_RSP command returns 0x0000000 if no value is set yet by this command.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x470B
Revision	0x0002	4	User-defined 4-byte value. (default 0x0000 0000)

22.12.8 WRITE_DEACTIVATE

Table 22-23: WRITE_DEACTIVATE

WRITE_DEACTIVATE	0x4429	Permanently Deactivate SPCPBL Interface	
Description	This command deactivates an SCPBL link. This command can be used once for each link. Once deactivated, a link cannot be reactivated.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x4429
Interface ID	0x0002	1	See *Table 22-1* for the instance ID of the desired interface

22.12.9 WRITE_DATA

Table 22-24: WRITE_DATA

WRITE_DATA	0x2402		Write Data to Memory
Description	The command writes data into the internal flash or RAM based on the target address.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x2402
Start Address	0x0002	4	This is the target start address. The MSB is sent first.
Length	0x0006	4	This is the length of the data segment in bytes. The MSB is sent first.
Data	0x000A	Variable	Write Data

22.12.10COMPARE_DATA

Table 22-25: COMPARE_DATA

COMPARE_DATA	0x2403	Compare Data Against Internal Memory	
Description	This command compares the supplied data against the internal flash or RAM contents based on the target address.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x2403
Start Address	0x0002	4	This is the target start address. The MSB is sent first.
Length	0x0006	4	This is the length of the data segment in bytes. The MSB is sent first.
Data	0x000A	Variable	Compare data

22.12.11ERASE_DATA

Table 22-26: ERASE_DATA

ERASE_DATA	0x4401	Erase the Range of Flash Memory	
Description	Erases the number of bytes specified in the Length field from flash memory, starting from the Start Address field.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x4401
Start Address	0x0002	4	This is the target start address. The MSB is sent first.
Length	0x0004	4	This is the length of the data segment in bytes. The MSB is sent first.

22.12.12EXECUTE_CODE

Table 22-27: EXECUTE_CODE

EXECUTE_CODE	0x2101	Execute Code From Flash Memory	
Description	The execute command is used indirectly as part of secondary level applets. This makes the memory commands fully flexible so that these commands can operate on any external memory within the security context of the SCP framework.		
Command Format			
Segment	**Start Address**	**Byte Length**	**Values**
Command	0x0000	2	0x2101
Start Address field	0x0002	4	This is the address to begin parsing for an application header in the target memory.

23. Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION
0	07/2022	Initial Release
1	02/2023	Added missing bits 23:22 in *MCR_ADCCFG2* register and marked them as do not modify. Marked flash controller registers as only reset on POR. Corrected *SPIn_CTRL1* register to show it as R/W and correctly describe the fields usage. Updated *Using GPIOWAKE for Wake-Up from All Power Modes* with additional steps for wake up. Updated *Table 4-1* with correct frequencies for f_{ERFO} and $f_{HF\_EXT\_CLK}$. Added instructions to the *100MHz Internal Primary Oscillator (IPO)* section to set the IPO as the system oscillator. Added *IPO Calibration* section. Updated field descriptions for *FCR_AUTOCAL0*, *FCR_AUTOCAL1*, and *FCR_AUTOCAL2* registers. Updated SPI *Serial Peripheral Interface (SPI)* chapter to use target instead of peripheral to be consistent with other chapters. Added details on *AES Key Generation* and *AES Key Storage*. Updated *Encryption of 128-Bit Blocks of Data Using FIFO*, *Encryption and Decryption of 128-Bit Blocks Using DMA*, and *Encryption of Blocks Less Than 128-Bits*. Updated *MCR_CLKCTRL*, *GCR_PM*, *GCR_PCLKDIS0*, *GCR_PCLKDIS1*, and *GCR_RST0*. Added missing pulse train registers (*PTG_READY_INTFL* and *PTG_READY_INTEN*).