MAX78000 USER GUIDE

UGXXXX; Rev 0.1; 05/20/2020

Abstract: This user guide provides application developers information on how to use the memory and peripherals of the MAX78000 microcontroller. Detailed information for all registers and fields in the device are covered. Guidance is given for managing all the peripherals, clocks, power and startup for the device family.

MAX78000 User Guide

Table of Contents

List of Figures

List of Tables

1. Overview

The MAX78000 is a new breed of low-power microcontrollers built to thrive in the rapidly evolving AI at the edge market. These products include Maxim's proven ultra-low power MCU IP along with deep neural network AI acceleration.

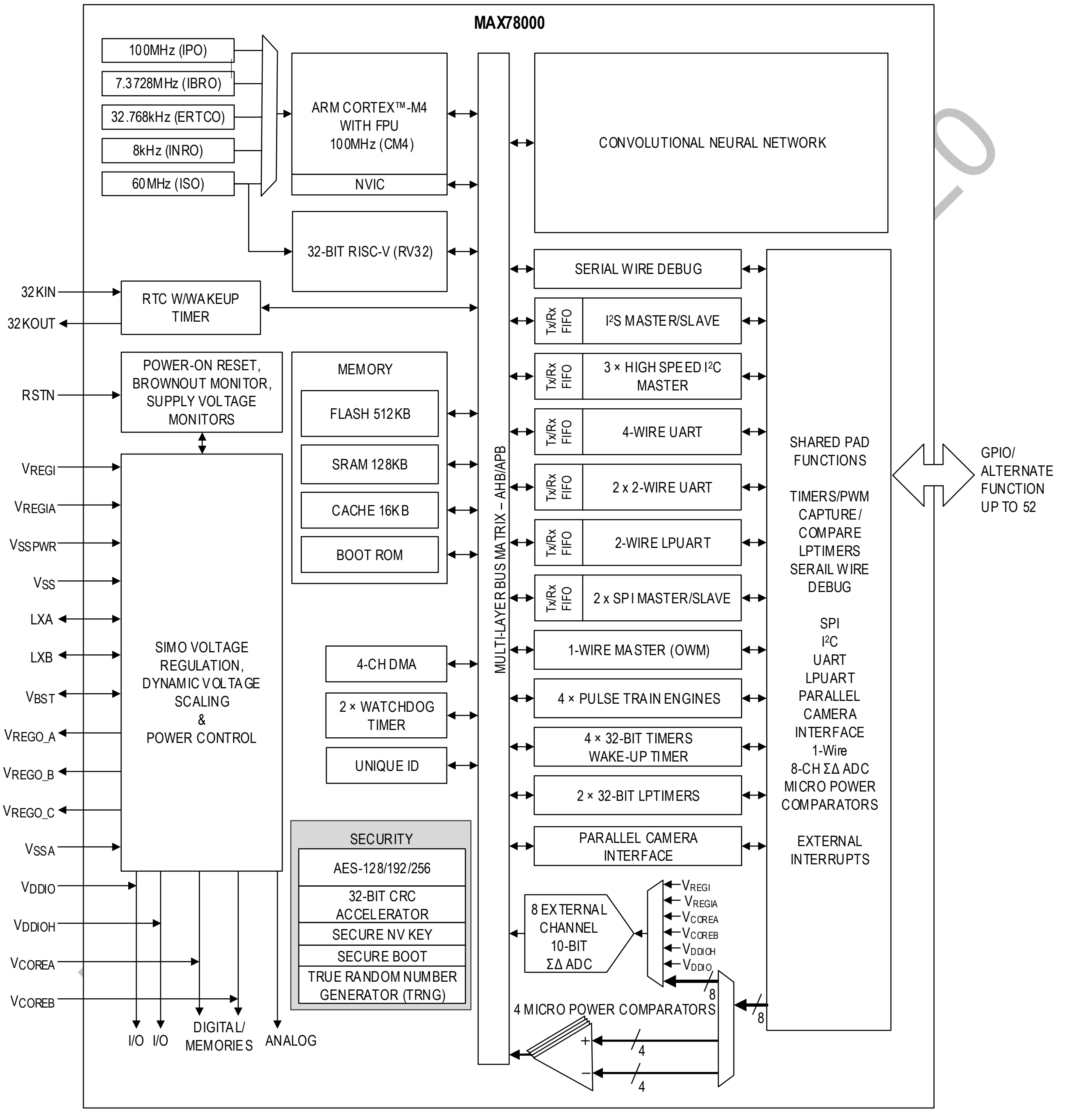
The MAX78000 is an advanced system-on-chip featuring an Arm® Cortex®-M4 with FPU CPU for efficient computation of complex functions and algorithms with integrated power management. It also includes 442KB weight CNN accelerator. The devices offer large on-chip memory with 512KB flash and up to 128KB SRAM. Multiple high-speed and low-power communications interfaces are supported including high-speed SPI, I2C serial interface, and LPUART. Additional low-power peripherals include flexible LPTIMER and analog comparators.

The device is available in 81-CTBGA, 8mm × 8mm, 0.8mm pitch.

The high-level block diagram for the MAX78000 is shown in *Figure 1-1*.

1.1 Block Diagram

Figure 1-1: MAX78000 Block Diagram



2. Memory, Register Mapping, and Access

2.1 Memory, Register Mapping, and Access Overview

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

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

Figure 2-1: Code Memory Mapping

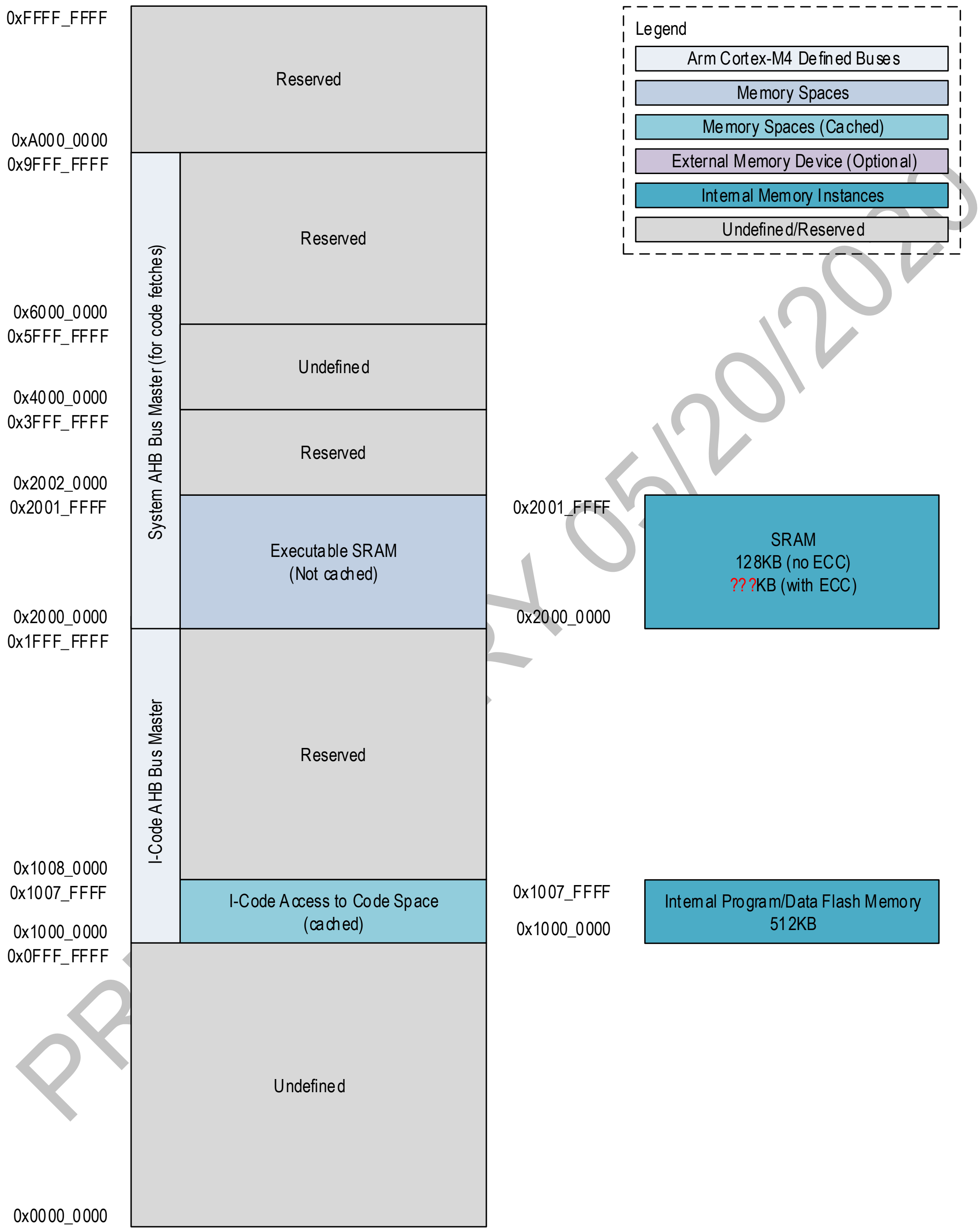
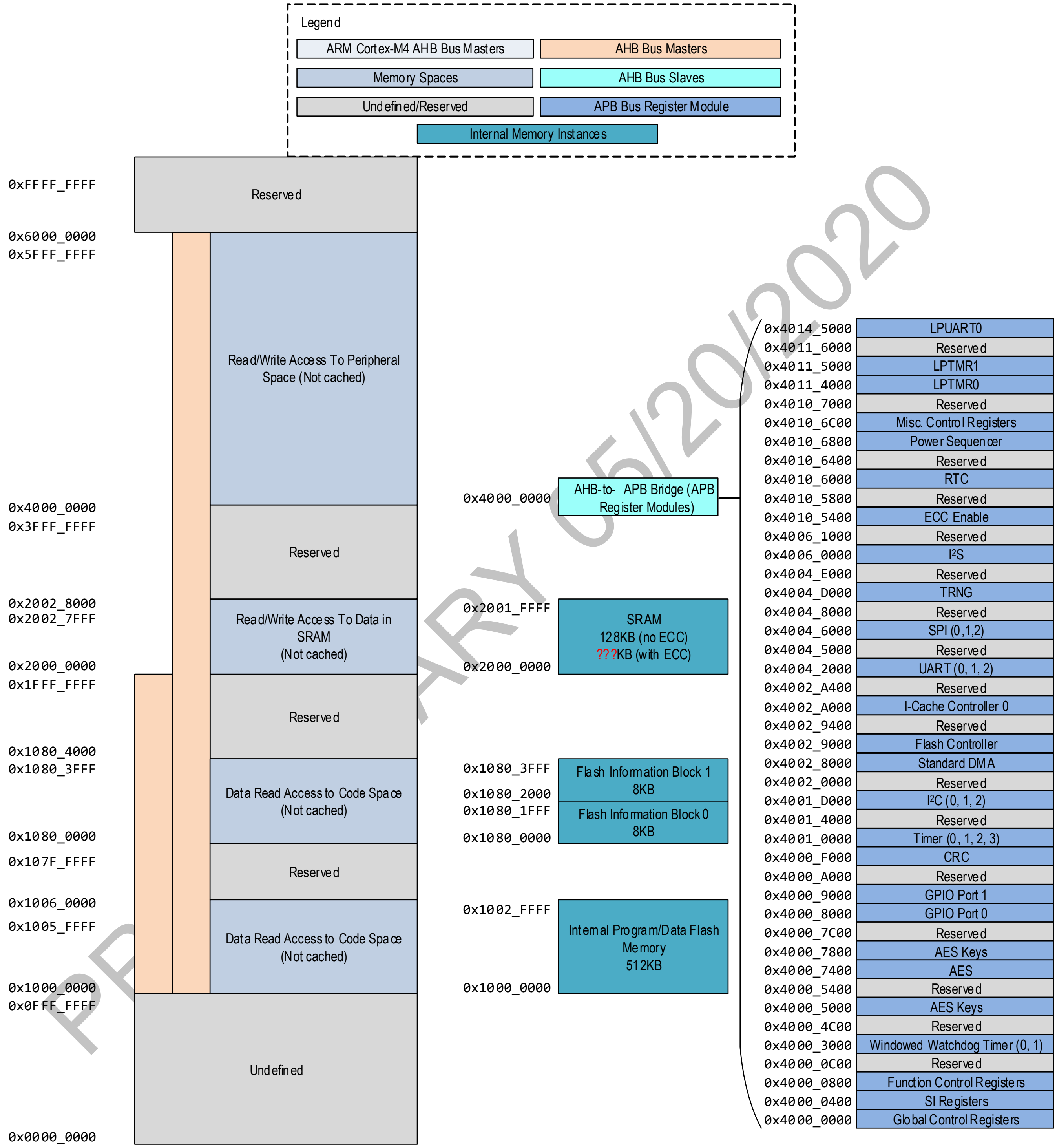


Figure 2-2: Data Memory Mapping



2.2 Field Access Definitions

All the fields that are accessible by user software have distinct access capabilities. Each register table contained in this user guide has an access type defined for each field. The definition of each field access type is presented in *Table 2-1. Field Access Definitions*.

Table 2-1. Field Access Definitions

Access Type	Definition
RO	**Reserved** This access type is reserved for static fields. Reads of this field will return the reset value. Writes are ignored.
DNM	**Reserved. Do Not Modify** Reads of this field will return indeterminate values. Software must first read this field and write the same value whenever writing to this register.
R	**Read Only** Reads of this field will return a value. Writes to the field have no effect on device operation.
W	**Write Only** Reads of this field will return indeterminate values. Writes to the field may change the field's state and may affect device operation.
R/W	**Unrestricted Read/Write** Reads of this field will return a value. Writes to the field may change the field's state and may affect device operation.
RC	**Read-to-Clear** Reads of this field may return a value. Any read of this register will clear the field to 0. Writes to the field have no effect on device operation.
RS	**Read-to-Set** Reads of this field may return a value. Any read of this register will set the field to 1. Writes to the field have no effect on device operation.
R/W0	**Read-Write-0-Only** Reads of this field may return a value. Writing 0 to this field may change the field's state and may affect device operation. Writing 1 to the field has no effect on device operation.
R/W1	**Read-Write-1-Only** Reads of this field may return a value. Writing 1 to this field may change the field's state and may affect device operation. Writing 0 to the field has no effect on device operation.
R/W1C	**Read-Write-1-to-Clear** Reads of this field may return a value. Writing 1 to this field will clear this field to 0. Writing 0 to the field has no effect on device operation.
W1C	**Write-1-to-Clear** Reads of this field will return indeterminate values. Writing 1 to this field will clear this field to 0. Writing 0 to the field has no effect on device operation.
R/W0S	**Read-Write-0-to-Set** Reads of this field may return a value. Writing 0 to this field will set this field to 1. Writing 1 to the field has no effect on device operation.

2.3 Standard Memory Regions

Several standard memory regions are defined for the Arm Cortex-M4 and RISC-V architectures; the use of many of these is optional for the system integrator. At a minimum, the MAX78000 must contain some code and data memory for application code and variable/stack use for CPU0, as well as certain components which are part of the instantiated Cortex-M4 core.

2.3.1 CPU0 Code Space

The CPU0 code space area of memory is designed to contain the primary memory used for code execution by the device. This memory area is defined from byte address range 0x0000 0000 to 0x1FFF FFFF (0.5GB maximum). Two different standard core bus masters are used by the Cortex-M4 core and Arm debugger 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 MAX78000 code memory mapping is illustrated in *Figure 2-1: Code Memory Mapping*. The code space memory area contains the main internal flash memory, which holds most of the instruction code that will be executed on the device. The internal flash memory is mapped into both code and data space from 0x1000 0000 to 0x1007 FFFF.

This program memory area must also contain the default system vector table and the initial settings for all system exception handlers and interrupt handlers. The reset vector for the device is 0x0000 0000 and contains the device ROM code which transfers execution to user code at address 0x1000 0000.

The code space memory on the MAX78000 also contains the mapping for the flash information block, from 0x1080 0000 to 0x1080 3FFF. However, this mapping is generally only present during Maxim Integrated production test; it is disabled once the information block has been loaded with valid data and the info block lockout option has been set. This memory is accessible for data reads only and cannot be used for code execution. The flash information block is user read only accessible and contains the Unique Serial Number (USN).

2.3.2 Information Block Flash Memory

The information block is a separate area of the Internal Flash Memory and is 16,384 Bytes. The information block is used to store trim settings (option configuration and analog trim) as well as other nonvolatile device-specific information. The information block also contains the device's Unique Serial Number (USN). The USN is a 104-bit field. USN bits 0 thru 7 contain the die revision.

Figure 2-3. Unique Serial Number Format

<table>
<tr><th></th><th></th><th colspan="32">Bit Position</th></tr>
<tr><th></th><th></th><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="5">Address</td><td>0x10800000</td><td colspan="17">USN bits 16 - 0</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>0x10800004</td><td>x</td><td colspan="31">USN bits 47-17</td></tr>
<tr><td>0x10800008</td><td colspan="17">USN bits 64 - 48</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
<tr><td>0x1080000C</td><td>x</td><td colspan="31">USN bits 95 - 65</td></tr>
<tr><td>0x10800010</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td colspan="8">USN bits 103 - 96</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td><td>x</td></tr>
</table>

Reading the USN requires the information block to be unlocked. Unlocking the information block does not enable write access to theblock, but allows the contents of the USN to be read from the block. Unlock the information block using the following steps:

1. Write 0x3A7F 5CA3 to *FLC_ACTNL*.
2. Write 0xA1E3 4F20 to *FLC_ACTNL*.
3. Write 0x0x9608 B2C1 to *FLC_ACTNL*.
4. Information block is now read-only accessible.

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

2.3.3 SRAM Space

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

The MAX78000 data memory mapping is illustrated in *Figure 2-2: Data Memory Mapping* and the SRAM configuration is defined in *Table 2-2: SRAM* Configuration. This memory area contains the main system SRAM. The total size of the internal SRAM is 128KB. The SRAM is divided into four blocks and the contiguous address range is 0x2000 0000 to 0x2001 FFFF.

The entirety of the SRAM memory space on the MAX78000 is contained within the dedicated Arm Cortex-M4 SRAM bit-banding region from 0x2000 0000 to 0x200F FFFF (1MB maximum for bit-banding). This means that the CPU can access the entire SRAM either using standard byte/word/doubleword access or using bit-banding operations. The bit-banding mechanism allows any single bit of any given SRAM byte address location to be set, cleared, or read individually by reading from or writing to a corresponding doubleword (32-bit wide) location in the bit-banding alias area.

The alias area for the SRAM bit-banding is located beginning at 0x2200 0000 and is a total of 32MB maximum, which allows the entire 128KB bit banding area to be accessed. Each 32-bit (4 byte aligned) address location in the bit-banding alias area translates into a single bit access (read or write) in the bit-banding primary area. Reading from the location performs a single bit read, while writing either a 1 or 0 to the location performs a single bit set or clear.

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

The SRAM area on the MAX78000 can be used to contain executable code. Code stored in the SRAM is accessed directly for execution (using the system bus) and is not cached. The SRAM is also where the Arm Cortex-M4 stack must be located, as it is the only general-purpose SRAM memory on the device. A valid stack location inside the SRAM must be set by the system exception table (which is, by default, stored at the beginning of the internal flash memory).

The MAX78000 specific AHB Bus Masters can access the SRAM to use as general storage or working space.

Table 2-2: SRAM Configuration

System RAM Block #	Size	Start Address	End Address	CPU0 Accessible	CPU1 Accessible
sysram0	32KB	2000 0000	2000 7FFF	✓	✓
sysram1	32KB	2000 8000	2000 FFFF	✓	✓
sysram2	48KB	2001 0000	2001 BFFF	Optional	✓
sysram3	16KB	2001 C000	2001 FFFF	Optional	Default Boot Memory Optional ICC1

2.3.4 Peripheral Space

The peripheral space area of memory is intended for mapping of control registers, internal buffers/working space, and other features needed for the firmware control of non-core peripherals. It is defined from byte address range 0x4000 0000 to 0x5FFF FFFF (0.5GB maximum). On the MAX78000, all device-specific module registers are mapped to this memory area, as well as any local memory buffers or FIFOs which are required by modules.

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

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

On the MAX78000, access to the region that contains most peripheral registers (0x4000 0000 to 0x400F FFFF) goes from the AHB bus through an AHB-to-APB bridge. This allows the peripheral modules to operate on the lower power APB bus matrix. This also ensures that peripherals with slower response times do not tie up bandwidth on the AHB bus, which must necessarily have a faster response time since it handles main application instruction and data fetching.

2.3.5 External RAM Space

The external RAM space area of memory is intended for use in mapping off-chip external memory and is defined from byte address range 0x6000 0000 to 0x9FFF FFFF (1GB maximum). *The MAX78000 does not implement this memory area.*

2.3.6 External Device Space

The external device space area of memory is intended for use in mapping off-chip device control functions onto the AHB bus. This memory space is defined from byte address range 0xA000 0000 to 0xDFFF FFFF (1GB maximum). The MAX78000 does not implement this memory area.

2.3.7 System Area (Private Peripheral Bus)

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

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

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

2.3.8 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 MAX78000 does not implement this memory region.

2.4 Device Memory Instances

This section details physical memory instances on the MAX78000 (including internal flash memory and SRAM instances) that are accessible as standalone memory regions using either the AHB or APB bus matrix. Memory areas which are only accessible via FIFO interfaces, or memory areas consisting of only a few registers for a specific peripheral, are not covered here.

2.4.1 Main Program Flash Memory

The main program flash memory is **Error! Unknown document property name.** and consists of 64 logical pages of 8,192 Bytes per page.

2.4.2 Instruction Cache Memory

The Instruction Cache Memory is used to cache instructions fetched via the I-Code bus, including instructions fetched by CPU0 from the internal flash memory. The Instruction Cache Memory is 16,384 Bytes in size and is managed using the Instruction Cache Controller 0 (ICC0).

2.4.3 System SRAM

The system SRAM is **Error! Unknown document property name.** in size and can be used for general purpose data storage, the Arm Cortex-M4 system stack and code execution, and the RISC-V system stack and code execution.

2.4.4 AES Key and Working Space Memory

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

2.5 AHB Interfaces

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

2.5.1 Core AHB Interfaces

2.5.1.1 I-Code

This AHB master is used by the Arm core for instruction fetching from memory instances located in code space from byte addresses 0x0000 0000 to 0x1FFF FFFF. This bus 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.

2.5.1.2 D-Code

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

2.5.1.3 System

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

2.5.2 AHB Masters

2.5.2.1 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.

2.6 Peripheral Register Map

2.6.1 APB Peripheral Base Address Map

Table 2-3, *below*, 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 2-3: APB Peripheral Base Address Map

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
Global Control	GCR_	0x4000 0000	0x4000 03FF
System Interface	SIR_	0x4000 0400	0x4000 07FF
Function Control	FCR_	0x4000 0800	0x4000 0BFF
Watchdog Timer 0	WDT0_	0x4000 3000	0x4000 33FF
Dynamic Voltage Scaling Controller	DVS_	0x4000 3C00	0x4000 3C3F
Single Input Multiple Output	SIMO_	0x4000 4400	0x4000 47FF
General Control Function Control	GCFCR_	0x4000 5800	0x4000 5BFF
Real time Clock	RTC_	0x4000 6000	0x4000 63FF
Wakeup Timer	WUT_	0x4000 6400	0x4000 67FF
Power Sequencer	PWRSEQ_	0x4000 6800	0x4000 6BFF
Miscellaneous Control	MCR_	0x4000 6C00	0x4000 6FFF
AES	AES_	0x4000 74000	0x4000 77FF
AES Key	AESK_	0x4000 7800	0x4000 7BFF
GPIO Port 0	GPIO0_	0x4000 8000	0x4000 8FFF
GPIO Port 1	GPIO1_	0x4000 9000	0x4000 9FFF
Parallel Camera Interface	PCIF_	0x4000 E000	0x4000 EFFF
CRC	CRC_	0x4000 F000	0x4000 FFFF
Timer 0	TMR0_	0x4001 0000	0x4001 0FFF
Timer 1	TMR1_	0x4001 1000	0x4001 1FFF
Timer 2	TMR2_	0x4001 2000	0x4001 2FFF
Timer 3	TMR3_	0x4001 3000	0x4001 3FFF
I²C 0	I2C0_	0x4001 D000	0x4001 DFFF
I²C 1	I2C1	0x4001 E000	0x4001 EFFF
I²C 2	I2C2	0x4001 F000	0x4001 FFFF
Standard DMA	DMA	0x4002 8000	0x4002 8FFF
Flash Controller 0	FLC0_	0x4002 9000	0x4002 93FF
Instruction-Cache Controller 0 (CM4)	ICC0_	0x4002 A000	0x4002 A7FF
Instruction Cache Controller 1 (RV32)	ICC1_	0x4002 A800	0x4002 AFFF
ADC	ADC_	0x4003 4000	0x4003 4FFF
Pulse Train Engine	PT_	0x4003 C000	0x4003 C09F
One Wire Master	OWM0_	0x4003 D000	0x4003 DFFF
Semaphore	SEMA_	0x4003 E000	0x4003 EFFF
UART 0	UART0_	0x4004 2000	0x4004 2FFF
UART 1	UART1_	0x4004 3000	0x4004 3FFF
UART 2	UART2_	0x4004 4000	0x4004 4FFF
SPI1	SPI1_	0x4004 6000	0x4004 7FFF
TRNG	TRNG_	0x4004 D000	0x4004 DFFF
I2S	I2S_	0x4006 0000	0x4006 0FFF
Low Power General Control	LPGCR_	0x4008 0000	0x4008 03FF
Low Power GPIO Port 2	GPIO2_	0x4008 0400	0x4008 05FF
GPIO Port 3	GPIO3_	0x4008 0600	0x4008 07FF

Peripheral Register Name	Register Prefix	APB Base Address	APB End Address
Low Power Watchdog Timer 1	WDT1_	0x4008 0800	0x4008 0BFF
Low Power Timer 4	TMR4_	0x4008 0C00	0x4008 0FFF
Low Power Timer 5	TMR5_	0x4008 1000	0x4008 13FF
Low Power UART 0 (UART3)	UART3_	0x4008 1400	0x4008 17FF
Low Power Comparator 0	LPCOMP0_	0x4008 8000	0x4008 83FF
Low Power Comparator 1	LPCOMP1_	0x4008 8400	0x4008 87FF
Low Power Comparator 2	LPCOMP2_	0x4008 8800	0x4008 8BFFF
Low Power Comparator 3	LPCOMP3_	0x4008 8C00	0x4008 8FFF
Trim System Initialization	TRIMSIR_	0x4010 5400	0x4010 57FF
SPI0	SPI0_	0x400B E000	0x4004 E3FF
Convolutional Neural Network FIFO	CNN_FIFO_	0x400C 03FF	0x400F FFFF
Convolutional Neural Network	CNN_	0x5000 0000	0x5FFF FFFF

2.7 Error Correction Coding (ECC) Module

This device features an Error Correction Coding (ECC) module which helps ensure data integrity by detecting and correcting bit corruption of the System RAM0 (***sysram0***) memory array. More specific, this feature is Single Error Correcting, Double Error Detecting (SEC-DED). It corrects any single bit flip, detects 2-bit errors, and features a transparent zero wait state operation for reads.

The ECC works by creating check bits for all data written to ***sysram0***. These check bits are then stored along with the data. During a read, both the data and check bits are used to determine if one or more bits have become corrupt. If a single bit has been corrupted this can be corrected. If two bits have been corrupted, it will be detected, but not corrected.

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

2.7.1 SRAM

A check bit RAM is used to store ***sysram0***'s check bits enabling ECC SEC-DED for ***sysram0***. The check bit RAM is not mapped to the user memory space and is not available for application usage.

2.7.2 Limitations

Any read from non-initialized memory can trigger an ECC error since the random check bits will most likely not match the random data bits contained in the memory. Writing ***sysram0*** to all zeroes prior to enabling ECC functionality can prevent this at the expense of the time required.

3. System, Power, Clocks, Reset

There are several clocks used by different peripherals and subsystems. These clocks are highly configurable by firmware, allowing developers to select the combination of application performance and power savings required for the target systems. Support for selectable core operating voltage is provided and the Internal Primary Oscillator (IPO) frequency is scaled based on the specific core operating voltage range selected.

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

- 100MHz Internal Primary Oscillator (IPO)
- 60MHz Internal Secondary Oscillator (ISO)
- 7.3728MHz Internal Baud Rate Oscillator (IBRO)
  - Selectable for UART baud rate generation
- 8kHz Internal Nano-Ring Oscillator (INRO)
- 32.768kHz External RTC Crystal Oscillator (ERTCO)
  - Clock source for the Real-Time Clock (RTC)
- External Clock, EXT_CLK1, P0.3 AF1

3.1 Core Operating Voltage Range Selection

The MAX78000 supports three selections for the core Operating Voltage Range (OVR). In single supply operation, changing the OVR sets the output of the internal LDO regulator to the voltage shown in *Table 3-1*. For dual supply designs, setting the OVR allows the MAX78000 to use an external PMIC to provide the required V_{CORE} voltage dynamically. Changing the OVR also reduces the output frequency of the Internal Primary Oscillator (IPO), further reducing power consumption.

Changes to the OVR affect the access time of the internal flash memory and the application firmware must set the flash wait states for each OVR setting as outlined in section *3.1.2 Flash Wait States* for details on minimum flash wait states for the internal flash memory.

Changing the core operating voltage immediately reduces the output frequency of the High Frequency Internal Oscillator as shown in *Table 3-1*, *below*. Operating the MAX78000 using dual external supplies requires special considerations and must be handled carefully in the application firmware.

Table 3-1: Operating Voltage Range Selection and the Effect on V_{CORE} and f_{IPO}

GCR_SYSCTRL.ovr	***FLCn_CTRL.lve***	**V_{CORE} Typical**	**f_{IPO}**
0b00	1	0.9	TBD
0b01	1	1.0	100
0b10	0	1.1	100

3.1.1 Setting the Operating Voltage Range

The Operating Voltage Range (OVR) selection is controlled using the Power Sequencer Low Power Control Register *PWRSEQ_LPCN.ovr* which is only reset by a Power-On Reset (POR) and these bits should be checked after every reset to determine the correct clock speed and flash wait states. Adjusting the OVR setting effects the frequency of the IPO. Prior to changing the OVR settings, it is required to set the system clock to either the INRO or the ERTCO. The MAX78000 coordinates OVR change between the internal LDO and the IPO set frequency. When changing the OVR setting, the MAX78000 must be operating from the internal LDO. In a system using an external supply for V_{CORE}, firmware must transition to the internal LDO prior to changing the OVR setting.

The following steps describe how to change the OVR:

1. Ensure the part is operating from the internal LDO for V_{CORE}.
    a. Set *PWRSEQ_LPCN.ldo_dis* to 0.
    b. 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. Refer to section *Oscillator Sources and Clock Switching* for details on system clock selection.
3. Set the number of Flash Wait States to the POR default value of 5.
    a. *GCR_MEMCTRL.fws* = 5
4. Change the OVR setting to the desired range.
    a. Set *PWRSEQ_LPCN.ovr* to either 0, 1, or 2 as shown in *Table 3-1*, *above*.
6. Set the Flash Low Voltage Enable according to the OVR setting set in step 4.a.
    a. Set *FLCn_CTRL.lve* to either 0 or 1 as required. Reference *Table 3-1*, *above* for the required value.
7. If desired, set the system clock source to the IPO and update the system clock prescaler to the desired value.
    a. Set *GCR_CLKCN.clksel* = 0.
    b. Wait for the system clock ready bit, *GCR_CLKCN.clkrdy,* to read 1.
    c. Set *GCR_CLKCN.psc* to the desired prescaler value.
8. Set the number of Flash Wait States per *Table 3-2*, *below*.
    a. Set *GCR_MEMCTRL.fws* to the minimum value shown for the selected OVR and System Clock
9. Perform a Peripheral Reset.
    a. Set *GCR_RSTR0.periph* = 1.

On each subsequent non-POR reset event:

1. Set the flash low voltage enable bit to 1 (*FLCn_CTRL.lve*) to match the setting of *PWRSEQ_LPCN.ovr*, since *PWRSEQ_LPCN.ovr* will not be reset.
2. Set the clock prescaler, *GCR_CLKCN.psc*, 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 3-2*, *below*.

3.1.2 Flash Wait States

Power-On Reset, System Reset and Watchdog Reset all reset the Flash Wait State field, *GCR_MEMCTRL.fws*, to the POR default setting of 5. The Flash Wait State field is the number of system clock cycles for accessing the internal flash memory and is dependent on the OVR settings, the system oscillator selected and the system oscillator prescaler.

The setting for the number of flash wait states effects performance and it is critical to set it correctly based on the OVR settings and the system clock frequency. Set the number of flash wait states using the field *GCR_MEMCTRL.fws* per *Table 3-2*, *below*. The *GCR_MEMCTRL.fws* field should always be set to the default POR reset value of 5 prior to changing the *PWRSEQ_LPCN.ovr* settings.

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

Note: When changing the system clock prescaler to move from a slower system clock frequency to a faster system clock frequency, always set the Flash Wait State field to the minimum required for the faster system clock frequency prior to changing the system oscillator prescaler.

Table 3-2: Minimum Flash Wait State Setting for Each OVR Setting ($f_{SYSCLK} = f_{IPO}$, GCR_CLKCN.ipo_div = 1)

Core Operating Voltage Range Setting		Core Voltage Range	f_{IPO} (MHz)	System Clock Prescaler	System Clock	Minimum Flash Wait State Setting
PWRSEQ_LPCN.ovr	***FLCn_CTRL.lve***	**V_{CORE} (V)**		***PWRSEQ_LPCN.psc***	***f_{SYS_CLK} (MHz)***	***GCR_MEMCTRL.fws***
0	1	0.9	12	0	12	2
				1	6	1
1	1	1.0	50	0	50	3
				1	25	2
2	0	1.1	100	0	100	4
				1	50	2
				2	25	1

3.2 Oscillator Sources and Clock Switching

The selected SYS_OSC is the input to the system oscillator prescaler to generate the System Clock (SYS_CLK). The system oscillator prescaler divides the selected SYS_OSC by a prescaler using the *GCR_CLKCN.psc* field as shown in *Equation 3-1*.

Equation 3-1: System Clock Scaling

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

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

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

- Advanced High-Performance Bus (AHB) Clock
  - *HCLK = SYS_CLK*
- Advanced Peripheral Bus (APB) Clock,
  - $PCLK = \frac{SYS\_CLK}{2}$

The Real-Time Clock (RTC) uses the 32.768kHz external real-time clock oscillator (ERTCO) for its clock source.

All oscillators are reset to their POR reset default state during:

- Power-On Reset
- System Reset
- Watchdog Reset

Oscillator settings are *not* reset during:

- Soft Reset
- Peripheral Reset

Table 3-3 shows each oscillator's enabled state for each type of reset source in the MAX78000. *Table 3-4* details the effect each reset source has on the System Clock selection and the System Clock prescaler settings.

Table 3-3: Reset Sources and Effect on Oscillator Status

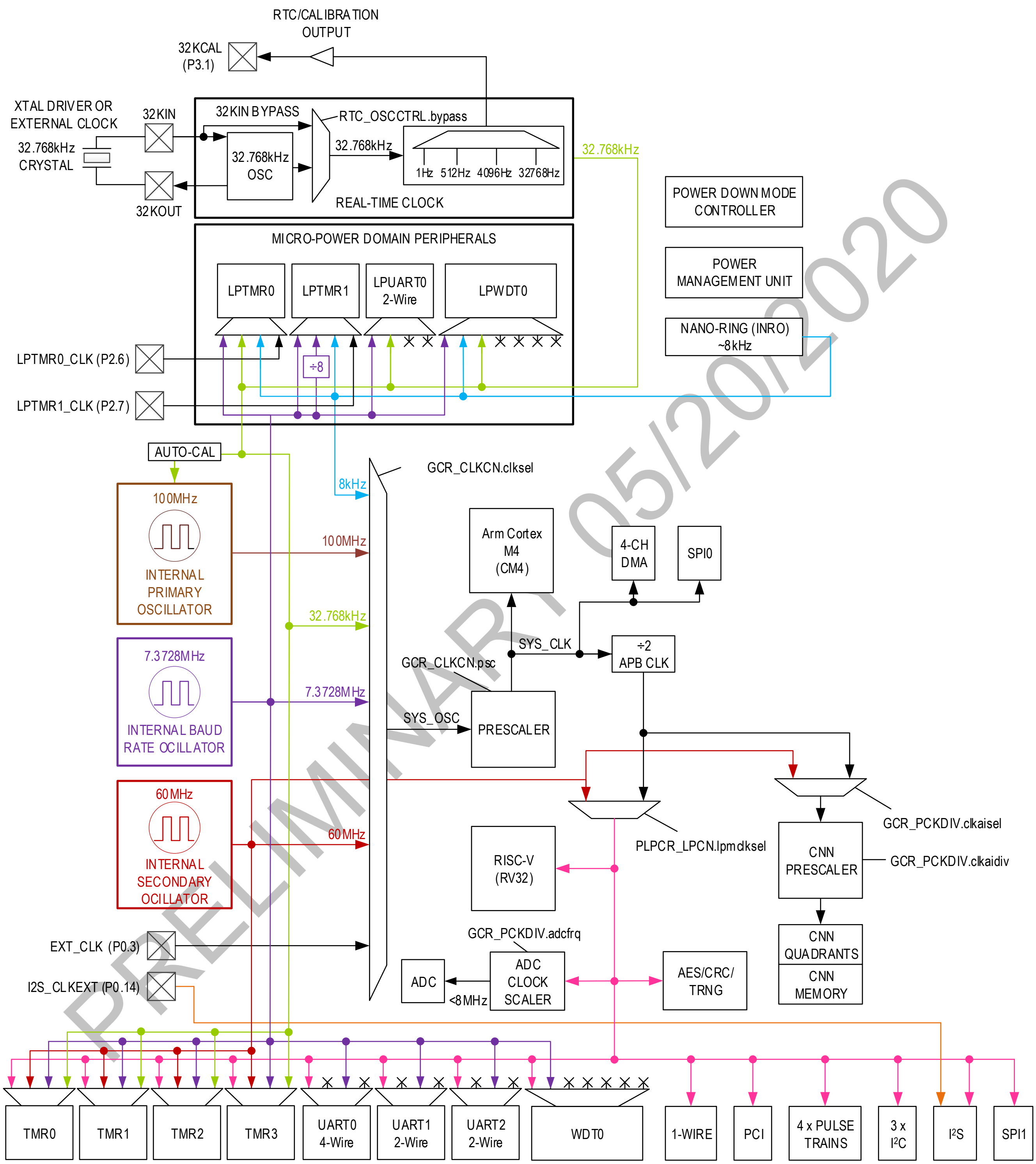
	Reset Source				
Oscillator	**POR**	**System**	**Watchdog**	**Soft**	**Peripheral**
IPO	Enabled	Enabled	Enabled	Retains State	Retains State
ISO					
IBRO					
INRO	Enabled	Enabled	Enabled	Enabled	Enabled
ERTCO	Disabled	Retains State	Retains State	Retains State	Retains State

Table 3-4: Reset Sources and Effect on System Oscillator Selection and Prescaler

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

Figure 3-1: MAX78000 Clock Block Diagram shows a high-level diagram of the MAX78000 clock tree.

Figure 3-1: MAX78000 Clock Block Diagram



3.3 Operating Modes

The MAX78000 includes multiple operating modes and the ability to fine tune power options to optimize performance and power. The system supports the following operating modes:

- ***ACTIVE***
- ***SLEEP***
- ***LOW POWER Mode*** (***LPM***)
- ***MICRO POWER Mode*** (***UPM***)
- ***STANDBY***
- ***BACKUP***
- ***POWER DOWN Mode*** (***PDM***)

3.3.1 ACTIVE Mode

In this mode, both the CM4 and the RV32 cores can execute application code and all digital and analog peripherals are available on demand. Dynamic clocking disables peripheral not in use, providing the optimal mix of high performance and low power consumption. The CM4 has access to all system RAM by default. The RV32 has access to ***sysram2*** and ***sysram3*** and optionally can be configured to have exclusive access to these RAM. Additionally, ***sysram3*** can be configured as a instruction cache controller for the RV32 allowing simultaneous code execution for the CM4 and RV32 from the internal flash memory.

3.3.2 SLEEP Mode

This mode consumes less power but wakes faster because the clocks can optionally be enabled.

The device status is as follows:

- CM4 (CPU0) is asleep
- RV32 (CPU1) is asleep
- The CNN is available for optional use
- Standard DMA is available for optional use
- All peripherals are on

Entry into SLEEP Mode:

Explicit handshaking between CM4 and RV32 (Use the Semaphore block)

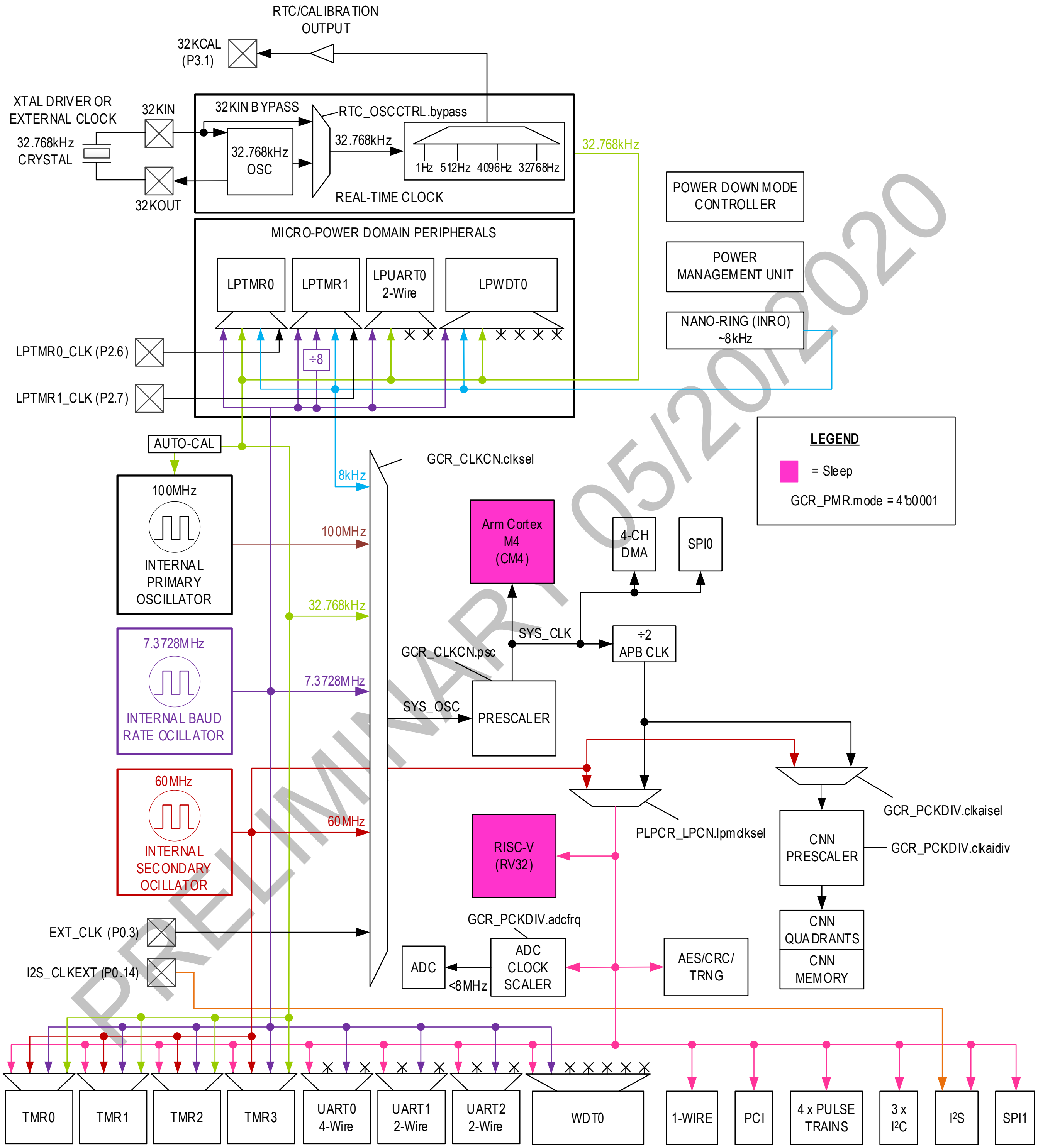
CM4 SLEEP Mode Entry

The following pseudocode places the device in SLEEP mode:

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

Alternate: If the CM4 is the primary CPU, the CM4 can notify RV32 using Semaphore and RV32 can perform a WFI to enter sleep, then the CM4 can enter Sleep using the SCR.sleepdeep, WFI (WFE) instruction.

Figure 3-2: SLEEP Mode Clock Control



3.3.3 LOW POWER Mode (LPM)

This mode is suitable for running the RV32 processor to collect and move data from enabled peripherals. The device status is a follows:

- The CM4, SRAM0, SRAM1 are in state retention
- CNN quadrants and memory are configurable and active
- The RV32 can access the SPI, all UARTS, all Timers, I2C, 1-Wire, timers, pulse train engines, I2S, CRC, AES, TRNG, PCIF, Comparators as well as SRAM2 and SRAM3. SRAM3 can be configured to operate as RV32 instruction cache
- The transition from Low Power mode to Active mode is faster than the transition from ***BACKUP*** Mode because system initialization is not required
- The DMA can access flash
- The following oscillators are powered down: NOT AUTO POWERDOWN> Configurable.
  - IPO
- The following oscillators are enabled:
  - IBRO
  - ERTCO
  - INRO
  - ISO

Method 1: CM4 puts itself into DEEPSLEEP and system will automatically enter LPM and set the .mode field to LPM

To place the CM4 in DEEPSLEEP mode, perform the following instructions.

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

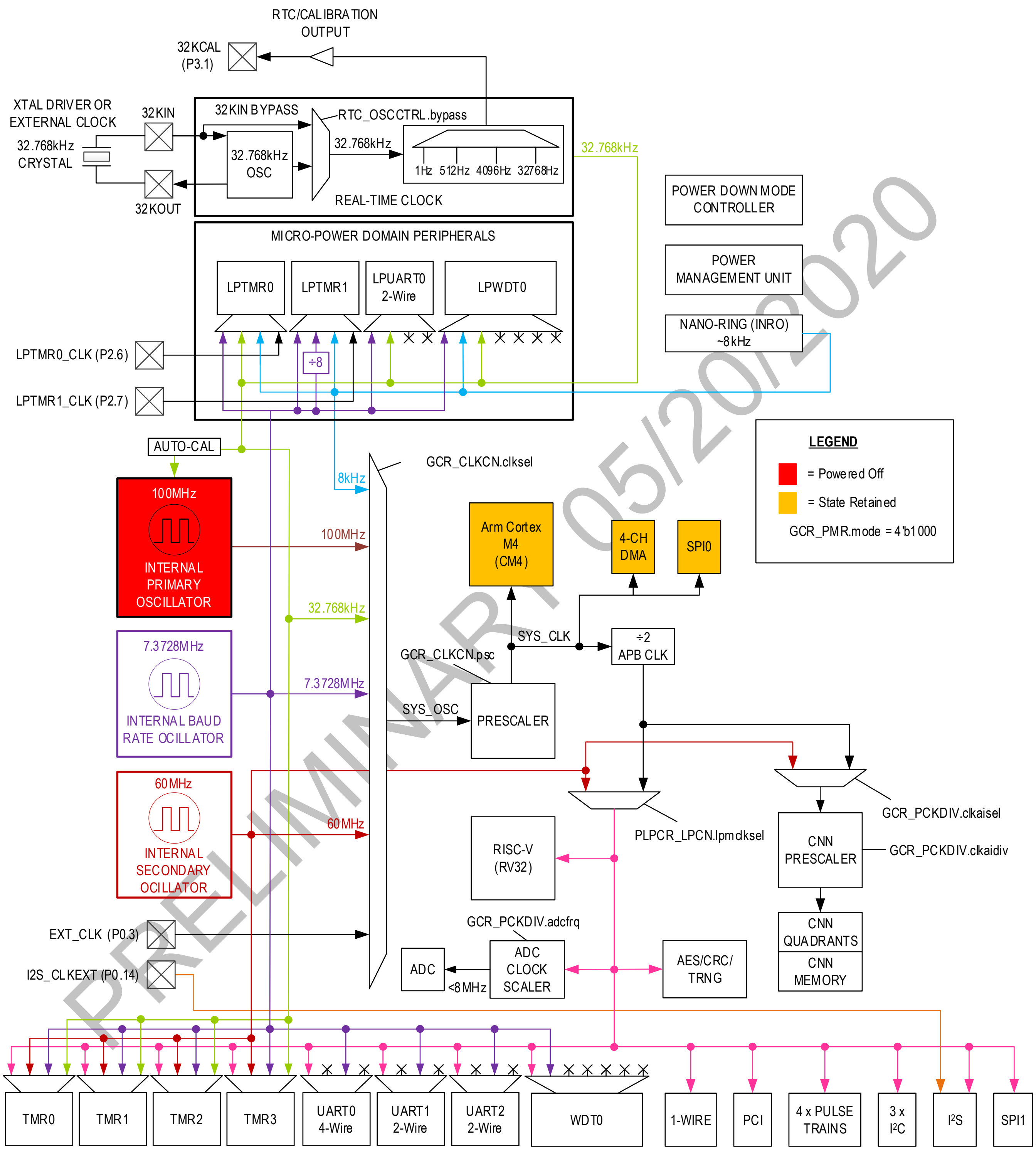
GCR_PMR.mode doesn't need to be set in this case. Hardware will enter mode automatically and set the .mode field to LPM.

Method 2: If CM4 is in SLEEP state and RV32 sets the GCR_PMR.mode to LPM.

```
SCR.sleepdeep = 0; // SLEEP mode enabled
WFI (or WFE);      // Enter SLEEP mode
```

RV32 can set GCR_PMR.mode to LPM. CM4 needs to be in a known sleep or deepsleep state.

Figure 3-3: LOW POWER Mode (LPM) Clock and State Retention Diagram



3.3.4 MICRO POWER Mode (UPM)

This mode is used for extremely low power consumption while using a minimal set of peripherals to provide wakeup capability. The device status is a follows:

- Both CM4 and RV32 are state retained.
- System state and all SRAM is retained
- CNN quadrants are powered off
- CNN memory provides selectable retention
- The GPIO pins retain their state
- All non-Micro Power peripherals are state retained
- The following oscillators are powered down:
  - IPO
  - ISO
- The following oscillators are enabled:
  - IBRO
  - ERTCO
  - INRO
- The following Micro Power mode peripherals are available for use to wakeup the device:
  - LPUART0
  - LPUART1
  - LPWDT0
  - All four low power analog comparators

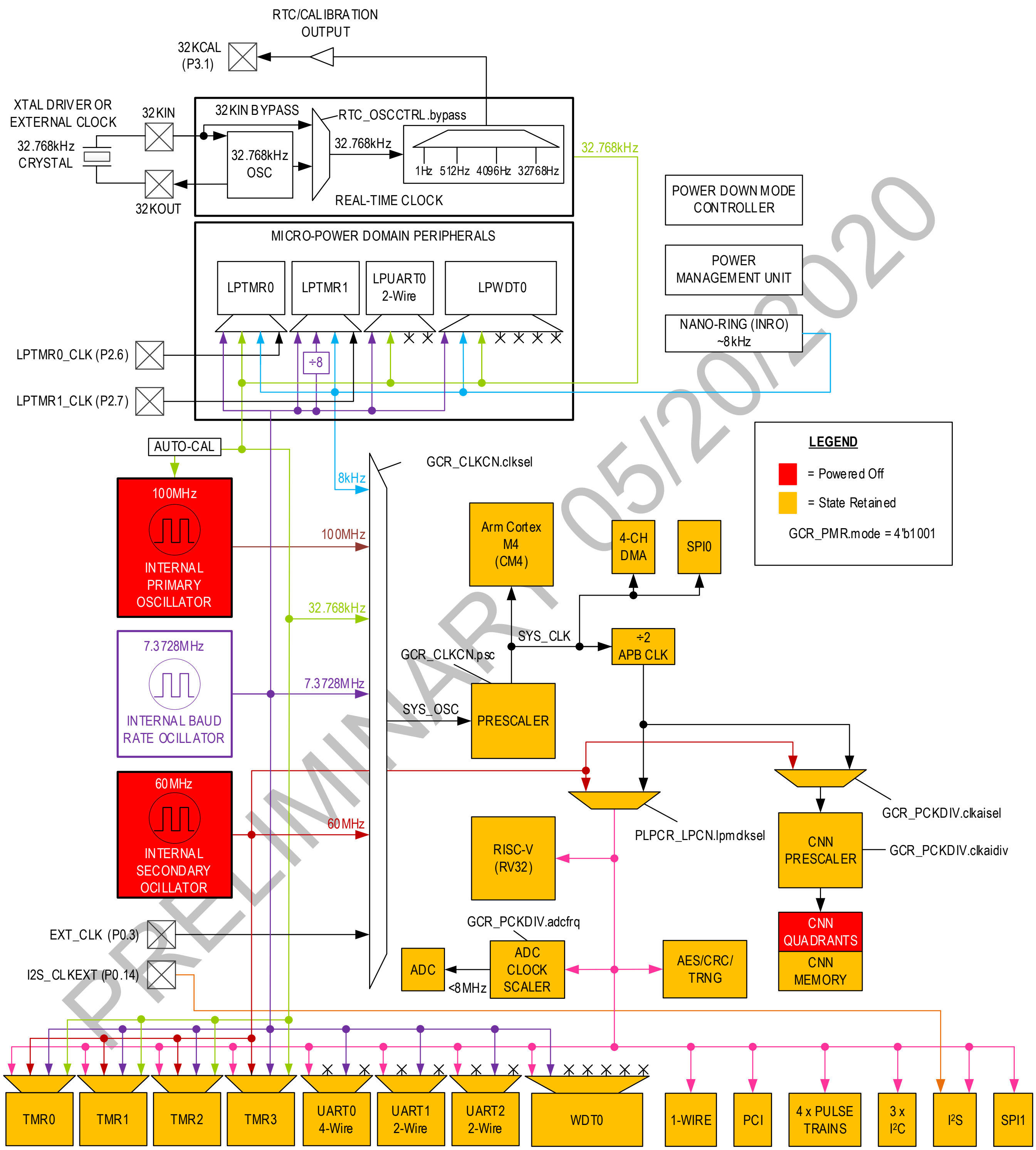
Entry method 1: Mailbox approach RV32 notify CM4 to enter UPM. CM4 enters ***SLEEP*** mode. Then RV32 sets GCR_PMR.mode to UPM.

Entry method 2: Mailbox approach. RV32 goes to sleep using WFI. CM4 sets the GCR_PMR.mode to UPM

```
GCR_PMR.mode = UPM;

// CM4 stops at this point and resumes execution from the following line of code.
```

Figure 3-4: MICRO POWER Mode (UPM) Clock and State Retention Block Diagram



3.3.5 STANDBY Mode

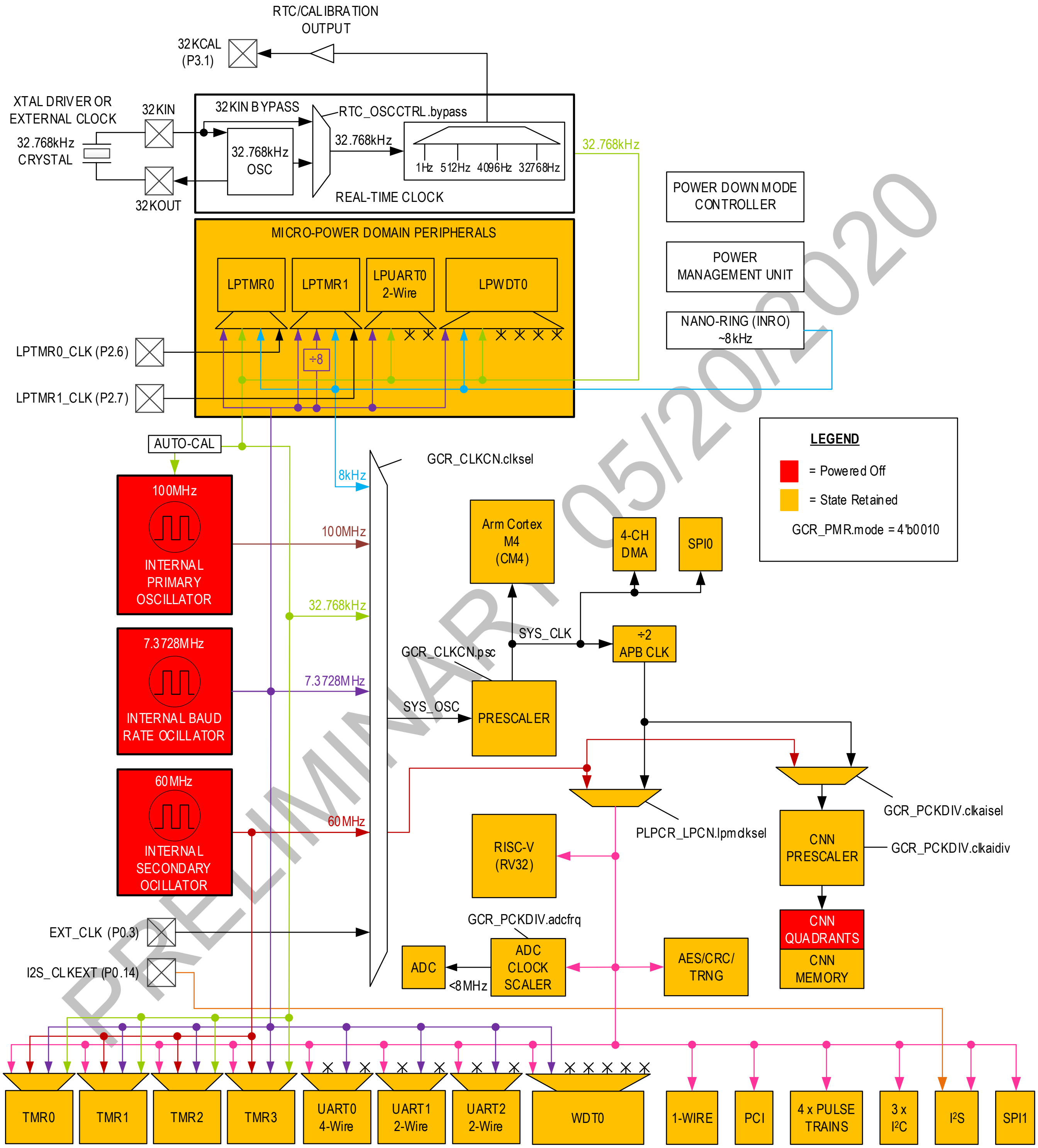
This mode is used maintain the system operation while keeping time with the RTC. The device status is a follows:

- Both CM4 and RV32 are state retained. System state and all SRAM is retained
- CNN quadrants are powered off
- CNN memory provides selectable retention
- The GPIO pins retain their state
- All peripherals are state retained
- The following oscillators are powered down:
  - IPO
  - ISO
  - IBRO
- The following oscillators are enabled:
  - ERTCO
  - INRO

Method 1: Mailbox/Semaphore (CM4 enter SLEEP), RV32 then sets the GCR_PMR.mode to STANDBY.

Method 2: Mailbox/Semaphore (RV32 enters SLEEP) using WFI, CM4 sets GCR_PMR.mode to STANDBY.

Figure 3-5: STANDBY Mode Clock and State Retention Block Diagram



3.3.6 BACKUP Mode

This mode is used maintain the system RAM. The device status is a follows:

- CM4 and RV32 are powered off.
- SRAM0, SRAM1, SRAM2 and SRAM3 can be configured to be state retained as per Table 1
- CNN memory provides selectable retention
- All peripherals are powered off
- The following oscillators are powered down:
  - IPO
  - ISO
  - IBRO
  - INRO
- The following oscillators are enabled:
  - ERTCO (RTC can be turned off, but not the oscillator)

*Table 3-5 System RAM Retention in **BACKUP** Mode*

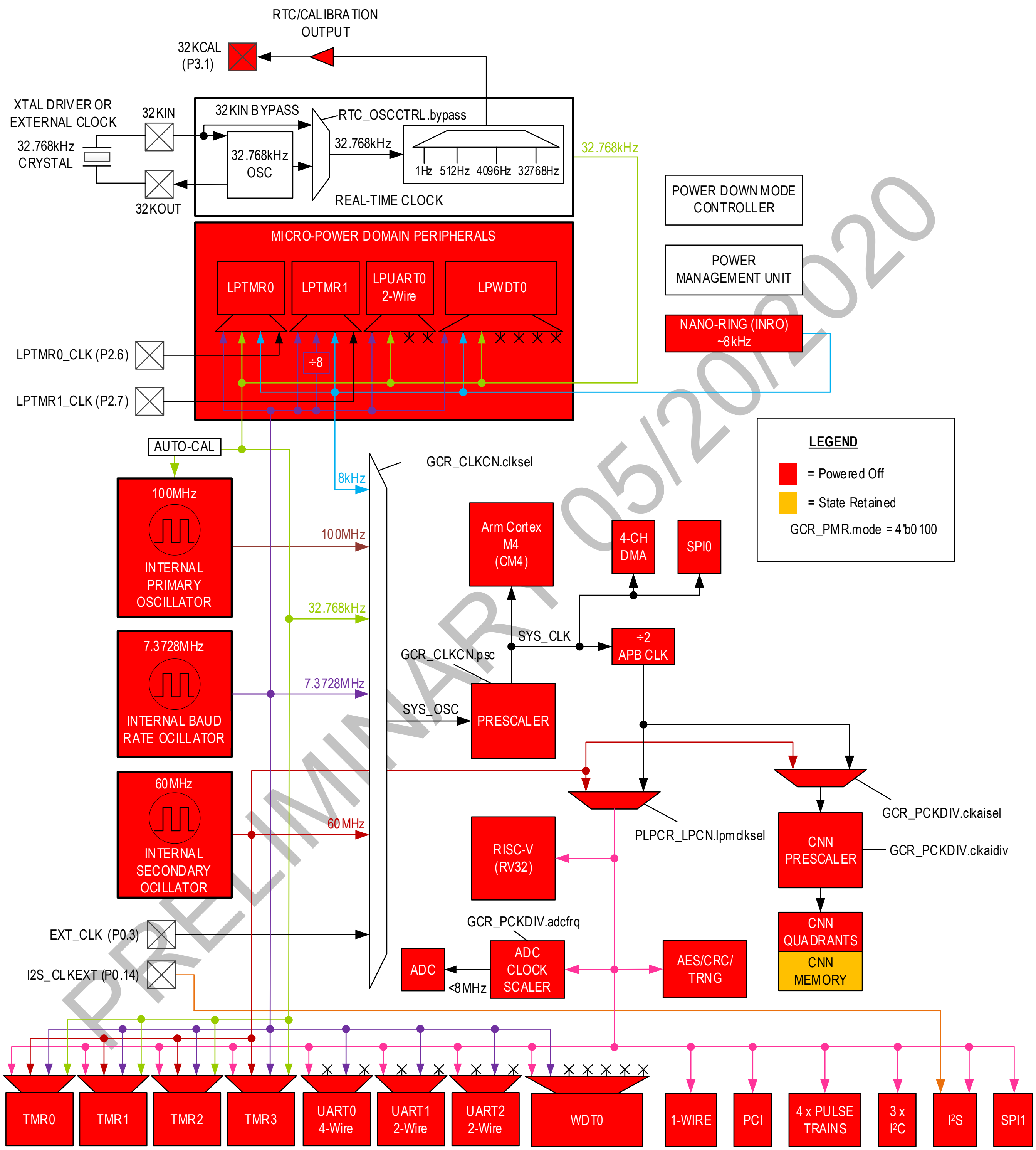
RAM Block #	Size	STATE RETENTION BITS	State Retention
sysram0	32KB + ECC		Configurable
sysram1	32KB		Configurable
sysram2	48KB		Configurable
sysram3	16KB		Configurable
CNN RAM			Configurable

Entry Method 1: Mailbox Negotiation Recommended to assure both processors are aware of the backup state and complete any memory transactions. (CM4 enters Sleep, then RV32 writes GCR_PMR.mode to BACKUP or vice-versa)

Entry Method 2: If not concerned about memory transactions, either CPU can set GCR_PMR.mode to BACKUP.

GPR0 is used to bypass the ROM when exiting from backup mode.

*Figure 3-6: **BACKUP** Mode Clock and State Retention Block Diagram*



3.3.7 POWER DOWN Mode (PDM)

This mode is used during product level distribution and storage The device status is a follows:

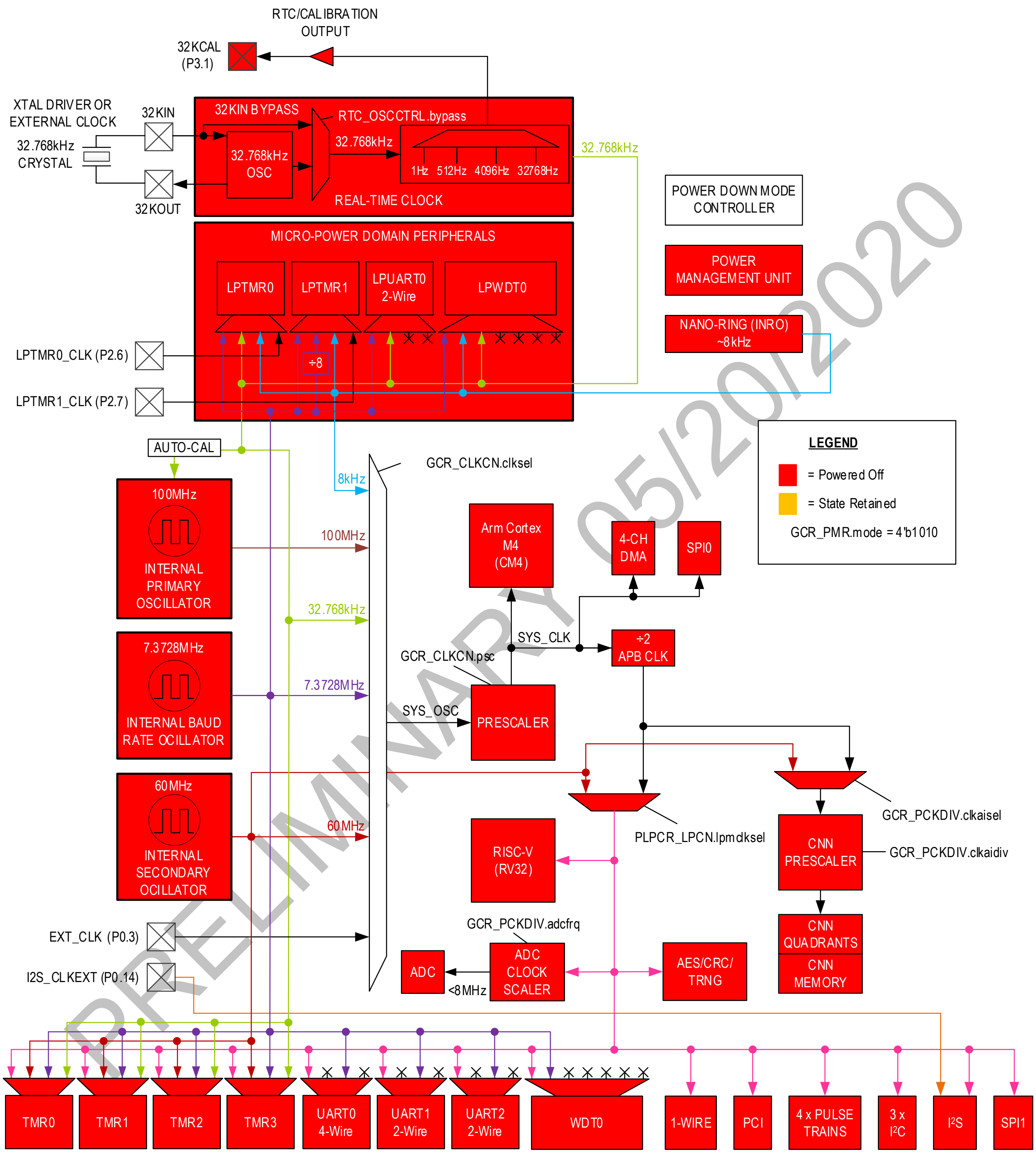
- The CM4 and RV32 are powered off
- All peripherals and SRAM are powered down
- All oscillators are powered down
- There is no data retention in this mode, but values in the flash are preserved
- VREGI POR Voltage monitor is operational.
- A 64-Bit register can be retained…..AoD register…

Exit from this mode: External RST or GPIO3.

Entry Method 1: Mailbox/Semaphore to ensure no flash transactions are in progress

PWRSEQ_GPR0, PWRSEQ_GPR1 retain

Figure 3-7: POWER DOWN Mode (PDM) Clock and State Retention Block Diagram



3.4 Operating Modes Wakeup Sources

In all operating modes other than ACTIVE, wakeup sources are required to re-enter ACTIVE Mode operation. The Table below shows available wakeup sources for each operating mode of the MAX78000.

Note: Each wakeup source must be enabled individually except for External Reset, which is hardware controlled.

Table 3-6: Wakeup Sources for Each Operating Mode in the MAX78000

Operating Mode	Any Peripheral Interrupts	External Reset	RV32	CNN	CNN FIFO	SPI1	SPI0	I2S	I2C2	I2C1	I2C0	LPUART0 (UART3)	UART2	UART1	UART0	LPTMR1 (TMR5)	LPTMR0 (TMR4)	TMR3	TMR2	TMR1	TMR0	LPWDT0 (WDT1)	WDT0	LPCMP3	LPCMP2	LPCMP1	LPCMP0	RTC	WUT	GPIO3	GPIO2	GPIO1	GPIO0
SLEEP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LPM		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
UPM		✓										✓				✓	✓					✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STANDBY		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BACKUP		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PDM		✓																												✓			

3.5 Device Resets

Four device resets are available:

- Peripheral Reset
- Soft Reset
- System Reset
- Power-On Reset

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 mode. Program execution begins at the reset vector address.

Contents of the Always-On Domain (AoD) are reset only on power-cycling V_{DD} and V_{CORE}.

Each of the on-chip peripherals can also be reset to their POR default state using the two reset registers *GCR_RSTR0* and *GCR_RSTR1*.

Table 3-7: Reset and Low Power Mode Effects shows the effects of the four reset types and the four power modes.

Table 3-7: Reset and Low Power Mode Effects

	Peripheral Reset[4]	Soft Reset[4]	System Reset[4]	POR	ACTIVE Mode	SLEEP Mode	DEEPSLEEP Mode	BACKUP[3] Mode	STORAGE Mode
GCR	-	-	Reset	Reset	R	-	-	-	-
IPO	-	-	Off	Off	R	-	Off	Off	Off
ISO	-	-	Off	Off	R	-	-	-	-

	Peripheral Reset[4]	Soft Reset[4]	System Reset[4]	POR	ACTIVE Mode	SLEEP Mode	DEEPSLEEP Mode	BACKUP[3] Mode	STORAGE Mode
ERTCO	-	-	-	Off	FW	FW	FW	FW	FW
IBRO	-	-	Off	Off	R	-	Off	Off	Off
ERFO	-	-	Off	Off	R	-	Off	Off	Off
INRO	On	On	On	On	On	On	On	On	On
SYS_CLK	On	On	On[2]	On[2]	On	On	Off	Off	Off
CPU Clock	On	On	On	On	On	Off	Off	Off	Off
RTC				Reset	FW	FW	FW	FW	FW
CPU	-	-	Reset	Reset	R	Off	Off	Off	Off
WDT0/1	-	-	Reset	Reset	R	-	Off	Off	Off
GPIO	-	Reset	Reset	Reset	R	-	-	-	-
Other Peripherals	Reset	Reset	Reset	Reset	R	-	Off	Off	Off
Always-On Domain[1]	-	-	-	Reset	-	-	-	-	-
RAM Retention	-	-	-	Reset	-	-	On	FW	Off

Table key:

FW = Controlled by firmware

On = Enabled by hardware (Cannot be disabled)

Off = Disabled by hardware (Cannot be enabled)

- = No Effect

R = Restored to previous ACTIVE mode setting when exiting DEEPSLEEP mode, restored to System Reset state when exiting BACKUP or STORAGE mode.

1: The Always On Domain (AoD) is only reset on power-cycling V_{DD} and V_{CORE}.

2: On a System Reset or POR, the IPO will automatically be selected as SYS_OSC.

3: A System Reset occurs when returning from BACKUP or STORAGE low-power mode.

4: Peripheral, Soft, and System Resets are initiated by firmware though the *GCR_RSTR0* register. System Reset can also be triggered by the RSTN device pin or Watchdog reset.

3.5.1 Peripheral Reset

This resets 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_RSTR0.periph* = 1. The reset will be completed immediately upon setting *GCR_RSTR0.periph* = 1.

3.5.2 Soft Reset

This is the same as a Peripheral Reset except that it also resets the GPIO to its Power-On Reset state.

To start a Soft Reset, set *GCR_RSTR0.soft* = 1. The reset will be completed immediately upon setting *GCR_RSTR0.soft* = 1.

3.5.3 System Reset

This is the same as Soft Reset except it also resets all GCR, resetting the clocks to their default state. The CPU state is reset as well as the watchdog timers. The AoD and RAM are unaffected.

A watchdog timer reset event initiates a System Reset. To start a System Reset from firmware, set *GCR_RSTR0.sys* = 1.

3.5.4 Power-On Reset

A POR resets everything in the device to its default state.

3.6 Instruction Cache Controller

The **MAX78000** includes two instruction cache controllers. ICC0 is the cache controller used for CPU0, the Arm Cortex-M4 core. ICC1 is dedicated to CPU1, the RISC-V core. ICC0 includes a line buffer, tag RAM and a 16KB 2-way set associative Data RAM.

3.6.1 Enabling the Instruction Cache Controller

Perform the following steps to enable ICCn:

1. Set **Error! Reference source not found.**.*icachesd* to 0 to ensure the cache power is on.
2. Set *ICCn_CTRL*.*en* to 1.
3. Read *ICCn_CTRL*.*rdy* until it returns 1.

3.6.2 Disabling the ICC

Disable ICC0 by setting *ICCn_CTRL*.*enable* to 0.

3.6.3 Invalidating the ICC0 Cache

The System Configuration Register (*GCR_*) includes a field for flushing the ICCO cache. Setting *GCR_*.*icc0_flush* to 1 flushes the ICCO 16KB cache and the tag RAM. Setting the *ICCn_INVALIDATE* register to 1 invalidates the ICCO cache and forces a cache flush. Read the *ICCn_CTRL*.*rdy* field until it returns 1 to determine when the flush is completed.

Heading 4

3.6.4 Instruction Cache Control Registers (ICC)

See *Table 2-3: APB Peripheral Base* Address Map 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 3-8: Instruction Cache Controller Register Summary*. 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 2-1. Field Access Definitions* 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 3-8: Instruction Cache Controller Register Summary

Offset	Register	Name
[0x0000]	*ICCn_INFO*	*Cache ID Register*
[0x0004]	*ICCn_SZ*	*Cache Memory Size Register*
[0x0100]	*ICCn_CTRL*	*Instruction Cache Control Register*
[0x0700]	*ICCn_INVALIDATE*	*Instruction Cache Controller Invalidate Register*

3.6.5 ICC0 Register Details

Table 3-9: ICC0 Cache Information Register

ICC0 Cache Information			ICCn_INFO	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**

ICC0 Cache Information				ICCn_INFO	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
15:10	id	R	-	**Cache ID** Returns the ID for this cache instance.	
9:6	partnum	R	-	**Cache Part Number** Returns the part number indicator for this cache instance.	
5:0	relnum	R	-	**Cache Release Number** Returns the release number for this cache instance.	

Table 3-10: ICC0 Memory Size Register

ICC0 Memory Size				ICCn_SZ	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:16	mem	R	-	**Addressable Memory Size** Indicates the size of addressable memory by this cache controller instance in 128KB units.	
15:0	cch	R	-	**Cache Size** Returns the size of the cache RAM memory in 1KB units. 16: 16KB Cache RAM	

Table 3-11: ICC0 Cache Control Register

ICC0 Cache Control				ICCn_CTRL	[0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:17	-	R/W	-	**Reserved**	
16	rdy	R	-	**Ready** This field is cleared by hardware anytime the cache as a whole is invalidated (including a POR). Hardware automatically sets this field to 1 when the invalidate operation is complete and the cache is ready. 0: Cache invalidation in process. 1: Cache is ready. *Note: While this field reads 0, the cache is bypassed and reads come directly from the line fill buffer.*	
15:1	-	R/W	-	**Reserved**	
0	en	R/W	0	**Cache Enable** Set this field to 1 to enable the cache. Setting this field to 0 automatically invalidates the cache contents and reads are handled by the line fill buffer. 0: Disable 1: Enable	

Table 3-12: ICC0 Invalidate Register

ICC0 Invalidate				ICCn_INVALIDATE	[0x0700]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	invalid	W	-	**Invalidate** Writing any value to this register invalidates the cache.	

3.7 RAM Memory Management

This device has many features for managing the on-chip RAM. The on-chip RAM includes data RAM, instruction and data cache (ICC0) for CPU0 and instruction and data cache (ICC1) for CPU1, and peripheral FIFOs.

3.7.1 On-Chip Cache Management

The MAX78000 includes two instruction cache controllers for code fetches from the flash memory. The caches can be enabled, disabled, and zeroized or flushed and the cache clock can be disabled by placing it in Light Sleep. Refer to section *Low-Power General Control Registers Details* for details.

3.7.2 RAM Zeroization

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

The following SRAM memories can be zeroized:

- Each of the System RAMs can be individually zeroized by setting the respective *GCR_MEMZ* bit.
  - *GCR_MEMZ.sram0*
  - *GCR_MEMZ.sram1*
  - *GCR_MEMZ.sram2*
  - *GCR_MEMZ.sram3*
- ICC0 16KB Cache
  - Write 1 to *GCR_MEMZ*.icache0z (DOES THIS FIELD EXIST SOMEWHERE?)
- ICC1 16KB Cache, if enabled
  - Write 1 to *GCR_MEMZ.icache1z*

~~3.7.3 RAM Low Power Modes~~

~~RAM low power modes and shutdown are controlled on a bank basis.~~ The System SRAM banks are shown with corresponding bank sizes, base address and ending addresses in *Table 3-13*, *below*.

Table 3-13 RAM Block Size and Base Address

System RAM Block #	Size	Base Address	End Address
sysram0	32KB	2000 0000	2000 7FFF
sysram1	32KB	2000 8000	2000 FFFF
sysram2	48KB	2001 0000	2001 BFFF
sysram3	16KB	2001 C000	2001 FFFF

~~3.7.3.1 RAM Shut Down~~

~~RAM memories can individually be shut down further reducing the power consumption for the device. Shutting down a memory gates off the clock and removes power to the memory. Shutting down a memory invalidates (destroys) the contents of the memory and results in a POR of the memory when it is enabled. RAM memory shut down is configured using the **Error! Reference source not found.** register.~~

3.8 Miscellaneous Control Registers (MCR)

See *Table 2-3: APB Peripheral Base* Address Map for the Error Correction Coding Enable Peripheral Address.

Table 3-14: Miscellaneous Control Register Summary

Offset	Register Name	Access	Description
[0x0000]	*MCR_ECC_EN*	R/W	*Error Correction Coding Enable Register*
[0x0004]	*MCR_IPO_MTRIM*	R/W	*IPO Manual Trim Register*
[0x0008]	*MCR_OUT_EN*	R/W	*Miscellaneous Output Enable Register*
[0x000C]	*MCR_CMP_CTRL*	R/W	*Comparator Control Register*
~~[0x0010]~~	~~MCR_CTRL~~	~~R/W~~	~~Miscellaneous Control Register~~
[0x0020]	*MCR_GPIO3_CTRL*	R/W	*GPIO3 Pin Control Register*

3.8.1 Miscellaneous Control Register Details

Table 3-15: Error Correction Coding Enable Register

Error Correction Coding Enable			MCR_ECC_EN	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved** Do not modify this field.
0	ram	R/W	0	**System RAM ECC Enable** Set this field to 1 to enable ECC for sysram0. 0: Disabled 1: Enabled

Table 3-16: IPO Manual Register

IPO Manual Trim			MCR_IPO_MTRIM	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8	trim_range	R/W	0	**Trim Range Select** If this bit is set to 1, the value loaded into *MCR_IPO_MTRIM.mtrim* must be greater than the trim setting in TRIMSIR_15[7:0]. If this bit is set to 0, the value loaded into *MCR_IPO_MTRIM.mtrim* must be less than the trim setting in TRIMSIR_16[23:16]. 0: *mtrim* must be less than TRIMSIR_16[23:16] 1: *mtrim* must be greater than TRIMSIR_15[7:0]
7:0	mtrim	R/W	0x04	**Manual Trim Value** Set this value to the desired manual trim based on the value set in *MCR_IPO_MTRIM.trim_range*.

Table 3-17: Output Enable Register

Output Enable			MCR_OUT_EN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved** Do not modify this field.

Output Enable				MCR_OUT_EN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**	
1	pdown_out_en	R/W	0	**Power Down Output Enable on P3.0** Set this field to 1 to enable the power down output, P3.0 AF1 (PDOWN). PDOWNZ is active in Backup and Standby Mode. 0: PDOWN output not enabled on P3.0 1: PDOWN output is enabled on P3.0	
0	sqwout_en	R/W	0	**Square Wave Output Enable on P3.1 (SQWOUT)** Set this field to 1 to enable the square wave output on P3.1 AF1 (SQWOUT). 0: Square wave output not enabled on P3.1 1: Square wave output enabled on P3.1	

Table 3-18: Comparator Control Register

Comparator Control				MCR_CMP_CTRL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:16	-	RO	0	**Reserved** Do not modify this field.	
8	ram	R/W	0	**TBD** Set this field to 1 to enable ECC for sysram0. 0: Disabled 1: Enabled	
7:0	-	RO	0	**Reserved** Do not modify this field.	

Table 3-19: Miscellaneous Control Register

Miscellaneous Control				MCR_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:10	-	RO	0	**Reserved** Do not modify this field.	
9	simo_rstd	R/W	0	**SIMO System Reset Disable** If this field is set, the SIMO is only reset by a Power-On Reset. When this bit is set, the VSET* stay's unchanged when exiting all low power modes. 0: The SIMO is reset by all system resets. 1: The SIMO is only reset by a Power-On Reset.	
8	simo_clkscl_en	R/W	0	**SIMO Clock Scaling Enable** Set this field to 1 to enable dynamic clock scaling to the SIMO based on load current. When enabled, the SIMO clock slows down in low power modes, reducing current consumption. 0: SIMO clock scaling disabled 1: SIMO clock scaling enabled	
7:4	-	DNM	0x01	**Reserved** Do not modify this field.	

Miscellaneous Control			MCR_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**
3	ertco_en	R/W	0	**ERTCO Enable** Set this field to 1 to enable the ERTCO. 0: ERTCO disabled 1: ERTCO enabled
2	inro_en	R/W	0	**INRO Enable** Set this field to 1 to enable the INRO. 0: INRO disabled 1: INRO enabled
1:0	-	RO	0	**Reserved** Do not modify this field.

Table 3-20: GPIO3 Pin Control Register

GPIO3 Pin Control			MCR_GPIO3_CTRL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved** Do not modify this field.
7	p31_in	RO	See Description	**GPIO3 Pin 1 Input Status** Read this field to determine the input status of P3.1. 0: Input Low 1: Input High
6	p31_pe	R/W	0	**GPIO3 Pin 1 Pull-up Enable** Set this bit to 1 to enable the pull-up resistor for P3.1 0: Pull-up Disabled 1: Pull-up Enabled
5	p31_oe	R/W	0	**GPIO3 Pin 1 Output Enable** Set this bit to 1 to enable P3.1 for output mode. 0: Input mode 1: Output mode enabled.
4	p31_do	R/W	0	**GPIO3 Pin 1 Data Output** If *p31_oe* is set to 1, this field is used to control the output state of P3.1. 0: Output low if *p31_oe* is 1 1: Output high if *p31_oe* is 1.
3	p30_in	RO	See Description	**GPIO3 Pin 0 Input Status** Read this field to determine the input status of P3.0. 0: Input Low 1: Input High
2	p30_pe	R/W	0	**GPIO3 Pin 0 Pull-up Enable** Set this bit to 1 to enable the pull-up resistor for P3.0 0: Pull-up Disabled 1: Pull-up Enabled

GPIO3 Pin Control			MCR_GPIO3_CTRL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
1	p30_oe	R/W	0	**GPIO3 Pin 0 Output Enable** Set this bit to 1 to enable P3.0 for output mode. 0: Input mode 1: Output mode enabled.
0	p30_do	R/W	0	**GPIO3 Pin 0 Data Output** If *p30_oe* is set to 1, this field is used to control the output state of P3.0. 0: Output low if *p30_oe* is 1 1: Output high if *p30_oe* is 1.

3.9 Low-Power General Control Registers (LPGCR)

This set of general control registers provides reset and clock control for the low-power peripherals including:

- Low Power UART3
- Low Power TMR4
- Low Power TMR5
- Low Power WDT1
- Low Power CMP

See *Table 2-3: APB Peripheral Base* Address Map 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 3-21 Miscellaneous Control Register Summary*. 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 2-1. Field Access Definitions* 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 3-21 Miscellaneous Control Register Summary

Offset	Register	Name
[0x0004]	*LPGCR_RST*	*Reset Control Register*
[0x0008]	*LPGCR_CLKDIS*	*Clock Control Register*

3.9.1 Low-Power General Control Registers Details

Table 3-22: Reset Control Register

Error Correction Coding Enable			**LPGCR_RST**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved** Do not modify this field.
6	lp_cmp	W1	0	**Low Power CMP Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See Operating Modes Wakeup Sources In all operating modes other than ACTIVE, wakeup sources are required to re-enter ACTIVE Mode operation. The Table below shows available wakeup sources for each operating mode of the MAX78000. Note: Each wakeup source must be enabled individually except for External Reset, which is hardware controlled.

Table 3-6: Wakeup Sources for Each Operating Mode in the MAX78000

Operating Mode	Any Peripheral Interrupts	External Reset	RV32	CNN	CNN FIFO	SPI1	SPI0	I2S	I2C2	I2C1	I2C0	LPUART0 (UART3)	UART2	UART1	UART0	LPTMR1 (TMR5)	LPTMR0 (TMR4)	TMR3	TMR2	TMR1	TMR0	LPWDT0 (WDT1)	WDT0	LPCMP3	LPCMP2	LPCMP1	LPCMP0	RTC	WUT	GPIO3	GPIO2	GPIO1	GPIO0
SLEEP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LPM		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
UPM		✓										✓				✓	✓					✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STANDBY		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BACKUP		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PDM		✓																												✓			

Device Resets for additional information.

Bits	Field	Access	Reset	Description
5	-	RO	0	**Reserved** Do not modify this field.

Error Correction Coding Enable			LPGCR_RST	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
4	lp_uart3	W1O	0	**Low Power UART3 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See Operating Modes Wakeup Sources In all operating modes other than ACTIVE, wakeup sources are required to re-enter ACTIVE Mode operation. The Table below shows available wakeup sources for each operating mode of the MAX78000. Note: Each wakeup source must be enabled individually except for External Reset, which is hardware controlled.

Table 3-6: Wakeup Sources for Each Operating Mode in the MAX78000

Operating Mode	Any Peripheral Interrupts	External Reset	RV32	CNN	CNN FIFO	SPI1	SPI0	I2S	I2C2	I2C1	I2C0	LPUART0 (UART3)	UART2	UART1	UART0	LPTMR1 (TMR5)	LPTMR0 (TMR4)	TMR3	TMR2	TMR1	TMR0	LPWDT0 (WDT1)	WDT0	LPCMP3	LPCMP2	LPCMP1	LPCMP0	RTC	WUT	GPIO3	GPIO2	GPIO1	GPIO0
SLEEP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LPM		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
UPM		✓										✓				✓	✓					✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STANDBY		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BACKUP		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PDM		✓																												✓			

Device Resets for additional information.

Error Correction Coding Enable			LPGCR_RST	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
3	lp_tmr5	W1O	0	**LPTMR1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See Operating Modes Wakeup Sources In all operating modes other than ACTIVE, wakeup sources are required to re-enter ACTIVE Mode operation. The Table below shows available wakeup sources for each operating mode of the MAX78000. Note: Each wakeup source must be enabled individually except for External Reset, which is hardware controlled. *Table 3-6: Wakeup Sources for Each Operating Mode in the MAX78000* (see table below) Device Resets for additional information.

Table 3-6: Wakeup Sources for Each Operating Mode in the MAX78000

Operating Mode	Any Peripheral Interrupts	External Reset	RV32	CNN	CNN FIFO	SPI1	SPI0	I2S	I2C2	I2C1	I2C0	LPUART0 (UART3)	UART2	UART1	UART0	LPTMR1 (TMR5)	LPTMR0 (TMR4)	TMR3	TMR2	TMR1	TMR0	LPWDT0 (WDT1)	WDT0	LPCMP3	LPCMP2	LPCMP1	LPCMP0	RTC	WUT	GPIO3	GPIO2	GPIO1	GPIO0
SLEEP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LPM		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
UPM		✓										✓				✓	✓					✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STANDBY		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BACKUP		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PDM		✓																												✓			

Error Correction Coding Enable			LPGCR_RST	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
2	lp_tmr4	W1O	0	**Low Power TMR4 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See Operating Modes Wakeup Sources

In all operating modes other than ACTIVE, wakeup sources are required to re-enter ACTIVE Mode operation. The Table below shows available wakeup sources for each operating mode of the MAX78000.
Note: Each wakeup source must be enabled individually except for External Reset, which is hardware controlled.

Table 3-6: Wakeup Sources for Each Operating Mode in the MAX78000

Operating Mode	Any Peripheral Interrupts	External Reset	RV32	CNN	CNN FIFO	SPI1	SPI0	I2S	I2C2	I2C1	I2C0	LPUART0 (UART3)	UART2	UART1	UART0	LPTMR1 (TMR5)	LPTMR0 (TMR4)	TMR3	TMR2	TMR1	TMR0	LPWDT0 (WDT1)	WDT0	LPCMP3	LPCMP2	LPCMP1	LPCMP0	RTC	WUT	GPIO3	GPIO2	GPIO1	GPIO0
SLEEP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LPM		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
UPM		✓										✓				✓	✓					✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STANDBY		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BACKUP		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PDM		✓																												✓			

Device Resets for additional information.

Error Correction Coding Enable			LPGCR_RST	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
1	lp_wdt1	W1O	0	**Low Power WDT1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See Operating Modes Wakeup Sources In all operating modes other than ACTIVE, wakeup sources are required to re-enter ACTIVE Mode operation. The Table below shows available wakeup sources for each operating mode of the MAX78000. Note: Each wakeup source must be enabled individually except for External Reset, which is hardware controlled. *Table 3-6: Wakeup Sources for Each Operating Mode in the MAX78000* (see table below) Device Resets for additional information.

Table 3-6: Wakeup Sources for Each Operating Mode in the MAX78000

Operating Mode	Any Peripheral Interrupts	External Reset	RV32	CNN	CNN FIFO	SPI1	SPI0	I2S	I2C2	I2C1	I2C0	LPUART0 (UART3)	UART2	UART1	UART0	LPTMR1 (TMR5)	LPTMR0 (TMR4)	TMR3	TMR2	TMR1	TMR0	LPWDT0 (WDT1)	WDT0	LPCMP3	LPCMP2	LPCMP1	LPCMP0	RTC	WUT	GPIO3	GPIO2	GPIO1	GPIO0
SLEEP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LPM		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
UPM		✓										✓				✓	✓					✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STANDBY		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BACKUP		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PDM		✓																												✓			

Error Correction Coding Enable				**LPGCR_RST**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**	
0	lp_gpio2	W1O	0	**Low Power GPIO2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See Operating Modes Wakeup Sources	

In all operating modes other than ACTIVE, wakeup sources are required to re-enter ACTIVE Mode operation. The Table below shows available wakeup sources for each operating mode of the MAX78000.
Note: Each wakeup source must be enabled individually except for External Reset, which is hardware controlled.

Table 3-6: Wakeup Sources for Each Operating Mode in the MAX78000

Operating Mode	Any Peripheral Interrupts	External Reset	RV32	CNN	CNN FIFO	SPI1	SPI0	I2S	I2C2	I2C1	I2C0	LPUART0 (UART3)	UART2	UART1	UART0	LPTMR1 (TMR5)	LPTMR0 (TMR4)	TMR3	TMR2	TMR1	TMR0	LPWDT0 (WDT1)	WDT0	LPCMP3	LPCMP2	LPCMP1	LPCMP0	RTC	WUT	GPIO3	GPIO2	GPIO1	GPIO0
SLEEP	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LPM		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
UPM		✓										✓				✓	✓					✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STANDBY		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
BACKUP		✓																						✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PDM		✓																												✓			

Device Resets for additional information.

Table 3-23: Clock Disable Register

Clock Disable Register				**LPGCR_CLKDIS**	**[0x008]**
Bits	**Field**	**Access**	**Reset**	**Description**	
31:7	-	RO	0	**Reserved** Do not modify this field.	
6	lpcmp	R/W	0	**LPCMP 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	-	RO	0	**Reserved** Do not modify this field.	

Clock Disable Register			LPGCR_CLKDIS	[0x008]
Bits	**Field**	**Access**	**Reset**	**Description**
4	lpuart0	R/W	0	**LPUART0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
3	lptmr1	R/W	0	**LPTMR1 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
2	lptmr0	R/W	0	**LPTMR0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
1	lpwdt	R/W	0	**LPWDT0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Enabled 1: Disabled
0	lpgpio	R/W	0	**LPGPIO 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

3.10 Power Sequencer Registers (PWRSEQ)

See *Table 2-3: APB Peripheral Base* Address Map 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 3-24: Power Sequencer and Always-On Domain Register Summary*. 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 2-1. Field Access Definitions* 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 3-24: Power Sequencer and Always-On Domain Register Summary

Offset	Register	Name
[0x0000]	*PWRSEQ_LPCN*	*Low Power Control Register*
[0x0004]	*PWRSEQ_LPWKST0*	*Low Power GPIO0 Wakeup Status Flags*
[0x0008]	*PWRSEQ_LPWKEN0*	*Low Power GPIO0 Wakeup Enable Register*
[0x000C]	*PWRSEQ_LPWKST1*	*Low Power GPIO1 Wakeup Status Flags*
[0x0010]	*PWRSEQ_LPWKEN1*	*Low Power GPIO1 Wakeup Enable Register*
[0x0014]	*PWRSEQ_LPWKST2*	*Low Power GPIO2 Wakeup Status Flags*

Offset	Register	Name
[0x0018]	*PWRSEQ_LPWKEN2*	*Low Power GPIO2 Wakeup Enable Register*
[0x001C]	*PWRSEQ_LPWKST3*	*Low Power GPIO3 Wakeup Status Flags*
[0x0020]	*PWRSEQ_LPWKEN3*	*Low Power GPIO3 Wakeup Enable Register*
[0x0030]	*PWRSEQ_LPPWKST*	*Low Power Peripheral Wakeup Status Register*
[0x0034]	*PWRSEQ_LPPWKEN*	*Low Power Peripheral Wakeup Enable Register*
[0x0048]	*PWRSEQ_GPR0*	*General Purpose Register 0*
[0x004C]	*PWRSEQ_GPR1*	*General Purpose Register 1*

3.10.1 Power Sequencer Register Details

Table 3-25: Low Power Control Register

Low Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31	lpwkst_clr	R/W1	0	**Low Power Wakeup Status Register Clear** Write 1 to this field to clear the Low Power Wakeup Status registers: • *PWRSEQ_LPWKST0* • *PWRSEQ_LPWKST1* • *PWRSEQ_LPWKST2* • *PWRSEQ_LPWKST3* • *PWRSEQ_LPPWKST* 1: Write 1 to initiate a clear of all of the Low Power Wakeup Status registers. Hardware automatically clears this field when the registers are cleared.
30:12	-	DNM	0	**Reserved** Do not modify this field. This field must always be set to 0 for future compatibility.
11	bg_dis	R/W	1	**Band Gap Disable for *DEEPSLEEP* and *BACKUP* Mode** Setting this field to 1 (default) disables the Bandgap during ***DEEPSLEEP*** and ***BACKUP*** mode. 0: System Bandgap is on in ***DEEPSLEEP*** and ***BACKUP*** modes 1: System Bandgap is off in ***DEEPSLEEP*** and ***BACKUP*** modes.
10	-	R/W	0	**Reserved** Do not modify this field.
9	lpmfast_en	R/W	0	**Low Power Mode Mode Clock Select** If the ISO is selected (default), fast ***LPM*** entry is enabled. Setting the clock to INRO disables fast ***LPM*** entry. 0: ISO used for entering ***LPM*** (Fast Mode Enable). 1: INRO used for ***LPM*** entry (Fast Mode Disabled).
8	lpmclksel	R/W	1	**RISC-V Low Power Mode Clock Select** This field selects the clock source to use for the RV32 (CPU1) during ***LPM***. 0: PCLK is used as the RV32 (CPU1) system clock during ***LPM***. 1: ISO is used as the RV32 (CPU1) system clock during ***LPM***.

Low Power Control			PWRSEQ_LPCN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
7:4	-	R/W	0	**Reserved for Future Use** Do not modify this field. *Note: This field must be set to 0 to maintain future compatibility.*
3	ram3ret_en	R/W	0	**System RAM 3 Data Retention Enable for *BACKUP* Mode** Set this field to 1 to enable Data Retention for System RAM 3. See *Table 3-13 RAM Block Size and* Base Address for System RAM configuration. 0: Disable data retention for System RAM 3 address space in *BACKUP* mode. 1: Enable data retention for System RAM 3 address space in *BACKUP* mode.
2	ram2ret_en	R/W	0	**System RAM 2 Data Retention Enable for *BACKUP* Mode** Set this field to 1 to enable Data Retention for System RAM 2. See *Table 3-13 RAM Block Size and* Base Address for System RAM configuration. 0: Disable data retention for System RAM 2 address space in *BACKUP* mode. 1: Enable data retention for System RAM 2 address space in *BACKUP* mode.
1	ram1ret_en	R/W	0	**System RAM 1 Data Retention Enable for *BACKUP* Mode** Set this field to 1 to enable Data Retention for System RAM 1. See *Table 3-13 RAM Block Size and* Base Address for System RAM configuration. 0: Disable data retention for System RAM 1 address space in *BACKUP* mode. 1: Enable data retention for System RAM 1 address space in *BACKUP* mode.
0	ram0ret_en	R/W	0	**System RAM 0 Data Retention Enable for *BACKUP* Mode** Set this field to 1 to enable Data Retention for System RAM 1. See *Table 3-13 RAM Block Size and* Base Address for System RAM configuration. 0: Disable data retention for System RAM 0 address space in *BACKUP* mode. 1: Enable data retention for System RAM 0 address space in *BACKUP* mode.

Table 3-26: GPIO0 Low Power Wakeup Status Flags

GPIO0 Low Power Wakeup Status Flags			PWRSEQ_LPWKST0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	st	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 pin's corresponding bit in this register is set. The device will transition from a low-power mode to ACTIVE mode if the corresponding GPIO pin's interrupt enable bit is set in the *PWRSEQ_LPWKEN0* register. *Note: Clear this register before entering any low power mode.*

Table 3-27: GPIO0 Low Power Wakeup Enable Registers

GPIO0 Low Power Wakeup Enable			PWRSEQ_LPWKEN0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	en	R/W	0	**GPIO0 Pin Wakeup Interrupt Enable** Setting a GPIO0 pin's bit in this register will cause an interrupt to be generated that will wakeup the device from any low power mode to ACTIVE mode. A wakeup event sets the corresponding GPIO0's bit in the *PWRSEQ_LPWKST0* register, enabling determination of which GPIO0 pin triggered the wakeup event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the **MAX78000** to wake up from a low power mode on a GPIO pin transition, first set the "GPIO Wakeup Enable" register bit GCR_PM.gpiowken = 1.*

Table 3-28: GPIO1 Low Power Wakeup Status Flags

GPIO1 Low Power Wakeup Status Flags			PWRSEQ_LPWKST1	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1C	0	**Reserved** Bits corresponding to unimplemented GPIO are ignored.
9:0	st	R/W1C	0	**GPIO1 Pin Wakeup Status Flag** Whenever a GPIO1 pin, in any power mode, transitions from low-to-high or high-to-low, the pin's corresponding bit in this register is set. The device will transition from a low-power to ACTIVE mode if the corresponding interrupt enable bit is set in *PWRSEQ_LPWKEN1*. *Note: Clear this register before entering any low power mode.*

Table 3-29: GPIO1 Low Power Wakeup Enable Registers

GPIO1 Low Power Wakeup Enable			PWRSEQ_LPWKEN1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:10		RO	0	**Reserved** Bits corresponding to unimplemented GPIO are ignored.
9:0	en	R/W	0	**GPIO1 Pin Wakeup Interrupt Enable** Setting a GPIO1 pin's bit in this register will cause an interrupt to be generated that will wakeup the device from any low power mode to ACTIVE mode. A wakeup event sets the corresponding GPIO1's bit in the *PWRSEQ_LPWKST1* register, enabling determination of which GPIO1 pin triggered the wakeup event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the **MAX78000** to wake up from a low power mode on a GPIO pin transition, first set the "GPIO Wakeup Enable" register bit GCR_PM.gpiowken = 1.*

Table 3-30: GPIO2 Low Power Wakeup Status Flags

GPIO2 Low Power Wakeup Status Flags			PWRSEQ_LPWKST2	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8		R/W1C	0	**Reserved** Bits corresponding to unimplemented GPIO are ignored.

GPIO2 Low Power Wakeup Status Flags			PWRSEQ_LPWKST2	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	st	R/W1C	0	**GPIO2 Pin Wakeup Status Flag** Whenever a GPIO2 pin, in any power mode, transitions from low-to-high or high-to-low, the corresponding bit in this register is set. Bits corresponding to unimplemented GPIO are ignored. *Note: The device will transition from a low-power to ACTIVE mode if the corresponding interrupt enable bit is set in PWRSEQ_LPWKEN. This register should be cleared before entering any low power mode.*

Table 3-31: GPIO2 Low Power Wakeup Enable Registers

GPIO2 Low Power Wakeup Enable			PWRSEQ_LPWKEN2	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8		RO	0	**Reserved** Bits corresponding to unimplemented GPIO are ignored.
7:0	en	R/W	0	**GPIO2 Pin Wakeup Interrupt Enable** Setting a GPIO2 pin's bit in this register will cause an interrupt to be generated that will wakeup the device from any low power mode to ACTIVE mode. A wakeup event sets the corresponding GPIO2's bit in the *PWRSEQ_LPWKST2* register, enabling determination of which GPIO2 pin triggered the wakeup event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the **MAX78000** to wake up from a low power mode on a GPIO pin transition, first set the "GPIO Wakeup Enable" register bit GCR_PM.gpiowken = 1.*

Table 3-32: GPIO3 Low Power Wakeup Status Flags

GPIO3 Low Power Wakeup Status Flags			PWRSEQ_LPWKST3	[0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2		RO	0	**Reserved** Bits corresponding to unimplemented GPIO are ignored.
1:0	st	R/W1C	0	**GPIO3 Pin Wakeup Status Flag** Whenever a GPIO3 pin, in any power mode, transitions from low-to-high or high-to-low, the corresponding bit in this register is set. Bits corresponding to unimplemented GPIO are ignored. *Note: The device will transition from a low-power to ACTIVE mode if the corresponding interrupt enable bit is set in PWRSEQ_LPWKEN. This register should be cleared before entering any low power mode.*

Table 3-33: GPIO3 Low Power Wakeup Enable Registers

GPIO3 Low Power Wakeup Enable			PWRSEQ_LPWKEN3	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2		RO	0	**Reserved** Do not modify this field.

GPIO3 Low Power Wakeup Enable			PWRSEQ_LPWKEN3	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
1:0	en	R/W	0	**GPIO3 Pin Wakeup Interrupt Enable** Setting a GPIO3 pin's bit in this register will cause an interrupt to be generated that will wakeup the device from any low power mode to ACTIVE mode. A wakeup event sets the corresponding GPIO3's bit in the *PWRSEQ_LPWKST3* register, enabling determination of which GPIO3 pin triggered the wakeup event. Bits corresponding to unimplemented GPIO are ignored. *Note: To enable the **MAX78000** to wake up from a low power mode on a GPIO pin transition, first set the "GPIO Wakeup Enable" register bit GCR_PM.gpiowken = 1.*

Table 3-34: Low Power Peripheral Wakeup Status Flags

Low Power Peripheral Wakeup Status Flags			PWRSEQ_LPPWKST	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:18		RO	0	**Reserved** Do not modify this field.
17	reset	R/W1C	0	**Reset Detected Wakeup Flag** This field is set when an external reset caused the wakeup event.
16	backup	R/W1C	0	**BACKUP Mode Wakeup Flag** This field is set when the device wakes up from BACKUP mode.
15:5	-	RO	0	**Reserved** Do not modify this field.
4	aincomp0	R/W1C	0	**Analog Input Comparator Wakeup Flag** This field is set if the wakeup event was the result of an input comparator trigger event.
3:0	-	RO	0	**Reserved** Do not modify this field.

Table 3-35: Low Power Peripheral Wakeup Enable Registers

Low Power Peripheral Wakeup Enable			PWRSEQ_LPPWKEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:27		RO	0	**Reserved** Do not modify this field.
26	lpcmp	R/W	0	**LPCMP IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the LPCMP IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
25	spi0	R/W	0	**SPI0 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the SPI0 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.

Low Power Peripheral Wakeup Enable			PWRSEQ_LPPWKEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
24	i2s	R/W	0	**I2S IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the I2S IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
23	i2c2	R/W	0	**I2C2 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the I2C2 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
22	i2c1	R/W	0	**I2C1 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the I2C1 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
21	i2c0	R/W	0	**I2C0 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the I2C0 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
20	uart3	R/W	0	**UART3 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the UART3 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
19	uart2	R/W	0	**UART2 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the UART2 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
18	uart1	R/W	0	**UART1 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the UART1 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
17	uart0	R/W	0	**UART0 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the UART0 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
16	tmr5	R/W	0	**TMR5 (LPTMR1) IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the TMR5 (LPTMR1) IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
15	tmr4	R/W	0	**TMR4 (LPTMR0) IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the TMR4 (LPTMR0) IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
14	tmr3	R/W	0	**TMR3 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the TMR3 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.

Low Power Peripheral Wakeup Enable			PWRSEQ_LPPWKEN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
13	tmr2	R/W	0	**TMR2 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the TMR2 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
12	tmr1	R/W	0	**TMR1 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the TMR1 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
11	tmr0	R/W	0	**TMR0 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the TMR0 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
10	cpu1	R/W	0	**CPU1 (RISC-V) IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the CPU1 (RISC-V) IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
9	wdt1	R/W	0	**WDT1 (LPWDT0) IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the WDT1 (LPWDT0) IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
8	wdt0	R/W	0	**WDT0 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the WDT0 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
7:5	-	RO	0	**Reserved** Do not modify this field.
4	aincomp0	R/W	0	**AIN Comparator 0 IRQ Wakeup Enable** Set this field to 1 to enable wakeup events from the AINCOMP0 IRQ. 0: Disable wakeup on IRQ. 1: Enable wakeup on IRQ.
3:0	-	RO	0	**Reserved** Do not modify this field.

Table 3-36: Low Power General Purpose Register 0

Low Power General Purpose Register 0			PWRSEQ_GPR0	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	gpf	R/W	0	**General Purpose Field** This register can be used as a general-purpose register by software.

Table 3-37: Low Power General Purpose Register 1

Low Power General Purpose Register 1			PWRSEQ_GPR1	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	gpf	R/W	0	**General Purpose Field** This register can be used as a general-purpose register by software

3.11 Global Control Registers (GCR)

See *Table 2-3: APB Peripheral Base* Address Map 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 3-38: Global Control Register Summary*. 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 2-1. Field Access Definitions* 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 General Control Registers are only reset on a System Reset or Power-On Reset. A Soft Reset or Peripheral Reset does not affect these registers.

Table 3-38: Global Control Register Summary

Offset	Register	Description
[0x0000]	*GCR_SYSCTRL*	*System Control Register*
[0x0004]	*GCR_RSTR0*	*Reset Register 0*
[0x0008]	*GCR_CLKCN*	*Clock Control Register*
[0x000C]	*GCR_PM*	*Power Management Register*
[0x0018]	*GCR_PCKDIV*	*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_RSTR1*	*Reset Register 1*
[0x0048]	*GCR_PCLKDIS1*	*Peripheral Clocks Disable 1*
[0x004C]	*GCR_EVENT_EN*	*Event Enable Register*
[0x0050]	*GCR_REVISION*	*Revision Register*
[0x0054]	*GCR_SYSINTEN*	*System Status Interrupt Enable*
[0x0064]	*GCR_ECCERR*	*Error Correction Coding Error Register*
[0x0068]	*GCR_ECCCED*	*Error Correction Coding Correctable Error Detected*
[0x006C]	*GCR_ECCIE*	*Error Correction Coding Interrupt Enable Register*
[0x0070]	*GCR_ECCADDR*	*Error Correction Coding Error Address Register*
[0x0080]	*GCR_GPR*	*General Purpose Register 0*

3.11.1 Global Control Register Details (GCR)

Table 3-39: System Control Register

System Control			GCR_SYSCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:18	-	RO	0	**Reserved** Do not modify this field.
17:16	ovr	R/W	0b10	**Operating Voltage Range** Set this field to match the V_{CORE} voltage to enable the on-chip RAM to operate at the optimal timing range. 0b00: 0.9V ± 10% 0b01: 1.0V ± 10% 0b10: 1.1V ± 10% 0b11: Reserved for Future Use
15	chkres	R	0	**ROM Checksum Calculation Pass/Fail** This is the result after setting bit *GCR_SYSCTRL.cchk*. This bit is only valid after the ROM checksum is complete and *GCR_SYSCTRL.cchk* is cleared. 0: Pass 1: Fail
14	swd_dis	R/W	0	**Serial Wire Debug Disable** This bit is used to disable the serial wire debug interface. 0: SWD Disabled 1: SWD Enabled *Note: This bit is only writeable if (FMV lock word is not programmed) or if (ICE lock word is not programmed and the ROM_DONE bit is not set).*
13	cchk	R/W	0	**Calculate ROM Checksum** This bit is self-clearing when the ROM checksum calculation is complete, and the result is available at bit *GCR_SYSCTRL.chkres*. Writing a 0 has no effect. 0: No operation 1: Start ROM checksum calculation
12	rom_done	R/W	0	**ROM Start Code Status** Used to disable SWD interface during system initialization procedure. 0: ROM Start Code not completed 1: ROM Start Code not completed *Note: If FMV lock word is programmed, software can only set this bit. Subsequently, this bit cannot be cleared by software.*
11:7	-	RO	0	**Reserved** Do not modify this field.
6	icc0_flush	R/W	0	**ICC0 Code Cache Flush** Write 1 to flush the code cache and the instruction buffer for CPU0. This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect and does not stop a cache in progress. 0: Code cache flush not in progress. 1: Code cache flush in progress.

System Control			GCR_SYSCTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
5	-	RO	0	**Reserved** Do not modify this field.
4	flash_page_flip	R/*	0	**Flash Page Flip Flag** Flips the bottom and top halves of Flash memory. This bit is controlled by hardware. *Note: Software/Firmware should not change the state of this bit during normal operation. Any change to this bit also flushes both code and data caches.* 0: Physical layout matches logical layout 1: Top and Bottom halves flipped
3:1	-	RO	0b001	**Reserved** Do not modify this field.
0	bstapen	R/W	*	**Boundary Scan Tap Enable** Reserved. Do not modify this field. Reset value matches *GCR_SYSST.icelock*.

Table 3-40: Reset Register 0

Reset 0			GCR_RSTR0	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31	system	R/W	0	**System Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *System Reset* for additional information.
30	periph	R/W	0	**Peripheral Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. *Note: Watchdog Timers, GPIO Ports, the AoD, RAM Retention and the General Control Registers (GCR) are unaffected. See Peripheral Reset for additional information.*
29	srst	R/W	0	**Soft Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete. See *Soft Reset* for additional information.
28	uart2	R/W	0	**UART2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
27	-	R/W	0	**Reserved** Do not modify this field.
26	adc	R/W	0	**ADC Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
25	cnn	R/W	0	**CNN Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
24	trng	R/W	0	**TRNG Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
23	-	R/W	0	**Reserved** Do not modify this field.

Reset 0				GCR_RSTR0 [0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
22	smphr	R/W	0	**Semaphore Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
21:18	-	R/W	0	**Reserved** Do not modify this field.
17	rtc	R/W	0	**RTC Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
16	i2c0	R/W	0	**I2C0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
15:14	-	RO	0	**Reserved** Do not modify this field.
13	spi0	R/W	0	**SPI0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
12	uart1	R/W	0	**UART1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
11	uart0	R/W	0	**UART0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
10:9	-	R/W	0	**Reserved** Do not modify this field.
8	timer3	R/W	0	**TMR3 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
7	timer2	R/W	0	**TMR2 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
6	timer1	R/W	0	**TMR1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
5	timer0	R/W	0	**TMR0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
4	-	RO	-	**Reserved** Do not modify this field.
3	gpio1	R/W	0	**GPIO1 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
2	gpio0	R/W	0	**GPIO0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
1	wdt0	R/W	0	**Watchdog Timer 0 Reset** Write 1 to reset. This field is cleared by hardware when the reset is complete.
0	dma	R/W	0	**DMA Access Block** Write 1 to reset. This field is cleared by hardware when the reset is complete.

Table 3-41: Clock Control Register

Clock Control				GCR_CLKCN	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:30	-	RO	0b10	**Reserved** Do not modify this field.	
29	inro_rdy		0	**8kHz Internal Nano-Ring Oscillator (INRO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.	
28	ibro_rdy	R	0	**7.3728MHz Internal Baud Rate Oscillator (IBRO) Ready Status** 0: Not ready. 1: Oscillator ready.	
27	ipo_rdy	R	0	100MHz **Internal Primary Oscillator (IPO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.	
26	iso_rdy	R	0	60MHz **Internal Secondary Oscillator (ISO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.	
25	ertco_rdy	R	0	**32.768kHz External RTC Oscillator (ERTCO) Ready Status** 0: Not ready or not enabled. 1: Oscillator ready.	
24:21	-	RO	0	**Reserved** Do not modify this field.	
20	ibro_en	RO	1	**7.3728MHz IBRO Enable** The IBRO is always enabled. 1: Enabled and ready when *GCR_CLKCN.ibro_rdy* = 1.	
19	ipo_en	R/W	0	100MHz **IPO Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCN.ipo_rdy* = 1.	
18	iso_en	R/W	1	60MHz **ISO Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCN.iso_rdy* = 1	
17	ertco_en	R/W	0	**32.768kHz ERTCO Enable** 0: Disabled 1: Enabled and ready when *GCR_CLKCN.ertco_rdy* = 1.	
16:14	-	RO	0	**Reserved** Do not modify this field.	
13	ckrdy	R	0	**SYS_OSC Select Ready** When SYS_OSC is changed by modifying *GCR_CLKCN.clksel*, there is a delay until the switchover is complete. This bit is cleared until the switchover is complete. 0: Switch to new clock source not yet complete. 1: SYS_OSC is clock source selected in *GCR_CLKCN.clksel*.	
12	-	RO	0	**Reserved** Do not modify this field.	

Clock Control			GCR_CLKCN	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
11:9	clksel	R/W	4	**System Oscillator Source Select** Selects the system oscillator (SYS_OSC) source used to generate the system clock (SYS_CLK). Modifying this field immediately clears *GCR_CLKCN.ckrdy*. 0: ISO 1: Reserved 2: Reserved 3: INRO 4: IPO 5: IBRO 6: ERTCO 7: External Clock, EXT_CLK1, P0.3 AF1
8:6	psc	R/W	0	**System Oscillator Prescaler** Sets the divider for generating SYS_CLK from the selected SYS_OSC as shown in the following equation: $SYS\_CLK = \frac{SYS\_OSC}{2^{psc}}$
5:0	-	RO	0b001000	**Reserved** Do not modify this field.

Table 3-42: Power Management Register

Power Management			GCR_PM	0x000C
Bits	**Field**	**Access**	**Reset**	**Description**
31:18	-	RO	0	**Reserved** Do not modify this field.
17	ibro_pd	R/W	1	**IBRO Power Down** Set this field to 1 to power down the IBRO during ***DEEPSLEEP***. 0: IBRO enabled during ***DEEPSLEEP*** 1: IBRO powered down during ***DEEPSLEEP***
16	ipo_pd	R/W	1	**IPO Power Down** Set this field to 1 to power down the IPO during ***DEEPSLEEP***. 0: IPO enabled during ***DEEPSLEEP*** 1: IPO powered down during ***DEEPSLEEP***
15	iso_pd	R/W	1	**ISO Power Down** This bit selects the power state in ***DEEPSLEEP*** mode for the ISO oscillator. 0: Active 1: Powered down in ***DEEPSLEEP***
14:10	-	DNM	0b11100	**Reserved** Do not modify this field.
9	aincompwken	R/W	0	**Analog Input Comparator Wakeup Enable** This bit enables the Analog Input Comparator IRQ to wake the device from ***SLEEP***, ***DEEPSLEEP***, or ***BACKUP*** modes.
8	-	RO	0	**Reserved** Do not modify this field.

Power Management			GCR_PM	0x000C
Bits	**Field**	**Access**	**Reset**	**Description**
7	wutwken	R/W	0	**Wakeup Time Enable** Set this field to 1 to enable the Wakeup Timer as a wakeup source. The Wakeup Timer will wake the device from ***SLEEP*** or ***DEEPSLEEP*** or ***BACKUP*** modes. 0: Wakeup source disabled 1: Wakeup source enabled.
6	-	RO	0	**Reserved** Do not modify this field.
5	rtcwken	R/W	0	**RTC Alarm Wakeup Enable** Set this field to 1 to enable an RTC alarm to wake the device. The RTC alarm will wake the device from ***SLEEP*** or ***DEEPSLEEP*** or ***BACKUP*** modes. 0: Wakeup source disabled 1: Wakeup source enabled
4	gpiowken	R/W	0	**GPIO Wakeup Enable** Set this field to 1 to enable all GPIO pins as potential wakeup sources. Any GPIO configured for wakeup will wake the device from ***SLEEP*** or ***DEEPSLEEP*** or ***BACKUP*** modes. 0: Wakeup source disabled 1: Wakeup source enabled
3:0	mode	R/W	0	**Operating Mode** This field controls the operating state of the device. 0b0000: ***ACTIVE*** Mode 0b0001: ***SLEEP*** CM4, RISCV Mode 0b0010: ***STANDBY*** Mode 0b0100: ***BACKUP*** Mode 0b1000: ***LPM*** or CM4 DEEPSLEEP Mode 0b1001: ***UPM*** 0b1010: ***PDM***

Table 3-43: Peripheral Clock Divisor Register

Peripheral Clocks Divisor			GCR_PCKDIV	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:18	-	RO	-	**Reserved** Do not modify this field.
17	cnnclksel	R/W	0	**CNN Peripheral Clock Select** Set this field to select the clock source for the CNN peripheral. 0: PCLK 1: ISO

Peripheral Clocks Divisor			GCR_PCKDIV	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
16:14	cnnfrq	R/W		**CNN Peripheral Clock Frequency Divider** This field is used as a divider of the CNN Peripheral Clock selected by the *GCR_PCKDIV.cnnclksel*. 0: $CNN\ Clock/_2$ 1: $CNN\ Clock/_4$ 2: $CNN\ Clock/_8$ 3: $CNN\ Clock/_{16}$ 4: Reserved 5: Reserved 6: Reserved 7: $CNN\ Clock/_1$
13:10	adcfrq	R/W	0	**ADC Peripheral Clock Frequency Select** Configures the frequency of the ADC peripheral clock from the PCLK. 0: Reserved 1: Reserved $2-15$: $f_{adc\_clk} = {f_{PCLK}}/_{adcfrq}$
9:0	-	RO	-	**Reserved** Do not modify this field.

Table 3-44: Peripheral Clock Disable Register 0

Peripheral Clocks Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	R/W	1	**Reserved** Do not modify this field.
29	pt	R/W	1	**Pulse Train 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: Clock enabled. 1: Clock 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: Clock enabled 1: Clock disabled
27:26	-	RO	1	**Reserved** Do not modify this field.
25	cnn	R/W	1	**CNN Clock Disable** Disabling a clock disables functionality while also saving power. Read and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled 1: Clock disabled

Peripheral Clocks Disable 0			GCR_PCLKDIS0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
24	-	RO	1	**Reserved** Do not modify this field.
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: Clock enabled 1: Clock disabled
22:19	-	RO	1	**Reserved** Do not modify this field.
18	timer3	R/W	1	**TMR3 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled 1: Clock disabled
17	timer2	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: Clock enabled 1: Clock disabled
16	timer1	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: Clock enabled 1: Clock disabled
15	timer0	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: Clock enabled 1: Clock disabled
14	-	RO	1	**Reserved** Do not modify this field.
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: Clock enabled 1: Clock disabled
12:11	-	RO	1	**Reserved** Do not modify this field.
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: Clock enabled 1: Clock disabled

Peripheral Clocks Disable 0				GCR_PCLKDIS0 [0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
9	uart0	R/W	1	**UART0 Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled 1: Clock disabled
8	-	RO	1	**Reserved** Do not modify this field.
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: Clock enabled 1: Clock 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: Clock enabled 1: Clock 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: Clock enabled 1: Clock disabled
4:2	-	RO	1	**Reserved** Do not modify this field.
1	gpio1	R/W	1	**GPIO1 Port and Pad Logic Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled 1: Clock disabled
0	gpio0	R/W	1	**GPIO0 Port and Pad Logic Clock Disable** Disabling a clock disables functionality while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained. 0: Clock enabled 1: Clock disabled

Table 3-45: Memory Clock Control Register

Memory Clock Control				GCR_MEMCTRL [0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved** Do not modify this field.
16	ram0	R/W	0	**Sysram0 ECC Enable** Set this field to 1 to enable ECC for System RAM 0. 0: Sysram0 Active, ECC disabled. 1: Sysram0 Active, ECC enabled.

Memory Clock Control			GCR_MEMCTRL	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
15:3	-	RO	0	**Reserved** Do not modify this field.
2:0	fws	R/W	5	**Program Flash Wait States** Number of wait-state cycles per Flash code read access. 0: Invalid 1 – 7: Number of Flash code access wait states *Note: For the IPO and ISO clocks the minimum wait state is 2.* *Note: For all other clock sources the minimum wait state is 1.*

Table 3-46: Memory Zeroize Control Register

Memory Zeroize			GCR_MEMZ	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:15	-	RO	-	**Reserved** Do not modify this field.
14	icache1z	R/W	0	**CPU1 ICC1 Cache Data and Tag Zeroization** Write 1 to initiate the operation. Only valid on devices with optional CPU1. 0: Operation complete. 1: Operation in progress.
13:4	-	RO	0	**Reserved** Do not modify this field.
3	sram3	R/W1C	0	**Sysram3 Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
2	sram2	R/W1C	0	**Sysram2 Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
1	sram1	R/W1C	0	**Sysram1 Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.
0	sram0	R/W1C	0	**Sysram0 Zeroization** Write 1 to initiate the operation. This field is automatically cleared by hardware on completion. 0: Operation complete. 1: Operation in progress.

Table 3-47: System Status Flag Register

System Status Flag			GCR_SYSST	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved** Do not modify this field.
0	icelock	R	0	**Arm ICE Lock Status Flag** 0: Arm ICE is unlocked (enabled) 1: Arm ICE is locked (disabled)

Table 3-48: Reset Register 1

Reset 1			GCR_RSTR1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
31	cpu1	RO	0	**CPU1 Reset** Write 1 to initiate the reset operation. 0: Operation complete. 1: Operation in progress.
30:26	-	RO	0	**Reserved** Do not modify this field.
25	simo	R/W	0	**Single Inductor Multiple Output Block Reset** Write 1 to initiate the reset operation. 0: Operation complete. 1: Operation in progress.
24	dvs	R/W	0	**Dynamic Voltage Scaling Controller Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
23:22	-	RO	0	**Reserved** **Do not modify this field.**
21	rpu	R/W	0	**Resource Protection Unit Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
20	i2c2	R/W	0	**I^{2}C2 Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
19	i2s	R/W	0	**Audio Interface Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
18:17	-	R/W	0	**Reserved** Do not modify this field.

Reset 1			GCR_RSTR1	[0x0044]
Bits	**Field**	**Access**	**Reset**	**Description**
16	smphr	R/W	0	**Semaphore Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
15:12	-	R/W	-	**Reserved** Do not modify this field.
11	spi0	R/W	0	**SPI0 AHB Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
10	aes	R/W	0	**AES Block Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
9	crc	R/W	0	**CRC Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
8	-	R/W	0	**Reserved** Do not modify this field.
7	owire	R/W	0	**One-Wire Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
6:2	-	RO	0	**Reserved** Do not modify this field.
1	pt	R/W	0	**Pulse Train Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.
0	i2c1	R/W	0	**I²C1 Reset** Write 1 to initiate the operation. 0: Operation complete. 1: Operation in progress.

Table 3-49: Peripheral Clock Disable Register 1

Peripheral Clock Disable 1			GCR_PCLKDIS1	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31	cpu1	R/W	1	**CPU1 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**
30:28	-	R/W1	1	**Reserved**
27	wdt0	R/W	1	**Watchdog Timer 0 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.
26:25	-	R/W1	1	**Reserved**
24	i2c2	R/W	1	**I^{2}C2 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.
23	i2s0	R/W	1	**I^{2}S Audio Interface 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.
22:16	-	R/W1	1	**Reserved**
15	aes	R/W	1	**AES Block Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
14	crc	R/W	1	**CRC Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
13	owire	R/W	1	**One-Wire 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.
12:10	-	R/W1	1	**Reserved** Do not modify this field.
9	smphr	R/W	1	**Semaphore 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.
8:3	-	R/W1	1	**Reserved** Do not modify this field.

Peripheral Clock Disable 1		GCR_PCLKDIS1		[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
2	trng	R/W	1	**TRNG Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
1	uart2	R/W	1	**UART2 Clock Disable** Disabling the clock disables functionality while also saving power. Associated register states are retained but read and write access is blocked. 0: Enabled. 1: Disabled.
0	-	R/W1	1	**Reserved** Do not modify this field.

Table 3-50: Event Enable Register

Event Enable		GCR_EVENT_EN		[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved** Do not modify this field.
2	cpu0tx	R/W	0	**CPU0 TXEV Event Enable** When set, a TXEV event from CPU0/CPU1 generates an RXEV to wake-up CPU0 from a low power mode entered with a WFE instruction. 0: Disabled. 1: Enabled.
1	-	RO	0	**Reserved** Do not modify this field.
0	cpu0dma	R/W	0	**CPU0 DMA CTZ Wake-Up Enable** Allows a DMA0 CTZ event to generate an RXEV to wake-up CPU0 from a low power mode entered with a WFE instruction. 0: Disabled. 1: Enabled.

Table 3-51: Revision Register

Revision		GCR_REVISION		[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved**
15:0	revision	R	*	**Device Revision** Returns the chip revision ID as packed BCD. For example, 0xA1 would indicate the device is revision A1.

Table 3-52: System Status Interrupt Enable Register

System Status Interrupt Enable				GCR_SYSINTEN	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:1	-	RO	-	**Reserved** Do not modify this field.	
0	iceulie	R/W	0	**Arm ICE Unlocked Interrupt Enable** Generates an interrupt if the *GCR_SYSST.icelock* is set. 0: Disabled. 1: Enabled.	

Table 3-53: Error Correction Coding Error Register

Error Correction Coding Error				GCR_ECCERR	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:1	-	RO	0	**Reserved** Do not modify this field.	
0	ram0	R/W1C	0	**System RAM0 ECC Error** Write to 1 to clear the flag. 0: No error 1: Error	

Table 3-54: Error Correction Coding Correctable Error Detected Register

Error Correction Coding Correctable Error Detected				GCR_ECCCED	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:1	-	RO	0	**Reserved** Do not modify this field.	
0	ram0	R/W1C	0	**System RAM0 Correctable ECC Error Detected** When set, indicates that there is a single correctable error in the RAM block. Write to 1 to clear the flag. 0: Error 1: No Error	

Table 3-55: Error Correction Coding Interrupt Enable Register

Error Correction Coding Interrupt Enable				GCR_ECCIE	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:1	-	RO	0	**Reserved** Do not modify this field.	
0	ram0	R/W	0	**System RAM0 ECC Error Interrupt Enable** 0: Disabled 1: Enabled	

Table 3-56: Error Correction Coding Error Address Register

Error Correction Coding Error Address				GCR_ECCADDR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**	
31	tagramerr	R	0	**ECC Error Address/TAG RAM Error** Data depends on which block has reported the error. If sysram0 then this bit represents the bit of the AMBA address of read which produced the error. If the error is the cache, then this bit is set as shown below: 0: No error 1: Tag Error. The error is in the TAG RAM	
30	tagrambank	R	0	**ECC Error Address/TAG RAM Error Bank** Data depends on which block has reported the error. If sysram0 then this bit represents the bit of the AMBA address of read which produced the error. If the error is from the cache, then this bit is set as shown below: 0: Error is in TAG RAM Bank 0 1: Error is in TAG RAM Bank 1	
29:16	tagramaddr	R	0	**ECC Error Address/TAG RAM Error Address** Data depends on which block has reported the error. If sysram0 these bits represents the bits of the AMBA address of read which produced the error. If the error is from the cache, then this field is set as shown below: [TAG ADDRESS]: Represents the TAG RAM Address	
15	dataramerr	R	0	**ECC Error Address/DATA RAM Error Address** Data depends on which block has reported the error. If sysram0 then this bit represents the bit of the AMBA address of read which produced the error. If the error is from the cache, then this bit is set as shown below: 0: No error 1: DATA RAM Error. The error is in the Data RAM	
14	datarambank	R	0	**ECC Error Address/DATA RAM Error Bank** Data depends on which block has reported the error. If sysram0 then this bit represents the bits of the AMBA address of read which produced the error. If the error is from the cache, then this bit is set as shown below: 0: Error is in DATA RAM Bank 0 1: Error is in DATA RAM Bank 1	
13:0	dataramaddr	R	0	**ECC Error Address/TAG RAM Error Address** Data depends on which block has reported the error. This field represents the bits of the AMBA address of read which produced the error. If the error is from the caches, then this field is set as shown below: [DATA ADDRESS]: Represents the DATA RAM Error Address	

Table 3-57: General Purpose Register

General Purpose				GCR_GPR	[0x0080]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	gpr	R/W	0	**General Purpose Register** General purpose register	

3.12 Error Correction Coding (ECC)

See *Table 2-3: APB Peripheral Base* Address Map for the Error Correction Coding Enable Peripheral Address.

Table 3-58: Error Correction Coding Enable Register Summary

Offset	Register Name	Access	Description
[0x0000]	*ECC_EN*	R/W	*Error Correction Coding Enable*

3.12.1 Error Correction Coding Enable Register Details

Table 3-59: Error Correction Coding Enable Register

Error Correction Coding Enable		ECC_EN		[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved**
8	ram	R/W	0	**System RAM ECC Enable** Set this field to 1 to enable ECC for sysram0. 0: Disabled 1: Enabled
7:0	-	RO	0	**Reserved**

3.13 System Initialization Registers (SIR)

See *Table 2-3: APB Peripheral Base* Address Map 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 3-60: System Initialization Register Summary*. 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 2-1. Field Access Definitions* 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 3-60: System Initialization Register Summary

Offset	Register Name	Access	Description
[0x0000]	*SIR_STAT*	RO	*System Initialization Status Register*
[0x0004]	*SIR_ADDR_ER*	RO	*System Initialization Address Error Register*
[0x0100]	*SIR_FSTAT*	RO	*System initialization Function Status Register*
[0x0104]	*SIR_SFSTAT*	RO	*System initialization Security Function Status Register*

3.13.1 System Initialization Register Details

Table 3-61: System Initialization Status Register

System Initialization Status		SIR_STAT		[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved for Future Use** Do not modify this field.

System Initialization Status			SIR_STAT	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
1	cfg_err	RO	See Description	**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. Please contact Maxim Integrated technical support for additional assistance providing the address contained in SIR_ADDR_ER.addr.*
0	cfg_valid	RO	See Description	**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 during system initialization. Please contact Maxim Integrated technical support for additional assistance.*

Table 3-62: System Initialization Address Error Register

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

Table 3-63: System Initialization Function Status Register

System Initialization Function Status Register			SIR_FSTAT	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved** Do not modify this field.
7	smphr	RO	See Description	**Semaphore Block** This field indicates if the device includes the Semaphore Block. 0: Block is not available. 1: Block is available.
6:3	--	RO	0	**Reserved** Do not modify this field.
2	adc	RO	See Description	**ADC** This field indicates if the device includes the ADC. 0: Block is not available. 1: Block is available.
1	-	RO	0	**Reserved** Do not modify this field.

System Initialization Function Status Register			SIR_FSTAT	[0x0100]
Bits	**Name**	**Access**	**Reset**	**Description**
0	fpu	RO	See Description	**FPU** This field indicates if the device includes the FPU. 0: Block is not available. 1: Block is available.

Table 3-64: System Initialization Security Function Status Register

System Initialization Security Function Status Register			SIR_SFSTAT	[0x0104]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved** Do not modify this field.
3	aes	RO	See Description	**AES** This field indicates if the device includes the AES block. 0: Block is not available. 1: Block is available.
2	trng	RO	See Description	**TRNG** This field indicates if the device includes the TRNG block. 0: Block is not available. 1: Block is available.
1:0	-	RO	0	**Reserved** Do not modify this field.

3.14 Function Control Registers

See *Table 2-3: APB Peripheral Base* Address Map 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 3-65: Function Control Register Summary*. 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 2-1. Field Access Definitions* 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 3-65: Function Control Register Summary

Offset	Register	Description
[0x0000]	*FCR_FCTRL0*	*Function Control 0 Register*
[0x0004]	*FCR_AUTOCAL0*	*Automatic Calibration 0 Register*
[0x0008]	*FCR_AUTOCAL1*	*Automatic Calibration 1 Register*
[0x000C]	*FCR_AUTOCAL2*	*Automatic Calibration 2 Register*
[0x0010]	*FCR_URVBOOT*	*RISC-V Boot Address Register*
[0x0014]	*FCR_URVCTRL*	*RISC-V Control Register*

3.14.1 Function Control Register Details

Table 3-66: Function Control 0 Register

Function Control 0			**FCR_FCTRL0**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:26	-	RO	0	**Reserved** Do not modify this field.
25	i2c2_scl_filter_en	R/W	0	**I^2C2 SCL Glitch Filter Enable** 0: Disabled 1: Enabled
24	i2c2_sda_filter_en	R/W	0	**I^2C2 SDA Glitch Filter Enable** 0: Disabled 1: Enabled
23	i2c1_scl_filter_en	R/W	0	**I^2C1 SCL Glitch Filter Enable** 0: Disabled 1: Enabled
22	i2c1_sda_filter_en	R/W	0	**I^2C1 SDA Glitch Filter Enable** 0: Disabled 1: Enabled
21	i2c0_scl_filter_en	R/W	0	**I^2C0 SCL Glitch Filter Enable** 0: Disabled 1: Enabled
20	i2c0_sda_filter_en	R/W	0	**I^2C0 SDA Glitch Filter Enable** 0: Disabled 1: Enabled
19:0	-	RO	0	**Reserved** Do not modify this field.

Table 3-67: Automatic Calibration 0 Register

Automatic Calibration 0			**FCR_AUTOCAL0**	**[0x0004]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:23	trim	RO	0	**IPO Trim Value** Initial factory trim value for the IPO.
22:20	-	RO		**Reserved** Do not modify this field.
19:8	gain	R/W	0	**IPO Trim Adaptation Gain** Describe this field!
7:5	-	RO	0	**Reserved** Do not modify this field.
4	atomic	R/W1	0	**IPO Trim Atomic Start** Set this bit to start an atomic automatic calibration of the IPO. The calibration will run for *FCR_AUTOCAL2.runtime* milliseconds. This bit is automatically cleared by hardware when the calibration is complete.

Automatic Calibration 0		FCR_AUTOCAL0		[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
3	invert	R/W	0	**IPO Trim Step Invert** Describe this field!
2	load	R/*	0	**IPO Initial Trim Load** Set this bit to load the initial trim value for the IPO from *FCR_AUTOCAL1.initial*. This bit will be cleared by hardware once the load is complete.
1	en	R/W	0	**IPO Automatic Calibration Continuous Mode Enable** 0: Disabled 1: Enabled
0	sel	R/W	0	**IPO Trim Select** 0: Use default trim 1: Use automatic calibration trim values

Table 3-68: Automatic Calibration 1 Register

Automatic Calibration 1		FCR_AUTOCAL1		[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved** Do not modify this field.
8:0	initial	R/W	0	**IPO Trim Automatic Calibration Initial Trim**

Table 3-69: Automatic Calibration 2 Register

Automatic Calibration 2		FCR_AUTOCAL2		[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0	**Reserved** Do not modify this field.
20:8	div	R/W	0	**IPO Trim Automatic Calibration Divide Factor** Target trim frequency for the IPO: $f_{IPO} = \text{div} \times 32768$ *Note: Setting div to 0 is equivalent to setting div to 1.*
7:0	runtime	R/W	0	**IPO Trim Automatic Calibration Run Time** $\text{Atomic Run Time} = runtime \text{ milliseconds}$

Table 3-70: RISC-V Boot Address Register

RISC-V Boot Address		FCR_URVBOOT		[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0x2000 C000	**RISC-V Boot Address** Set this field to the boot address for the RISC-V core. The reset value for this register is 0x2001 C000.

Table 3-71: RISC-V Control Register

RISC-V Boot Address		FCR_URVCTRL		[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved** Do not modify this field.
1	iflush	R/W	0	**RISC-V Instruction Flush Enable** 0: Disable 1: Enable
0	memsel	R/W	0	**RISC-V Memory Select** This field determines if sysram2 and sysram3 are shared between the CM4 and RISC-V cores. Set this field to 1 to set the RISC-V as the exclusive master for sysram2 and sysram3.

3.15 Global CNN Function Control Registers (GCFCR)

See *Table 2-3: APB Peripheral Base* Address Map 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 3-65: Function Control Register Summary*. 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 2-1. Field Access Definitions* 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 3-72: Global CNN Function Control Register Summary

Offset	Register	Description
[0x0000]	*GCFCR_*	*CNN Power Domain Enable Register*
[0x0004]	*GCFCR_RAMPWR*	*Automatic Calibration 0 Register*
[0x0008]	*GCFCR_PWRISO*	*Automatic Calibration 1 Register*
[0x000C]	*GCFCR_PWRDMN_RST*	*Automatic Calibration 2 Register*

3.15.1 Global CNN Function Control Register Details

Table 3-73: Global CNN Function Control Power Domain Register

CNN Power Domain Enable		GCFCR_PWRDMN_EN		[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved** Do not modify this field.
3	dmn3_en	R/W	0	**CNN Power Domain 3 Enable** 0: Disabled 1: Enabled
2	dmn2_en	R/W	0	**CNN Power Domain 2 Enable** 0: Disabled 1: Enabled
1	dmn1_en	R/W	0	**CNN Power Domain 1 Enable** 0: Disabled 1: Enabled

CNN Power Domain Enable			GCFCR_PWRDMN_EN	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
0	dmn0_en	R/W	0	**CNN Power Domain 0 Enable** 0: Disabled 1: Enabled

Table 3-74: Global CNN Function Control CNN RAM Power Register

CNN RAM Power Enable			GCFCR_RAMPWR_EN	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved** Do not modify this field.
3	ram3_en	R/W	0	**CNN RAM Power 3 Enable** 0: Disabled 1: Enabled
2	ram2_en	R/W	0	**CNN RAM Power 2 Enable** 0: Disabled 1: Enabled
1	ram1_en	R/W	0	**CNN RAM Power 1 Enable** 0: Disabled 1: Enabled
0	ram0_en	R/W	0	**CNN RAM Power 0 Enable** 0: Disabled 1: Enabled

Table 3-75: Global CNN Function Control Power Domain Isolation Register

CNN Power Domain Isolation			GCFCR_PWRISO	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved** Do not modify this field.
3	dmn3_iso	R/W	0	**CNN Power Domain Isolation 3** 0: Disabled 1: Enabled
2	dmn2_iso	R/W	0	**CNN Power Domain Isolation 2** 0: Disabled 1: Enabled
1	dmn1_iso	R/W	0	**CNN Power Domain Isolation 1** 0: Disabled 1: Enabled
0	dmn0_iso	R/W	0	**CNN Power Domain Isolation 0** 0: Disabled 1: Enabled

Table 3-76: Global CNN Function Control Power Reset Register

CNN Power Domain Reset			GCFCR_PWRDMN_RST	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:4	-	RO	0	**Reserved** Do not modify this field.
3	dmn3_rst	R/W	0	**CNN Power Domain 3 Reset** 0: Disabled 1: Enabled
2	dmn2_rst	R/W	0	**CNN Power Domain 2 Reset** 0: Disabled 1: Enabled
1	dmn1_rst	R/W	0	**CNN Power Domain 1 Reset** 0: Disabled 1: Enabled
0	dmn0_rst	R/W	0	**CNN Power Domain 0 Reset** 0: Disabled 1: Enabled

4. Interrupts and Exceptions

Interrupts and exceptions are managed by the Arm Cortex-M4 with FPU Nested Vector Interrupt Controller (NVIC). The NVIC handles the interrupts, exceptions, priorities and masking. *Table 4-1* details the **MAX78000** interrupt vector table and describes each exception and interrupt.

4.1 CPU0 Interrupt and Exception Features

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

4.2 CPU0 Interrupt Vector Table

Table 4-1 lists the interrupt and exception table for the **MAX78000**'s CPU0 (Cortex-M4) core. There are 119 interrupt entries for the **MAX78000**, including reserved for future use interrupt place holders. Including the 15 system exceptions for the Arm Cortex-M4 with FPU, the total number of entries is 134.

*Table 4-1: **MAX78000** CPU0 Interrupt Vector Table*

Exception (Interrupt) Number	Offset	Name	Description
1	[0x0004]	Reset_Handler	Reset
2	[0x0008]	NMI_Handler	Non-Maskable Interrupt
3	[0x000C]	HardFault_Handler	Hard Fault
4	[0x0010]	MemManage_Handler	Memory Management Fault
5	[0x0014]	BusFault_Handler	Bus Fault
6	[0x0018]	UsageFault_Handler	Usage Fault
7:10	[0x001C]-[0x0028]	-	Reserved
11	[0x002C]	SVC_Handler	Supervisor Call Exception
12	[0x0030]	DebugMon_Handler	Debug Monitor Exception
13	[0x0034]	-	Reserved
14	[0x0038]	PendSV_Handler	Request Pending for System Service
15	[0x003C]	SysTick_Handler	System Tick Timer
16	[0x0040]	PF_IRQHandler	Power Fail interrupt
17	[0x0044]	WDT0_IRQHandler	Windowed Watchdog Timer 0 Interrupt
18	[0x0048]	-	Reserved
19	[0x004C]	RTC_IRQHandler	Reserved
20	[0x0050]	TRNG_IRQHandler	True Random Number Generator Interrupt
21	[0x0054]	TMR0_IRQHandler	Timer 0 Interrupt
22	[0x0058]	TMR1_IRQHandler	Timer 1 Interrupt
23	[0x005C]	TMR2_IRQHandler	Timer 2 Interrupt
24	[0x0060]	TMR3_IRQHandler	Timer 3 Interrupt

Exception (Interrupt) Number	Offset	Name	Description
25	[0x0064]	LPTMR0_IRQHandler	Low Power Timer 0 (TMR4) Interrupt
26	[0x0068]	LPTMR1_IRQHandler	Low Power Timer 1 (TMR5) Interrupt
27:28	[0x006C]:[0x0070]	-	Reserved
29	[0x0074]	I2C0_IRQHandler	I^2C Port 0 Interrupt
30	[0x0078]	UART0_IRQHandler	UART Port 0 Interrupt
31	[0x007C]	UART1_IRQHandler	UART Port 1 Interrupt
32	[0x0080]	SPI1_IRQHandler	SPI Port 1 Interrupt
33:35	[0x0084]:[0x008C]	-	Reserved
36	[0x90]	ADC_IRQHandler	ADC Interrupt
37:38	[0x0094]:[0x0098]	-	Reserved
39	[0x009C]	FLC0_IRQHandler	Flash Controller 0 Interrupt
40	[0x00A0]	GPIO0_IRQHandler	GPIO Port 0 Interrupt
41	[0x00A4]	GPIO1_IRQHandler	GPIO Port 1 Interrupt
42	[0x00A8]	GPIO2_IRQHandler	Low Power GPIO Port 2 Interrupt
43	[0x00AC]	-	Reserved
44	[0x00B0]	DMA0_IRQHandler	DMA0 Interrupt
45	[0x00B4]	DMA1_IRQHandler	DMA1 Interrupt
46	[0x00B8]	DMA2_IRQHandler	DMA2 Interrupt
47	[0x00BC]	DMA3_IRQHandler	DMA3 Interrupt
48:49	[0x00C0 : 0x00C4]	-	Reserved
50	[0x00C8]	UART2_IRQHandler	UART Port 2 Interrupt
51	[0x00CC]	-	Reserved
52	[0x00D0]	I2C1_IRQHandler	I^2C Port 1 Interrupt
53:68	[0x00D4]: [0x0110]	-	Reserved
69	[0x0114]	WUT_IRQHandler	Wakeup Timer Interrupt
70	[0x0118]	GPIOWAKE_IRQHandler	GPIO Wakeup Interrupt
71	[0x011C]	-	Reserved
72	[0x0120]	SPI1_IRQHandler	SPI Port 0 Interrupt
73	[0x0124]	LPWDT0_IRQHandler	Low Power Watchdog Timer 0 (WDT1) Interrupt
74	[0x0128]	-	Reserved
75	[0x012C]	PT_IRQHandler	Pulse Train Interrupt
76:77	[0x0130]:[0x0134]	-	Reserved
78	[0x0138]	I2C2_IRQHandler	I^2C Port 2 Interrupt
79	[0x013C]	RISCV_IRQHandler	RISC-V Interrupt

Exception (Interrupt) Number	Offset	Name	Description
80:82	[0x0140]:[0x0148]	-	Reserved
83	[0x014C]	OWM_IRQHandler	1-Wire Master Interrupt
84:97	[0x0150]:[0x0184]	-	Reserved
98	[0x0188]	ECC_IRQHandler	Error Correction Coding Block Interrupt
99	[0x018C]	DVS_IRQHandler	Digital Voltage Scaling Interrupt
100	[0x0190]	SIMO_IRQHandler	Single Input Multiple Output Interrupt
101:103	[0x0194]:[0x019C]	-	Reserved
104	[0x01A0}	LPUART0_IRQHandler	Low Power UART 0 (UART3) Interrupt
105:106	[0x01A4]:[0x01A8]	-	Reserved
107	[0x01AC]	PCIF_IRQHandler	Parallel Camera Interface Interrupt
108:112	[0x01B0]:[0x01C0]	-	Reserved
113	[0x01C4]	AES_IRQHandler	AES Interrupt
114	[0x01C8]	-	Reserved
115	[0x01CC]	I2S_IRQHandler	I2S Interrupt
116	[0x01D0]	CNN_FIFO_IRQHandler	CNN Transmit FIFO Interrupt
117	[0x01D4]	CNN_IRQHandler	CNN Interrupt
118	[0x01D8]	-	Reserved
119	[0x01DC]	LPCMP_IRQHandler	Low Power Comparator Interrupt

4.3 CPU1 Interrupt Vector Table

Table 4-1 lists the interrupt and exception table for the MAX78000's CPU1 RISC-V core. There are 119 interrupt entries for the MAX78000, including reserved for future use interrupt place holders. Including the 16 system exceptions for the Arm Cortex-M4 with FPU, the total number of entries is 135.

*Table 4-2: **MAX78000** CPU1 RISC-V Interrupt Vector Table*

Exception (Interrupt) Number	Name	Description
4	PF_IRQHandler	Power Fail interrupt
5	WDT0_IRQHandler	Windowed Watchdog Timer 0 Interrupt
6	GPIOWAKE_IRQHandler	GPIO Wakeup Interrupt
6	LPCMP_IRQHandler	Low Power Comparator Interrupt
7	RTC_IRQHandler	RTC Interrupt
8	TMR0_IRQHandler	Timer 0 Interrupt
9	TMR1_IRQHandler	Timer 1 Interrupt
10	TMR2_IRQHandler	Timer 2 Interrupt
11	TMR3_IRQHandler	Timer 3 Interrupt

Exception (Interrupt) Number	Name	Description
12	LPTMR0_IRQHandler	Low Power Timer 0 (TMR4) Interrupt
13	LPTMR1_IRQHandler	Low Power Timer 1 (TMR5) Interrupt
14	I2C0_IRQHandler	I^2C Port 0 Interrupt
15	UART0_IRQHandler	UART Port 0 Interrupt
16	-	Reserved
17	I2C1_IRQHandler	I^2C Port 1 Interrupt
18	UART1_IRQHandler	UART Port 1 Interrupt
19	UART2_IRQHandler	UART Port 2 Interrupt
20	I2C2_IRQHandler	I^2C Port 2 Interrupt
21	LPUART0_IRQHandler	Low Power UART 0 (UART3) Interrupt
22	SPI1_IRQHandler	SPI Port 1 Interrupt
23	WUT_IRQHandler	Wakeup Timer Interrupt
24	FLC0_IRQHandler	Flash Controller 0 Interrupt
25	GPIO0_IRQHandler	GPIO Port 0 Interrupt
26	GPIO1_IRQHandler	GPIO Port 1 Interrupt
27	GPIO2_IRQHandler	GPIO Port 2 Interrupt
28	DMA0_IRQHandler	DMA0 Interrupt
29	DMA1_IRQHandler	DMA1 Interrupt
30	DMA2_IRQHandler	DMA2 Interrupt
31	DMA3_IRQHandler	DMA3 Interrupt
32:45	-	Reserved
46	AES_IRQHandler	AES Interrupt
47	TRNG_IRQHandler	TRNG Interrupt
48	LPWDT0_IRQHandler	Low Power Watchdog Timer 0 (WDT1) Interrupt
49	DVS_IRQHandler	Digital Voltage Scaling Interrupt
50	SIMO_IRQHandler	Single Input Multiple Output Interrupt
51	-	Reserved
52	PT_IRQHandler	Pulse Train Interrupt
53	ADC_IRQHandler	ADC Interrupt
54	OWM_IRQHandler	1-Wire Master Interrupt
55	I2S_IRQHandler	I2S Interrupt
56	CNN_FIFO_IRQHandler	CNN FIFO Interrupt
57	CNN_IRQHandler	CNN Interrupt
58	-	Reserved

Exception (Interrupt) Number	Name	Description
59	PCIF_IRQHandler	Parallel Camera Interface Interrupt

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

General-purpose I/O (GPIO) pins can be individually configured to operate in a digital I/O mode or in an alternate function (AF) mode which maps a signal associated with an enabled peripheral to that GPIO. The GPIO support dynamic switching between I/O mode and alternate function mode. Configuring a pin for an alternate function supersedes its use as a digital I/O, however the state of the GPIO can still be read via the *GPIOn_IN* register.

The electrical characteristics of a GPIO pin are identical whether the pin is configured as an I/O or alternate function, except where explicitly noted in the data sheet electrical characteristics tables.

GPIO are logically divided into ports of 32 pins. Package variants may not implement all pins of a specific 32-bit GPIO port.

Each pin of a port has an interrupt function that can be independently enabled and configured as a level- or edge-sensitive interrupt. All GPIOs of a given port share the same interrupt vector as detailed in the section *GPIO Interrupt Handling*.

Note: GPIO3 is controlled differently and has different features than the other GPIO ports in the MAX78000. Details for using GPIO3 are covered in the system chapter.

The features for each GPIO pin include:

- Full CMOS outputs with configurable drive strength settings.
- Input modes/options:
    - High impedance
    - Weak pullup/pulldown
    - Strong pullup/pulldown
- Output data can be from *GPIOn_OUT* register or an enabled peripheral
- Input data can be read from *GPIOn_IN* input register or the enabled peripheral
- Bit set and clear registers for efficient bit-wise write access to the pins and configuration registers
- Wake from low-power modes using edge triggered inputs
- Selectable GPIO voltage supply:
    - VDDIO
    - VDDIOH
- Selectable interrupt events:
    - Level triggered low
    - Level triggered high
    - Edge triggered rising edge
    - Edge triggered falling edge
    - Edge triggered rising or falling edge
- All GPIO pins default to input mode with weak-pullup during power-on-reset events

5.1 Instances

Table 5-1. MAX78000 GPIO Pin Count 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 5-1. MAX78000 GPIO Pin Count

PACKAGE	GPIO	PINS
81-CTBGA	GPIO0[30:0]	31
	GPIO1[9:0]	10
	GPIO2[7:0]	8
	GPIO3[1:0]	2

Note: Refer to the MAX78000 System Chapter for details on using GPIO3.

Note: Refer to the MAX78000 device datasheet for a description of alternate functions for each GPIO port pin.

Note: MAX78000 does not support Selectable GPIO Voltage Supply feature.

5.2 Configuration

Each device pin can be individually configured as a GPIO or an alternate function. The correct alternate function setting must be selected for each pin of a given multi-pin peripheral for proper operation.

Table 5-2. MAX78000 GPIO Pin Function Configuration depicts the bit settings for the *GPIOn_EN0*, *GPIOn_EN1*, and *GPIOn_EN2* registers to configure the function of the GPIOn port pins. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO0_EN0.config[25]*, *GPIO0_EN1.config[25]*, and *GPIO0_EN2.config[25]* all represent configuration for device pin P0.25. See *Table 5-6. MAX78000 GPIO Alternate Function Configuration* for a detailed example of how each of these bits applies to each of the GPIO device pins.

Table 5-2. MAX78000 GPIO Pin Function Configuration

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

Table 5-3. MAX78000 Input Mode Configuration depicts the bit settings for the digital I/O input mode. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO0_PAD_CFG2.config[25]*, *GPIO0_PAD_CFG1.config[25]*, *GPIO0_PS.str[25]*, and *GPIO0_VSSEL.vlvl[25]* all represent configuration for device pin P0.25. See *Table 5-9. MAX78000 GPIO Pullup/Pulldown/Drive Strength/Voltage Configuration*

Reference for a detailed example of how each of these bits applies to each of the GPIO device pins. Refer to the data sheet for details of specific electrical characteristics.

Table 5-3. MAX78000 Input Mode Configuration

Input Mode	Mode Select		Pullup/Pulldown Strength	Power Supply
	GPIOn_PAD_CFG2.config[pin]	***GPIOn_PAD_CFG1.config[pin]***	***GPIOn_PS.str[pin]***	***GPIOn_VSSEL.vlvl[pin]***
High-impedance	0	0	N/A	Not Available on MAX78000
Weak Pull-up to VDDIO (1MΩ)	0	1	0	Not Available on MAX78000
Strong Pull-up to VDDIO (25KΩ)	0	1	1	Not Available on MAX78000
Weak Pull-down to VDDIO (1MΩ)	1	0	0	Not Available on MAX78000
Strong Pull-down to VDDIO (25KΩ)	1	0	1	Not Available on MAX78000
Reserved	1	1	NA	N/A

Table 5-4. MAX78000 Output Mode Configuration shows the configuration options for digital I/O in output mode. Each of the bits within these registers represents the configuration of a single pin on the GPIO port. For example, *GPIO0_DS.str[25]*, *GPIO0_DS1.str[25]*, and *GPIO0_VSSEL.vlvl[25]* all represent configuration for device pin P0.25. See *Table 5-9. MAX78000 GPIO Pullup/Pulldown/Drive Strength/Voltage Configuration Reference* for a detailed example of how each of these bits applies to each of the GPIO device pins. Refer to the data sheet for details of specific electrical characteristics.

Table 5-4. MAX78000 Output Mode Configuration

Input Mode	Drive Strength		Power Supply
	GPIOn_DS1.str[pin]	***GPIOn_DS.str[pin]***	***GPIOn_VSSEL.vlvl[pin]***
Output Drive Strength 0, VDDIO Supply	0	0	Not Available on MAX78000
Output Drive Strength 1, VDDIO Supply	0	1	Not Available on MAX78000
Output Drive Strength 2, VDDIO Supply	1	0	Not Available on MAX78000
Output Drive Strength 3, VDDIO Supply	1	1	Not Available on MAX78000
Output Drive Strength 0, VDDIOH Supply	0	0	Not Available on MAX78000
Output Drive Strength 1, VDDIOH Supply	0	1	Not Available on MAX78000
Output Drive Strength 2, VDDIOH Supply	1	0	Not Available on MAX78000
~~Output Drive Strength 3, VDDIOH Supply~~	~~1~~	~~1~~	~~Not Available on MAX78000~~

Each GPIO port is assigned a dedicated interrupt vector as shown in *Table 5-5. MAX78000 GPIO Port Interrupt Vector Mapping*.

Table 5-5. MAX78000 GPIO Port Interrupt Vector Mapping

GPIO Interrupt Source	GPIO Interrupt Status Register	Device Specific Interrupt Vector Number	GPIO Interrupt Vector
GPIO0[31:0]	*GPIOn_INT_STAT*	40	GPIO0_IRQHandler
GPIO1[31:0]	*GPIOn_INT_STAT*	41	GPIO1_IRQHandler
GPIO2[31:0]	*GPIOn_INT_STAT*	42	GPIO2_IRQHandler
GPIO3[31:0]	*GPIOn_INT_STAT*	43	GPIO3_IRQHandler

5.3 Reference Tables

The tables in this section provide example references for register bit assignment to configure a device's GPIO pins.

Table 5-6. MAX78000 GPIO Alternate Function Configuration Reference

Device Pin	Alternate Function Configuration Bits		
P0.0	*GPIO0_EN0.config[0]*	*GPIO0_EN1.config[0]*	*GPIO0_EN2.config[0]*
P0.1	*GPIO0_EN0.config[1]*	*GPIO0_EN1.config[1]*	*GPIO0_EN2.config[1]*
…	…	…	…
P0.30	*GPIO0_EN0.config[30]*	*GPIO0_EN1.config[30]*	*GPIO0_EN2.config[30]*
P0.31	*GPIO0_EN0.config[31]*	*GPIO0_EN1.config[31]*	*GPIO0_EN2.config[31]*

Table 5-7. MAX78000 GPIO Output/Input Configuration Reference

Device Pin	GPIO Output Enable	GPIO Output Write	GPIO Input Enable	GPIO Input Read
P0.0	*GPIO0_OUT_EN.en[0]*	*GPIO0_OUT.lvl[0]*	*GPIO0_IN_EN.en[0]*	*GPIO0_IN.input[0]*
P0.1	*GPIO0_OUT_EN.en[1]*	*GPIO0_OUT.lvl[1]*	*GPIO0_IN_EN.en[1]*	*GPIO0_IN.input[1]*
…	…	…	…	…
P0.30	*GPIO0_OUT_EN.en[30]*	*GPIO0_OUT.lvl[30]*	*GPIO0_IN_EN.en[30]*	*GPIO0_IN.input[30]*
P0.31	*GPIO0_OUT_EN.en[31]*	*GPIO0_OUT.lvl[31]*	*GPIO0_IN_EN.en[31]*	*GPIO0_IN.input[31]*

Table 5-8. MAX78000 GPIO Interrupt Configuration Reference

Device Pin	Enable	Status	Dual Edge	Polarity	Trigger	Wakeup
P0.0	*GPIO0_INT_EN*.en[0]	*GPIO0_INT_STAT*.int[0]	*GPIO0_INT_DUAL_EDGE.en[0]*	*GPIO0_INT_POL*.pol[0]	*GPIO0_INT_MOD.trigger*[0]	GPIO0_WAKE_EN.en[0]
P0.1	GPIO0_INT_EN.en[1]	GPIO0_INT_STAT.int[1]	GPIO0_INT_DUAL_EDGE.en[1]	GPIO0_INT_POL.pol[1]	GPIO0_INT_MOD.trigger[1]	GPIO0_WAKE_EN.en[1]
…	…	…	…	…	…	…
P0.30	*GPIO0_INT_EN*.en[30]	*GPIO0_INT_STAT*.int[30]	*GPIO0_INT_DUAL_EDGE*.en[30]	*GPIO0_INT_POL*.pol[30]	*GPIO0_INT_MOD*.trigger[30]	*GPIO0_WAKE_EN*.en[30]
P0.31	*GPIO0_INT_EN*.en[31]	*GPIO0_INT_STAT*.int[31]	*GPIO0_INT_DUAL_EDGE*.en[31]	*GPIO0_INT_POL*.pol[31]	*GPIO0_INT_MOD*.trigger[31]	*GPIO0_WAKE_EN*.en[31]

Table 5-9. MAX78000 GPIO Pullup/Pulldown/Drive Strength/Voltage Configuration Reference

Device Pin	Pullup/Pulldown/Strength Select			Drive Strength		Voltage
P0.0	*GPIO0_PAD_CFG1.config[0]*	*GPIO0_PAD_CFG2.config[0]*	*GPIO0_PS.str[0]*	*GPIO0_DS.str[0]*	*GPIO0_DS1.str[0]*	*Not Available on MAX78000*
P0.1	*GPIO0_PAD_CFG1.config[1]*	*GPIO0_PAD_CFG2.config[1]*	*GPIO0_PS.str[1]*	*GPIO0_DS.str[1]*	*GPIO0_DS1.str[1]*	*Not Available on MAX78000*
…	…	…	…	…	…	Not Available on MAX78000
P0.30	*GPIO0_PAD_CFG1.config[30]*	*GPIO0_PAD_CFG2.config[30]*	*GPIO0_PS.str[30]*	*GPIO0_DS.str[30]*	*GPIO0_DS1.str[30]*	*Not Available on MAX78000*
P0.31	*GPIO0_PAD_CFG1.config[31]*	*GPIO0_PAD_CFG2.config[31]*	*GPIO0_PS.str[31]*	*GPIO0_DS.str[31]*	*GPIO0_DS1.str[31]*	*Not Available on MAX78000*

5.4 Usage

5.4.1 Reset State

During a power-on-reset event, each GPIO is reset to the default input mode with the weak pullup resistor enabled as follows:

1. The GPIO Configuration Enable bits shown in *Table 5-2* are set to I/O (transition to AF1) mode.
2. Input mode is enabled (*GPIOn_IN_EN.en[pin]* = 1).
3. High impedance mode enabled (*GPIOn_PAD_CFG1.config[pin]* = 0, *GPIOn_PAD_CFG2.config[pin]* = 0), pullup and pulldown disabled.
4. Output mode disabled (*GPIOn_OUT_EN.en[pin]* = 0)
5. Interrupt disabled (*GPIOn_INT_EN.en[pin]* = 0)

5.4.2 Input Mode Configuration

Perform the following steps to configure one or more pins for input mode:

1. Set the GPIO Configuration Enable bits shown in *Table 5-2* to any one of the I/O mode settings.
2. Configure the electrical characteristics of the pin as desired as shown in *Table 5-3*.
3. Enable the input buffer connection to the GPIO pin by setting *GPIOn_IN_EN.en[pin]* to 1.
4. Read the input state of the pin using the *GPIOn_IN.input[pin]* field.

5.4.3 Output Mode Configuration

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

1. Set the GPIO Configuration Enable bits shown in *Table 5-2* to any one of the I/O mode settings.
2. Configure the electrical characteristics of the pin as desired as shown in *Table 5-4*.
3. Set the output logic high or logic low using the *GPIOn_OUT.lvl[pin]* bit.
4. Enable the output buffer for the pin by setting *GPIOn_OUT_EN.en[pin]* to 1.

5.4.4 Alternate Function Configuration

Most GPIO support one or more alternate functions which are selected with the GPIO Configuration Enable bits shown in *Table 5-2.* The bits that select the AF must only be changed while the pin is in one of the I/O modes (*GPIOn_EN0* = 1). The specific I/O mode must match the desired AF. For example, if a transition to AF1 is desired, first select the setting corresponding to I/O (transition to AF1). Then enable the desired mode by selecting the AF1 mode.

1. Set the GPIO Configuration Enable bits shown in *Table 5-2* to the I/O mode that corresponds with the desired new AF setting. For example, select "I/O (transition to AF1)" if switching to AF1. Switching between different I/O mode settings will not affect the state or electrical characteristics of the pin.
2. Configure the electrical characteristics of the pin. See *Table 5-3* if the assigned alternate function will use the pin as an input. See *Table 5-4* if the assigned alternate function will use the pin as an output.
3. Set the GPIO Configuration Enable bits shown in *Table 5-2* to the desired alternate function.

5.5 Configuring GPIO (External) Interrupts

Each GPIO pin supports external interrupt events when the GPIO is configured for I/O mode and the input mode is enabled. If the GPIO is configured as a peripheral alternate function, the interrupts are peripheral-controlled.

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

Each GPIO pin has a dedicated status bit in its corresponding *GPIOn_INT_STAT* register. A GPIO interrupt will occur when the a status bit transitions from 0 to 1 if the corresponding bit is set in the corresponding *GPIOn_INT_EN* register. Note that the interrupt status bit will always be set when the current interrupt configuration event occurs, but an interrupt will only be generated if explicitly enabled.

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

1. Disable interrupts by setting the *GPIOn_INT_EN.en[pin]* field to 0. This will prevent any new interrupts on the pin from triggering but will not clear previously triggered (pending) interrupts. The application can disable all interrupts for a GPIO port by writing 0 to the *GPIOn_IN_EN* register. To maintain previously enabled interrupts, read the *GPIOn_IN_EN* register and save the state prior to setting the register to 0.
2. Clear pending interrupts by writing 1 to the *GPIOn_INT_CLR.clr[pin]* bit.
3. Configure the pin for the desired interrupt event
4. Set *GPIOn_INT_MOD.trigger[pin]* to select the desired interrupt.
5. For level triggered interrupts, the interrupt triggers on an input high (*GPIOn_INT_POL.pol[pin]* = 0) or input low level.
6. For edge triggered interrupts, the interrupt triggers on a transition from low to high(*GPIOn_INT_POL.pol[pin]* = 0) or high to low (*GPIOn_INT_POL.pol[pin]* = 1).
7. Optionally set *GPIOn_INT_DUAL_EDGE.en[pin]* to 1 to trigger on both the rising and falling edges of the input signal.
   a. Set *GPIOn_INT_EN.en[pin]* to 1 to enable the interrupt for the pin.

5.5.1 GPIO Interrupt Handling

Each GPIO port is assigned its own dedicated interrupt vector as shown in *Table 5-5. MAX78000 GPIO Port Interrupt Vector Mapping*.

To handle GPIO interrupts in your interrupt vector handler, complete the following steps:

1. Read the *GPIOn_INT_STAT* register to determine the GPIO pin that triggered the interrupt.
2. Complete interrupt tasks associated with the interrupt source pin (application defined).
3. Clear the interrupt flag in the *GPIOn_INT_STAT* register by writing a 1 to the *GPIOn_INT_CLR* bit position that triggered the interrupt. This also clears and rearms the edge detectors for edge triggered interrupts.
4. Return from the interrupt vector handler.

5.5.2 Using GPIO for Wakeup from Low Power Modes

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

A single wakeup interrupt vector, GPIOWAKE_IRQHandler, is assigned for all pins of all GPIO ports. When the GPIO wakeup event occurs, the application software must interrogate each *GPIOn_INT_STAT* register to determine which external port pin caused the wake-up event.

Table 5-10. MAX78000 GPIO Wakeup Interrupt Vector

GPIO Wake Interrupt Source	GPIO Wake Interrupt Status Register	Device Specific Interrupt Vector Number	GPIO Wakeup Interrupt Vector
GPIO0	*GPIOn_INT_STAT*	70	GPIOWAKE_IRQHandler
GPIO1	*GPIOn_INT_STAT*	70	GPIOWAKE_IRQHandler
GPIO2	*GPIOn_INT_STAT*	70	GPIOWAKE_IRQHandler
GPIO3	GPIOn_INT_STAT	70	GPIOWAKE_IRQHandler

To enable low power mode wakeup (SLEEP, DEEPSLEEP and BACKUP) using an external GPIO interrupt, complete the following steps:

1. Clear pending interrupt flags by writing a logic 1 to *GPIOn_INT_CLR.clr[pin]*.
2. Activate the GPIO wakeup function by writing a logic 1 to *GPIOn_WAKE_EN.en[pin]*.
3. Configure the power manager to use the GPIO as a wakeup source by setting the TBD Link >>> to GCR_PM.gpiowken field to 1.

5.6 Registers

Each GPIO port has a unique base address as shown in TBD LINK >>>> Table 2 1. APB Peripheral Base Address Map. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific reset, if applicable.

Table 5-11. GPIO Register Summary

Offset	Register	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_OUT_EN*	*GPIO Port n Output Enable Register*
[0x0010]	*GPIOn_OUT_EN_SET*	*GPIO Port n Output Enable Atomic Set Register*
[0x0014]	*GPIOn_OUT_EN_CLR*	*GPIO Port n Output Enable Atomic Clear Register*
[0x0018]	*GPIOn_OUT*	*GPIO Port n Output Register*
[0x001C]	*GPIOn_OUT_SET*	*GPIO Port n Output Atomic Set Register*
[0x0020]	*GPIOn_OUT_CLR*	*GPIO Port n Output Atomic Clear Register*
[0x0024]	*GPIOn_IN*	*GPIO Port n Input Register*
[0x0028]	*GPIOn_INT_MOD*	*GPIO Port n Interrupt Mode Register*
[0x002C]	*GPIOn_INT_POL*	*GPIO Port n Interrupt Polarity Register*
[0x0030]	*GPIOn_IN_EN*	*GPIO Port n Input Enable Register*
[0x0034]	*GPIOn_INT_EN*	*GPIO Port n Interrupt Enable Register*
[0x0038]	*GPIOn_INT_EN_SET*	*GPIO Port n Interrupt Enable Atomic Set Register*
[0x003C]	*GPIOn_INT_EN_CLR*	*GPIO Port n Interrupt Enable Atomic Clear Register*
[0x0040]	*GPIOn_INT_STAT*	*GPIO Port n Interrupt Status Register*
[0x0048]	*GPIOn_INT_CLR*	*GPIO Port n Interrupt Clear Register*
[0x004C]	*GPIOn_WAKE_EN*	*GPIO Port n Wakeup Enable Register*
[0x0050]	*GPIOn_WAKE_EN_SET*	*GPIO Port n Wakeup Enable Atomic Set Register*
[0x0054]	*GPIOn_WAKE_EN_CLR*	*GPIO Port n Wakeup Enable Atomic Clear Register*
[0x005C]	*GPIOn_INT_DUAL_EDGE*	*GPIO Port n Interrupt Dual Edge Mode Register*
[0x0060]	*GPIOn_PAD_CFG1*	*GPIO Port n Pad Configuration 1 Register*
[0x0064]	*GPIOn_PAD_CFG2*	*GPIO Port n Pad Configuration 2 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*
[0x00B0]	*GPIOn_DS*	*GPIO Port n Output Drive Strength Bit 0 Register*

Offset	Register	Description
[0x00B4]	*GPIOn_DS1*	*GPIO Port n Output Drive Strength Bit 1 Register*
[0x00B8]	*GPIOn_PS*	*GPIO Port n Pulldown/Pullup Strength Select Register*
[0x00C0]	*GPIOn_VSSEL*	*Not Available on MAX78000*

5.6.1 Register Details

Table 5-12. GPIO Port n Configuration Enable Bit 0 Register

GPIO Port n Configuration Enable Bit 0				GPIOn_EN0	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	config	R/W	1	**GPIO Configuration Enable Bit 0** These bits, in conjunction with bits in *Table 5-2. MAX78000 GPIO Pin Function Configuration* configure the corresponding device pin for digital I/O or an alternate function mode. This field can be modified directly by writing to this register or indirectly through *GPIOn_EN0_SET* or *GPIOn_EN0_CLR*. *Table 5-6. MAX78000 GPIO Alternate Function Configuration* depicts a detailed example of how each of these bits applies to each of the GPIO device pins *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect input and interrupt functionality of the associated pin.*	

Table 5-13. GPIO Port n Configuration Enable Atomic Set Bit 0 Register

GPIO Port n Configuration Enable Atomic Set Bit 0				GPIOn_EN0_SET	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W1	0	**GPIO Configuration Enable Atomic Set Bit 0** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_EN0* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN0* register set to 1.	

Table 5-14. GPIO Port n Configuration Enable Atomic Clear Bit 0 Register

GPIO Port n Configuration Enable Atomic Clear Bit 0				GPIOn_EN0_CLR	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W1	0	**GPIO Configuration Enable Atomic Clear Bit 0** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_EN0* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN0* register cleared to 0.	

Table 5-15. GPIO Port n Output Enable Register

GPIO Port n Output Enable				GPIOn_OUT_EN	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	en	R/W	0	**GPIO Output Enable** Set bit to 1 to enable the output driver for the corresponding GPIO pin. A bit can be enabled directly by writing to this register or indirectly through *GPIOn_OUT_EN_SET* or *GPIOn_OUT_EN_CLR*. 0: Pin is set to input mode; output driver disabled. 1: Pin is set to output mode.	

Table 5-16. GPIO Port n Output Enable Atomic Set Register

GPIO Port n Output Enable Atomic Set				GPIOn_OUT_EN_SET [0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1	0	**GPIO Output Enable Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_OUT_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUT_EN* set to 1.

Table 5-17. GPIO Port n Output Enable Atomic Clear Register

GPIO Port n Output Enable Atomic Clear				GPIOn_OUT_EN_CLR [0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1	0	**GPIO Output Enable Atomic Clear** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_OUT_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUT_EN* cleared to 0.

Table 5-18. GPIO Port n Output Register

GPIO Port n Output				GPIOn_OUT [0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	lvl	R/W	0	**GPIO Output** Set the corresponding output pin high or low. 0: Drive the corresponding output pin low (logic 0). 1: Drive the corresponding output pin high (logic 1).

Table 5-19. GPIO Port n Output Atomic Set Register

GPIO Port n Output Atomic Set				GPIOn_OUT_SET [0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	R/W1	0	**GPIO Output Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_OUT* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUT*_EN set to 1.

Table 5-20. GPIO Port n Output Atomic Clear Register

GPIO Port n Output Atomic Clear				GPIOn_OUT_CLR [0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	-	WO	0	**GPIO Output Atomic Clear** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_OUT* register. 0: No effect. 1: Corresponding bits in *GPIOn_OUT*_EN cleared to 0.

Table 5-21. GPIO Port n Input Register

GPIO Port n Input				GPIOn_IN	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	input	RO	-	**GPIO Input** Returns the state of the input pin only if the corresponding bit in the *GPIOn_IN_EN* register is set. The state is not affected by the pin's configuration as an output or alternate function. 0: Input pin low 1: Input pin high.	

Table 5-22. GPIO Port n Interrupt Mode Register

GPIO Port n Interrupt Mode				GPIOn_INT_MOD	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	trigger	R/W	0	**GPIO Interrupt Mode** Selects interrupt mode for the corresponding GPIO pin. 0: Level triggered interrupt. 1: Edge triggered interrupt. *Note: This bit has no effect unless the corresponding bit in the GPIOn_INT_EN register is set.*	

Table 5-23. GPIO Port n Interrupt Polarity Register

GPIO Port n Interrupt Polarity				GPIOn_INT_POL	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	pol	R/W	0	**GPIO Interrupt Polarity** Interrupt polarity selection bit for the corresponding GPIO pin. Level triggered mode (*GPIOn_INT_MOD.trigger*[pin] = 0): 0: Input low (logic 0) triggers interrupt. 1: Input high (logic 1) triggers interrupt. Edge triggered mode (*GPIOn_INT_MOD.trigger*[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_INT_EN register is set.*	

Table 5-24. GPIO Port n Input Enable Register

GPIO Port n Input Enable				GPIOn_IN_EN	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	en	R/W	1	**GPIO Input Enable** 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 to the pad for reading via *GPIOn_IN* register.	

Table 5-25. GPIO Port n Interrupt Enable Register

GPIO Port n Interrupt Enable				GPIOn_INT_EN	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	en	R/W	0	**GPIO Interrupt Enable** Enable or disable the interrupt for the corresponding GPIO pin. 0: GPIO interrupt disabled. 1: GPIO interrupt enabled. *Note: Disabling a GPIO interrupt does not clear pending interrupts for the associated pin. Use the GPIOn_INT_CLR register to clear pending interrupts.*	

Table 5-26. GPIO Port n Interrupt Enable Atomic Set Register

GPIO Port Interrupt Enable Atomic Set				GPIOn_INT_EN_SET	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W1	0	**GPIO Interrupt Enable Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_INT_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_INT_EN* register set to 1.	

Table 5-27. GPIO Port n Interrupt Enable Atomic Clear Register

GPIO Port Interrupt Enable Atomic Clear				GPIOn_INT_EN_CLR	[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W1	0	**GPIO Interrupt Enable Atomic Clear** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_INT_EN* register. 0: No effect. 1: Corresponding bits in *GPIOn_INT_EN* register cleared to 0.	

Table 5-28. GPIO Port n Interrupt Status Register

GPIO Port Interrupt Status				GPIOn_INT_STAT	[0x0040]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	int	RO	0	**GPIO Interrupt Status** An interrupt is pending for the associated GPIO pin when this bit reads 1. 0: No interrupt pending for associated GPIO pin. 1: GPIO interrupt pending for associated GPIO pin. *Note: Write a 1 to the corresponding bit in the GPIOn_INT_CLR register to clear the interrupt pending status flag.*	

Table 5-29. GPIO Port n Interrupt Clear Register

GPIO Port Interrupt Clear				GPIOn_INT_CLR	[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	clr	R/W1C	0	**GPIO Interrupt Clear** Write 1 to clear the associated interrupt status (*GPIOn_INT_STAT*). 0: No effect on the associated *GPIOn_INT_STAT* flag. 1: Clear the associated interrupt pending flag in the *GPIOn_INT_STAT* register.	

Table 5-30. GPIO Port n Wakeup Enable Register

GPIO Port n Wakeup Enable				GPIOn_WAKE_EN	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	en	R/W	0	**GPIO Wakeup Enable** Enable the I/O as a wakeup from low power modes (SLEEP, DEEPSLEEP, BACKUP). 0: GPIO is not enabled as a wakeup source from low power modes. 1: GPIO is enabled as a wakeup source from low power modes.	

Table 5-31. GPIO Port n Wakeup Enable Atomic Set Register

GPIO Port Wakeup Enable Atomic Set				GPIOn_WAKE_EN_SET	[0x0050]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W1	0	**GPIO Wakeup Enable Atomic Set** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_WAKE_EN*r register. 0: No effect. 1: Corresponding bits in *GPIOn_WAKE_EN* register set to 1.	

Table 5-32. GPIO Port n Wakeup Enable Atomic Clear Register

GPIO Port Wakeup Enable Atomic Clear				GPIOn_WAKE_EN_CLR	[0x0054]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W1	0	**GPIO Wakeup Enable Atomic Clear** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_WAKE_EN*r register. 0: No effect. 1: Corresponding bits in *GPIOn_WAKE_EN* register cleared to 0.	

Table 5-33. GPIO Port n Interrupt Dual Edge Mode Register

GPIO Port n Interrupt Dual Edge Mode				GPIOn_INT_DUAL_EDGE	[0x005C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	en	R/W	0	**GPIO Interrupt Dual-Edge Mode Select** Setting this bit triggers interrupts on both the rising and falling edges of the corresponding GPIO if the associated *GPIOn_INT_MOD* bit is set to edge triggered. The associated polarity (*GPIOn_INT_POL*) setting has no effect when this bit is set. 0: Disable 1: Enable	

Table 5-34. GPIO Port n Pad Configuration 1 Register

GPIO Port n Pad Configuration 1				GPIOn_PAD_CFG1	[0x0060]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	config	R/W	0	**GPIO Pad Configuration 1** Input mode configuration for the associated GPIO pin. Input mode selection and the selection of a weak or strong pullup or weak or strong pulldown resistor are described in *Table 5-3*.	

Table 5-35. GPIO Port n Pad Configuration 2 Register

GPIO Port n Pad Configuration 2				GPIOn_PAD_CFG2	[0x0064]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	config	R/W	0	**GPIO Pad Configuration 2** Input mode configuration for the associated GPIO pin. Input mode selection and the selection of a weak or strong pullup or weak or strong pulldown resistor are described in *Table 5-3. MAX78000 Input Mode Configuration*.	

Table 5-36. GPIO Port n Configuration Enable Bit 1 Register

GPIO Port n Configuration Enable Bit 1				GPIOn_EN1	[0x0068]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	config	R/W	0	**GPIO Configuration Enable Bit 1** These bits, in conjunction with bits in *Table 5-2. MAX78000 GPIO Pin Function Configuration* configure the corresponding device pin for digital I/O or an alternate function mode. This field can be modified directly by writing to this register or indirectly through *GPIOn_EN1_SET* or *GPIOn_EN1_CLR*. *Table 5-6. MAX78000 GPIO Alternate Function Configuration* depicts a detailed example of how each of these bits applies to each of the GPIO device pins *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect input and interrupt functionality of the associated pin.*	

Table 5-37. GPIO Port n Configuration Enable Atomic Set Bit 1 Register

GPIO Port n Configuration Enable Atomic Set Bit 1				GPIOn_EN1_SET	[0x006C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W1	0	**GPIO Configuration Enable Atomic Set Bit 1** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register set to 1.	

Table 5-38. GPIO Port n Configuration Enable Atomic Clear Bit 1 Register

GPIO Port n Configuration Enable Atomic Clear Bit 1				GPIOn_EN1_CLR	[0x0070]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W1	0	**GPIO Configuration Enable Atomic Clear Bit 1** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_EN1* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN1* register cleared to 0.	

Table 5-39. GPIO Port n Configuration Enable Bit 2 Register

GPIO Port n Configuration Enable Bit 2				GPIOn_EN2	[0x0074]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	config	R/W	0	**GPIO Configuration Enable Bit 2** These bits, in conjunction with bits in These bits, in conjunction with bits in *Table 5-2. MAX78000 GPIO Pin Function Configuration* configure the corresponding device pin for digital I/O or an alternate function mode. This field can be modified directly by writing to this register or indirectly through *GPIOn_EN2_SET* or *GPIOn_EN2_CLR.* *Table 5-6. MAX78000 GPIO Alternate Function Configuration* depicts a detailed example of how each of these bits applies to each of the GPIO device pins *Note: Some GPIO are not implemented in all devices. The bits associated with unimplemented GPIO should not be changed from their default value.* *Note: This register setting does not affect input and interrupt functionality of the associated pin.*	

Table 5-40. GPIO Port n Configuration Enable Atomic Set Bit 2 Register

GPIO Port n Configuration Enable Atomic Set Bit 2				GPIOn_EN2_SET	[0x0078]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W1	0	**GPIO Alternate Function Select Atomic Set Bit 2** Writing 1 to one or more bits sets the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register set to 1.	

Table 5-41. GPIO Port n Configuration Enable Atomic Clear Bit 2 Register

GPIO Port n Configuration Enable Atomic Clear Bit 2				GPIOn_EN2_CLR	[0x007C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	-	R/W1	0	**GPIO Alternate Function Select Atomic Clear Bit 2** Writing 1 to one or more bits clears the corresponding bits in the *GPIOn_EN2* register. 0: No effect. 1: Corresponding bits in *GPIOn_EN2* register cleared to 0.	

Table 5-42. GPIO Port n Output Drive Strength Bit 0 Register

GPIO Port n Output Drive Strength Bit 0				GPIOn_DS	[0x00B0]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	str	R/W	0	**GPIO Output Drive Strength Selection 0** See *Table 5-4. MAX78000 Output Mode Configuration* for details on how to set the GPIO output drive strength and other electrical characteristics.	

Table 5-43. GPIO Port n Output Drive Strength Bit 1 Register

GPIO Port n Output Drive Strength Bit 1				GPIOn_DS1	[0x00B4]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	str	R/W	0	**GPIO Output Drive Strength Selection 1** See *Table 5-4. MAX78000 Output Mode Configuration* for details on how to set the GPIO output drive strength and other electrical characteristics.	

Table 5-44. GPIO Port n Pulldown/Pullup Strength Select Register

GPIO Port n Pulldown/Pullup Strength Select				GPIOn_PS	[0x00B8]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	str	R/W	0	**GPIO Pulldown/Pullup Strength Select** Selects the strength of the pullup or pulldown resistor for a pin configured for input mode. 0: Weak pulldown/pullup resistor for input pin. 1: Strong pulldown/pullup resistor for input pin. *Note: Refer to the datasheet for specific electrical characteristics of the Pulldown/Pullup resistances.*	

Table 5-45. GPIO Port n Voltage Select Register

GPIO Port n Voltage Select				GPIOn_VSSEL	[0x00C0]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	vlvl	R/W	0	**GPIO Supply Voltage Select** Not Available on MAX78000	

6. Flash Controller (FLC)

The MAX32670 Flash Controller manages read, write, and erase accesses to the internal flash. It provides the following features:

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

6.1 Instances

The device provides one instance of the FLC.

The 512KB of internal flash memory is programmable via serial wire debug interface (in-system) or directly with user application code (in-application).

The flash is organized as an array of 2,048 words by 128 bits, or 8,192 bytes per page. *Table 6-1*, *below*, shows the page start address and page end address of the internal flash memory.

Table 6-1. MAX78000 Internal Flash Memory Organization

Instance Number	Page Number	Size (per page)	Start Address	End Address
FLC0	1	8,192 Bytes	0x1000 0000	0x1000 1FFF
	2	8,192 Bytes	0x1000 2000	0x1000 3FFF
	3	8,192 Bytes	0x1000 4000	0x1000 5FFF
	4	8,192 Bytes	0x1000 6000	0x1000 7FFF
	.	.	.	.
	63	.	0x1007 C000·	0x1007 DFFF
	64	.	0x1007 E000	0x1007 FFFF

6.2 Usage

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

6.2.1 Clock Configuration

The FLC requires a 1MHz APB PCLK clock. See *Oscillator Sources and Clock Switching* for details. Use the FLC clock divisor to generate $f_{FLCnCLK} = 1MHz$, as shown in *Equation 6-1 below*. If using the IPO as the system clock, the *FLCn_CLKDIV.clkdiv* should be set to50 (0x32).

Equation 6-1: FLC Clock Frequency

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

6.2.2 Lock Protection

A locking mechanism prevents accidental memory writes and erases. All writes and erase operations require the *FLCn_CTRL.unlock* field be set to 0x2 prior to starting the operation. Writing any other value to this field, *FLCn_CTRL.unlock*, results in:

1. The flash instance remaining locked,

   or,

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

Note: If a write, page erase or mass erase operation is started and the unlock code was not set to 0x2, the flash controller hardware sets the access fail flag, FLCn_INTFL.access_fail, to indicate an access violation occurred.

6.2.3 Flash Write Width

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

Table 6-2. Valid Addresses Flash Writes

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

6.2.4 Flash Write

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

1. If desired, enable flash controller interrupts by setting the *FLCn_INTFL.access_error_ie* and *FLCn_INTFL.done_ie* bits.
2. Read the *FLCn_CTRL.busy* bit until it returns 0.
3. Configure *FLCn_CTRL.clkdiv* to match the SYS_CLK frequency.
4. Set the *FLCn_ADDR* register to a valid target address. Reference *Table 6-2*.
5. Set *FLCn_DATA3*, *FLCn_DATA2*, *FLCn_DATA1*, and *FLCn_DATA0* to the data to write. *FLCn_DATA3* is the most significant word and *FLCn_DATA0* is the least significant word. Each word of the data to write follows the little-endian format where the least significant byte of the word is stored at the lowest-numbered byte and the most significant byte is stored at the highest-numbered byte.
6. Set *FLCn_CTRL.unlock* to 0x2 to unlock the flash instance.
7. Set *FLCn_CTRL.write* to 1. This field is automatically cleared by the FLC when the write operation is finished.
8. *FLCn_INTFL.done* is set by hardware when the write completes and if an error occurred, the *FLCn_INTFL.access_fail* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.
9. Set *FLCn_CTRL.unlock* to any value other than 0x2 to re-lock the flash instance.

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

6.2.5 Flash Write

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

1. If desired, enable flash controller interrupts by setting the *FLCn_INTFL.access_error_ie* and *FLCn_INTFL.done_ie* bits.
2. Read the *FLCn_CTRL.busy* bit until it returns 0.
3. Configure *FLCn_CTRL.clkdiv* to match the SYS_CLK frequency.
4. Set the *FLCn_ADDR* register to a valid target address. Reference *Table 6-2*.
5. Set *FLCn_DATA3*, *FLCn_DATA2*, *FLCn_DATA1*, and *FLCn_DATA0* to the data to write. *FLCn_DATA3* is the most significant word and *FLCn_DATA0* is the least significant word. Each word of the data to write follows the little-endian format where the least significant byte of the word is stored at the lowest-numbered byte and the most significant byte is stored at the highest-numbered byte.
6. Set *FLCn_CTRL.unlock* to 0x2 to unlock the flash instance.
7. Set *FLCn_CTRL.write* to 1. This field is automatically cleared by the FLC when the write operation is finished.
8. *FLCn_INTFL.done* is set by hardware when the write completes and if an error occurred, the *FLCn_INTFL.access_fail* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.
9. Set *FLCn_CTRL.unlock* to any value other than 0x2 to re-lock the flash instance.

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

6.2.6 Page Erase

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

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

1. If desired, enable flash controller interrupts by setting the *FLCn_INTFL.access_error_ie* and *FLCn_INTFL.done_ie bits*.
2. Read the *FLCn_CTRL.busy* bit until it returns 0.
3. Configure *FLCn_CLKDIV.clkdiv* to match the APB clock frequency.
4. Set the *FLCn_ADDR* register to an address within the target page to be erased. *FLCn_ADDR*[12:0] are ignored by the FLC to ensure the address is page aligned.
5. Set *FLCn_CTRL.unlock* to 0x2 to unlock the flash instance.
6. Set *FLCn_CTRL.erase_code* to 0x55 for page erase.
7. Set *FLCn_CTRL.page_erase* to 1 to start the page erase operation.
8. The *FLCn_CTRL.busy* bit is set by the flash controller while the page erase is in progress and the *FLCn_CTRL.page_erase* and *FLCn_CTRL.busy* are cleared by the flash controller when the page erase is complete.
9. *FLCn_INTFL.done* is set by hardware when the page erase completes and if an error occurred, the *FLCn_INTFL.access_fail* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.
10. Set *FLCn_CTRL.unlock* to any value other than 0x2 to re-lock the flash instance.

6.2.7 Mass Erase

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

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

1. Read the *FLCn_CTRL.busy* bit until it returns 0.
2. Configure *FLCn_CLKDIV.clkdiv* to match the SYS_CLK frequency.
3. Set *FLCn_CTRL.unlock* to 0x2 to unlock the internal flash.
4. Set *FLCn_CTRL.erase_code* to 0xAA for mass erase.
5. Set *FLCn_CTRL.mass_erase* to 1 to start the mass erase operation.
6. The *FLCn_CTRL.busy* bit is set by the flash controller while the mass erase is in progress and the *FLCn_CTRL.mass_erase* and *FLCn_CTRL.busy* are cleared by the flash controller when the mass erase is complete.
7. *FLCn_INTFL.done* is set by the flash controller when the mass erase completes and if an error occurred, the *FLCn_INTFL.access_fail* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.
8. Set *FLCn_CTRL.unlock* to any value other than 0x2 to re-lock the flash instance.

6.3 Registers

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

See *Table 2-1. Field Access Definitions* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 6-3. Flash Controller Register Summary

Offset	Register Name	Access	Description
[0x0000]	*FLCn_ADDR*	R/W	Flash Controller Address Pointer Register
[0x0004]	*FLCn_CLKDIV*	R/W	Flash Controller Clock Divisor Register
[0x0008]	*FLCn_CTRL*	R/W	Flash Controller Control Register
[0x0024]	*FLCn_INTFL*	R/W	Flash Controller Interrupt Register
[0x0028]	*FLCn_ECCDATA*	R/W	Flash Controller Error Correction Code Data
[0x0030]	*FLCn_DATA0*	R/W	Flash Controller Data Register 0
[0x0034]	*FLCn_DATA1*	R/W	Flash Controller Data Register 1
[0x0038]	*FLCn_DATA2*	R/W	Flash Controller Data Register 2

6.3.1 Register Details

Table 6-4: Flash Controller Address Pointer Register

Flash Controller Address Pointer			FLCn_ADDR	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	addr	R/W	*	**Flash Address** This field contains the target address for a write operation. A valid internal flash memory address is required for all write operations. *The reset value for this field is always 0x0010 0000.

Table 6-5: Flash Controller Clock Divisor Register

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

Table 6-6: Flash Controller Control Register

Flash Controller Control Register			FLCn_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:28	unlock	R/W	0	**Flash Unlock** Write the unlock code, 2, prior to any flash write or erase operation to unlock the Flash. Writing any other value to this field locks the internal flash. 2: Flash unlock code
27:26	-	RO	-	**Reserved**
25	lve	R/W	0	Low Voltage Enable Set this field to 1 to enable low voltage operation for the flash memory. See *Core Operating Voltage Range Selection* for detailed usage information on this setting. 0: Low voltage operation disabled (Default). 1: Low voltage operation enabled. *Note: The PWRSEQ_LPCN.ovr field must be set to 0b00 prior to setting this field to 1.*

Flash Controller Control Register			FLCn_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
24	pend	RO	0	**Flash Busy Flag** When this field is set, writes to all flash registers except the *FLCn_INTFL* register are ignored by the Flash Controller. This bit will be cleared by hardware once the flash becomes accessible. *Note: If the Flash Controller is busy (FLCn_CTRL.pend = 1), reads, writes and erase operations are not allowed and result in an access failure (FLCn_INTFL.af = 1).* 0: Flash idle 1: Flash busy
23:16	-	RO	0	**Reserved**
15:8	erase_code	R/W	0	**Erase Code** Prior to an erase operation this field must be set to 0x55 for a page erase or 0xAA for a mass erase. The flash must be unlocked prior to setting the erase code. This field is automatically cleared after the erase operation is complete. 0x00: Erase disabled. 0x55: Page erase code. 0xAA: Enable mass erase.
7:3	-	RO	0	**Reserved**
2	pge	R/W1	0	**Page Erase** Write a 1 to this field to initiate a page erase at the address in *FLCn_ADDR.addr*. The flash must be unlocked prior to attempting a page erase, see *FLCn_CTRL.unlock* for details. The Flash Controller hardware clears this bit when a page erase operation is complete. 0: No page erase operation in process or page erase is complete. 1: Write a 1 to initiate a page erase. If this field reads 1, a page erase operation is in progress.
1	me	R/W1	0	**Mass Erase** Write a 1 to this field to initiate a mass erase of the internal flash memory. The flash must be unlocked prior to attempting a mass erase, see *FLCn_C.unlock* for details. The Flash Controller hardware clears this bit when the mass erase operation completes. 0: No operation 1: Initiate mass erase
0	wr	R/W1O	0	**Write** If this field reads 0, no write operation is pending for the flash. To initiate a write operation, set this bit to 1 and the Flash Controller will write to the address set in the *FLCn_ADDR* register. 0: No write operation in process or write operation complete. 1: Write 1 to initiate a write operation. If this field reads 1, a write operation is in progress. *Note: This field is protected and cannot be set to 0 by application code.*

Table 6-7: Flash Controller Interrupt Register

Flash Controller Interrupt Register			FLCn_INTFL	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:10	-	RO	0	**Reserved**
9	afie	R/W	0	**Flash Access Fail Interrupt Enable** Set this bit to 1 to enable interrupts on flash access failures. 0: Disabled 1: Enabled
8	doneie	R/W	0	**Flash Operation Complete Interrupt Enable** Set this bit to 1 to enable interrupts on flash operations complete. 0: Disabled 1: Enabled
7:2	-	RO	0	**Reserved** Do not modify this field.
1	af	R/W0C	0	**Flash Access Fail Interrupt Flag** This bit is set when an attempt is made to write or erase the flash while the flash is busy or locked. Only hardware can set this bit to 1. Writing a 1 to this bit has no effect. This bit is cleared by writing a 0. 0: No access failure has occurred. 1: Access failure occurred.
0	done	R/W0C	0	**Flash Operation Complete Interrupt Flag** This flag is automatically set by hardware after a flash write or erase operation completes. 0: Operation not complete or not in process. 1: Flash operation complete.

Table 6-8: Flash Controller ECC Data Register

Flash Controller ECC Data			FLCn_ECCDATA	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:25	-	RO	0	**Reserved** Do not modify this field.
24:16	odd	R	0	**Error Correction Code Odd Data** 9-bit ECC data recorded from the last flash read memory location of odd address of the even/odd pair of 128-bit flash memory content.
15:9	-	RO	0	**Reserved** Do not modify this field.
8:0	even	R	0	**Error Correction Code Even Data** 9-bit ECC data recorded from the last flash read memory location of even address of the even/odd pair of 128-bit flash memory content.

Table 6-9: Flash Controller Data 0 Register

Flash Controller Data 0			FLCn_DATA0	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Flash Data 0** Flash data for bits 31:0.

Table 6-10: Flash Controller Data Register 1

Flash Controller Data 1			FLCn_DATA1	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Flash Data 1** Flash data for bits 63:32

Table 6-11: Flash Controller Data Register 2

Flash Controller Data 2			FLCn_DATA2	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Flash Data 2** Flash data for bits 95:64

Table 6-12: Flash Controller Data Register 3

Flash Controller Data 3			FLCn_DATA3	[0x003C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	data	R/W	0	**Flash Data 3** Flash data for bits 127:96.

Table 6-13. Flash Controller Access Control Register

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

7. Dynamic Voltage Scaling Controller

The Dynamic Voltage Scaling (DVS) Controller works in conjunction with the SIMO to optimally operate the **MAX78000** at the lowest practical voltage in a battery powered application. The ability to adaptively adjust the voltage provides a significant reduction in dynamic power consumption when compared to a fixed operating voltage.

The DVS monitors the system operating voltage and adjusts the SIMO voltage downward periodically until the device is operating at the optimal power/performance point. This creates a closed loop system enabling Dynamic Voltage Scaling of the operating voltage during high power consumption and low power consumption operations.

When switching the system clock from a lower frequency source to a higher frequency source, firmware must set the SIMO to a pre-determined "safe" value prior to switching the system clock source. Once the system clock is set to the higher frequency source, the DVS can be setup to monitor the voltage using the new system clock source.

The DVS provides the following features:

- Continuous monitoring with a programmable monitoring sample rate
- Controlled transition to a programmable operating point
- Independent high and low operating limits for safe, bounded operation
- Programmable adjustment rate when an adjustment is required
- Interrupt capability for error conditions

7.1 Operation

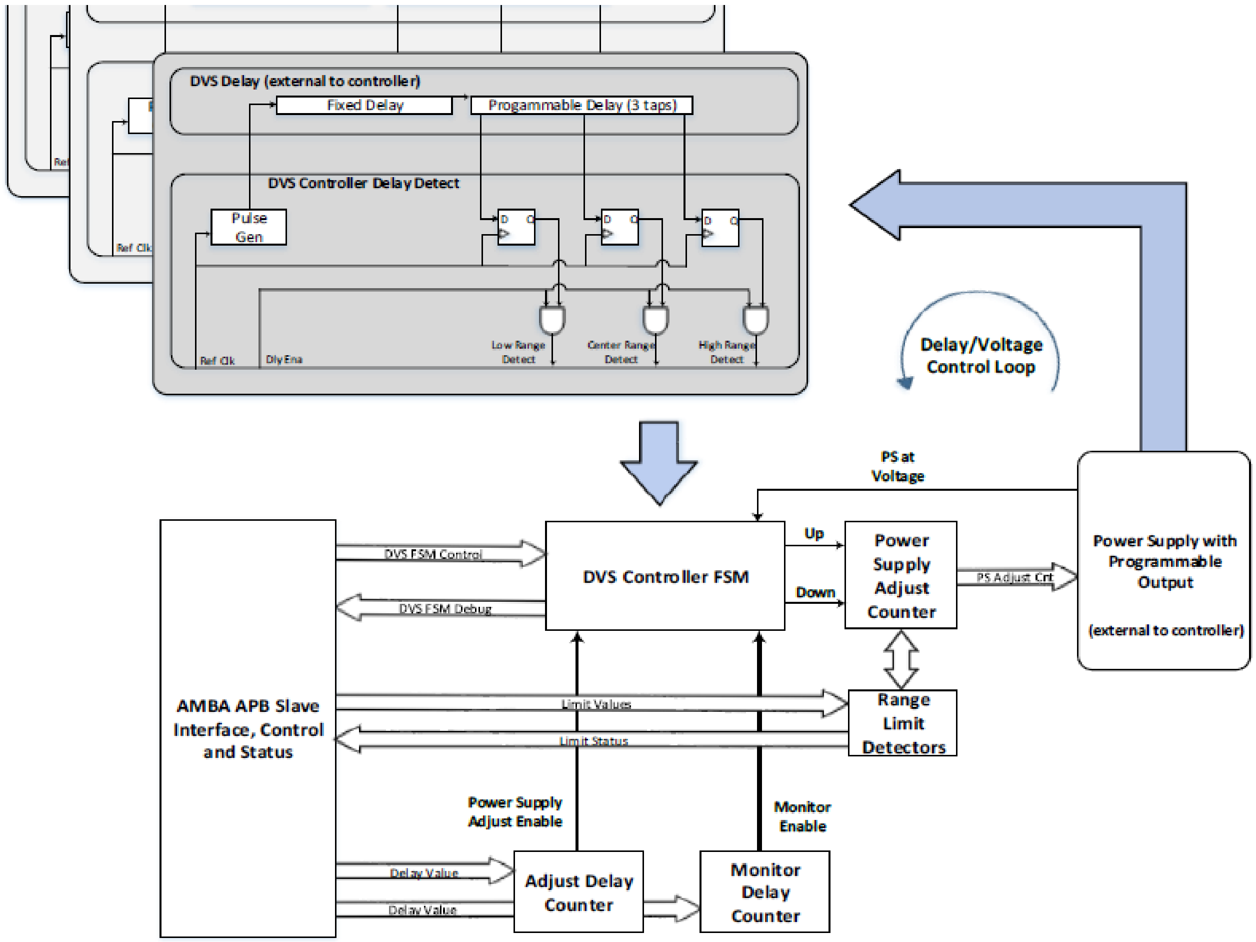
The DVS is disabled by default on a Power On Reset and firmware must configure the DVS prior to enabling the DVS. The DVS supports three modes of operation:

1. Measurement only
2. Automatic adjustment
3. Direct transition

7.1.1 Measurement Only Mode

In measurement only mode, the DVS state machine remains in an idle state until a measurement request is made by the Monitor Delay Counter.

Figure 7-1: DVS Controller Block Diagram



7.2 DVS Registers

See *Table 2-3: APB Peripheral Base* Address Map for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 13-8: I2S Register Summary*. 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 2-1. Field Access Definitions* 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 7-1: Register Summary

Offset	Register Name	Description
[0x0000]	DVS_CTRL	Control Register
[0x0004]	DVS_STAT	Status Register
[0x0008]	DVS_DIR_CTRL	Direct Control Register
[0x000C]	DVS_MON	Monitor Delay Register
[0x0010]	DVS_ADJ_UP	Adjustment Up Delay Register
[0x0014]	DVS_ADJ_DWN	Adjustment Down Delay Register

Offset	Register Name	Description
[0x0018]	DVS_THRES_CMP	Power Supply Threshold Compare Register
[0x001C]	DVS_TAP_SEL0	Tap Select 0 Register
[0x0020]	DVS_TAP_SEL1	Tap Select 1 Register
[0x0024]	DVS_TAP_SEL2	Tap Select 2 Register
[0x0028]	DVS_TAP_SEL3	Tap Select 3 Register
[0x002C]	DVS_TAP_SEL4	Tap Select 4 Register

7.2.1 DVS Register Details

Table 7-2: DVS Control Register

DVS Control Register			DVS_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field
25	adj_abort	R/W	0	**Bit Field Control**

Table 7-3: DVS Status Register

DVS Status Register			DVS_STAT	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field
25	adj_abort	R/W	0	**Bit Field Control**

Table 7-4: DVS Direct Control Register

DVS Direct Control			DVS_DIR_CTRL	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field
25	adj_abort	R/W	0	**Bit Field Control**

Table 7-5: DVS Monitor Register

DVS Monitor			DVS_MON	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field
25	adj_abort	R/W	0	**Bit Field Control**

Table 7-6: DVS Adjustment Up Register

DVS Adjustment Up			**DVS_ADJ_UP**	**[0x0010]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field
25	adj_abort	R/W	0	**Bit Field Control**

Table 7-7: DVS Adjustment Down Register

DVS Adjustment Down			**DVS_ADJ_DWN**	**[0x0014]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field
25	adj_abort	R/W	0	**Bit Field Control**

Table 7-8: DVS Threshold Compare Register

DVS Threshold Compare			**DVS_THRES_CMP**	**[0x0018]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field
25	adj_abort	R/W	0	**Bit Field Control**

Table 7-9: DVS Tap Select 0 Register

DVS Tap Select 0			**DVS_TAP_SEL0**	**[0x001C]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field
25	adj_abort	R/W	0	**Bit Field Control**

Table 7-10: DVS Tap Select 1 Register

DVS Tap Select 1			**DVS_TAP_SEL1**	**[0x0020]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field
25	adj_abort	R/W	0	**Bit Field Control**

Table 7-11: DVS Tap Select 2 Register

DVS Tap Select 2			**DVS_TAP_SEL2**	**[0x0024]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field

DVS Tap Select 2			DVS_TAP_SEL2	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
25	adj_abort	R/W	0	**Bit Field Control**

Table 7-12: DVS Tap Select 3 Register

DVS Tap Select 3			DVS_TAP_SEL3	[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field
25	adj_abort	R/W	0	**Bit Field Control**

Table 7-13: DVS Tap Select 4 Register

DVS Tap Select 4			DVS_TAP_SEL4	[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:26	0	R	0	**Reserved for Future Use** Do not modify this field
25	adj_abort	R/W	0	**Bit Field Control**

8. Standard DMA (DMA)

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

DMA transfers are one of three types:

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

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

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

8.1 Instances

There is one instance of the DMA, generically referred to as DMAm. Each instance provides 4 channels, generically referred to as DMA_CHn. Each instance of the DMA has a set of interrupt registers common to all its channels, and a set of registers unique to each channel instance.

Table 8-1: MAX78000 DMA and Channel Instances

DMAm Instance	DMA_CHn Channel Instance
DMA0	DMA_CH0
	DMA_CH1
	DMA_CH2
	DMA_CH3

8.2 DMA Channel Operation (DMA_CH)

8.2.1 DMA Channel Arbitration and DMA Bursts

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

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

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

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

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

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

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

Only an error condition can interrupt an ongoing data transfer.

8.2.2 DMA Source and Destination Addressing

The source and destination for DMA transfers are dictated by the request select dedicated to the peripheral instance. The *DMA_CHn_CTRL.request* field dictates the source and destination for a channel's DMA transfer as shown in *Table 8-2.* The *DMA_CHn_SRC* and *DMA_CHn_DST* registers hold the source and/or destination memory addresses, depending on the specific operation.

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

Table 8-2: MAX78000 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	Reserved		
0x07	I2C0	I2C0 Receive FIFO	*DMA_CHn_DST*
0x08	I2C1	I2C1 Receive FIFO	*DMA_CHn_DST*
0x09	Reserved		
0x0A	I2C2	I2C2 Receive FIFO	*DMA_CHn_DST*
0x0B:0x0D	Reserved		
0x0E	UART2	UART2 Receive FIFO	*DMA_CHn_DST*
0x0F	Reserved		
0x10	AES	AES Receive	*DMA_CHn_DST*
0x11:0x1B	Reserved		
0x1C	LPUART0	LPUART Receive	*DMA_CHn_DST*
0x1D	Reserved		
0x1E	I2S	I2S Receive	
0x1F:0x20	Reserved		
0x21	SPI0	*DMA_CHn_SRC*	SPI0 Transmit FIFO
0x22	SPI0	*DMA_CHn_SRC*	SPI1 Transmit FIFO
0x23	Reserved		
0x24	UART0	*DMA_CHn_SRC*	UART0 Transmit FIFO
0x25	UART1	*DMA_CHn_SRC*	UART1 Transmit FIFO
0x26	Reserved		
0x27	I2C0	*DMA_CHn_SRC*	I2C0 Transmit FIFO
0x28	I2C1	*DMA_CHn_SRC*	I2C1 Transmit FIFO
0x29	Reserved		
0x2A	I2C2	*DMA_CHn_SRC*	I2C2 Transmit FIFO
0x2B	Reserved		
0x2C	CRC	*DMA_CHn_SRC*	CRC
0x2D	Reserved		
0x2E	UART2	*DMA_CHn_SRC*	UART2 Transmit FIFO

DMA_CHn_CTRL.request	Peripheral	DMA Source	DMA Destination
0x2F	Reserved		
0x30	AES	*DMA_CHn_SRC*	AES
0x31:0x3B	Reserved		
0x3C	LPUART	*DMA_CHn_SRC*	LPUART Transmit FIFO
0x3D	Reserved		
0x3E	I2S	*DMA_CHn_SRC*	I2S TransmitFIFO
0x3F	Reserved		

8.2.3 Data Movement from Source to DMA

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

Table 8-3: Data Movement from Source to DMA FIFO

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

8.2.4 Data Movement from DMA to Destination

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

Table 8-4: Data Movement from the DMA FIFO to Destination

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

8.3 Usage

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

1. Ensure *DMA_CHn_CTRL.en, DMA_CHn_CTRL.rlden* = 0, and *DMA_CHn_STATUS.ctz_if* = 0.
2. If using memory for the destination of the DMA transfer, configure *DMA_CHn_DST* to the starting address of the destination in memory.
3. If using memory for the source of the DMA transfer, configure *DMA_CHn_SRC* 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_CNTRLD.rlden* to 1 in this step:
6. Configure *DMA_CHn_CTRL.req_sel* to select the transfer operation associated with the DMA channel.
7. Configure *DMA_CHn_CTRL.burst* for the desired burst size.
8. Configure *DMA_CHn_CTRL.priority* to set the channel priority relative to other DMA channels.
9. Configure *DMA_CHn_CTRL.dst_width* to dictate the number of bytes written in each transaction.
10. If desired, set *DMA_CHn_CTRL.dst_inc* to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.
11. Configure *DMA_CHn_CTRL.src_width* to dictate the number of bytes read in each transaction.
12. If desired, set *DMA_CHn_CTRL.src_inc* to 1 to enable automatic incrementing of the *DMA_CHn_DST* register upon every AHB transaction.
13. If desired, set *DMA_CHn_CTRL.chd_ien* = 1 to generate an interrupt when the channel becomes disabled. The channel becomes disabled when the DMA transfer completes or a bus error occurs.
14. If desired, set *DMA_CHn_CTRL.ctz_ien* 1 to generate an interrupt when the *DMA_CHn_CNT* register is decremented to zero.
15. If using the reload feature, configure the reload registers to set the destination, source, and count for the following DMA transaction.
16. If desired, enable the channel timeout feature described in *Channel Timeout Detect*. Clear *DMA_CHn_CTRL.to_prescale* to 0x0 to disable the channel timeout feature.
17. Set *DMA_CHn_CNTRLD.rl_den* to 1 to enable the reload feature.
18. Set *DMA_CHn_CTRL.ch_en* = 1 to immediately start the DMA transfer.
19. Wait for the interrupt flag to become 1 to indicate the completion of the DMA transfer.

8.4 Count-To-Zero (CTZ) Condition

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

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

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

8.5 Chaining Buffers

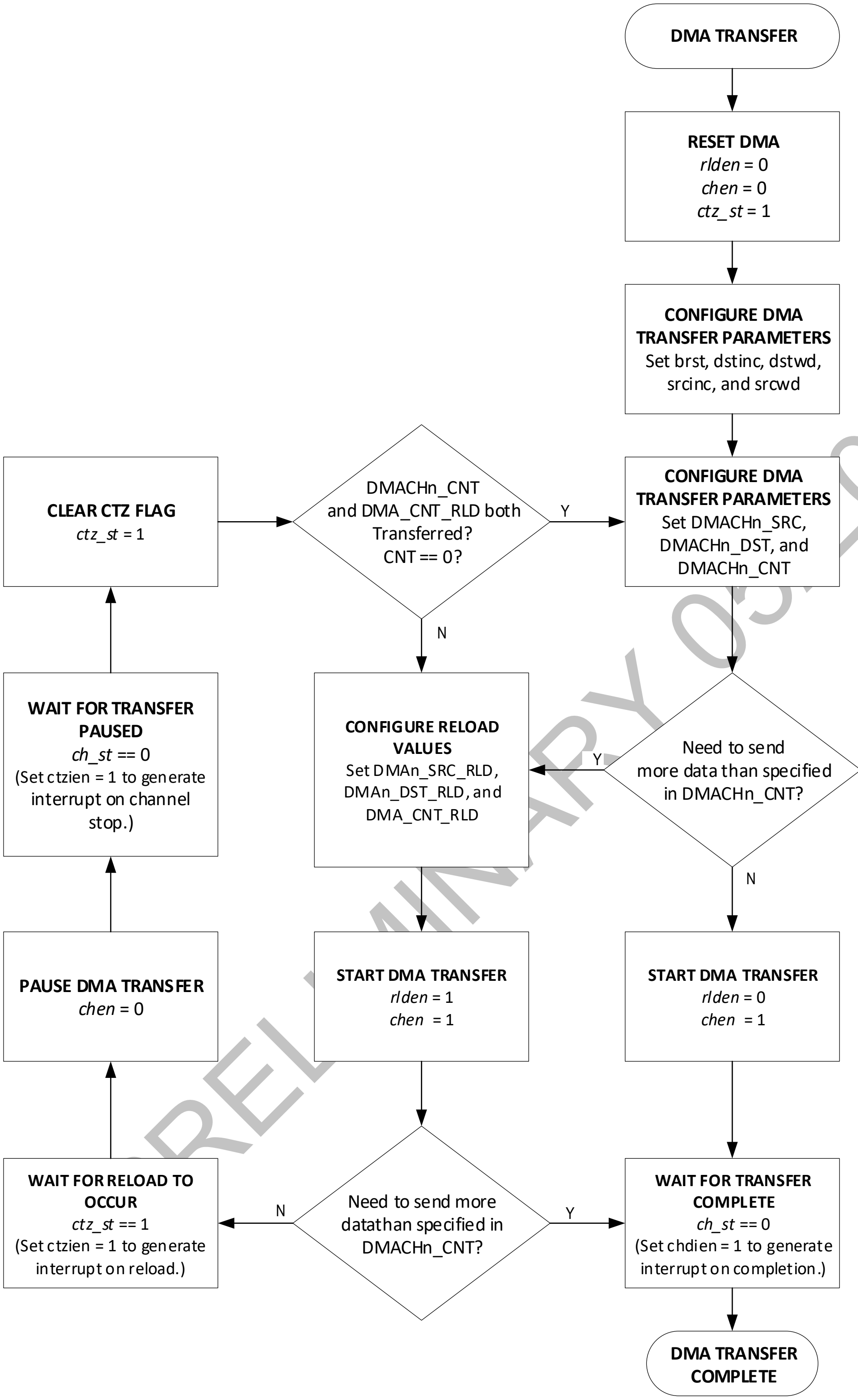
Chaining buffers reduces the DMA ISR response time and allows DMA to service requests without intermediate processing from the CPU. *Figure 8-1: DMA Block-Chaining Flowchart* shows the procedure for generating a DMA transfer using one or more chain buffers.

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

- *DMA_CHn_SRC*
- *DMA_CHn_DST*
- *DMA_CHn_CNT*
- *DMA_CHn_SRCRLD*
- *DMA_CHn_DST*
- *DMA_CHn_CNTRLD*

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

Figure 8-1: DMA Block-Chaining Flowchart



8.6 DMA Interrupts

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

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

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

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

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

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

8.7 Channel Timeout Detect

Each channel can optionally generate an interrupt when its associated peripheral does not request a transfer in a user-configurable period. When the timeout start conditions are met, an internal 10-bit counter begins incrementing at a frequency determined by the AHB clock, *DMA_CHn_CTRL.to_clkdiv*, and *DMA_CHn_CTRL.to_per* shown in *Table 8-5: DMA Channel Timeout Configuration*. A channel timeout event is generated if the timer is not reset by one of the events listed below before the timeout period expires.

Table 8-5: DMA Channel Timeout Configuration

DMA_CHn_CTRL.to_clkdiv	**Timeout Period (µs)**
0	*Channel timeout disabled.*
1	$\frac{2^8 * [Value\ from\ \mathrm{DMA\_CHn\_CTRL}.to\_\mathrm{per}]}{f_{HCLK}}$
2	$\frac{2^{16} * [Value\ from\ \mathrm{DMA\_CHn\_CTRL}.tosel]}{f_{HCLK}}$
3	$\frac{2^{24} * [Value\ from\ \mathrm{DMA\_CHn\_CTRL}.tosel]}{f_{HCLK}}$

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

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

The timer is reset whenever:

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

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

8.8 Memory-to-Memory DMA

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

8.9 DMA Registers

See *Table 2-3: APB Peripheral Base Address Map* 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 8-6: DMA Register Summary*. 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 2-1. Field Access Definitions* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 8-6: DMA Register Summary

Offset	Register	Description
[0x0000]	*DMAn_INTEN*	DMA Control register
[0x0004]	*DMAn_INTFL*	DMA Interrupt Status register

8.9.1 DMA Register Details

Table 8-7: DMAn Interrupt Flag Register

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

Table 8-8: DMAn Interrupt Enable Register

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

8.10 DMA Channel Register Summary

Table 8-9: Standard DMA Channel 0 to Channel 3 Register Summary

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

8.11 DMA Channel Registers

See *Table 2-3: APB Peripheral Base Address Map* 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 8-10: DMA Channel Registers Summary*. 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 2-1. Field Access Definitions* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 8-10: DMA Channel Registers Summary

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

8.11.1 DMA Channel Register Details

Table 8-11: DMA_CHn 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. *DMAn_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. *DMAn_INTFL.ch<n>_ipend* bit is set to 1 whenever *DMA_CHn_STATUS.status* changes from 1 to 0.
29	-	RO	0	**Reserved**
28:24	*burst_size*	R/W	0	**Burst Size** The number of bytes transferred into and out of the DMA FIFO in a single burst. 0b00000: 1 byte 0b00001: 2 bytes 0b00010: 3 bytes ... 0b11111: 32 bytes
23	-	RO	0	**Reserved**
22	*dstinc*	R/W	0	**Destination Increment Enable** This bit enables the automatic increment of the *DMA_CHn_DST* register upon every AHB transaction. This bit is ignored for a DMA transmit to peripherals. 0: Disabled 1: Enabled
21:20	*dstwd*	R/W	0	**Destination Width** Indicates the width of each AHB transaction to the destination peripheral or memory (the actual width might be less than this if there are insufficient bytes in the DMA FIFO for the full width). 0: One byte 1: Two bytes 2: Four bytes 3: Reserved
19	-	RO	0	**Reserved**
18	*srcinc*	R/W	0	**Source Increment on AHB Transaction Enable** This bit enables the automatic increment of the *DMA_CHn_SRC* register upon every AHB transaction. This bit is ignored for a DMA receive from peripherals. 0: Disabled 1: Enabled
17:16	*srcwd*	R/W	0	**Source Width** Indicates the width of each AHB transaction from the source peripheral or memory. The actual width might be less than this if the *DMA_CHn_CNT* register indicates a smaller value. 0: One byte 1: Two bytes 2: Four bytes 3: Reserved

DMA Channel n Control				DMA_CHn_CTRL [0x0100]
Bits	**Field**	**Access**	**Reset**	**Description**
15:14	*to_clkdiv*	R/W	0	**Timeout Timer Clock Pre-Scale Select** Selects the Pre-Scale divider for the timer clock input. 0: Timer disabled. 1: f_{HCLK} / 28 2: f_{HCLK} / 216 3: f_{HCLK} / 224
13:11	*to_per*	R/W	0	**Timeout Period Select** Selects the number of pre-scaled clocks seen by the channel timer before a timeout condition is generated. The value is approximate because of synchronization delays between timers 0: 3-4 1: 7-8 2: 15-16 3: 31-32 4: 63-64 5: 127-128 6: 255-256 7: 511-512
10	*to_wait*	R/W	0	**Request DMA Timeout Timer Wait Enable** 0: Start timer immediately when enabled. 1: Delay timer start until after the first DMA transaction occurs.
9:4	*request*	R/W	0	**Request Select** Selects the source and destination for the transfer as shown in *Table 8-2: MAX78000 DMA Source and Destination by Peripheral*.
3:2	*pri*	R/W	0	**Channel Priority** Sets the priority of the channel relative to other channels of DMAm. Channels of the same priority are serviced in a round-robin fashion. 0: Highest priority 1: … 2: … 3: Lowest priority
1	*rlden*	R/W	0	**Reload Enable** Setting this bit to 1 allows reloading the *DMA_CHn_SRC*, *DMA_CHn_DST*, and *DMA_CHn_CNT* registers with their corresponding reload registers upon CTZ. *Note: This bit is also writeable in the DMA_CHn_CNTRLD register.*
0	*en*	R/W	0	**Channel Enable** This bit is automatically cleared when *DMA_CHn_STATUS.status* changes from 1 to 0. 0: Disabled 1: Enabled

Table 8-12: DMA Status Register

DMA Channel n Status				DMA_CHn_STATUS [0x0104]
Bits	**Field**	**Access**	**Reset**	**Description**
31:7	-	RO	0	**Reserved for Future Use** Do not modify this field from its reset default value.

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

Table 8-13: DMA_CHn Source Register

DMA Channel n Source			**DMA_CHn_SRC**	**[0x0108]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:0	addr	R/W	0	**Source Device Address** For peripheral transfers, the actual address field is either ignored or forced to zero because peripherals only have one location to read/write data based on the request select chosen. If *DMA_CHn_CTRL.srcinc* = 1, then this register is incremented on each AHB transfer cycle by one, two, or four bytes depending on the data width. If *DMA_CHn_CTRL.srcinc* = 0, this register remains constant. If a CTZ condition occurs while *DMA_CHn_CTRL.rlden* = 1, then this register is reloaded with the contents of the *DMA_CHn_SRCRLD* register.

Table 8-14: DMA Channel n Destination Register

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

Table 8-15: DMA Channel n Count Register

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

Table 8-16: DMA Channel n Source Reload Register

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

Table 8-17: DMA Channel n Destination Reload Register

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

Table 8-18: DMA Channel n Count Reload Register

DMA Count Reload			DMA_CHn_CNTRLD	[0x011C]
Bits	**Field**	**Access**	**Reset**	**Description**
31	rlden	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_rld	R/W	0	**Count Reload Value.** If *DMA_CHn_CNTRLD.rlden* = 1, then the value of this register is loaded into *DMA_CHn_CNT* upon a CTZ condition.

9. Analog to Digital Converter and Comparators (ADC)

The Analog to Digital Converter (ADC) is a 10-bit sigma-delta ADC with a single-ended input multiplexer and an integrated reference generator. The multiplexer selects an input channel from either of the 8 external analog input signals or the internal power supply inputs. The external analog input signals are defined as alternate functions on GPIO as shown in Table 9-1, below.

. The 10-bit ADC conversions are stored as a 16-bit value selectable as most-significant bit (MSB) or least-significant bit (LSB) aligned. The 8 external analog inputs can be configured as 4 two-input comparators with interrupt capabilities.

9.1 Features

8MHz maximum ADC clock rate
Two reference sources, an internal 1.22V bandgap or the VDDA analog supply
8 External analog inputs that can be configured as 4 two-input comparators
10 Internal power supply monitor inputs
Fixed 10-bit word conversion time of 1024 ADC clock cycles
Programmable out-of-range (limit) detection
Interrupt generation for limit detection, conversion start, conversion complete, and internal reference powered on
Serial ADC data measurements
ADC conversion 10 output either MSB or LSB aligned

9.2 Instances

~~Table 5-1 lists the locations of the External ADC inputs for each package option.~~

*Table 9-1: **MAX78000** ADC Peripheral Pins*

ALTERNATE FUNCTION	MAPPING	81-CTBGA
AIN0/AIN0N	AF1	P2.0
AIN1/AIN0P	AF1	P2.1
AIN2/AIN1N	AF1	P2.2
AIN3/AIN1P	AF1	P2.3
AIN4/AIN2N	AF1	P2.4
AIN5/AIN2P	AF1	P2.5
AIN6/AIN3N	AF1	P2.6
AIN7/AIN3P	AF1	P2.7

9.3 Architecture

The ADC is a first-order sigma-delta converter with a 10-bit output. The ADC operates at a maximum frequency of 8MHz with a fixed-sample rate as shown in *Equation 9-1*. Details of selecting the ADC clock frequency, f_{adcclk}, are covered in the *Clock Configuration* section.

Equation 9-1: ADC 10-bit Word Sample Rate

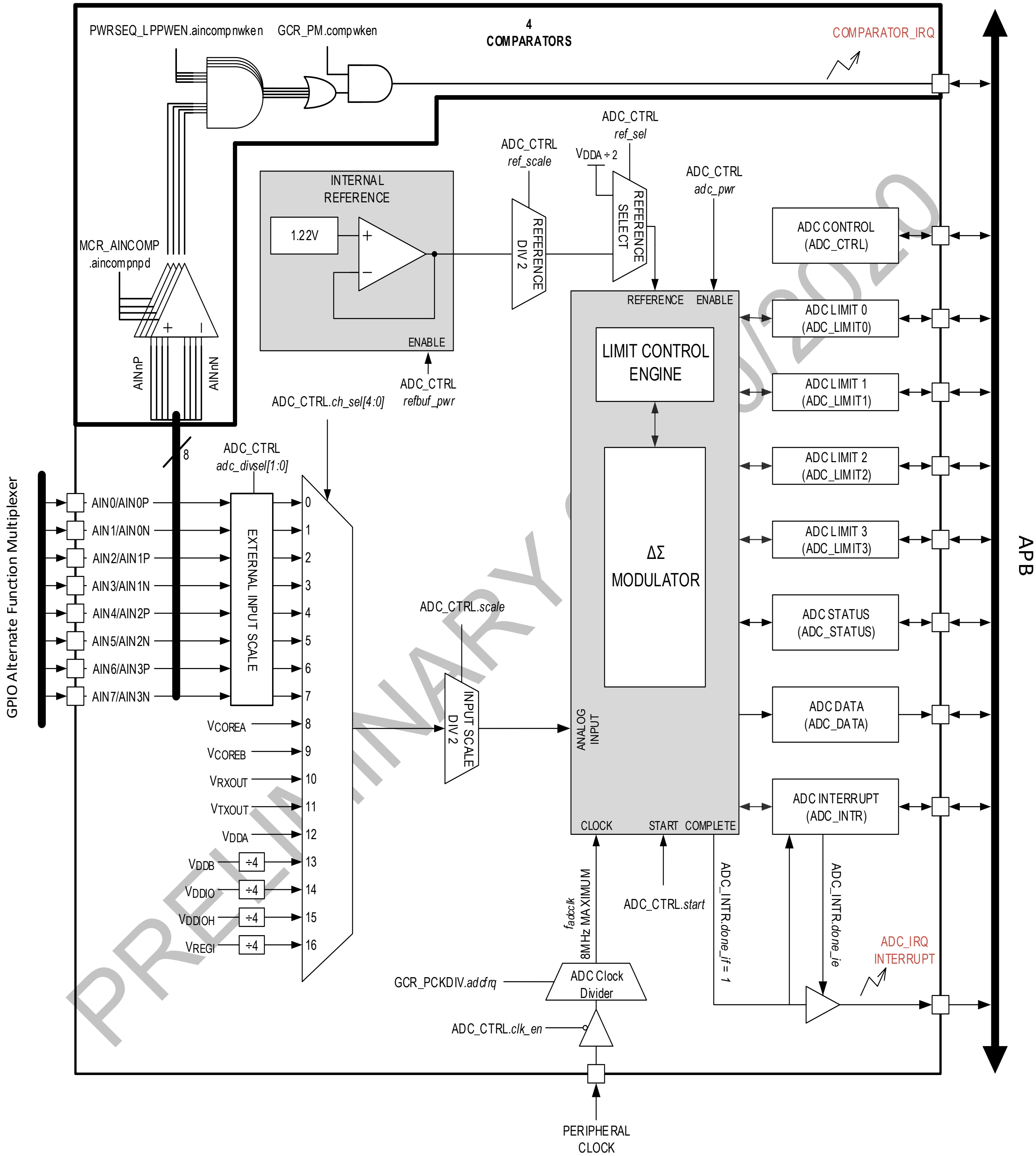
$$t_{adc\_sample} = 1024 \times \left(\frac{1}{f_{adcclk}}\right)$$

ADC offset is factory trimmed and automatically loaded into the ADC controller during system power-up.

The ADC uses a switched capacitor network to perform the conversion; this results in dynamic switching current and requires settling time for the external analog input signals (AIN0 – AIN7). This dynamic switching current sets the upper limit of the source impedance of the external analog input signals to approximately 10kΩ.

The ADC supports a gain of 2× to provide additional conversion resolution if the input signals are less than half the reference voltage.

Figure 9-1: Analog to Digital Converter Block Diagram



9.4 Clock Configuration

The ADC clock, *adcclk*, is controlled by the GCR_PCKDIV.*adcfrq* register field. Configure this field for the target ADC sample frequency. The maximum clock supported by the ADC is 8MHz. The divisor selection, GCR_PCKDIV.*adcfrq*, for the ADC depends on the peripheral clock. *Equation 9-2* shows the calculation for the ADC clock frequency, where:

$$f_{PCLK} = {}^{f_{SYSCLK}}/_{2}$$

Equation 9-2: ADC Clock Frequency

$$f_{adcclk} = \frac{f_{PCLK}}{\text{GCR\_PCKDIV}.adcfrq}$$

The GCR_PCKDIV.adcfrq register field setting must result in a value for $\mathrm{f_{adcclk}} \leq 8\mathrm{MHz}$ as shown in *Table 9-2* with the System Clock set as the 96MHz high frequency oscillator.

Table 9-2: ADC Clock Frequency and ADC Conversion Time ($f_{SYSCLK} = 100MHz, f_{PCLK} = 50MHz$)

GCR_PCKDIV.*adcfrq*[3:0]	**ADC Clock Frequency (Hz)** f_{adcclk}	**10-Bit Word Conversion Time (µs)** t_{adc_sample}
0x– 0x7	Invalid	Invalid
0x8	6,000,000	171
0x9	5,333,333	192
0xA	4,800,000	214
0xB	4,363,636	235
0xC	4,000,000	256
0xD	3,692,308	278
0xE	3,428,571	299
0xF	3,200,000	320

9.5 Power-Up Sequence

Complete the following steps to configure the ADC:

1. Disable the ADC clock by setting ADC_CTRL.*clk_en* to 0.
2. Set the ADC clock (adcclk) using GCR_PCKDIV.*adcfrq*. See Clock Configuration
3. Enable the ADC clock by setting ADC_CTRL.*clk_en* to 1
4. Clear the ADC reference ready interrupt flag by writing a 1 to ADC_INTR.*ref_ready_if*.
5. Optionally enable the ADC reference ready interrupt (ADC_INTR.*ref_ready_ie* = 1), and enable the ADC interrupt vector (ADC IRQ).
6. Select one of the following ADC reference sources:
7. Internal 1.22V bandgap reference (ADC_CTRL.*ref_sel* = 0).
8. VDDA ÷ 2 reference (ADC_CTRL.*ref_sel* = 1).
9. Complete the following steps to enable power:
10. Set ADC_CTRL.*pwr* to 1 to turn on the ADC.
11. Set ADC_CTRL.*refbuf_pwr* to 1 to turn on the internal reference buffer If using the internal bandgap reference.
12. Wait until hardware sets the ADC_INTR.*ref_ready_if* bit to 1 indicating the charge pump is fully stabilized.
13. Clear the ADC reference ready interrupt flag by writing 1 to ADC_INTR.*ref_ready_if*.
14. Optionally disable the ADC reference ready interrupt (ADC_INTR.*ref_ready_ie* = 0).

9.6 Conversion

After the power-up sequence is complete, the ADC is ready for data conversion. Complete the following steps to perform a data conversion.

1. Select the ADC input channel for the conversion by setting ADC_CTRL.*ch_sel* field. See ADC Channel Select for details.
2. Optionally set input and reference scaling. See Reference Scaling and Input Scaling for details on each input channel's scale requirements.
3. Set the data alignment for the conversion output data using the ADC_CTRL.*data_align* field, 0 for LSB alignment or 1 for MSB alignment. See Table 9-4 for alignment details of the DATA register.
4. Clear the ADC done interrupt flag by writing 1 to the ADC_INTR.*done_if*.
5. Optionally enable the ADC done interrupt (ADC_INTR.*done_ie* = 1), and enable the ADC interrupt vector (ADC IRQ). See the Interrupt chapter for details.
6. Start the ADC conversion by setting ADC_CTRL.*start* to 1.
7. Poll the ADC_INTR.*done_if* flag until you read 1, or wait for the ADC interrupt to occur if enabled in step 5.
8. Read the data from the ADC_DATA.*data*, and clear the ADC done interrupt flag by writing 1 to ADC_INTR.*done_if*.

9.7 Reference Scaling and Input Scaling

For small signals, the ADC input, ADC reference or both can be scaled by 50%. This enables flexibility to achieve better resolution on the ADC conversion. Each input channel, supports the default of no scaling of the input (ADC_CTRL.*scale* = 0) and no scaling of the reference (ADC_CTRL.*ref_scale* = 0). The following sections describe the scale options for each of the ADC input channels.

9.7.1 AIN0 – AIN7 Scale Limitations

The external inputs, AIN0 through AIN7, support scaling of the input by 50%, the reference by 50%, or both by 50%. Also, the scaling can further be modified by additional factors of 2, 3, or 4 as defined by *ADC_CTRL.adc_divsel*. The scale settings for the given input signal and reference must satisfy the following equation to be valid:

Equation 9-3: Input and Reference Scale Requirements Equation

$$\frac{AINn}{2^{scale}} < \frac{V_{REF}}{2^{ref\_scale}}$$

9.7.2 Scale Limitations for All Other Input Channels

For the remaining internal input channels, the scale settings must either both be disabled, or both be enabled as shown in *Table 9-3*, *below*.

Table 9-3: Input and Reference Scale Support by ADC Input Channel

ADC Channel	ADC Input Signal	ADC_CTRL scale	ADC_CTRL ref_scale
8	VCOREA	0	0
		1	1
9	VCOREB	0	0
		1	1
10	VRXOUT	0	0
		1	1
11	VTXOUT	0	0
		1	1
12	VDDA	0	0
		1	1
13	VDDB/4	0	0
		1	1
14	VDDIO/4	0	0
		1	1
15	VDDIOH/4	0	0
		1	1
16	VREGI/4	0	0
		1	1

9.7.3 Data Conversion Output Alignment

The ADC outputs a total of 10-bits per conversion and stores the data in the DATA register LSB justified by default. *Table 9-4* shows the ADC data alignment based on the value of the *ADC_CTRL.data_align* bit.

Table 9-4: ADC Data Register Alignment Options

ADC_CTRL.data_align = 0																
	MSB															LSB
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC_DATA	0	0	0	0	0	0	data									

ADC_CTRL.data_align = 1																
	MSB															LSB
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC_DATA	data										0	0	0	0	0	0

9.7.4 Data Conversion Value Equations

Use the following equations to calculate the ADC data value for a conversion for the selected channel. If using the internal reference, VREF = 1.22V; otherwise VREF = VDDA.

Equation 9-4: ADC Data Calculation for Input Signal ADC_CTRL.ch_sel = 0x00 thru 0x07 (AIN0 – AIN7)

$$ADC\_DATA = round\left\{\left(\frac{\left(\frac{Input\ Signal}{2^{scale} * (adc\_divsel + 1)}\right)}{\left(\frac{V_{REF}}{2^{ref\_scale}}\right)}\right) \times (2^{10} - 1)\right\}$$

Note: Must satisfy Equation 9-3.

Equation 9-5: ADC Data Equation for Input Signal ADC_CTRL.ch_sel = 0x08 thru 0x0C (VCOREA, VCOREB, VRXOUT, VTXOUT, VDDA)

$$ADC\_DATA = round\left\{\left(\frac{\left(\frac{Input\ Signal}{2^{scale}}\right)}{\left(\frac{V_{REF}}{2^{ref\_scale}}\right)}\right) \times (2^{10} - 1)\right\}$$

Note: See Table 9-3 for limitations.

Equation 9-6: ADC Data Calculation Input Signal ADC_CTRL.ch_sel = 0x0D thru 0x10 (VDDB, VDDIO, VDDIOH, VREGI)

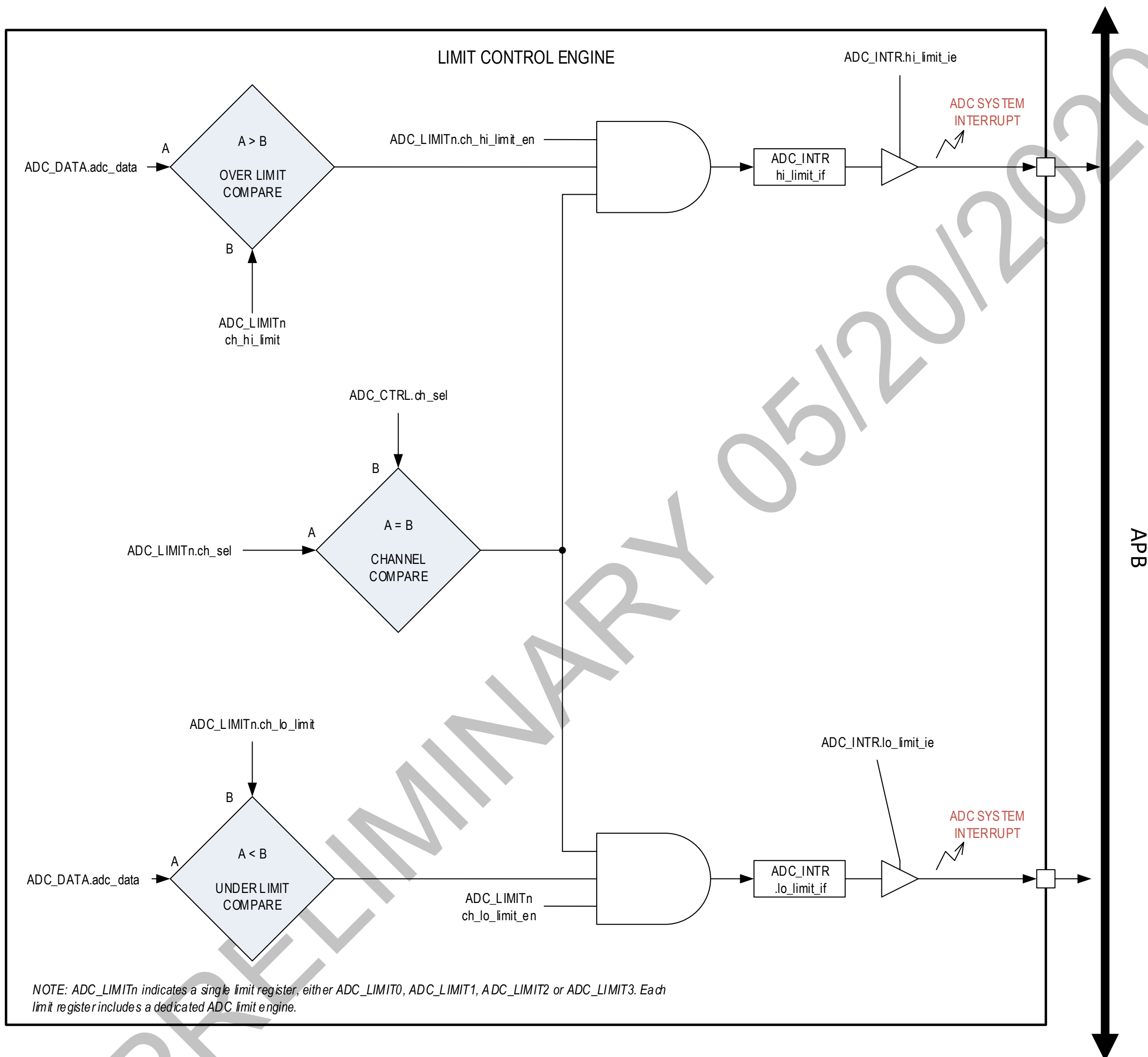
$$ADC\_DATA = round\left\{\left(\frac{\left(\frac{\frac{Input\ Signal}{4}}{2^{scale}}\right)}{\left(\frac{V_{REF}}{2^{ref\_scale}}\right)}\right) \times (2^{10} - 1)\right\}$$

Note: See Table 9-3 for limitations.

9.7.5 Data Limits and Out of Range Interrupts

Channel limits are implemented to minimize power consumption for power supply monitoring. The ADC includes four limit registers, *ADC_LIMIT0* to *ADC_LIMIT3*, that you can use to set a high limit, low limit, and the ADC channel number to apply the limits against. A block diagram of the limit engine for each of the four limit registers is shown in *Figure 9-2*.

Figure 9-2: ADC Limit Engine



When a measurement is taken on the ADC, the limit engine determines if the channel measured matches one of the channels selected by the limit registers. If it does and the data converted is above or below the high or low limit, an interrupt flag is set resulting in an ADC interrupt if the interrupt is enabled.

Complete the following steps to enable a high and low limit for an ADC input channel using the *ADC_LIMIT0* register. Perform these steps after the ADC is configured for measurement, and the configuration is identical for all four limit registers except for the limit register name:

1. Verify the ADC is not actively taking a measurement by checking *ADC_STATUS.active* until it reads 0.
2. Set *ADC_LIMIT0.ch_sel* field to the selected channel for the high and low limit.
3. Set the high limit, *ADC_LIMIT0.ch_hi_limit*, to the selected 10-bit trip point. When enabled, an ADC measurement greater than this field on the channel selected (*ADC_LIMIT0.ch_sel*) generates an ADC interrupt.
4. Set the low limit, *ADC_LIMIT0.ch_lo_limit*, to the selected 10-bit low trip point. When enabled, an ADC measurement lower than this field on the channel selected (*ADC_LIMIT0.ch_sel*) generates an ADC interrupt.
5. Enable the high limit, the low limit, or both interrupt signals by writing a 1 to *ADC_LIMIT0.ch_high_limit_en*, *ADC_LIMIT0.ch_low_limit_en*, or both. Note: Each limit register is independently enabled for high- and low-limit interrupts.
6. Clear the ADC interrupt high and low interrupt flags by writing 1 to *ADC_INTR.hi_limit_if* and *ADC_LIMIT0.lo_limit_if*.
7. Enable the high, low, or both interrupts for the ADC by setting *ADC_INTR.hi_limit_if* to 1, *ADC_INTR.lo_limit_ie* to 1, or both.
8. If an ADC conversion occurs that is above or below the enabled limits, an ADC_IRQ is generated with the *ADC_LIMIT0.adc_high_limit_if*, *ADC_LIMIT0.adc_low_limit_if*, or both set to 1. The *ADC_CTRL.ch_sel* value indicates the channel that caused the interrupt, and the value of the ADC conversion that is out of bounds is in the *ADC_DATA.data* field.

9.7.6 Power-Down Sequence

Complete the following steps to power-down the ADC:

1. Set *ADC_CTRL.pwr* to 0, disabling the ADC converter power.
2. *ADC_CTRL.refbuf_pwr* to 0, disabling the internal reference buffer power.
3. Set *ADC_CTRL.clk_en* to 0, disabling the ADC internal clock.

9.8 Comparator Operation

Each comparator is individually enabled using **Error! Reference source not found.**.aincompnpd. When the inputs to any comparator cross their bias potential, the corresponding flag bit in the **Error! Reference source not found.** register will be set. Should the user desire, an interrupt can be generated by setting the corresponding bit in the ***Error! Reference source not found.*** register. The interrupts must be globally enabled by setting the *GCR_PM.compwken* bit.

9.9 Registers

See *Table 2-3: APB Peripheral Base* Address Map for the base peripheral address of these registers. All fields are reset on peripheral, system, or power-on reset events unless otherwise specified.

Table 9-5. ADC Registers Summary

Offset	Name	Description
[0x0000]	*ADC_CTRL*	ADC Control Register
[0x0004]	*ADC_STATUS*	ADC Status Register
[0x0008]	*ADC_DATA*	ADC Output Data Register
[0x000C]	*ADC_INTR*	ADC Interrupt Control Register
[0x0010]	*ADC_LIMIT0*	ADC Limit 0 Register
[0x0014]	*ADC_LIMIT1*	ADC Limit 1 Register
[0x0018]	*ADC_LIMIT2*	ADC Limit 2 Register
[0x001C]	*ADC_LIMIT3*	ADC Limit 3 Register

9.9.1 Register Details

Table 9-6: ADC Control Register

ADC Control			**ADC_CTRL**	**[0x0000]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:21	-	RO	0x050	**Reserved for Future Use** Do not modify this field.
20	data_align	R/W	0	**ADC Data Alignment** Selects the alignment of the 16-bit data conversion stored in the DATA register. 0: Data is LSB justified in 16-bit DATA register. DATA[15:10] = 0. 1: Data is MSB justified in 16-bit DATA register. DATA[5:0] = 0.
19	-	RO	0	**Reserved for Future Use** Do not modify this field.
18:17	adc_divsel	R/W	0	**External Input Scale** Scales the external inouts AIN0-AIN7. All eight of external inputs are scaled by the same value 0x0: No scaling. 0x1: Divide by 2 0x2: Divide by 3 0x3: Divide by 4

<table>
<tr><th colspan="4">ADC Control</th><th>ADC_CTRL</th><th>[0x0000]</th></tr>
<tr><th>Bits</th><th>Field</th><th>Access</th><th>Reset</th><th colspan="2">Description</th></tr>
<tr><td>16:12</td><td>ch_sel</td><td>R/W</td><td>0</td><td colspan="2">ADC Channel Select
Selects the active channel for the next ADC conversion.
<table>
<tr><th>ch_sel</th><th>ADC Input Channel</th><th>Input</th></tr>
<tr><td>0x00</td><td>0</td><td>AIN0</td></tr>
<tr><td>0x01</td><td>1</td><td>AIN1</td></tr>
<tr><td>0x02</td><td>2</td><td>AIN2</td></tr>
<tr><td>0x03</td><td>3</td><td>AIN3</td></tr>
<tr><td>0x04</td><td>4</td><td>AIN4</td></tr>
<tr><td>0x05</td><td>5</td><td>AIN5</td></tr>
<tr><td>0x06</td><td>6</td><td>AIN6</td></tr>
<tr><td>0x07</td><td>7</td><td>AIN7</td></tr>
<tr><td>0x08</td><td>8</td><td>VCOREA</td></tr>
<tr><td>0x09</td><td>9</td><td>VCOREB</td></tr>
<tr><td>0x0A</td><td>10</td><td>VRXOUT</td></tr>
<tr><td>0x0B</td><td>11</td><td>VTXOUT</td></tr>
<tr><td>0x0C</td><td>12</td><td>VDDA</td></tr>
<tr><td>0x0D</td><td>13</td><td>VDDB / 4</td></tr>
<tr><td>0x0E</td><td>14</td><td>VDDIO / 4</td></tr>
<tr><td>0x0F</td><td>15</td><td>VDDIOH / 4</td></tr>
<tr><td>0x10</td><td>16</td><td>VREGI / 4</td></tr>
<tr><td>0x11 – 0x1F</td><td>Reserved for future use</td><td>Reserved for future use</td></tr>
</table></td></tr>
<tr><td>11</td><td>clk_en</td><td>R/W</td><td>0</td><td colspan="2">ADC Clock Enable
0: Disabled
1: Enabled</td></tr>
<tr><td>10</td><td>-</td><td>RO</td><td>0</td><td colspan="2">Reserved for Future Use
Do not modify this field.</td></tr>
<tr><td>9</td><td>scale</td><td>R/W</td><td>0</td><td colspan="2">ADC Input Scale
Scales ADC input by 50 percent.
0: ADC input is not scaled
1: ADC input is scaled by ½.
Note: See Reference Scaling and Input Scaling for valid settings for each ADC input.</td></tr>
<tr><td>8</td><td>ref_scale</td><td>R/W</td><td>0</td><td colspan="2">Reference Scale
Scales the internal bandgap reference by 50 percent.
0: Internal bandgap reference is not scaled.
1: Internal bandgap reference is scaled by ½.
Note: See Reference Scaling and Input Scaling for valid settings for each ADC input</td></tr>
<tr><td>7:5</td><td>-</td><td>RO</td><td>0</td><td colspan="2">Reserved for Future Use
Do not modify this field.</td></tr>
<tr><td>4</td><td>ref_sel</td><td>R/W</td><td>0</td><td colspan="2">ADC Reference Select
0: Internal bandgap reference is used for the ADC reference
1: V_{DDA} ÷ 2 is used for the ADC reference</td></tr>
<tr><td>3</td><td>refbuf_pwr</td><td>R/W</td><td>0</td><td colspan="2">Reference Buffer Power Enable
0: Disabled
1: Enabled</td></tr>
<tr><td>2</td><td>-</td><td>RO</td><td>0</td><td colspan="2">Reserved for Future Use
Do not modify this field.</td></tr>
</table>

ADC Control			ADC_CTRL		[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
1	pwr	R/W	0	**ADC Power Enable** 0: Disabled 1: Enabled	
0	start	R/W	0	**Start ADC Conversion** Write this bit to 1 to start an ADC conversion. When the conversion is complete, the hardware automatically sets this bit to 0 indicating the conversion is complete. 0: ADC inactive or data conversion complete. 1: Start ADC conversion and remains set until complete.	

Table 9-7: ADC Status Register

ADC Status			ADC_STATUS		[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:4	-	RO	0	**Reserved for Future Use** Do not modify this field.	
3	overflow	RO	0	**ADC Overflow Flag** 0: No overflow on last conversion 1: Overflow on last conversion	
2	afe_pwr_up_active	RO	0	**ADC Power-Up State** This field is set to 1 when the ADC charge pump is powering up. 0: AFE is not in power-up delay. 1: AFE is currently in the power-up delay state.	
1	-	RO	0	**Reserved for Future Use** Do not modify this field.	
0	active	RO	0	**ADC Conversion in Progress** 0: ADC is idle 1: ADC conversion is in progress	

Table 9-8: ADC Data Register

ADC Data			ADC_DATA		[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**	
15:0	data	RO	0	**ADC Data** This field holds the ADC conversion output data. See the *Data Conversion Output Alignment* for details.	

Table 9-9: ADC Interrupt Control Register

ADC Interrupt Control			ADC_INTR		[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:23	-	RO	0	**Reserved for Future Use** Do not modify this field.	
22	pending	RO	0	**ADC Interrupt Pending** 0: No ADC interrupt pending. 1: At least one ADC interrupt is pending, and the corresponding interrupt enable bit is set.	
21	-	RO	0	**Reserved for Future Use** Do not modify this field.	
20	overflow_if	R/W1C	0	**ADC Overflow Interrupt Flag** 1: The last conversion resulted in an overflow	

ADC Interrupt Control			ADC_INTR	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
19	lo_limit_if	R/W1C	0	**ADC Low Limit Interrupt Flag** 1: The last conversion resulted in a low-limit condition for one of the limit registers.
18	hi_limit_if	R/W1C	0	**ADC High Limit Interrupt Flag** 1: The last conversion resulted in a high-limit condition for one of the limit registers.
17	ref_ready_if	R/W1C	0	**ADC Reference Ready Interrupt Flag** 0: Not Ready 1: Ready.
16	done_if	R/W1C	0	**ADC Conversion Complete Interrupt Flag** Set by the ADC hardware when an ADC conversion is complete. 1: ADC conversion complete
15:5	-	RO	0	**Reserved for Future Use** Do not modify this field.
4	overflow_ie	R/W	0	**ADC Overflow Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR.overflow_if*.
3	lo_limit_ie	R/W	0	**ADC Low Limit Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets the *ADC_INTR.lo_limit_if*.
2	hi_limit_ie	R/W	0	**ADC High Limit Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR.lo_limit_if*.
1	ref_ready_ie	R/W	0	**ADC Reference Ready Interrupt Enable** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR.ref_ready_if*.
0	done_ie	R/W	0	**ADC Conversion Complete** 0: Disabled. 1: Enables interrupt assertion when hardware sets *ADC_INTR.done_if*.

Table 9-10: ADC Limit 0 to 3 Registers

ADC Limit 0			ADC_LIMIT0	[0x0010]
ADC Limit 1			**ADC_LIMIT1**	**[0x0014]**
ADC Limit 2			**ADC_LIMIT2**	**[0x0018]**
ADC Limit 3			**ADC_LIMIT3**	**[0x001C]**
Bits	**Field**	**Access**	**Reset**	**Description**
31:30	-	RO	0	**Reserved for Future Use** Do not modify this field.
29	ch_hi_limit_en	R/W	0	**High Limit Monitoring Enable** If set, then an ADC conversion that results in a value greater than the *ch_high_limit* field generates an ADC interrupt if the ADC high-limit interrupt is enabled. (*ADC_INTR.hi_limit_ie* = 1). 1: The high-limit comparison for the ch_sel channel is active. 0: The high-limit comparison is not enabled.
28	ch_lo_limit_en	R/W	0	**Low Limit Monitoring Enable** If set, then an ADC conversion that results in a value less than the ch_high_limit field generates an ADC interrupt if the ADC low-limit interrupt is enabled (*ADC_INTR.lo_limit_ie* = 1). 1: The low-limit comparison for the ch_sel channel is active. 0: The low-limit comparison is not enabled.

ADC Limit 0				ADC_LIMIT0	[0x0010]
ADC Limit 1				ADC_LIMIT1	[0x0014]
ADC Limit 2				ADC_LIMIT2	[0x0018]
ADC Limit 3				ADC_LIMIT3	[0x001C]

Bits	Field	Access	Reset	Description
27:24	ch_sel	R/W	0	**ADC Channel for Limit Monitoring** Sets the ADC input channel for high- and low-limit thresholds. See ADC_CTRL.*ch_sel* for valid values for this field.
23:22	-	RO	0	**Reserved for Future Use** Do not modify this field.
21:12	ch_hi_limit	R/W	0x3FF	**High Limit Threshold** Sets the threshold for high-limit comparisons. This field is a 10-bit value compared against any ADC conversion on the channel set in the *ch_sel* field. ADC conversions greater than this field are over threshold and can result in interrupt assertion if the *ch_hi_limit_en* field is set. Valid values for this field are 0x000 to 0x3FF.
11:10	-	RO	0	**Reserved for Future Use** Do not modify this field.
9:0	ch_lo_limit	R/W	0	**Low Limit Threshold** Sets the threshold for low-limit comparisons. This field is a 10-bit value compared against any ADC conversion on the channel set in the *ch_sel* field. ADC conversions less than this field are under threshold and can result in interrupt assertion if the *ch_lo_limit_en* field is set. Valid values for this field are 0x000 to 0x3FF.

10. UART (UART/LPUART)

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

The LPUART is a special version of the peripheral that can receive characters at 9600 baud while in the low power modes shown in *Table 10-1. MAX78000 UART/LPUART Instances*. Valid characters are loaded into the RX FIFO and can generate a wakeup to ACTIVE mode if enabled.

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

The peripheral provides the following features:

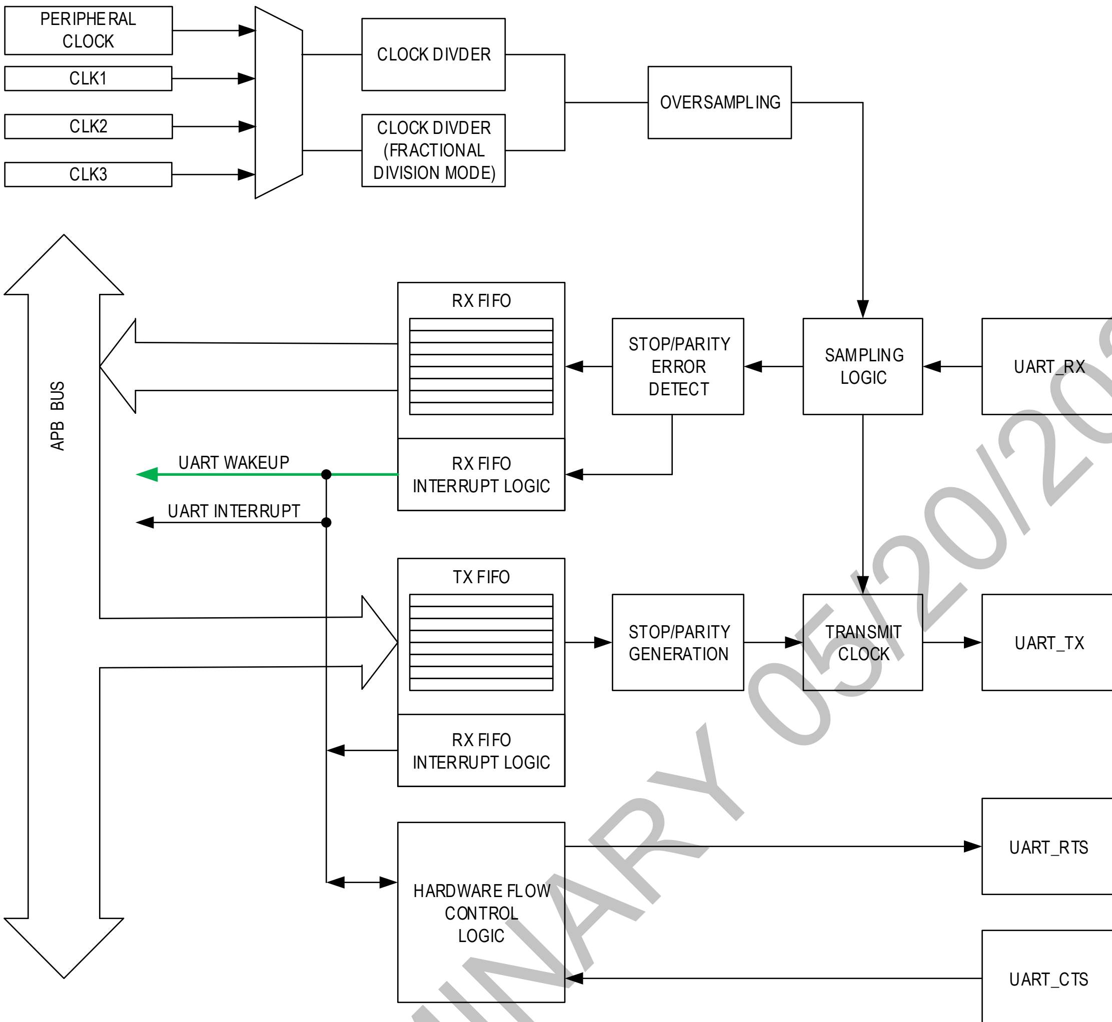
- Flexible baud rate generation up to 4 Mbps with ±2% accuracy (update for highest clock speed.
- Programmable character size of 5-bits to 8-bits
- Stop bit settings of 1, 1.5, or 2-bits
- Parity settings of even, odd, mark (always 1), space (always 0), and no parity
- Automatic parity error detection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

The LPUART instance provides these additional features:

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

Figure 10-1. UART Block Diagram shows a high level description of the UART peripheral.

Figure 10-1. UART Block Diagram



10.1 Instances

Instances of the peripheral are shown in *Table 10-1. MAX78000 UART/LPUART Instances*

The standard UARTs and the Low Power UARTs are functionally very similar, so for common functionality they are referred to as UART. The Low Power UART instances, UART3 and UART4, support fractional division mode (FDM) and are identified by the phrase "instances supporting the FDM function."

Table 10-1. MAX78000 UART/LPUART Instances

Instance	Register Access Name	FDM Support	Power Modes	Peripheral Clock	Clk2	Clk3	Hardware Flow Control	Tx/Rx Fifo Depth
UART0	UART0	NO	ACTIVE SLEEP	PCLK	INRO	ERFO	YES	8/8
UART1	UART1						NO	
UART2	UART2						NO	
LPUART0	UART3	YES	Active Sleep Low Power	PCLK	ERTCO	INRO	NO	8/8
LPUART1	UART4		UPM DEEPSLEEP BACKUP	N/A	ERTCO	INRO	NO	8/8

10.2 DMA

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

The DMA channels are configured using the DMA Configuration Register, *UARTn_DMA*. Enable the RX FIFO DMA channel by setting *UARTn_DMA.rxdma_en* to 1 and enable the TX FIFO DMA channel by setting *UARTn_DMA.txdma_en* to 1. DMA transfers are automatically triggered based on the number of bytes in the RX or TX FIFO

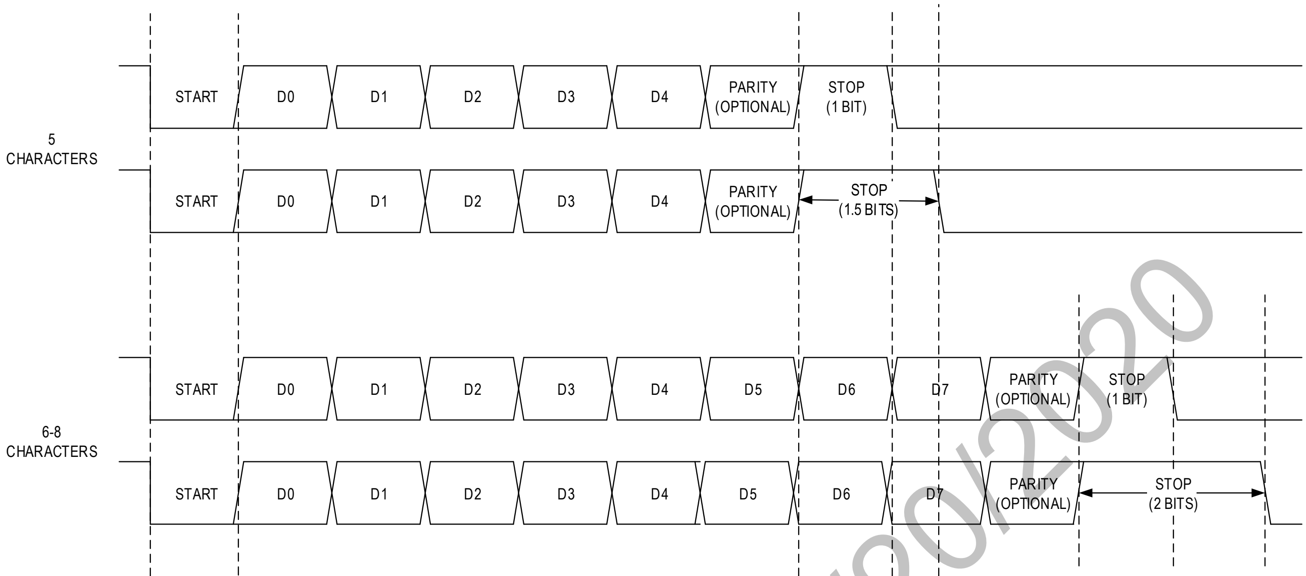
The followings describe the behavior of the RX and TX DMA requests when the DMA function is enabled.

- RX DMA request is asserted when the number of valid bytes in the RX FIFO is greater than or equal to the RX FIFO threshold.
- TX DMA request is asserted when the number of valid bytes in the TX FIFO is less than the TX FIFO threshold.

10.3 UART Frame

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

Figure 10-2. UART Frame Structure



10.4 FIFOs

Separate read (RXFIFO) and write (TX FIFO) FIFOs are provided. They are both accessed through the same UARTn_DATA.*data* field. The current level of the TX FIFO is read from *UARTn_STATUS.tx_num*, and the current level of the RX FIFO is read from *UARTn_STATUS.rx_num*.

10.4.1 TX FIFO Operation

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

Writes to the TX FIFO will be ignored while *UARTn_STATUS.tx_num* = C_TX_FIFO_DEPTH shown in *Table 10-1. MAX78000 UART/LPUART Instances*

10.4.2 RX FIFO Operation

Reads of *UARTn_DATA.data* return the character values in the RX FIFO and decrement *UARTn_STATUS.rx_num*. Data for character sizes less than 7 bits are right justified. An overrun event will occur if a valid frame, including parity, is detected while *UARTn_STATUS.rx_num* = C_RX_FIFO_DEPTH shown in *Table 10-1. MAX78000 UART/LPUART Instances*. In this case the frame will be discarded.

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

10.4.3 Flushing

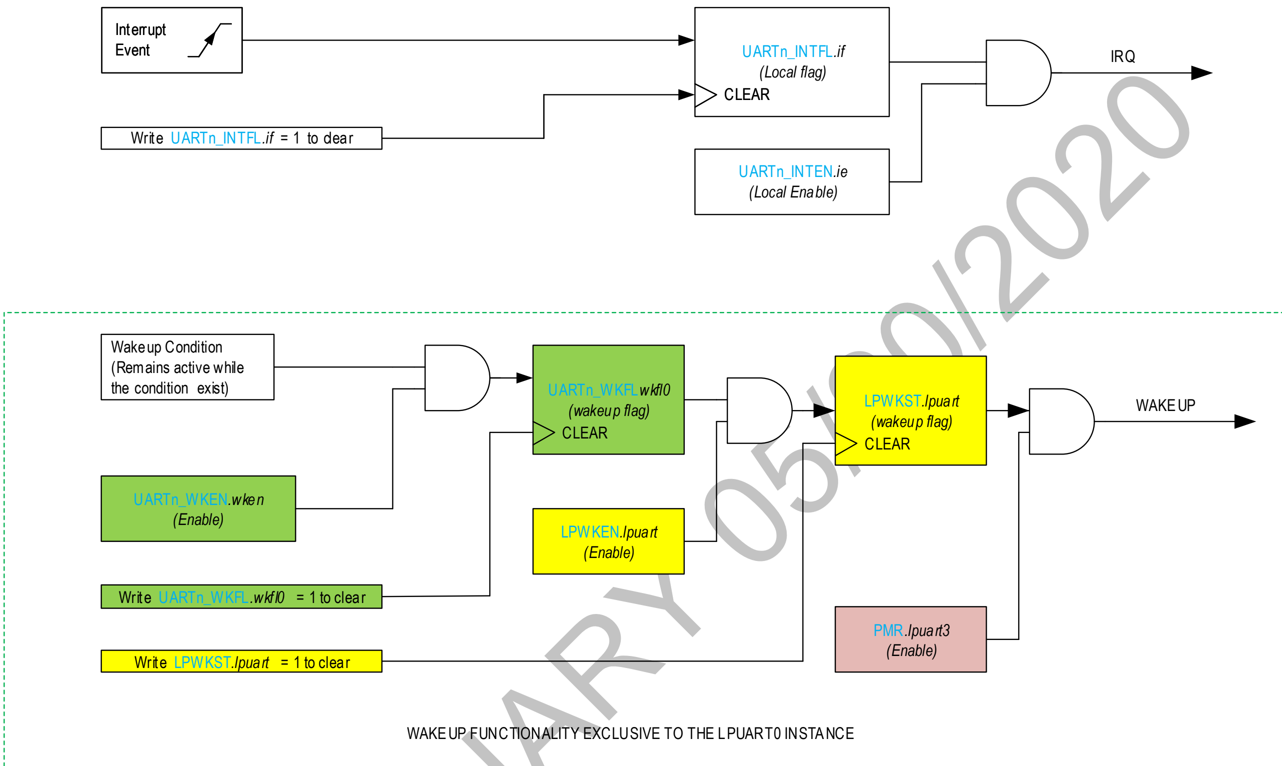
The FIFOs are flushed on the following conditions:

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

10.5 Interrupt Events

The peripheral generates interrupts for the events shown in *Figure 10-3. MAX78000 UART Interrupt and Wakeup Functional Diagram*. Unless noted otherwise, each instance has its own independent set of interrupts, and higher-level flag and enable fields as shown in *Table 10-2. MAX78000 Interrupt Events*.

Figure 10-3. MAX78000 UART Interrupt and Wakeup Functional Diagram



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

Table 10-2. MAX78000 Interrupt Events

EVENT	LOCAL INTERRUPT FLAG	LOCAL INTERRUPT ENABLE
Frame Error	*UARTn_INTFL.rx_ferr_if*	*UARTn_INTEN.rx_ferr_ie*
Parity Error	*UARTn_INTFL.rx_par_if*	*UARTn_INTEN.rx_par_ie*
CTS Signal Change	*UARTn_INTFL.cts_if*	*UARTn_INTEN.cts_ie*
RX FIFO Overrun	*UARTn_INTFL.rx_ov_if*	*UARTn_INTEN.rx_ov_ie*
RX FIFO Threshold	*UARTn_INTFL.rx_thd_if*	*UARTn_INTEN.rx_thd_ie*
TX FIFO Half-Empty	*UARTn_INTFL.tx_he_if*	*UARTn_INTEN.tx_he_ie*

10.5.1 Frame Error

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

The frame error criteria are different depending on whether or not FDM is present in that instance and enabled or not as shown in *Table 10-3. Frame Error Detection (FDM not present, or FDM = 0 and DPFEE = 0)* and *Table 10-4. Frame Error Detection (FDM = 1 and DPFEE = 1)*.

Table 10-3. Frame Error Detection (FDM not present, or FDM = 0 and DPFEE = 0)

PAR_EN	PAR_MD	PAR_EO	START	DATA	*PARITY*	*STOP*
0	N/A	N/A	3/3 = 0	2/3	Not Present	3/3=1
1	0	0			3/3 = 1 if even number "1" 3/3 = 0 if odd number "0"	
	0	1			3/3 = 1 if odd number "1" 3/3 = 0 if even number "0"	
	1	0			3/3 = 1 if even number "0" 3/3 = 0 if odd number "1"	
	1	1			3/3 = 1 if odd number "0" 3/3 = 0 if even number "1"	

Table 10-4. Frame Error Detection (FDM = 1 and DPFEE = 1)

PAR_EN	PAR_MD	PAR_EO	START	DATA	*PARITY*	*STOP*
0	N/A	N/A	3/3 = 0	3/3 = 0 or 3/3 = 1	Not Present	3/3=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"	

10.5.2 Parity Error

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

10.5.3 CTS Signal Change

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

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

10.5.4 Overrun

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

10.5.5 RX FIFO Threshold

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

10.5.6 TX FIFO One Byte Remaining

This event occurs when the TX FIFO transitions from *UARTn_STATUS.tx_num* = 2 to *UARTn_STATUS.tx_num* = 1.

10.5.7 TX FIFO Half-Empty

The TX FIFO Half-Empty event occurs when *UARTn_STATUS.tx_num* transitions from:

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

10.6 Wakeup Events

Instances which support fractional division mode can receive characters while in the low-power modes listed in *Table 10-1. MAX78000 UART/LPUART Instances*. If enabled, any of the wakeup sources in *Table 10-5. MAX78000 Wakeup Events* will cause the device to exit the current low-power mode and return to ACTIVE mode.

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

Table 10-5. MAX78000 Wakeup Events

CONDITION	LOCAL WAKEUP FLAG *UARTn_WKFL*	LOCAL WAKEUP ENABLE *UARTn_WKEN*	LOW POWER PERIPHERAL WAKEUP FLAG	LOW POWER PERIPHERAL WAKEUP ENABLE	POWER MANAGEMENT WAKEUP ENABLE
RX FIFO Threshold	*rx_thd_wf*	*rx_thd_we*	LPWKFL.*lpuart_wf*	LPWKFL.*lpuart_we*	PMR.lpuart3wken
RX FIFO Full	*rx_fu_wf*	*rx_fl_we*			
RX FIFO Not Empty	*rx_ne_wf*	*rx_ne_we*			

10.6.1 RX FIFO Threshold

This condition persists while *UARTn_STATUS.rx_num* ≥ *UARTn_CTRL.rx_thd*.

10.6.2 RX FIFO Full

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

10.6.3 RX Not Empty

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

10.7 Resets

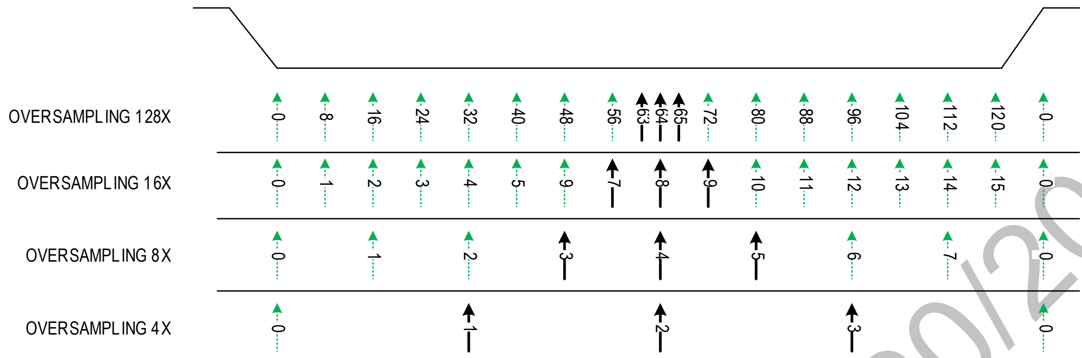
The following stimuli can reset the peripheral.

- The device will remain in reset while *UARTn_CTRL.bcken* = 0.
- The device will remain in reset while *UARTn_CTRL.bckrdy* = 0 after setting *UARTn_CTRL.bcken* to 1.
- Any write to *UARTn_CKDIV.clkdiv* while *UARTn_CTRL.bcken* = 1.
- Any write to *UARTn_OSR.osr* while *UARTn_CTRL.bcken* = 1.

10.8 RX 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 will be evaluated based on three samples at the midpoint of each bit time as shown in *Figure 10-4. Oversampling Example*.

Figure 10-4. Oversampling Example



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

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

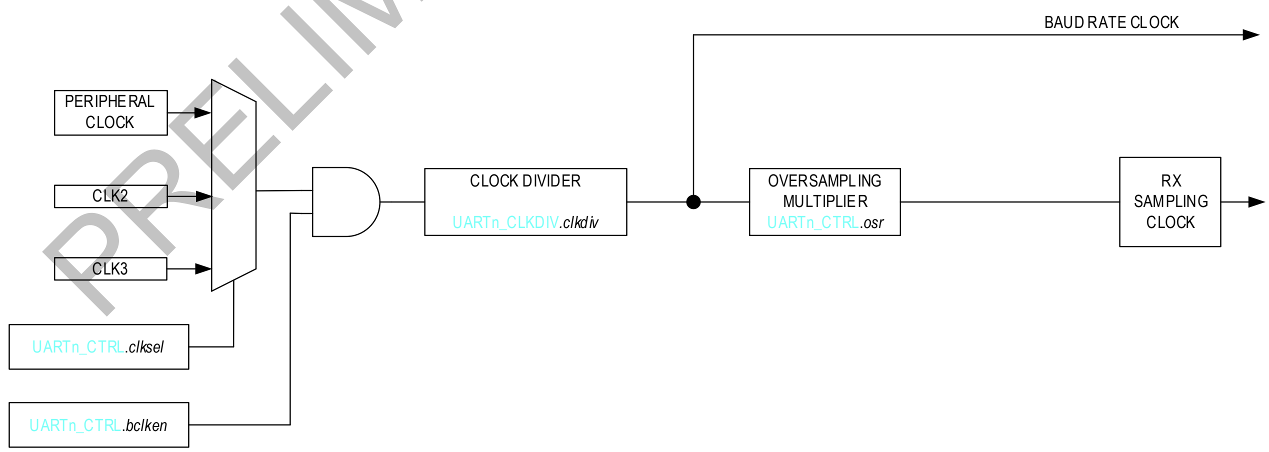
10.9 Baud Rate Generation

The baud rate is determined by the selected UART clock source and value of the clock divider circuit. Multiple clock sources are available for each instance and can be configured independently for each UART instance as shown in *Table 10-1. MAX78000 UART/LPUART Instances*.

10.9.1 Clock Sources (FDM not supported)

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

Figure 10-5. UART Timing Generation (FDM not supported)

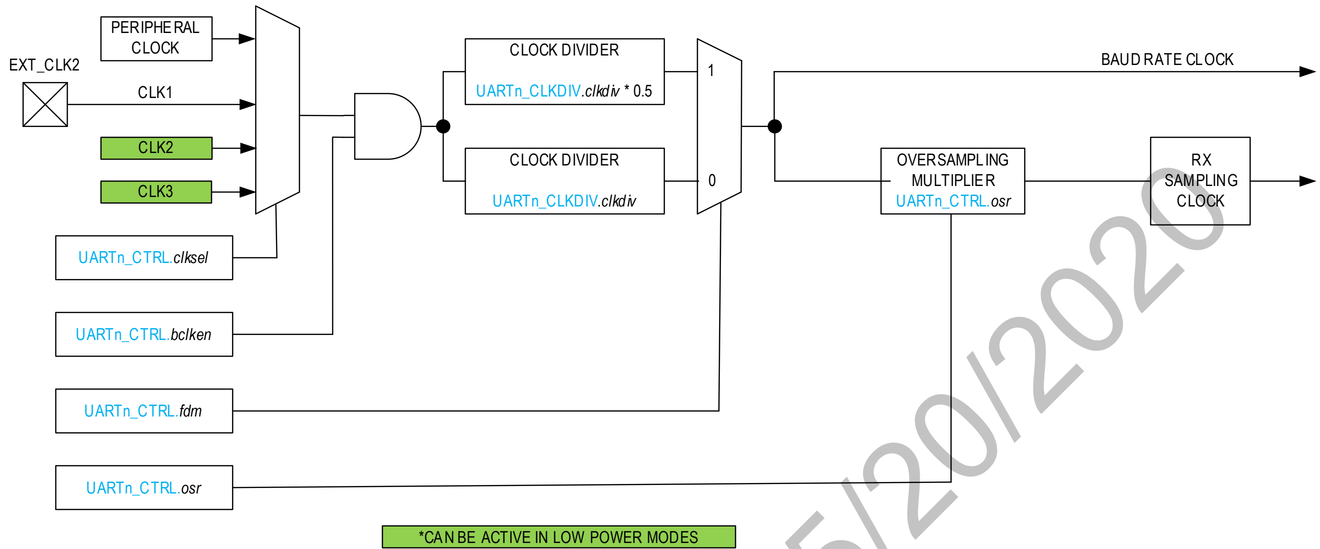


10.9.2 Clock Sources (FDM supported)

Instances that support FDM can be configured to operate the LPUART receiver at 9600 baud using the 32kHz RTC as shown in *Table 10-1. MAX78000 UART/LPUART Instances*. This clock source can be configured to remain active in DEEPSLEEP and

BACKUP modes, allowing the device to receive data and serve as a wakeup source. while power consumption is at a minimum.

Figure 10-6. LPUART Timing Generation (FDM supported)



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

10.10 Baud Rate Calculation

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

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

UARTn_CTRL.fdm = 0: *UARTn_CKDIV.clkdiv* = INT [(UART clock)/Baud Rate]

UARTn_CTRL.fdm = 1: *UARTn_CKDIV.clkdiv* = INT [(UART clock)/Baud Rate × 2]

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

- When FDM=0, *UARTn_CKDIV.clkdiv* =50,000,000/115,200 = 434
- When FDM=1, *UARTn_CKDIV.clkdiv* =50,000,000/115,200 × 2 = 434.03 × 2 = 868

10.11 Low-Power 9600 Baud Receiver Operation

Peripheral instances which support FDM have the option to configure the receiver for 9600 baud and keep it enabled in the low- power modes listed in *Table 10-1. MAX78000 UART/LPUART Instances*. Receipt of a valid frame will load the RX FIFO and increment *UARTn_STATUS.rx_num*. If enabled, any of the conditions in *Table 10-5. MAX78000 Wakeup Events* will cause the device to exit the current low-power mode and return to ACTIVE mode.

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

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

UARTn_CTRL.fdm = 1:

UARTn_CKDIV.clkdiv = INT [(UART clock)/Baud Rate × 2]

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

Clock Source	BAUD	Ratio	***UARTn_CKDIV.clkdiv*** **(Value)**	Error	***UARTn_OSR.osr***
32,768Hz	9,600 bit/s	3.413	0x00007 (3.5)	-2.5%	N/A (1x)
	7,200 bit/s	4.551	0x00009 (4)	+1.1%	N/A (1x)
	4,800 bit/s	6.827	0x00014 (7)	-2.5%	N/A (1x)
	2,400 bit/s	13.653	0x00027	+1.1%	0x0: 8x 0x1: 12x
	1,800 bit/s	18.204	0x00036	+1.1%	0x0: 8x 0x1: 12x 0x2: 16x
	1,200 bit/s	27.307	0x00054	+1.1%	0x0: 8x 0x1: 12x 0x2: 16x 0x3: 20x 0x4: 24x

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

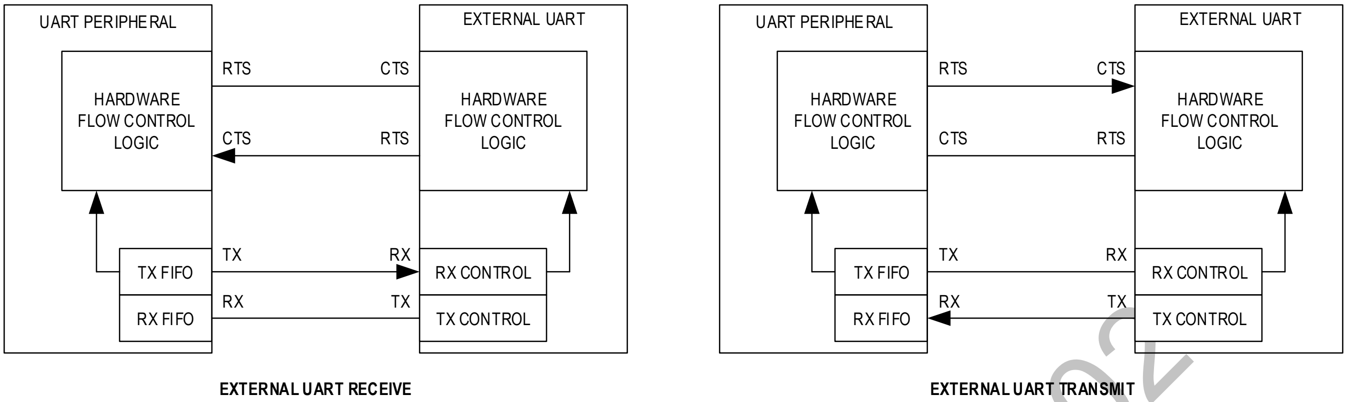
Use the following procedure for the specific use case to receive characters at 9600 baud while in low-power modes:

1. Clear *UARTn_CTRL.bcken* = 0 to disable the baud clock. Hardware will immediately clear *UARTn_CTRL.bckrdy* to 0.
2. Set *PWRSEQ_LPCN.x32ken*= 1 ensure the 32kHz clock source remains active in DEEPSLEEP and BACKUP modes
3. Ensure *UARTn_CTRL.ucagm* = 1.
4. Configure *UARTn_CTRL.clksel* to the 32kHz clock source.
5. Set *UARTn_CTRL.fdm* = 1.
6. Set *UARTn_CKDIV.clkdiv* = 7.
7. Set *UARTn_CTRL.desm* = 1.
8. Choose the desired wakeup conditions from *Table 10-5. MAX78000 Wakeup Events* and set the corresponding wakeup enable fields to 1. Ensure the conditions are not active before setting the wakeup enable fields.
9. Set *UARTn_CTRL.bcken* = 1 to re-enable the baud clock.
10. Poll until hardware sets *UARTn_CTRL.bckrdy* = 1.
11. Enter the desired low power mode.

10.12 Hardware Flow Control

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

Figure 10-7. Hardware Flow Control, Physical Connection



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

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

10.12.1 Automated HFC

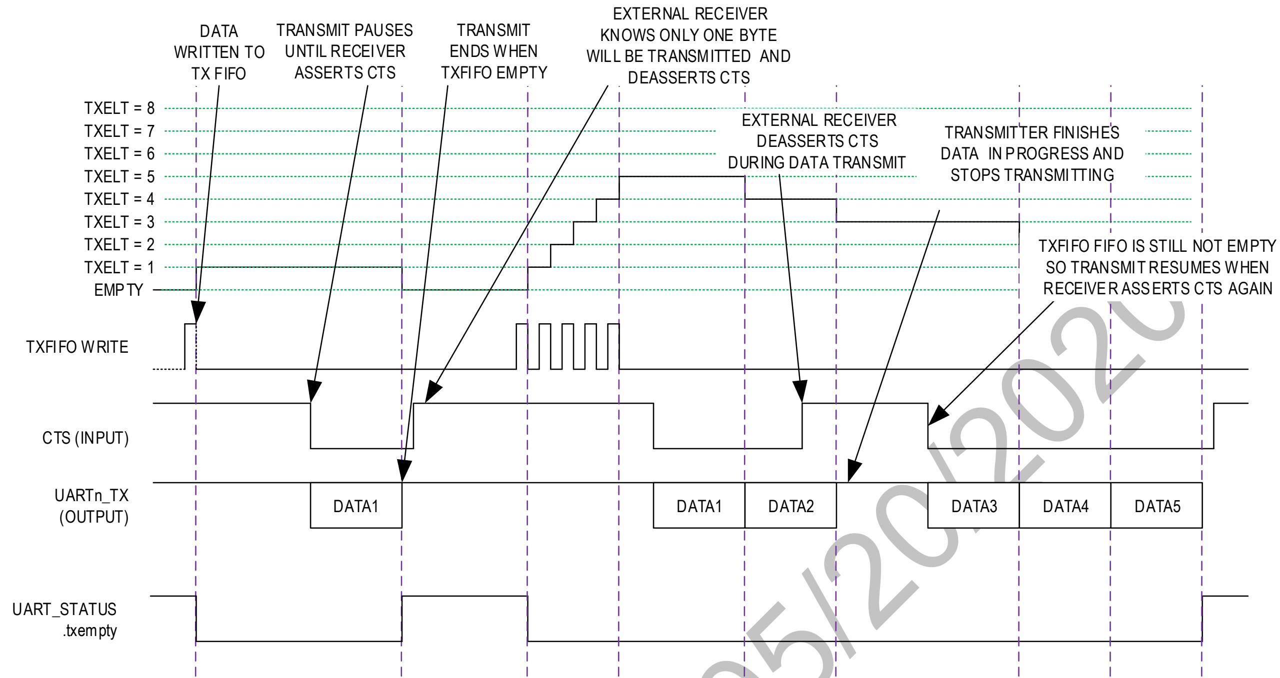
Setting *UARTn_CTRL.flow_ctrl* = 1 enables hardware flow control. The CTS and RTS external signals are directly managed by hardware without CPU intervention. The assertion of the RTS to CTS signal is dependent on the *UARTn_CTRL.rtslegacy* field:

- *UARTn_CTRL.rtslegacy* = 0: Deassert RTS while *UARTn_STATUS.rx_num* = C_RX_FIFO_DEPTH
- *UARTn_CTRL.rtslegacy* = 1: Deassert RTS while *UARTn_STATUS.rx_num* ≥= *UARTn_CTRL.rx_thd*

The transmitter will continue to send data as long as the CTS signal is asserted and there is data in the TX FIFO. If the receiver deasserts the transmitter's CTS, the transmitter will finish transmission of the current character and then wait until CTS is asserted again. *Figure 10-8. Hardware Flow Control Signaling, Transmit to External Receiver* shows the state of the CTS pin during a transmission under hardware flow control.

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

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



10.12.2 Software-Controlled HFC

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

UARTn_CTRL.ctssd is reset to 1 after power-on reset. When UART baud clock starts and runs, CTS sampling process continues if *UARTn_CTRL.ctssd*) is not 1. The sampled value will be set to 1 again when UART baud clock stops running because *UARTn_CTRL.bcken* has been cleared to 0 again.

10.13 Registers

See *Table 2-3: APB Peripheral Base* Address Map for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 10-7. UART Register Summary*. 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 2-1. Field Access Definitions* 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 10-7. UART Register Summary

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

Offset	Register	Name
0x000C	*UARTn_INTFL*	UART Interrupt Flag Register
0x0010	*UARTn_CKDIV*	UART Clock Divisor Register
0x0014	*UARTn_OSR*	UART Oversampling Control Register
0x0018	*UARTn_TXFIFO*	UART Transmit FIFO
0x0020	*UARTn_PNR*	UART Pin Control Register
0x0030	*UARTn_DATA*	UART FIFO Data Register
0x0034	*UARTn_DMA*	UART DMA Control Register
0x002C	*UARTn_WKEN*	UART Wakeup Interrupt Enable Register
0x0030	*UARTn_WKFL*	UART Wakeup Interrupt Flag Register

10.13.1 Register Details

Table 10-8. UART Control Register

UART Control				UARTn_CTRL [0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:23	-	DNM	0	**Reserved.** Do Not Modify.
22	desm	R/W	0	**RX Dual Edge Sampling Mode** 0: Sample RX on clock rising only 1: Sample RX on both rising and falling edges This field is undefined if the instance does not support FDM.
21	fdm	R/W	0	**Fractional Division Mode** 0: Baud rate divisor is an integer 1: Baud rate divisor supports 0.5 division resolution. This field is undefined if the instance does not support FDM.
20	ucagm	R/W	0	**UART Clock Auto Gating Mode** Software must always set this field to 1 for proper operation. 0: No gating 1: UART clock is paused during TX/RX idle states
19	bckrdy	R	0	**Baud Clock Ready** 0: Baud clock not ready 1: Baud clock ready
18	dpfee	R/W	0	**Data/Parity bit frame error detection enable** 0: Disable. Do not detect frame errors on RX between start bit and stop bit. 1: Enable. Detect frame errors when RX changes at the center of bit time. This field is undefined if the instance does not support FDM.
17:16	clksel	R/W	0	**Baud Clock Source** Selects the baud clock source. See *Table 10-1. MAX78000 UART/LPUART Instances* for specific values for each instance. 0: Peripheral Clock 1: External clock 2: CLK2 3: CLK3
15	bcken	R/W	0	**Baud Clock Enable** 0: Disabled 1: Enabled

UART Control				UARTn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
14	rtslegacy	R	0	**Hardware Flow Control RTS Deassert Condition** 0: Deassert RTS when RX FIFO Level = C_RX_FIFO_DEPTH (FIFO full) 1: Deassert RTS while RX FIFO Level >= *UARTn_CTRL.rx_thd*	
13	flow_ctrl	R/W	0	**Hardware Flow Control Enable** 0: Disabled 1: Enabled	
12	stopbits	R/W	0	**Number of Stop Bits** 0: 1 stop bit 1: 1.5 stop bits (for 5 bit mode) or 2 stop bits (for 6/7/8 bit mode)	
11:10	char_size	R/W	0	**Character Length** 0x0: 5 bits 0x1: 6 bits 0x2: 7 bits 0x3: 8 bits	
9	rx_flush	W1	0	**RX FIFO Flush** Write 1 to flush the FIFO. This bit will always read 0. 0: N/A 1: Flush FIFO	
8	tx_flush	W1	0	**TX FIFO Flush** Write 1 to flush the FIFO. This bit will always read 0. 0: N/A 1: Flush FIFO	
7	cts_samples_dis	R/W	0	**CTS Sampling Disable** 0: Enabled 1: Disabled	
6	par_md	R/W	0	**Parity Value Select** 0: Parity calculation is based on "1" bits. (Mark) 1: Parity calculation is based on "0" bits. (Space)	
5	par_eo	R/W	0	**Parity Odd/Even Select** 0: Even parity 1: Odd parity	
4	par_en	R/W	0	**TX 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	R/W	0	**Receive FIFO Threshold** Valid settings are from 0x01 to (C_RX_FIFO_DEPTH – 1). 0x0: Reserved 0x1: 1 0x2: 2 0x3: 3 0x4: 4 0x5: 5 0x6: 6 0x7: 7 0x8: 8 0x9: Reserved 0xA: Reserved 0xB: Reserved 0xC: Reserved 0xD: Reserved 0xE: Reserved 0xF: Reserved	

Table 10-9. UART Status Register

UART Status				UARTn_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:16	-	RO	0	**Reserved**	
15:12	tx_num	R	0	**TX FIFO Level** Number of characters in the TX FIFO. 0x0: 0 0x1: 1 0x2: 2 0x3: 3 0x4: 4 0x5: 5 0x6: 6 0x7: 7 0x8: 8 0x9: Reserved 0xA: Reserved 0xB: Reserved 0xC: Reserved 0xD: Reserved 0xE: Reserved 0xF: Reserved	
11:8	rx_num	R	0	**RX FIFO Level** Number of characters in the RX FIFO. 0x0: 0 0x1: 1 0x2: 2 0x3: 3 0x4: 4 0x5: 5 0x6: 6 0x7: 7 0x8: 8 0x9: Reserved 0xA: Reserved 0xB: Reserved 0xC: Reserved 0xD: Reserved 0xE: Reserved 0xF: Reserved	
7	tx_fu_st	R	0	**Transmit FIFO Full** 0: Not full 1: Full	
6	tx_em_st	R	1	**Transmit FIFO Empty** 0: Not empty 1: Empty	
5	rx_fu_st	R	0	**Receive FIFO Full** 0: Not full 1: Full	
4	rx_em_st	R	1	**Receive FIFO Empty** 0: Not empty 1: Empty	
3:2	-	RO	0	**Reserved**	

UART Status				UARTn_STATUS	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
1	rx_busy	R	0	**Receive Busy** 0: UART is not receiving a character 1: UART is receiving a character	
0	tx_busy	R	0	**Transmit Busy** 0: UART is not transmitting data 1: UART is transmitting data	

Table 10-10. UART Interrupt Enable Register

UART Interrupt Enable				UARTn_INTEN	[0x0008]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:7	-	RO	0	**Reserved**	
6	tx_he_ie	R/W	0	**TX FIFO Half-Empty Event Interrupt Enable** 0: Disabled 1: Enabled	
5	-	RO	0	**Reserved**	
4	rx_thd_ie	R/W	0	**RX FIFO Threshold Event Interrupt Enable** 0: Disabled 1: Enabled	
3	rx_ov_ie	R/W	0	**RX FIFO Overrun Event Interrupt Enable** 0: Disabled 1: Enabled	
2	cts_ie	R/W	0	**CTS Signal Change Event Interrupt Enable** 0: Disabled 1: Enabled	
1	rx_par_ie	R/W	0	**RX Parity Event Interrupt Enable** 0: Disabled 1: Enabled	
0	rx_ferr_ie	R/W	0	**RX Frame Error Event Interrupt Enable** 0: Disabled 1: Enabled	

Table 10-11.UART Interrupt Flag Register

UART Interrupt Flag				UARTn_INTFL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:7	-	RO	0	**Reserved**	
6	tx_he_if	R/W1C	0	**TX FIFO Half-Empty Event Interrupt Flag** 0: Disabled 1: Enabled	
5	-	RO	0	**Reserved**	
4	rx_thd_if	R/W1C	0	**RX FIFO Threshold Event Interrupt Flag** 0: Disabled 1: Enabled	
3	rx_ov_if	R/W1C	0	**RX FIFO Overrun Event Interrupt Flag** 0: Disabled 1: Enabled	
2	cts_if	R/W1C	0	**CTS Signal Change Event Interrupt Flag** 0: Disabled 1: Enabled	

UART Interrupt Flag				UARTn_INTFL	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**	
1	rx_par_if	R/W1C	0	**RX Parity Error Event Interrupt Flag** 0: Disabled 1: Enabled	
0	rx_ferr_if	R/W1C	0	**RX Frame Error Event Interrupt Flag** 0: Disabled 1: Enabled	

Table 10-12. UART Clock Divisor Register

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

Table 10-13. UART Oversampling Control Register

UART Oversampling Control				UARTn_OSR	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:3	-	RO	0	**Reserved**	
2:0	osr	R/W	0	**Over Sampling Rate** In Fractional Division Mode (FDM=1), 0x0: 8x, 0x1: 12x, 0x2: 16x, 0x3: 20x, 0x4: 24x 0x5: 28x, 0x6: 32x, 0x7: 36x Not in Fractional Division Mode (FDM=0) 0x0: 128x 0x1: 64x 0x2: 32x 0x3: 16x 0x4: 8x 0x5: 4x 0x6: Reserved 0x7: Reserved	

Table 10-14. UART TXFIFO Register

UART TXFIFO				UARTn_TXFIFO	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:8	-	RO	0	**Reserved**	

UART TXFIFO				UARTn_TXFIFO	[0x0018]
Bits	**Name**	**Access**	**Reset**	**Description**	
7:0	data	R	0	**Transmit FIFO Data** Read TX FIFO next data. Reading from this field doesn't affect the contents of TX FIFO. Note that the parity bit is available from this field. Reading from this field returns the next character available at the output of the TX FIFO (if one is available, otherwise 00h is returned).	

Table 10-15. UART Pin Control Register

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

Table 10-16. UART Data Register

UART Data				UARTn_DATA	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:9	-	RO	0	**Reserved**	
8	rx_parity	R	0	**Receive FIFO Byte Parity** If parity feature is disabled, this bit always read 0. If a parity error occurred during the reception of the character at the output end of the RX FIFO (that would be returned by reading the DATA field), this bit will read 1, otherwise it will read 0.	
7:0	data	R/W	0	**Transmit/Receive FIFO Value** Writing to this field loads the next character into the TX FIFO (if space is available). Reading from this field returns the next character available at the output of the RX FIFO (if one is available, otherwise 00h is returned). For 5/6/7-bit width characters, the unused bit(s) are ignored when the TX FIFO is loaded, and the unused high bit(s) read 0 on characters read from RX FIFO	

Table 10-17. UART DMA Register

UART DMA				UARTn_DMA	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:10	-	RO	0	**Reserved**	
9	dma_rx_en	0	0	**RX DMA Channel Enable** 0: Disabled 1: Enabled	
8:5	dma_rx_thd	0	0	**RX FIFO Level DMA Threshold** If *UARTn_STATUS.rx_num* < *UARTn_DMA.dma_rx_thd*, then the RX FIFO DMA interface will send a signal to the system DMA to notify that RX FIFO has characters to transfer to memory	

UART DMA				UARTn_DMA	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**	
4	dma_tx_en	R/W	0	**TX DMA Channel Enable** 0: Disabled 1: Enabled	
3:0	dma_tx_thd	R/W	0	**TX FIFO Level DMA Threshold** If *UARTn_STATUS.tx_num* < *UARTn_DMA.dma_tx_thd*, then the TX FIFO DMA interface will send a signal to system DMA to notify that TX FIFO is ready to receive data from memory.	

Table 10-18. UART Wakeup Enable

UART Wakeup Enable				UARTn_WKEN	[0x0034]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:3	-	RO	0	**Reserved**	
2	rx_thd_we	R/W	0	**RX FIFO Threshold Wake-up Event Enable** 0: Disabled 1: Enabled	
1	rx_fu_we	R/W	0	**RX FIFO Full Wake-Up Event Enable** 0: Disabled 1: Enabled	
0	rx_ne_we	R/W	0	**RX FIFO Not Empty Wake-Up Event Enable** 0: Disabled 1: Enabled	

Table 10-19. UART Wakeup Flag Register

UART Wakeup Flag				UARTn_WKFL	[0x0038]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:3	-	RO	0	**Reserved**	
2	rx_thd_wf	R/W	0	**RX FIFO Threshold Wake-up Event** 0: Disabled 1: Enabled	
1	rx_fu_wf	R/W	0	**RX FIFO Full Wake-Up Event** 0: Disabled 1: Enabled	
0	rx_ne_wf	R/W	0	**RX FIFO Not Empty Wake-Up Event** 0: Disabled 1: Enabled	

11. Serial Peripheral Interface (SPI)

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

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

Features include:

Dedicated Bit Rate Generator for precision serial clock generation in Master Mode
- Up to $\frac{f_{PCLK}}{2}$ for instances on the APB bus
- Up to $\frac{f_{HCLK}}{2}$ for instances on the AHB bus
- Programmable SCK duty cycle timing

Full-duplex, synchronous communication of 2 to 16-bit characters
- 1-bit and 9-bit characters are not supported
- 2-bit and 10-bit characters do not support maximum clock speed. *SPIn_CLK_CFG*.scale must be > 0

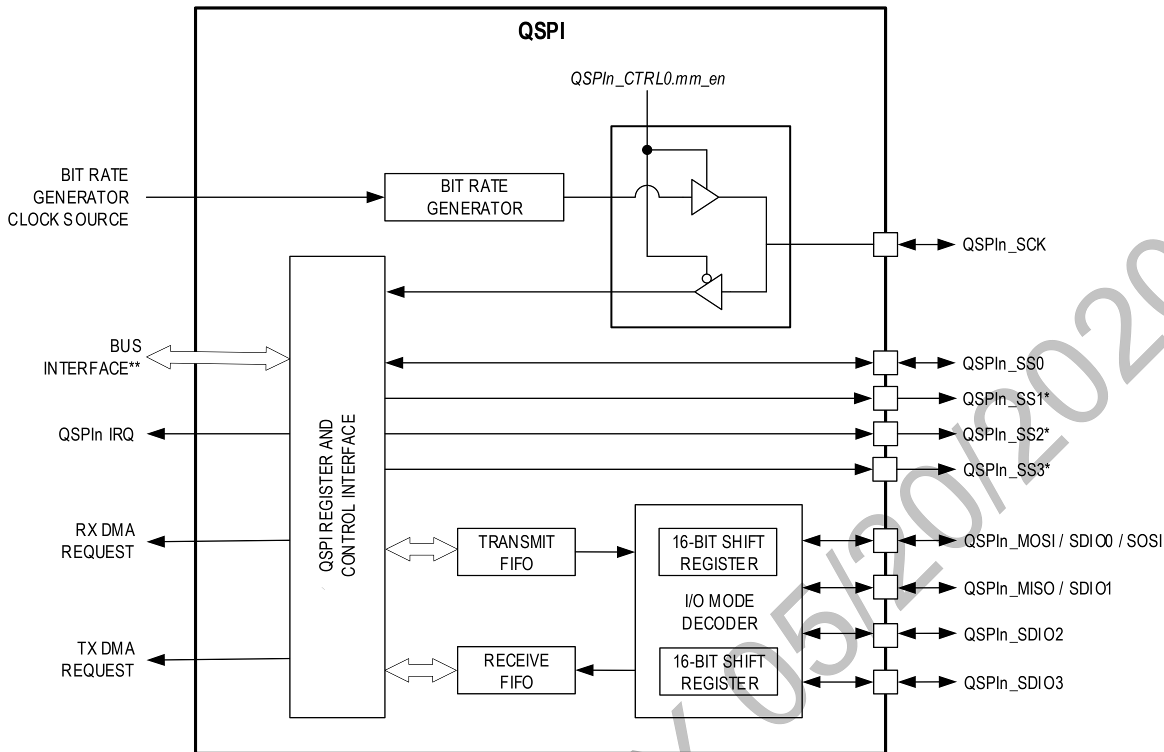
3-wire and 4-wire SPI operation for single-bit communication
Single, Dual and Quad I/O
Byte-wide Transmit and Receive FIFOs with 32-byte depth
- For character sizes greater than 8, each character uses 2 entries per character resulting in 16 entries for the Transmit and Receive FIFO

Transmit and Receive DMA support
SPI modes 0, 1, 2, 3
Configurable slave select lines
- Programmable slave select level

Programmable slave select timing with respect to SCK starting edge and ending edge
Multi-master mode fault detection

Figure 11-1: SPI Block Diagram shows the structure of the peripheral. See *Table 11-1: MAX78000 SPI* Instances for the peripheral-specific peripheral bus assignment and bit rate generator clock source.

Figure 11-1: SPI Block Diagram



* The number of slave select signals can vary for each instance of the peripheral.
** The bus interface (APB or AHB) can vary for each instance of the peripheral.

11.1 Instances

The following instances of the peripheral are provided. For a specific instance replace n in register names with either 0, 1, or 2 depending on the instance of the peripheral.

Table 11-1: MAX78000 SPI Instances

Name	Formats				Bus Assignment	Bit Rate Generator Clock Source Frequency	Slave Select Signals	
	3-Wire	4-Wire	Dual	Quad			40-TQFN	24-WLP
SPI0	Yes	Yes	No	No	APB	f_{PCLK}	1	N/A
SPI1	Yes	Yes	No	No	APB	f_{PCLK}	1	N/A
SPI2	Yes	Yes	No	No	APB	f_{PCLK}	0 (SPI2 Not Available)	N/A

Note: For SPI signal mapping refer to the MAX78000 data sheet.

11.2 SPI Formats

11.2.1 Four-Wire SPI

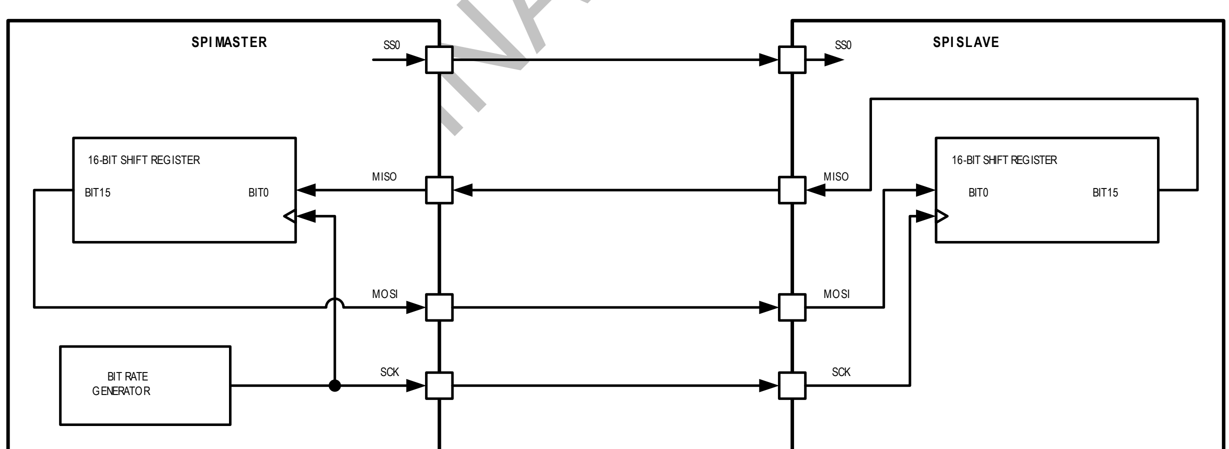
SPI devices operate as either a master or slave device. In four-wire SPI, four signals are required for communication as shown in *Table 11-2*.

Table 11-2: Four-Wire Format Signals

Signal	Description	Direction
SCK	Serial Clock	The master generates the Serial Clock signal, which is an output from the master and an input to the slave.
MOSI	Master Output Slave Input	In master mode, this signal is used as an output for sending data to the slave. In slave mode this is the input data from the master.
MISO	Master Input Slave Output	In master mode, this signal is used as an input for receiving data from the slave. In slave mode, this signal is an output for transmitting data to the master.
SS	Slave Select	In master mode, this signal is an output used to select a slave device prior to communication. Peripherals may have multiple slave select outputs to communicate with one or more external slave devices. In slave mode SPIn_SS0 is a dedicated input which indicates an external master is going to start communication. Other slave select signals into the peripheral are ignored in slave mode.

In a typical SPI network, the master device selects the slave device using the slave select output. The master starts the communication by selecting the slave device by asserting the slave select output. The master then starts the SPI clock via the SCK output pin. When a slave device's slave select pin is deasserted, the device is required to put the SPI pins in tri-state mode.

Figure 11-2. 4-Wire SPI Connection Diagram



11.2.2 Three-Wire SPI

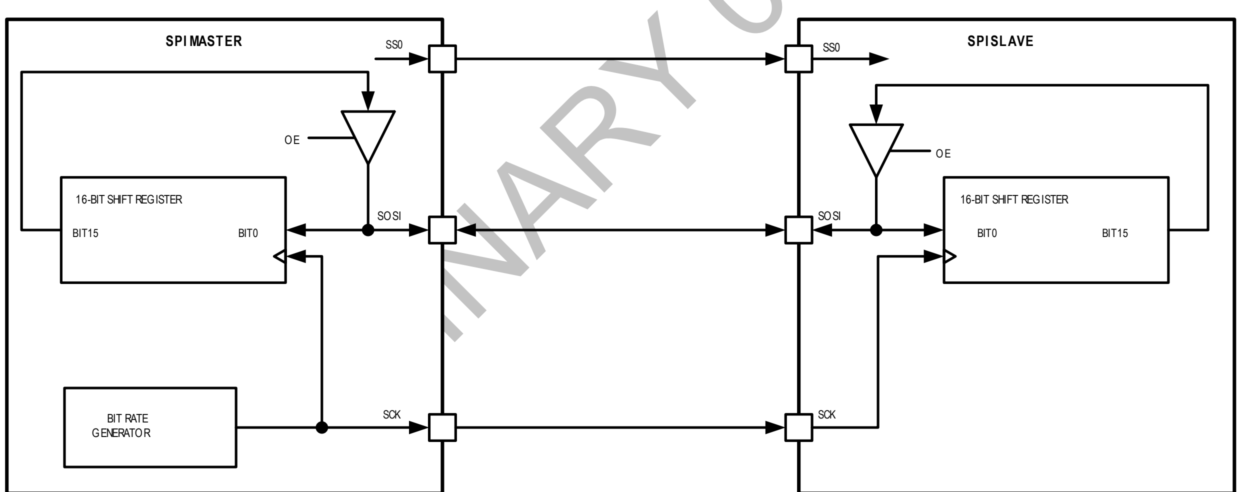
The signals in three-wire SPI operation are shown in *Table 11-3: Three-Wire Format* Signals, The MOSI signal is used as a bi-directional, half-duplex I/O referred to as Slave Input Slave Output (SISO). Three-wire SPI also uses a serial clock signal generated by the master and a slave select pin controlled by the master.

Table 11-3: Three-Wire Format Signals

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

A three-wire SPI network is shown in *Figure 11-3*. The master device selects the slave device using the slave select output. The communication starts with the master asserting the slave select line and then starting the clock (SCK). In three-wire SPI communication, the master and slave must both know the intended direction of the data to prevent bus contention. For a write, the master drives the data out the SISO pin. For a read, the master must release the SISO line and let the slave drive the SISO line. The direction of transmission is controlled using the FIFO. Writing to the FIFO starts the three-wire SPI write and reading from the FIFO starts a three-wire SPI read transaction.

Figure 11-3: Generic 3-Wire SPI Master to Slave Connection



11.3 Pin Configuration

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

11.3.1 SPIn Alternate Function Mapping

Pin selection and configuration is required to use the SPI port. The following information applies to SPI master and slave operation as well as three-wire, 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 MAX78000 data sheet for pin availability for a specific package.

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

11.3.2 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 slave select pin for a transaction, resulting in more than four wires total, however the communication is referred to as four wire for legacy reasons.

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

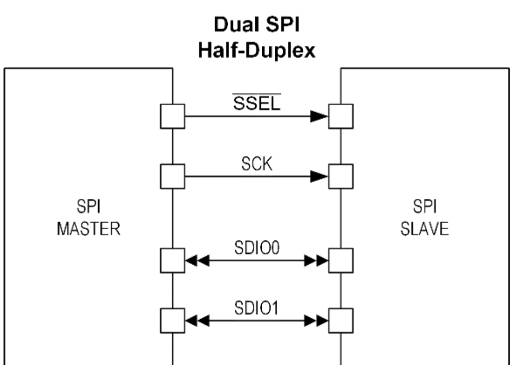
11.3.3 Three-Wire Format Configuration

Three wire SPI uses SCK and MOSI, and one or more slave select pins for an SPI transaction. Three-wire SPI configuration is identical to the four wire configuration except 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 SPIn 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.

11.3.4 Dual Mode Format Configuration

In Dual mode SPI two I/O pins are used to transmit 2-bits of data per SCK clock cycle. The communication is half-duplex and the direction of the data transmission must be known by both the master and slave for a given transaction. Dual mode SPI uses SCK, SDIO0, SDIO1 and one or more slave select lines as shown in *Figure 11-4*. 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 SPIn hardware to use SDIO0 and SDIO1 for half-duplex communication rather than full-duplex communication.

Figure 11-4: Dual Mode SPI Connection Diagram



11.3.5 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 master and slave must know the direction of transmission for each transaction. Quad mode SPI uses SCK, SDIO0, SDIO1, SDIO2, SDIO3 and one or more slave select pins.

Quad mode SPI transmits four bits per SCK cycle. Selection of Quad mode SPI is selected by setting *SPIn_CTRL2.data_width* to 2.

11.4 SPI Clock Configuration

11.4.1 Serial Clock

The SCK signal synchronizes data movement in and out of the device. The master drives SCK as an output to the slave's SCK pin. When SPIn is set to master mode, the SPIn bit rate generator creates the serial clock and outputs it on the configured SPIn_SCK pin. When SPIn is configured for slave operation the SPIn_SCK pin is an input from the external master and the

SPIn hardware synchronizes communications using the SCK input. Operating as a slave, if a SPIn slave select input is not asserted, the SPIn ignores any signals on the serial clock and serial data lines.

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

The SPIn 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.phase*) selects one of two different transfer formats.

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

11.4.2 SPI Peripheral Clock

The System Peripheral Clock, PCLK, drives the SPIn peripheral clock. The SPIn provides an internal clock, SPIn_CLK, that is used within the SPI peripheral for the base clock to control the module and generate the SCK clock when in master mode. Set the SPIn internal clock using the field *SPIn_CLK_CFG.scale* as shown in *Equation 11-1*. Valid settings for *SPIn_CLK_CFG.scale* are 0 to 8, allowing a divisor of 1 to 256.

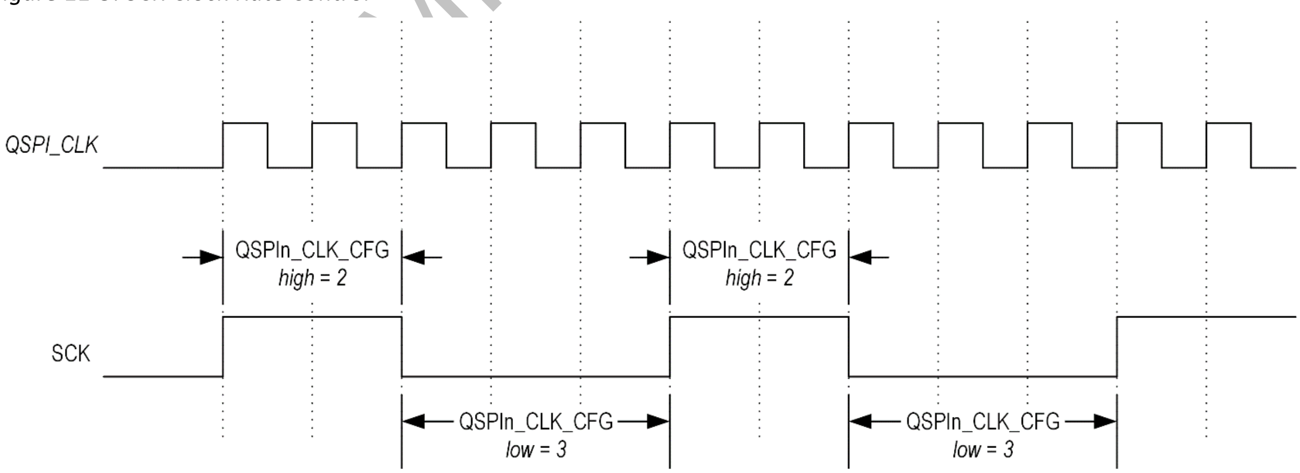
Equation 11-1: SPI Peripheral Clock

$$f_{SPI\_CLK} = \frac{f_{PCLK}}{2^{scale}}$$

11.4.3 Master Mode Serial Clock Generation

In master and multi-master mode the SCK clock is generated by the master. The SPIn 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 SPIn peripheral clock as a base value and the high and low values are a count of the number of f_{SPI_CLK} clocks. *Figure 11-5*, visually represents the use of the *SPIn_CLK_CFG.hi* and *SPIn_CLK_CFG.lo* fields for a non-50% duty cycle serial clock generation. See *Equation 11-2* and *Equation 11-3* for calculating the SCK high and low time from the *SPIn_CLK_CFG.hi* and *SPIn_CLK_CFG.lo* field values.

Figure 11-5: SCK Clock Rate Control



Equation 11-2: SCK High Time

$$t_{SCK\_HI} = t_{SPInCLK} \times \mathrm{SPIn\_CLK\_CFG.hi}$$

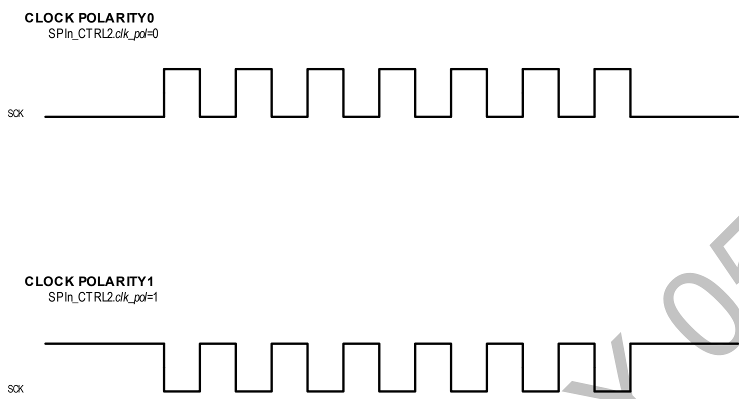
Equation 11-3: SCK Low Time

$$t_{SCK\_LOW} = t_{SPInCLK} \times \mathrm{SPIn\_CLK\_CFG.lo}$$

11.4.4 Clock Phase and Polarity Control

SPIn supports four combinations of clock and phase polarity as shown in *Table 11-4*. Clock polarity is controlled using the bit *SPIn_CTRL2.clkpol* and determines if the clock is active high or active low as shown in *Figure 11-6*. Clock polarity does not affect the transfer format for SPI. Clock phase determines when the data must be stable for sampling. Setting the clock phase to 0, *SPIn_CTRL2.phase* = 0, dictates the SPI data is sampled on the initial SPI clock edge regardless of clock polarity. Phase 1, *SPIn_CTRL2.phase* = 1, results in data sample occurring on the second edge of the clock regardless of clock polarity.

Figure 11-6: SPI Clock Polarity



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

Table 11-4: SPI Modes Clock Phase and Polarity Operation

SPI Mode	*SPIn_CTRL2* *phase*	*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

11.4.5 SPIn 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.

11.4.6 SPI Interrupts and Wakeups

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

The following FIFO interrupts are supported:

Transmit FIFO Empty
Transmit FIFO Threshold
Receive FIFO Full
Receive FIFO Threshold
Transmit FIFO Underrun

- Slave mode only, Master mode stalls the serial clock

Transmit FIFO Overrun
Receive FIFO Underrun
Receive FIFO Overrun

- Slave Mode only, Master Mode stalls the serial clock

SPIn supports interrupts for the internal state of the SPI as well as external signals. The following transmission interrupts are supported:

SSn asserted or deasserted
SPI transaction complete

- Master mode only

Slave mode transaction aborted
Multi-master fault

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

Receive FIFO full
Transmit FIFO empty
Receive FIFO threshold
Transmit FIFO threshold

11.5 Registers

See *Table 2-3: APB Peripheral Base Address Map* 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: SPIn Register* Summary. 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 2-1. Field Access Definitions* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 11-5: SPIn Register Summary

Offset	Register Name	Access	Description
[0x0000]	*SPIn_DATA*	R/W	*SPIn FIFO Data Register*
[0x0004]	*SPIn_CTRL0*	R/W	*SPIn Master Signals Control Register*
[0x0008]	*SPIn_CTRL1*	R/W	*SPIn Transmit Packet Size Register*
[0x000C]	*SPIn_CTRL2*	R/W	*SPIn Static Configuration Register*

Offset	Register Name	Access	Description
[0x0010]	*SPIn_SS_TIME*	R/W	*SPIn Slave Select Timing Register*
[0x0014]	*SPIn_CLK_CFG*	R/W	*SPIn Master Clock Configuration Register*
[0x001C]	*SPIn_DMA*	R/W	*SPIn DMA Control Register*
[0x0020]	*SPIn_INT_FL*	R/W1C	*SPIn Interrupt Flag Register*
[0x0024]	*SPIn_INT_EN*	R/W	*SPIn Interrupt Enable Register*
[0x0028]	*SPIn_WAKE_FL*	R/W1C	*SPIn Wakeup Flags Register*
[0x002C]	*SPIn_WAKE_EN*	R/W	*SPIn Wakeup Enable Register*
[0x0030]	*SPIn_STAT*	RO	*SPIn Status Register*

11.5.1 Register Details

Table 11-6: SPIn FIFO Data Register

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

Table 11-7: SPIn Control 0 Register

SPIn Control 0 Register			**SPIn_CTRL0**	**[0x0004]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:20	-	R/W	0	**Reserved for Future Use** Do not modify this field.
19:16	ss	R/W	0	**Master Slave Select** The SPIn includes up to four slave select lines for each port. This field selects which slave select pin is active when the next SPI transaction is started (*SPIn_CTRL0.start* = 1). One or more slave select pins can be selected for each SPI transaction by setting the bit for each slave select pin. For example, use SPIn_SS0 and SPIn_SS2 by setting this field to 0b0101 or select all slave selects by setting this field to 0b1111 *Note: This field is only used when the SPIn is configured for Master Mode (SPIn_CTRL0.master = 1).*
15:9	-	R/W	0	**Reserved for Future Use** Do not modify this field.

SPIn Control 0 Register				SPIn_CTRL0	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
8	ss_ctrl	R/W	0	**Master Slave Select Control** This field controls the behavior of the slave select pins at the completion of a transaction. The default behavior, *ss_ctrl* = 0, deasserts the slave select pin at the completion of the transaction. Set this field to 1 to leave the slave select pins asserted at the completion of the transaction. If the external device supports this behavior, leaving the slave select pins asserted allows multiple transactions without the delay associated with deassertion of the slave select pin between transactions. 0: Slave Select is deasserted at the end of a transmission 1: Slave Select stays asserted at the end of a transmission	
7:6	-	R/W	0	**Reserved for Future Use** Do not modify this field.	
5	start	R/W1O	0	**Master Start Data Transmission** Set this field to 1 to start a SPI master mode transaction. 0: No master mode transaction active. 1: Master initiates a data transmission. Ensure that all pending transactions are complete before setting this field to 1. *Note: This field is only used when the SPIn is configured for Master Mode (SPIn_CTRL0.master = 1).*	
4	ss_io	R/W	0	**Master Slave Select Signal Direction** Set the I/O direction for 0: Slave Select is an output 1: Slave Select is an input *Note: This field is only used when the SPIn is configured for Master Mode (SPIn_CTRL0.master = 1).*	
3:2	-	R/W	0	**Reserved for Future Use** Do not modify this field.	
1	master	R/W	0	**SPI Master Mode Enable** This field selects between slave mode and master mode operation for the SPI port. Write this field to 0 to operate as an SPI slave. Setting this field to 1 sets the port as an SPI master. 0: Slave mode SPI operation. 1: Master mode SPI operation.	
0	en	R/W	0	**SPI Enable/Disable** This field enables and disables the SPIn port. Disable the SPIn port by setting this field to 0. Disabling the SPIn port does not affect the SPIn FIFOs or register settings. Access to SPIn registers is always available. 0: SPIn port is disabled 1: SPIn port is enabled	

Table 11-8: SPIn Transmit Packet Size Register

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

Table 11-9: SPIn Control 2 Register

SPIn Control 2 Register				SPIn_CTRL2	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:20	-	R/W	0	**Reserved for Future Use** Do not modify this field.	
19:16	ss_pol	R/W	0	**Slave 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 for Future Use** Do not modify this field.	

SPIn Control 2 Register				SPIn_CTRL2 [0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**
13:12	data_width	R/W	0b00	**SPI Data Width** This field controls the number of data lines used for SPI communications. Three-wire SPI, *data_width* = 0 Set this field to 0, indicating SPIn_MOSI is used for half-duplex communication. Four-wire full-duplex SPI operation, *data_width* = 0 Set this field to 0, indicating SPIn_MOSI and SPIn_MISO are used for the SPI data output and input respectively. Dual SPI, *data_width* = 1, uses SPIn_SDIO0 and SPIn_SDIO1 for half-duplex communication. Quad SPI, *data_width* = 2, uses SPIn_SDIO0, SPIn_SDIO1, SPIn_SDIO2 and SPIn_SDIO3 for half-duplex communication. 0: 1-bit per SCK cycle (Three-wire half-duplex SPI and Four-wire full-duplex SPI) 1: 2-bits per SCK cycle (Dual SPI) 2: 4-bits per SCK cycle (Quad 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 2: 2-bits per character ... 14: 14-bits per character 15: 15-bits per character *Note: 1-bit and 9-bit character lengths are not supported.* *Note: 2-bit and 10-bit character lengths do not support maximum SCK speeds in Master mode. SPIn_CLK_CFG.scale must be > 0* *Note: For Dual and Quad mode SPI, the character size should be divisible by the number of bits per SCK cycle.*
7:2	-	R/W	0	**Reserved for Future Use** Do not modify this field.
1	cpol	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	cpha	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 11-10: SPIn Slave Select Timing Register

SPIn Slave Select Timing Register				SPIn_SS_TIME	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:24	-	R/W	0	**Reserved for Future Use** Do not modify this field. *Note: The SPIn_SS_TIME register bit settings only apply when SPIn is operating in Master mode (SPIn_CTRL0.master = 1)*	
23:16	inact	R/W	0	**Inactive Stretch** This field controls the number of system clocks the bus is inactive between the end of a transaction (Slave Select inactive) and the start of the next transaction (Slave Select active). 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255	
15:8	post	R/W	0	**Slave Select Hold Post Last SCK** Number of system clock cycles that SS remains active after the last SCK edge. 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255	
7:0	pre	R/W	0	**Slave Select Delay to First SCK** Set the number of system clock cycles the Slave Select is held active prior to the first SCK edge. 0: 256 1: 1 2: 2 3:3 … … 254: 254 255: 255	

Table 11-11: SPIn Master Clock Configuration Registers

SPIn Master Clock Configuration Register				SPIn_CLK_CFG	[0x0014]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:20	-	R/W	0	**Reserved for Future Use** Do not modify this field.	

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

Table 11-12: SPIn DMA Control Registers

SPIn DMA Control Register			SPIn_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
31	rx_dma_en	R/W	0	**RX DMA Enable** 0: Disabled. Any pending DMA requests are cleared 1: Enabled
30	-	R/W	0	**Reserved for Future Use** Do not modify this field.
29:24	rx_fifo_cnt	R	0	**Number of Bytes in the RX FIFO** Read returns the number of bytes currently in the RX FIFO
23	rx_fifo_clear	W	-	**Clear the RX FIFO** 1: Clear the RX FIFO and any pending RX FIFO flags in *SPIn_INT_FL*. This should be done when the RX FIFO is inactive. Writing a 0 has no effect.
22	rx_fifo_en	R/W	0	**RX FIFO Enabled** 0: Disabled 1: Enabled
21	-	R/W	0	**Reserved for Future Use** Do not modify this field.

SPIn DMA Control Register			SPIn_DMA	[0x001C]
Bits	**Name**	**Access**	**Reset**	**Description**
20:16	rx_fifo_level	R/W	0x00	**RX FIFO Threshold Level** Set this value to the desired RX FIFO threshold level. When the RX FIFO level crosses above this setting, a DMA request is triggered if enabled by setting *SPIn_DMA*.*rx_dma_enable*, and *SPIn_INT_FL*.*rx_thresh* becomes set. Valid values are 0 to 30. *Note: 31 is an invalid setting and reserved for future use.*
15	tx_dma_en	R/W	0	**TX DMA Enable** 0: Disabled. Any pending DMA requests are cleared 1: TX DMA is enabled
14	-	R/W	0	**Reserved for Future Use** Do not modify this field.
13:8	tx_fifo_cnt	RO	0	**Number of Bytes in the TX FIFO** Read this field to determine the number of bytes currently in the TX FIFO.
7	tx_fifo_clear	R/W	0	**TX FIFO Clear** Set this bit to clear the TX FIFO and all TX FIFO flags in the *SPIn_INT_FL* register. *Note: The TX FIFO should be disabled (SPIn_DMA.tx_fifo_en = 0) prior to setting this field.* *Note: Setting this field to 0 has no effect.*
6	tx_fifo_en	R/W	0	**TX FIFO Enabled** 0: Disabled 1: Enabled
5	-	R/W	0	**Reserved for Future Use** Do not modify this field.
4:0	tx_fifo_level	R/W	0x10	**TX FIFO Threshold Level** Set this value to the desired TX FIFO threshold level. When the TX FIFO count (*SPIn_DMA*.*tx_fifo_cnt*) falls below this value, a DMA request is triggered if enabled by setting *SPIn_DMA*.*tx_dma_en* and *SPIn_INT_FL*.*tx_thresh* becomes set.

Table 11-13: SPIn Interrupt Status Flags Registers

SPIn Interrupt Status Flags Register			SPIn_INT_FL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field.
15	rx_und	R/1	0	**RX FIFO Underrun Flag** Set when a read is attempted from an empty RX FIFO.
14	rx_ovr	R/W1C	0	**RX FIFO Overrun Flag** Set if SPI is in Slave Mode, and a write to a full RX FIFO is attempted. If the SPI is in Master Mode, this bit is not set as the SPI stalls the clock until data is read from the RX FIFO.

SPIn Interrupt Status Flags Register			SPIn_INT_FL	[0x0020]
Bits	**Name**	**Access**	**Reset**	**Description**
13	tx_und	R/W1C	0	**TX FIFO Underrun Flag** Set if SPI is in Slave Mode, and a read from empty TX FIFO is attempted. If SPI is in Master Mode, this bit is not set as the SPI stalls the clock until data is written to the empty TX FIFO.
12	tx_ovr	R/W1C	0	**TX FIFO Overrun Flag** Set when a write is attempted to a full TX FIFO.
11	m_done	R/W1C	0	**Master Data Transmission Done Flag** Set if SPIn is in Master Mode, and all transactions have completed. *SPIn_CTRL1.tx_number* has been reached.
10	-	R/W	0	**Reserved for Future Use** Do not modify this field.
9	abort	R/W1C	0	**Slave Mode Transaction Abort Detected Flag** Set if the SPI is in Slave Mode, and SS is deasserted before a complete character is received.
8	fault	R/W1C	0	**Multi-Master Fault Flag** Set if the SPI is in Master Mode, Multi-Master Mode is enabled, and a Slave Select input is asserted. A collision also sets this flag.
7:6	-	R/W	0	**Reserved for Future Use** Do not modify this field.
5	ssd	R/W1C	0	**Slave Select Deasserted Flag**
4	ssa	R/W1C	0	**Slave Select Asserted Flag**
3	rx_full	R/W1C	0	**RX FIFO Full Flag**
2	rx_thresh	R/W1C	0	**RX FIFO Threshold Level Crossed Flag** Set when the RX FIFO exceeds the value in *SPIn_DMA.rx_fifo_level*. Cleared once RX FIFO level drops below *SPIn_DMA.rx_fifo_level.*
1	tx_empty	R/W1C	1	**TX FIFO Empty Flag**
0	tx_thresh	R/W1C	0	**TX FIFO Threshold Level Crossed Flag** Set when the TX FIFO is less than the value in *SPIn_DMA.tx_fifo_level*. Cleared once TX FIFO level exceeds *SPIn_DMA.tx_fifo_level,*

Table 11-14: SPIn Interrupt Enable Registers

SPIn Interrupt Enable Register			SPIn_INT_EN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field.
15	rx_und	R/W	0	**RX FIFO Underrun Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled

SPIn Interrupt Enable Register				SPIn_INT_EN	[0x0024]
Bits	**Name**	**Access**	**Reset**	**Description**	
14	rx_ovr	R/W	0	**RX FIFO Overrun Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	
13	tx_und	R/W	0	**TX FIFO Underrun Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	
12	tx_ovr	R/W	0	**TX FIFO Overrun Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	
11	m_done	R/W	0	**Master Data Transmission Done Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	
10	-	R/W	0	**Reserved for Future Use** Do not modify this field.	
9	abort	R/W	0	**Slave Mode Abort Detected Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	
8	fault	R/W	0	**Multi-Master Fault Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	
7:6	-	R/W	0	**Reserved for Future Use** Do not modify this field.	
5	ssd	R/W	0	**Slave Select Deasserted Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	
4	ssa	R/W	0	**Slave Select Asserted Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	
3	rx_full	R/W	0	**RX FIFO Full Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	
2	rx_thresh	R/W	0	**RX FIFO Threshold Level Crossed Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	
1	tx_empty	R/W	0	**TX FIFO Empty Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	
0	tx_thresh	R/W	0	**TX FIFO Threshold Level Crossed Interrupt Enable** 0: Interrupt is disabled 1: Interrupt is enabled	

Table 11-15: SPIn Wakeup Status Flags Registers

SPIn Wakeup Flags Register		SPIn_WAKE_FL		[0x0028]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved for Future Use** Do not modify this field.
3	rx_full	R/W1C	0	**Wake on RX FIFO Full Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
2	rx_thresh	R/W1C	0	**Wake on RX FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
1	tx_empty	R/W1C	0	**Wake on TX FIFO Empty Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.
0	tx_thresh	R/W1C	0	**Wake on TX FIFO Threshold Level Crossed Flag** 0: Wake condition has not occurred. 1: Wake condition occurred.

Table 11-16: SPIn Wakeup Enable Registers

SPIn Wakeup Enable Register		SPIn_WAKE_EN		[0x002C]
Bits	**Name**	**Access**	**Reset**	**Description**
31:4	-	R/W	0	**Reserved for Future Use** Do not modify this field.
3	rx_full	R/W	0	**Wake on RX FIFO Full Enable** 0: Wake event is disabled 1: Wake event is enabled.
2	rx_thresh	R/W	0	**Wake on RX FIFO Threshold Level Crossed Enable** 0: Wake event is disabled 1: Wake event is enabled.
1	tx_empty	R/W	0	**Wake on TX FIFO Empty Enable** 0: Wake event is disabled 1: Wake event is enabled.
0	tx_thresh	R/W	0	**Wake on TX FIFO Threshold Level Crossed Enable** 0: Wake event is disabled 1: Wake event is enabled.

Table 11-17: SPIn Slave Select Timing Registers

SPIn Status Register		SPIn_STAT		[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	R/W	0	**Reserved for Future Use** Do not modify this field.

SPIn Status Register			SPIn_STAT	[0x0030]
Bits	**Name**	**Access**	**Reset**	**Description**
0	busy	R	0	**SPI Active Status** 0: SPIn is not active. In Master mode the *busy* flag is cleared when the last character is sent. In Slave mode the *busy* field is cleared when the configured slave select input is deasserted. 1: SPIn is active. In Master mode the *busy* flag is set when a transaction starts. In Slave mode the *busy* flag is set when a configured slave select input is asserted. *Note: SPIn_CTRL0, SPIn_CTRL1, SPIn_CTRL2, SPIn_SS_TIME, and SPIn_CLK_CFG should not be configured if this bit is set.*

12. I²C Master/Slave Serial Communications Peripheral (I2C)

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

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

12.1 I²C Master/Slave Features

Each I²C Master/Slave is compliant with the I²C Bus Specification and include the following features:

- Communicates via a serial data bus (SDA) and a serial clock line (SCL)
- Operates as either a master or slave device as a transmitter or receiver
- Supports I²C Standard Mode, Fast Mode, Fast Mode Plus and High Speed (Hs) mode
- Transfers data at rates up to:
  - 100kbps in Standard Mode
  - 400kbps in Fast Mode
  - 1Mbps in Fast Mode Plus
  - 3.4Mbps in Hs Mode
- Supports multi-master systems, including support for arbitration and clock synchronization for Standard, Fast and Fast Plus modes
- 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 via firmware
- Provides glitch filter and Schmitt trigger hysteresis on SDA and SCL
- Provides control, status, and interrupt events for maximum flexibility
- Provides independent 8-byte RX FIFO and 8-byte TX FIFO
- Provides TX FIFO preloading
- Provides programmable interrupt threshold levels for the TX and RX FIFO

12.2 Instances

The three instances of the peripheral are shown in *Table 12-1: MAX78000 I2C Peripheral Pins* lists the locations of the SDA and SCL signals for each of the I2Cn peripherals per package.

Table 12-1: MAX78000 I²C Peripheral Pins

I²C Instance	Alternate Function	Alternate Function #	Package 81-CTBGA
I2C0	I2C0_SCL	AF1	P0.10
	I2C0_SDA	AF1	P0.11
I2C1	I2C1_SCL	AF1	P0.16
	I2C1_SDA	AF1	P0.17
I2C2	I2C2_SCL	AF1	P0.30
	I2C2_SDA	AF1	P0.31

12.3 I²C Overview

12.3.1 I²C Bus Terminology

Table 12-2, *below*, contains terms and definitions used in this chapter for the I²C Bus Terminology.

Table 12-2: I²C Bus Terminology

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

12.3.2 I²C Transfer Protocol Operation

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

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

12.3.3 START and STOP Conditions

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

12.3.4 Master Operation

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

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

12.3.5 Acknowledge and Not Acknowledge

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

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

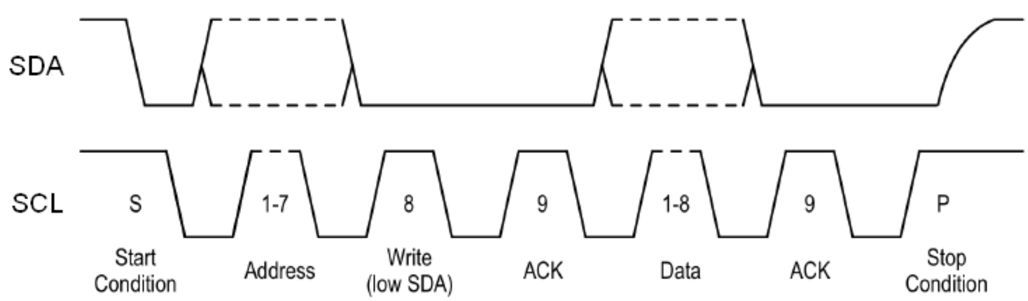
A receiver can generate a NACK after a byte transfer if any of the following conditions occur:

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

12.3.6 Bit Transfer Process

Both SDA and SCL circuits are open-drain, bidirectional circuits. Each requires an external pullup resistor that ensures each circuit is high when idle. The I^2C specification states that during data transfer, the SDA line can change state only when SCL is low, and that SDA is stable and able to be read when SCL is high as shown in *Figure 12-1*, *below*.

Figure 12-1: I^2C Write Data Transfer



An example of an I^2C data transfer is as follows:

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

12.4 I²C Configuration and Usage

12.4.1 SCL and SDA Bus Drivers

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

12.4.2 SCL Clock Configurations

The SCL frequency is dependent upon the values of I²C peripheral clock and the values of the external pullup resistor and trace capacitance on the SCL clock line.

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

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

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

The SCL high time is configured in the I²C Clock High Time register field *I2Cn_CLK_HI.scl_hi using* Equation 12-2. The SCL low time is configured in the I²C Clock Low Time register field *I2Cn_CLK_LO.scl_lo using* Equation 12-3. Each of these fields is 8-bits. The I²C frequency value is shown in *Equation 12-1*, *below*.

Equation 12-1: I²C Clock Frequency

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

Equation 12-2: I²C Clock High Time Calculation

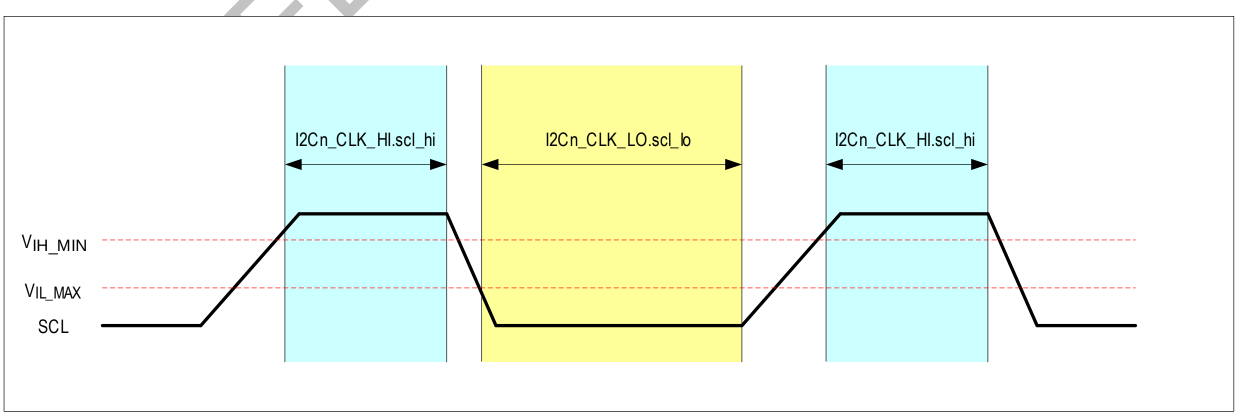
$$t_{SCL\_HI} = t_{I2C\_CLK} \times (\mathrm{I2Cn\_CLK\_HI}.scl\_hi + 1)$$

Equation 12-3: I²C Clock Low Time Calculation

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

Figure 12-2 shows the association between the SCL clock low and high times for Standard, Fast and Fast Plus I²C frequencies.

Figure 12-2: I²C SCL Timing for Standard, Fast and Fast-Plus Modes



During synchronization, external masters or external slaves may be driving SCL simultaneously. This affects the SCL duty cycle. By monitoring SCL, the controller can determine whether an external master or slave is holding SCL low. In either case, the controller waits until SCL is high before starting to count the number of SCL high cycles. Similarly, if an external master pulls SCL low before the controller has finished counting SCL high cycles, then the controller starts counting SCL low cycles and releases SCL once the time period, *I2Cn_CLK_LO.scl_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. These include bus loading, other devices on the bus holding SCL low, and the filter delay time of this device.

12.4.4 SCL Clock Generation for Hs-mode

To operate the I^{2}C interface in Hs-mode at its maximum speed (~3.4MHz), values to be programmed into the *I2Cn_HS_CLK.hs_clk_lo* register and *I2Cn_HS_CLK.hs_clk_hi* register must be determined. Since the Hs-mode operation is entered by first using one of the lower speed modes for pre-amble, a relevant lower speed mode must also be configured. See *SCL Clock Generation for Standard, Fast and Fast-Plus Modes* for information regarding configuration of lower speed modes.

12.4.4.1 Hs-Mode Timing

With I^{2}C bus capacitances less than 100pf, the following specifications are extracted from the I^{2}C-bus Specification User Manual Rev. 6 April 2014 *https://www.nxp.com/docs/en/user-guide/UM10204.pdf*

t_{LOW_MIN} = 160ns, the minimum low time for the I^{2}C bus clock.

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

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

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

12.4.4.2 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 12-4: I^{2}C Target SCL Frequency

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

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

Equation 12-5: Determining the I2Cn_HS_CLK.hs_clk_lo Register Value

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

Equation 12-6: Determining the I2Cn_HS_CLK.hs_clk_hi Register Value

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

Equation 12-7: The Calculated Frequency of the I^{2}C Bus Clock Using the Results of *Equation 12-5* and *Equation 12-6*

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

Table 12-3: Calculated I2C Bus Clock Frequencies shows the I^{2}C bus clock calculated frequencies given different f_{SYS_CLK} frequencies.

Table 12-3: Calculated I²C Bus Clock Frequencies

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

12.4.5 I²C Addressing

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

Table 12-4: I²C Slave Address Format

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

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

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

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

In 10-bit addressing mode (I2Cn_MASTER_CTRL.*sl_ex_addr* = 1), the first byte the master sends is the 10-bit Slave Addressing byte which includes the first two bits of the 10-bit address, followed by a 0 for the R/W bit. That is followed by a second byte representing the remainder of the 10-bit address. If the operation is a write, this is followed by data bytes to be written to the slave. If the operation is a read, it is followed by a repeated START. Firmware then writes the 10-bit address again with a 1 for the R/W bit. This I²C then starts receiving data from the slave device.

12.4.6 I²C Master Mode Operation

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

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

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

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

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

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

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

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

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

1. The I^2C peripheral is disabled, **or**
2. The I^2C bus is guaranteed to be idle/free.

If this peripheral is the only master on the bus, then changing the registers outside of a transaction (*I2Cn_MASTER_CTRL*.start = 0) will satisfy this requirement:

- *I2Cn_CTRL.mst*
- *I2Cn_CTRL.rx_mode*
- *I2Cn_CTRL.scl_pp_mode*
- *I2Cn_CTRL.hs_mode*
- *I2Cn_RX_CTRL1.rx_cnt*
- *I2Cn_MASTER_CTRL.sl_ex_addr*
- *I2Cn_MASTER_CTRL.mcode*
- *I2Cn_CLK_LO.scl_lo*
- *I2Cn_CLK_HI.scl_hi*
- *I2Cn_HS_CLK.hs_clk_lo*
- *I2Cn_HS_CLK.hs_clk_hi*

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

- All interrupt flags and interrupt enables
- *I2Cn_TX_CTRL0.tx_thresh*
- *I2Cn_RX_CTRL0.rx_thresh*
- *I2Cn_TIMEOUT.to*
- *I2Cn_DMA.rx_en*
- *I2Cn_DMA.tx_en*
- *I2Cn_FIFO.data*
- *I2Cn_MASTER_CTRL.start*
- *I2Cn_MASTER_CTRL.restart*
- *I2Cn_MASTER_CTRL.stop*

12.4.6.1 I²C Master Mode Receiver Operation

When in Master Mode, initiating a Master Receiver operation begins with the following sequence:

1. Write the number of data bytes to receive to the I²C Receive Count field (*I2Cn_RX_CTRL1*.*rx_cnt)*.
2. Write the I²C Slave Address Byte to the *I2Cn_FIFO* register with the R/W bit set to 1
3. Send a START condition by setting *I2Cn_MASTER_CTRL*.*start* = 1
4. The slave address is transmitted by the controller from the *I2Cn_FIFO* register.
5. The I²C controller receives an ACK from the slave and the controller sets the address ACK interrupt flag (*I2Cn_INT_FL0*.*addr_ack* = 1).
6. The I²C controller receives data from the slave and automatically ACKs each byte. Firmware must retrieve this data by reading the *I2Cn_FIFO* register.
7. Once *I2Cn_RX_CTRL1*.rx_cnt data bytes have been received, the I²C controller sends a NACK to the slave and sets the Transfer Done Interrupt Status Flag (*I2Cn_INT_FL0*.*done* = 1).
8. If *I2Cn_MASTER_CTRL*.*restart* or *I2Cn_M*.*stop* is set, then the I²C controller sends a repeated START or STOP, respectively.

12.4.6.2 I²C Master Mode Transmitter Operation

When in Master Mode, initiating a Master Transmitter operation begins with the following sequence:

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

12.4.6.3 I²C Multi-Master Operation

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

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

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

If the I^2C master peripheral detects it has lost arbitration, it stops generating SCL; sets *I2Cn_INT_FL0.areri*; sets *I2Cn_INT_FL0.tx_lock_out*, flushing any remaining data in the TX FIFO; and clears *I2Cn_MASTER_CTRL.start*, *I2Cn_MASTER_CTRL.restart*, and *I2Cn_MASTER_CTRL.stop* to 0. So long as the peripheral is not itself addressed by the winning master, the I^2C peripheral stays in master mode (*I2Cn_CTRL.mst = 1)*. If *at any time* another master addresses this peripheral using the address programmed in I2Cn_SLAVE_ADDR.*slave_addr*, then the I^2C peripheral clears *I2Cn_CTRL.mst* to 0 and begins responding as a slave. This can even occur during the same address transmission during which the peripheral lost arbitration.

Note: Arbitration loss is considered an error condition, and like the other error conditions will set I2Cn_INT_FL0.tx_lock_out. Therefore, after an arbitration loss, firmware will need to clear I2Cn_INT_FL0.tx_lock_out and reload the TX FIFO.

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

1. Wait for the other master to complete the transaction(s) by sending a STOP,
2. Count out the bus free time using $t_{BUF} = t_{SCL\_LO}$ (see Equation 12-3)*, and then*
3. Send a START condition and begin transmitting the slave address byte(s) in the TX FIFO, followed by the rest of the transfer.

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

12.4.7 I^2C Slave Mode Operation

When in slave mode, the I^2Cn peripheral operates as a slave device on the I^2C bus and responds to an external master's requests to transmit or receive data. To configure the I^2Cn peripheral as a slave, write the *I2Cn_CTRL.mst* bit to zero. The I^2Cn clock is driven by the master on the bus, so the SCL device pin will be driven by the external master and *I2Cn_STATUS.clk_mode* remains a zero. The desired slave address must be set by writing to the *I2Cn_SLAVE_ADDR.slave_addr* register.

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

- *I2Cn_CTRL.mst* = 0 for Slave operation.
- *I2Cn_CTRL.gen_call_addr*
- *I2Cn_CTRL.rx_mode*
  - The recommended value for this field is 0. *Note that a setting of 1 is incompatible with slave mode operation with clock stretching disabled (I2Cn_CTRL.scl_clk_stretch_dis = 1).*
- *I2Cn_CTRL.scl_clk_stretch_dis*
- *I2Cn_CTRL.hs_mode*
- *I2Cn_RX_CTRL0.dnr*
  - SMBus/PMBus applications should set this to 0, while other applications should set this to 1.
- *I2Cn_TX_CTRL0.tx_nack_afd*
- *I2Cn_TX_CTRL0.tx_amr_afd*
- *I2Cn_TX_CTRL0.tx_amw_afd*
- *I2Cn_TX_CTRL0.tx_amgc_afd*
- *I2Cn_TX_CTRL0.tx_preload*
  - Recommended value is 0 for applications that can tolerate slave clock stretching (*I2Cn_CTRL.scl_clk_stretch_dis* = 0).
  - Recommend value is 1 for applications that do not allow slave clock stretching (*I2Cn_CTRL.scl_clk_stretch_dis* = 1).
- *I2Cn_CLK_HI.scl_hi*
  - Applies to Slave Mode when clock stretching is enabled (*I2Cn_CTRL.scl_clk_stretch_dis* = 0)
    - This will be used to satisfy $t_{SU;DAT}$ after clock stretching; program it so that the value defined by *Equation 12-2* is >= $t_{SU;DAT(min)}$
- *I2Cn_HS_CLK.hs_clk_hi*
  - Applies to Slave Mode in Hs Mode when clock stretching is enabled (*I2Cn_CTRL.scl_clk_stretch_dis* = 0)
    - This will be used to satisfy $t_{SU;DAT}$ after clock stretching during Hs-Mode operation; program it so that the value defined by *Equation 12-6* is >= $t_{SU;DAT(min)}$
- I2Cn_SLAVE_ADDR.*slave_addr*
- *I2Cn_SLAVE_ADDR.ex_addr*

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

- All Interrupt Flags and Interrupt Enables
- *I2Cn_TX_CTRL0.tx_thresh* and *I2Cn_RX_CTRL0.rx_thresh*
  - TX and RX FIFO Threshold Levels
- *I2Cn_TX_CTRL1.tx_rdy*
  - Transmit Ready (Can only be cleared by hardware)
- *I2Cn_TIMEOUT.to*
  - Time Out Control
- *I2Cn_DMA.rx_en* and *I2Cn_DMA.tx_en*
  - TX and RX DMA Enables
- *I2Cn_FIFO.data*
  - FIFO access register

12.4.7.1 Slave Transmitter

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

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

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

1. With *I2Cn_CTRL.i2c_en* = 0, initialize all relevant registers, including specifically for this mode *I2Cn_CTRL.scl_clk_stretch_dis* = 0, *I2Cn_TX_CTRL0*[5:2] = 0x8 and *I2Cn_TX_CTRL0.tx_preload* = 0. Don't forget to program *I2Cn_CLK_HI.scl_hi* and *I2Cn_HS_CLK.hs_clk_hi* with appropriate values satisfying $t_{SU;DAT}$ (and HS $t_{SU;DAT}$).
2. Software sets *I2Cn_CTRL.i2c_en* = 1.
   a. The controller is now listening for its address. For either a transmit (R/W = 1) or receive (R/W = 0) operation, the peripheral will respond to its address with an ACK.
   b. When the address match occurs, hardware will set *I2Cn_INT_FL0.addr_match* and *I2Cn_INT_FL0.tx_lock_out*.
3. Software waits for *I2Cn_INT_FL0.addr_match* =1, either via polling the interrupt flag or setting *I2Cn_INT_EN0.addr_match* to interrupt the CPU.
4. After reading *I2Cn_INT_FL0.addr_match* =1, Software reads *I2Cn_CTRL.read* to determine whether the transaction is a transmit (read=1) or receive (read=0) operation. In this case we assume read=1, indicating transmit.
   a. At this point, Hardware will hold SCL low until Software clears *I2Cn_INT_FL0.tx_lock_out* and loads data into the FIFO.
5. Software clears *I2Cn_INT_FL0.addr_match* and *I2Cn_INT_FL0.tx_lock_out*. Now that *I2Cn_INT_FL0.tx_lock_out* is 0, Software can begin loading the transmit data into *I2Cn_FIFO*.
6. As soon as there is data in the FIFO, Hardware will release SCL (after counting out *I2Cn_CLK_HI.scl_hi*) and send out the data on the bus.
7. While the master keeps requesting data and sending ACKs, *I2Cn_INT_FL0.ddone* will remain 0 and Software should continue to monitor the TX FIFO and refill it as needed.
   a. The FIFO level can be monitored synchronously via the TX FIFO status/interrupt flags, or asynchronously by setting *I2Cn_TX_CTRL0.tx_thresh* and setting *I2Cn_INT_EN0.tx_thresh* interrupt.
   b. If the TX FIFO ever empties during the transaction, the Hardware will start clock stretching and wait for it to be refilled.
8. The master ends the transaction by sending a NACK. Once this happens the *I2Cn_INT_FL0.done* interrupt flag is set, and Software can stop monitoring the TX FIFO.
9. The transaction is complete, Software should "clean up", including clearing *I2Cn_INT_FL0.done* and clearing *I2Cn_INT_EN0.tx_thresh* interrupt. We return to step 3, waiting on an address match.
10. If Software needs to know how many data bytes were transmitted to the master, it should check the TX FIFO level as soon as Software sees *I2Cn_INT_FL0.done* = 1 and use that to determine how many data bytes were successfully sent.
    a. Note that any data remaining in the TX FIFO will be discarded prior to the next transmit operation; it is NOT necessary for Software to manually flush the TX FIFO for this to occur.

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

To use slave transmit preload mode:

1. With *I2Cn_CTRL.i2c_en* = 0, initialize all relevant registers, including specifically for this mode *I2Cn_CTRL.cl_clk_stretch_dis* = 1, *I2Cn_TX_CTRL0*[5:2] = 0xF and *I2Cn_TX_CTRL0.tx_preload* = 1.
2. Software sets *I2Cn_CTRL.i2c_en* = 1.
   a. Even though the controller is enabled, at this point it will not ACK an address match with R/W=1 until Software sets *I2Cn_TX_CTRL1.tx_ready* =1.
3. Software prepares for the transmit operation by loading data into the transmit FIFO, enabling DMA, setting *I2Cn_TX_CTRL0.tx_thresh* and setting *I2Cn_INT_EN0.tx_thresh* interrupt, etc.
   a. If clock stretching is disabled, then an empty TX FIFO during the transmit operation will cause a TX underrun error. Therefore, firmware should take any necessary steps to avoid an underrun *prior* to setting *I2Cn_TX_CTRL1.tx_ready* =1.
   b. If clock stretching is enabled, then an empty TX FIFO will not cause a TX underrun error. However, it is recommended to follow the same preparation steps in order to minimize the amount of time spent clock stretching, which will let the transaction complete as quickly as possible.
4. Once Software has prepared for the transmit operation, it sets *I2Cn_TX_CTRL1.tx_ready* =1.
   a. The controller is now fully enabled and will respond with an ACK to an address match.
   b. Hardware will set *I2Cn_INT_FL0.addr_match* once an address match has occurred. *I2Cn_INT_FL0.tx_lock_out*. will NOT be set and will remain 0.
5. Software waits for *I2Cn_INT_FL0.addr_match* =1, either via polling the interrupt flag or setting *I2Cn_INT_EN0*.amie to interrupt the CPU.
6. After seeing *I2Cn_INT_FL0.addr_match* =1, Software reads *I2Cn_CTRL.read* to determine whether the transaction is a transmit (read=1) or receive (read=0) operation. In this case we assume *I2Cn_CTRL.read*, indicating transmit.
   a. At this point, Hardware will begin sending out the data that was preloaded into the TX FIFO.
   b. Once the first data byte is sent, Hardware will also automatically clear *I2Cn_TX_CTRL1.tx_ready* to 0.
7. While the master keeps requesting data and sending ACKs, *I2Cn_INT_FL0.done* will remain 0 and Software should continue to monitor the TX FIFO and refill it as needed.
   a. The FIFO level can be monitored synchronously via the TX FIFO status/interrupt flags, or asynchronously by setting *I2Cn_TX_CTRL0.tx_thresh* and setting *I2Cn_INT_EN0.tx_thresh* interrupt.
   b. If clock stretching is disabled and the TX FIFO ever empties during the transaction, the Hardware will set *I2Cn_INT_FL1.tx_underflow* = 1 and send 0xFF for all following data bytes requested by the master.
8. The master ends the transaction by sending a NACK. Once this happens the *I2Cn_INT_FL0.done* interrupt flag is set.
   a. If the TX FIFO goes empty at the same time that the master indicates the transaction is complete by sending a NACK, this is not considered an underrun event, and the *I2Cn_INT_FL1.tx_underflow* flag will remain 0.
9. The transaction is complete, Software should "clean up", which should include clearing *I2Cn_INT_FL0.done*. We return to step 3 and prepare for the next transaction.
   a. If Software needs to know how many data bytes were transmitted to the master, it should check the TX FIFO level as soon as Software sees *I2Cn_INT_FL0.done* = 1 and use that to determine how many data bytes were successfully sent.
   b. By default, any data remaining in the TX FIFO will NOT be discarded, and instead will be reused for the next transmit operation.
   c. If this is not desired, Software can flush the TX FIFO. The safest way to do this is by clearing and then re-setting *I2Cn_CTRL.i2c_en*. This will flush both the TX and RX FIFOs.

Once a slave starts transmitting out of its *I2Cn_FIFO*, detection of out of sequence STOP, START or RESTART condition will terminate the current transaction. When a transaction is terminated in such manner, *I2Cn_INT_FL0.start_er* or *I2Cn_INT_FL0.stop_er* will be set to 1.

If the TX FIFO is not ready (*I2Cn_TX_CTRL1.tx_ready* = 0) and the I²C controller receives a data read request from the master, the hardware automatically sends a NACK at the end of the first address byte. The setting of the Do Not Respond field is ignored by the hardware in this case because the only opportunity to send a NACK for an I²C read transaction is after the address byte.

12.4.7.2 Slave Receiver

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

Program flow for a receive operation is as follows:

1. With *I2Cn_CTRL.i2c_en* = 0, initialize all relevant registers.
2. Set *I2Cn_CTRL.i2c_en* = 1.
    a. If an address match with R/W=0 occurs, and the RX FIFO is empty, the peripheral will respond with an ACK and the *I2Cn_INT_FL0.addr_match* flag will be set.
    b. If the RX FIFO is not empty, then depending on the value of *I2Cn_RX_CTRL0.dnr*, the peripheral will NACK either the address byte (*I2Cn_RX_CTRL0.dnr* = 1) or the first data byte (*I2Cn_RX_CTRL0.dnr* = 0).
3. Wait for *I2Cn_INT_FL0.addr_match* = 1, either by polling or by enabling the *wr_addr_match* interrupt to the CPU. Once a successful address match occurs, hardware will set *I2Cn_INT_FL0.addr_match* =1.
4. Read *I2Cn_CTRL.read* to determine whether the transaction is a transmit (*I2Cn_CTRL.read* = 1) or receive (*I2Cn_CTRL.read* = 0) operation. In this case we assume *I2Cn_CTRL.read* = 0, indicating receive. At this point, the device will begin receiving data into the RX FIFO.
5. Clear *I2Cn_INT_FL0.addr_match*, and while the master keeps sending data, *I2Cn_INT_FL0.done* will remain 0 and software should continue to monitor the RX FIFO and empty it as needed.
    a. The FIFO level can be monitored synchronously via the RX FIFO status/interrupt flags, or asynchronously by setting *I2Cn_RX_CTRL0.rx_thresh* and enabling the *I2Cn_INT_FL0.rx_thresh* interrupt.
    b. If the RX FIFO ever fills up during the transaction, then hardware will set *I2Cn_INT_FL1.rx_underflow* and then either:
        i. If *I2Cn_CTRL.scl_clk_stretch_dis* = 0, start clock stretching and wait for software to read from the RX FIFO, ***or***
        ii. If *I2Cn_CTRL.scl_clk_stretch_dis* = 1, respond to the master with a NACK and the last byte is discarded.
6. The master ends the transaction by sending a RESTART or STOP. Once this happens the *I2Cn_INT_FL0.done* interrupt flag is set, and software can stop monitoring the RX FIFO.
7. Once a slave starts receiving into its RX_FIFO, detection of out of sequence STOP, START or RESTART condition will release the I²C bus to the Idle state and hardware will set *I2Cn_INT_FL0.start_er* or *I2Cn_INT_FL0.stop_er* to 1.

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

12.4.8 I²C 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, firmware determines the cause of the interrupt by reading the I²C Interrupt Flags registers *I2Cn_INT_FL0* and *I2Cn_INT_FL1*. Interrupts can be generated for the following events:

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

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

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

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

12.4.9 TX FIFO and RX FIFO

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

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

To offload work from the CPU, the DMA can read and write to each FIFO. See section *I2C DMA Control* for more information on configuring the DMA.

During a receive transaction (which during master operation is a READ, and during slave operation is a WRITE), received bytes are automatically written to the RX FIFO. Software should monitor the RX FIFO level and unload data from it as needed by reading *I2Cn_FIFO*. If the receive FIFO becomes full during a master mode transaction, then the controller sets the I2Cn_INT_FL1.*rx_underflow* the I2Cn_INT_FL1.*rx_underflow* bit and one of two things will happen depending on the value of I2Cn_CTRL.*scl_clk_stretch_dis*:

- If clock stretching is enabled (I2Cn_CTRL.*scl_clk_stretch_dis* = 0), then the controller stretches the clock until software makes space available in the RX FIFO by reading from I2Cn_FIFO. Once space is available, the peripheral moves the data byte from the shift register into the RX FIFO, the SCL device pin is released, and the master is free to continue the transaction.
- If clock stretching is disabled (I2Cn_CTRL.*scl_clk_stretch_dis* = 1), then the controller responds to the master with a NACK and the data byte is lost. The master can return the bus to idle with a STOP condition or start a new transaction with a RESTART condition.

During a transmit transaction (which during master operation is a WRITE, and during slave operation is a READ), either user software or the DMA can provide data to be transmitted by writing to the TX FIFO. Once the peripheral finishes transmitting each byte, it removes it from the TX FIFO and, if available, begins transmitting the next byte.

Interrupts can be generated for the following FIFO status:

- TX FIFO level less than or equal to threshold
- RX FIFO level greater than or equal to threshold
- TX FIFO underflow
- RX FIFO overflow
- TX FIFO locked for writing

Both the RX FIFO and TX FIFO are flushed when the I^2Cn port is disabled by clearing I2Cn_CTRL.*i2c_en*=0. While the peripheral is disabled, writes to the TX FIFO have no effect and reads from the RX FIFO return 0xFF.

The TX FIFO and RX FIFO can be flushed by setting the Transmit FIFO Flush bit (I2Cn_TX_CTRL0.*tx_flush*=1) or the Receive FIFO Flush bit (I2Cn_RX_CTRL0.*rx_flush*=1), respectively. In addition, under certain conditions the TX FIFO is automatically locked by hardware and flushed so stale data is not unintentionally transmitted. The TX FIFO is automatically flushed, and writes locked out from software under the following conditions:

- General Call Address Match. Automatic flushing and lockout can be disabled by setting I2Cn_TX_CTRL0.*tx_amgc_afd*.
- Slave Address Match Write. Automatic flushing and lockout can be disabled by setting I2Cn_TX_CTRL0.*tx_amw_afd*.
- Slave Address Match Read. Automatic flushing and lockout can be disabled by setting I2Cn_TX_CTRL0.*tx_amr_afd*.
- During operation as a slave transmitter, a NACK is received. Automatic flushing and lockout can be disabled by setting I2Cn_TX_CTRL0.*tx_nack_afd*.
- Any of the following interrupts: Arbitration Error, Timeout Error, Master Mode Address NACK Error, Master Mode Data NACK Error, Start Error, and STOP Error. Automatic flushing cannot be disabled for these conditions.

When the above conditions occur, the TX FIFO is flushed so that data intended for a previous transaction is not unintentionally transmitted for a new transaction. In addition to flushing the the TX FIFO, the Transmit Lockout Flag is set (I2Cn_INT_FL0.*tx_lock_out* = 1) and writes to the TX FIFO are ignored until firmware acknowledges the external event by clearing I2Cn_INT_FL0.*tx_lock_out*.

12.4.10 TX FIFO Preloading

There may be situations during slave mode operation where software wants to preload the TX FIFO prior to a transmission, such as when clock stretching is disabled. In this scenario, rather than responding to an external master requesting data with an ACK and clock stretching while software writes the data to the TX FIFO, the controller will instead respond with a NACK until software has preloaded the requested data into the TX FIFO.

When TX FIFO Preloading is enabled, the software controls ACKs to the external master using the TX Ready (I2Cn_TX_CTRL1.*tx_ready*) bit. When I2Cn_TX_CTRL1.*tx_ready* is set to 0, hardware automatically NACKs all read transactions from the Master. Setting *I2Cn_TX_CTRL1.tx_ready* to 1 sends an ACK to the Master on the next read transaction and transmits the data in the TX FIFO. Preloading the TX FIFO should be complete prior to setting the *I2Cn_TX_CTRL1.tx_ready* field to 1.

The required steps for implementing TX FIFO Preloading in an application are as follow:

1. Enable TX FIFO Preloading by setting *I2Cn_TX_CTRL0.tx_preload* to 1.
   This will automatically clear *I2Cn_TX_CTRL1.tx_ready* to 0
2. If the TX FIFO Lockout Flag (*I2Cn_INT_FL0.tx_lock_out*) is set to 1, write 1 to clear the flag and enable writes to the TX FIFO.
3. Enable DMA or Interrupts if required.
4. Load the TX FIFO with the data to send when the Master sends the next read request.
5. Set *I2Cn_TX_CTRL1.tx_ready* to 1 to automatically let the hardware send the preloaded FIFO on the next read from a Master.
6. *I2Cn_TX_CTRL1.tx_ready* is cleared by hardware once it finishes transmitting the first byte, and data is transmitted from the TX FIFO. Once cleared, the application firmware may repeat the Preloading process or disable TX FIFO Preloading.

Note: To prevent the preloaded data from being cleared when the master tries to read it, firmware must at least set I2Cn_TX_CTRL0.tx_amr_afd to 1, disabling autoflush on READ address match. The software will determine whether the other autoflush disable bits should be set. For example, if a master uses I2C WRITE transactions to determine what data the slave should send in the following READ transactions, then software can clear I2Cn_TX_CTRL0.tx_amw_afd to 0. Then when a WRITE occurs, the TX FIFO is flushed, giving firmware time to load the new data. For the READ transaction, the external master can poll the slave address until the new data has been loaded and I2Cn_TX_CTRL1.tx_ready is set, at which point the peripheral responds with an ACK.

12.4.11 Interactive Receive Mode (IRXM)

In some situations, the I2Cn might want to inspect and respond to each byte of received data. In this case, Interactive Receive Mode (IRXM) can be used. Interactive Receive Mode is enabled by setting *I2Cn_CTRL.rx_mode* = 1. If Interactive Receive Mode is enabled, it must occur before any I^2C transfer is initiated.

When Interactive Receive Mode (IRXM) is enabled, after every data byte received the I2Cn peripheral automatically holds SCL low before the ACK bit. Additionally, after the 8th SCL falling edge, the I2Cn peripheral sets the IRXM Interrupt Status Flag (*I2Cn_INT_FL0.rx_mode* = 1). Application firmware must read the data and generate a response (ACK or NACK) by setting the IRXM Acknowledge (*I2Cn_CTRL.rx_mode_ack*) bit accordingly. Send an ACK by clearing the *I2Cn_CTRL.rx_mode_ack* bit to 0. Send a NACK by setting the *I2Cn_CTRL.rx_mode_ack* bit to 1.

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

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

During IRXM, received data is not placed in the RX FIFO. Instead, the *I2Cn_FIFO* address is repurposed to directly read the receive shift register, bypassing the RX FIFO. Therefore, before disabling Interactive Receive Mode, firmware 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 RX FIFO returns 0xFF.

Note: Interactive Receive Mode does not apply to address bytes, only to data bytes.

Note: Interactive Receive Mode does not apply to general call address responses or START byte responses.

Note: When enabling Interactive Receive Mode and operating as a slave, clock stretching must remain enabled (I2Cn_CTRL.scl_clk_stretch_dis = 0).

12.4.12 Clock Stretching

When the I2Cn peripheral requires some response or intervention from the application firmware in order to continue with a transaction, it will hold SCL low, preventing the transfer from continuing. This is called 'clock stretching' or 'stretching the clock'. While the I^{2}C Bus Specification defines the term 'clock stretching' to only apply to a slave device holding the SCL line low, this section describes situations where the I2Cn peripheral holds the SCL line low in either slave *or* master mode and refers to *both* as clock stretching.

When the I2Cn peripheral stretches the clock, it typically does so in response to either a full RX FIFO during a receive operation, or an empty TX 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 Interactive Receive Mode (*I2Cn_CTRL.rx_mode* = 1), the peripheral can also clock stretch *before* the ACK bit, allowing firmware to decide whether to send an ACK or NACK.

For a transmit operation (as either master or slave), when the TX FIFO is empty, SCL is automatically held low after the ACK bit and before the next data byte begins. To stop clock stretching and continue the transaction, firmware must write data to *I2Cn_FIFO.data*. If operating in master mode, however, instead of sending more data, firmware may also set either *I2Cn_MASTER_CTRL.stop* or *I2Cn_MASTER_CTRL.restart* to send a STOP or RESTART condition, respectively.

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

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

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

12.4.13 I^{2}C Bus Timeout

The Timeout register, *I2Cn_TIMEOUT.to,* is used to detect bus errors. *Equation 12-8* and *Equation 12-9* show equations for calculating the maximum and minimum timeout values based on the value loaded into the *I2Cn_TIMEOUT.to* field.

Equation 12-8: I^{2}C Timeout Maximum

$$t_{TIMEOUT} \leq \left(\frac{1}{f_{I2C\_CLK}}\right) \times \left((\text{I2Cn\_TIMEOUT}.to \times 32) + 3\right)$$

Due to clock synchronization, the timeout is guaranteed to meet the following minimum time calculation shown in *Equation 12-9*.

Equation 12-9: I^{2}C Timeout Minimum

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

The timeout feature is disabled when *I2Cn_TIMEOUT.to* = 0 and is enabled for any non-zero value. When the timeout is enabled, the timeout timer starts counting when the I2Cn peripheral hardware drives SCL low and is reset by the I2Cn peripheral hardware when the SCL line is released.

The timeout counter only monitors if the I2Cn peripheral hardware is driving the SCL line low. It does not monitor if an external I2Cn device is actively holding the SCL line low. The timeout counter also does not monitor the status of the SDA line.

If the timeout timer expires, a bus error condition has occurred. When a timeout error occurs, the I2Cn peripheral hardware releases the SCL and SDA lines and sets the timeout error interrupt flag to 1 (*I2Cn_INT_FL0.to_er* = 1).

For applications where the device may 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.to* = 0).

12.4.14 I²C DMA Control

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

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

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

Equation 12-10: DMA Burst Size Calculation for I²C Transmit

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

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

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

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

Equation 12-11: DMA Burst Size Calculation for I²C Receive

$$DMA\ Burst\ Size \leq \mathrm{I2Cn\_RX\_CTRL0}.rx\_thresh$$

$$where\ 1 \leq \mathrm{I2Cn\_RX\_CTRL0}.rx\_thresh \leq 8$$

Applications trying to avoid receive overflow and/or clock stretching should use a smaller burst size and lower *I2Cn_RX_CTRL0.rx_thresh*. This results in reading from the Receive FIFO more frequently but increases internal bus traffic.

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

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

12.5 Registers

See *Table 2-3: APB Peripheral Base* Address Map 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 12-5: I2C Register Summary*. 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 2-1. Field Access Definitions* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 12-5: I^{2}C Register Summary

Offset	Register	Description
[0x0000]	*I2Cn_CTRL*	I^{2}C Control Register
[0x0004]	*I2Cn_STATUS*	I^{2}C Status Register
[0x0008]	*I2Cn_INT_FL0*	I^{2}C Interrupt Flags 0 Register
[0x000C]	*I2Cn_INT_EN0*	I^{2}C Interrupt Enable 0 Register
[0x0010]	*I2Cn_INT_FL1*	I^{2}C Interrupt Flags 1 Register
[0x0014]	*I2Cn_INT_EN1*	I^{2}C Interrupt Enable 1 Register
[0x0018]	*I2Cn_FIFO_LEN*	I^{2}C FIFO Length Register
[0x001C]	*I2Cn_RX_CTRL0*	I^{2}C Receive Control 0 Register
[0x0020]	*I2Cn_RX_CTRL1*	I^{2}C Receive Control 1 Register
[0x0024]	*I2Cn_TX_CTRL0*	I^{2}C Transmit Control 0 Register
[0x0028]	*I2Cn_TX_CTRL1*	I^{2}C Transmit Control 1 Register
[0x002C]	*I2Cn_FIFO*	I^{2}C Transmit and Receive FIFO Register
[0x0030]	*I2Cn_MASTER_CTRL*	I^{2}C Master Control Register
[0x0034]	*I2Cn_CLK_LO*	I^{2}C Clock Low Time Register
[0x0038]	*I2Cn_CLK_HI*	I^{2}C Clock High Time Register
(0x003C)	*I2Cn_HS_CLK*	I^{2}C Hs-Mode Clock Control Register
[0x0040]	*I2Cn_TIMEOUT*	I^{2}C Timeout Register
[0x0048]	*I2Cn_DMA*	I^{2}C DMA Enable Register
[0x004C]	*I2Cn_SLAVE_ADDR*	I^{2}C Slave Address Register

12.5.1 Register Details

Table 12-6: I^{2}C Control Register

I2C Control			I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved for Future Use** Do not modify this field.
15	hs_mode	R/W	0	**Hs-Mode Enable** I2C high speed mode operation 0: Disable 1: Enable
14	-	RO	0	**Reserved for Future Use** Do not modify this field.
13	scl_pp_mode	R/W	0	**Single Master Only** When set to 1, the device MUST ONLY be used in a single master application with slave devices that are NOT going to hold SCL low (i.e. the slave devices will never clock stretch)
12	scl_clk_stretch_dis	R/W	0	**Slave Mode Clock Stretching** 0: Enabled 1: Disabled

I2C Control				I2Cn_CTRL	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**	
11	read	R	0	**Slave Read/Write Bit Status** Returns the logic level of the R/W bit on a received address match (*I2Cn_INT_FL0.addr_match* = 1) or general call match (*I2Cn_INT_FL0.gen_call_addr* = 1). This bit is valid three sys_clk clock cycles after the address match status flag is set.	
10	sw_out_en	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 hardware. 1: SDA and SCL are controller by firmware 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.sw_out_en=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.sw_out_en =1*	
5	-	RO	0	**Reserved for Future Use** Do not modify this field.	
4	rx_mode_ack	R/W	0	**Interactive Receive Mode (IRXM) Acknowledge** If IRXM is enabled (*I2Cn_CTRL.rx_mode* = 1), this field determines if the hardware sends an ACK or a NACK to an IRM transaction. 0: Respond to IRXM with ACK 1: Respond to IRXM with NACK	
3	rx_mode	R/W	0	**Interactive Receive Mode (IRXM)** When receiving data, allows for an Interactive Receive Mode (IRM) interrupt event after each received byte of data. The I2Cn peripheral hardware can be enabled to send either an ACK or NACK for IRXM. See *Interactive Receive Mode* section for detailed information. 0: Disable 1: Enable *Note: Only set this field when the I2C bus is inactive.*	
2	gen_call_addr	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	R/W	0	**Master Mode Enable** 0: Slave mode enabled. 1: Master mode enabled.	
0	i2c_en	R/W	0	**I2C Peripheral Enable** 0: Disabled 1: Enabled	

Table 12-7: I²C Status Register

I2C Status				I2Cn_STATUS	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:6	-	RO	0	**Reserved for Future Use** Do not modify this field.	
5	clk_mode	RO	0	**Master Mode I2C Bus Transaction Active** The peripheral is operating in Master mode and a valid transaction beginning with a START command is in progress on the I²C bus. This bit will read 1 until the master ends the transaction with a STOP command. This bit will continue to read 1 while a slave performs clock stretching. 0: Device not actively driving SCL clock cycles. 1: Device operating as master and actively driving SCL clock cycles.	
4	tx_full	RO	0	**TX FIFO Full** 0: Not full 1: Full	
3	tx_empty	RO	1	**TX FIFO Empty** 0: Not empty 1: Empty	
2	rx_full	RO	0	**RX FIFO Full** 0: Not full 1: Full	
1	rx_empty	RO	1	**RX FIFO Empty** 0: Not empty 1: Empty	
0	bus	RO	0	**Master or Slave Mode I2C Bus Transaction Active** The peripheral is operating in Master or Slave mode and a valid transaction beginning with a START command is in progress on the I2C bus. This bit will read 1 until the peripheral acting as a master or an external master ends the transaction with a STOP command. This bit will continue to read 1 while a slave performs clock stretching. 0: I²C bus is idle. 1: I²C bus transaction in progress.	

Table 12-8: I²C Interrupt Flag 0 Register

I2C Interrupt Flag 0				I2Cn_INT_FL0	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:24	-	RO	0	**Reserved for Future Use** Do not modify this field.	

I2C Interrupt Flag 0		I2Cn_INT_FL0		[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
23	wr_add_match	R/W1C	0	**Slave Write Address Match Interrupt Flag** If set, the device has been accessed for a write (i.e. receive) transaction in slave mode and the address received matches the device slave address. 0: No address match. 1: Address match.
22	rd_addr_match	R/W1C	0	**Slave Read Address Match Interrupt Flag** If set, the device has been accessed for a read (i.e. transmit) transaction in slave mode and the address received matches the device slave address. 0: No address match. 1: Address match.
21:16	-	RO	0	**Reserved for Future Use** Do not modify this field.
15	tx_lock_out	R/W1C	0	**TX FIFO Locked Interrupt Flag** If set, the TX FIFO is locked and writes to the TX FIFO are ignored. When set, the TX FIFO is automatically flushed. Writes to the TX FIFO are ignored until this flag is cleared. Write 1 to clear. 0: TX FIFO not locked. 1: TX FIFO is locked and all writes to the TX FIFO are ignored.
14	stop_er	R/W1C	0	**Out of Sequence STOP Interrupt Flag** This flag is set if a STOP condition occurs out of expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence STOP condition occurred.
13	start_er	R/W1C	0	**Out of Sequence START Interrupt Flag** This flag is set if a START condition occurs out of expected sequence. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: Out of sequence START condition occurred.
12	do_not_resp_er	R/W1C	0	**Slave Mode Do Not Respond Interrupt Flag** This occurs if an address match is made, but the TX FIFO or RX FIFO is not ready. Write 1 to clear this field. Writing 0 has no effect. 0: Error condition has not occurred. 1: I^2C address match has occurred and either the TX or RX FIFO is not configured.
11	data_er	R/W1C	0	**Master Mode Data NACK from External Slave Interrupt Flag** This flag is set by hardware if a NACK is received from a slave. This flag is only valid if the I2Cn peripheral is configured for Master Mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Data NACK received from a slave.
10	addr_nack_er	R/W1C	0	**Master Mode Address NACK from Slave Error Flag** This flag is set by hardware if an Address NACK is received from a slave bus. This flag is only valid if the I2Cn peripheral is configured for Master Mode operation. Write 1 to clear. Write 0 has no effect. 0: Error condition has not occurred. 1: Address NACK received from a slave.

I2C Interrupt Flag 0		I2Cn_INT_FL0		[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
9	to_er	R/ W1C	0	**Timeout Error Interrupt Flag** This occurs when this device holds SCL low longer than the programmed timeout value. Applies to both Master and Slave Mode. Write 1 to clear. Write 0 has no effect. 0: Timeout error has not occurred. 1: Timeout error occurred.
8	arb_er	R/ W1C	0	**Master Mode Arbitration Lost Interrupt Flag** Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: Condition occurred.
7	addr_ack	R/ W1C	0	**Master Mode Address ACK from External Slave Interrupt Flag** This field is set when a slave address ACK is received. Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: The slave device ACK for the address was received.
6	stop	R/ W1C	0	**Slave Mode STOP Condition Interrupt Flag** This flag is set by hardware when a STOP condition is detected. Write 1 to clear. Write 0 has no effect. 0: Condition has not occurred. 1: Condition occurred.
5	tx_thresh	RO	1	**TX FIFO Threshold Level Interrupt Flag** This field is set by hardware if the number of bytes in the Transmit FIFO is less than or equal to the Transmit FIFO threshold level. Write 1 to clear. This field is automatically cleared by hardware when the TX FIFO contains fewer bytes than the TX threshold level. 0: TX FIFO contains more bytes than the TX threshold level. 1: TX FIFO contains TX threshold level or fewer bytes (Default).
4	rx_thresh	R/W1C	1	**RX FIFO Threshold Level Interrupt Flag** This field is set by hardware if the number of bytes in the Receive FIFO is greater than or equal to the Receive FIFO threshold level. This field is automatically cleared when the RX FIFO contains fewer bytes than the RX threshold setting. 0: RX FIFO contains fewer bytes than the RX threshold level. 1: RX FIFO contains at least RX threshold level of bytes (Default).
3	addr_match	R/W1C	0	**Slave Mode Incoming Address Match Status Interrupt Flag** Write 1 to clear. Writing 0 has no effect. 0: Slave address match has not occurred. 1: Slave address match occurred.
2	gen_call_addr	R/W1C	0	**Slave Mode General Call Address Match Received Interrupt Flag** Write 1 to clear. Writing 0 has no effect. 0: General call address match has not occurred. 1: General call address match occurred.
1	rx_mode	R/W1C	0	**Interactive Receive Mode Interrupt Flag** Write 1 to clear. Writing 0 is ignored. 0: Interrupt condition has not occurred. 1: Interrupt condition occurred.

I2C Interrupt Flag 0				I2Cn_INT_FL0 [0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
0	done	R/W1C	0	**Transfer Complete Interrupt Flag** This flag is set for both Master and Slave mode once a transaction completes. Write 1 to clear. Writing 0 has no effect. 0: Transfer is not complete. 1: Transfer complete.

Table 12-9: I²C Interrupt Enable 0 Register

I2C Interrupt Enable 0				I2Cn_INT_EN0 [0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:24	-	RO	0	**Reserved for Future Use** Do not modify this field.
23	wr_addr_match	R/W	0	**Slave Write Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in slave mode and the address received matches the device slave addressed for a write transaction. 0: Disabled. 1: Enabled.
22	rd_addr_match	R/W	0	**Slave Read Address Match Interrupt Enable** This bit is set to enable interrupts when the device is accessed in slave mode and the address received matches the device slave addressed for a read transaction. 0: Disabled. 1: Enabled.
21:16	-	RO	0	**Reserved for Future Use** Do not modify this field.
15	tx_lock_out	R/W	0	**TX FIFO Lock Out Interrupt Enable** 0: Disabled. 1: Enabled.
14	stop_er	R/W	0	**Out of Sequence STOP Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
13	start_er	R/W	0	**Out of Sequence START Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.
12	do_not_resp_er	R/W	0	**Slave Mode Do Not Respond Interrupt Enable** Set this field to enable interrupts in Slave Mode when the “Do Not Respond” condition occurs. 0: Interrupt disabled. 1: Interrupt enabled.
11	data_er	R/W	0	**Master Mode Received Data NACK from Slave Interrupt Enable** 0: Disabled. 1: Enabled.
10	addr_nack_er	R/W	0	**Master Mode Received Address NACK from Slave Interrupt Enable** 0: Disabled. 1: Enabled.

I2C Interrupt Enable 0				I2Cn_INT_EN0	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**	
9	to_er	R/W	0	**Timeout Error Interrupt Enable** 0: Disabled. 1: Enabled.	
8	arb_er	R/W	0	**Master Mode Arbitration Lost Interrupt Enable** 0: Disabled. 1: Enabled.	
7	addr_ack	R/W	0	**Received Address ACK from Slave Interrupt Enable** Set this field to enable interrupts for Master Mode slave device address ACK events. 0: Interrupt disabled. 1: Interrupt enabled.	
6	stop	R/W	0	**STOP Condition Detected Interrupt Enable** 0: Disabled. 1: Enabled.	
5	tx_thresh	R/W	0	**TX FIFO Threshold Level Interrupt Enable** 0: Disabled. 1: Enabled.	
4	rx_thresh	R/W	0	**RX FIFO Threshold Level Interrupt Enable** 0: Disabled. 1: Enabled.	
3	addr_match	R/W	0	**Slave Mode Incoming Address Match Interrupt Enable** 0: Disabled. 1: Enabled.	
2	gen_call_addr	R/W	0	**Slave Mode General Call Address Match Received Interrupt Enable** 0: Disabled. 1: Enabled.	
1	rx_mode	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 12-10: I²C Interrupt Flag 1 Register

I2C Interrupt Status Flags 1				I2Cn_INT_FL1	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:3	-	RO	0	**Reserved for Future Use** Do not modify this field.	
2	start	R/W1C	0	**START Condition Status Flag** If set, a device START condition has been detected. 0: START condition not detected. 1: START condition detected.	

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

Table 12-11: I²C Interrupt Enable 1 Register

I2C Interrupt Enable 1			I2Cn_INT_EN1	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:3	-	RO	0	**Reserved for Future Use** Do not modify this field.
2	start	R/W	0	**START Condition Interrupt Enable** 0: Disabled. 1: Enabled.
1	tx_underflow	R/W	0	**Slave Mode TX FIFO Underflow Interrupt Enable** 0: Disabled. 1: Enabled.
0	rx_underflow	R/W	0	**Slave Mode RX FIFO Overflow Interrupt Enable** 0: Disabled. 1: Enabled.

Table 12-12: I²C FIFO Length Register

I2C FIFO Length			I2Cn_FIFO_LEN	[0x0018]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	RO	0	**Reserved for Future Use** Do not modify this field.
15:8	tx_len	RO	8	**TX FIFO Length** Returns the length of the TX FIFO. 8: 8-byte TX FIFO.
7:0	rx_len	RO	8	**RX FIFO Length** Returns the length of the RX FIFO. 8: 8-byte RX FIFO.

Table 12-13: I²C Receive Control 0 Register

I2C Receive Control 0				I2Cn_RX_CTRL0 [0x001C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved for Future Use** Do not modify this field.
11:8	rx_thresh	R/W	0	**RX FIFO Threshold Level** Set this field to the required number of bytes to trigger a RX FIFO threshold event. When the number of bytes in the RX FIFO is equal to or greater than this field, the hardware sets the *I2Cn_INT_FL0.rx_thresh* bit indicating an RX FIFO threshold level event. 0: 0 bytes or more in the RX FIFO causes a threshold event. 1: 1+ bytes in the RX FIFO triggers a receive threshold event (recommended minimum value). … 8: RX FIFO threshold event only occurs when the RX FIFO is full.
7	rx_flush	R/W1O	0	**Flush RX FIFO** Write 1 to this field to initiate a RX FIFO flush, clearing all data in the RX FIFO. This field is automatically cleared by hardware when the RX FIFO flush completes. Writing 0 has no effect. 0: RX FIFO flush complete or not active. 1: Flush the RX FIFO
6:1	-	RO	0	**Reserved for Future Use** Do not modify this field.
0	dnr	R/W	0	**Slave Mode Do Not Respond** Slave mode operation only. If the device has been addressed for a write operation, and there is still data in the RX_FIFO then: 0: Always respond to an address match with an ACK but then always respond to data bytes with a NACK. 1: NACK the address.

Table 12-14: I²C Receive Control 1 Register

I2C Receive Control 1				I2Cn_RX_CTRL1 [0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved for Future Use** Do not modify this field.
11:8	rx_fifo	R	0	**RX FIFO Byte Count Status** 0: No data in the RX FIFO. 1: 1 byte in the RX FIFO. 2: 2 bytes in the RX FIFO. 3: 3 bytes in the RX FIFO. 4: 4 bytes in the RX FIFO. 5: 5 bytes in the RX FIFO. 6: 6 bytes in the RX FIFO. 7: 7 bytes in the RX FIFO. 8: 8 bytes in the RX FIFO (max value).

I2C Receive Control 1			I2Cn_RX_CTRL1	[0x0020]
Bits	**Field**	**Access**	**Reset**	**Description**
7:0	rx_cnt	R/W	1	**RX FIFO Transaction Byte Count Configuration** When in Master Mode, write the number of bytes to be received in a transaction from 1 to 256. 0x00 represents 256. 0: 256 byte receive transaction. 1: 1 byte receive transaction. 2: 2 byte receive transaction. … 255: 255 byte receive transaction. *This field is ignored when I2Cn_CTRL.rx_mode = 1. To receive more than 256 bytes, use I2Cn_CTRL.rx_mode = 1*

Table 12-15: I²C Transmit Control 0 Register

I2C Transmit Control 0			I2Cn_TX_CTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**
31:12	-	RO	0	**Reserved**
11:8	*tx_thresh*	R/W	0	**TX FIFO Threshold Level** Sets the level for a Transmit FIFO threshold event interrupt. If the number of bytes remaining in the TX FIFO falls to this level or lower the interrupt flag *I2Cn_INT_FL0.tx_thresh* is set indicating a TX FIFO Threshold Event occurred. 0: 0 bytes remaining in the TX FIFO triggers a TX FIFO threshold event. 1: 1 byte or fewer remaining in the TX FIFO triggers a TX FIFO threshold event (recommended minimum value). … 7: 7 or fewer bytes remaining in the TX FIFO triggers a TX FIFO threshold event
7	tx_flush	R/W1O	0	**TX FIFO Flush** A TX FIFO flush clears all remaining data from the transmit FIFO. 0: TX FIFO flush is complete or not active. 1: Flush the TX FIFO *Note: Hardware automatically clears this bit to 0 after it is written to 1 when the flush is completed.* If *I2Cn_INT_FL0.tx_lock_out* = 1, then *I2Cn_TX_CTRL0.tx_flush* = 1.
6	-	RO	0	**Reserved for Future Use** Do not modify this field.
5	tx_nack_afd	R/W	0	**TX FIFO received NACK Auto Flush Disable** Various situations or conditions are described in this user guide that lead to the Transmit FIFO being flushed and locked out (*I2Cn_INT_FL0.tx_lock_out* = 1). 0: Received NACK at end of Slave Transmit operation enabled 1: Received NACK at end of Slave Transmit operation disabled. *Note: upon entering TX Preload Mode, hardware will automatically set this bit to 0 Software can subsequently set to any value desired (i.e. Hardware does not continuously force the bitfield to this value).*

I2C Transmit Control 0				I2Cn_TX_CTRL0	[0x0024]
Bits	**Field**	**Access**	**Reset**	**Description**	
4	tx_amr_afd	R/W	0	**TX FIFO Slave Address Match Read Auto Flush Disable** Various situations or conditions are described in this user guide that lead to the Transmit FIFO being flushed and locked out (*I2Cn_INT_FL0.tx_lock_out* = 1). 0: Enabled. 1: Disabled. *Note: upon entering TX Preload Mode, hardware will automatically set this bit to 1 Software can subsequently set to any value desired (i.e. Hardware does not continuously force the bitfield to this value).*	
3	tx_amw_afd	R/W	0	**TX FIFO Slave Address Match Write Auto Flush Disable** Various situations or conditions are described in this user guide that lead to the Transmit FIFO being flushed and locked out (*I2Cn_INT_FL0.tx_lock_out* = 1). 0: Enabled 1: Disabled. *Note: upon entering TX Preload Mode, hardware will automatically set this bit to 1 Software can subsequently set to any value desired (i.e. Hardware does not continuously force the bitfield to this value).*	
2	tx_amgc_afd	R/W	0	**TX FIFO General Call Address Match Auto Flush Disable** Various situations or conditions are described in this user guide that lead to the Transmit FIFO being flushed and locked out (*I2Cn_INT_FL0.tx_lock_out* = 1). 0: Enabled. 1: Disabled. *Note: upon entering TX Preload Mode, hardware will automatically set this bit to 1 Software can subsequently set to any value desired (i.e. Hardware does not continuously force the bitfield to this value).*	
1	tx_ready_mode	R/W	0	**TX FIFO Ready Manual Mode** 0: Hardware will control *I2Cn_TX_CTRL1.tx_ready* 1: Software control of *I2Cn_TX_CTRL1.tx_ready*	
0	tx_preload	R/W	0	**TX FIFO Preload Mode Enable** 0: Normal operation. An address match in Slave Mode, or a General Call address match, will flush and lock the TX FIFO so it cannot be written and set *I2Cn_INT_FL0.tx_lock_out.* 1: TX FIFO Preload Mode. An address match in Slave Mode, or a General Call address match, will not lock the TX FIFO and will not set *I2Cn_INT_FL0.tx_lock_out*. This allows firmware to preload data into the TX FIFO. The status of the I2C is controllable at *I2Cn_TX_CTRL1.tx_ready.*	

Table 12-16: I²C Transmit Control 1 Register

I2C Transmit Control Register 1				I2Cn_TX_CTRL1	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:12	-	RO	0	**Reserved for Future Use** Do not modify this field.	

I2C Transmit Control Register 1			I2Cn_TX_CTRL1	[0x0028]
Bits	**Field**	**Access**	**Reset**	**Description**
11:8	tx_fifo	R	0	**Transmit FIFO Byte Count Status** 0: No data in the TX FIFO. 1: 1 byte in the TX FIFO. 2: 2 bytes in the TX FIFO. 3: 3 bytes in the TX FIFO. 4: 4 bytes in the TX FIFO. 5: 5 bytes in the TX FIFO. 6: 6 bytes in the TX FIFO. 7: 7 bytes in the TX FIFO. 8: 8 bytes in the TX FIFO (max value).
7:1	-	RO	0	**Reserved for Future Use** Do not modify this field.
0	tx_ready	R/1	1	**Transmit FIFO Preload Ready Status** When TX FIFO Preload Mode is enabled, *I2Cn_TX_CTRL0.tx_preload* = 1, this bit is automatically cleared to 0. While this bit is 0, if the I2Cn hardware receives a slave address match a NACK is sent. Once the I2Cn hardware is ready (firmware has preloaded the TX FIFO, configured the DMA, etc.) application firmware must set this bit to 1 so the I2Cn hardware will send an ACK on a slave address match. When TX FIFO Preload Mode is disabled, *I2Cn_TX_CTRL0.tx_preload* = 1, this bit is forced to 1 and the I2Cn hardware behaves normally.

Table 12-17: I²C Data Register

I2C Data			I2Cn_FIFO	[0x002C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved for Future Use** Do not modify this field.
7:0	data	R/W	0xFF	**I2C FIFO Data Register** Reads from this register pops data off the RX FIFO. Writes to this register pushes data onto the TX FIFO. Reading from an empty RX FIFO returns 0xFF. Writes to a full TX FIFO are ignored.

Table 12-18: I²C Master Control Register

I2C Master Control			I2Cn_MASTER_CTRL	[0x0030]
Bits	**Field**	**Access**	**Reset**	**Description**
31:11	-	RO	0	**Reserved for Future Use** Do not modify this field.
10:8	mcode	R/W	0	**MCODE** These bits set the master code used in Hs-mode operation
7	sl_ex_addr	R/W	0	**Slave Extended Addressing Enable** 0: Send a 7-bit address to the slave 1: Send a 10-bit address to the slave
6:3	-	RO	0	**Reserved for Future Use** Do not modify this field.

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

Table 12-19: I²C SCL Low Control Register

I2C Clock Low Control			I2Cn_CLK_LO	[0x0034]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved for Future Use** Do not modify this field.
8:0	scl_lo	R/W	0x001	**Clock Low Time** In Master Mode, this configures the SCL low time. $t_{SCL\_LO} = f_{I2C\_CLK} \times (scl\_lo + 1)$ *Note: 0 is not a valid setting for this field.*

Table 12-20: I²C SCL High Control Register

I2C Clock High Control			I2Cn_CLK_HI	[0x0038]
Bits	**Field**	**Access**	**Reset**	**Description**
31:9	-	RO	0	**Reserved for Future Use** Do not modify this field.
8:0	scl_hi	R/W	0x001	**Clock High Time** In Master Mode, this configures the SCL high time. $t_{SCL\_HI} = {}^{1}/_{f_{I2C\_CLK}} \times (scl\_hi + 1)$ In both Master and Slave Mode, this also configures the time SCL is held low after new data is loaded from the TX FIFO or after firmware clears *I2Cn_INT_FL0.rx_mode* during Interactive Receive Mode. *Note: 0 is not a valid setting for this field.*

Table 12-21: I²C Hs-Mode Clock Control Register

I2C Hs-Mode Clock Control		I2Cn_HS_CLK		[0x003C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field.
15:8	hs_clk_hi	R/W	0	**Hs-Mode Clock High Time** This field sets the Hs-Mode clock high count. In Slave mode, this is the time SCL is held high after data is output on SDA. *Note: See section 12.4.4 SCL Clock Generation for Hs-mode for details on the requirements for the Hs-mode clock high and low times.*
7:0	hs_clk_lo	R/W	0	**Hs-Mode Clock Low Time** This field sets the Hs-Mode clock low count. In Slave mode, this is the time SCL is held low after data is output on SDA. *Note: See section 12.4.4 SCL Clock Generation for Hs-mode for details on the requirements for the Hs-mode clock high and low times.*

Table 12-22: I²C Timeout Register

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

Table 12-23: I²C DMA Register

I2C DMA		I2Cn_DMA		[0x0048]
Bits	**Field**	**Access**	**Reset**	**Description**
31:2	-	RO	0	**Reserved for Future Use** Do not modify this field.
1	rx_en	R/W	0	**RX DMA Channel Enable** 0: Disable 1: Enable
0	tx_en	R/W	0	**TX DMA Channel Enable** 0: Disable 1: Enable

Table 12-24: I²C Slave Address Register

I2C Slave Address				I2Cn_SLAVE_ADDR	[0x004C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:16	-	RO	0	**Reserved for Future Use** Do not modify this field.	
15	ex_addr	R/W	0	**Slave Mode Extended Address Length Select** 0: 7-bit addressing. 1: 10-bit addressing.	
14:10	-	RO	0	**Reserved for Future Use** Do not modify this field.	
9:0	slave_addr	R/W	0	**Slave Mode Slave Address** In Slave Mode Operation, (*I2Cn_CTRL.mstr* = 0), set this field to the slave address for the I2Cn port. For 7-bit addressing, the address occupies the least significant 7 bits. For 10-bit addressing, the 9-bits of address occupies the most significant 9 bit and the R/W bit occupies the least significant bit. *Note: I2Cn_SLAVE_ADDR.ex_addr controls if this field is a 7-bit or 10-bit address.*	

13. Inter-Integrated Sound Interface (I²S)

I²S (Inter-IC Sound) is a serial audio interface for communicating PCM data information between devices.

Both master and slave modes are supported. The peripheral provides separate data in and data out signals supporting full-duplex communication for higher throughput.

Key features:

- Supports both master and slave modes
- Stereophonic (2 channel) and monophonic (left or right channel option) formats
- Each channel supports full duplex operations
- Two DMA requests for each I²S channel can be connected (one to read FIFO, one to write FIFO)
- Flexible timing
  - Configurable sampling rate from $^{1}/_{65536}$ to 1 of the I²S input clock
  - BCLK clock source selection in Master and Slave mode
  - LRCLK clock source selection in Master and Slave mode
- Flexible data format
  - Programmable word size (byte, half word and word) can be saved to memory
  - Number of bits per data word can be selected from 1 to 32 (typically 8, 16, 24, 32-bit data frame)
  - Word select polarity control, not in I²S Specification
  - First bit position selection, not in I²S Specification
  - MSB/LSB aligned, not in I²S Specification
  - Sign extended control, not in I²S Specification
- Full-duplex serial communication with separate I2S Data In (SDI) and I2S Data Out (SDO) pings.

13.1 Instances

Table 13-1. MAX78000 I²S Instances

1) Instance	2) Supported Channels	3) I2S_CLK Clock Options		4) SDI	5) SDO	6) RX/TX FIFO Depth
7) I2S0	8) Stereo	9) PCLK	10) IS2_CLKEXT (P0.14 AF2)	11)	12)	13) 8/8

13.2 Details

The I²S supports full-duplex serial communication with separate I²S data in (SDI) and data out (SDO) pins. *Figure 13-1: Full Duplex Connection, I2S Master Mode* shows an interconnect between a peripheral configured in host mode, communicating with an external I²S slave receiver and an external I²S transmitter. In master mode the BCLK and LRCLK signals are generated by the peripheral and sent to each slave device.

Figure 13-1: Full Duplex Connection, I²S Master Mode

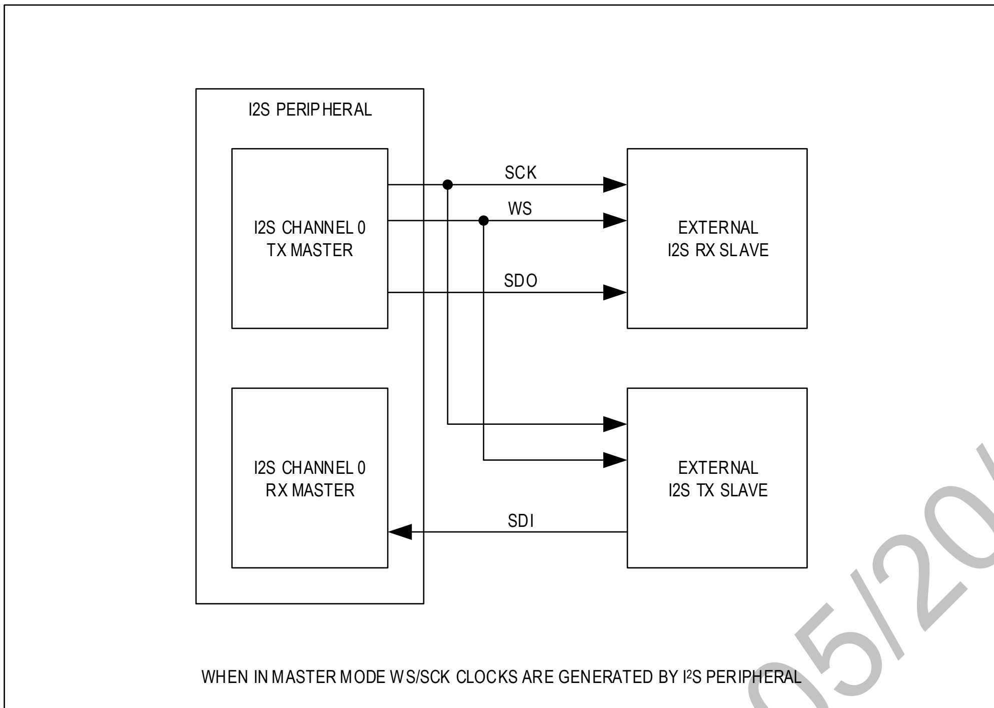
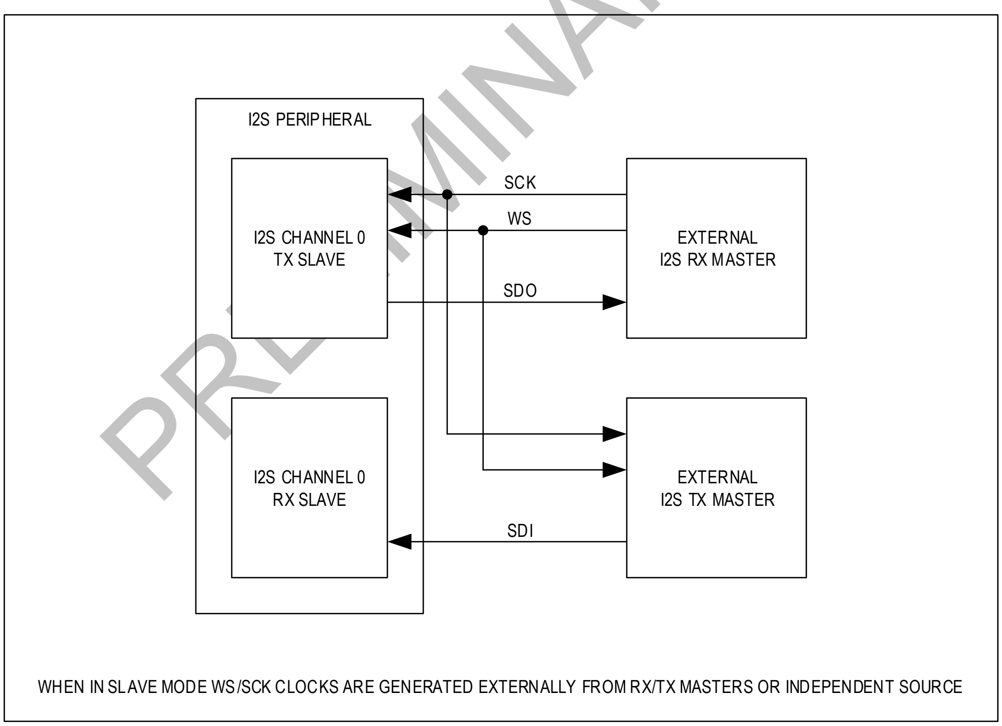


Figure 13-2: Full Duplex Connection, I2S Slave Mode shows a peripheral configured in slave mode. An external I²S system clock, from one of the external I²S master devices, or an independent source, provides BCLK and LRCLK signals to the I²S peripheral.

Figure 13-2: Full Duplex Connection, I²S Slave Mode



13.3 I²S Clocking

I²S communication is synchronized by two clocks: the bit clock (BCLK, also known as SCK) and the left/right channel clock (LRCLK, also known as WS.) In master mode the internal timing generator creates the BCLK and LRCLK signals. The pins are configured as outputs to supply the clock to slave devices.

In slave mode the BCLK and LRCLK pins are configured as inputs. An external master generates the BCLK and LRCLK signals which the peripheral uses to synchronize itself to the I²S bus.

A typical bit clock rate is between 512kHz for 8kHz sampling, and 12.288MHz for 192kHz sampling rate. Equation 13-1 shows the bit clock rate for CD stereo audio using a sample frequency of 44.1kHz and 16-bits per sample.

Equation 13-1: CD Audio Bit Frequency Calculation

$$f_{BCLK} = 44.1kHz \times 16 \times 2 \implies 1.4122MHz$$

Clock switching must be done while the I²S clock is disabled.

13.3.1 BCLK Generation for Master

The required BCLK frequency is determined using:

1. Audio sample frequency
2. Number of bits per sample
3. Number of channels (2 for stereo)

Equation 13-2, *below*, shows the formula to determine the target BCLK frequency.

Equation 13-2: BCLK Target Frequency based on Audio Frequency, Bits per Word and Channels

$$f_{BCLK} = f_{SAMPLE} \times \left({}^{Bits}/_{Sample} \right) \times \text{Number of Channels}$$

Calculate BCLK frequency using *Equation 13-3* using the selected I2S_CLK from *Table 13-1*.

Equation 13-3: BLCK Frequency Equation for the I²S Peripheral

$$f_{BCLK} = \frac{f_{I2S\_CLK}}{(I2Sn\_CTRL1CH0.clkdiv + 1)}$$

13.3.2 LRCLK Generation

An I²S data stream can carry monophonic (either left or right channel) or stereophonic (one left and one right channel) data. The audio sample rate is determined by the LRCLK frequency.

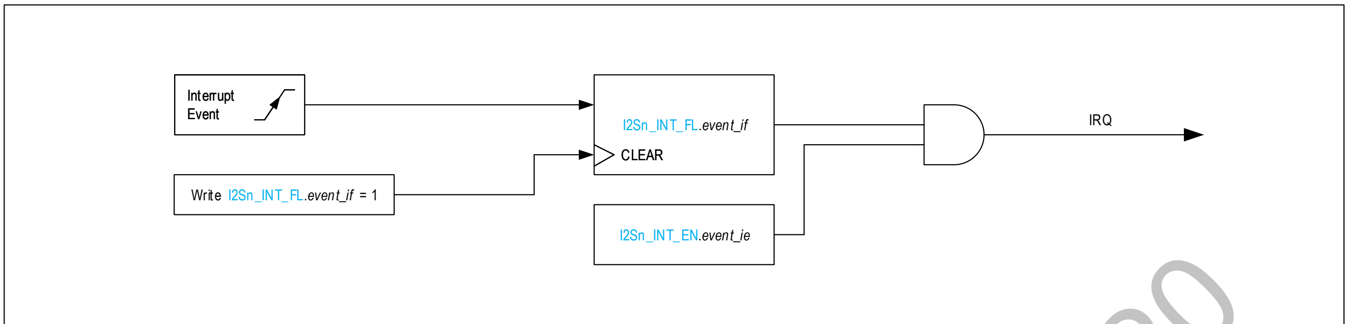
LRCLK selects the right or left data channel. The polarity of the LRCLK is programmable. Left channel data is transferred when LRCLK is low, and right channel data transferred when LRCLK is high.

$$f_{LRCLK} = \frac{f_{BCLK}}{(1 + bits\_word)}$$

13.4 Interrupt Events

The peripheral generates interrupts for the events shown in *Table 13-2. MAX78000 I2S Interrupt Events*. Unless noted otherwise, each instance has its own independent set of interrupts flag and enable fields. *Figure 13-3. MAX78000 I2S Interrupt Functional Diagram* shows how the interrupts work.

Figure 13-3. MAX78000 I²S Interrupt Functional Diagram



Unless stated otherwise hardware sets interrupt flags on the occurrence of on the event ("edge-triggered"), rather than the condition ("level-triggered"). Multiple events may set the same flag. An interrupt request (IRQ) will be generated if the corresponding interrupt enable field is set. The interrupt flags remain set until cleared by software, usually in their respective interrupt service routine (ISR).

Table 13-2. MAX78000 I²S Interrupt Events (Channel 0 Example)

EVENT	LOCAL INTERRUPT FLAG	LOCAL INTERRUPT ENABLE
RX Overrun	*I2Sn_INTFL.rx_ov_ch0*	*I2Sn_INTEN.rx_ov_ch0*
RX Threshold	*I2Sn_INTFL.rx_thd_ch0*	*I2Sn_INTEN.rx_thd_ch0*
TX FIFO Half-Empty	*I2Sn_INTFL.tx_he_ch0*	*I2Sn_INTEN.tx_he_ch0*

13.4.1 RX FIFO Overrun

An RX overrun event occurs if the number of data words in a FIFO (*.rx_lvl*) is equal to 8 and a valid character has been shifted in. The received character is discarded and the FIFO remains unchanged.

13.4.2 RX FIFO Threshold

An RX FIFO threshold event occurs when a valid character is shifted in and the number of data words in an RX FIFO (*.rx_lvl*) increments from *.rx_thd* -1 to. *rx_thd*. The event does not occur if the opposite transition occurs.

13.4.3 TX FIFO Half-Empty (Fixed)

A TX FIFO half-empty event occurs when the number of bytes in the TX FIFO (*.tx_lvl*) transitions from C_RX_FIFO DEPTH/2 to C_RX_FIFO DEPTH//2 +1.

13.5 Wakeup Events

The peripheral does not support wakeup events from low power modes.

13.6 Direct Memory Access

The I²S supports DMA transfers in both the transmit and receive directions; separate DMA channels can be connected to the RX and TX FIFO buffers. The followings describe the behavior of the RX and TX DMA requests when the DMA function is enabled.

- RX DMA request is asserted when the number of valid bytes in the RX FIFO is greater than or equal to the RX FIFO threshold.
- TX DMA request is asserted when the number of valid bytes in the TX FIFO is less than the TX FIFO threshold.
- The IP support DMA transfers in both the transmit and receive directions; separate DMA channels can be connected to the RX and TX FIFO buffers. The interrupts are not rerouted to the DMA controller and must still be handled by the CPU in the usual manner.

13.7 Block Operation

After exiting a power-on reset, the IP is disabled by default. It must be enabled and configured by software to establish the I²S serial communication. A typical software sequence is shown below.

- Clear *PERCKCN1.i2sd* to 0 to enable the I²S clock source shown in *Table 13-1. MAX78000 I2S Instances*.
- Wait for all transmissions and receptions to complete based on the customer-implemented protocol.
- Disable the peripheral clock.
- Set *I2Sn_CTRL0CH0.rst* to 1 to clear the remaining value in the counter.
- Set *I2Sn_CTRL1CH0.flush* to 1 to flush the FIFO buffers.
- Configure the *I2Sn_CTRL0CH0.ch_mode* to select master or slave configuration.
- If in master mode configure the baud rate by programming the *I2Sn_CTRL1CH0* register.
- Configure the threshold of the RX FIFO by programming the *I2Sn_CTRL1CH0.rx_thd*. The threshold of the TX FIFO is a fixed value, which is half of the TX FIFO depth.
- Enable DMA function by programming *I2Sn_DMACH0* register if DMA function is needed.
- Enable interrupt function by programming *I2Sn_INTEN* register if interrupt function is needed.
- Program the *clkdiv* bits in *I2Sn_CTRL1CH0* register for the new baudrate.
- Re-enable the baud clock by setting *I2Sn_CTRL1CH0.en*.
- Start I²S operations.

13.8 I²S Master/Slave Configuration

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

Table 13-3: I²S Channel mode Configuration (Channel 0 Example)

Mode	*I2Sn_CTRL0CH0.ch_mode*	*I2Sn_CTRL0CH0.ext_sel*	LRCLK	BCLK
Master	00	x	Output to Slave	Input from Slave
Slave	11	0	Output to Master	Input from Master

13.9 Data Formatting

Data is signed two's complement, with controllable data direction allowing users to select MSB first or LSB firs. The data frame can have arbitrary number of bits per channel.

The audio data word is programmable from 1 to 32 bits long (typically 8, 16, 24, and 32 bits). The transfer size, I²S data frame can be from 1 to 32 bits, is set by writing to the *I2Sn_CTRL0CH0.wsize* register. For word sizes less than 32 bits, the data frame may still be 64 bits and the unused bits are driven low by the transmitting device.

TX data is written to the SDO pin on the falling edge of BCLK, and RX data is read from the SDI pin on the rising edge of BCLK. The most significant bit (MSB) is always transmitted first.

13.9.1 Word Select Polarity

The Word Select polarity can be configured by the *I2Sn_CTRL0CH0.ws_pol* register. By default, LRCLK low is for the left channel, high is for the right channel. Setting the register bit will invert the LRCLK signal definition.

13.9.2 MSB Location Control

By the default the first bit of the transmit data is located at the second BCLK cycle as required by the I2SBUS specification. Optionally the MSB of the transmit data can be configured to be at the first BCLK cycle of the data word via *I2Sn_CTRL0CH0*.*msb_loc*. The waveform shows the transmit data with the register bit set.

13.9.3 Data Alignment

The I^{2}S data can be left aligned (aligned to the MSB) or right aligned (aligned to the LSB) via the *I2Sn_CTRL0CH0*.*align* field.

Left aligned: *I2Sn_CTRL0CH0*.*align* = 0

- Number of data bits (BCLK pulses) per half-frame is higher than the FIFO data width
- RX: All bits after LSB of FIFO data width will be discarded
- TX: All bits after LSB of FIFO data width will be sent as 0
- Number of data bits (BCLK pulses) per half-frame is lower than the FIFO data width
- RX & TX: Lower bits of TX and RX FIFO will be removed first

Right aligned: *I2Sn_CTRL0CH0*.*align* = 1

- Number of data bits (samples) per half-frame is higher than the FIFO data width
- RX: All bits before MSB of the sample value will be discarded
- TX: All bits before MSB of the sample value will be 0
- Number of data bits (samples) per half-frame is lower than the FIFO data width
- The data will be sign extended and saved to FIFO
- Lower bits of TX FIFO will be removed first, same as left aligned case

13.9.4 FIFO Word Control

The data width of the TX/RX FIFO can be configured to be byte, half word, or word size by using the *I2Sn_CTRL0CH0*.*wsize* field. The data always begins with left channel first from the lowest address. The tables below describe the data ordering based on the *I2Sn_CTRL0CH0*.*wsize* setting.

When a channel is reconfigured for a new setup, the FIFOs must be flushed and the channel is reset by setting the I2S_CHMODEn.RST register.

Table 13-4: Byte Ordering when I2Sn_CTRL0CH0.wsize = 0 (Byte

Address	MSByte			LSByte
Low Address	Right Channel Byte 1	Left Channel Byte 1	Right Channel Byte 0	Left Channel Byte 0
Low Address + 4	Right Channel Byte 3	Left Channel Byte 3	Right Channel Byte 2	Left Channel Byte 2
	…	…	…	…
High Address + (n+1)	Right Channel Byte *n*	Left Channel Byte *n*	Right Channel Byte *n-1*	Left Channel Byte *n-1*

Table 13-5: Half-Word Ordering when I2Sn_CTRL0CH0.wsize = 1

Address	MSByte	LSByte
Low Address	Right Channel Half-Word 0	Left Channel Half Word 0
	Right Channel Half-Word 1	Left Channel Half Word 1
	…	…
High Address	Right Channel Half Word *n*	Left Channel Half Word *n*

Table 13-6: Word Ordering when I2Sn_CTRL0CH0.wsize = 2 or 3

Address	Data Word
Low Address	Left Channel Word 0
	Right Channel Word 0
	Left Channel Word 1
	Right Channel Word 1
	…
High Address	

13.9.5 Samples Data

Table 13-7: Number of BCLK per Half Frame and Sample Number shows the relationship between the Number of BCLKs per half frame and the number of samples per half frame. Equation 13-4 shows the required relationship between the Number of Samples and Number of BCLK per Half Frame.

Equation 13-4: Number of Samples Relationship to Number of BCLK

$$\frac{\#\ Samples}{Half\ Frame} \le \frac{\#\ BCLK}{Half\ Frame}$$

The *I2Sn_CTRL1CH0.bits_word* column in *Table 13-7*, *below*, is determined by $\frac{\#\ BCLK}{Half\ Frame} - 1$. The *I2Sn_CTRL1CH0.smp_size* column is the sample number captured from the I^{2}S bus and is determined by $\frac{\#\ Samples}{Half\ Frame} - 1$.

Table 13-7: Number of BCLK per Half Frame and Sample Number

	$\frac{\#\ BCLK}{Half\ Frame}$	$\frac{\#\ Samples}{Half\ Frame}$	FIFO Depth	I2Sn_CTRL1CH0			Sign extend when I2Sn_CTRL0CH0.align=1 & smp_size < FIFO Depth
				bits_word	smp_size	wsize	
8-bit / 16	8	8	8	7	7	0	
8-bit / 32	16	8	8	15	7	0	
8-bit / 40	20	8	8	19	7	0	
8-bit / 48	24	8	8	23	7	0	
8-bit / 64	32	8	8	31	7	0	

	# BCLK / Half Frame	# Samples / Half Frame	FIFO Depth	I2Sn_CTRL1CH0			Sign extend when I2Sn_CTRL0CH0.align=1 & smp_size < FIFO Depth
				bits_word	smp_size	wsize	
16-bit / 32	16	16	16	15	15	1	
16-bit / 40	20	16	16	19	15	1	
16-bit / 48	24	16	16	23	15	1	
16-bit / 64	32	16	16	31	15	1	
20-bit / 16	8	8	8	7	7	0	
20-bit / 32	16	16	16	15	15	1	
20-bit / 40	20	20	32	19	19	2	sign
20-bit / 48	24	20	32	23	19	2	sign
20-bit / 64	32	20	32	31	19	2	sign
24-bit / 16	8	8	8	7	7	0	
24-bit / 32	16	16	16	15	15	1	
24-bit / 40	20	20	32	19	19	2	sign
24-bit / 48	24	24	32	23	23	2	sign
24-bit / 64	32	24	32	31	23	2	sign
32-bit / 16	8	8	8	7	7	0	
32-bit / 32	16	16	16	15	15	1	
32-bit / 40	20	20	32	19	19	2	sign
32-bit /48	24	24	32	23	23	2	sign
32-bit / 64	32	32	32	31	31	2	

13.10 Stereo/Mono Configuration

The I²S can transfer stereo or monophonic data based on the *I2Sn_CTRL0CH0.stereo* field. In stereo mode, both left and right channels contain data, which is the default setting. In mono mode, only the left or right channel contains data and the channel is selected using the *I2Sn_CTRL0CH0.stereo* field settings of 2 or 3.

13.11 FIFOs

Data to be transferred must be written to the TX FIFO by writing to the *I2Sn_FIFOCH0.data* register (either directly or using a DMA channel). This data will be transmitted out by the hardware automatically a character (word/half word/byte) at a time, in the order that it was received. The status flags and associated I2S interrupts can be used to monitor the FIFO status and determine when the transfer cycle or cycles have completed.

Received data is loaded into the RX FIFO, and it can then be unloaded by reading from the *I2Sn_FIFOCH0.data* register. An overrun event occurs if the RX FIFO is full and another character is received.

In the case of TX FIFO empty, FIFO read point will increase when the read the last non-empty FIFO data and point to the first nonvalid data and keep sending that data till new data is written. For the exception case of FIFO over run or under run, we suggest to reset the channel and reconfigure.

Data is written into the TX FIFO to start the I²S transmission. The I²S data is continuously sent after the TX is enabled and the I²S clock divider is ready. The I²S will send previous (duplicate) data when the FIFO is empty. The TX FIFO operates in dual clock domains and supports asynchronous data movements from the write port to the read port. Synchronization is done by converting the pointer values from binary to gray code, and double registers are used for clock domain crossing.

Before data transmit, the format transfer block will fetch data from TX FIFO based on the word size and write to I2S data buffer based on the format definition (LSB first, MSB location, align, stereo, etc.). The I^{2}S block will serially shift out the data.

13.12 Registers

See *Table 2-3: APB Peripheral Base* Address Map for the base address of this peripheral/module. If multiple instances of the peripheral are provided, each instance has its own, independent set of the registers shown in *Table 13-8: I2S Register Summary*. 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 2-1. Field Access Definitions* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 13-8: I^{2}S Register Summary

14) Offset	15) Register Name	16) Description
17) [0x0000]	*18) I2Sn_CTRL0CH0*	19) Global Mode Control 0 Channel 0
20) [0x0010]	*21) I2Sn_CTRL1CH0*	22) Local Setup Channel 0 Register
23) [0x0030]	*24) I2Sn_DMACH0*	25) DMA Control Channel 0
26) [0x0040]	*27) I2Sn_FIFOCH0*	28) FIFO Channel 0 Register
29) [0x0050]	*30) I2Sn_INTFL*	31) Interrupt Status Register
32) [0x0054]	*33) I2Sn_INTEN*	34) Interrupt Enable Register
35) [0x0058]	*36) I2Sn_EXTSETUP*	37) Control Register

13.12.1 Register Details

Table 13-9: Global Mode Control Channel 0

Global Mode Control 0 Channel				I2Sn_CTRL0CH0	0x0000
Bits	**Name**	**Access**	**Reset**	**Description**	
31:24	rx_thd_val	R/W	0	**RX FIFO Interrupt Threshold Event** This field specifies the depth of receive FIFO for threshold interrupt generation. Values equal to 0 or greater than the FIFO depth are ignored.	
23:21	-	RO	0	**Reserved** Do not modify this field.	

Global Mode Control 0 Channel				I2Sn_CTRL0CH0	0x0000
Bits	**Name**	**Access**	**Reset**	**Description**	
20	fifo_lsb	R/W	0	**Bit Field Control** Only used if the FIFO size is larger than the sample size and *I2Sn_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	W1C	0	**Channel Reset** 0: No action 1: Reset channel	
18	flush	W1C	0	**FIFO Buffer Flush** Write 1 to initiate a flush of the RX and TX FIFOs. Hardware clears this bit to 0 when the operation is complete. 0: No operation. 1: Flush RX FIFO and TX FIFO	
17	rx_en	R/W	0	**RX Enable** 0: Disabled 1: Enabled	
16	tx_en	R/W	0	**TX Enable** 0: Disabled 1: Enabled	
15:14	wsize	R/W	0x3	**Data Size When Writing to FIFO** 0: Byte 1: Half-word (16 bits) 2: Word (32 bits) 3: Word (32 bits)	
13:12	stereo	R/W	0	**I²S Mode** 0: Stereo 1: Stereo 2: Monophonic left channel 3: Monophonic right channel	
11	ext_sel	R/W	0	**External Clock Select** 0: External source 0 (BCLK0/ws0) 1: External source 1 (BCLK1/ws1)	
10	align	R/W	0	**Data Alignment** 0: MSB 1: LSB	
9	msb_loc	R/W	0	**MSB Position Within Field** 0: Second bit of the word 1: First bit of the word	
8	ws_pol	R/W	0	**LRCLK Polarity Select** This setting has no effect if LRCLK is generated from a paired channel. 0: LRCLK low for left channel 0: LRCLK high for left channel	
7:6	ch_mode	R/W	0	**Channel Mode** 0: Master mode, internal generation of BCLK/LRCLK 1: Reserved 2: Reserved 3: Slave mode, external generation of BCLK/LRCLK	

Global Mode Control 0 Channel				I2Sn_CTRL0CH0	0x0000
Bits	**Name**	**Access**	**Reset**	**Description**	
5:2	-	DNM	0	**Reserved** Do not modify this field.	
1	lsb_first	R/W	0	**LSB transmit/received first in SDO/SDI** 0: Disabled 1: Enabled	
0	-	RO	0	**Reserved**	

Table 13-10: Local Setup Channel n Register

Local Setup Channel n				I2Sn_CTRL1CH0	0x0010
Bits	**Name**	**Access**	**Reset**	**Description**	
31:16	clkdiv	R/W	0	**I²S Frequency Divisor**	
15	adjst	R/W	0	**Data Justification** 0: Sample MSB of the data. 1: Sample LSB of the data.	
14	-	RO	0	**Reserved**	
13:9	smp_size	R/W	0	**(Word length of sample size per half frame) – 1** The length equal to *bits_word* when *smp_size* = 0 or *smp_size* > *bits_word*.	
8	en	R/W	0	**I²S Clock Enable** Should be enabled when *I2Sn_CTRL0CH0.msb_loc* = 1. 0: Disabled 1: Enabled	
7:5	-	RO	0	**Reserved**	
4:0	bits_word	R/W	0	**I²S Word Length Per Half Frame** This field is defined as the I²S data bits per half frame minus 1. *Example: If the bit number is 16 per half frame, bits_word is 15.*	

Table 13-11: I²S DMA Channel 0 Register

I²S DMA Channel n				I2Sn_DMACH0	0x0030
Bits	**Name**	**Access**	**Reset**	**Description**	
31:24	rx_lvl	R	0	**Number of Data Words in the TX FIFO**	
23:16	tx_lvl	R	0	**Number of Data Words in the TX FIFO**	
15	dma_rx_en	RW	0	**DMA RX Channel Enable** 0: Disabled 1: Enabled	
14:8	dma_rx_thd_val	RW	0	**DMA RX FIFO Event Threshold** If the RX FIFO level is greater than this value, then the RX FIFO DMA interface will send a signal to the system DMA to notify that RX FIFO has characters to transfer to memory.	
7	dma_tx_en	RW	0	**DMA TX Channel Enable** 0: Disabled 1: Enabled	
6:0	dma_tx_thd_val	RO	0	**DMA TX FIFO Event Threshold** If the TX FIFO level is less than this value, then the TX FIFO DMA interface will send a signal to system DMA to notify that TX FIFO is ready to receive data from memory.	

Table 13-12: I^2S FIFO Channel 0 Register

I^2S FIFO Channel 0				I2Sn_FIFOCH0	0x0040
Bits	**Name**	**Access**	**Reset**	**Description**	
31:0	data	R/W	0	**I^2S FIFO** Writing to this field loads the next character into the TX FIFO and increments the *I2Sn_DMACH0.tx_lvl*. Writes are ignored if the TX FIFO is full. Reads of this field returns the next character available from the RX FIFO and decrements the *I2Sn_DMACH0.rx_lvl*. The value 0x0000_0000 is returned if *I2Sn_DMACH0.rx_lvl* = 0.	

Table 13-13: I^2S Interrupt Flag Register

I^2S Status				I2Sn_INTFL	0x0050
Bits	**Name**	**Access**	**Reset**	**Description**	
31:4	-	DNM	0	**Reserved, Do Not Modify** Do not modify this field.	
3	tx_he_ch0	W1C	0	**TX FIFO Half-Empty Event Interrupt Flag Channel 0** 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	**TX FIFO One Byte Remaining Event Interrupt Flag Channel 0** 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	**RX Threshold Event Interrupt Flag Channel 0** 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	**RX FIFO Overrun Event Interrupt Flag Channel 0** If this field is set to 1 the event has occurred. Write 1 to clear. 0: No event 1: Event occurred	

Table 13-14: I^2S Interrupt Enable Register

I^2S Interrupt Enable				I2Sn_INTEN	0x0054
Bits	**Name**	**Access**	**Reset**	**Description**	
31:4	-	RO	0	**Reserved, Do Not Modify** Reserved for Future Use, do not modify this field.	
3	tx_he_ch0	R/W	0	**TX FIFO Half-Empty Event Interrupt Enable Channel 0** 0: Disabled 1: Enabled	
2	tx_ob_ch0	R/W	0	**TX FIFO One Byte Remaining Event Interrupt Enable Channel 0** 0: Disabled 1: Enabled	
1	rx_thd_ch0	R/W	0	**RX FIFO Threshold Event Interrupt Enable Channel 0** 0: Disabled 1: Enabled	

I²S Interrupt Enable				I2Sn_INTEN	0x0054
Bits	**Name**	**Access**	**Reset**	**Description**	
0	rx_ov_ch0	R/W	0	**RX FIFO Overrun Event Interrupt Enable Channel 0** 0: Disabled 1: Enabled	

Table 13-15: I²S External Setup Register

I²S External Setup				I2Sn_EXTSETUP	0x0058
Bits	**Name**	**Access**	**Reset**	**Description**	
31:5	-	RO	0	**Reserved**	
4:0	ext_bits_word	R/W	0	**Word length for *ch_mode* = 2**	

14. 1-Wire Master (OWM)

The device provides a 1-Wire master (OWM) that you can use to communicate with one or more external 1-Wire slave devices using a single-signal, combined clock, data protocol. The OWM is contained in the OWM module. The OWM module handles the lower-level details (including timing and drive modes) required by the 1-Wire protocol, allowing the CPU to communicate over the 1-Wire bus at a logical data level.

14.1 1-Wire Master Features

The OWM provides the following features:

- Flexible, 1-Wire timing generation (required 1MHz timing base) using the OWM module clock frequency, which is in turn derived from the current system clock source. You can also prescale the OWM module clock to allow proper 1-Wire timing generation using a range of base frequencies.
- Automatic generation of proper 1-Wire time slots for both standard and overdrive timing modes.
- Flexible configuration for 1-Wire line pullup modes: options for internal pullup, external fixed pullup, and optional external strong pullup are available.
- Long-line compensation and bit banging (direct firmware drive) modes.
- 1-Wire reset generation and presence-pulse detection.
- Generation of 1-Wire read and write time slots for single-bit and eight-bit byte transmissions.
- Search ROM Accelerator (SRA) mode, which simplifies the generation of multiple-bit time slots and discrepancy resolution required when completing the Search ROM function to determine the IDs of multiple, unknown 1-Wire slaves on the bus.
- Transmit data completion, received data available, presence pulse detection, and 1-Wire line-error condition interrupts.

For more information about the Maxim 1-Wire protocol and supporting devices, refer to the following resources:

- *AN937: The Book of iButton Standards*
    - *www.maximintegrated.com/AN937*
- *AN1796: Overview of 1-Wire Technology and its Use*
    - *www.maximintegrated.com/AN1796*
- *AN187: 1-Wire Search Algorithm*
    - *www.maximintegrated.com/AN187*

14.2 1-Wire Pins and Configuration

The one instance of the peripheral is shown in *Table 14-1: MAX78000 1-Wire Master Peripheral Pins* lists the locations of the OWM_IO and OWM_PE signals for each of the OWM*n* peripherals per package.

Table 14-1: MAX78000 1-Wire Master Peripheral Pins

OWMn	ALTERNATE FUNCTION	ALT FUNCTION #	81-CTBGA
OWM0	OWM_IO	AF1	P0.6
	OWM_PE	AF1	P0.7

14.2.1 1-Wire I/O (OWM_IO)

The OWM_IO pin is a bidirectional I/O that is used to directly drive the external 1-Wire bus. As described in the *Book of iButton Standards*, this I/O is generally driven as an open-drain output. The 1-Wire bus requires a common pullup to return the 1-Wire bus line to an idle high state when no master or slave device is actively driving the line low. This pullup can consist of a fixed resistor pullup (connected to the 1-Wire bus outside the microcontroller), an internal pullup enabled by setting *OWM_CFG.int_pullup_enable* to 1, or an OWM module controlled external pullup enabled by setting *OWM_CFG.ext_pullup_mode* to 1.

14.2.2 Pullup Enable (OWM_PE)

The 1-Wire pullup enable (PE) signal is an active high output used to enable an optional external pullup on the 1-Wire bus. This pullup is intended to provide a stronger (lower impedance) pullup on the 1-Wire bus under certain circumstances, such as during overdrive mode.

14.2.3 Clock Configuration

To correctly generate the timing required by the 1-Wire protocol in Standard or Overdrive timing modes, the OWM clock must be set to achieve $f_{owmclk} = 1\text{MHz}$. This clock generates both the Standard and Overdrive timing, so it does not need adjustment when transitioning from Standard to Overdrive mode or vice versa.

The OWM peripheral uses the system peripheral clock, PCLK, divided by the value in the *OWM_CLK_DIV_1US.divisor* field as shown in *Equation 14-1*, below, where $f_{PCLK} = {f_{SYSCLK}}/{2}$.

Equation 14-1: OWM 1MHz Clock Frequency

$$f_{owmclk} = 1MHz = \frac{f_{PCLK}}{OWM\_CLK\_DIV\_1US.divisor}$$

14.3 1-Wire Protocol

The general timing and communication protocols used by the OWM interface are those standardized for the 1-Wire network.

Because the 1-Wire interface is a master interface, it initiates and times all communication on the 1-Wire bus. Except for the present pulse generation when a device first connects to the 1-Wire bus, 1-Wire slave devices complete 1-Wire bus communication only as directed by the 1-Wire bus master. From a firmware perspective, the lowest-level timing and electrical details of how the 1-Wire network operates are unimportant. The application can configure the OWM module properly and direct it to complete low-level operations such as reset, read, and write bit/byte operations. Thus, the OWM module on the microcontroller is designed to interface to the 1-Wire bus at a low level.

14.3.1 Networking Layers

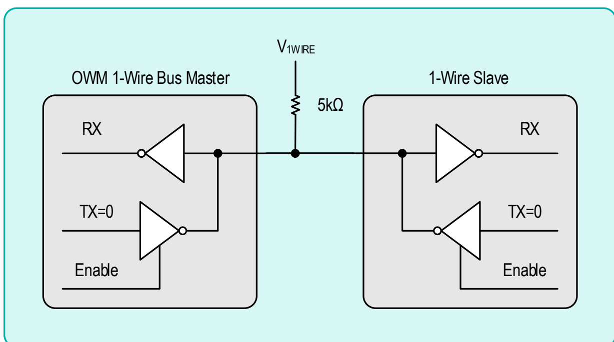
In the *Book of iButton Standards*, the 1-Wire communication protocol is described in terms of the ISO-OSI model (International Organization of Standardization (ISO) Open System Interconnection (OSI) network layer model). Network layers that apply to this description are the Physical, Link, Network, and Transport layers. The Transport layer consists of the software that transfers memory data other than ROM ID contents to and from the individual 1-Wire network nodes. The Presentation layer would correspond to higher-level application software functions (such as library layers) that implement communication protocols using the 1-Wire layers as a foundation. This document describes the details of the Physical, Link and Network layers with regards to the OSI network layer model. The Transport and Presentation layers are beyond the scope of this document.

14.3.1.1 Physical Layer

The 1-Wire communication bus consists of a single data/power line plus ground. devices (either master or slave) that interface to the 1-Wire communication bus using an open-drain (active low) connection, which means that the 1-Wire bus normally idles in a high state.

An external pullup resistor is used to pull the 1-Wire line high when no master or slave device is driving the line. This means that 1-Wire devices do not actively drive the 1-Wire line high. Instead, they either drive the line low or release it (set their output to high impedance) to allow the external resistor to pull the line high. This allows the 1-Wire bus to operate in a wired-AND manner as shown in *Figure 14-1* and avoids bus contention if more than one device attempts to drive the 1-Wire bus at the same time.

Figure 14-1: 1-Wire Signal Interface



14.3.1.2 Link Layer

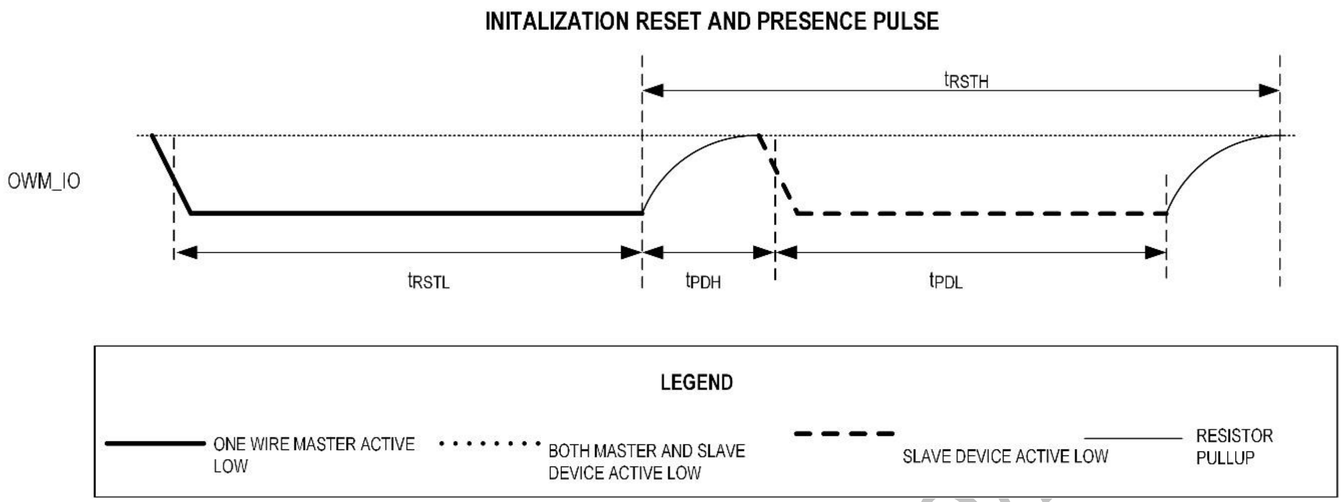
The 1-Wire Bus supports a single master and one or more slave devices (multidrop). Slave devices can connect to and disconnect from the 1-Wire Bus dynamically (as is typically the case with iButton devices that operate using an intermittent touch contact interface), which means that it is the master's responsibility to poll the bus as needed to determine the number and types of 1-Wire devices that are connected to the bus.

All communication sequences on the 1-Wire Bus are initiated by the OWM. The OWM determines when 1-Wire data transmissions begin, as well as the overall communication speed that is used. There are three different communication speeds supported by the 1-Wire specification: standard speed, overdrive speed, and hyperdrive speed. However, only standard speed and overdrive speed are supported by the OWM peripheral in the devices.

14.3.1.2.1 OWM Reset and Presence Detect

The OWM begins each communication sequence by sending a reset pulse as shown in *Figure 14-2*. This pulse resets all 1-Wire slave devices on the line to their initial states and causes them all to begin monitoring the line for a command from the OWM. Each 1-Wire slave device on the line responds to the reset pulse by sending out a presence pulse. These pulses from multiple 1-Wire slave devices are combined in wired-AND fashion, resulting in a pulse whose length is determined by the slowest 1-Wire slave device on the bus.

Figure 14-2: 1-Wire Reset Pulse



In general, the 1-Wire line must idle in a high state when communication is not taking place. It is possible for the master to pause communication in between time slots. There is not an overall "timeout" period that causes a slave to revert to the reset state if the master takes too long between one time slot and the next time slot.

The 1-Wire communication protocol relies on the fact that the maximum allowable length for a bit transfer (write 0/1 or read bit) time slot is less than the minimum length for a 1-Wire reset. At any time, if the 1-Wire line is held low (by the master or by any slave device) for more than the minimum reset pulse time, all slave devices on the line interpret this as a 1-Wire reset pulse.

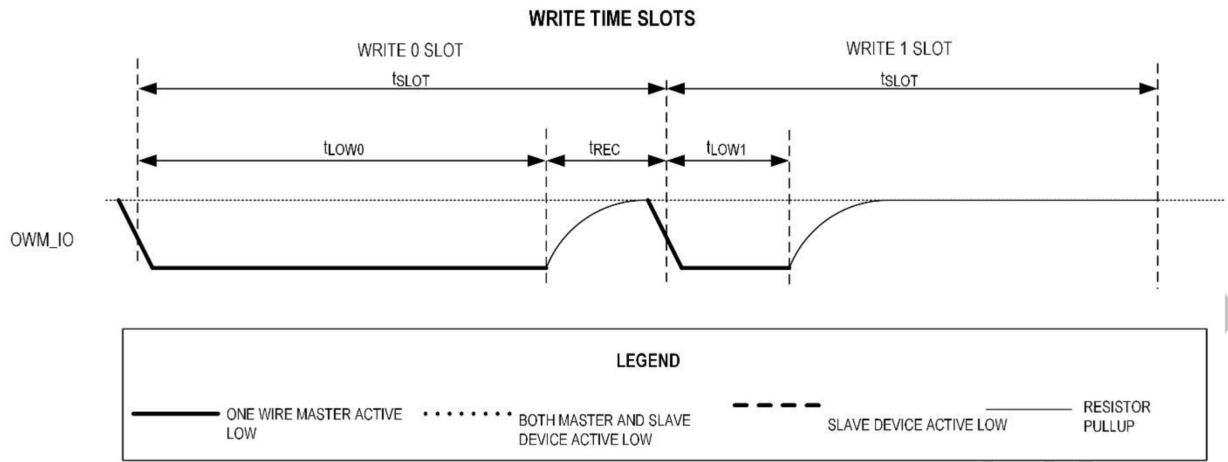
14.3.1.2.2 OWM Write Time Slot

All 1-Wire bit time slots are initiated by the 1-Wire bus master and begin with a single falling edge. There is no indication given by the beginning of a time slot whether a read bit or write bit operation is intended, as the time slots all begin in the same manner. Rather, the 1-Wire command protocol enforces agreement between the OWM and slave as to which time slots are used for bit writes and which time slots are used for bit reads.

When multiple bits of a value are transmitted (or read) in sequence, the least significant bit of the value is always sent or received first. The 1-Wire bus is a half-duplex bus, so data is transmitted in only one direction (from master to slave or from slave to master) at any given time.

As shown in *Figure 14-3*, the time slots for writing a 0 bit and writing a 1 bit begin identically, with the falling edge and a minimum-width low pulse sent by the master. To write a one bit, the master releases the line after the minimum low pulse, allowing it to be pulled high. To write a zero bit, the master continues to hold the line low until the end of the time slot.

Figure 14-3: 1-Wire Write Time Slot



From the slave's perspective, the initial falling edge of the time slot triggers the start of an internal timer, and when the proper amount of time has passed, the slave samples the 1-Wire line that is driven by the master. This sampling point is in between the end of the minimum-width low pulse and the end of the time slot.

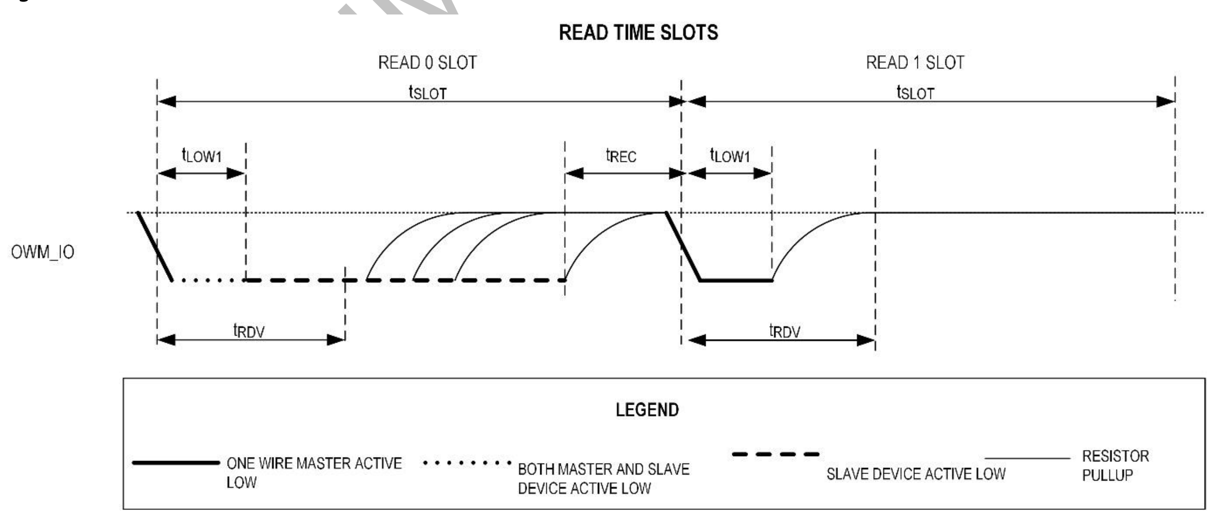
14.3.1.2.3 OWM Read Time Slot

As with all 1-Wire transactions, the master initiates all bit read time slots. Like the bit write time slots, the bit read time slot begins with a falling edge. From the master's perspective, this time slot is transmitted identically to the "Write 1 Bit" time slot shown in *Figure 14-3*. The master begins by transmitting a falling edge, holds the line low for a minimum-width period, and then releases the line.

The difference here is that instead of the slave sampling the line, the slave begins transmitting either a 0 (by holding the line low) or a 1 (by leaving the line to float high) after the initial falling edge. The master then samples the line to read the bit value that is transmitted by the slave device.

As an example, *Figure 14-4* shows a sequence in which the slave device transmits data back to the 1-Wire bus master upon request. Note that to transmit a 1 bit, the slave device does not need to do anything. It simply leaves the line alone (to float high) and waits for the next time slot. To transmit a 0 bit, the slave device holds the line low until the end of the time slot.

Figure 14-4: 1-Wire Read Time Slot



14.3.1.2.4 Standard Speed and Overdrive Speed

By default, all 1-Wire communications following reset begin at the lowest rate of speed (that is, standard speed). For 1-Wire devices that support it, it is possible for the OWM to increase the rate of communication from standard speed to overdrive speed by sending the appropriate command.

The protocols and time slots operate identically for standard and overdrive speeds. The difference comes in the widths of the time slots and pulses. The OWM automatically adjusts the timings based on the setting of the *OWM_CFG.overdrive* field.

If a 1-Wire slave device receives a standard speed reset pulse, it resets and reverts to standard speed communication. If the device is already communicating in overdrive mode, and it receives a reset pulse at the overdrive speed, it resets but remains in overdrive mode.

14.3.1.3 Network Layer

14.3.1.3.1 ROM Commands

Following the initial 1-Wire reset pulse on the bus, all slave 1-Wire devices are active, which means that they are monitoring the bus for commands. Because the 1-Wire bus can have multiple slave devices present on the bus at any time, the OWM must go through a process (defined by the 1-Wire command protocol) to activate only the 1-Wire slave device that it intends to communicate with and deactivate all others. This is the purpose of the ROM commands (network layer) shown in *Table 14-2*.

Table 14-2: 1-Wire ROM Commands

ROM Command	Hex Value
Read ROM	0x33
Match ROM	0x55
Search ROM	0xF0
Skip ROM	0xCC
Overdrive Skip ROM	0x3C
Overdrive Match ROM	0x69
Resume Communication	0xA5

The ROM command layer relies on the fact that all 1-Wire slave devices are assigned a globally-unique, 64-bit ROM ID. This ROM ID value is factory programmed to ensure that no two 1-Wire slave devices have the same value.

14.3.1.3.2 ROM ID

Figure 14-5 is a visual representation of the 1-Wire ROM ID fields and shows the organization of the fields within the 64-bit ROM ID for a device.

Figure 14-5: 1-Wire ROM ID Fields

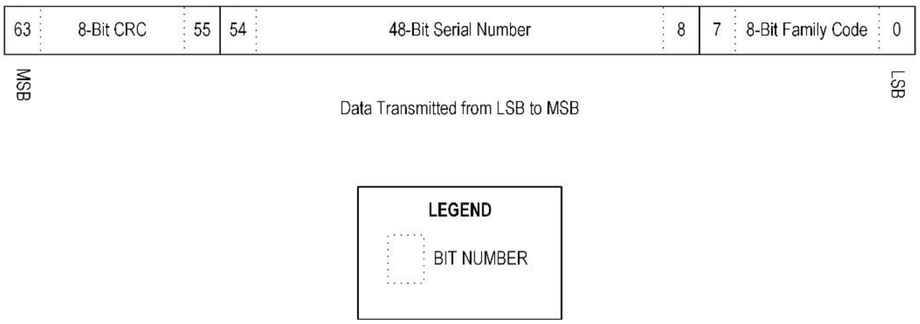


Table 14-3 provides a detailed description of each of the ROM ID fields.

Table 14-3: 1-Wire Slave Device ROM ID Field

Field	Bit Number	Description
Family code	0-7	This 8-bit value is used to identify the type of a 1-Wire slave device.
Unique ID	8-55	This 48-bit value is factory-programmed to give each 1-Wire slave device (within a given family code group) a globally unique identifier.
CRC	56-63	This is the 8-bit, 1-Wire CRC as defined in the *Book of iButton Standards*. The CRC is generated using the polynomial ($x^8 + x^5 + x^4 + 1$).

Note: For certain operations that consist only of writing from the OWM to the slave, it is technically possible for the master to communicate with more than one slave at a time on the same 1-Wire bus. For this to work, the exact same data must be transmitted to all slave devices, and any values read back from the slaves must either be identical as well or must be disregarded by the master device (because different slaves can attempt to transmit different values). The descriptions below assume, however, that the master is communicating with only one slave device at a time because this is the method that is normally used.

As explained above, the ROM ID contents play an important role in addressing and selecting devices on the 1-Wire bus. All devices except one are in an idle/inactive state after the Match ROM command or the Search ROM command is executed. They return to the active state only after receiving a 1-Wire reset pulse.

Devices with overdrive capability are distinguished from others by their family code and two additional ROM commands: Overdrive Skip ROM and Overdrive Match ROM. The first transmission of the ROM command itself takes place at the normal speed that is understood by all 1-Wire devices. After a device with overdrive capability is addressed and set into overdrive mode (that is, after the appropriate ROM command is received), further communication to that device must occur at overdrive speed. Because all deselected devices remain in the idle state as long as no reset pulse of regular duration is detected, even multiple overdrive components can reside on the same 1-Wire bus. A reset pulse of regular duration resets all 1-Wire devices on the bus and simultaneously sets all overdrive-capable devices back to the default standard speed.

14.3.2 Read ROM Command

The Read ROM command allows the OWM to obtain the 8-byte ROM ID of any slave device connected to the 1-Wire bus. Each slave device on the bus responds to this command by transmitting all eight bytes of its ROM ID value starting with the least significant byte (Family Code) and ending with the most significant byte (CRC).

Because this command is addressed to all 1-Wire devices on the bus, if more than one slave is present on the bus there is a data collision as multiple slaves attempt to transmit their ROM IDs at once. This condition is detectable by the OWM because the CRC value does not match the ROM ID value received. In this case, the OWM should reset the 1-Wire bus and select a single slave device on the bus to continue either by using the Match ROM command (if the ROM ID values are already known) or the Search ROM command (if the master has not yet identified some or all devices on the bus).

After the Read ROM command is complete, all slave devices on the 1-Wire bus are selected or active, and communication proceed to the Transport layer.

14.3.3 Skip ROM and Overdrive Skip ROM Commands

The Skip ROM command is used to activate all slave devices present on the 1-Wire bus regardless of their ROM ID. Normally, this command is used when only a single 1-Wire slave device is connected to the bus. After the Skip ROM command is complete, all slave devices on the 1-Wire bus are selected or active and communication proceeds to the Transport layer.

The Overdrive Skip ROM command operates in an identical manner except that running it also causes the receiving slave devices to shift communication speed from standard speed to overdrive speed. The Overdrive Skip ROM command byte itself (0x3C) is transmitted at standard speed. All subsequent communication is sent at overdrive speed.

14.3.4 Match ROM and Overdrive Match ROM Commands

The Match ROM command is used by the OWM to select one and only one slave 1-Wire device when the ROM ID of the device has already been determined. When transmitting this command, the master sends the command byte (that is, 55h for standard speed and 69h for overdrive speed) and then sends the entire 64-bit ROM ID for the device selected, least significant bit first.

During the transmission of the ROM ID by the master, all slave devices monitor the bus. As each bit is transmitted, each of the slave devices compares it against the corresponding bit of their ROM ID. If the bits match, the slave device continues to monitor the bus. If the bits do not match, the slave device transitions to the inactive state (waiting for a 1-Wire reset) and stops monitoring the bus.

At the end of the transmission, at most one slave device is active, which is the slave device whose ROM ID matched the ROM ID that was transmitted. All other slave devices are inactive. Communication then proceeds to the Transport layer for the device that was selected.

The Overdrive Match ROM command operates in an identical manner except that it also causes the slave device that is selected by the command to shift communication speed from standard speed to overdrive speed. The Overdrive Match ROM command byte (69h) and the 64-bit ROM ID bits are transmitted at standard speed. All subsequent communication is sent at overdrive speed.

14.3.5 Search ROM Command

The Search ROM command allows the OWM to determine the ROM ID values of all 1-Wire slave devices connected to the bus using an iterative search process. Each execution of the Search ROM command reveals the ROM ID of one slave device on the bus.

The operation of the Search ROM command resembles a combination of the Read ROM and Match ROM commands. First, all slaves on the bus transmit the least significant bit (Bit 0) of their ROM IDs. Next, all slaves on the bus transmit a complement of the same bit. By analyzing the two bits received, the master can determine whether the Bit 0 values were 0 for all slaves, 1 for all slaves, or a combination of the two. Next, the master selects which slaves remain activated for the next step in the Search ROM process by transmitting the Bit 0 value for the slaves it selects. All slaves whose Bit 0 matches the value transmitted by the master remain active, while slaves with a different Bit 0 value go to the inactive state and do not participate in the remainder of the Search ROM command.

Next, the same process is followed for Bit 1, then Bit 2, and so on until the 63rd bit (most significant bit) of the ROM ID is transmitted. At this point only one slave device remains active, and the master can either continue with communication at the Transport layer or issue a 1-Wire reset pulse to go back for another pass at the Search ROM command.

The *Book of iButton Standards* goes into more detail about the process that is used by the master to obtain ROM IDs of all devices on the 1-Wire bus using multiple executions of the Search ROM command. The algorithm resembles a binary tree search and is used regardless of how many devices are on the bus.

There is no overdrive equivalent version of the Search ROM command.

14.3.6 Search ROM Accelerator Operation

To allow the Search ROM command to process more quickly, the OWM module provides a special accelerator mode for use with the Search ROM command. This mode is activated by setting *OWM_CTRL_STAT.sra_mode* to 1.

When this mode is active, ROM IDs being processed by the Search ROM command are broken into 4-bit nibbles where the current 64-bit ROM ID varies with each pass through the search algorithm. Each 4-bit processing step is initiated by writing the 4-bit value to *OWM_DATA.tx_rx*. This causes the generation of twelve 1-Wire time slots by the OWM as each bit in the 4-bit value (starting with the LSB) results in a read of two bits (all active slaves transmitting bit N of their ROM IDs, then all active slaves transmitting the complement of bit N of their ROM ID), and then a write of a single bit by the OWM.

After the 4-bit processing stage is complete, the return value loaded into *OWM_DATA.tx_rx* consists of 8 bits. The low nibble (bits 0 through 3) contains the four discrepancy flags: one for each ID bit processed. If the discrepancy bit is set to 1, it means that either two slaves with differing ID bits in that position both responded (the 2 bits read were both zero), or that no slaves responded (the 2 bits read were both 1). If the discrepancy bit is set to 0, then the 2 bits read were complementary (either 0, 1 or 1, 0) meaning that there was no bus conflict.

In this way, at each step in the Search ROM command, the master either follows the ID of the responding slaves or deselects some of the slaves on the bus in case of a conflict. By the time the end of the 64-bit ROM ID is reached (the sixteenth 4-bit group processing step), the combination of all bits from the high nibbles of the received data are equal to the ROM ID of one of the slaves remaining on the bus. Subsequent passes through the Search ROM algorithm are used to determine additional slave ROM ID values until all slaves are identified. Refer to the *Book of iButton Standards* for a detailed explanation of the search function and possible variants of the search algorithm applicable to specific circumstances.

14.3.7 Resume Communication Command

If more than one 1-Wire slave device is on the bus, then the master must specify which one it wishes to communicate with each time a new 1-Wire command (starting with a reset pulse) begins. Using the commands discussed previously, this would normally involve sending the Match ROM command each time, which means the master must explicitly specify the full 64-bit ROM ID of the part it communicates with for each command.

The Resume Communication command provides a shortcut for this process by allowing the master to repeatedly select the same device for multiple commands without having to transmit the full ROM ID each time.

When the OWM selects a single device (using the Match ROM or Search ROM commands), an internal flag called the RC (for Resume Communication) flag is set in the slave device. (Only one device on the bus has this flag set at any one time; the Skip ROM command selects multiple devices, but the RC flag is not set by the Skip ROM command.)

When the master resets the 1-Wire bus, the RC flag remains set. At this point, it is possible for the master to send the Resume Communication command. This command does not have a ROM ID attached to it, but the device that has the RC flag set responds to this command by going to the active state while all other devices deactivate and drop off the 1-Wire bus.

Issuing any other ROM command clears the RC flag on all devices. So, for example, if a Match ROM command is issued for device A, its RC flag is set. The Resume Communication command can then be used repeatedly to send commands to device A. If a Match ROM command is then sent with the ROM ID of device B, the RC flag on device A will clear to 0, and the RC flag on device B is set.

14.4 1-Wire Operation

Once the OWM peripheral is correctly configured, then using the OWM peripheral to communicate with the 1-Wire network involves directing the OWM to generate the proper reset, read, and write operations to communicate with the 1-Wire slave devices used in a specific application.

The OWM handles the following 1-Wire protocol primitives directly in either Standard or Overdrive mode:

- 1-Wire bus reset (including detection of presence pulse from responding slave devices)
- Write single bit (a single write time slot)
- Write 8-bit byte, least significant bit first (eight write time slots)
- Read single bit (a single write-1 time slot)
- Read 8-bit byte, least significant bit first (eight write-1 time slots)
- Search ROM Acceleration Mode allowing the generation of four groups of three time slots (read, read, and write) from a single 4-bit register write to support the Search ROM command

14.4.1 Resetting the OWM

The first step in any 1-Wire communication sequence is to reset the 1-Wire bus. To direct the OWM module to complete a 1-Wire reset, write *OWM_CTRL_STAT.start_ow_reset* to 1. This generates a reset pulse and checks for a replying presence pulse from any connected slave devices.

Once the reset time slot is complete, the *OWM_CTRL_STAT.start_ow_reset* field is automatically cleared to zero. Then, the interrupt flag *OWM_INTFL.ow_reset_done* is set to 1 by hardware. This flag must be cleared by writing a 1 bit to the flag.

If a presence pulse is detected on the 1-Wire bus during the reset sequence (that should normally be the case unless no 1-Wire slave devices are present on the bus), the *OWM_CTRL_STAT.presence_detect* flag is also set to 1. This flag does not result in the generation of an interrupt.

14.5 1-Wire Data Reads

14.5.1 Reading a Single Bit Value from the 1-Wire Bus

The procedure for reading a single bit is like the procedure for writing a single bit because the operation is completed by writing a 1 bit that the slave device either leaves unchanged (to transmit a 1 bit) or overrides by forcing the line low (to transmit a 0 bit).

To read a single bit value from the 1-Wire Bus, complete the following steps:

1. Set *OWM_CFG.single_bit_mode* to 1. This setting causes the OWM to transmit/receive a single bit of data at a time instead of the default 8 bits.
2. Write *OWM_DATA.tx_rx* to 1. Only bit 0 of this field is used in this instance; the other bits in the field are ignored. Writing to the *OWM_DATA* register initiates the read of the bit on the 1-Wire bus.
3. Once the single-bit transmission is complete, hardware sets the interrupt flag *OWM_INTFL.tx_data_empty* to 1. This flag (that triggers an OWM module interrupt if *OWM_INTEN.tx_data_empty* is also set to 1) is cleared by writing a 1 to the flag.
4. As the hardware shifts the bit value out, it also samples the value returned from the slave device. Once this value is ready to read, the interrupt flag *OWM_INTFL.rx_data_ready* is set to 1. If *OWM_INTEN.rx_ready* is set to 1, an OWM module interrupt occurs.
5. Read *OWM_DATA.tx_rx* (only bit 0 is used) to determine the value returned by the slave device. Note that if no slave devices are present or the slaves are not communicating with the master, bit 0 remains set to 1.

14.5.2 Reading an 8-Bit Value from the 1-Wire Bus

The procedure for reading an 8-bit byte is like the procedure for writing an 8-bit byte because the operation is completed by writing eight 1 bits that the slave device either leaves unchanged (to transmit 1 bits) or overrides by forcing the line low (to transmit 0 bits).

Set *OWM_CFG.single_bit_mode* to 0. This setting causes the OWM to transmit/receive in the default 8-bit mode.

Write *OWM_DATA.tx_rx* to 0x0FFh.

Once the 8-bit transmission completes, hardware sets the interrupt flag *OWM_INTFL.tx_data_empty* to 1. This flag (that triggers an OWM module interrupt if *OWM_INTEN.tx_data_empty* is also set to 1) is cleared by writing a 1 to the flag.

As the hardware shifts the bit values out, it also samples the values returned from the slave device. Once the full 8-bit value is ready to be read, the interrupt flag *OWM_INTFL.rx_data_ready* is set to 1. If *OWM_INTEN.rx_ready* is set to 1, an OWM module interrupt occurs.

Read *OWM_DATA.tx_rx* to determine the 8-bit value returned by the slave device. *Note that if no slave devices are present or the slave devices are not communicating with the master, the return value 0x0FF is the same as the transmitted value.*

14.6 Registers

See *Table 2-3: APB Peripheral Base Address Map* for this peripheral/module's base address. If multiple instances are provided, each will have a unique base address. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific reset, if applicable.

Table 14-4: OWM Register Summary

Offset	Register	Description
[0x0000]	*OWM_CFG*	OWM Configuration Register
[0x0004]	*OWM_CLK_DIV_1US*	OWM Clock Divisor Register
[0x0008]	*OWM_CTRL_STAT*	OWM Control/Status Register
[0x000C]	*OWM_DATA*	OWM Data Buffer Register
[0x0010]	*OWM_INTFL*	OWM Interrupt Flag Register

Offset	Register	Description
[0x0014]	*OWM_INTEN*	OWM Interrupt Enable Register

14.7 Register Details

Table 14-5: OWM Configuration Register

OWM Configuration Register		OWM_CFG		[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	0	**Reserved for Future Use** Do not modify this field.
7	int_pullup_enable	R/W	0	**Internal Pullup Enable** Set this field to enable the internal pullup resistor. 0: Internal pullup disabled. 1: Internal pullup enabled.
6	overdrive	R/W	0	**Overdrive Enable** Set this field to 1 to enable overdrive mode for 1-Wire communications. Clearing this field sets 1-Wire communications to standard speed. 0: Overdrive mode disabled, standard speed mode. 1: Overdrive mode enabled.
5	single_bit_mode	R/W	0	**Bit Mode Enable** When set to 1, only a single bit at a time is transmitted and received (LSB of *OWM_DATA*) rather than the whole byte. 0: Byte mode enabled, single bit mode disabled. 1: Single bit mode enabled, byte mode disabled.
4	ext_pullup_enable	R/W	0	**External Pullup Enable** Enables external FET pullup when the 1-Wire master is idle. FET is designed to pull the wire high regardless of its enable state (that is, high or low). Idle means the 1-Wire master is idle, and there are no 1-Wire accesses in progress. 0: External pullup pin is not driven to high. 1: External pullup pin is driven high when the 1-Wire bus is idle, actively pulling the 1-Wire IO high.
3	ext_pullup_mode	R/W	0	**External Pullup Mode** Provides an extra output to control an external pullup. For long wires, a pullup resistor strong enough to pull the wire high in a reasonable amount of time might need to be so strong that it would be difficult to drive the line low. In this case, implement an external FET to actively drive the wire high for a brief amount of time. Then, let the resistor keep the line high.
2	bit_bang_en	R/W	0	**Bit-Bang Mode Enable** Enable bit-bang control of the I/O pin. If this bit is set to 1, *OWM_CTRL_STAT.bit_bang_oe* controls the state of the I/O pin. 0: Bit-bang mode disabled. 1: Bit-bang mode enabled.

OWM Configuration Register			OWM_CFG	[0x0000]
Bits	**Field**	**Access**	**Reset**	**Description**
1	force_pres_det	R/W	1	**Presence Detect Force** Setting this bit to 1 drives the OWM_IO pin low during presence detection. Use this bit field to prevent a large number of 1-Wire slaves on the bus from all responding at different times, which might cause ringing. When this bit is set to 1, the *OWM_CTRL_STAT.presence_detect* bit is always set as the result of a 1-Wire reset even if no slave devices are present on the bus. 0: OWM_IO pin floats during presence detection portion of 1-Wire reset. 1: OWM_IO pin is driven low during presence detection portion of 1-Wire reset.
0	long_line_mode	R/W	0	**Long Line Mode Enable** Selects alternate timings for 1-Wire communication. The recommended setting depends on the length of the wire. For lines less than 40 meters, 0 should be used. Setting this bit to 0 leaves the write one release, the data sampling, and the time-slot recovery times at approximately 5µs, 15µs, and 7µs, respectively. Setting this bit to 1 enables long line mode timings during standard mode communications. This mode moves the write one release, the data sampling, and the time-slot recovery times out to approximately 8µs, 22µs, and 14µs, respectively. 0: Standard operation for lines less than 40 meters. 1: Long line mode enabled, see description above.

Table 14-6: OWM Clock Divisor Register

OWM Clock Divisor Register			OWM_CLK_DIV_1US	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R	0	**Reserved for Future Use** Do not modify this field.
7:0	divisor	R/W	0	**OWM Clock Divisor** Divisor for the OWM peripheral clock. The target is to achieve a 1MHz clock. See the *Clock Configuration* section for details. 0x00: OWM clock disabled. 0x01: $f_{owmclk} = \frac{f_{PCLK}}{1}$ 0x02: $f_{owmclk} = \frac{f_{PCLK}}{2}$ … 0xFF: $f_{owmclk} = \frac{f_{PCLK}}{255}$

Table 14-7: OWM Control/Status Register

OWM Control/Status Register			OWM_CTRL_STAT	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	RO	0	**Reserved for Future Use** Do not modify this field.
7	presence_detect	RO	0	**Presence Detect Flag** Set to 1 when a presence pulse is detected from one or more slaves during the 1-Wire reset sequence. 0: No presence detect pulse during previous 1-Wire reset sequence. 1: Presence detect pulse on bus during previous 1-Wire reset sequence.

OWM Control/Status Register			OWM_CTRL_STAT	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**
6:5	-	RO	0	**Reserved for Future Use** Do not modify this field.
4	od_spec_mode	RO	0	**Overdrive Spec Mode** Returns the version of the overdrive spec.
3	ow_input	RO	-	**OWM_IN State** Returns the current logic level on the OWM_IO pin. 0: OWM_IO pin is low. 1: OWM_IO pin is high.
2	bit_bang_oe	R/W	0	**OWM Bit-Bang Output** When bit-bang mode is enabled (*OWM_CFG.bit_bang_en* = 1), this bit sets the state of the OWM_IO pin. Setting this bit to 1 drives the OWM_IO pin low. Setting this bit to 0 releases the line, allowing the OWM_IO pin to be pulled high by the pullup resistor or held low by a slave device. 0: OWM_IO pin floating. 1: Drive OWM_IO pin to low state.
1	sra_mode	R/W	0	**Search ROM Accelerator Enable** Enable Search ROM Accelerator mode. This mode is used to identify slaves and their addresses that are attached to the 1-Wire bus. 0: Search ROM accelerator mode disabled. 1: Search ROM accelerator mode enabled.
0	start_ow_reset	R/W	0	**Start 1-Wire Reset Pulse** Write 1 to start a 1-Wire reset sequence. Automatically cleared by the OWM hardware when the reset sequence is complete. 0: 1-Wire reset sequence complete or inactive. 1: Start a 1-Wire reset sequence.

Table 14-8: OWM Data Buffer Register

OWM Data Register			OWM_DATA	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	-	R/W	0	**Reserved for Future Use** Do not modify this field.
7:0	tx_rx	R/W	0	**OWM Data Field** Writing to this field sets the transmit data and initiates a 1-Wire data transmit cycle. Reading from this field returns the data received by the master during the last 1-Wire data transmit cycle.

Table 14-9: OWM Interrupt Flag Register

OWM Interrupt Flag Register			OWM_INTFL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	R/W	0	**Reserved for Future Use** Do not modify this field.
4	line_low	R/W1C	0	**Line Low Flag** If this flag is set, the OWM_IO pin was in a low state. Write 1 to clear this flag.
3	line_short	R/W1C	0	**Line Short Flag** The OWM hardware detected a short on the OWM_IO pin. Write 1 to clear this flag.

OWM Interrupt Flag Register			OWM_INTFL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
2	rx_data_ready	R/W1C	0	**RX Data Ready** Data received from the 1-Wire bus and is available in the *OWM_DATA.tx_rx* field. Write 1 to clear this flag. 0: RX data not available. 1: Data received and is available in the *OWM_DATA.tx_rx* field.
1	tx_data_empty	R/W1C	0	**TX Empty** The OWM hardware automatically sets this interrupt flag when the data transmit is complete. Write 1 to clear this flag. 0: Either no data was sent or the data in the *OWM_DATA.tx_rx* field has not completed transmission. 1: Data in the *OWM_DATA.tx_rx* field was transmitted.
0	ow_reset_done	R/W1C	0	**Reset Complete** This flag is set when a 1-Wire reset sequence completes. To start a 1-Wire reset sequence, see *OWM_CTRL_STAT.start_ow_reset*. Write 1 to clear this flag. 0: 1-Wire reset sequence not complete or bus idle. 1: 1-Wire reset sequence complete.

Table 14-10: OWM Interrupt Enable Register

OWM Interrupt Enable Register			OWM_INTEN	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:5	-	RO	0	**Reserved for Future Use** Do not modify this field.
4	line_low	R/W	0	**Line Low Interrupt Enable** I/O pin low detected interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
3	line_short	R/W	0	**Line Short Interrupt Enable** I/O pin short detected interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
2	rx_data_ready	R/W	0	**Receive Data Ready Interrupt Enable** RX data ready interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
1	tx_data_empty	R/W	0	**Transmit Data Empty Interrupt Enable** TX data empty interrupt enable. 0: Interrupt disabled. 1: Interrupt enabled.
0	ow_reset_done	R/W	0	**1-Wire Reset Sequence Complete Interrupt Enable** 1-Wire reset sequence completed. 0: Interrupt disabled. 1: Interrupt enabled.

15. Real-Time Clock (RTC)

15.1 Overview

The Real-Time Clock (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 datasheet 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 register *RTC_SEC* is incremented every time there is a rollover of the *RTC_SSEC.ssec* sub-seconds field.

Two alarm functions are provided:

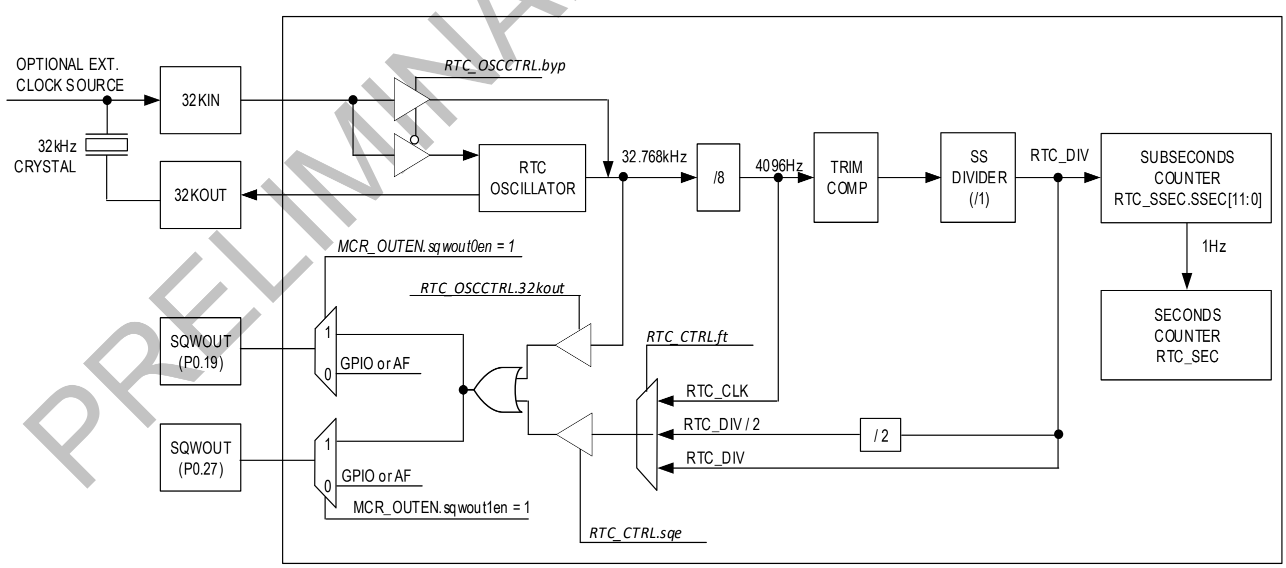
1. A programmable time-of-day alarm provides a single event, alarm timer using the *RTC_TODA* alarm register, *RTC_SEC* register, and *RTC_CTRL.ade* field.
2. A programmable sub-second provides a recurring alarm using the *RTC_SSECA* and *RTC_CTRL.ase* field

The RTC is powered in the always-on domain or if applicable, while there is a valid voltage on the V_{COREA} device pin.

Disabling the RTC stops incrementing *RTC_SSEC*, *RTC_SEC*, and the internal RTC sub-second counter, but preserves their current value. The 32kHz oscillator is not affected by the *RTC_CTRL.rtce* field.

The RTC increments the *RTC_TRIM.vrtc_tmr* field every 32 seconds while the RTC is enabled.

Figure 15-1: MAX78000 RTC Block Diagram (12-bit Sub-Second Counter)



15.2 Instances

One instance of the RTC peripheral is provided. The RTC counter and alarm registers are shown in *Table 15-1: MAX78000 RTC Counter and Alarm Registers*.

Table 15-1: MAX78000 RTC Counter and Alarm Registers

Field	Length	Counter Increment	Minimum	Maximum	Description
RTC_SEC	32	1s	1s	136 yrs	Seconds Counter Register
RTC_SSEC	12	244µs (1/4kHz)	244µs	1s	Sub-Seconds Counter Register
RTC_TODA	20	1s	1s	12 days	Time-of-Day Alarm Register
RTC_SSECA	32	244 µs (1/4kHz)	244 µs	12 days	Sub-Second Alarm Register

15.3 Register Access Control

Access protection mechanisms prevent software from accessing critical registers and fields while RTC while hardware is updating them. Monitoring the *RTC_CTRL.busy* and *RTC_CTRL.rdy* fields allows software to determine when it is safe to write to registers and when registers will return valid results.

Table 15-2: RTC Register Access

Register	Field	Read Access	Write Access	Busy = 1 during write	Description
RTC_SEC	All	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	Y	Seconds Counter Register
RTC_SSEC	ssec	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	*RTC_CTRL.busy* = 0 *RTC_CTRL.rdy* = 1†	Y	Sub-Seconds Counter Register
RTC_TODA	All	Always	• *RTC_CTRL.busy* = 0 • *RTC_CTRL.ade* = 0 • *RTC_CTRL.rtce = 0*	Y	Time-of-Day Alarm Register
RTC_SSECA	All	Always	• *RTC_CTRL.busy* = 0 • *RTC_CTRL.ase* = 0 • *RTC_CTRL.rtce = 0*	Y	Sub-Second Alarm Register
RTC_TRIM	All	Always	• *RTC_CTRL.busy* = 0 • *RTC_CTRL.we* = 1	Y	Trim Register
RTC_OSCCTRL	All	Always	*RTC_CTRL.we* = 1	N	Oscillator Control Register

Register	Field	Read Access	Write Access	Busy = 1 during write	Description
RTC_CTRL	rtce	Always	RTC_CTRL.busy = 0 RTC_CTRL.we = 1	Y	RTC Enable field
	All other bits	Always	††	Y	

- † See *RTC_SEC and RTC_SSEC Read Access Control* for details.
- †† See *RTC_CTRL.busy* for limitations on specific bits.

15.3.1 RTC_SEC and RTC_SSEC Read Access Control

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

Alternatively, the software can set *RTC_CTRL.acre* to 1 to allow asynchronous reads of both the *RTC_SEC* and *RTC_SSEC* registers.

Software can use three methods to ensure valid results when reading *RTC_SEC* and *RTC_SSEC*:

1. Software clears *RTC_CTRL.rdy* to 0. Software polls *RTC_CTRL.rdy* until it reads 1 before reading the registers. The software has approximately 120µs to read the *RTC_SEC* and *RTC_SSEC* registers to ensure accuracy.
2. Set the *RTC_CTRL.rdye* field to 1 to generate an RTC interrupt when the next update cycle is complete and hardware sets *RTC_CTRL.rdy* to 1. RTC interrupt must be serviced and *RTC_SEC* and/or *RTC_SSEC* registers must be read while *RTC_CTRL.rdy* = 1 to ensure accuracy. This avoids the software overhead associated with polling to observe the state of *RTC_CTRL.rdy*.
3. Set *RTC_CTRL.acre* to 1 to allow asynchronous reads of both the *RTC_SEC* and *RTC_SSEC* registers. Multiple consecutive reads of *RTC_SEC* and *RTC_SSEC* must be executed until two consecutive reads are identical to ensure data accuracy.

15.3.2 RTC Write Access Control

The read-only status field *RTC_CTRL.busy* is set to 1 by hardware following a software instruction that writes to specific registers. The bit remains 1 while the software updates are being synchronized into the RTC. Software should not write to any of the registers until 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 value stored in the alarm register. The sub-second interval alarm provides an auto-reload timer that is 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 lower 20 bits of the *RTC_SEC* seconds count register. This allows programming the time-of-day-alarm to any future value between 1 second and 12 days relative to the current time with a resolution of 1 second. You must disable the time-of-day alarm before changing the *RTC_TODA .tod* field.

When the alarm occurs, a single event sets the Time-of-Day Alarm Interrupt Flag (*RTC_CTRL.aldf*) to 1.

Setting the *RTC_CTRL.aldf* bit to 1 in software results in an interrupt request to the processor if the Alarm Time-of-Day Interrupt Enable (*RTC_CTRL.ade*) bit is set to 1, and the RTC's system interrupt enable is set.

15.4.2 Sub-Second Alarm

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

You must disable the sub-second interval alarm, *RTC_CTRL.ase*, prior to changing the interval alarm value, *RTC_SSECA*.

The delay (uncertainty) associated with enabling the sub-second alarm is up to one period of the sub-second clock. This uncertainty is propagated to the first interval alarm. Thereafter, if the interval alarm remains enabled, the alarm triggers after each sub-second interval as defined without the first alarm uncertainty because the sub-second alarm is an auto-reload timer. Enabling the sub-second alarm with the sub-second alarm register set to 0 (*RTC_SSECA* = 0) results in the maximum sub-second alarm interval.

15.4.3 RTC Interrupt and Wakeup Configuration

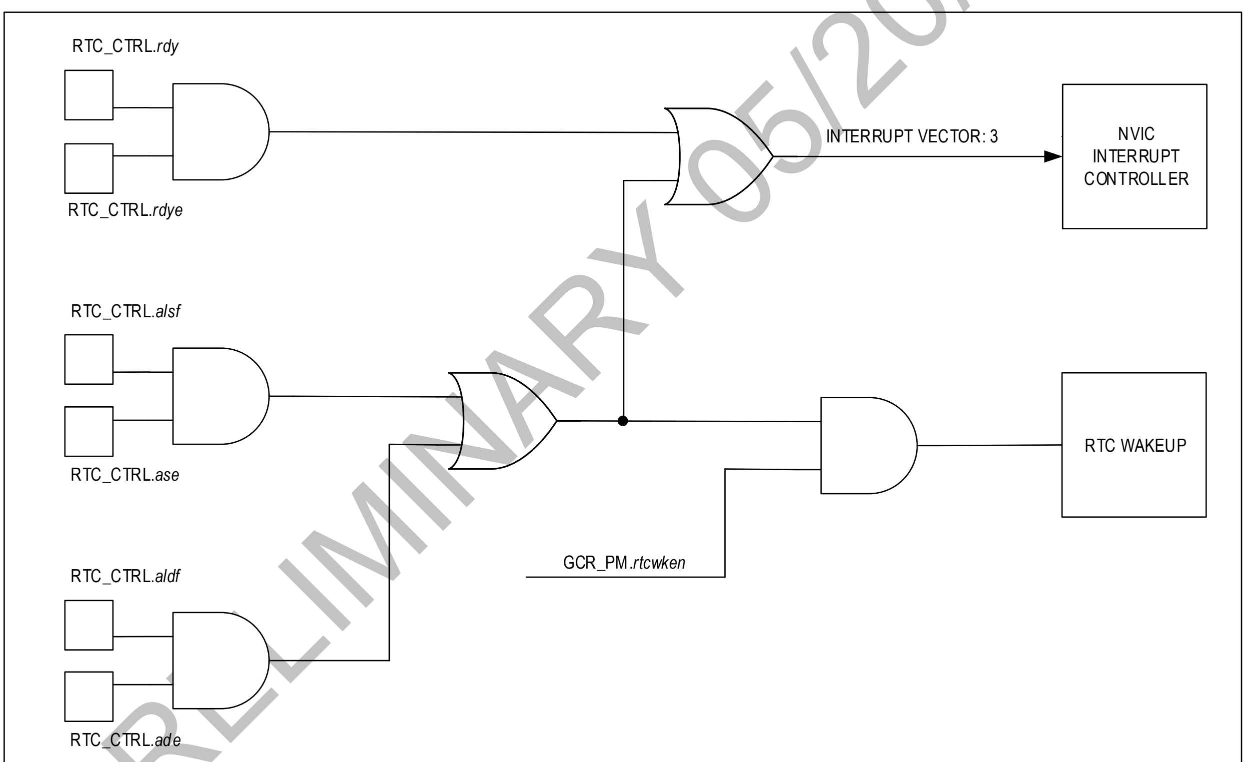
The following are a list of conditions that, when enabled, can generate an RTC interrupt.

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 wakeup source for exiting the following low power modes:

1. BACKUP
2. DEEP SLEEP
3. UPM
4. SLEEP

Figure 15-2: RTC Interrupt/Wakeup Diagram Wakeup Function



Use this procedure to enable the RTC as a wakeup source:

1. Software configures the RTC interrupt enable bits so one or more interrupt conditions will generate an RTC interrupt.
2. Create a RTC IRQ handler function and register the address of the RTC IRQ handler using the NVIC.
3. Software sets *GCR_PM.rtcwken* to 1 to enable the system wakeup for by the RTC.
4. Software places the device into the desired low power mode. Refer to section Operating Modes for details on entering DEEPSLEEP or BACKUP mode.

15.4.4 Square Wave Output

The RTC can output a 50% duty cycle square wave signal derived from the 32kHz oscillator on a selected device pin. Refer to *Table 15-3:* for the device pins, frequency options, and control fields specific to this device. Frequencies noted as compensated are used during the RTC frequency calibration procedure because they incorporate the frequency adjustments provided by the digital trim function.

Table 15-3: MAX78000 RTC Square Wave Output Configuration

Function	Option	Control Field
Output Pin†	P0.19: 32KCAL	**Error! Reference source not found.**.sqwout0en = 1
Frequency Select	1Hz (Compensated)	*RTC_OSCCTRL*.*freq_sel* = 0b00
	512Hz (Compensated)	*RTC_OSCCTRL*.*freq_sel* = 0b01
	4kHz	*RTC_OSCCTRL*.*freq_sel* = 0b10
	32kHz	*RTC_OSCCTRL*.*freq_sel* = 0bxx *RTC_OSCCTRL*.32*k_out* = 1
Output Enable	1Hz (Compensated)	*RTC_CTRL*.*sqe* = 1 *RTC_OSCCTRL*.32*k_out* = 0
	512Hz (Compensated)	*RTC_CTRL*.*sqe* = 1 *RTC_OSCCTRL*.32*k_out* = 0
	4kHz	*RTC_CTRL*.*sqe* = 1 *RTC_OSCCTRL*.32*k_out* = 0
	32kHz	*RTC_OSCCTRL*.32*k_out* = 1

†Note: The square wave output can be enabled on more than one pin if desired.

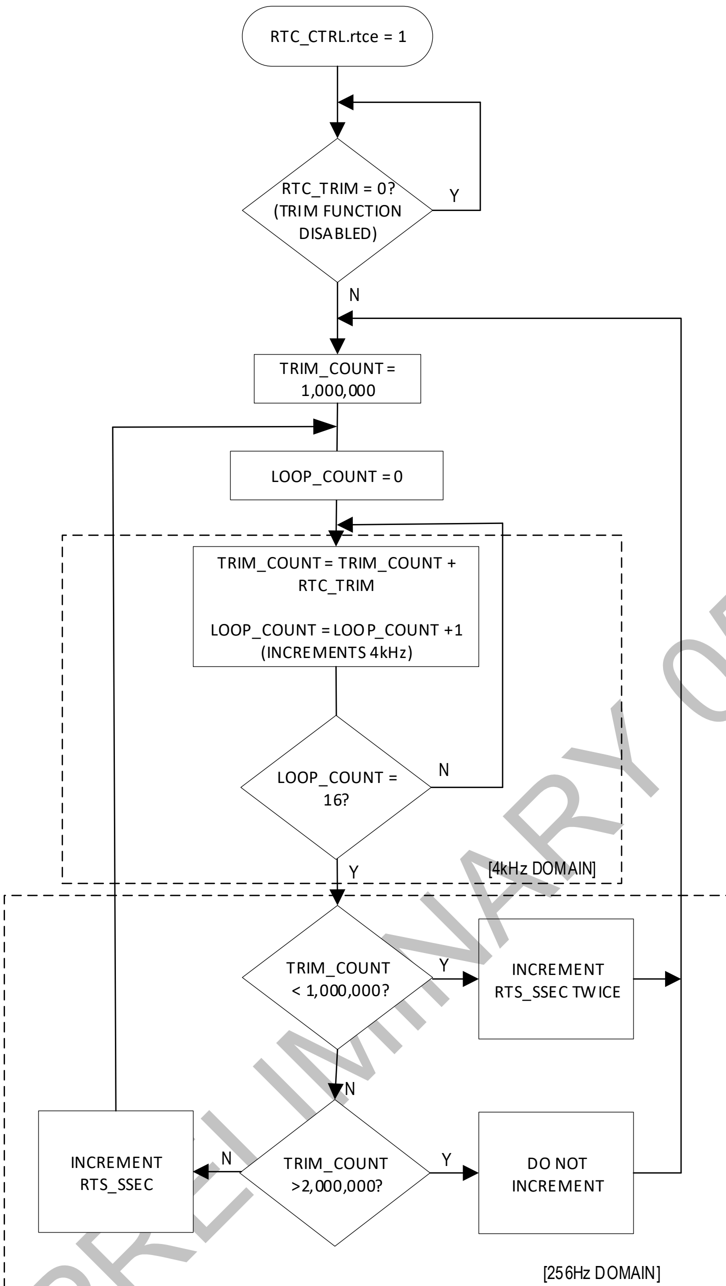
Use the following procedure to generate the square wave:

1. Software configures the fields shown in *Table 15-3:* to select the desired frequency.
2. If more than one output pin is available, software configures the fields shown in *Table 15-3: MAX78000 RTC Square Wave Output* Configuration to select the output pin.
3. Software configures the fields shown in *Table 15-3:* to enable the square wave output.

15.5 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 which increments the sub-second register based on a user-supplied, 2's compliment value in the *RTC_TRIM* register as shown in *Figure 15-3: Internal Implementation of Digital Trim,* 4kHz.

Figure 15-3: Internal Implementation of Digital Trim, 4kHz



Complete the following steps to perform an RTC calibration:

15.5.1 If not already, software configures and enables one of the compensated calibration frequencies as described in section Square Wave Output.

1. Measure the frequency on the square wave output pin and compute the deviation from an accurate reference clock.
2. Software clears *RTC_CTRL*.*rdy* to 0.
3. Wait for *RTC_CTRL*.*rdy* = 1 by:
   a. Software sets *RTC_CTRL*.*rdye* to 1 to generate an interrupt when *RTC_CTRL*.*rdy* = 1, or
   b. Software polls *RTC_CTRL*.*rdy* until the field = 1.
4. Software polls until *RTC_CTRL*.*busy* = 0 to make sure any previous operations are complete
5. Software sets *RTC_CTRL*.*we* to 1 to allow access to *RTC_TRIM*.
6. Software writes to *RTC_TRIM* register as desired to correct for measured inaccuracy.
7. Hardware clears *RTC_CTRL*.*busy* =0.
8. Software clears *RTC_CTRL*.*we* to 0.

Repeat the process as needed until the desired accuracy is achieved

15.6 Registers

See *Table 2-3: APB Peripheral Base* Address Map for this peripheral/module's base address. If multiple instances are provided, each will have a unique base address. All fields in this peripheral are reset on POR only; refer to the field description for details.

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

15.6.1 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 (12-bit)

RTC Sub-Seconds Counter				RTC_SSEC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:12	-	R/W	0	**Reserved for Future Use** Do not modify this field.	

RTC Sub-Seconds Counter				RTC_SSEC	[0x0004]
Bits	**Field**	**Access**	**Reset**	**Description**	
11:0	ssec	R/W	0	**Sub-Seconds Counter (12-bit)** *RTC_SEC* increments when this field rolls from 0x0FFF to 0x0000	

Table 15-7: RTC Time-of-Day Alarm Register

RTC Time-of-Day Alarm				RTC_TODA	[0x0008]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:20	-	R/W	0	**Reserved for Future Use** Do not modify this field.	
19:0	tod_alarm	R/W	0	**Time-of-Day Alarm** Sets the time-of-day alarm from 1 second up to 12-days. When this field matches *RTC_SEC*[19:0], an RTC system interrupt is generated.	

Table 15-8: RTC Sub-Second Alarm Register

RTC Sub-Second Alarm				RTC_SSECA	[0x000C]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:0	ssec_alarm	R/W	0	**Sub-second Alarm (4KHz)** Sets the starting and reload value of the internal sub-second alarm counter. The internal counter increments and generates an alarm when the internal counter rolls from 0xFFFF FFFF to 0x0000 0000.	

Table 15-9: RTC Control Register

RTC Control Register				RTC_CTRL	[0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**	
31:16	-	R/W	0	**Reserved for Future Use** Do not modify this field.	
15	we	R/W	0*	**Write Enable** This field controls access to the *RTC_TRIM* register, the RTC enable (*RTC_CTRL.rtce*) fields. 1: Writes to the *RTC_TRIM* register and the *RTC_CTRL.rtce* field are allowed. 0: Writes to the *RTC_TRIM* register and the *RTC_CTRL.rtce* field are ignored. **Note: Reset on System Reset, Soft Reset, and GCR_RSTR0.rtc assertion.*	
14	acre	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 will require software interaction to ensure data accuracy.	
13	-	R/W	0	**Reserved for Future Use** Do not modify this field.	
12:11	-	R/W	0	**Reserved for Future Use** Do not modify this field.	

RTC Control Register				RTC_CTRL [0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
10:9	ft	R/W	0*	**Frequency Output Select** Selects the RTC-derived frequency to output on the square wave output pin. See *Table 15-3: MAX78000 RTC Square Wave Output* Configuration for configuration details. 0b00: 1Hz (Compensated) 0b01: 512Hz (Compensated) 0b1x: 4kHz **Note: Reset on POR only.*
8	sqe	R/W	0*	**Square Wave Output Enable** Enables the square wave output. *Table 15-3: MAX78000 RTC Square Wave Output* Configuration 0: Disabled. 1: Enabled. *Note: Reset on POR only.
7	alsf	R/W	0*	**Sub-second Alarm Interrupt Flag** This interrupt flag is set when a sub-second alarm condition occurs. This flag is a wake-up source for the processor. 0: No sub-second alarm pending. 1: Sub-second interrupt pending. *Note: Reset on POR only.
6	aldf	R/W	0*	**Time-of-Day Alarm Interrupt Flag** This interrupt flag is set by hardware when a time-of-day alarm occurs. 0: No Time-of-Day alarm interrupt pending. 1: Time-of-day interrupt pending. *Note: Reset on POR only.
5	rdye	R/W	0*	**RTC Ready Interrupt Enable** 0: Disabled. 1: Enabled. *Note: Reset on System Reset, Soft Reset, and GCR_RSTR0.rtc assertion.
4	rdy	R/W0	0*	**RTC Ready** This bit is set to 1 for 120µs by hardware once a hardware update of the *RTC_SEC* and *RTC_SSEC* registers has occurred. Software should read *RTC_SEC* and *RTC_SSEC* while this hardware bit is set to 1. Software can clear this bit at any time. An RTC interrupt will be generated if *RTC_CTRL.rdye* = 1. 0: Software reads of *RTC_SEC* and *RTC_SSEC* are invalid. 1: Software reads of *RTC_SEC* and *RTC_SSEC* are valid. *Note: Reset on System Reset, Soft Reset, and GCR_RSTR0.rtc assertion.

RTC Control Register				RTC_CTRL [0x0010]
Bits	**Field**	**Access**	**Reset**	**Description**
3	busy	RO	0*	**RTC Busy Flag** This field is set to 1 by hardware to indicate a register update is in progress when software writes to: *RTC_SEC* register *RTC_SSEC* register *RTC_TRIM* register The following bits cannot be written if this field is set: *RTC_CTRL.rtce* *RTC_CTRL.ade* *RTC_CTRL.ase* *RTC_CTRL.rdye* *RTC_CTRL.aldf* *RTC_CTRL.alsf* *RTC_CTRL.sqe* *RTC_CTRL.ft* *RTC_CTRL.acre* This field is automatically cleared by hardware when the update is complete. Software should poll this field for 0 after changing a RTC register, prior to making any other RTC register modifications. 0: RTC not busy. 1: RTC busy. **Note: Reset on POR only.*
2	ase	R/W	0*	**Sub-Second Alarm Interrupt Enable** Check the *RTC_CTRL.busy* flag after writing to this field to determine when the RTC synchronization is complete. 0: Disable. 1: Enable. **Note: Reset on POR only.*
1	ade	R/W	0*	**Time-of-Day Alarm Interrupt Enable** Check the *RTC_CTRL.busy* flag after writing to this field to determine when the RTC synchronization is complete. 0: Disable. 1: Enable. **Note: Reset on POR only.*
0	rtce	R/W	0*	**Real-Time Clock Enable** The RTC write enable (*RTC_CTRL.we*) bit must be set and RTC Busy (*RTC_CTRL.busy*) must read 0 before writing to this field. After writing to this bit, check the *RTC_CTRL.busy* flag for 0 to determine when the RTC synchronization is complete. 0: Disabled. 1: Enabled. **Note: Reset on POR only.*

Table 15-10: RTC 32KHz Oscillator Digital Trim Register

RTC 32KHz Oscillator Digital Trim			RTC_TRIM	[0x0014]
Bits	**Field**	**Access**	**Reset**	**Description**
31:8	vrtc_tmr	R/W	0*	**VRTC Time Counter** Hardware increments this field every 32s 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 for Future Use** Do not modify this field.
5	32kout	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 be driven on the 32KIN pin. 0: Disable bypass. RTC time base is external 32kHz crystal. 1: Enable bypass. RTC time base is external square wave driven on 32KIN. **Note: Reset on POR only.*
3:0	-	R/W	0b1001	**Reserved for Future Use** Software must not modify this field from its current value.

16. Watchdog Timer (WDT)

The watchdog timer (WDT) protects against corrupt or unreliable software, power faults, and other system-level problems which may place the IC into an improper operating state. Application software must periodically write a special 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. In the event the application cannot regain normal operation, as a last resort the watchdog timer can generate a system reset.

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

Each time the application performs a reset, as early as possible in the application software, the peripheral control register should be examined to determine if the reset was cause by at WDT late reset event (or WDT early reset event if the window function is supported). If so, software should take the desired action as part of its restart sequence.

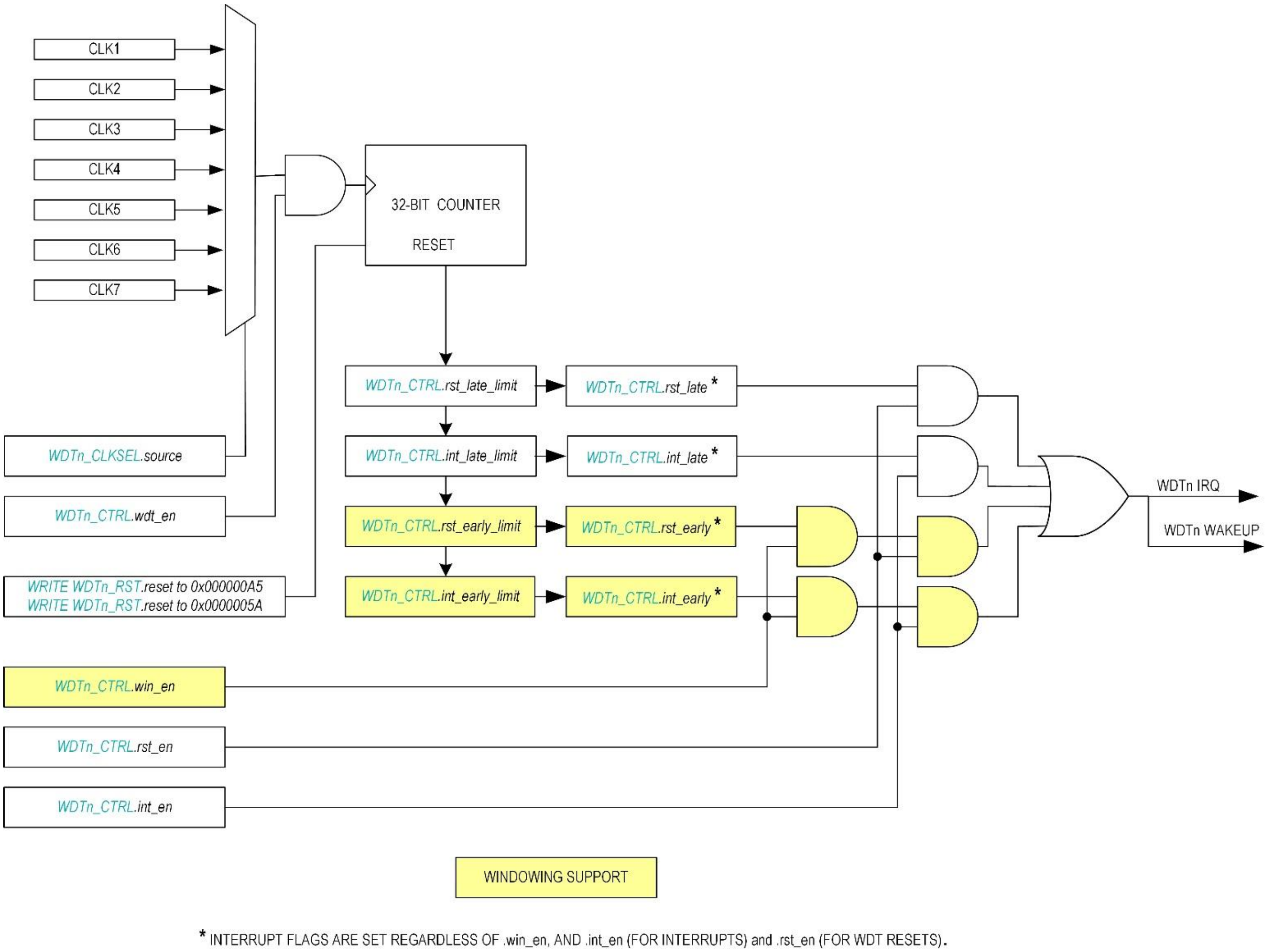
The WDT is a critical safety feature, and most fields are reset on POR or system reset events only.

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^{1327} time-base ticks
- New register to read the WDT counter register, simplifying code development

Figure 16-1: Windowed Watchdog Timer Block Diagram shows the block diagram of the WDT.

Figure 16-1: Windowed Watchdog Timer Block Diagram



16.1 Instances

Table 16-1: MAX78000 WDT Instances Summary shows the peripheral instance, the available clock sources, and indicates which instances support the windowed watchdog functionality.

Table 16-1: MAX78000 WDT Instances Summary

Instance	Register Access Name	Window Support	External Clock	CLK1	CLK2	CLK3
WDT0	WDT0	Yes	N/A	PCLK	IBRO	-
LPWDT0	WDT1	Yes	N/A	IBRO	ERTC0	INRO

16.2 Usage

When enabled, *WDTn_CNT.count* increments once every t_{WDTCLK} period. During correct operation, the WDT will periodically execute the feed sequence and reset *WDTn_CNT.count* to 0x0000 0000 within the target window.

The upper and lower limits of the target window are user-configurable to accommodate different applications and non-deterministic execution times within an application.

The WDT can generate interrupt and/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 may have temporarily delayed the execution of the feed sequence, so the event can be diagnosed in an interrupt routine and control returned to the system. When the WDT feed sequence occurs much earlier than expected or not at all, a reset event can be generated that will force 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 will attempt to regain program control by vectoring to the dedicated WDT 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 will generate a late reset event which sets the WDT late reset flag and generates a system interrupt.

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

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

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

16.3 WDT Feed Sequence

The WDT feed sequence protects the system against unintentional altering of the WDT count and unintentional enabling or disabling of the timer itself.

Two consecutive write instructions to the *WDTn_RST*.*reset* field are required to reset the *WDTn_CNT*.*count* = 0. Global interrupts should be disabled immediately before, and reenabled after the writes, to ensure both writes to the *WDTn_RST*.*reset* field complete without interruption.

The feed sequence must also be performed immediately before enabling the WDT, to prevent an accidental triggering of the reset or interrupt as soon as the timer is enabled. There is no timed access window for these write operations; the operations can be separated by any length of time as long as they occur in the required sequence

1. Disable interrupts.
2. In consecutive write operations:
   a. Write *WDTn_RST*.*reset*: 0x000000A5
   b. Write *WDTn_RST*.*reset*: 0x0000005A
3. If desired, enable or disable the timer
4. Re-enable interrupts

16.4 WDT Events

Multiple events are supported as shown in *Table 16-2: MAX78000 WDT Event* Summary. The corresponding event flag is set when the event occurs.

Typically the system is configured such that the late interrupt events will occur prior to the late reset events, and early interrupts will occur when the feed sequence has the least error from the target time, prior to the early reset events.

The event flags will be set even if the corresponding interrupt or reset enable flags regardless of the state of the corresponding enable fields. This includes the early interrupt and early event flags, even if *WDTn_CTRL.win_en* = 0.

Software must clear the all event flags before enabling the timers.

Table 16-2: MAX78000 WDT Event Summary

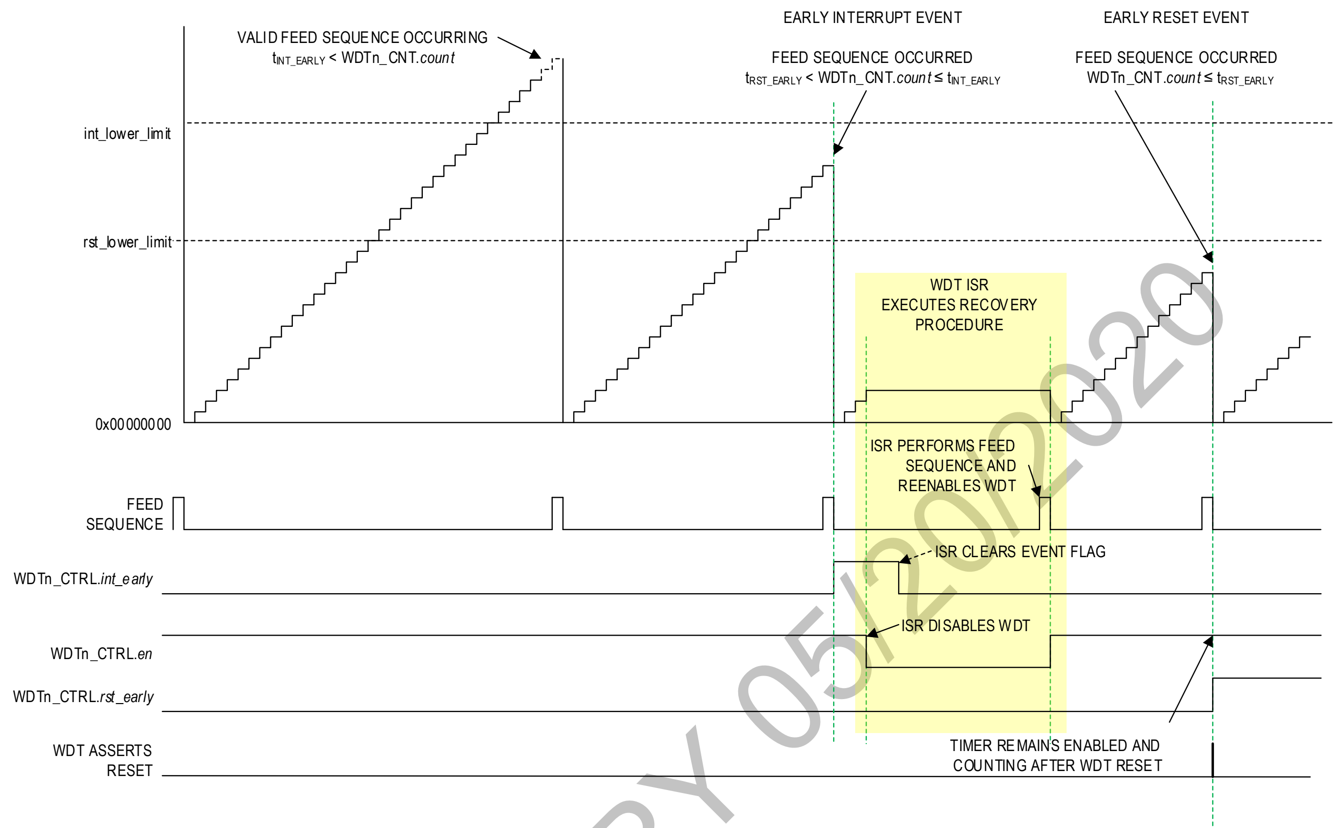
Event	Condition	Local Interrupt Event Flag	Local Interrupt Event Enable
Early Interrupt	Feed sequence occurs while *WDTn_CTRL.rst_early_val* ≤ *WDTn_CNT.count* < *WDTn_CTRL.int_early_val* *WDTn_CTRL.win_en = 1*	*WDTn_CTRL.int_early*	*WDTn_CTRL.wdt_int_en*
Early Reset	Feed sequence occurs while *WDTn_CNT.count* < *WDTn_CTRL.rst_early_val* *WDTn_CTRL.win_en = 1*	*WDTn_CTRL.rst_early*	*WDTn_CTRL.wdt_rst_en*
Interrupt Late	*WDTn_CNT.count* = *WDTn_CTRL.int_late_val*	*WDTn_CTRL.int_late*	*WDTn_CTRL.wdt_int_en*
Reset Late	*WDTn_CNT.count* = *WDTn_CTRL.rst_late_val*	*WDTn_CTRL.rst_late*	*WDTn_CTRL.wdt_rst_en*
Timer Enabled	*WDTn_CTRL.clkrdy* 0 → 1	No event flags are set by a timer enabled event	

16.4.1 WDT Early Reset

The early reset event occurs if software performs the WDT feed sequence while *WDTn_CNT.count* < *WDTn_CTRL.rst_late_val* threshold as shown in *Table 16-2: MAX78000 WDT Event* Summary.

Figure 16-2: WDT Early Interrupt and Reset Event Sequencing Details shows the sequencing details associated with an early reset event.

Figure 16-2: WDT Early Interrupt and Reset Event Sequencing Details



The following occurs when a WDT early reset event occurs:

1. Hardware sets *WDTn_CTRL.rst_early* to 1.
2. Hardware initiates a system reset:
3. Hardware resets *WDTn_CNT.count* to 0x0000_0000 during the reset event.
4. The *WDTn_CTRL.en* field is unaffected by a system reset. The WDT continues incrementing.
5. The *WDTn_CTRL.rst_early* field is unaffected by a system reset.

16.4.2 WDT Early Interrupt

The early interrupt event occurs if software performs the WDT feed sequence while *WDTn_CTRL.rst_early_val* <= *WDTn_CNT.count* < *WDTn_CTRL.int_early_val* as shown in *Table 16-2: MAX78000 WDT Event* Summary. *Figure 16-2: WDT Early Interrupt and Reset Event Sequencing* Details shows the sequencing details associated with an early reset event. shows the sequencing details associated with an early interrupt event, including the required functions performed by the WDT ISR.

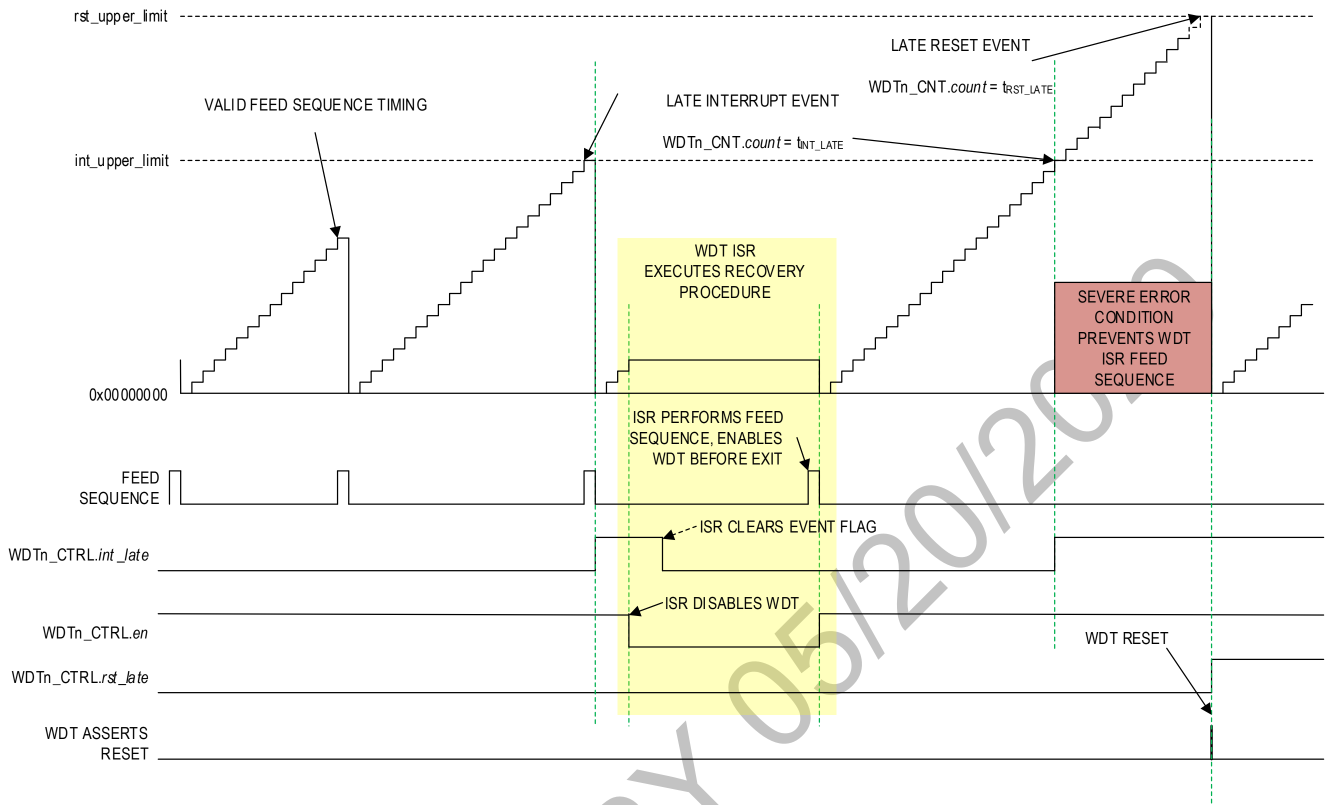
The following occurs when WDT late interrupt event occurs:

1. Hardware sets *WDTn_CTRL.int_late* to 1.
2. Hardware initiates an interrupt if enabled.

16.4.3 WDT Late Reset

The late reset event occurs if the counter increments to the point where *WDTn_CNT.count* = late_reset threshold as shown in *Table 16-2: MAX78000 WDT Event* Summary. *Figure 16-3: WDT Late Interrupt and Reset Event Sequencing* Details shows the sequencing details associated with a late reset event.

Figure 16-3: WDT Late Interrupt and Reset Event Sequencing Details



The following occurs when a WDT late reset event occurs:

1. Hardware sets *WDTn_CTRL.rst_late* to 1.
2. Hardware initiates a system reset:
3. Hardware resets *WDTn_CNT.count* to 0x0000_0000 during the reset event.
4. The *WDTn_CTRL.en* field is unaffected by a system reset. The WDT continues incrementing.
5. The *WDTn_CTRL.rst_late* field is unaffected by a system reset.

16.4.4 WDT Late Interrupt

The late reset event occurs if the counter increments to the point where *WDTn_CNT.count* = late_reset threshold as shown in *Table 16-2: MAX78000 WDT Event* Summary. *Figure 16-3: WDT Late Interrupt and Reset Event Sequencing* Details shows the sequencing details associated with an late interrupt event, including the required functions performed by the WDT ISR.

The following occurs when WDT late interrupt event occurs:

1. Hardware sets *WDTn_CTRL.int_late* to 1.
2. Hardware initiates an interrupt if enabled.

16.5 Initializing the WDT

The full procedure for configuring the WDT is shown below:

1. Execute the WDT feed sequence and disable the WDT:
2. Disable global interrupts
3. Write *WDTn_RST.reset* to 0x000000A5.
4. Write *WDTn_RST.reset* to 0x0000005A. Hardware will reset *WDTn_CNT.count* = 0x00000000.
5. Set *WDTn_CTRL.en* to 0 to disable the WDT.
6. Verify the peripheral is disabled before proceeding:
7. Poll *WDTn_CTRL.clkrdy* until it reads 1, or
8. Set *WDTn_CTRL.clk_rdy_ie* = 1 to generate a WDT enabled interrupt event.
9. Re-enable global interrupts.
10. Configure *WDTn_CLKSEL.source* to select the clock source.
11. Configure the traditional/legacy thresholds:
12. Configure *WDTn_CTRL.int_late_thd* to the desired threshold for the WDT late interrupt event.
13. Configure *WDTn_CTRL.rst_late_val* to the desired threshold for the WDT late reset event.
14. 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.
15. 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 will be generated by both a WDT late interrupt event and also a WDT early interrupt event.
16. 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 will be generated by a WDT late reset event and also a WDT early reset event.
17. Execute the WDT feed sequence and enable the WDT:
18. Disable global interrupts
19. Write *WDTn_RST.reset* to 0x000000A5.
20. Write *WDTn_RST.reset* to 0x0000005A. Hardware will reset *WDTn_CNT.count* = 0x00000000.
21. Set *WDTn_CTRL.en* to 1 to enable the WDT.
22. Verify the peripheral is enabled before proceeding:
23. Poll *WDTn_CTRL.clkrdy* until it reads 1, or
24. Set *WDTn_CTRL.clkrdy_ie* = 1 to generate a WDT enabled event interrupt.
25. Re-enable global interrupts.

16.6 Resets

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

16.7 Using the WDT as a Long-Interval Timer

One application of the WDT is as a very long interval timer in ACTIVE mode. The timer can be configured to generate a WDT late interrupt event for as long as 2^{32} periods of the selected watchdog clock source. The WDT should not be enabled to generate WDT reset events in this application.

16.8 Using the WDT as a Long-Interval Wakeup Timer

Another application of the WDT is as a very long interval wakeup source from SLEEP mode. The timer can be configured to generate a WDT wakeup event for as long as 2^{32} periods of the selected watchdog clock source. Perform the following procedure to initialize the WDT for this purpose:

16.9 Registers

See *Table 2-3: APB Peripheral Base* Address Map 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 16-3: WDT Register* Summary. 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 2-1. Field Access Definitions* for an explanation of the read and write access of each field. Unless specified otherwise, all fields are reset on a system reset, soft reset, POR, and the peripheral-specific resets.

Table 16-3: 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

16.9.1 Register Details

Table 16-4: 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 will be set even if *WDTn_CTRL.win_en* = 0 or *WDTn_CTRL.wdt_rst_en* = 0. The user will have to clear the flag appropriately in case of carried over flags from prior operations. 0: Watchdog did not cause reset event. 1: Watchdog reset occurred after *WDTn_CTRL.rst_early_val*.	
30	rst_early	R/W	0	**Reset Early Event** A watchdog reset event occurred before the time specified in *WDTn_CTRL.rst_early_val*. This flag will be set even if *WDTn_CTRL.win_en* = 0 or *WDTn_CTRL.wdt_rst_en* = 0. The user will have to clear the flag appropriately in case of carried over flags from prior operations. 0: Watchdog did not cause reset event. 1: Watchdog reset occurred before *WDTn_CTRL.rst_early_val*.	
29	win_en	R/W	0	**Window Function Enable** 0: Disabled. WDT recognizes interrupt late and reset late events, supporting legacy implementations. 1: Enabled	

WDT Control				WDTn_CTRL	[0x0000]
Bits	**Name**	**Access**	**Reset**	**Description**	
28	clkrdy	R	0	**Clock Status** This field is cleared by hardware when software changes the state of *WDTn_CTRL.en*. Hardware sets the field to 1 when the change to the requested enable/disable is complete. 0: WDT clock is off 1: WDT clock is on	
27	clkrdy_ie	R/W	0	**Clock Switch Ready Interrupt Enable** This interrupt prevents the user from needing to poll the *WDTn_CTRL.clkrdy* all the time. When *WDTn_CTRLclkrdy* goes from high to low, the interrupt signals the transition is complete. 0: Disabled 1: Enabled	
26:24	-	RO	0	**Reserved**	
23:20	rst_early_val	R/W	0	**Reset Early Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 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: $2^{16} \times t_{WDTCLK}$	
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: $2^{16} \times t_{WDTCLK}$	
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 was performed earlier than the time determined by the *WDTn_CTRL.int_early_thd* field. This flag will be set even if *WDTn_CTRL.win_en* = 0. 0: No interrupt event 1: Interrupt event occurred. Generates a WDT IRQ if *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. Generates a WDT IRQ if *WDTn_CTRL.wdt_int_en* = 1.	
8	en	R/W	0	**WDT Enable** This field enables/disables the WDTCLK into the peripheral. *WDTn_CTRL.count* holds its value while the WDT is disabled. The WDT feed sequence must be performed immediately before any change to this field. 0: Disabled 1: Enabled	
7:4	rst_late_val	R/W	0	**Reset Late Event Threshold** 0x0: $2^{31} \times t_{WDTCLK}$ 0x1: $2^{30} \times t_{WDTCLK}$ 0x2: $2^{29} \times t_{WDTCLK}$ 0x3: $2^{28} \times t_{WDTCLK}$ 0x4: $2^{27} \times t_{WDTCLK}$ 0x5: $2^{26} \times t_{WDTCLK}$ 0x6: $2^{25} \times t_{WDTCLK}$ 0x7: $2^{24} \times t_{WDTCLK}$ 0x8: $2^{23} \times t_{WDTCLK}$ 0x9: $2^{22} \times t_{WDTCLK}$ 0xA: $2^{21} \times t_{WDTCLK}$ 0xB: $2^{20} \times t_{WDTCLK}$ 0xC: $2^{19} \times t_{WDTCLK}$ 0xD: $2^{18} \times t_{WDTCLK}$ 0xE: $2^{17} \times t_{WDTCLK}$ 0xF: $2^{16} \times t_{WDTCLK}$	

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: $2^{16} \times t_{WDTCLK}$	

Table 16-5: WDT Reset Register

WDT Reset				WDTn_RST	[0x0004]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:8	-	RO	0	**Reserved** Do not modify this field.	
7:0	reset	R/W	0*	**Reset Watchdog Timer Count** Writing the WDT feed sequence, in two consecutive write instructions, to this register resets the internal counter to 0x0000_0000. 1. Write *WDTn_RST.reset*: 0x0000 00A5 2. Write *WDTn_RST.reset*: 0x0000 005A Writes to the *WDTn_CTRL*.en field, which enable or disable the WDT, must be the next instruction following the WDT feed sequence. **Note: This field is set to 0 on a POR and is not affected by other resets.*	

Table 16-6: 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** 0x0: PCLK 0x1: CLK1 (*See Table 16-1: MAX78000 WDT Instances* Summary) 0x2: CLK2 (*See Table 16-1: MAX78000 WDT Instances* Summary) 0x3: CLK3 (*See Table 16-1: MAX78000 WDT Instances* Summary) 0x4: CLK4 (*See Table 16-1: MAX78000 WDT Instances* Summary) 0x5: CLK5 (*See Table 16-1: MAX78000 WDT Instances* Summary) 0x6: CLK6 (*See Table 16-1: MAX78000 WDT Instances* Summary) 0x7: CLK7 (*See Table 16-1: MAX78000 WDT Instances* Summary) *Note: This field is only reset on a POR and unaffected by other resets.	

Table 16-7: WDT Count Register

WDT Count				WDTn_CNT	[0x000C]
Bits	**Name**	**Access**	**Reset**	**Description**	
31:0	count	R	0	**WDT Counter** The counter value for debug. This register is reset by system reset, as well as the watchdog feeding sequence. When the WDT clock is off, the feeding sequence generates an asynchronous reset of 1 PCLK width. When the WDT clock is on, the feeding sequence generates a synchronous reset that is handshake to the WDT clock domain.	

17. Wakeup Timer (WUT0)

The Wakeup Timer (WUT) is a unique instance of a 32-bit timer.

1. Timebase is 32.768kHz clock source
2. Programmable prescaler with values from 1 to 4096
3. Supports standard 32-bit timer modes
    a. One-Shot: Timer counts up to terminal value then halts.
    b. Continuous: Timer counts up to terminal value then repeats.
4. Independent interrupt

17.1 Basic Operation

The timer modes operate by incrementing the *WUTn_CNT.count* register. The *WUTn_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 timer period condition. The end of a timer period will always set the corresponding interrupt bit and can generate an interrupt, if enabled.

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

The timer clock frequency, f_{CNT_CLK}. is a divided version of the 32.768kHz RTC clock. The divisor/prescaler can be set from 1 to 4096 using the *WUTn_CTRL.pres3*:*WUTn_CTRL.pres* as shown in *Table 17-1: MAX78000 WUT Clock Period*:

Table 17-1: MAX78000 WUT Clock Period

pres3	pres	Prescaler	f_{CNT_CLK} (kHz)
0	0b000	1	32.768
0	0b001	2	16.384
0	0b010	4	8.192
0	0b011	8	4.096
0	0b100	16	2.048
0	0b101	32	1.024
0	0b110	64	0.512
0	0b111	128	0.256
1	0b000	256	0.128
1	0b010	512	0.064
1	0b011	1024	0.032
1	0b100	2048	0.016
1	0b101	4096	0.008
1	0b110	Reserved	Reserved

pres3	pres	Prescaler	f_{CNT_CLK} (kHz)
1	0b111	Reserved	Reserved

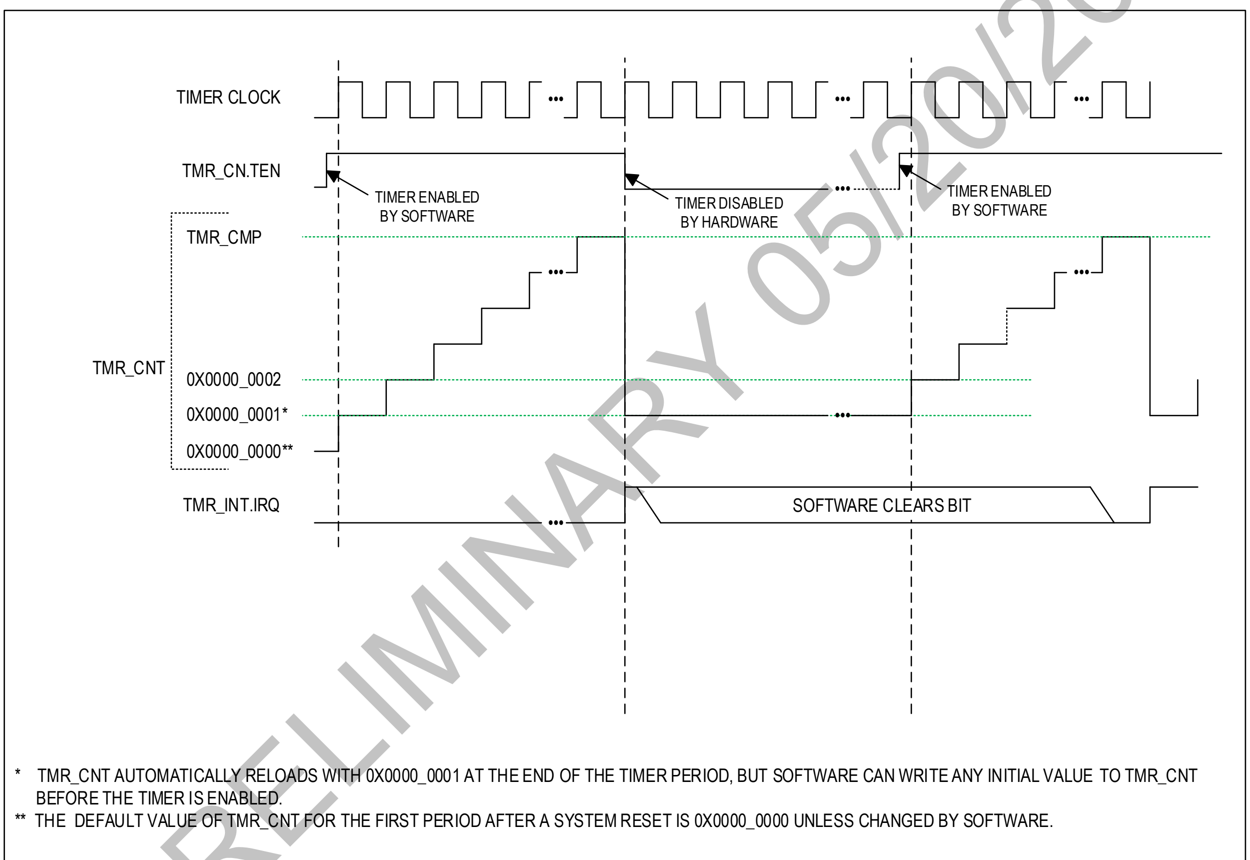
17.2 Timer Pin Functionality

The WUT does not have any timer pin functionality. Timer modes that modes that require external inputs or outputs are not implemented on this device.

17.3 One-Shot Mode (000b)

In One-shot mode the timer peripheral increments *WUTn_CNT.count* until it matches *WUTn_CMP* and then stops incrementing and disables the timer. In this mode, the timer must be re-enabled to start another one-shot mode event.

Figure 17-1: One-Shot Mode Diagram



17.3.1 One-Shot Mode Timer Period

The timer period ends on the timer clock following *WUTn_CNT.count* = *WUTn_CMP*.

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

1. *WUTn_CNT.count* is reset to 0x0000 0001.
2. The timer is disabled by setting *WUTn_CTRL.ten* = 0.
3. The timer interrupt bit *WUTn_INTR.irq_clr* will be set. An interrupt will be generated if enabled.

17.3.2 One-Shot Mode Configuration

Configure the timer for One-Shot mode by doing the following:

1. Set *WUTn_CTRL.ten* = 0 to disable the timer.
2. Set *WUTn_CTRL.tmode* to 000b to select One-shot mode.
3. Set *WUTn_CTRL.pres3*:*WUTn_CTRL.pres* to determines the timer period.
4. Enable the interrupt and set the interrupt priority.
5. Write an initial value to *WUTn_CNT.count*, if desired. This effects only the first period; subsequent timer periods always reset *WUTn_CNT.count* = 0x0000 0001.
6. Write the compare value to *WUTn_CMP*.
7. Set *WUTn_CTRL.ten* = 1 to enable the timer.

The timer period is calculated using the following equation:

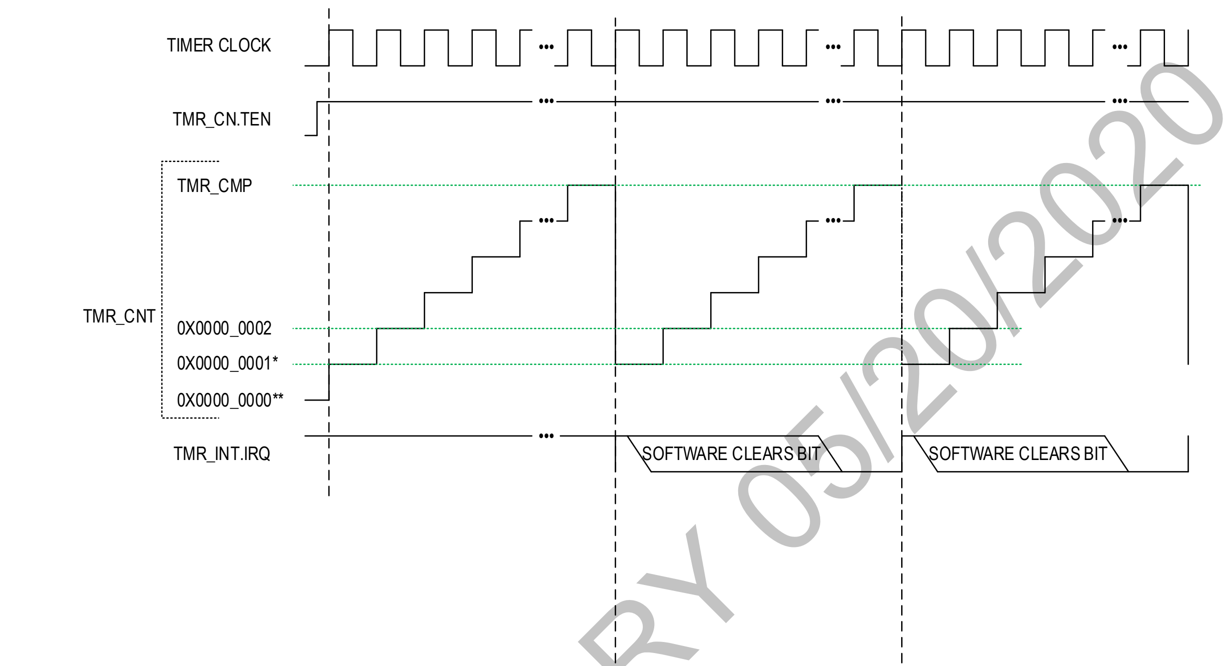
Equation 17-1: One-shot Mode Timer Period

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

17.4 Continuous Mode (001b)

In Continuous mode, the timer peripheral increments *WUTn_CNT.count* until it matches *WUTn_CMP*, resets *WUTn_CNT.count* to 0x0000 0001, and continues incrementing.

Figure 17-2: Continuous Mode Diagram



* TMR_CNT AUTOMATICALLY RELOADS WITH 0X0000_0001 AT THE END OF THE TIMER PERIOD, BUT SOFTWARE CAN WRITE ANY INITIAL VALUE TO TMR_CNT BEFORE THE TIMER IS ENABLED.

** THE DEFAULT VALUE OF TMR_CNT FOR THE FIRST PERIOD AFTER A SYSTEM RESET IS 0X0000_0000 UNLESS CHANGED BY SOFTWARE.

17.4.1 Continuous Mode Timer Period

The timer period ends on the timer clock following *WUTn_CNT.count* = *WUTn_CMP*.

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

1. *WUTn_CNT.count* is reset to 0x0000 0001. The timer remains enabled and continues incrementing.
2. The timer interrupt bit *WUTn_INTR.irq_clr* will be set. An interrupt will be generated if enabled.

17.4.2 Continuous Mode Configuration

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

1. Set *WUTn_CTRL.ten* = 0 to disable the timer.
2. Set *WUTn_CTRL.tmode* to 001b to select Continuous mode.
3. Set *WUTn_CTRL.pres3*:*WUTn_CTRL.pres* to determines the timer period.
4. Enable the interrupt and set the interrupt priority.
5. Write an initial value to *WUTn_CNT.count*, if desired. The initial value is only used for the first period; subsequent timer periods always reset the *WUTn_CNT.count* register to 1.
6. Write the compare value to *WUTn_CMP*.
7. Set *WUTn_CTRL.ten* to 1 to enable the timer.

The Continuous Mode Timer Period is calculated using *Equation 17-2*.

Equation 17-2: Continuous Mode Timer Period

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

17.5 Registers

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

See *Table 2-1. Field Access Definitions* 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 17-2: Timer Register Summary

Offset	Register Name	Description
[0x0000]	*WUTn_CNT*	Wakeup Timer Counter Register
[0x0004]	*WUTn_CMP*	Wakeup Timer Compare Register
[0x0008]	Reserved	
[0x000C]	*WUTn_INTFL*	Wakeup Timer Interrupt Register
[0x0010]	*WUTn_CTRL*	Wakeup Timer Control Register

17.5.1 Register Details

Table 17-3: Timer Count Register

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

Table 17-4: Timer Compare R egister

Timer Compare Register			**WUTn_CMP**	**[0x0004]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:0	compare	R/W	0	**Timer Compare Value** The value in this register is used as the compare value for the timer's count value. The 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 17-5: Timer Interrupt Register

Timer Interrupt Register			**WUTn_INTFL**	**[0x000C]**
Bits	**Name**	**Access**	**Reset**	**Description**
31:1	-	RO	0	**Reserved for Future Use** Do not modify this field from its default value.
0	irq_clr	RW	0	**Timer Interrupt** If set, this field indicates a timer interrupt condition occurred. Writing any value to this bit clears the timer's interrupt. 0: Timer interrupt is not active. 1: Timer interrupt occurred.

Table 17-6: Timer Control Register

<table>
<tr><th colspan="4">Timer Control Register</th><th>WUTn_CTRL</th><th>[0x0010]</th></tr>
<tr><th>Bits</th><th>Name</th><th>Access</th><th>Reset</th><th colspan="2">Description</th></tr>
<tr><td>31:13</td><td>-</td><td>RO</td><td>0</td><td colspan="2">Reserved
Do not modify this field</td></tr>
<tr><td>12:9</td><td>-</td><td>DNM</td><td>0</td><td colspan="2">Reserved Do Not Modify</td></tr>
<tr><td>8</td><td>pres3</td><td>R/W</td><td>0</td><td colspan="2">Timer Prescale Select MSB
See WUTn_CTRL.pres for details on this field’s usage.</td></tr>
<tr><td>7</td><td>ten</td><td>R/W</td><td>0</td><td colspan="2">Timer Enable
1: Timer enabled
0: Timer disabled</td></tr>
<tr><td>6</td><td>-</td><td>DNM</td><td>0</td><td colspan="2">Reserved Do Not Modify</td></tr>
<tr><td>5:3</td><td>pres</td><td>R/W</td><td>0</td><td colspan="2">Timer Prescaler Select
Sets the timer’s prescaler value. The prescaler divides the PCLK input to the timer and sets the timer’s count clock,
$$f_{CNT\_CLK} = \frac{f_{CLK\_SOURCE}}{prescaler}$$
The timer’s prescaler setting is a 4-bit value with pres3 as the most significant bit and pres as the three least significant bits. The table below shows the prescaler values based on pres3:pres.
<table>
<tr><th>pres3</th><th>pres</th><th>Prescaler</th></tr>
<tr><td>0</td><td>0b000</td><td>1</td></tr>
<tr><td>0</td><td>0b001</td><td>2</td></tr>
<tr><td>0</td><td>0b010</td><td>4</td></tr>
<tr><td>0</td><td>0b011</td><td>8</td></tr>
<tr><td>0</td><td>0b100</td><td>16</td></tr>
<tr><td>0</td><td>0b101</td><td>32</td></tr>
<tr><td>0</td><td>0b110</td><td>64</td></tr>
<tr><td>0</td><td>0b111</td><td>128</td></tr>
<tr><td>1</td><td>0b000</td><td>256</td></tr>
<tr><td>1</td><td>0b010</td><td>512</td></tr>
<tr><td>1</td><td>0b011</td><td>1024</td></tr>
<tr><td>1</td><td>0b100</td><td>2048</td></tr>
<tr><td>1</td><td>0b101</td><td>4096</td></tr>
<tr><td>1</td><td>0b110</td><td>Reserved</td></tr>
<tr><td>1</td><td>0b111</td><td>Reserved</td></tr>
</table></td></tr>
</table>

Timer Control Register				WUTn_CTRL	[0x0010]
Bits	**Name**	**Access**	**Reset**	**Description**	
2:0	tmode	R/W	0	**Timer Mode Select** Sets the timer's operating mode. **tmode** / **Timer Mode** 0b000: One-Shot 0b001: Continuous 0b010: Reserved 0b011: Reserved 0b100: Reserved 0b101: Reserved 0b110: Reserved 0b111: Reserved	

18. Timers

Multiple 32-bit and dual 16-bit, reloadable timers are provided.

The features include:

- Operation as single 32-bit counter or single/dual 16-bit counters
- Programmable prescaler with values from 1 to 4096
- Non-overlapping PWM output generation with configurable off-time
- Capture, compare, and capture/compare capability
- Timer input and output signals available, mapped as alternate function
- Configurable input pin for event triggering, clock gating, or capture signal
- Timer output pin for event output and PWM signal generation
- Independent interrupts

Certain instances (denoted LPTMR) seen in *Table 18-1. MAX78000 TMR/LPTMR Instances* can be configured to operate in low power modes and wake the device from the low-power back to ACTIVE mode

Each timer provides multiple operating modes. See *Table 18-2. MAX78000 I/O Signal Configuration* for which operating modes are available for each configuration:

- One-Shot: Timer counts up to terminal value then halts.
- Continuous: Timer counts up to terminal value then repeats.
- Counter: Timer counts input edges received on timer input pin.
- Pulse Width Modulated (PWM) / PWM Differential.
- Capture: Captures a snapshot of the current timer count when timer input edge transitions.
- Compare: Timer pin toggles when timer exceeds terminal count.
- Gated: Timer increments only when timer input pin is asserted.
- Capture/Compare: Timer counts when timer input is asserted, captures timer count when input is deasserted.

18.1 Instances

Instances of the peripheral are listed in *Table 18-1. MAX78000 TMR/LPTMR Instances*. Both the TMR and LPTMR are functionally very similar, so for convenience they are referred to as just TMR. The LPTMR instances can function while the device is in certain low power modes.

Table 18-1. MAX78000 TMR/LPTMR Instances

<table>
<tr><th>INSTANCE</th><th>REGISTER ACCESS NAME</th><th>CASCADE 32-BIT MODE</th><th>16 BIT MODE</th><th>POWER MODES</th><th>CLK0</th><th>CLK1</th><th>CLK2</th><th>CLK3</th></tr>
<tr><td>TMR0</td><td>TMR0</td><td rowspan="4">YES</td><td rowspan="4">Dual</td><td rowspan="4">ACTIVE
SLEEP
LPM</td><td rowspan="4">PCLK</td><td rowspan="4">ISO</td><td rowspan="4">IBRO</td><td rowspan="4">ERTCO</td></tr>
<tr><td>TMR1</td><td>TMR1</td></tr>
<tr><td>TMR2</td><td>TMR2</td></tr>
<tr><td>TMR3</td><td>TMR3</td></tr>
<tr><td rowspan="2">LPTMR0</td><td rowspan="2">TMR4</td><td rowspan="2">NO</td><td rowspan="2">Single</td><td>ACTIVE
SLEEP
LPM</td><td>IBRO</td><td>ERTCO</td><td>INRO</td><td>LPTMR0_CLK
GPIO2.6 (AF1)</td></tr>
<tr><td>UPM</td><td>N/A</td><td>N/A</td><td>ERTCO</td><td>INRO</td></tr>
<tr><td rowspan="2">LPTMR1</td><td rowspan="2">TMR5</td><td rowspan="2">NO</td><td rowspan="2">Single</td><td>ACTIVE
SLEEP
LPM</td><td>IBRO</td><td>IBRO / 8</td><td>INRO</td><td>LPTMR1_CLK
GPIO2.7 (AF1)</td></tr>
<tr><td>MICRO POWER</td><td>N/A</td><td>N/A</td><td>ERTCO</td><td>INRO</td></tr>
</table>

Table 18-2. MAX78000 I/O Signal Configuration

INSTANCE	REGISTER ACCESS NAME	TMRn_IOA	TMRn_IOB	TMRn_IOAN	TMRn_IOBN
TMR0	TMR0	GPIO5.0	GPIO5.1	GPIO5.2	GPIO5.3
TMR1	TMR1	GPIO5.4	GPIO5.5	GPIO5.6	GPIO5.7
TMR2	TMR2	GPIO5.8	GPIO5.9	GPIO5.10	GPIO5.11
TMR3	TMR3	GPIO5.12	GPIO5.13		
LPTMR0	TMR4	GPIO5.14			
LPTMR1	TMR5	GPIO5.15			

18.2 Basic Operation

The timer modes operate by incrementing the *WUTn_CNT*.count register, driven by either the timer clock, an external stimulus on the timer pin, or a combination of both. The *WUTn_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 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 *WUTn_CNT*.count with a new starting value, or disabling the counter. The end of a timer period will always set the corresponding interrupt bit and can generate an interrupt, if enabled.

In most modes the timer peripheral automatically sets *WUTn_CNT*.count to 0x0000_0001 at the end of a timer period, but *WUTn_CNT*.count is set to 0x0000_0000 following a system reset. This means the first timer period following a system reset will be one timer clock longer than subsequent timer periods if *WUTn_CNT*.count is not initialized to 0x0000_0001 during the timer configuration step.

18.3 32-Bit Single / 32-Bit Cascade / Dual 16-Bit Operation

Most instances contain two 16-bit timers, which may support combinations of single or cascaded 32-bit modes, and single or dual 16-bit modes as shown in *Table 18-1. MAX78000 TMR/LPTMR Instances*. In most cases the two 16-bit timers have the same functionality.

The terminology TimerA and TimerB are used to differentiate the organization of the 32-bit registers shown in *Table 18-3. TimerA/TimerB 32-Bit Field Allocation*. Most of the other registers have the same fields duplicated in the upper and lower 16-bits are differentiated with the _a and _b suffixes. When the same functionality is supported by both 16-bit timers, the examples are shown more generically with the _a/_b omitted from the field names.

In the 32-bit modes, the fields and controls associated with TimerA are used to control the 32-bit functions.

When in dual 16-bit mode it is allowed to use TimerB without using TimerA.

Table 18-3. TimerA/TimerB 32-Bit Field Allocations

REGISTER	CASCADE 32-BIT MODE	DUAL 16-BIT MODE	
Timer Counter	TimerA(CNT)[31:0] = *TMRn_CNT*.count [31:0]	TimerA(CNT) [15:0] = *TMRn_CNT*.count [15:0]	TimerB(CNT)[15:0] = *TMRn_CNT*.count [31:16]
Timer Compare	TimerA(CMP)[31:0] = *TMRn_CMP*.compare [31:0]	TimerA(CMP)[15:0] = *TMRn_CMP*.compare [15:0]	TimerB(CMP)[15:0] = *TMRn_CMP*.compare [31:16]
Timer PWM	TimerA(PWM)[31:0] = *TMRn_PWM*.pwm [31:0]	TimerA(PWM)[15:0] = *TMRn_PWM*.pwm [15:0]	TimerB(PWM)[15:0] = *TMRn_PWM*.pwm [31:16]

18.4 Timer Clock Sources

Clocking of timer functions is driven by the timer clock frequency, f_{CNT_CLK}, which is a function of the selected clock source shown in *Table 18-1. MAX78000 TMR/LPTMR Instances*. 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 *WUTn_CTRL*.pres field.

Equation 18-1: Timer Peripheral Clock Equation

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

Software writes to the timer registers, and external events on timer pins will be asynchronous events to the slower timer clock frequency. These events are latched on the next rising edge of the timer clock. Since it is not possible to observe the timer clock directly, input events may have up to 0.5 timer clock delay before being recognized.

Use this procedure to change the timer peripheral clock source.

1. Disable the timer peripheral
   a. Clear *WUTn_CTRL*.en = 0 to initiate the disable.
   b. Wait until hardware clears *TMRn_CTRL1*.clken = 0 to confirm the timer peripheral is disabled.
2. Disable the timer clock source
   a. Clear *WUTn_CTRL*.clken = 0 to initiate the disable.
   b. Wait until hardware clears *WUTn_CTRL*.tmclkrd to 0 to confirm the timer clock source is disabled.
3. Set *TMRn_CTRL1*.clksel to the new desired clock source.
4. Enable the timer clock source
   a. Set *WUTn_CTRL*.clken = 1 to initiate the enable.
   b. Wait until hardware sets *WUTn_CTRL*.tmclkrd to 1 to confirm the timer clock source is enabled.
5. Enable the timer peripheral
   a. Set *WUTn_CTRL*.en = 1 to initiate the enable.
   b. Wait until hardware sets TMRn_CTRL1.clken = 1 to confirm the timer peripheral is enabled.

The timer peripheral should be disabled while changing any of the registers in the peripheral. If changing the clock source is part of the configuration process, it may be preferable to execute steps 1-4, configure the timer peripheral, and lastly enable the timer peripheral again as described in step 5.

18.5 Timer Pin Functionality

Each timer instance has external input function and an external output function. Depending on the specific device and package, the input and output signals may be on different device pins or may be multiplexed on a single bi-directional pin. Timer pin assignments are detailed in the data sheet for the specific device.

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 will have the same electrical characteristics, such as pullup/pulldown strength, drive strength, etc. as the GPIO mode settings for that pin. The pin characteristics must be configured before enabling the timer. When configured as an output, the corresponding bit in the GPIO_OUT register should be configured 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.

18.6 Wakeup Events

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

Wakeup example Programming Sequence Example:

1. Enable CLK_TMR(1~3) which is asynchronous to each other and CLK
2. Enable CLK
3. Set *TMRn_CTRL0*.clken, change clock source by configurate *TMRn_CTRL1*.clksel, better in dual timer mode and set different value for 2 timers
4. Check *TMRn_CTRL1*.clkrdy==1
5. Set *TMRn_CTRL0*.rst if C_GLITCHLESS_CLK_MUX==0
6. Configure the timer,
   a. There are 2 registers *WUTn_CNT*.count and *TMRn_PWM.*pwm in the CLK_TMR domains, access other register is no different for C_APB_ASYNC_WRITE_ACK/ C_APB_ASYNC_READ_ACK
   b. The PREADY should be always high if *TMR_CFG*.clksel==0 or (C_APB_ASYNC_WRITE_ACK==0 and C_APB_ASYNC_READ_ACK==0) or (not CNT/PWM)
   c. Need to check *TMRn_INTFL*.wrdone==1 after Write *WUTn_CNT*.count /*TMRn_PWM.*pwm when C_APB_ASYNC_WRITE_ACK==0
   d. Need to “double read and compare the data” after read *WUTn_CNT*.count /*TMRn_PWM.*pwm when C_APB_ASYNC_READ_ACK==0
7. Set *WUTn_CTRL*.en = 1
8. Check *TMRn_CTRL1*.clken==1
9. Enable TMR_CFG.WKENx
10. Turn off CLK
11. Wait for TMR_WAKE_O
12. Turn on CLK
13. Clear *WUTn_CTRL*.en = 0.
14. Check *TMRn_CTRL1*.clken ==0
15. Check status
16. Clear *TMRn_CTRL0*.clken
17. Check *TMRn_CTRL0*.clkrdy==0
18. Go to step 3

18.7 PWM Operation

Synchronous and differential PWM modes are supported:

18.7.1 PWM Synchronous Mode

The peripheral is in PWM synchronous mode if *TMRn_CTRL0*.pwmsync = 1 and *TMRn_CTRL0*.tmode is configured for PWM. In the dual timer mode, we can configure the timers to either 2 master, 2 slaves or “1 master and 1 slave”.

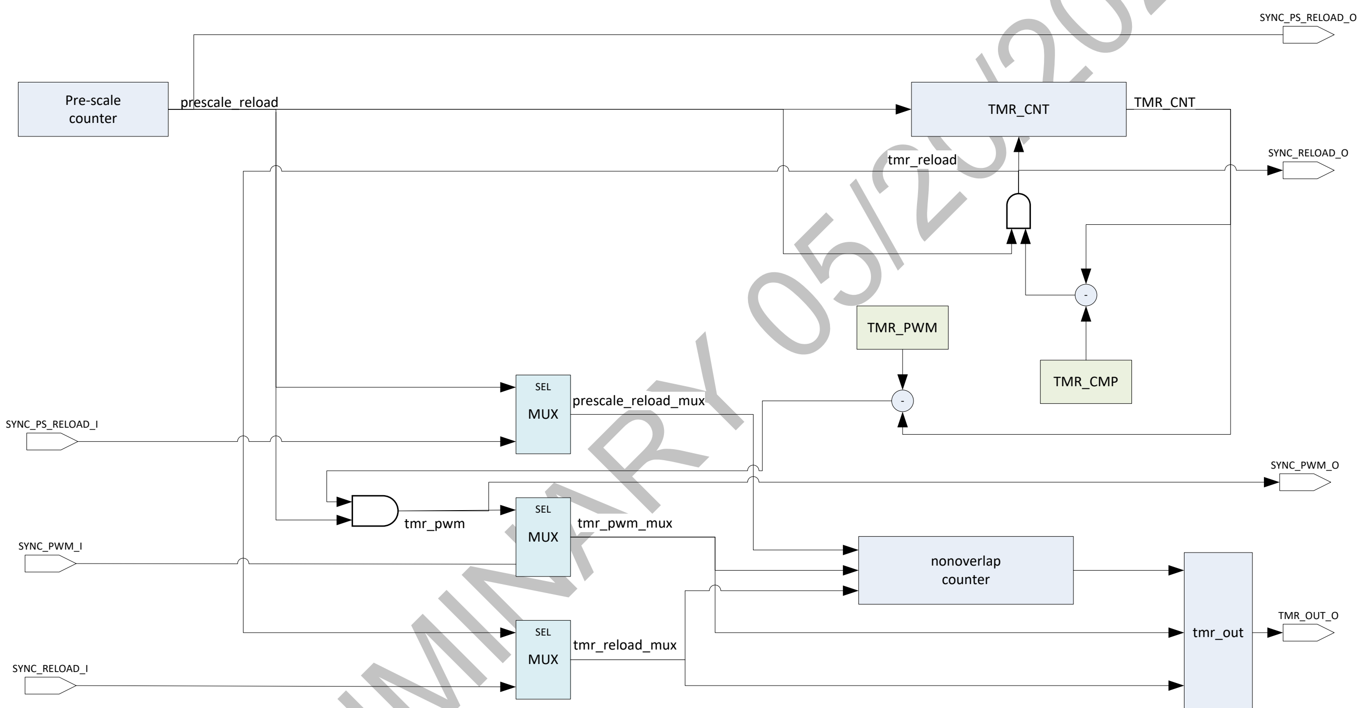
The reload signals of the timer counter and PWM counter are coming from the pins which can be connected to the other timer circuitry which are synchronous to each other. The periods of the two timers are the same, and the duty cycle can be configured by TMR_NOLCMP register.

The timer is in master mode when *TMRn_CTRL0*.pwmsync =0. The PWM operations are based on the setting from register. Three control signals (prescale_reload_mux, tmr_pwm_mux, tmr_reload_mux) control the behavior of nonoverlap counters and PWM output. SYNC input signals are don’t care in master mode.

The timer is in slave mode when *TMRn_CTRL0*.pwmsync =1. The PWM operations are based on the SYNC input pins. Three control signals (prescale_reload_mux, tmr_pwm_mux, tmr_reload_mux) are coming from SYNC input pins, so that the behavior of nonoverlap counters and PWM output are controlled by another timer which can be in master mode or slave mode. Pre-scale counter, *WUTn_CNT*.count counter and SYNC output signals are controlled by the configuration of the registers, so are the behavior of interrupt and wake up functions.

The SYNC related signals are active for one CLK_TMR period and no asynchronous interfaces between them, that mean the master and slave should work on the same clock domain.

Figure 18-1: PWM Synchronous Mode Circuit Diagram



18.7.2 PWM Differential Mode

In Differential Mode, an additional PWM output ØA’ (phase A prime) is available . ØA’ functions as a differential or complementary output to the primary PWM output (ØA). By default, ØA’ will behave as an exact inverse. For nonoverlapping control of power transistors and other complementary switching applications, an 8-bit Non-Overlapping High and Low counter is featured to configure the differential output phases. This counter adjusts the 2 two “dead time” phases from PWM output ØA’ falling edge to ØA rising edge using the Non-Overlapping High Compare register (NOLHCMP) and vice-versa for the Non-Overlapping Low Compare register (NOLLCMP). This scheme provides independent control of these two dead phases. In PWM mode, each output has an additional independent polarity control bit *TMRn_CTRL0*.nolhpol and *TMRn_CTRL0*.nollhpol for additional waveform configuration which can result in some redundancies with the primary polarity bit *TMRn_CTRL0*.pol. *Figure 18-2: Timer Output Complementary Waveforms* shows the effect these three fields on the output waveforms.

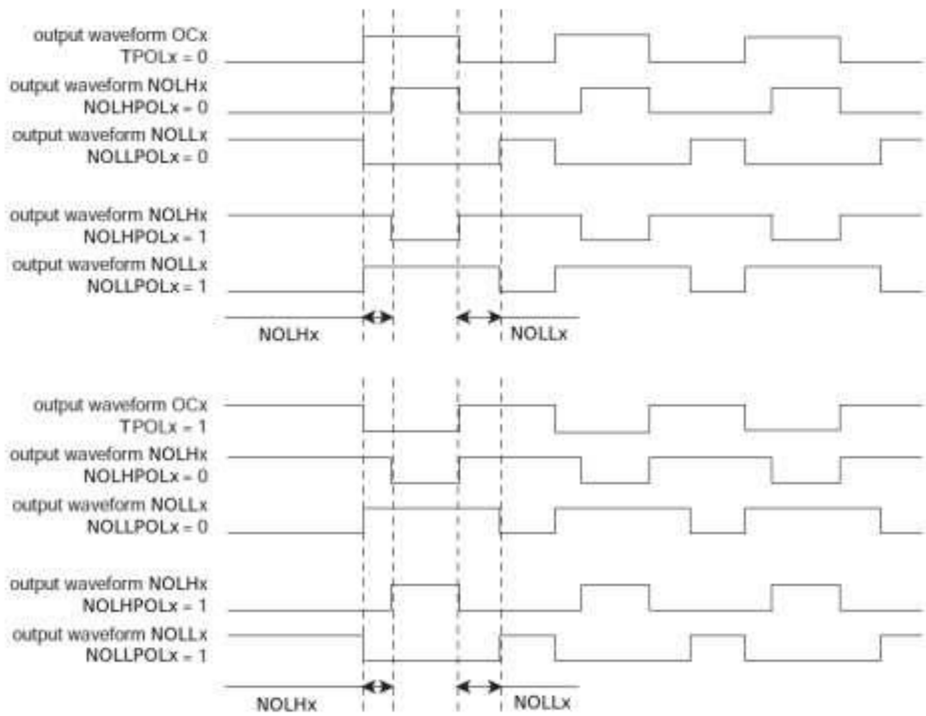
Finally, a timer PWM Synchronization Mode bit is available *TMRn_CTRL0*.pwmsync that multiplexes a primary timer’s main counter to other secondary (or slave) timers. This attribute will synchronize the timers’ periods, primary polarity polarities and duty-cycles to fix a base PWM waveform depicted as OCx in *Figure 18-2: Timer Output Complementary Waveforms* below. However, the Non-Overlapping registers are independent and configurable for each of the synchronized timers, regardless of the *TMRn_CTRL0*.pwmsync bit. This provides independent adjustments to several sets of complementary switches.

The device is set up to have two sets of complementary outputs.

For TimerA configured in PWM Differential Mode with *TMRn_CTRL0*.pwmsync =1, Ø**A =** TCLK0, Ø**A' =** TCLK1.

For TimerB configured in PWM Differential Mode with *TMRn_CTRL0*.pwmsync =1, Ø**A =** TCLK2, Ø**A' =** TCLK3.

Figure 18-2: Timer Output Complementary Waveforms



18.8 Operating Modes

Multiple operating modes are supported. The availability of some operating modes is dependent on the device and package-specific implementation of the external input and output signals.

Table 18-4. MAX78000 Operating Mode Availability, Base Package

TIMER MODE	TMR0	TMR1	TMR2	TMR3	TMR4	TMR5
One-Shot Mode (0x0)	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	Single 16-bit	Single 16-bit
Continuous Mode (0x1)	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	Single 16-bit	Single 16-bit
Counter Mode (0x2)	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	Single 16-bit	Single 16-bit
PWM Mode (0x3)	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	Single 16-bit	Single 16-bit
Capture Mode (0x4)	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	Single 16-bit	Single 16-bit

TIMER MODE	TMR0	TMR1	TMR2	TMR3	TMR4	TMR5
Compare Mode (0x5)	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	Single 16-bit	Single 16-bit
Gated Mode (0x6)	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	Single 16-bit	Single 16-bit
Capture/Compare Mode (0x7)	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	Single 16-bit	Single 16-bit
Dual Edge Capture Mode (0x8)	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	Single 16-bit	Single 16-bit
Reserved (0x9)						
Reserved (0xA)						
Reserved (0xB)						
Reserved (0xC)						
Reserved (0xD)						
Inactive Gated Mode (0xE)	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	32-bit Cascade Dual 16-bit	Single 16-bit	Single 16-bit
Reserved (0xF)						

18.8.1 One-Shot Mode (0x0)

In One-shot mode the timer peripheral increments *WUTn_CNT*.count until it matches *WUTn_CMP*.compare and then stops incrementing and disables the timer. The timer optionally pulses the timer output signal to its active state for one clock cycle. In this mode, the timer must be re-enabled to start another one-shot mode event.

The timer period ends on the timer clock following *WUTn_CNT*.count = *WUTn_CMP*.compare.

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

1. Hardware resets *WUTn_CNT*.count to 0x0000_0001
2. Hardware disables the timer by clearing *WUTn_CTRL*.en = 0.
3. If the timer output is enabled, the timer pin is driven to its active state for one timer clock. It then returns to its inactive state.
4. The corresponding *WUTn_INT*.irq field will be set to 1 to indicate a timer interrupt event occurred.

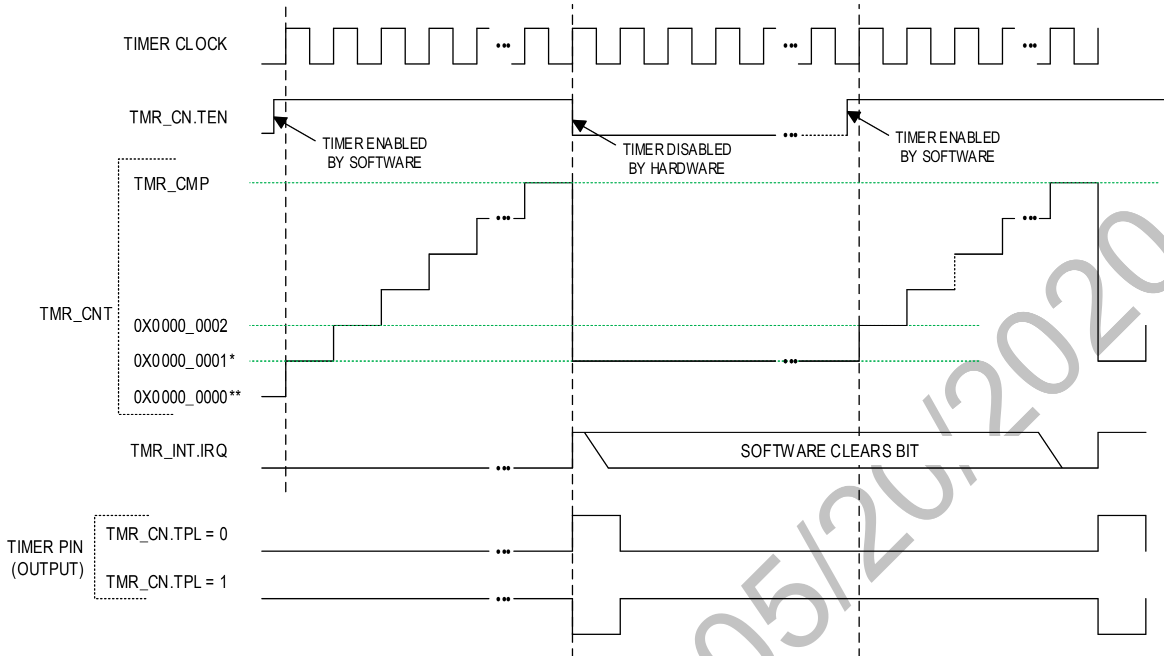
The timer period is calculated using the following equation:

Equation 18-2: One-shot Mode Timer Period

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

The timer period ends on the timer clock following *WUTn_CNT*.count = *WUTn_CMP*.compare.

Figure 18-3: One-Shot Mode Diagram



* TMR_CNT AUTOMATICALLY RELOADS WITH 0X0000_0001 AT THE END OF THE TIMER PERIOD, BUT SOFTWARE CAN WRITE ANY INITIAL VALUE TO TMR_CNT BEFORE THE TIMER IS ENABLED.

** THE DEFAULT VALUE OF TMR_CNT FOR THE FIRST PERIOD AFTER A SYSTEM RESET IS 0X0000_0000 UNLESS CHANGED BY SOFTWARE.

Configure the timer for One-Shot mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Source*.
2. If desired, change the timer clock source as described in *Timer Clock Source*.
3. Set *WUTn_CTRL*.tmode to 0x0 to select One-shot mode.
4. Set *WUTn_CTRL*.pres to set the prescaler that determines the timer frequency.
5. If using the timer output function:
   a. Configure the pin as a timer output, select the correct alternate function mode, and configure the electrical characteristics as needed.
   b. Set *WUTn_CTRL*.pol to match the desired (inactive) state.
   c. Set *TMRn_CTRL1*.outen = 1 and *TMRn_CTRL1*.outben = 0 to enable the timer output signal on the timer output pin. Set *TMRn_CTRL1*.outen = 0 and *TMRn_CTRL1*.outben = 1 to enable the inverted timer output signal on the timer output pin.
6. If using the timer interrupt, enable corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to *WUTn_CMP*.compare. Hardware will immediately clear *TMRn_INTFL*.wrdone to 0.
8. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again to confirm the register has been updated with the new value.
9. If desired, write an initial value to *WUTn_CNT*.count. This effects only the first period; subsequent timer periods always reset *WUTn_CNT*.count = 0x0000_0001. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again.
10. Enable the timer peripheral as described in *Timer Clock Source*.

18.8.2 Continuous Mode (0x1)

In Continuous mode, the timer peripheral increments *WUTn_CNT*.count until it matches *WUTn_CMP*.compare, resets *WUTn_CNT*.count to 0x0000_0001, and continues incrementing. The timer optionally toggles the state of the timer output signal at the end of the timer period.

The timer period ends on the timer clock following *WUTn_CNT*.count = *WUTn_CMP*.compare.

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

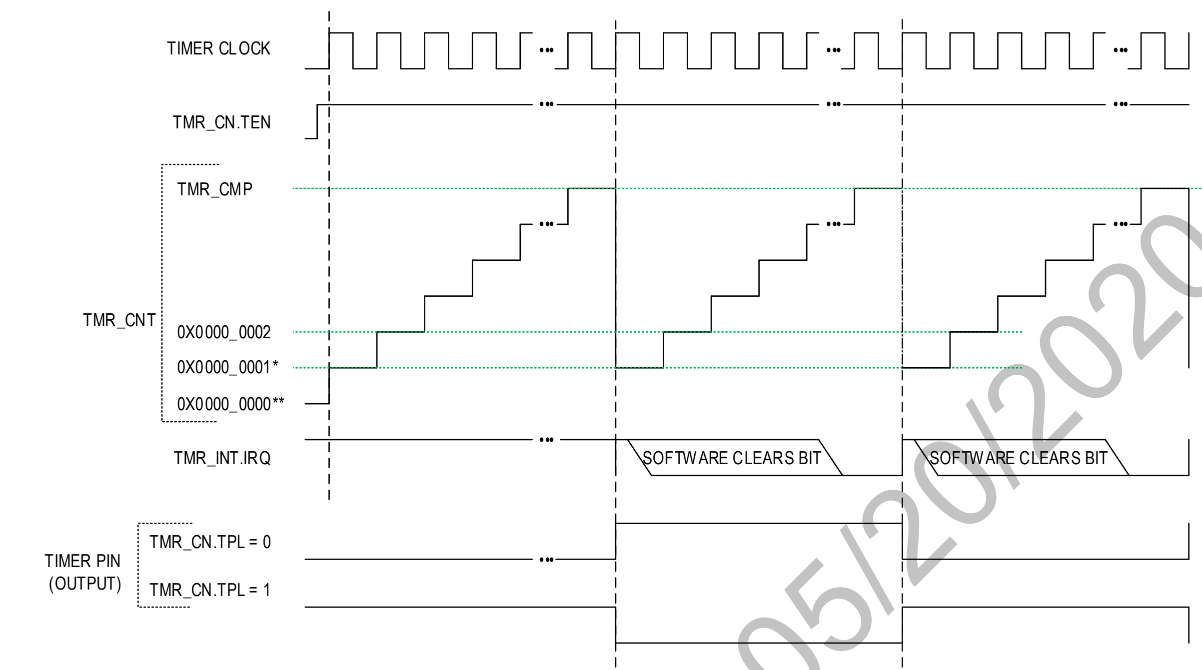
1. Hardware resets *WUTn_CNT*.count to 0x0000_0001. The timer remains enabled and continues incrementing.
2. If the timer output signal toggles state. The timer output pin will change state if the timer output is enabled.
3. The corresponding *WUTn_INT*.irq field will be set to 1 to indicate a timer interrupt event occurred.

The Continuous Mode Timer Period is calculated using *Equation 17-2*.

Equation 18-3: Continuous Mode Timer Period

$$\text{Continuous mode timer period in seconds} = \frac{\text{TMRn\_CMP} - \text{TMRn\_CNT}_{\text{INITIAL\_VALUE}} + 1}{\text{f}_{\text{CNT\_CLK}}\ (\text{Hz})}$$

Figure 18-4: Continuous Mode Diagram



* TMR_CNT AUTOMATICALLY RELOADS WITH 0X0000_0001 AT THE END OF THE TIMER PERIOD, BUT SOFTWARE CAN WRITE ANY INITIAL VALUETO TMR_CNT BEFORE THE TIMER IS ENABLED.

** THE DEFAULT VALUE OF TMR_CNT FOR THE FIRST PERIOD AFTER A SYSTEM RESET IS 0X0000_0000 UNLESS CHANGED BY SOFTWARE.

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

1. Disable the timer peripheral as described in *Timer Clock Source*.
2. If desired, change the timer clock source as described in *Timer Clock Source*.
3. Set *WUTn_CTRL*.tmode to 0x1 to select Continuous mode.
4. Set *WUTn_CTRL*.pres to set the prescaler that determines the timer frequency.
5. If using the timer output function:
    a. Configure the designated pin to output the timer output signal, select the correct alternate function mode, and configure the electrical characteristics as needed.
    b. Set *WUTn_CTRL*.pol to match the desired (inactive) state.
    c. Set *TMRn_CTRL1*.outen = 1 and *TMRn_CTRL1*.outben = 0 to enable the timer output signal on the timer output pin. Set *TMRn_CTRL1*.outen = 0 and *TMRn_CTRL1*.outben = 1 to enable the inverted timer output signal on the timer output pin.
6. If using the timer interrupt, enable corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to *WUTn_CMP*.compare. Hardware will immediately clear *TMRn_INTFL*.wrdone to 0.
8. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again to confirm the register has been updated with the new value.
9. If desired, write an initial value to *WUTn_CNT*.count. This effects only the first period; subsequent timer periods always reset *WUTn_CNT*.count = 0x0000_0001. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again.
10. Enable the timer peripheral as described in *Timer Clock Source*.

18.8.3 Counter Mode (0x2)

In Counter mode, the timer peripheral increments *WUTn_CNT*.count each time when a transition occurs on the timer input signal. When *WUTn_CNT*.count = *WUTn_CMP*.compare the interrupt bit is set and *WUTn_CNT*.count is set to 0x0000_0001 and continues incrementing. The timer can be configured to increment on either the rising edge or the falling edge, but not both.

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

Equation 18-4: Counter Mode Maximum Clock Frequency

$$f_{CTR\_CLK} \leq \frac{f_{PCLK}\ (Hz)}{4}$$

The timer toggles the state of the timer output signal at the end of the timer period.

The timer period ends on the rising edge of PCLK following *WUTn_CNT*.count = *WUTn_CMP*.

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

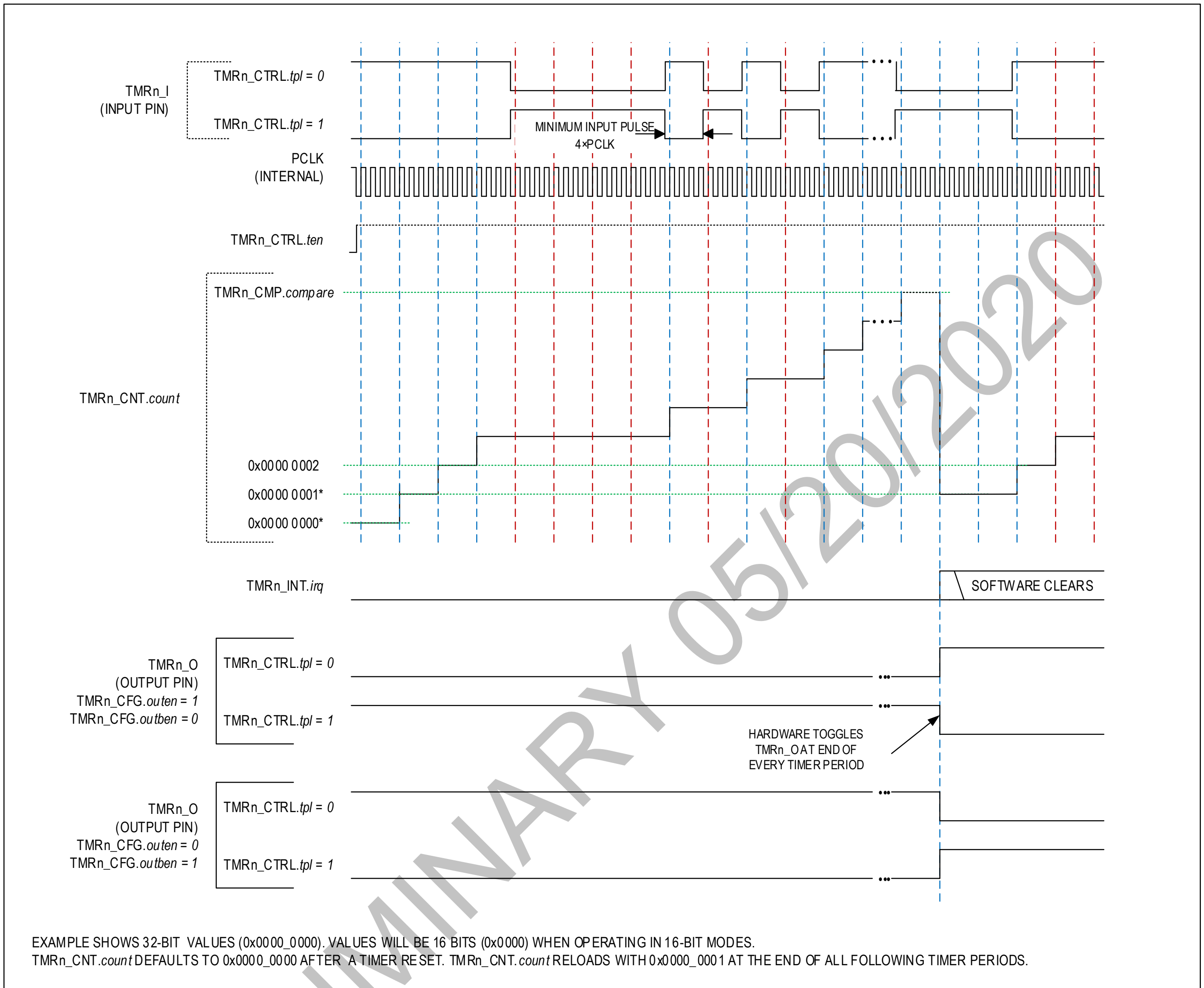
1. Hardware resets *WUTn_CNT*.count to 0x0000_0001. The timer remains enabled and continues incrementing.
2. Hardware toggles the state of the timer output signal. The timer output pin will change state if the timer output is enabled.
3. Hardware sets the corresponding *WUTn_INT*.irq field to 1 to indicate a timer interrupt event occurred..

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

Equation 18-5: Counter Mode Timer Input Transitions

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

Figure 18-5: Counter Mode Diagram



Configure the timer for Counter mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Source*.
2. If desired, change the timer clock source as described in *Timer Clock Source*.
3. Set *WUTn_CTRL*.tmode to 0x2 to select Counter mode.
4. Configure the timer input pin:
    a. Configure the pin as a timer input and configure the electrical characteristics as needed.
    b. Set *WUTn_CTRL*.pol to match the desired initial (inactive) state.
5. If using the timer interrupt, enable corresponding field in the *TMRn_CTRL1* register.
6. Write the compare value to *WUTn_CMP*.compare. Hardware will immediately clear *TMRn_INTFL*.wrdone to 0.
7. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again to confirm the register has been updated with the new value.
8. If desired, write an initial value to *WUTn_CNT*.count. This effects only the first period; subsequent timer periods always reset *WUTn_CNT*.count = 0x0000_0001. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again.
9. Enable the timer peripheral as described in *Timer Clock Source*.

18.8.4 PWM Mode (0x3)

In PWM mode, the timer sends a Pulse-Width Modulated (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 *WUTn_CNT*.count value matches the *TMRn_PWM*.pwm, the timer output signal toggles state. The timer continues counting until it reaches the *WUTn_CMP*.compare value.

The timer period ends on the rising edge of f_{CNT_CLK} following *WUTn_CNT*.count = *WUTn_CMP*.compare.

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

1. The *WUTn_CNT*.count is reset to 0x0000_0001, and the timer resumes counting.
2. The timer output signal is toggled.
3. The timer interrupt bit *WUTn_INT*.irq will be set. An interrupt will be generated if enabled.

When *WUTn_CTRL*.pol = 0, the timer output signal starts low and then transitions to high when the *WUTn_CNT*.count value matches the *TMRn_PWM* value. The timer output signal remains high until the *WUTn_CNT*.count value reaches the *WUTn_CMP*.compare, resulting in the timer output signal transitioning low, and the *WUTn_CNT*.count value resetting to 0x0000_0001.

When *WUTn_CTRL*.pol = 1, the Timer output signal starts high and transitions low when the *WUTn_CNT*.count value matches the *TMRn_PWM* value. The timer output signal remains low until the *WUTn_CNT*.count value reaches *WUTn_CMP*.compare, resulting in the timer output signal transitioning high, and the *WUTn_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 Source*.
2. If desired, change the timer clock source as described in *Timer Clock Source*.
3. Set *WUTn_CTRL*.tmode to 0x3 to select PWM mode.
4. Set *WUTn_CTRL*.pres 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 *WUTn_CTRL*.pol to match the desired initial (inactive) state.
   a. Set *WUTn_CTRL*.pol to select the initial logic level (high or low) and PWM transition state for the timer's output.
   b. Set *WUTn_CNT*.count to the starting count, typically 0x0000_0001. The initial *WUTn_CNT*.count *WUTn_CNT*.count value only effects the initial period in PWM mode with subsequent periods always setting *WUTn_CNT*.count to 0x0000_0001.
   c. Set the *TMRn_PWM* value to the transition period count.
7. Set the *WUTn_CMP*.compare value for the PWM second transition period. Note: *WUTn_CMP*.compare must be greater than the *TMRn_PWM* value.
8. If using the timer interrupt, set the interrupt priority and enable the interrupt.
9. Enable the timer peripheral as described in *Timer Clock Source*.

The PWM period is calculated using the following equation:

Equation 18-6: Timer PWM Period

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

If an initial starting value other than 0x0000_0001 is loaded into the *WUTn_CNT*.count register, use the One-Shot mode equation to determine the initial PWM period.

If *WUTn_CTRL*.pol is 0, the ratio of the PWM output high time to the total period is calculated using *Equation 18-7*, *below*.

Equation 18-7: Timer PWM Output High Time Ratio with Polarity 0

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

If *WUTn_CTRL*.pol is set to 1, the ratio of the PWM output high time to the total period is calculated using *Equation 18-8*.

Equation 18-8: Timer PWM Output High Time Ratio with Polarity 1

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

18.8.5 Capture Mode (0x4)

The Capture mode is most often used to measure the time between user-determined events. The timer increments from an initial value until an event occurs. The event can be the external transition of the timer input signal or a software-generated pseudo-event that captures the current value of *WUTn_CNT*.count and writes it to *TMRn_PWM*.pwm.

A rollover event is a special timer period event which serves as a "timeout" feature for the timer. Write the maximum period for a valid capture event to *TMRn_CMP*.compare. If the selected capture event does not occur before *WUTn_CNT*.count = *TMRn_CMP*.compare a rollover timer period event will occur.

All timer period events will reset *WUTn_CNT*.count = 0x0000_0001 and set the corresponding timer interrupt event flag.

Software should read *TMRn_PWM*.pwm to determine the event that set timer interrupt event flag. *TMRn_PWM*.pwm = 0 indicates a rollover timer period event occurred. Any other value indicates a capture timer period event occurred.

Three events are possible in this mode

18.8.5.1 Capture Event (Timer Input Signal)

A Capture event occurs on the timer clock following the user-specified positive or negative transition of the external input signal. This triggers a 'capture' timer period event which copies *WUTn_CNT*.count to the *TMRn_PWM*.pwm register.

The timer peripheral automatically performs the following actions when a timer input capture event occurs:

1. Hardware copies the value in *WUTn_CNT*.count to *TMRn_PWM*.pwm.
2. The timer remains enabled and continues incrementing.
3. Hardware sets the corresponding *WUTn_INT*.irq field to 1 to indicate a timer interrupt event occurred.

18.8.5.2 Capture Event (Software Capture)

A software capture event occurs when software generates a pseudo-transition on the timer input signal.

18.8.5.3 Rollover Event

The Rollover event occurs when *WUTn_CNT*.count = *TMRn_CMP*.compare, indicating that a capture event did not occur with the desired timer period. The timer peripheral automatically performs the following actions when a rollover event occurs:

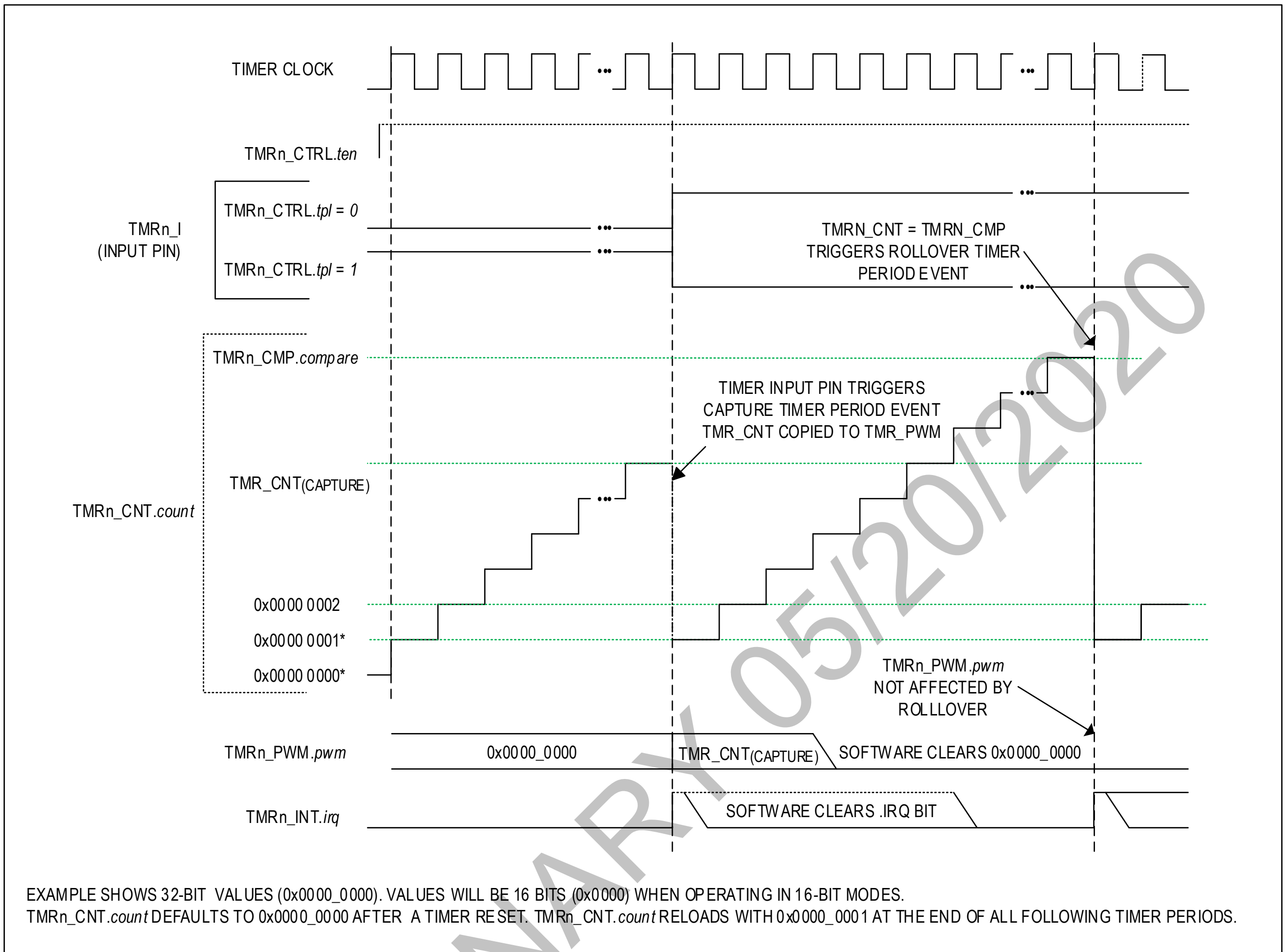
1. Hardware resets *WUTn_CNT*.count to 0x0000_0001. The timer remains enabled and continues incrementing. The value in *TMRn_PWM*.pwm remains unchanged.
2. Hardware sets the corresponding *WUTn_INT*.irq field to 1 to indicate a timer interrupt event occurred.

Equation 18-9: 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.

Figure 18-6: Capture Mode Diagram



Configure the timer for Capture mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Source*.
2. If desired, change the timer clock source as described in *Timer Clock Source*.
3. Select Capture mode by setting *WUTn_CTRL*.tmode to 0x4.
4. Set *WUTn_CTRL*.pres to set the prescaler that determines the timer frequency.
5. If using the TMRn_I pin to generate the capture event:
   a. Configure the pin as a timer input and configure the electrical characteristics as needed.
   b. Set *WUTn_CTRL*.pol to match the desired (inactive) state.
6. If using the timer interrupt, enable the interrupt and set the interrupt priority.
7. Write the initial value to *WUTn_CNT*. This effects only the first period; subsequent periods always begin with 0x0000_0001.
8. Write the compare value to *WUTn_CMP*.compare.
9. Select the capture event by setting *TMRn_CTRL1*.capeventsel.
10. Enable the timer peripheral as described in *Timer Clock Source*.

The timer period is calculated using the following equation:

Equation 18-10: Capture Mode Elapsed Time Calculation in Seconds

$$\text{Capture elapsed time in seconds} = \frac{\text{TMR\_PWM} - \text{TMR\_CNT}_{\text{INITIAL\_VALUE}}}{\text{f}_{\text{CNT\_CLK}}}$$

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

18.8.6 Compare Mode (0x5)

In Compare mode the timer peripheral increments continually from 0x0000_0000 (after the first timer period) to the maximum value of the 32- or 16-bit mode, then rolls over to 0x0000_0000 and continues incrementing. The end of timer period event occurs when the timer value matches the compare value, but the timer continues to increment until the count reaches 0xFFFF FFFF. The timer counter then rolls over and continues counting from 0x0000_0000.

The timer period ends on the timer clock following *WUTn_CNT.count* = *WUTn_CMP.compare*.

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

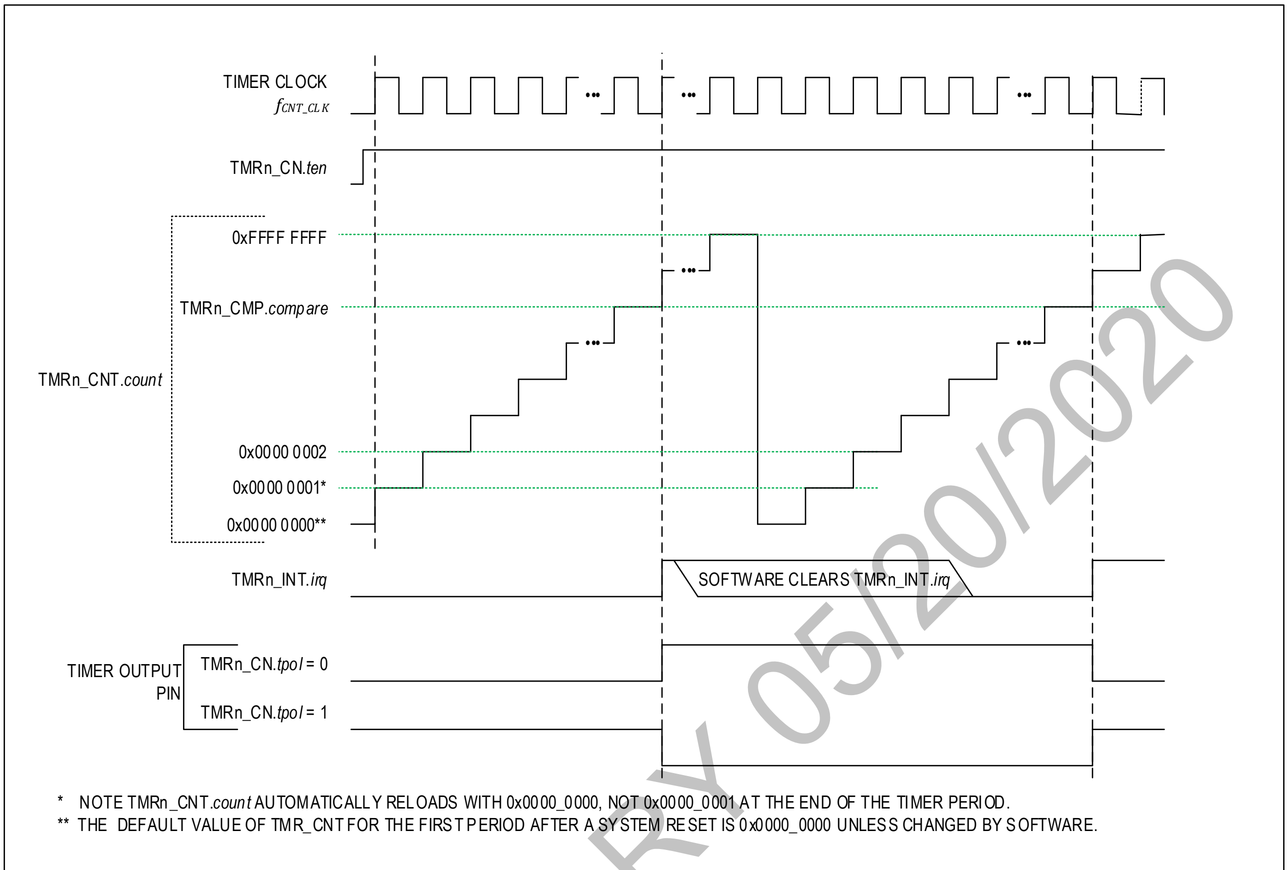
1. 1. Unlike other modes, *WUTn_CNT.count* is reset to 0x0000_00000 not 0x0000_0001 at the end of the timer period. The timer remains enabled and continues incrementing.
2. Hardware sets the corresponding *WUTn_INT.irq* field to 1 to indicate a timer interrupt event occurred.
3. Hardware toggles the state of the timer output signal. The timer output pin will change state if the timer output is enabled.

The Compare Mode timer period is calculated using *Equation 18-11: Compare Mode Timer Period*.

Equation 18-11: Compare Mode Timer Period

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

Figure 18-7: Compare Mode Diagram



Configure the timer for Compare mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Source*.
2. If desired, change the timer clock source as described in *Timer Clock Source*.
3. Set *WUTn_CTRL*.tmode to 0x5 to select Compare mode.
4. Set *WUTn_CTRL*.pres to set the prescaler that determines the timer frequency.
5. If using the timer pin:
   a. Configure the pin for the timer output alternate function and configure the electrical characteristics as needed.
   b. Set *WUTn_CTRL*.pol to match the desired (inactive) state.
6. If using the timer interrupt, enable corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to *WUTn_CMP*.compare. Hardware will immediately clear *TMRn_INTFL*.wrdone to 0.
8. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again to confirm the register has been updated with the new value.
9. If desired, write an initial value to *WUTn_CNT*.count. This effects only the first period; subsequent timer periods always reset *WUTn_CNT*.count = 0x0000_0001. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again.
10. Enable the timer peripheral as described in *Timer Clock Source*.

18.8.7 Gated Mode (0x6)

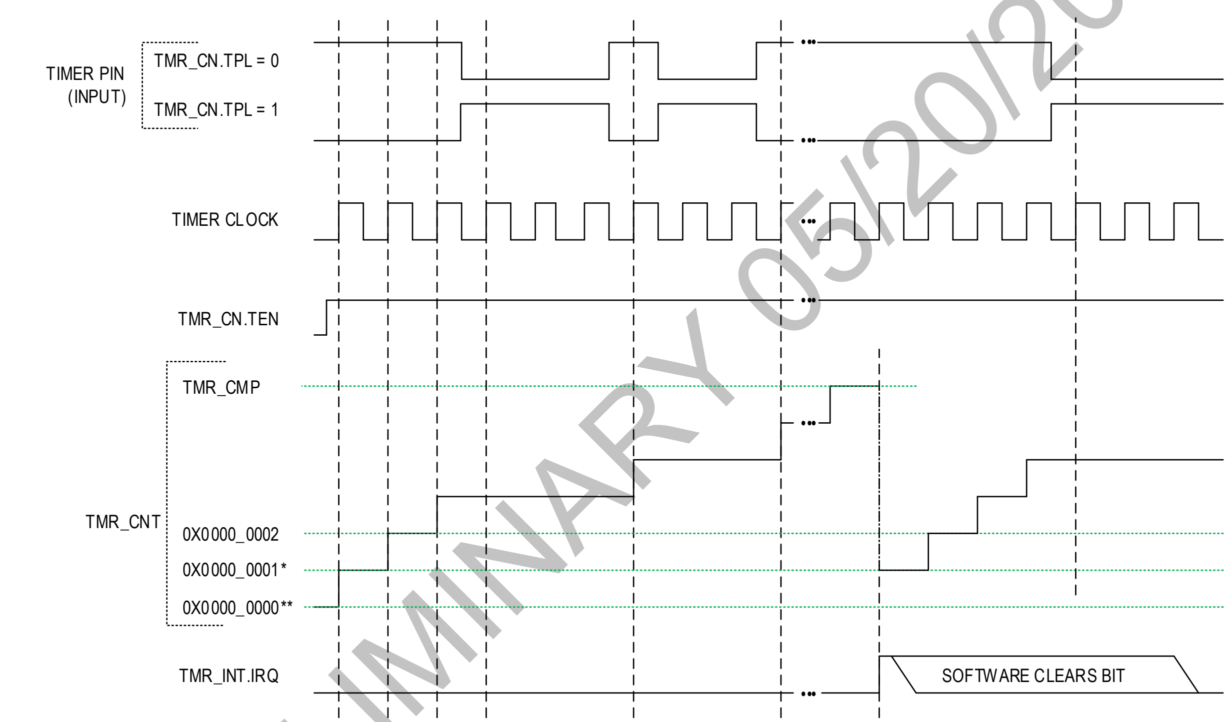
Gated mode is similar to continuous mode, except that *WUTn_CNT*.count only increments when the timer input signal is in its active state.

The timer period ends on the timer clock following *WUTn_CNT*.count = *WUTn_CMP*.compare.

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

1. Hardware resets *WUTn_CNT*.count to 0x0000_0001. The timer remains enabled and continues incrementing.
2. If the timer output signal toggles state. The timer output pin will change state if the timer output is enabled.
3. The corresponding *WUTn_INT*.irq field will be set to 1 to indicate a timer interrupt event occurred.

Figure 18-8: Gated Mode Diagram



* TMR_CNT AUTOMATICALLY RELOADS WITH 0X0000_0001 AT THE END OF THE TIMER PERIOD, BUT SOFTWARE CAN WRITE ANY INITIAL VALUETO TMR_CNT BEFORE THE TIMER IS ENABLED.
** THE DEFAULT VALUE OF TMR_CNT FOR THE FIRST PERIOD AFTER A SYSTEM RESET IS 0X0000_0000 UNLESS CHANGED BY SOFTWARE.

Configure the timer for Gated mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Source*.
2. If desired, change the timer clock source as described in *Timer Clock Source*.
3. Set *WUTn_CTRL*.tmode to 0x6 to select Gated mode.
4. Set *WUTn_CTRL*.pres to set the prescaler that determines the timer frequency.
5. Configure the timer pin:
   a. Configure the pin as a timer input and configure the electrical characteristics as needed.
   b. Set *WUTn_CTRL*.pol to match the desired initial (inactive) state.

6. If using the timer interrupt, enable corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to *WUTn_CMP*.compare. Hardware will immediately clear *TMRn_INTFL*.wrdone to 0.
8. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again to confirm the register has been updated with the new value.
9. If desired, write an initial value to *WUTn_CNT*.count. This effects only the first period; subsequent timer periods always reset *WUTn_CNT*.count = 0x0000_0001. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again.
10. Enable the timer peripheral as described in *Timer Clock Source*.

18.8.8 Capture/Compare Mode (0x7)

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 *WUTn_CTRL*.pol bit.

Each subsequent transition, after the first transition of the timer input signal, captures the *WUTn_CNT*.count value, writing it to the *TMRn_PWM*.pwm register (capture event). When a capture event occurs, a timer interrupt is generated, the *WUTn_CNT*.count value is reset to 0x0000_0001, and the timer resumes counting.

If no capture event occurs, the timer counts up to *WUTn_CMP*.compare. At the end of the cycle where the *WUTn_CNT*.count equals the *WUTn_CMP*.compare, a timer interrupt is generated, the *WUTn_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 *WUTn_CNT*.count = *WUTn_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 was caused by a transition on the timer pin:

1. Hardware copies the value in *WUTn_CNT*.count to *TMRn_PWM*.pwm.
2. Hardware resets *WUTn_CNT*.count to 0x0000_0001.The timer remains enabled and continues incrementing.
3. The timer interrupt bit, *WUTn_INT*.irq, is set and an interrupt is generated if enabled..

In Capture/Compare mode, the elapsed time from the timer start to the capture event is calculated using *Equation 18-12, below*.

Equation 18-12: Capture Mode Elapsed Time

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

Configure the timer for Capture/Compare mode by doing the following:

1. Disable the timer peripheral as described in *Timer Clock Source*.
2. If desired, change the timer clock source as described in *Timer Clock Source*.
3. Set *WUTn_CTRL*.tmode to 0x7 to select Capture/Compare mode.
4. Set *WUTn_CTRL*.pres to set the prescaler that determines the timer frequency.
5. Configure the timer pin:
   a. Configure the pin as a timer input and configure the electrical characteristics as needed.
   b. Set *WUTn_CTRL*.pol to select the positive edge (*WUTn_CTRL*.pol = 0) or negative edge (*WUTn_CTRL*.pol = 0) transition causes the capture event.

6. If using the timer interrupt, enable corresponding field in the *TMRn_CTRL1* register.
7. Write the compare value to *WUTn_CMP*.compare. Hardware will immediately clear *TMRn_INTFL*.wrdone to 0.
8. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again to confirm the register has been updated with the new value.
9. If desired, write an initial value to *WUTn_CNT*.count. This effects only the first period; subsequent timer periods always reset *WUTn_CNT*.count = 0x0000_0001. Wait until hardware sets *TMRn_INTFL*.wrdone to 1 again.
10. Enable the timer peripheral as described in *Timer Clock Source*.

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

18.8.9 Dual Edge Capture Mode (0x8)

Dual edge capture mode is similar to capture mode except the counter can capture on both edges of the timer input pin.

18.8.10 Inactive Gated Mode (0xE)

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

18.9 Registers

See Table 2-1. APB Register Base Address Map 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. MAX78000 TMR/LPTMR Instances* 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 TBD Table 2-2. Field Access Definitions 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 18-5: Timer Register Summary

Offset	Name	Description
0x0000	*TMRn_CNT*	Timer Counter Register
0x0004	*WUTn_CMP*	Timer Compare Register
0x0008	*TMRn_PWM*	Timer PWM Register
0x000C	*WUTn_INT*	Timer Interrupt Register
0x0010	*TMRn_CTRL0*	Timer Control Register
0x0014	*TMRn_NOLCMP*	Timer Non-Overlapping Compare Register
0x0018	*TMRn_CTRL1*	Timer Configuration Register
0x001C	*TMRn_WKFL*	Timer Wakeup Status Register

18.10 Register Details

Table 18-6: 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. Hardware will automatically clear the corresponding *WUTn_INT*.wrdone field to 0 when this field is written, and software must wait until *WUTn_INT*.wrdone =1 before proceeding. Disable the timer by setting *TMRn_CTRL1*.wrdis to 1 before writing to this field.	

Timer Count				TMRn_CNT		0x0000	
Bits	Field	Access	Reset	Description			
				Single 32-Bit	Cascade 32-Bit	Dual 16-Bit	
				TimerA[31:0] = *WUTn_CNT[31:0]*	TimerA[31:0] = *WUTn_CNT[31:0]*	TimerA[15:0] = *WUTn_CNT[15:0]*	TimerB[15:0] = *WUTn_CNT[31:16]*

Table 18-7: 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 18-8: 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 *WUTn_CNT*.count = *WUTn_CMP*.compare, the PWM output transitions to the second period of the PWM cycle. The second PWM period count is stored in *WUTn_CMP*.compare. *TMRn_PWM*.pwm must be less than *WUTn_CMP*.compare for PWM mode operation. **Timer Capture Value** In Capture, Compare, and Capture/Compare modes, this field is used to store the *WUTn_CNT*.count value when a Capture, Compare, or Capture/Compare event occurs. Hardware will automatically clear the corresponding *WUTn_INT*.wrdone field to 0 when this field is written, and software must wait until *WUTn_INT*.wrdone =1 before proceeding. Disable the timer by setting *TMRn_CTRL1*.wrdis to 1 before writing to this field.	

Table 18-9: 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	**Dual Timer Mode Write Disable TimerB** This field disables write access to *WUTn_CNT*.count[31:16] and *TMRn_PWM.*pwm[31:16] so that only the 16 bits associated with updating TimerA are modified during writes these 32-bit registers. This bit will always read 0 if the timer is configured in a 32-bit mode. 0: Enabled 1: Disabled	
25	wrdone_b	R	0	**Write Done TimerB** This field is cleared to 0 by hardware when software performs a write to *WUTn_CNT*.count[31:16] or *TMRn_PWM.*pwm[31:16] when in dual timer mode. Wait until the field is set to 1 again before proceeding. 0: Operation in progress.. 1: Operation complete.	
23:17	-	RO	0	**Reserved**	
16	irq_b	RW	0	**Interrupt Event TimerB** 0: No event. 1: Interrupt event occurred.	

Timer Interrupt				TMRn_INTFL	0x000C
Bits	**Field**	**Access**	**Reset**	**Description**	
15:10	-	RO	0	**Reserved**	
9	wr_dis_a	R/W	0	**Dual Timer Mode Write Disable TimerA** This field disables write access to *WUTn_CNT*.count[15:0] and *TMRn_PWM*.pwm[15:0] so that only the 16 bits associated with updating TimerB are modified during writes to these 32-bit registers. This bit will always read 0 if the timer is configured in a 32-bit mode. 0: Enabled 1: Disabled	
8	wrdone_a	R	0	**Write Done TimerA** This field is cleared to 0 by hardware when software performs a write to *WUTn_CNT*.count[15:0] and *TMRn_PWM*.pwm[15:0] when in dual timer mode. Wait until the field is set to 1 again before proceeding. 0: Operation in progress.. 1: Operation complete.	
7:1	-	RO	0	**Reserved**	
0	irq_a	W1C	0	**Interrupt Event TimerA** 0: No event. 1: Interrupt event occurred.	

Table 18-10: Timer Control 0 Register

Timer Control 0				TMRn_CTRL0	0x0010
Bits	**Field**	**Access**	**Reset**	**Description**	
31	en_b	R/W	0	**Enable TimerB** 0: Disabled 1: Enabled	
30	clken_b	R/W	0	**Timer Clock Enable TimerB** 0: Disabled 1: Enabled	
29	rst_b	W1	0	**RST TimerB** 0: No action 1: Reset all FF in CLK_TMR domain	
28:24	-	RO	0	**Reserved**	

<table>
<tr><th colspan="4">Timer Control 0</th><th>TMRn_CTRL0</th><th>0x0010</th></tr>
<tr><th>Bits</th><th>Field</th><th>Access</th><th>Reset</th><th colspan="2">Description</th></tr>
<tr><td>23:20</td><td>clkdivb</td><td>R/W</td><td>0</td><td colspan="2">Timer Prescaler Select TimerB
The prescaler divides the clock input to the timer and sets the timer’s count clock, $f_{CNT\_CLK} = \mathrm{CNT\_CLK\ (Hz)}/_{\mathrm{prescaler}}$. Refer to the operational mode details to determine which modes use the prescaler.
<table>
<tr><th>38) pres</th><th>39) Prescaler</th></tr>
<tr><td>40) 0x0</td><td>41) 1</td></tr>
<tr><td>42) 0x1</td><td>43) 2</td></tr>
<tr><td>44) 0x2</td><td>45) 4</td></tr>
<tr><td>46) 0x3</td><td>47) 8</td></tr>
<tr><td>48) 0x4</td><td>49) 16</td></tr>
<tr><td>50) 0x5</td><td>51) 32</td></tr>
<tr><td>52) 0x6</td><td>53) 64</td></tr>
<tr><td>54) 0x7</td><td>55) 128</td></tr>
<tr><td>56) 0x8</td><td>57) 256</td></tr>
<tr><td>58) 0x9</td><td>59) 512</td></tr>
<tr><td>60) 0xA</td><td>61) 1024</td></tr>
<tr><td>62) 0xB</td><td>63) 2048</td></tr>
<tr><td>64) 0xC</td><td>65) 4096</td></tr>
<tr><td>66) 0xD</td><td>67) Reserved</td></tr>
<tr><td>68) 0xE</td><td>69) Reserved</td></tr>
<tr><td>70) 0xF</td><td>71) Reserved</td></tr>
</table></td></tr>
</table>

Timer Control 0				TMRn_CTRL0	0x0010
Bits	**Field**	**Access**	**Reset**	**Description**	
19:16	mode_b	R/W	0	**Timer Mode Select TimerB** See *Table 18-4. MAX78000 Operating Mode Availability, Base Package* for which modes are supported. 0x0: One-Shot 0x1:Continuous 0x2:Counter 0x3:PWM 0x4:Capture 0x5:Compare 0x6:Gated 0x7:Capture/Compare 0x8: Dual-Edge Capture 0x9: Reserved 0xA: Reserved 0xB: Reserved 0xC: Reserved 0xD: Reserved 0xE: Inactive gated 0xF: Reserved	
15	en_a	R/W	0	**Enable TimerA** 0: Disabled 1: Enabled	
14	clken_a	R/W	0	**Timer Clock Enable TimerA** 0: Disabled 1: Enabled	
13	rst_a	W1	0	**RST TimerA** 0: No action 1: Reset all FF in CLK_TMR domain	
12	pwmckbd_a	R/W	1	**PWM Output $\phi_{A'}$ Disable TimerA** 1: Disable PWM Output $\phi_{A'}$ 0: Enable PWM Output $\phi_{A'}$	
11	nollpol_a	R/W	0	**PWM Output $\phi_{A'}$ Polarity Bit TimerA** 1: Output $\phi_{A'}$ inverted 0: Output $\phi_{A'}$ non-inverted	
10	nolhpol_a	R/W	0	**PWM Output ϕ_A Polarity Bit TimerA** 1: Output ϕ_A inverted 0: Output ϕ_A non-inverted	
9	pwmsync_a	R/W	0	**PWM Synchronization Mode TimerA** 0: Disabled 1: Enabled	
8	pol_a	R/W	0	**Timer Polarity TimerA** Selects the polarity of the timer's input and output signal. This setting is not used if the GPIO is not configured for the alternate function. The pol field meaning is determined by the specific mode of the timer. See the mode's detailed configuration section for pol usage.	

<table>
<tr><th colspan="4">Timer Control 0</th><th>TMRn_CTRL0</th><th>0x0010</th></tr>
<tr><th>Bits</th><th>Field</th><th>Access</th><th>Reset</th><th colspan="2">Description</th></tr>
<tr><td>7:4</td><td>clkdiva</td><td>R/W</td><td>0</td><td colspan="2">Timer Prescaler Select TimerA
The prescaler divides the clock input to the timer and sets the timer's count clock, $f_{CNT\_CLK} = {CNT\_CLK\ (Hz)}/{prescaler}$. Refer to the operational mode details to determine which modes use the prescaler.
<table>
<tr><th>72) pres</th><th>73) Prescaler</th></tr>
<tr><td>74) 0x0</td><td>75) 1</td></tr>
<tr><td>76) 0x1</td><td>77) 2</td></tr>
<tr><td>78) 0x2</td><td>79) 4</td></tr>
<tr><td>80) 0x3</td><td>81) 8</td></tr>
<tr><td>82) 0x4</td><td>83) 16</td></tr>
<tr><td>84) 0x5</td><td>85) 32</td></tr>
<tr><td>86) 0x6</td><td>87) 64</td></tr>
<tr><td>88) 0x7</td><td>89) 128</td></tr>
<tr><td>90) 0x8</td><td>91) 256</td></tr>
<tr><td>92) 0x9</td><td>93) 512</td></tr>
<tr><td>94) 0xA</td><td>95) 1024</td></tr>
<tr><td>96) 0xB</td><td>97) 2048</td></tr>
<tr><td>98) 0xC</td><td>99) 4096</td></tr>
<tr><td>100) 0xD</td><td>101) Reserved</td></tr>
<tr><td>102) 0xE</td><td>103) Reserved</td></tr>
<tr><td>104) 0xF</td><td>105) Reserved</td></tr>
</table></td></tr>
</table>

Timer Control 0				TMRn_CTRL0	0x0010
Bits	**Field**	**Access**	**Reset**	**Description**	
3:0	mode_a	R/W	0	**Timer Mode Select TimerA** See *Table 18-4. MAX78000 Operating Mode Availability, Base Package* for which modes are supported. 0x0: One-Shot 0x1:Continuous 0x2:Counter 0x3:PWM 0x4:Capture 0x5:Compare 0x6:Gated 0x7:Capture/Compare 0x8: Dual-Edge Capture 0x9: Reserved 0xA: Reserved 0xB: Reserved 0xC: Reserved 0xD: Reserved 0xE: Inactive gated 0xF: Reserved	

Table 18-11: Timer Non-Overlapping Compare Register

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

Table 18-12: Timer Control 1 Register

Timer Control 1				TMRn_CTRL1	0x0014
Bits	**Field**	**Access**	**Reset**	**Description**	
31	cascade	R/W	0	**32-bit Cascade Timer Enable** This field is only used by instances that support the 32-bit cascade figuration. 0: Dual 16-bit timers 1: Cascade TimerB:TimerA into one 32-bit timer TimerA	
30:29	-	RO	0	**Reserved**	
28	we_b	R/W	0	**Timer Wakeup Function TimerB** 0: Disabled 1: Enabled.	

Timer Control 1				TMRn_CTRL1	0x0014
Bits	**Field**	**Access**	**Reset**	**Description**	
27	sw_capevent_b	R/W	0	**Software Capture Event TimerB** The bit is just like TMR_CAP_EVENT_Ix and TMR_GATE_I in the capture related mode(capture, capture/compare, dual-edge capture). It will create positive/negative edge to create the capture event. 0: Event=0 1: Event=1	
26:25	capevent_sel_b	R/W	0	**Capture Event Selection TimerB** 0x0: TMR_GATE_I 0x1: Cap event 0 0x2: Cap event 1 0x3: SW_CAPEVENT	
24	ie_b	R/W	0	**IRQ Interrupt Enable TimerB** 0: Disabled 1: Enabled	
23	negtrig_b	R/W	0	**Negative Edge Trigger for Event TimerB** 0: Positive-edge trigger 1: Negative-edge trigger	
22:20	event_sel_b	R/W	0	**Event Selection TimerB** 0x0: Event disable 0x1: Start event 0 0x2: Start event 1 0x3: Start event 2 0x4: Start event 3 0x5: Reserved 0x6: Reserved 0x7: Reserved	
19	clkrdy_b	R/W	0	**TimerB Clock Ready** 0: Timer not enabled or synchronization in progress 1: Timer is enabled	
18	clken_b	R/W	0	**TimerB Clock Enable** 0: Timer not enabled or synchronization in progress 1: Timer is enabled	
17:16	clksel_b	R/W	0	**Clock Source TimerB** See *Table 18-1. MAX78000 TMR/LPTMR Instances* for the clock sources supported by each instance. 0x0: Peripheral clock 0x1: Timer Clock 1 0x2: Timer Clock 2 0x3: Timer Clock 3	
15	-	RO	0	**Reserved**	
14	outben_a	R/W	0	**TMR_PWM_CKB_EN_O Enable TimerA** 1: TMR_PWM_CKB_EN_O=1 0: TMR_PWM_CKB_EN_O=1 when complimentary output enabled in PWM mode 0: Disabled 1: Enabled	
13	outen_a	R/W	0	**Timer Output Signal Enable MR_OUT_OE_O Enable TimerA** 1: TMR_OUT_OE_O=1 0: TMR_OUT_OE_O=1 when in counter/capture/gated/capture-compare/dual-edge capture/inactive-gated mode 0: Disabled 1: Enabled.	

Timer Control 1				TMRn_CTRL1 0x0014
Bits	**Field**	**Access**	**Reset**	**Description**
12	we_a	R/W	0	**Timer Wakeup Function TimerA** 0: Disabled 1: Enabled.
11	sw_capevent_a	R/W	0	**Software Capture Event TimerA** The bit is just like TMR_CAP_EVENT_Ix and TMR_GATE_I in the capture related mode(capture, capture/compare, dual-edge capture). It will create positive/negative edge to create the capture event. 0: Event=0 1: Event=1
10:9	capeventsel_a	R/W	0	**Capture Event Selection TimerA** 0x0: TMR_GATE_I 0x1: Cap event 0 0x2: Cap event 1 0x3: SW_CAPEVENT 0x4: Reserved 0x5: Reserved 0x6: Reserved 0x7: Reserved
8	ie_a	R/W	0	**IRQ Interrupt Enable TimerA** 0: Disabled 1: Enabled
7	negtrig_a	R/W	0	**Negative Edge Trigger for Event TimerA** 0: Positive-edge trigger 1: Negative-edge trigger
6:4	event_sel_a	R/W	0	**Event Selection TimerA** 0x0: Event disable 0x1: Start event 0 0x2: Start event 1 0x3: Start event 2 0x4: Start event 3 0x5: Reserved 0x6: Reserved 0x7: Reserved
3	clkrdy_a	R/W	0	**Timer Clock Ready TimerA** 0: Not ready 1: Ready
2	clken_a	R/W	0	**Timer Enable Sync TimerA** 0: Timer not enabled or synchronization in progress 1: Timer is enabled
1:0	clksel_a	R/W	0	**Clock Source TimerA** *See Table 18-1. MAX78000 TMR/LPTMR Instances* for the clock sources supported by each instance. 0x0: Peripheral clock 0x1: Timer Clock 1 0x2: Timer Clock 2 0x3: Timer Clock 3

Table 18-13: Timer Wakeup Status Register

Timer Wakeup Status				TMRn_WKFL 0x0010
Bits	**Field**	**Access**	**Reset**	**Description**
31:17	-	RO	0	**Reserved**

Timer Wakeup Status				TMRn_WKFL	0x0010
Bits	**Field**	**Access**	**Reset**	**Description**	
16	b	R/W	1	**Wakeup Event TimerB** 0: No event 1: Wakeup event occurred	
15:1	-	RO	0	**Reserved**	
0	a	R/W	1	**Wakeup Event TimerB** 0: No event 1: Wakeup event occurred	

19. Bootloader

The bootloader provides for secure and unsecure program loading and debug. The physical interface between the external host and the bootloader is by default the UART, but the serial wire debug (SWD) can also be used.

Features:

- Common Functionality (Modes: General Use and Secure Boot)
- Checksum verification of ROM image before further ROM execution
- Less than 1 second boot time
- SWD enabled only with device in UNLOCKED state
- UART operates at 115200 bps
- Flash programming blocked via SWD or UART bootloader when device is Locked
- LOCKED mode disables SWD and disallows any change to flash via bootloader
- Unlock erases all flash and secrets before unlocking SWD
- Optional PERMLOCKED state only allows for program validation and Permanent Lock
- Authentication of user code before execution via Secret Keyed HMAC-SHA256
- Customer is responsible to write secret key to device via SWD or UART bootloader
- Customer is responsible to write length of flash code image to device via SWD or UART bootloader
- Optional Challenge/Response gating entry to bootloader

19.1 Instances

The dedicated pins and features of the bootloader are shown in *Table 19-1:MAX78000 Bootloader Interface*:

Table 19-1:MAX78000 Bootloader Interface

PART NUMBER	UART0 RX PIN	SWDCLK PIN	SECURE BOOT
MAX78000	GPIO0.0	GPIO0.29	Yes

19.2 Bootloader Operating Modes

Each bootloader supports the modes shown in *Table 19-2: MAX78000 Bootloader Operating Modes and* Prompts. Each bootloader mode has a unique prompt.

Table 19-2: MAX78000 Bootloader Operating Modes and Prompts

MODE	RECOGNIZED COMMANDS	PROMPT
UNLOCKED	All Commands U/L/P	"ULDR> " <0x55> <0x4C> <0x44> <0x52> <0x3E> <0x20>
LOCKED	Only L/P	"LLDR> " <0x4C> <0x4C> <0x44> <0x52> <0x3E> <0x20>
PERMLOCK	Only P	"PLLDR> " <0x50> <0x4C> <0x4C> <0x44> <0x52> <0x3E> <0x20>
CHALLENGE	*GC – Get Challenge* *SR – Send Response*	"CR> " <0x50> <0x4C > <0x3E> <0x20>
APPVERIFY	N/A	N/A

19.2.1 UNLOCKED

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

Transitioning from the LOCKED to UNLOCKED states erases memory and clears stored keys in the secure version of the bootloader.

19.2.2 LOCKED

The LOCKED state disables access to program memory other than to verify it. It also enables security features other provides access to load keys

Transitioning from the LOCKED to UNLOCKED states erases memory and clears stored keys in the secure version of the bootloader.

19.2.3 PERMLOCKED

The PERMLOCKED state disables access to program memory other than to verify it. The commands available in PERMLOCKED state are shown in *Table 19-3: MAX78000 PERMLOCK Command Summary*.

Table 19-3: MAX78000 PERMLOCK Command Summary

Command
H – Check Device
I – Get ID

If the challenge key is activated, the device will start in the CHALLENGE mode, before transitioning to the PERMLOCKED mode.

19.2.4 CHALLENGE

The device enters CHALLENGE mode immediately following a reset if the challenge key has been activated. Upon receiving the CHALLENGE mode prompt shown in *Table 19-2: MAX78000 Bootloader Operating Modes and Prompts*, the host must execute the GC command and calculate the appropriate response. The host must execute the SR command as the next command or an error will occur.

Following a successful response, the bootloader will return the prompt appropriate to the current state.

Table 19-4: MAX78000 CHALLENGE Command Summary

Command
GC – Get Challenge
SR – Send Response

If the challenge key is activated, the device will start in the CHALLENGE mode, before transitioning to the PERMLOCKED mode.

19.2.5 APPVERIFY

The APPVERIFY mode is not a user-accessible mode. It defines the state of a secure device when the device exits reset with the following conditions:

1. The device does not detect the bootloader activation pins.
2. The application key has been previously enabled

The device then performs the operation described in *Automatic Application Validation on Reset*.

19.3 Bootloader Activation

Perform the following sequence to activate the bootloader:

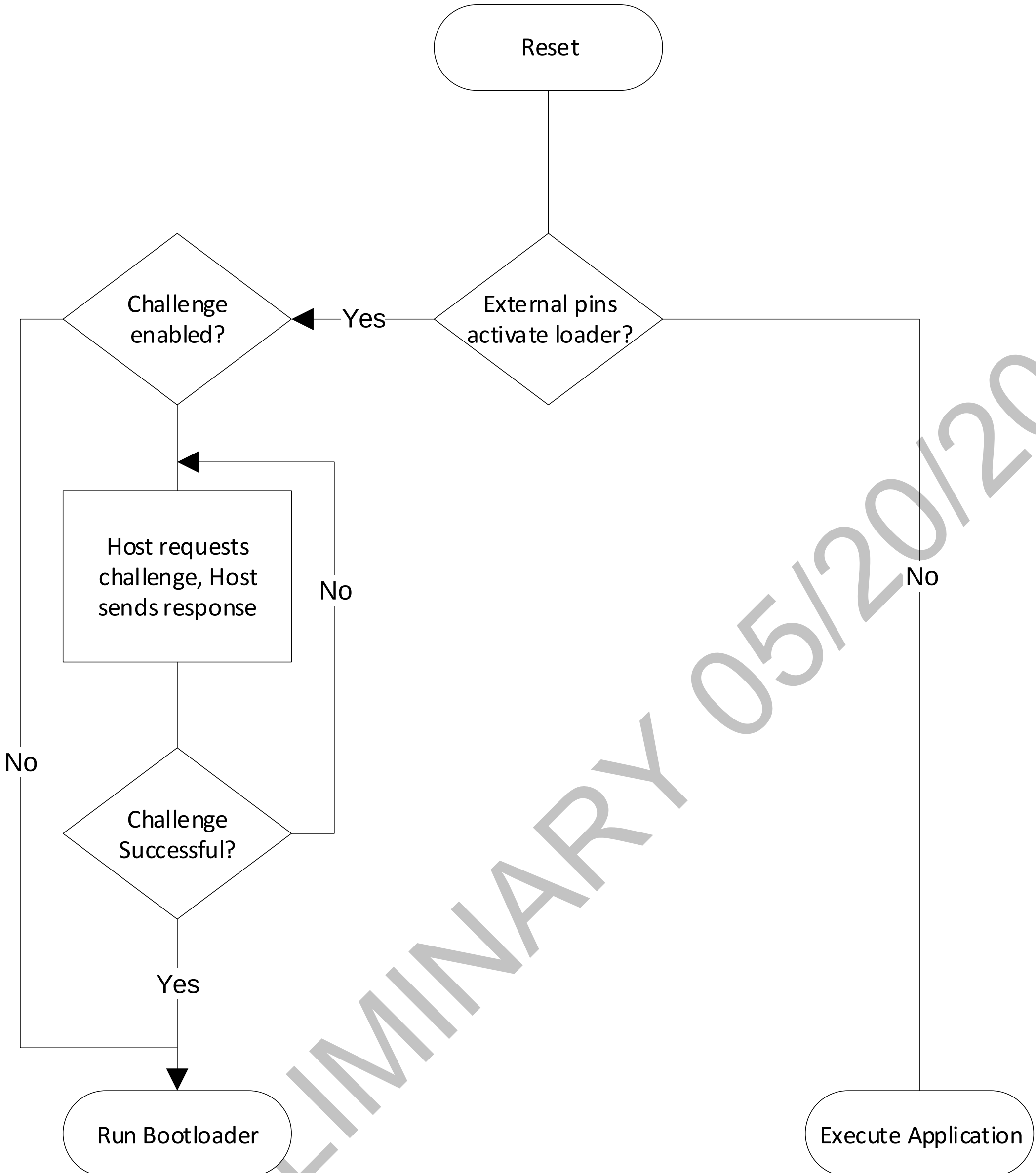
1. Host asserts the UART0 RX pin and SWDCLK pins low as shown in in *Table 19-1:MAX78000 Bootloader Interface*.
2. Host asserts RSTN pin low.
3. Host deasserts the RSTN pin
4. Host samples the UART0 RX and SWDCLK pins. If they are both low the hardware will activate the bootloader.
5. Hardware will perform its system initialization and configure the bootloader for 115200 bps.
6. The bootloader will output the status prompt the UART0 TX pin. The prompt is unique for each bootloader mode as shown in *Table 19-2: MAX78000 Bootloader Operating Modes and* Prompts.
7. The host releases the UART0 RX and SWDCLK pins once the host confirms the correct bootloader prompt.
8. Host begins bootloader session by sending commands on the UART0 RX pin.

19.4 Non-Secure Bootloader

The non-secure version of the bootloader provides the basic functionality of the bootloader including the Lock and Unlock command features. It also supports transition into the PERMLOCKED state.

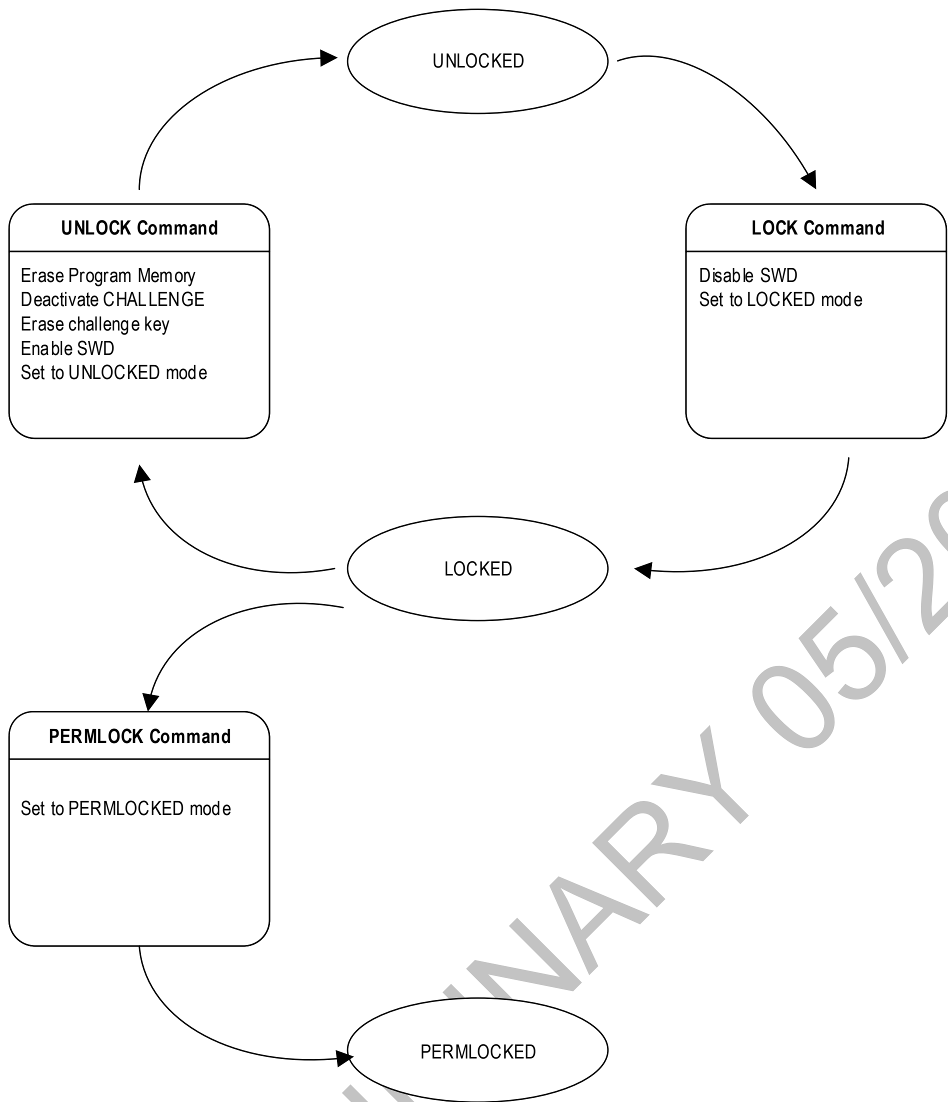
The flow chart of the operation of the non-secure bootloader following a reset is shown in *Figure 19-1: Non-Secure Bootloader Flow*.

Figure 19-1: Non-Secure Bootloader Flow



The flow chart of the operation of the non-secure bootloader following a reset is shown in *Figure 19-1: Non-Secure Bootloader Flow*.

Figure 19-2: Non-Secure Bootloader State Diagram



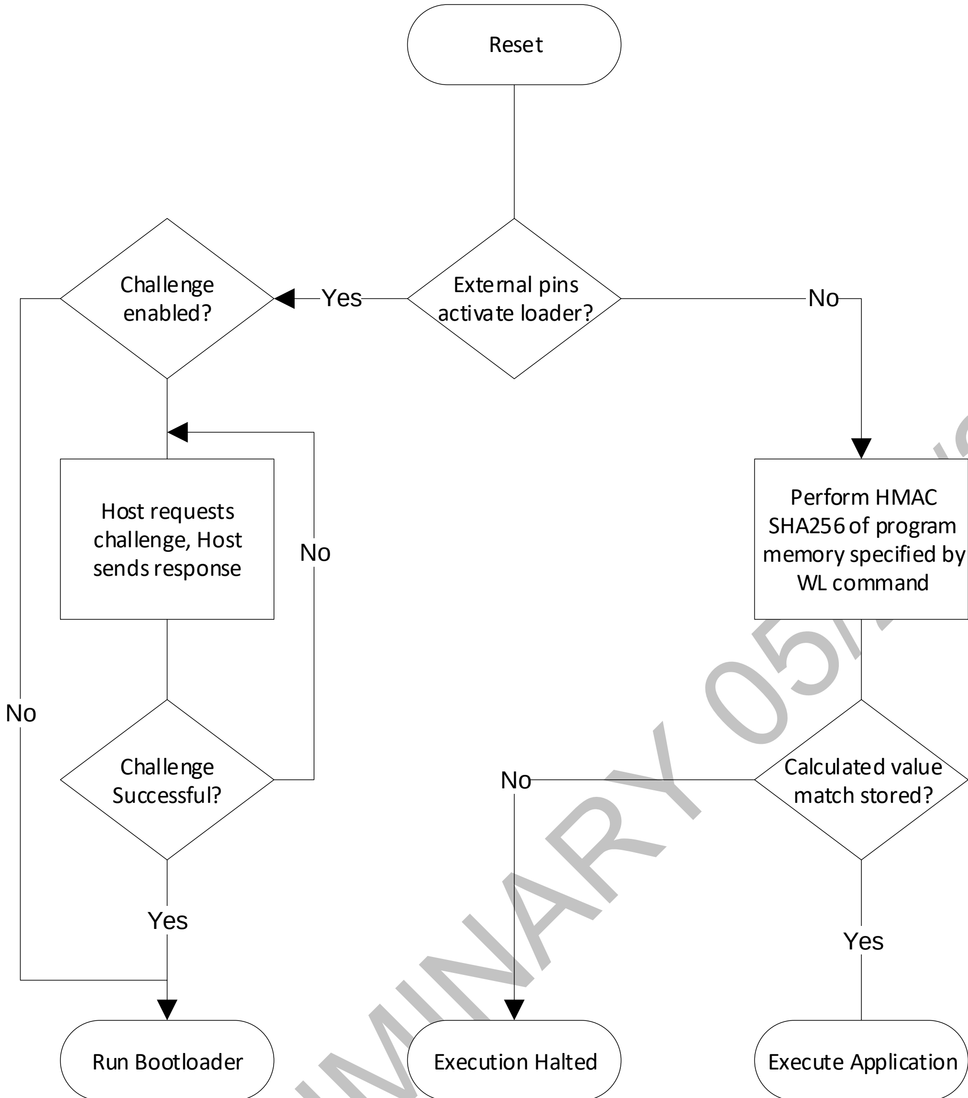
19.5 Secure Bootloader

The secure version of the bootloader provides additional features for secure and authenticated loading. The flow chart of the operation of the secure bootloader following a reset is shown in *Figure 19-3: Secure Bootloader Flow*.

Loading and activating the application key will cause the device to perform *Automatic Application Validation on Reset*. The hashed value is then compared against the application key previously stored. A correct value indicates the program memory has not been altered and control will transfer to the application.

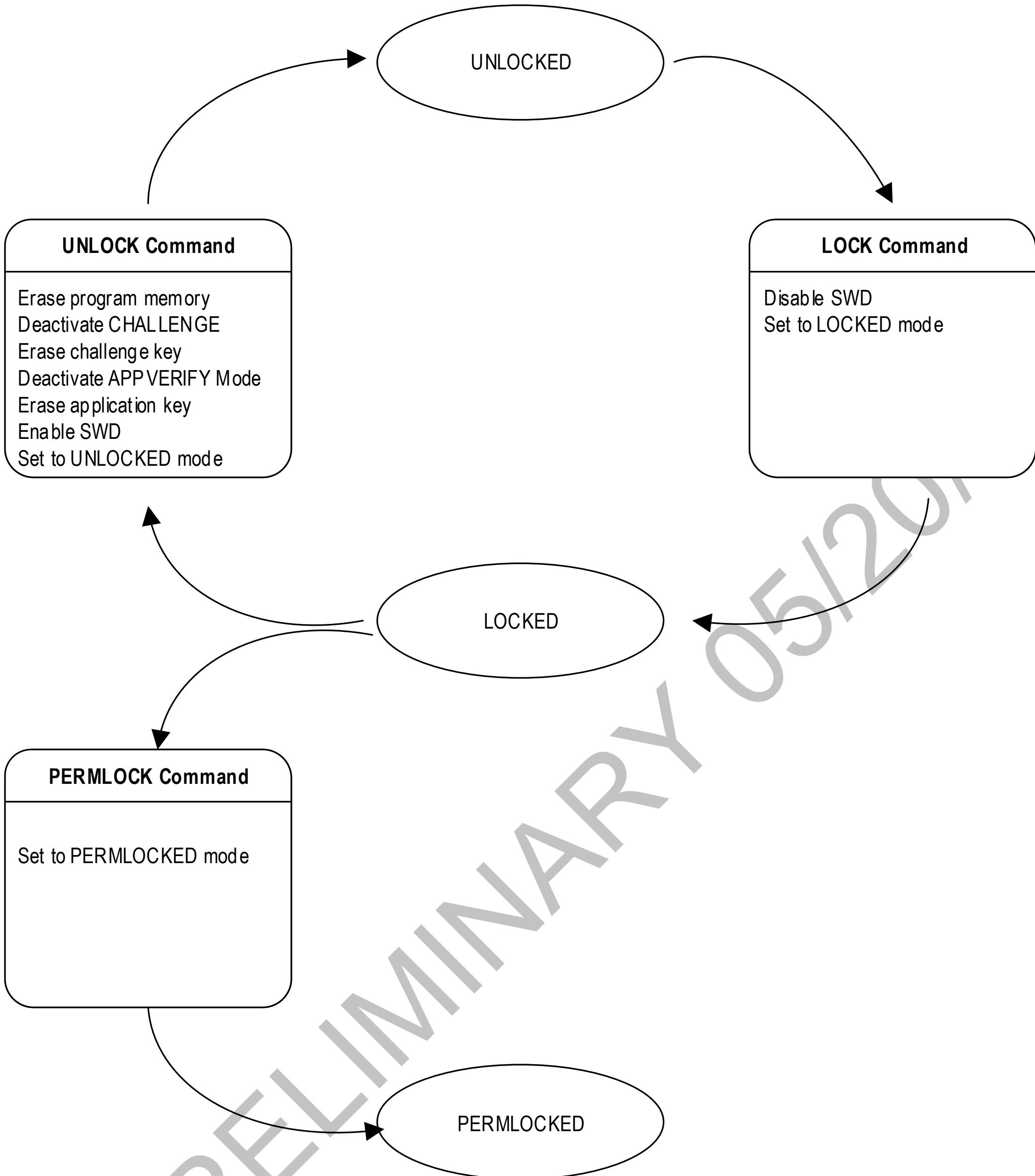
A mismatch between the hash of the program memory and the stored value will cause the device to halt execution and remain in a continual reset state.

Figure 19-3: Secure Bootloader Flow



The state diagram for the secure bootloader modes is shown in *Figure 19-4: Secure Bootloader State Diagram*.

Figure 19-4: Secure Bootloader State Diagram



19.5.1 Automatic Application Validation on Reset

The device can verify the contents of its application each time the device reset a reset. This prohibits the device from executing corrupted or tampered program software.

When enabled, the device performs a HMAC SHA-256 hash of the application code and compares it against a calculated value previously loaded through the bootloader. A correct match allows the device to begin application code at the restart vector. A failure of the two values to match will prevent any application execution and render the device unresponsive.

Follow this procedure to enable the Application Validation feature:

1. Calculate the HMAC SHA-256 of program memory using the secret key.
2. Activate the bootloader as described in the *Bootloader Activation* section.
3. Ensure the device returns the unlocked prompt.
4. Execute the WL command to specify the range over which the hash will be performed.
5. Execute the L command to load program memory.
6. Execute the V command to verify the program was correctly loaded.
7. Execute the LK command to load the HMAC SHA-256 secret key.
8. Execute the VK command to verify the HMAC SHA-256 secret key was correctly loaded.
9. Execute the AK command. Device will automatically verify program memory on all subsequent resets.

19.5.2 Secure Challenge/Response Authorized Access

The optional secure Challenge/Response in mode prevents access to the bootloader by a host that does not know the unique HMAC-SHA256 challenge key. When enabled in the LOCKED or PERMLOCKED states, the device will enter CHALLENGE mode any bootloader commands can be executed.

When the challenge/response feature has been enabled, the device will return the challenge/response prompt shown in *Table 19-2: MAX78000 Bootloader Operating Modes and Prompts*. The host requests a 128 bit random challenge from the host, performs a HMAC SHA26 hash on the challenge, and sends the response back to the bootloader. A successful response returns the prompt and access to the loader.

Follow this procedure to enable the configure and enable the Challenge/Response feature:

1. Execute the LK command to load the Challenge/Response HMAC SHA-256 secret key.
2. Execute the VK command to verify the Challenge/Response HMAC SHA-256 secret key was correctly loaded.
3. The Challenge/Response will be required after the next device reset. It does affect current operation until the next reset.

Follow this procedure to successful perform the Challenge/Response:

1. Execute the GC command to generate a random challenge
2. Host performs HMAC SHA-256 of the bootloader challenge to create the response.
3. Execute the SR command with the calculated response.

A correct response will return the prompt. An incorrect response will return an error message and the challenge response prompt again. The host can perform steps 1-3 again to request another challenge from the bootloader.

19.6 Command Protocol

The bootloader presents a mode-specific prompt based on the current state of the loader as shown as in *Table 19-2: MAX78000 Bootloader Operating Modes and Prompts*. The general format of commands is the ASCII character(s) of the command, followed by a <CR><LF> which is hexadecimical <0x0D><0x0A>. Commands with arguments always have a space (0x20) between the command mnemonic and the argument.

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

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

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

19.7 General Commands

Table 19-5: MAX78000 General Command Summary

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

19.8 General Command Details

Table 19-6: L - Load

L - Load	Load Motorola SREC File into Program Memory
Description	Load a Motorola SREC file format into Flash program memory. The end of the file is detected automatically, so there is no need to send <0x0D><0x0A>at the end. Address records must be 32-bit aligned and length of each line must be multiple of 4 bytes or the load will fail. A load success will confirm the base address and length of the program code as specified in the SREC file.
Modes	U

Command	`L<0x0D><0x0A>`
	`Ready to load SREC<0x0D><0x0A>`
	`[Motorola SREC File]`

Response: Success	`Load success, image loaded with the following parameters:<0x0D><0x0A>`
	`Base address: 0xnnnnnnnn<0x0D><0x0A>`
	`Length: 0xnnnnnnnn<0x0D><0x0A>`

Response: Failure	`Load failed.<0x0D><0x0A>`

Table 19-7: P – Page Erase

P – Page Erase	Erase Page of Flash Program Memory
Description	Erases specified page of memory. Argument is 32-bit address. Address must be aligned on 8K boundaries.
Modes	U
Command	P 0xnnnnnnnn<0x0D><0x0A>
Response: Success	Erase Page Address: 0xnnnnnnnn<0x0D><0x0A>OK<0x0D><0x0A>
Response: Failure	Bad page address input<0x0D><0x0A> or Erase failed<0x0D><0x0A> or Invalid Page Address: 0xnnnnnnnn<0x0D><0x0A>

Table 19-8: V – Verify

V – Verify	Verify Flash Program Memory Against Motorola SREC File
Description	Verifies contents of Flash program memory against a Motorola SREC file.
Modes	U

Command	`Verify<CR>`
	`Ready to verify SREC<0x0D><0x0A>`
	`[Motorola SREC File]`

Response: Success	`Verify success, image verified with the following parameters: <0x0D><0x0A>`
	`Base address: 0xnnnnnnnn<0x0D><0x0A>`
	`Length: 0xnnnnnnnn<0x0D><0x0A>`

Response: Failure	`Verify failed.<0x0D><0x0A>`

Table 19-9: LOCK – Lock Device

LOCK – Lock Device	**Lock Device**
Description	Locks the device and disables SWD on the next device reset. See *LOCKED* section for a detailed description of this command. The Lock command should be immediately followed by the Q command (which generates a device reset) for the Lock command to take effect. A failure response indicates a defective device.
Modes	U
Command	`LOCK<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`
Response: Failure	`Failed<0x0D><0x0A>`

Table 19-10: PLOCK – Permanent Lock

PLOCK – Permanent Lock	**Permanently Lock Device**
Description	Permanently locks the device if the argument matches the device ID.
Modes	U/L
Command	`PLOCK <USN>`
Response: Success	`OK<0x0D><0x0A>`
Response: Failure	`Failed<0x0D><0x0A>`

Table 19-11: UNLOCK – Unlock Device

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

Table 19-12: H – Check Device

H – Check Device	Perform SHA-256 Hash of Specified Flash Memory
Description	Perform a HMAC SHA-256 hash of bytes starting at 32-bit address 0xnnnnnnnn to 0xmmmmmmmm. Minimum hash input size is 512 bytes. Function returns 16-byte hash value.
Modes	U/L/P
Command	H 0xnnnnnnnn 0xmmmmmmmm<0x0D><0x0A>
Response: Success	yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>
Response: Failure	<0x0D><0x0A>

Table 19-13: I – Get ID

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

Table 19-14: S – Status

S – Status	Read Device Status
Description	Returns the state of the loader and the application key and challenge key features: The Lock <response> is: "Inactive" if the device is in the unlocked state. "Active" if the device is in the locked or permanent lock state. The PLock <response> is: "Inactive" if the device is in the unlocked or locked state. "Active" if the device is in the permanent lock state. The Application Length <response> is: "Not Set" if the Write Code Length command has not previously loaded a non-zero value "0xnnnnnnnn" which is the previously entered value using the Write Code Length command. The Application Key <response> is: "None Inactive" if no application key has been loaded using the LK command. "Loaded Inactive" if the application key has been loaded but the application key feature has not been activated by the AK command "Loaded Active" If the application key has been loaded and the application key feature has been activated. The Challenge Key <response> is: "None Inactive" if no challenge key has been loaded using the LK command. "Loaded Inactive" if the challenge key has been loaded but the challenge key feature has not been activated by the AK command "Loaded Active" if the challenge key has been loaded and the challenge key feature has been activated.
Modes	U
Command	S<0x0D><0x0A> Status<0x0D><0x0A> Lock: <response><0x0D><0x0A> PLock: <response><0x0D><0x0A> Application Length: <response><0x0D><0x0A> Application Key: <response><0x0D><0x0A> Challenge Key: <response><0x0D><0x0A>
Response: Success	None.
Response: Failure	None.

Table 19-15: Q – Quit

Q – Quit	**Quit Bootloader Session**
Description	Terminates the bootloader session and forces a reset of the device.
Modes	`U/L/P`
Command	`Q<0x0D><0x0A>`
Response: Success	None
Response: Failure	None

19.9 Secure Commands

Table 19-16: MAX78000 Secure Command Summary

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

19.10 Secure Command Details

Table 19-17: LK – Load Application Key

LK – Load Application Key	Load Application HMAC-SHA256 Key
Description	Write 128-bit HMAC secret key to non-volatile memory.
Modes	U
Command	`LK yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Bad key input<0x0D><0x0A>`
	or
	`Key already loaded<0x0D><0x0A>`

Table 19-18: LK – Load Challenge Key

LC – Load Challenge Key	Load Challenge Key
Description	Write 128-bit challenge key to non-volatile memory.
Modes	U
Command	LC yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>
Response: Success	OK<0x0D><0x0A>
Response: Failure	Bad key input<0x0D><0x0A> or Error, no key loaded<0x0D><0x0A> or Key Mismatch<0x0D><0x0A>

Table 19-19: VK – Verify Application Key

VK – Verify Application Key	VK – Verify Application Key
Description	Verify the Application Key against a value provided by the Host.
Modes	U
Command	`LK yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Bad key input<0x0D><0x0A>`
	or
	`Error, no key loaded<0x0D><0x0A>`
	or
	`Key Mismatch<0x0D><0x0A>`

Table 19-20: VC – Verify Challenge Key

VC – Verify Challenge Key	**VC – Verify Challenge Key**
Description	Verify the Challenge Key against a value provided by the Host.
Modes	U
Command	`LK yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Bad key input<0x0D><0x0A>`
	or
	`Error, no key loaded<0x0D><0x0A>`
	or
	`Key Mismatch<0x0D><0x0A>`

Table 19-21: AK – Activate Application Key

AK – Activate Application Key	**Activate Application Key**
Description	Activate application key. All application software loads must be encrypted with the application key. The UNLOCK command deactivates the application key.
Modes	U
Command	`AK<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Key activate failed<0x0D><0x0A>`
	or
	`Error, no key loaded<0x0D><0x0A>`

Table 19-22: AC – Activate Challenge Key

AC – Activate Challenge Mode	**Activate Challenge Mode**
Description	Activate CHALLENGE mode. All subsequent bootloader sessions in LOCKED or PERMLOCKED states will start in CHALLENGE mode. by the
Modes	U
Command	`AC<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Key activate failed<0x0D><0x0A>`
	or
	`Error, no key loaded<0x0D><0x0A>`

Table 19-23: WL – Write Code Length

WL – Write Code Length	**Write Code Length**
Description	Write the length of the application code in bytes. This specifies the range code length in bytes for the application key verification.
Modes	U
Command	`WL 0xnnnnnnnn<0x0D><0x0A>`
Response: Success	`Length set to: 0xnnnnnnnn<0x0D><0x0A>`

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

19.11 Challenge/Response Commands

Table 19-24: MAX78000 Challenge/Response Command Summary

Register Name
GC – Get Challenge
SR – Send Response

19.12 Challenge/Response Command Details

Table 19-25: GC – Get Challenge

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

Table 19-26: SR – Send Response

SR – Send Response	Send Response
Description	Host calculates HMAC SHA-256 on the 16-byte challenge and sends the 32-byte hexadecimal ASCII response to bootloader. The response is sent MSB first.
Modes	L
Command	`SR 0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF<0x0D><0x0A>`
Response: Success	`OK<0x0D><0x0A>`

Response: Failure	`Bad response input<0x0D><0x0A> (Not a valid hexadecimal ASCII string)`
	Or
	`Verification failed<0x0D><0x0A> (Incorrect response)`
	Or
	`Error, request challenge<0x0D><0x0A> (Response sent before challenge request)`

20. Revision History

REVISON NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0.1	5/20/20	Initial Release	-