# MAX32650–MAX32652 USER GUIDE

*UG6766; Rev 0; 9/18*

**Abstract:** This user guide provides application developers information on how to use the memory and peripherals of the MAX32650–MAX32652 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.

# MAX32650–MAX32652 User Guide

## Table of Contents

## List of Figures

## List of Tables

# 1 Overview

The MAX32650–MAX32652 are low-power, mixed signal microcontrollers based on the Arm® Cortex®-M4 with FPU CPU, operating at a maximum frequency of 120MHz. The devices feature five powerful and flexible power modes. A SmartDMA performs complex background processing on data being transferred, from simple arithmetic to multiply/accumulate, while the CPU is off. This function dramatically reduces overall power consumption compared to conventional solutions. This allows, for example, an external display to be refreshed while most of the chip is powered off. Built-in dynamic clock gating and firmware-controlled power gating allows the user to optimize power for the specific application.

Application code executes from an onboard 3MB program flash memory, with 1MB SRAM available for general application use. A 16KB cache improves execution throughput. Additionally, a SPI execute in place (XIP) external memory interface allows application code and data (up to 128MB) to be accessed from an external SPI flash and/or SRAM memory device. A 10-bit delta-sigma ADC is provided with a multiplexer front end for four external input channels (two of which are 5V tolerant) and six internal power supply monitoring channels. Dedicated divided supply input channels allow direct monitoring of internal power supply voltages by the ADC. Built-in limit monitors allow converted input samples to be compared against user-configurable high and low limits, with an option to trigger an interrupt and wake the CPU from a low power mode if attention is required.

A wide variety of communications and interface peripherals are provided, including a Hi-Speed USB 2.0 device interface, three master/slave SPI interfaces, one QuadSPI master/slave interface, three UART interfaces with flow control support, two master/slave $I^2C$ interfaces, $I^2S$ bidirectional slave interface. A Cypress® Spansion® HyperBus® interface and a Xccela® Bus interface provides support for HyperFlash®, HyperRAM® and Xccela PSRAM operating up to 120MB/s throughput with access up to 512MB. A SD/SDIO/MMC interface running up to 60MB/s supporting media file storage. A 24-bit TFT LCD controller provides color and monochrome display support.

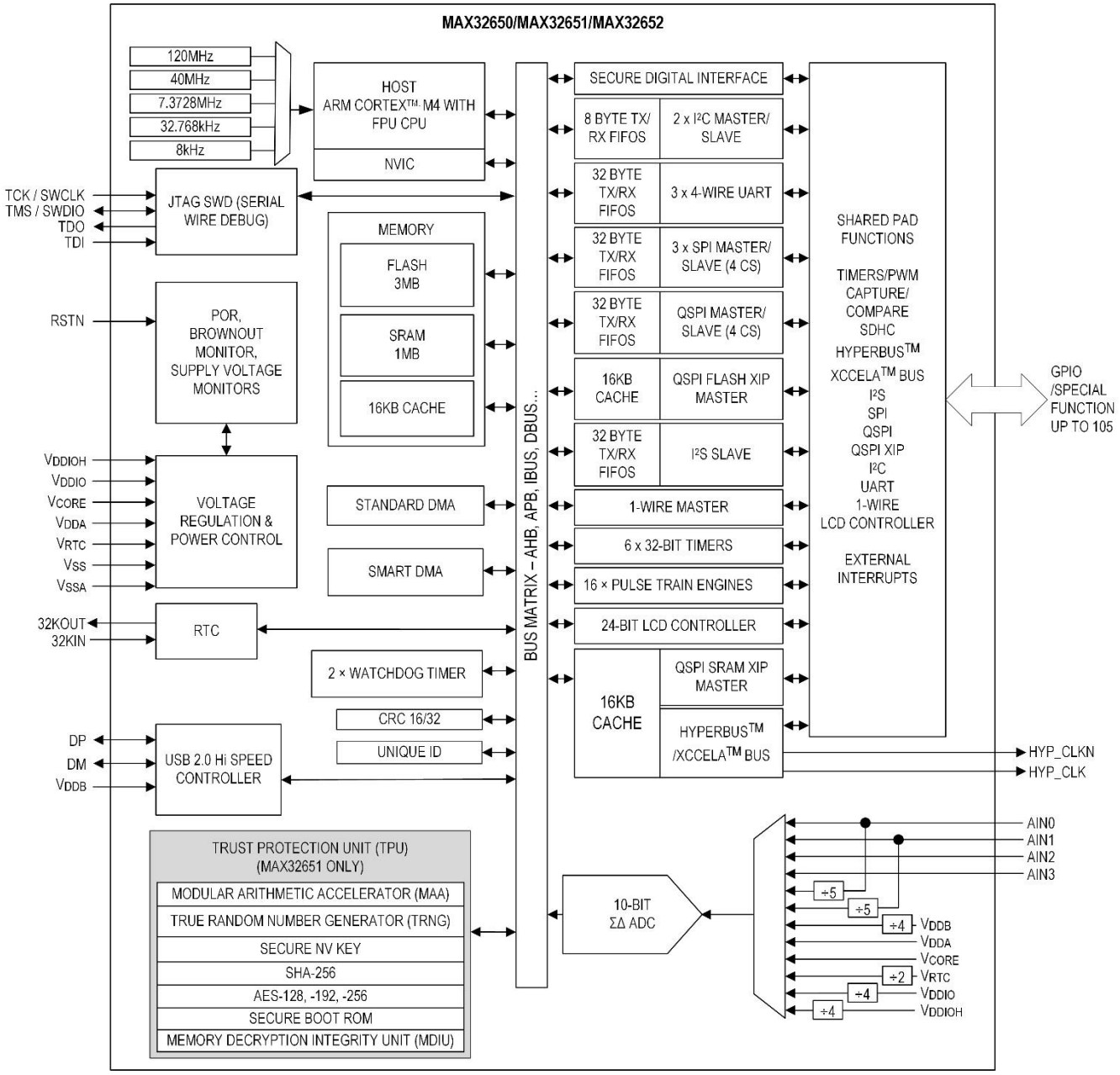
The MAX32651 is a secure version of the MAX32650–MAX32652. It provides a trust protection unit (TPU) with encryption and advanced security features. These features include a modular arithmetic accelerator (MAA) for fast ECDSA and RSA-4096 computation. A hardware AES engine uses 128/192/256-bit keys. A memory decryption integrity unit (MDIU) provides on-the-fly code or data decryption stored in external flash. A hardware TRNG and a hardware SHA-256 HASH function are also provided. A secure bootloader authenticates applications before they are allowed to execute and update firmware with confidentiality.

The MAX32652 is a high-density, 0.35mm pitch, 140-bump WLP targeted for tiny form factor products that require high I/O counts.

The high-level block diagram for the MAX32650–MAX32652 is shown in *Figure 1-1*, *below*.

## 1.1 Block Diagram

*Figure 1-1: MAX32650–MAX32652 Block Diagram*




The device features five flexible power modes. A smart DMA engine reduces power consumption by executing data collection tasks while the Arm Cortex-M4 with FPU core is powered off. Application-controlled power gating combined with built-in, dynamic clock gating optimizes power consumption.

Multiple SPI, UART, and I²C serial interfaces; a 1-Wire® Master; and a USB slave enable interconnection to a wide variety of external sensors. A four-input, 10-bit ADC with selectable references is available to monitor analog input from external power sources, sensors, and meters. Also, an I²S port is available for audio streaming to a codec.

External memory is supported using multiple industry standard interfaces. A HyperBus SRAM interface operating at a double-data rate of 96 MB/second can access up to 1 GB. A Secure Digital High Capacity (SDHC) interface provides support for large file data storage.

The Trust Protection Unit (TPU) with encryption and advanced security features is available for enhanced security. The TPU includes a Modular Arithmetic Accelerator (MAA) for fast ECDSA, a hardware AES Engine, a hardware TRNG entropy generator, and a Secure Boot Loader.

# 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 in width (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: MAX32650–MAX32652 Code Memory Mapping*

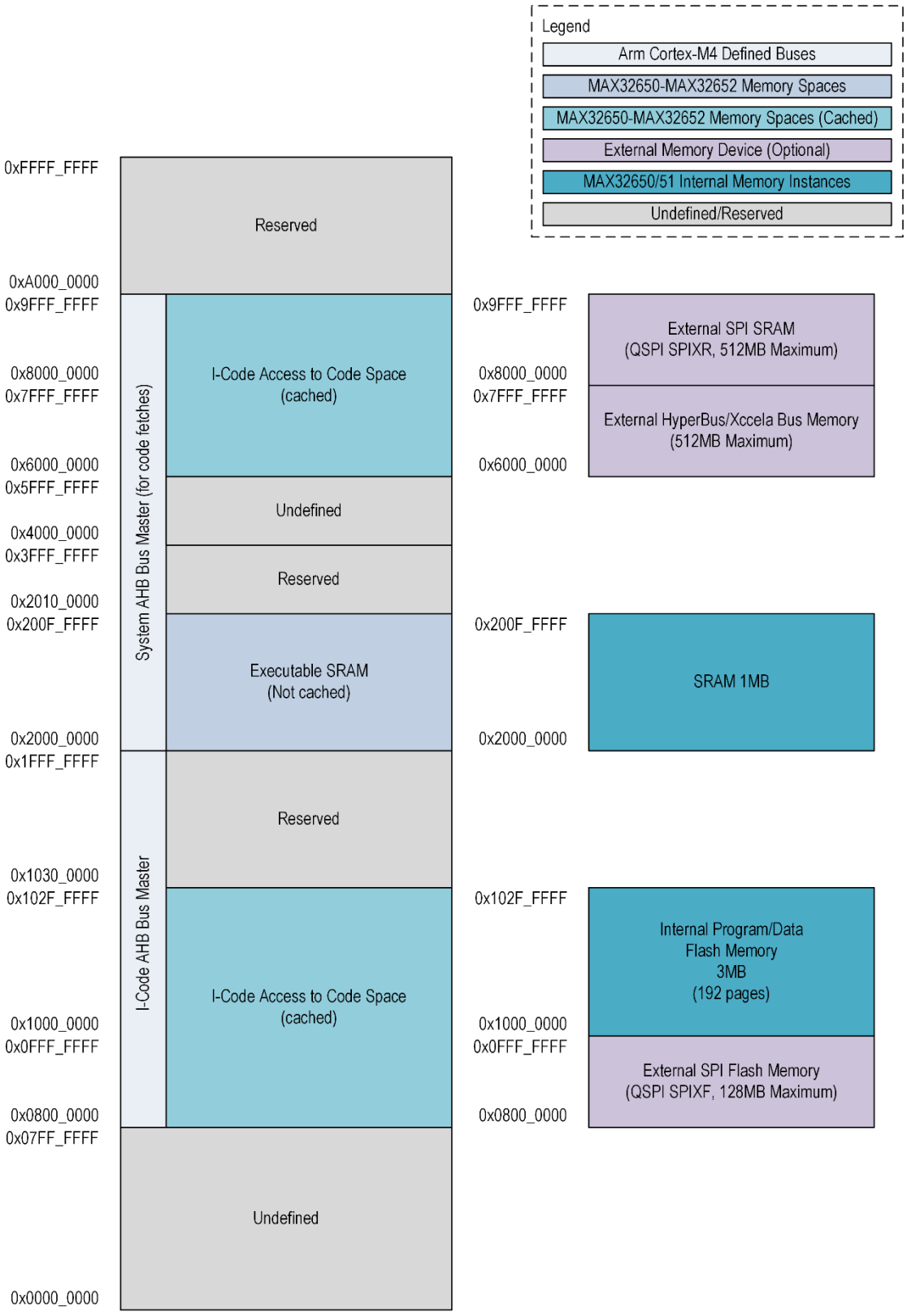
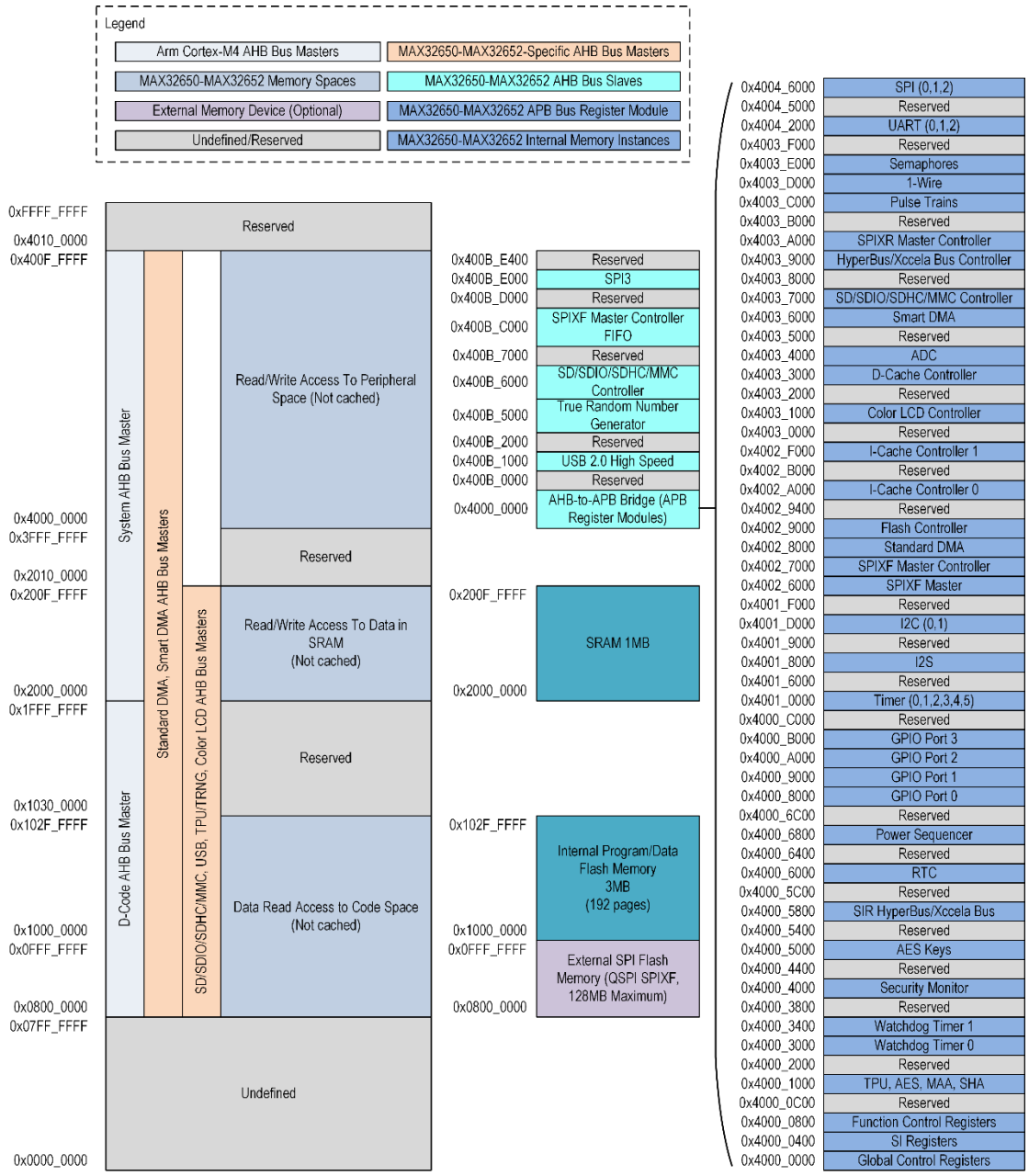


*Figure 2-2 Code Memory Mapping*




## 2.2 Standard Memory Regions

Several standard memory regions are defined for the Arm Cortex-M4 architecture; the use of many of these is optional for the system integrator. At a minimum Arm Cortex-M4-based devices must contain code and data memory for the application storage, variables and stack use, as well as certain components which are part of the instantiated core.

### 2.2.1 *Code Space*

The code space area of memory is designed to contain the primary memory used for code execution by the device. This memory area is defined from byte address range 0x0000 0000 to 0x1FFF FFFF (0.5GB maximum). Two different standard core bus 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.

On the MAX32650–MAX32652, 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 0x102F 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 where a vector to re-direct to 0x1000 0000 is located.

The code space memory on the MAX32650–MAX32652 also contains the mapping for the flash information block, from 0x1080 4000 to 0x1080 7FFF. However, this mapping is generally only present during 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.

Optionally, the SPIXF (SPI Execute In Place Flash) and the SPIXR (SPI Execute In Place RAM) modules can be used to expand the available code and data memory space for the MAX32650–MAX32652. This expansion consists of mapping the contents of an external SPI flash memory device (up to 128MB) or an external SPI RAM memory device (up to 512MB) into a read-only area of the code memory map. If enabled, the external SPIXF memory is mapped starting at byte address 0x0800 0000 up to a maximum of 0x0FFF FFFF (for a 128MB device). Also, if enabled, the external SPIXR memory is mapped starting at byte 0x8000 000 up to a maximum of 0x9FFF FFFF (for a 512MB device). This external memory can be used for code execution as well as static data storage.

The HyperBus/Xccela Bus interface module can be also be used to expand the available code and data memory space for the MAX32650–MAX32652. This expansion consists of mapping the contents of an external HyperBus or Xccela Bus memory device (up to 512MB) into a read-only area of the code memory map. If enabled, the HyperBus/Xccela Bus is mapped starting at byte address 0x6000 0000 up to a maximum of 0x7FFF FFF (for a 512MB device). This external memory can be used for code execution as well as static data storage.

### 2.2.2 *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.

On the MAX32650–MAX32652, this memory area contains the main system SRAM 1MB, which is mapped from 0x2000 0000 to 0x200F FFFF.

The entirety of the SRAM memory space on the MAX32650–MAX32652 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 1MB bit banding area to be accessed. Each 32-bit (4 byte aligned) address location in the bit-banding alias area translates into a single bit access (read or write) in the bit-banding primary area. Reading from the location performs a single bit read, while writing either a 1 or 0 to the location performs a single bit set or clear.

*Note: The Arm Cortex-M4 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 such as the Smart DMA AHB bus master will not trigger a bit-banding operation and will instead result in an AHB bus error.*

The SRAM area on the MAX32650–MAX32652 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 MAX32650–MAX32652 specific AHB Bus Masters can access the SRAM to use as general storage or working space. Specifically, in the case of the USB interface, SRAM memory area can be used to store the descriptor table for the endpoint buffers as well as the endpoint buffers themselves.

### 2.2.3 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 MAX32650–MAX32652, 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 (such as the Smart DMA AHB 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 MAX32650–MAX32652, access to the region that contains most peripheral registers (0x4000 0000 to 0x400F FFFF) goes from the AHB bus through an AHB-to-APB bridge. This allows the peripheral modules to operate on the slower, easier to handle APB bus matrix. This also ensures that peripherals with slower response times do not tie up bandwidth on the AHB bus, which must necessarily have a faster response time since it handles main application instruction and data fetching.

*Note: The APB bus supports 32-bit width access only. All access to the APB peripheral register area (0x4000 0000 to 0x400F FFFF) must be 32-bit width only with 32-bit (4 byte) alignment. Access using 8-bit or 16-bit width to this memory region is not supported and will result in an AHB memory fault exception (returned by the AHB-to-APB bridge interface).*

A secondary region within the peripheral memory space (0x0400B 0000 to 0x400F FFFF) allows peripherals that require more rapid data transfer to handle this data transfer using their own local AHB slave instances (instead of going indirectly through the AHB-to-APB bridge). This allows peripherals which have FIFOs or other functions requiring large amounts of data to be transferred quickly (such as the SD/SDIO/SDHC/MMC or communications peripherals like SPI) to benefit from the more rapid data transfer rate of the AHB bus.

*Note: Only 32-bit width access is permitted when reading or writing registers in the APB mapped area. Accesses of 8-bit width and 16-bit width are not supported and will result in an AHB bus fault.*

### 2.2.4 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 MAX32650–MAX32652 implements support for external HyperBus/Xccela Bus SRAM and external SPI SRAM. The external HyperBus/Xccela Bus interface is mapped to byte address 0x6000 0000 to 0x7FFF FFFF (up to 512MB). The external SPI SRAM SPIXR interface is mapped to byte address 0x8000 000 to 0x9FFF FFFF (up to 512MB).

### 2.2.5 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 MAX32650–MAX32652 does not implement this memory area.

### 2.2.6 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 Smart DMA or the SD/SDIO/SDHC/MMC 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.

### 2.2.7 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 MAX32650–MAX32652 does not implement this memory region.

## 2.3 Device Memory Instances

This section details physical memory instances on the MAX32650–MAX32652 (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.

### 2.3.1 Main Program Flash Memory

The main program flash memory is 3MB in size and consists of 192 logical pages of 16,384 Bytes per page.

### 2.3.2 Cache Memories

#### *2.3.2.1 Instruction Cache Controller 0 (ICC0)*

The internal flash memory instruction cache is 16,384 Bytes in size and is used to cache instructions fetched using the I-Code bus, including instructions fetched from the internal flash memory. This instruction cache controller is referred to as ICC0 throughout this document.

#### *2.3.2.2 Instruction Cache Controller 1 (ICC1)*

The SPIXF instruction cache, managed by ICC1, is also 16,384 Bytes and is used to cache instructions fetched from an external SPI memory device. ICC1 is only available if the SPIXF controller is enabled.

*Note: The instruction caches, ICC0 and ICC1, are used for instruction fetches only. Data fetches (including code literal values) from the internal flash memory or external SPIXF memory do not use the instruction cache.*

### 2.3.3 External Memory Cache Controller (EMCC)

Both the HyperBus/Xccela Bus interface and the SPIXR RAM interface are supported by a dedicated 16,384 Byte 2-way set-associative Least Recently Used (LRU) write-through cache. This cache is managed through the EMCC interface.

### 2.3.4 Information Block Flash Memory

The information block is a separate flash instance of 16KB. It is used to store trim settings (option configuration and analog trim) as well as other nonvolatile device-specific information intended for use by firmware.

### 2.3.5 System SRAM

The system SRAM is 1MB in size and can be used for general purpose data storage, the Arm system stack, USB data transfers (endpoints), Smart DMA, Color LCD-TFT Controller, SD Host Controller (SDHC) interface, TPU and code execution if desired.

### 2.3.6 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.

### 2.3.7 MAA Key and Working Space Memory

The MAA contains a dedicated memory for key storage, input and output parameters for operations, and working space. It is mapped into the AHB memory space for ease of loading and unloading.

### 2.3.8 TPU Memory

The MAX32650–MAX32652 contains a specialized 128-bit memory that is designed to preserve critical data (such as a 128-bit AES key) even when the device is in the lowest power-saving state. As long as the RTC power supply is still available, the contents of this memory will be retained, even if the AES block and the main SRAM are shut down completely.

The Secure Key Storage Area consists of four $V_{RTC}$ supply backed 32-bit registers: TPU_TSR_SKS0, TPU_TSR_SKS1, TPU_TSR_SKS2, and TPU_TSR_SKS3.

## 2.4 AHB Interfaces

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

### 2.4.1 Core AHB Interfaces

#### *2.4.1.1 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 and the external SPIF flash memory (if SPIXF is enabled). 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 or the external SPIXF flash memory when a cache miss occurs.

#### *2.4.1.2 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, the external SPIXF flash memory (if SPIXF is enabled), and the information block (if it has not been locked).

#### *2.4.1.3 System*

This AHB master is used by the Arm core for all instruction fetches and data read and write operations involving the SRAM and the HyperBus/Xccela Bus 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.

### 2.4.2 AHB Masters

#### *2.4.2.1 Smart DMA*

The Smart 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.

#### 2.4.2.2 USB Endpoint Buffer Manager

The USB AHB bus master is used to manage endpoint buffers in the SRAM. It has access to the SRAM (read/write, for storage and retrieval of endpoint buffer data), as well as the internal and/or external flash data contents (which can be used to contain static data for transmission by the USB).

#### 2.4.2.3 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.

#### 2.4.2.4 SDHC

The SDHC 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.

#### 2.4.2.5 Color Liquid Crystal Display (CLCD) Controller

The CLCD Controller 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.

#### 2.4.2.6 Trust Protection Unit (TPU)

The TPU 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.

## 2.5 Peripheral Register Map

### 2.5.1 APB Peripheral Base Address Map

*Table 2-1*, *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 2-1: APB Peripheral Base Address Map*

| Peripheral Register Name | Peripheral 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 |
| Trust Protection Unit | TPU_ | 0x4000 1000 | 0x4000 1FFF |
| Watchdog Timer 0 | WDT0_ | 0x4000 3000 | 0x4000 33FF |
| Watchdog Timer 1 | WDT1_ | 0x4000 3400 | 0x4000 37FF |
| Security Monitor | SMON | 0x4000 4000 | 0x4000 43FF |
| AES Keys | AES_ | 0x4000 5000 | 0x4000 53FF |
| Real-Time Clock | RTC_ | 0x4000 6000 | 0x4000 63FF |
| Power Sequencer | PWRSEQ_ | 0x4000 6800 | 0x4000 6BFF |
| GPIO Port 0 | GPIO0_ | 0x4000 8000 | 0x4000 8FFF |
| GPIO Port 1 | GPIO1_ | 0x4000 9000 | 0x4000 9FFF |

| Peripheral Register Name | Peripheral Register Prefix | APB Base Address | APB End Address |
|---|---|---|---|
| GPIO Port 2 | GPIO2_ | 0x4000 A000 | 0x4000 AFFF |
| GPIO Port 3 | GPIO3_ | 0x4000 B000 | 0x4000 BFFF |
| 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 |
| Timer 4 | TMR4_ | 0x4001 4000 | 0x4001 4FFF |
| Timer 5 | TMR5_ | 0x4001 5000 | 0x4001 5FFF |
| SPIMSS ($I^2S$) | SPIMSS_ | 0x4001 8000 | 0x4001 8FFF |
| $I^2C$ 0 | I2C0_ | 0x4001 D000 | 0x4001 DFFF |
| $I^2C$ 1 | I2C1_ | 0x4001 E000 | 0x4001 EFFF |
| SPIXF Master | SPIXF_ | 0x4002 6000 | 0x4002 6FFF |
| SPIXF Master Controller | SPIXFC_ | 0x4002 7000 | 0x4002 7FFF |
| Standard DMA | DMA_ | 0x4002 8000 | 0x4002 8FFF |
| Flash Controller | FLC_ | 0x4002 9000 | 0x4002 93FF |
| Instruction-Cache Controller 0 | ICC0_ | 0x4002 A000 | 0x4002 AFFF |
| Instruction Cache Controller 1 | ICC1_ | 0x4002 F000 | 0x4002 FFFF |
| Color LCD-TFT Controller | CLCD_ | 0x4003 1000 | 0x4003 1FFF |
| External Memory Cache Controller | EMCC_ | 0x4003 3000 | 0x4003 3FFF |
| Analog to Digital Converter | ADC_ | 0x4003 4000 | 0x4003 4FFF |
| SD Host Controller | SDHC_ | 0x4003 7000 | 0x4003 7FFF |
| HyperBus/Xccela Bus Controller | HBMC_ | 0x4003 9000 | 0x4003 9FFF |
| SPIXR Master Controller | SPIXR_ | 0x4003 A000 | 0x4003 AFFF |
| Pulse Train Engine | PT_ | 0x4003 C000 | 0x4003 CFFF |
| 1-Wire | OW_ | 0x4003 D000 | 0x4003 DFFF |
| Semaphores | 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 |
| SPI0 | SPI0_ | 0x4004 6000 | 0x4004 6FFF |
| SPI1 | SPI1_ | 0x4004 7000 | 0x4004 7FFF |
| SPI2 | SPI2_ | 0x4004 8000 | 0x4004 8FFF |

### 2.5.2 AHB Peripheral Base Address Map

*Table 2-1*, *above*, contains the base address for each of the AHB 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 AHB peripheral base address plus the registers offset.

*Table 2-2: AHB Peripheral Base Address Map*

| AHB Peripheral Register Name | Peripheral Register Prefix | AHB Base Address | AHB End Address |
|---|---|---|---|
| USB Hi-Speed Host | USBHS_ | 0x400B 1000 | 0x400B 1FFF |
| True Random Number Generator | TRNG_ | 0x400B 5000 | 0x400B 5FFF |
| SPIXF Master Controller FIFO | SPIXF_FIFO | 0x400B C000 | 0x400B CFFF |
| SPI3 | SPI3_ | 0x400B E000 | 0x400B E3FF |

# 3 System Clocks, Reset, and Power Management

There are several clocks used by different peripherals and subsystems on the MAX32650–MAX32652. These clocks are highly configurable by firmware, allowing developers to select the combination of application performance and power savings required for the target systems.

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

- 120MHz Internal High-Frequency Oscillator
- 40MHz Low-Power Internal Oscillator
- 7.3728MHz Internal Oscillator
  - Selectable for UART baud rate generation
- 8kHz Internal Ultra-Low Power Nano-Ring Oscillator
- 32.768kHz External Crystal Oscillator
  - Clock source for the Real-Time Clock (RTC)

The selected SYSOSC is the input to the system oscillator prescaler to generate the System Clock (SYSCLK). The system oscillator prescaler divides SYSOSC by a prescaler using the *GCR_CLK_CTRL.sysclk_prescale* field as shown in *Equation 3-1*.

*Equation 3-1: System Clock Scaling*

$$f_{SYSCLK} = \frac{SYSOSC}{2^{sysclk\_prescale}}$$

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

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

- Advanced High-Performance Bus (AHB) Clock,
  - $HCLK = SYSCLK$
- Advanced Peripheral Bus (APB) Clock,
  - $PCLK = \frac{SYSCLK}{2}$
- Always On Domain (AOD) Clock,
  - $AODCLK = \frac{PCLK}{2^{GCR\_CLK\_CTRL.aondiv}}$
  - *GCR_CLK_CTRL.aondiv* is selectable from 0 to 3 for divisors of 1, 2, 4 or 8

There are additional internal clocks that are generated. These clocks are independent of SYSOSC and SYSCLK as follows:

- The USB PHY uses the 120MHz oscillator
- The SDHC/SDIO controller uses the high speed 120MHz oscillator divided by 2, independent of PCLK
- The RTC uses the 32.768kHz oscillator
- (MAX32651 only) The Trust Protection Unit (TPU) uses the 40MHz Low-Power Internal Oscillator

All oscillators are reset to default at Power-On Reset (POR) and System Reset. Oscillator status is not reset by a Soft Reset or Peripheral Reset.

## 3.1 Oscillator Sources and Clock Switching

Before using any oscillator, the desired oscillator must first be enabled by setting the oscillator's enable bit in the GCR_CLK_CTRL register. Once an oscillator's enable bit is set, the oscillator's ready bit must read 1 prior to attempting to use the oscillator as a system oscillator source. The oscillator ready status flags are contained in the GCR_CLK_CTRL register.

Once the corresponding oscillator ready bit is set, the oscillator can then be selected as SYSOSC by configuring the Clock Source Select field (GCR_CLK_CTRL.*sysosc_sel*).

Any time firmware changes SYSOSC by changing GCR_CLK_CTRL.*sysosc_sel*, the Clock Ready bit GCR_CLK_CTRL.*sysosc_rdy* is automatically cleared to indicate that a SYSOSC switchover is in progress. When switchover is complete, GCR_CLK_CTRL.*sysosc_rdy* is set to 1 by hardware indicating the oscillator selected is ready for use.

Immediately before entering any low-power mode, enable the SYSOSC to be used in that low-power mode.

### 3.1.1 *120MHz Internal Main High-Speed Oscillator*

The MAX32650–MAX32652 is available with a 120MHz internal high-speed oscillator. This is the fastest oscillator and draws the most power. This oscillator is automatically enabled after a Power-On Reset (POR) and following a System Reset.

This oscillator is also used by the USB PHY and the SDHC. If the USB or SDHC is enabled, the 120MHz oscillator must be enabled, independent of the selection of SYSOSC.

Optionally, this oscillator can be powered down automatically when in DEEPSLEEP mode by setting register bit GCR_PMR.*hircmmpd*.

The high-speed oscillator is disabled when the device is in BACKUP mode

### 3.1.2 *40MHz Low Power Internal Oscillator*

This is a low-power internal oscillator that can be selected as SYSOSC. This oscillator is also the dedicated clock for the TPU. If the TPU is enabled, the 40MHz internal oscillator must be enabled, independent of the selection of SYSOSC. When used as the TPU clock it can be divided by 2.

This oscillator can optionally be automatically powered down when in DEEPSLEEP mode by setting register bit GCR_PMR.*hirc40mpd*.

This oscillator is enabled by default at power-up and is set as the SYSOSC.

### 3.1.3 *7.3728MHz Internal Oscillator*

This is a very low power internal oscillator that can be selected as SYSOSC.

This clock can optionally be used as a dedicated baud rate clock for the three UARTs. This is useful if the SYSOSC selected does not allow the targeted UART baud rate.

Firmware selection of the voltage that controls this oscillator is controlled by the register bit GCR_CLK_CTRL.*hirc7m_vs*. The internal CPU core supply voltage ($V_{CORE}$) is the default option. The external pin $V_{DDA}$ can also be selected. The $V_{DDA}$ pin goes to an internal 1V regulator that also provides the analog supply voltage for this device.

$V_{CORE}$ is gated off in BACKGROUND, DEEPSLEEP, and BACKUP modes and is unavailable to this oscillator in those modes. In DEEPSLEEP the voltage for this oscillator is automatically switched to $V_{DDA}$. On exiting DEEPSLEEP the voltage is automatically switched back to the setting in bit GCR_CLK_CTRL.*hirc7m_vs*.

This oscillator can optionally be automatically powered down when in DEEPSLEEP mode by setting register bit GCR_PMR.*hirc7mpd*.

This oscillator is disabled by default at power-up.

### 3.1.4 *32.768kHz External Crystal Oscillator*

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

This oscillator is the dedicated clock for the Real-Time Clock (RTC). If the RTC is enabled, the 32.768kHz external oscillator must be enabled, independent of the selection of SYSOSC. This oscillator is disabled at power-up.

When this oscillator is active, an RTC alarm can wake this device from SLEEP or DEEPSLEEP mode.

### 3.1.5 *8kHz Ultra-Low Power Nano-Ring Internal Oscillator*

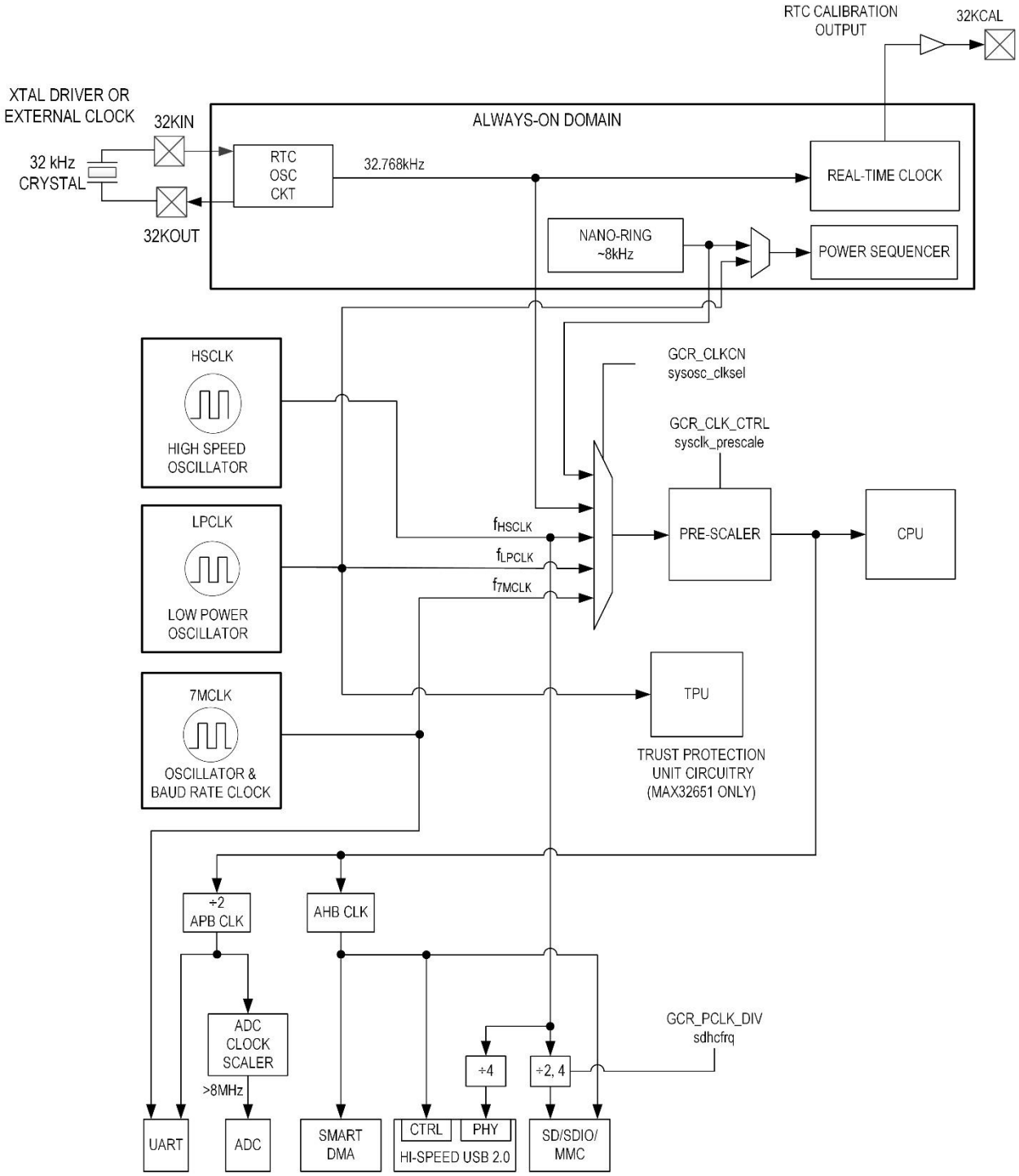
This is an ultra-low power internal oscillator that can be selected as SYSOSC.

This oscillator is enabled at power-up and cannot be disabled by firmware.

## 3.2 System Oscillators Reset

On Power-On Reset (POR) and System Reset, all oscillator states are reset to the default: The 40MHz, 120MHz, and 8kHz oscillators are enabled, while the 32.768kHz and 7.3728MHz oscillators are disabled. Oscillators are not reset on Soft Reset or Peripheral Reset.

*Figure 3-1: Clock Block Diagram*



## 3.3 Power Management

When applying power to the MAX32650–MAX32652, $V_{RTC}$ should be powered above the $V_{RTC}$ power-on reset level prior to supplying power to any other power pins. $V_{CORE}$ should be applied last except for $V_{USB}$. $V_{USB}$ may be applied at any time after the device is powered on. Once $V_{RTC}$ reaches its operating voltage, $V_{DDIO}$, $V_{DDIOH}$, and $V_{DDA}$ should be powered on followed by $V_{CORE}$.

## 3.4 Operating Modes

The MAX32650–MAX32652 supports five operating modes. ACTIVE is the highest performance operating mode. Any low power state can wake up to ACTIVE by a wakeup event. Wakeup events include any external or internal interrupt, USB wakeup, RTC wakeup, and Watchdog Interrupt.

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

SCR.*sleepdeep* = 0; // SLEEP mode enabled
SCR.*sleepdeep* = 1; // DEEPSLEEP mode enabled

Once enabled, the device enters the enabled low power mode when either a WFI (Wait For Interrupt) or WFE (Wait For Event) instruction is executed.

Immediately before entering any low-power mode, enable the SYSOSC to be used in that low-power mode. If DEEPSLEEP or BACKUP is to be entered, ensure that the selected SYSOSC is not automatically disabled in that low power mode. If the selected SYSOSC is disabled in that low power mode, it will be enabled upon returning to ACTIVE mode.

Refer to the Arm Cortex-M4 core reference for more information on SCR.

### 3.4.1 *ACTIVE Mode*

This is the highest performance mode. All internal clocks, registers, memory, and peripherals are enabled. The CPU is running and executing application code. The Smart DMA can perform background processing and data transfers. All oscillators are available.

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

### 3.4.2 *SLEEP Low Power Mode*

This is a low power mode that suspends the CPU with a fast wakeup time to ACTIVE mode. It is like ACTIVE mode except the CPU clock is disabled, which temporarily prevents the CPU from executing code. The Smart DMA can operate in the background to perform background processing and data transfers. All oscillators remain active if enabled and the Always On Domain (AOD) and RAM retention is retain state.

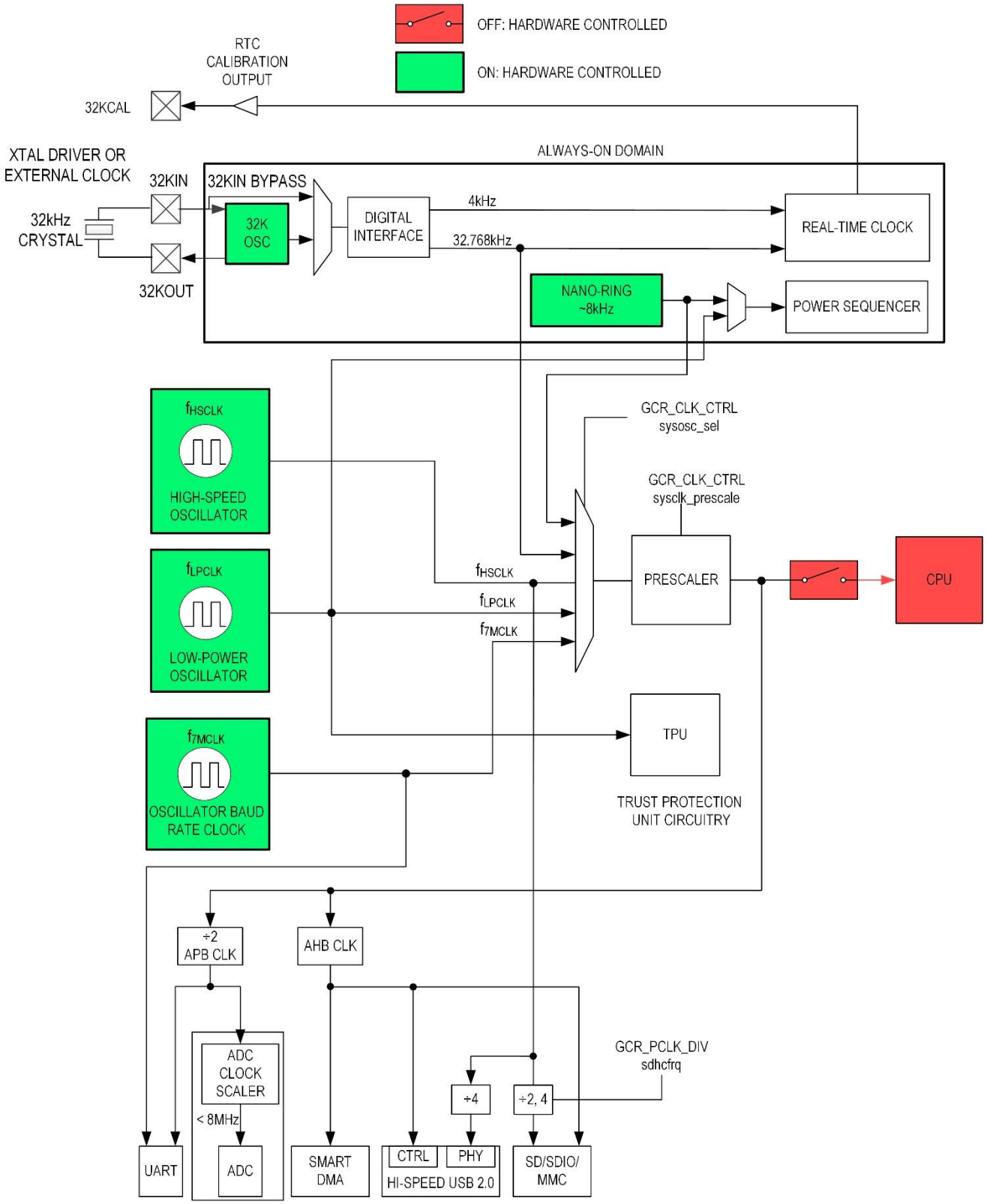
The device returns to ACTIVE mode from any internal or external interrupt.

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
```

*Figure 3-2*, *below*, shows the clocks available and blocks disabled during SLEEP mode.

*Figure 3-2: SLEEP Mode Clock Control*



### 3.4.3 BACKGROUND Low Power Mode

This mode is suitable for the Smart DMA to operate in the background and perform background processing data transfers on peripheral and SRAM data.

This is the same as SLEEP mode except both the CPU clock and CPU power (VCORE) are temporarily gated off. State Retention of the CPU is enabled, allowing all CPU registers to maintain their contents and the oscillators remain active if enabled.

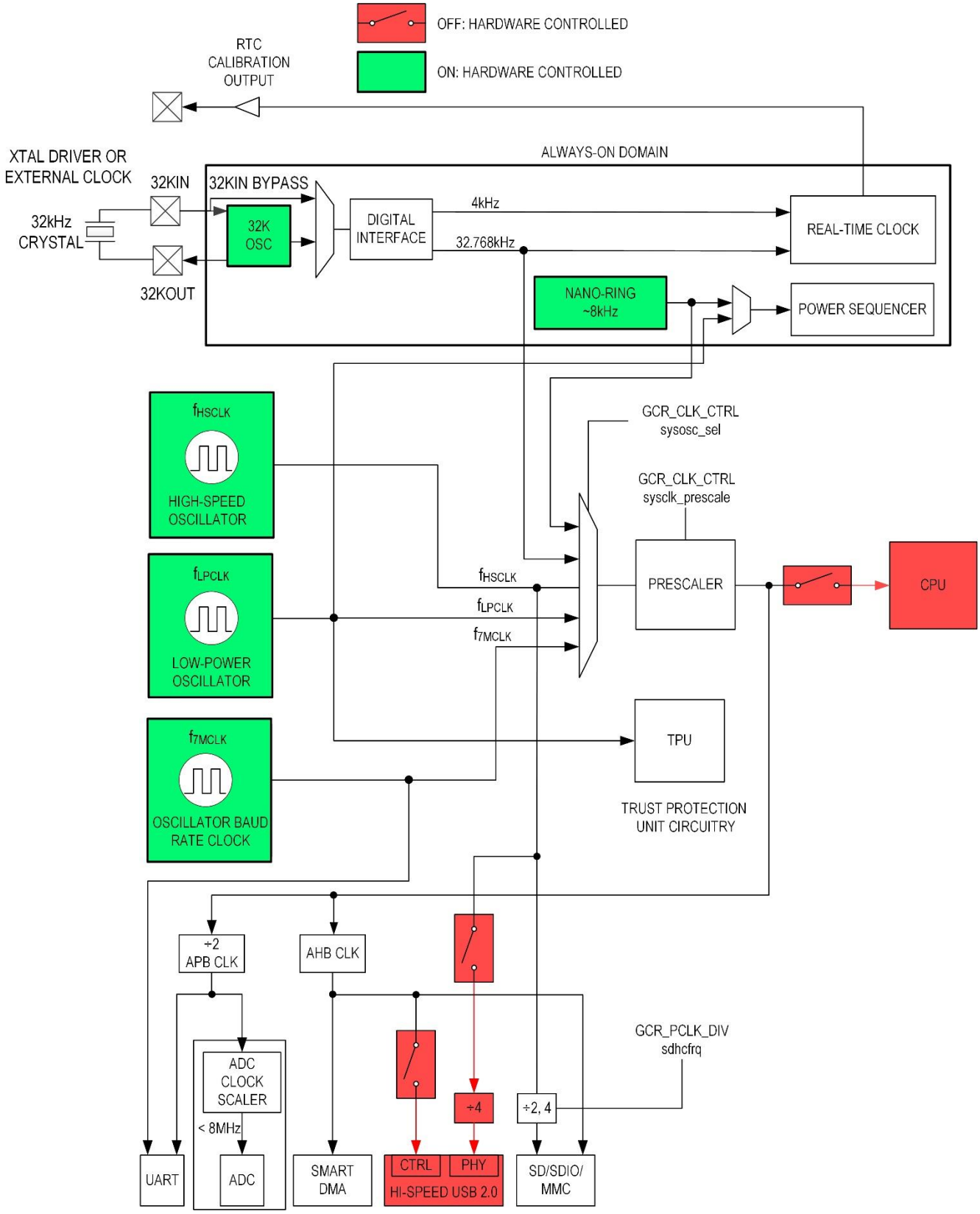
Because both the clock and power to the CPU is disabled, this has the advantage of drawing less power than SLEEP. However, the CPU takes longer to wakeup compared to SLEEP.

BACKGROUND mode is an enhancement to the built-in Arm Cortex DEEPSLEEP mode. To enter BACKGROUND mode when entering DEEPSLEEP, first enable the mode by setting the *LP_CTRL.bkgrnd* bit in the Low Power Control Register, *LP_CTRL*, as shown in the following pseudocode.

```
LP_CTRL.bkgrnd = 1;   // BACKGROUND mode enabled when entering DEEPSLEEP
SCR.sleepdeep = 1; // DEEPSLEEP mode enabled
WFI (or WFE);      // Enter BACKGROUND mode
```

*Figure 3-3*, *below*, shows the clock control during BACKGROUND mode.

*Figure 3-3: BACKGROUND Mode Clock Control*



### 3.4.4 *DEEPSLEEP Low Power Mode*

This is like BACKGROUND mode except that all internal clocks are gated off. SYSOSC is gated off, so the two main bus clocks PCLK and HCLK are inactive. The CPU state is retained.

Because the main bus clocks are disabled, all peripherals are inactive except for the RTC which has its own independent oscillator. Only the RTC, USB wakeup or external interrupt can return the device to ACTIVE. The Smart DMA and Watchdog Timers are inactive in this mode.

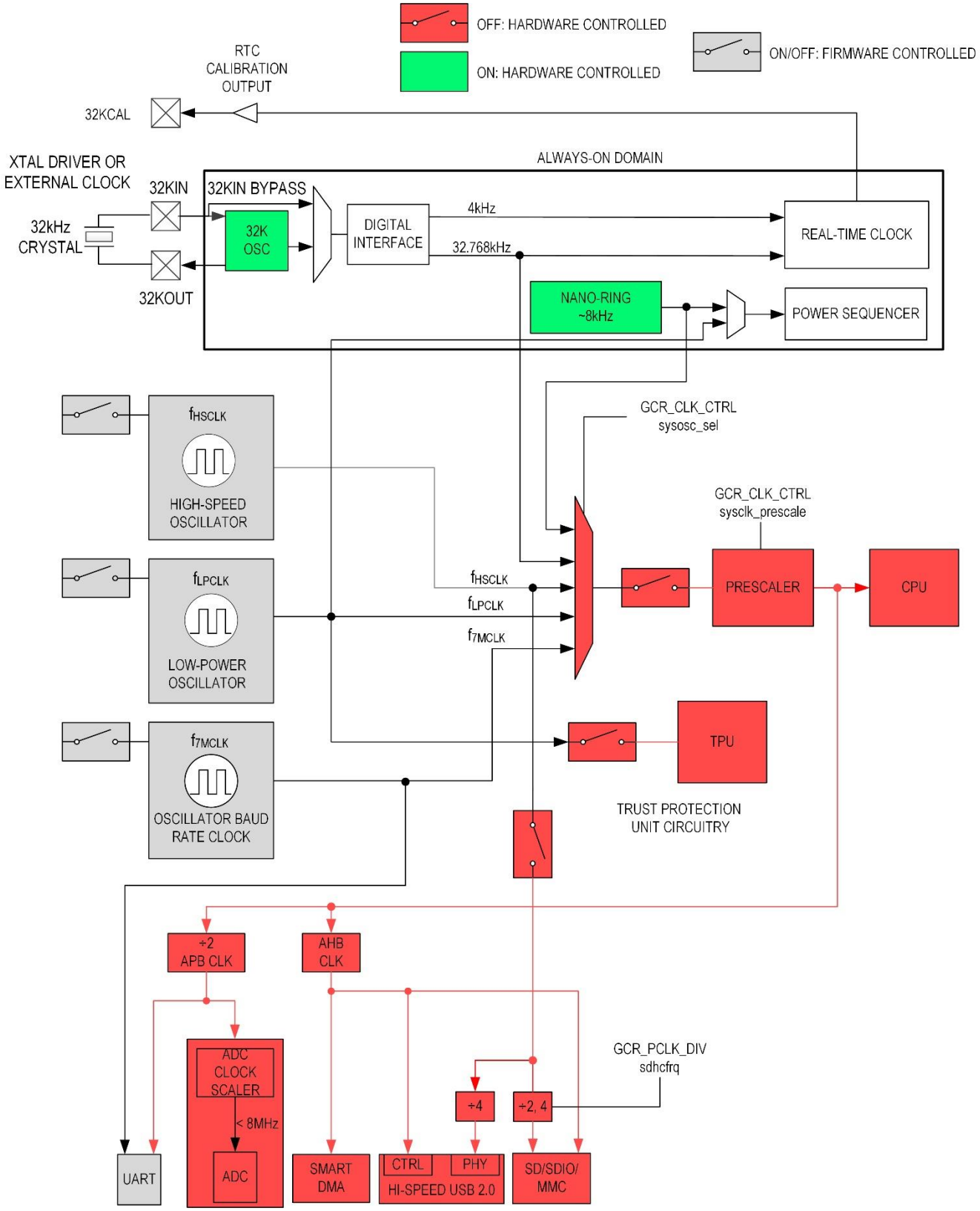
All internal register contents and all RAM contents are preserved. The GPIO pins retain their state in this mode. The Always-on Domain (AoD) and RAM Retention are available.

Three oscillators can be set to optionally automatically disable themselves when the device enters DEEPSLEEP mode: the 7.3728MHz oscillator, the 40MHz oscillator, and the 120MHz oscillator. The 8Khz and 32.768kHz oscillators are available.

BACKGROUND mode must be disabled before entering DEEPSLEEP. To enter DEEPSLEEP mode, first disable BACKGROUND mode by setting *LP_CTRL.bkgrnd*=0.

```
LP_CTRL.bkgrnd = 0;   // BACKGROUND mode disabled when entering DEEPSLEEP
SCR.sleepdeep = 1; // DEEPSLEEP mode enabled
WFI (or WFE);      // Enter DEEPSLEEP mode
```

*Figure 3-4: DEEPSLEEP Clock Control*



### 3.4.5 *BACKUP Low Power Mode*

This is the lowest power operating mode. All oscillators are disabled except for the 8kHz and the 32kHz oscillator. SYSOSC is gated off, so PCLK and HCLK are inactive. The CPU state is not maintained.

Only the RTC can operate in BACKUP mode. The AoD and RAM Retention can optionally be set to automatically disable (and clear) themselves when entering this mode. Data retention in this mode is maintained using VCORE and/or VRTC. The type of data retained is dependent upon whether only one, or both, of these voltages are enabled.

Optionally, VCORE can be gated off and the internal retention regulator enabled, allowing the device to be powered only by VRTC. Enabling VCORE will wake the device to ACTIVE mode.

The amount of RAM memory retained is dependent upon which voltages are enabled.

If only VRTC is enabled, up to 96KBytes of SRAM can be retained.

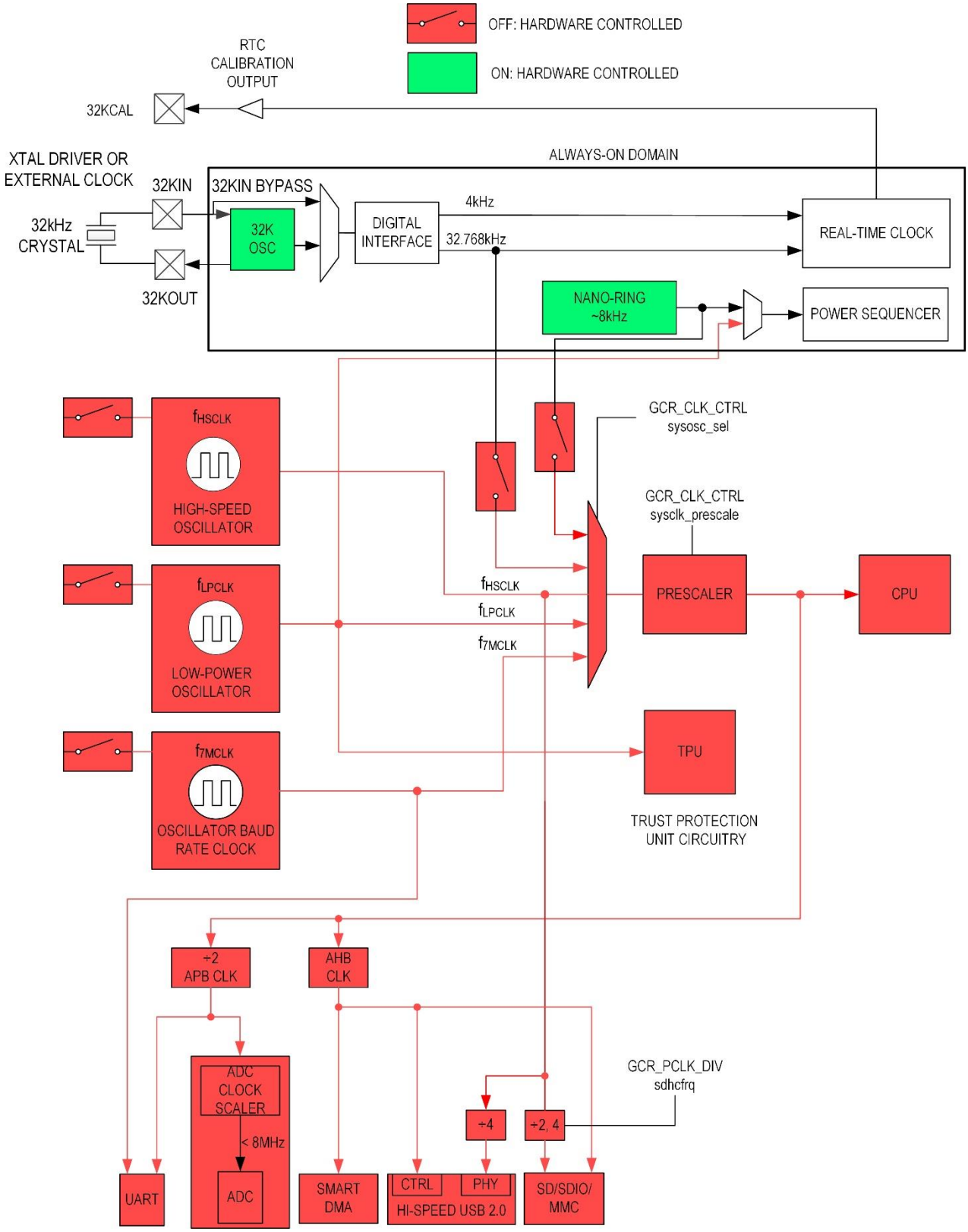
If both VRTC and VCORE are enabled, up to 1024KBytes SRAM can be retained.

BACKUP mode supports the same wakeup sources as DEEPSLEEP mode.

To immediately enter BACKUP mode, write *GCR_PMR.mode* = 0b100.

*Figure 3-5*, *below*, shows the clock control during BACKUP mode.

*Figure 3-5: BACKUP Mode Clock Control*



## 3.5 Shutdown State

The Shutdown State is not a low-power mode. It is intended to wipe all volatile memory from the device. In the Shutdown state, internal logic gates off all internal power. There is no data, register, or RAM retention in this mode. All wakeup sources, wakeup logic, and interrupts are disabled. The Always-on Domain (AoD) is disabled, clearing all AES keys. The device only recovers through a Power-On Reset (POR) which re-initializes the device.

In security-related applications it might be necessary to completely disable the device if a breach or other security threat is detected. In this situation, clearing all memory including the AoD and RTC might be required. Shutdown state is also useful in a test environment where the device must be completely powered off to save power after testing is completed.

To immediately put the device into Shutdown, write *GCR_PMR.mode* = 0b111.

## 3.6 Device Resets

Four device resets are available – Peripheral Reset, Soft Reset, System Reset, and Power-On Reset. On completion of any of the four reset cycles, all peripherals, including the Smart DMA, 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 $V_{RTC}$.

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

### 3.6.1 *Peripheral Reset*

This resets the all peripherals. The CPU retains its state. The GPIO, Watchdog Timers, AoD, RAM Retention, and General Control Registers (GCR), including the clock configuration, are unaffected.

To start a Peripheral Reset, set *GCR_RST0.periph_rst* = 1.

### 3.6.2 *Soft Reset*

This is the same as a Peripheral Reset except that it also resets the GPIO to its Power-On Reset state. All alternate functions are tri-stated.

To start a Soft Reset, set *GCR_RST0.soft_rst* = 1.

### 3.6.3 *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 Retention are unaffected.

A watchdog timer reset event initiates a System Reset. To start a System Reset from firmware, set *GCR_RST0.sys_rst* = 1.

### 3.6.4 *Power-On Reset*

A POR resets everything in the device to its default state, as if power had been cycled to the device. RAM are cleared except for RAM in the AoD. *Table 3-1* shows the effects of the four reset types and the five power modes supported by the MAX32650–MAX32652.

*Table 3-1: Reset and Low Power Mode Effects*

| | Peripheral Reset | Soft Reset | System Reset | POR | ACTIVE Mode | SLEEP Mode | BACK-GROUND Mode | DEEP- SLEEP Mode | BACKUP Mode |
|---|---|---|---|---|---|---|---|---|---|
| **GCR Reset** | No | No | Reset | Reset | N/A | N/A | N/A | N/A | N/A |
| **8kHz Osc** | On | On | On | On | On | On | On | On | On |
| **32kHz Osc** | - | - | Off | Off | Y | Y | Y | Y | Y |
| **7.3728 MHz Osc** | - | - | Off | Off | Y | Y | Y | Auto Off | Off |
| **40MHz Osc** | - | - | On | On | Y | Y | Y | Auto Off | Off |
| **120MHz Osc** | - | - | On | On | Y | Y | Y | Auto Off | Off |
| **PCLK** | On | On | On | On | On | On | On | Off | Off |
| **HCLK** | On | On | On | On | On | On | On | Off | Off |
| **CPU Clock** | On | On | On | On | On | Off | Off | Off | Off |
| **VCORE** | On | On | On | On | On | On | On | Off | Off |
| **CPU State Retention** | On | On | Reset | Reset | N/A | On | On | On | Off |
| **RTC** | Reset | Reset | Reset | Reset | Y | Y | Y | Y | Y |
| **Standard DMA** | Reset | Reset | Reset | Reset | Y | Y | Off | Off | Off |
| **Smart DMA** | Reset | Reset | Reset | Reset | Y | Y | Y | Off | Off |
| **WDT1 & WDT2** | - | - | Reset | Reset | Y | Y | Y | Off | Off |
| **GPIO** | - | Reset | Reset | Reset | Y | Y | Y | Y | Y |
| **USB<br>HyperBus<br>SPIXR<br>Flash<br>All Cache Instances** | Reset | Reset | Reset | Reset | Y | Y | Off | Off | Off |
| **Other Peripherals** | Reset | Reset | Reset | Reset | Y | Y | Y | Off | Off |
| **External Reset Wakeup[1]** | - | - | - | - | - | Y | Y | Y | Y |
| **GPIO Wakeup** | - | - | - | - | - | Y | Y | Y | Y |
| **USB Wakeup** | - | - | - | - | - | Y | Y | Y | Y |

| | Peripheral Reset | Soft Reset | System Reset | POR | ACTIVE Mode | SLEEP Mode | BACK-GROUND Mode | DEEP- SLEEP Mode | BACKUP Mode |
|---|---|---|---|---|---|---|---|---|---|
| **RTC Wakeup** | - | - | - | - | - | Y | Y | Y | Y |
| **AOD[1]** | On | Y | Y | Y | Y | On | On | On | Auto Off |
| **RAM Retention** | Y | Y | Y | Reset | Y | Y | Y | Y | Auto Off |

Table key:
Y = Enabled, can be disabled by firmware
On = Enabled by hardware (Cannot be disabled)
Off = Disabled by hardware (Cannot be enabled)
Auto Off = Can either be left on, or automatically gated off when in this power mode.
- = No Effect
N/A = Not Applicable

*Note: SLEEP, BACKGROUND, DEEPSLEEP, and BACKUP low-power modes wake-up directly to ACTIVE with no reset.*

*Note: The Always on Domain (AoD) includes the oscillator trim settings, AES Keys, RTC, RAM retention & sleep registers, and Low Power Wakeup Control Registers and is only reset on power-cycling $V_{RTC}$.*

*Note: RAM Retention applies to data SRAM, all caches, and all FIFOs.*

*Note: Peripheral, Soft, and System Resets are initiated by firmware though the GCR_RST0 register.*

*Note: A Watchdog Reset initiates a System Reset.*

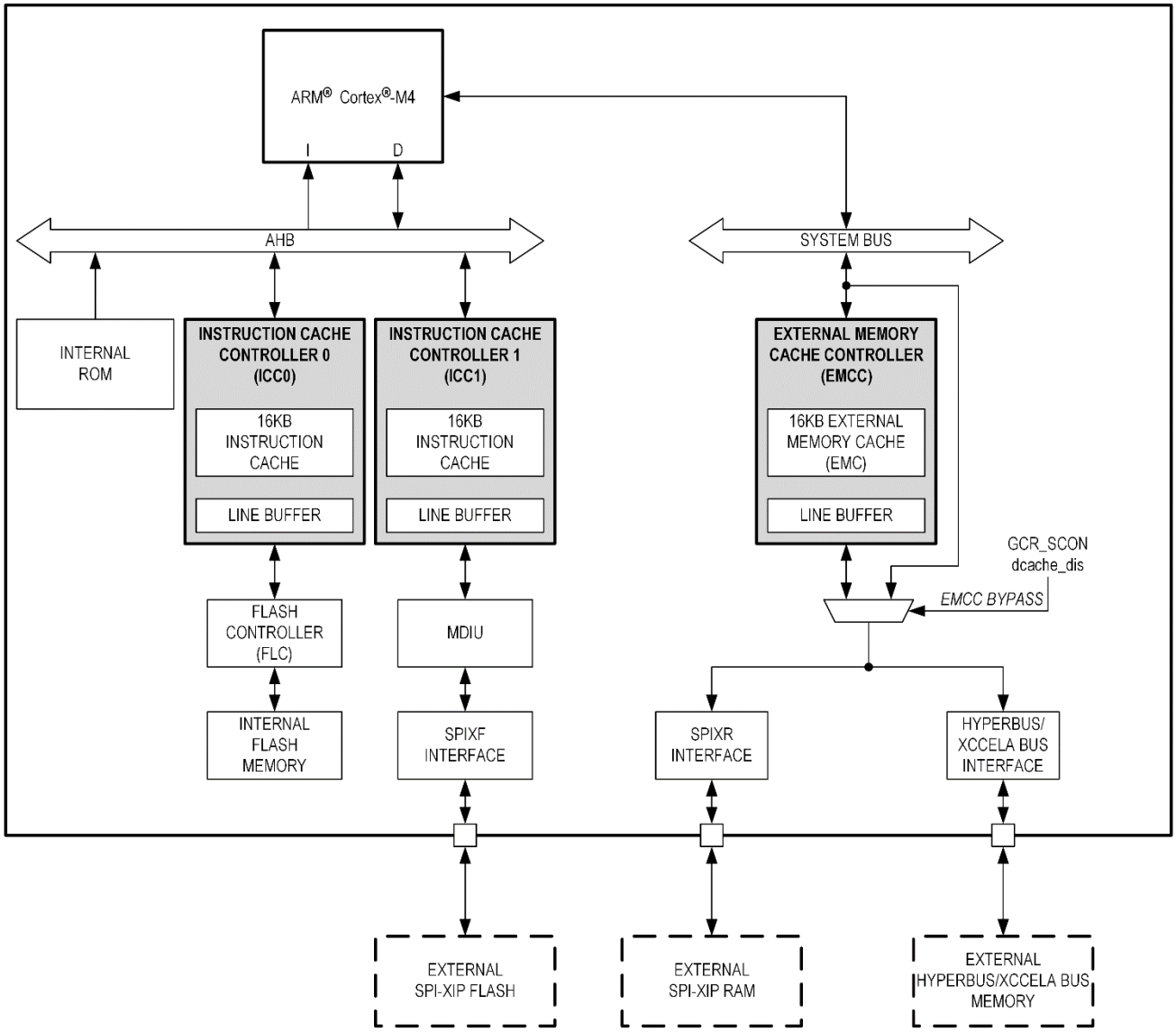
*[1] The Always On Domain (AOD) is only reset on power-cycling $V_{RTC}$*

## 3.7 Cache

There are three cache controllers in the MAX32650–MAX32652. Each cache controller is independently managed. *Figure 3-6* shows the three cache controllers and their memory interfaces. Instruction Cache Controller 0 (ICC0) and Instruction Cache Controller 1 (ICC1) are used for instruction caching only. ICC0 interfaces to the internal 3MB Flash and ICC1 interfaces to an external SPI-XiP Flash for external code execution. The External Memory Cache Controller (EMCC) is used for Data and Instruction caching for HyperBus or Xccella Bus memories. The EMCC is implemented as a write through cache.

All three caches are managed separately using their specific cache controller, ICC0, ICC1 or EMCC. Each controller be enabled, disabled, and invalidated. Each cache clock can be disabled by placing it in Light Sleep.

*Figure 3-6: MAX32650–MAX32652 Cache Controllers Diagram*




## 3.8 Instruction Cache Controller

ICC0 and ICC1 are independent cache controllers and each is controlled directly using their respective register set.

### 3.8.1 *Enabling ICC0/ICC1*

Perform the following steps to enable ICC0 or ICC1.

1. Set *ICCn_CACHE_CTRL.enable* to 1.
2. Read *ICCn_CACHE_CTRL.ready* until it returns 1.

### 3.8.2 *Disabling ICC0/ICC1*

Disable either ICC0 or ICC1 by setting the *ICCn_CACHE_CTRL.enable* to 0.

### 3.8.3 *Flushing the ICC0/ICC1 Cache*

The System Configuration Register (*GCR_SCON*) includes a field for flushing both ICC0 and ICC1 simultaneously. Setting *GCR_SCON.ccache_flush* to 1 performs a flush of both ICC0 and ICC1 cache. Flush only one of the ICC caches by invalidating the cache contents. Set the ICCn_INVALIDATE register to 1 invalidates the respective cache and forces a cache flush. Read the *ICCn_CACHE_CTRL*.ready field until it returns 1 to determine when the flush is completed.

## 3.9 Instruction Cache Controller Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the ICC0 and ICC1 Base Peripheral Address.

*Table 3-2: Instruction Cache Controller Register Addresses and Descriptions*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *ICCn_CACHE_ID* | RO | Cache ID Register |
| [0x0004] | *ICCn_MEM_SIZE* | R0 | Cache Memory Size Register |
| [0x0100] | *ICCn_CACHE_CTRL* | R/W | Clock Control Register |
| [0x0700] | *ICCn_INVALIDATE* | R/W | Power Management Register |

## 3.10 Instruction Cache Controller Register Details

*Table 3-3: ICC Cache ID Register*

| ICC Cache ID Register | | | ICCn_CACHE_ID | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 15:10 | cchid | RO | - | **Cache ID**<br>Returns the Cache ID for this Cache instance. |
| 9:6 | partnum | RO | - | **Cache Part Number**<br>Returns the part number indicator for this Cache instance. |
| 5:0 | relnum | RO | - | **Cache Release Number**<br>Returns the release number for this Cache instance. |

*Table 3-4: ICC Memory Size Register*

| ICC Memory Size Register | | | ICCn_MEM_SIZE | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | memsz | RO | - | **Addressable Memory Size**<br>Indicates the size of addressable memory by this cache controller instance in 128KB units. |
| 15:0 | cchsz | RO | - | **Cache Size**<br>Returns the size of the cache RAM memory in 1KB units.<br>16: 16KB Cache RAM |

*Table 3-5: ICC Cache Control Register*

| ICC Cache Control Register | | | ICCn_CACHE_CTRL | [0x0100] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | - | **Reserved for Future Use**<br>Do not modify this field. |
| 16 | ready | RO | - | **Ready**<br>This field is cleared by hardware anytime the cache as a whole is invalidated (including a Power On Reset event). Hardware automatically sets this field to 1 when the invalidate operation is complete and the cache is ready.<br>0: Cache Invalidate in process.<br>1: Cache is ready.<br>*Note: While this field reads 0, the cache is bypassed and reads come directly from the line fill buffer.* |
| 15:1 | - | R/W | - | **Reserved for Future Use**<br>Do not modify this field. |
| 0 | enable | R/W | 0 | **Enable**<br>Set this field to 1 to enable the cache. Setting this field to 0 automatically invalidates the cache contents. When this cache is disabled, reads are handled by the line fill buffer.<br>0: Disable Cache<br>1: Enable Cache |

*Table 3-6: ICC Invalidate Register*

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

## 3.11 External Memory Cache Controller

See Section *7.4: External Memory Cache Controller (EMCC)* for detailed usage information for the EMCC and the EMCC register interface.

## 3.12 RAM Memory Management

This device has many features for managing the on-chip RAM. The on-chip RAM includes data RAM, instruction and data caches, and peripheral FIFOs.

### 3.12.1 *RAM Zeroization*

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

The following RAM memories can be zeroized:

- The Internal Data RAMs 0 through 6.
  - Each of the internal data RAM segments can be zeroized independently by setting the *GCR_MEM_ZERO.sram0z* through *GCR_MEM_ZERO.sram6z* fields to 1.
- The USB FIFO
  - Write 1 to *GCR_MEM_ZERO.usbfifoz*
- ICC0 16KB Cache
  - Write 1 to *GCR_MEM_ZERO.icachez*
- ICC1 16KB Cache
  - Write 1 to *GCR_MEM_ZERO.icachexipz*
- EMCC Cache Tags
  - Write 1 to *GCR_MEM_ZERO.scachetagz*
  - This clears on the Cache tags, ultimately invalidating the EMCC cache memory as well.
- EMCC 16KB Cache
  - Write 1 to *GCR_MEM_ZERO.scachedataz*
- Crypto MAA RAM (MAX32651 only)

### 3.12.2 *RAM Low Power Modes*

#### *3.12.2.1 RAM Light Sleep*

RAM can be placed in a low power mode, referred to as Light Sleep, using the Memory Clock Control Register, *GCR_MEM_CLK*. Light Sleep gates off the clock to the RAM and makes the RAM unavailable for read/write operations, while memory contents are retained, reducing power consumption. Light Sleep is available for the seven Data RAM blocks, the USB FIFO, Crypto RAM, code cache, and the shared SPI-XIPF and HyperBus/Xccela interface cache. RAM contents are available when exiting Light Sleep mode.

#### *3.12.2.2 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 *LP_MEM_PWR* register.

## 3.13 Global Control Registers (GCR)

Refer to *Table 2-1: APB Peripheral Base Address Map* for the General Control Register's Base Peripheral Address.

*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 3-7: Global Control Register Addresses and Descriptions*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *GCR_SCON* | R/W | System Control Register |
| [0x0004] | *GCR_RST0* | R/W | Reset Register 0 |

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0008] | *GCR_CLK_CTRL* | R/W | Clock Control Register |
| [0x000C] | *GCR_PMR* | R/W | Power Management Register |
| [0x0018] | *GCR_PCLK_DIV* | R/W | Peripheral Clocks Divisor |
| [0x0024] | *GCR_PCLK_DIS0* | R/W | Peripheral Clocks Disable 0 |
| [0x0028] | *GCR_MEM_CLK* | R/W | Memory Clock Control |
| [0x002C] | *GCR_MEM_ZERO* | R/W | Memory Zeroize Register |
| [0x0040] | *GCR_SYS_STAT* | RO | System Status Flags |
| [0x0044] | *GCR_RST1* | R/W | Reset Register 1 |
| [0x0048] | *GCR_PCLK_DIS1* | R/W | Peripheral Clocks Disable 1 |
| [0x004C] | *GCR_EVENT_EN* | R/W | Event Enable Register |
| [0x0050] | *GCR_REV* | RO | Revision Register |
| [0x0044] | *GCR_SYS_STAT_IE* | R/W | System Status Interrupt Enable |

## 3.14 Global Control Register Details

*Table 3-8: System Control Register*

| System Control Register | | GCR_SCON | [0x0000] |
|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:18 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 17:16 | ovr | R/W | 0 | **Operating Voltage Range**<br>0b00: 0.9V ±10%<br>0b01: 1.0V ±10%<br>0b10: 1.1V ±10% is this valid?<br>0b11: Reserved, do not use.<br>To allow on-chip volatile memory to operate at the optimal timing range, set this to be the same as VCORE. |
| 15 | chkres | R | 0 | **ROM Checksum Calculation Pass/Fail**<br>This is the result after setting bit *GCR_SCON.cchk*.<br>This bit is only valid after the ROM checksum is complete and cchk is cleared. |
| 14 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 13 | cchk | R/W1 | 0 | **Calculate ROM Checksum**<br>0: No operation<br>1: Start ROM checksum calculation.<br>This bit is self-clearing when the ROM checksum calculation is complete, and the result is available at bit *GCR_SCON.chkres*.<br>Writing a 0 has no effect. |
| 12:10 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |

| System Control Register | | | GCR_SCON | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 9 | dcache_dis | R/W | 0 | **External Memory Cache Controller Disable**<br>This disables the EMCC used for SPIXR or HyperBus/Xccela Bus code and data cache. Setting this field disables the EMC and bypass the EMCC line buffer.<br>0: EMCC enabled<br>1: EMCC disabled and line buffer bypassed |
| 8 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 7 | dcache_flush | R/W1 | 0 | **External Memory Controller Cache (EMCC) Flush**<br>Write 1 to flush the external memory controller's 16KB cache. This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect.<br>0: Reads 0 if the EMCC is not actively being flushed.<br>1: Set to 1 to flush the 16KB EMC.<br>*Note that this cache is shared between the SPIXR and the HyperBus/Xccela Bus interface.* |
| 6 | ccache_flush | R/W1 | 0 | **ICC Code Cache Flush**<br>Write 1 to flush both the ICC0 and ICC1 caches. ICC0 is used for the internal flash memory's instruction cache and ICC1 is used for the external SPIXF's instruction cache. This bit is automatically cleared to 0 when the flush is complete. Writing 0 has no effect.<br>1: Write 1 to flush the ICC0 and ICC1 caches.<br>0: Reads 0 when the flush is complete or not actively being flushed. |
| 5 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 4 | flash_page_flip | R/* | 0 | **Flash Page Flip Flag**<br>Flips the bottom and top halves of Flash memory. This bit is controlled by hardware. Firmware should not change the state of this bit during normal operation. Any change to this bit also flushes both code and data caches.<br>0 = Physical layout matches logical layout<br>1 = Top and Bottom halves flipped |
| 3 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 2:1 | - | R/W | 1 | **Reserved for Future Use**<br>Always Write 1 for proper device operation. |
| 0 | bstapen | R/W | See Desc | **Boundary Scan Tap Enable**<br>0: JTAG port connected to Arm ICE<br>1: JTAG port connected to Boundary Scan Tap<br>*Note: If the Arm ICE is unlocked (GCR_SYS_ST.icelock=0), the reset value for this bit is 1. If the Arm ICE is locked (GCR_SYS_ST.icelock=1), the reset value for this bit is 0.* |

*Table 3-9: Reset Register 0*

| Reset Register 0 | | | GCR_RST0 | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31 | sys_rst | R/W1 | 0 | **System Reset**<br>This resets everything on the device except AoD and RAM retention. GCR are reset.<br>1: Write 1 to perform a System Reset.<br>0: System Reset complete or not active.<br>*Refer to Device Resets section for additional information.* |
| 30 | periph_rst | R/W1 | 0 | **Peripheral Reset**<br>Write 1 to reset all peripherals and both the Standard and Smart DMA.<br>1: Write 1 to perform a Peripheral Reset.<br>0: Peripheral Reset complete or not active.<br>*Note: Watchdog Timers, GPIO Ports, the AoD, RAM Retention and the General Control Registers (GCR) are unaffected.*<br>*Refer to Device Resets section for additional information.* |
| 29 | soft_rst | R/W1 | 0 | **Soft Reset**<br>Write 1 to perform a Peripheral Reset including the GPIO Ports.<br>1: Write 1 to perform a Soft Reset.<br>0: Soft Reset complete or not active.<br>*Refer to Device Resets section for additional information.* |
| 28 | uart2 | R/W1 | 0 | **UART2 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 27 | - | R/W | - | **Reserved for Future Use**<br>Do not modify this field. |
| 26 | adc | R/W1 | 0 | **Analog to Digital Converter Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 25 | - | R/O | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 24 | - | R/W | - | **Reserved for Future Use**<br>Do not modify this field. |
| 23 | usb | R/W1 | 0 | **USB Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 22 | tft | R/W1 | 0 | **TFT Controller Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 21 | hbc | R/W1 | 0 | **HyperBus/Xccela Controller Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 20:19 | - | R/W | - | **Reserved for Future Use**<br>Do not modify this field. |

| Reset Register 0 | | | GCR_RST0 | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 18 | crypto | R/W1 | 0 | **Cryptographic Reset**<br>This resets the AES block, SHA block, and DES block.<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 17 | rtc | R/W1 | 0 | **RTC Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 16 | i2c0 | R/W1 | 0 | **I2C0 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 15 | spi2 | R/W1 | 0 | **SPI2 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 14 | spi1 | R/W1 | 0 | **SPI1 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 13 | spi0 | R/W1 | 0 | **SPI0 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 12 | uart1 | R/W1 | 0 | **UART1 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 11 | uart0 | R/W1 | 0 | **UART0 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 10 | timer5 | R/W1 | 0 | **Timer5 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 9 | timer4 | R/W1 | 0 | **Timer4 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 8 | timer3 | R/W1 | 0 | **Timer3 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |

| Reset Register 0 | | | GCR_RST0 | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7 | timer2 | R/W1 | 0 | **Timer2 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 6 | timer1 | R/W1 | 0 | **Timer1 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 5 | timer0 | R/W1 | 0 | **Timer0 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 4 | gpio2 | R/W1 | 0 | **GPIO2 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 3 | gpio1 | R/W1 | 0 | **GPIO1 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 2 | gpio0 | R/W1 | 0 | **GPIO0 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 1 | wdt0 | R/W1 | 0 | **Watchdog Timer 0 Reset**<br>Write 1 to reset the peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |
| 0 | dma | R/W1 | 0 | **Standard DMA Reset**<br>Write 1 to reset the Standard DMA peripheral.<br>1: Reset peripheral or peripheral reset not yet complete.<br>0: Peripheral reset complete or not actively being reset. |

*Table 3-10: System Clock Control Register*

| System Clock Control Register | | | GCR_CLK_CTRL | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:30 | - | RO | 3 | **Reserved for Future Use**<br>Do not modify this field. |
| 29 | lirc8k_rdy | RO | 0 | **8kHz Internal Oscillator Ready Status**<br>0: Not ready or not enabled.<br>1: Oscillator ready to use. |
| 28 | hirc7m_rdy | RO | 0 | **7.3728MHz Internal Oscillator Ready Status**<br>0: Not ready or not enabled.<br>1: Oscillator ready to use. |

| System Clock Control Register | | GCR_CLK_CTRL | | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 27 | hircmm_rdy | RO | 1 | **120MHz Internal Oscillator Ready Status**<br>On POR or System Reset this field reads 1 until the oscillator is ready.<br>1: Oscillator not ready/warmed up and cannot be used.<br>0: Oscillator ready for use. |
| 26 | hirc40m_rdy | RO | 1 | **40MHz Internal Oscillator Ready Status**<br>On POR or System Reset this field reads 1 until the oscillator is ready.<br>1: Oscillator not ready/warmed up and cannot be used.<br>0: Oscillator ready for use. |
| 25 | x32k_rdy | RO | 0 | **32.768kHz External Oscillator Ready Status**<br>On POR or System Reset this field reads 0 until the oscillator is ready.<br>0: Oscillator not ready/warmed up and cannot be used.<br>1: Oscillator ready for use. |
| 24:22 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 21 | hirc7m_vs | R/W | 0 | **7.3728MHz Internal Oscillator Voltage Source Select**<br>0: $V_{CORE}$<br>1: Internal 1V regulator sourced from pin VDDA<br>In DEEPSLEEP the 7.3728MHz oscillator voltage is sourced by pin VDDA. When exiting DEEPSLEEP the voltage is automatically switched back to this bit setting. |
| 20 | hirc7m_en | R/W | 0 | **7.3728MHz Internal Oscillator Enable**<br>Set to 1 to enable the oscillator. Read the *hirc7m_rdy* field to determine when the oscillator is ready for use after enabling it.<br>0: Disabled<br>1: Enabled |
| 19 | hircmm_en | R/W | 1 | **120MHz Internal Oscillator Enable**<br>Write 0 to disable the internal 120MHz oscillator.<br>0: Disabled<br>1: Enabled |
| 18 | hirc40m_en | R/W | 1 | **40MHz Internal Oscillator Enable**<br>Write 0 to disable the internal 40MHz oscillator.<br>0: Disabled<br>1: Enabled |
| 17 | x32k_en | R/W | 0 | **32.768kHz External Oscillator Enable**<br>Set to 1 to enable the 32kHz external oscillator. Read the *x32k_rdy* field to determine when the oscillator is ready for use after enabling it.<br>0: Disabled<br>1: Enabled |
| 16 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | ccd | R/W | 0 | **Crypto Accelerator Clock Divider Status**<br>0: Crypto clock divide by 1<br>1: Crypto clock is divide by 2 |
| 14 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |

| System Clock Control Register | | GCR_CLK_CTRL | | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 13 | sysosc_rdy | R/W | 0 | **SYSOSC Select Ready**<br>When SYSOSC is changed by modifying sysosc_sel, there is a delay until the switchover is complete. This bit is cleared until the switchover is complete.<br>0 = Switch to new clock source not yet complete.<br>1 = SYSOSC is clock source selected in sysosc_sel. |
| 12 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 11:9 | sysosc_sel | R/W | 0 | **System Oscillator Source Select**<br>Selects the system oscillator (SYSOSC) source used to generate the system clock (SYSCLK). Modifying this field immediately clears the *sysosc_rdy* field.<br>0: 40MHz LP Internal Oscillator<br>1: Reserved for Future Use<br>2: Reserved for Future Use<br>3: 8kHz Internal Oscillator<br>4: 120MHz Internal Oscillator<br>5: 7.3728MHz Internal Oscillator<br>6: 32.768kHz External Oscillator<br>7: Reserved for Future Use |
| 8:6 | sysclk_prescale | R/W | 0 | **System Oscillator Prescaler**<br>Sets the divider for generating SYSCLK from the selected SYSOSC as shown in the following equation:<br>$SYSCLK = \frac{SYSOSC}{2^{sysclk\_prescale}}$ |
| 5:0 | - | R/W | 0b01000 | **Reserved for Future Use**<br>Do not modify this field. |

*Table 3-11: Power Management Register*

| Power Management Register | | GCR_PMR | | 0x000C |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:18 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 17 | hirc7mpd | R/W | 1 | **7.3728MHz DEEPSLEEP Auto Off**<br>When set, this oscillator is automatically powered off when in DEEPSLEEP mode. It is restored to its previous state when returned to ACTIVE. |
| 16 | hircmmpd | R/W | 1 | **120MHz DEEPSLEEP Auto Off**<br>When set, this oscillator is automatically powered off when in DEEPSLEEP mode. It is restored to its previous state when returned to ACTIVE. |
| 15 | hirc40mpd | R/W | 1 | **40MHz DEEPSLEEP Auto Off**<br>When set, this oscillator is automatically powered off when in DEEPSLEEP mode. It is restored to its previous state when returned to ACTIVE. |
| 14:12 | - | R/W | 0b111 | **Reserved for Future Use**<br>Do not modify this field. |
| 11:7 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| Power Management Register | | | GCR_PMR | 0x000C |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 6 | usbwken | R/W | 0 | **USB Wakeup Enable**<br>When enabled, a USB wakeup event causes an exit from all low power modes and transitions directly to ACTIVE mode.<br>0: Wakeup from USB disabled.<br>1: Wakeup from USB enabled.<br>*Any USB bus activity or USB power on/off wakes up the device, except in BACKUP mode when only a USB power on/off wakes the device.* |
| 5 | rtcwken | R/W | 0 | **RTC Alarm Wakeup Enable**<br>Set this field to 1 to enable an RTC alarm to wake the device from any low power mode to ACTIVE mode.<br>0: Wakeup from RTC alarm disabled, regardless of if the RTC is configured to generate a wakeup alarm.<br>1: Wakeup from RTC alarm enabled. |
| 4 | gpiowken | R/W | 0 | **GPIO Wakeup Enable**<br>0: Wakeup from GPIO disabled.<br>1: Wakeup from GPIO enabled.<br>When enabled, activity on any GPIO pin configured for wakeup causes an exit from all low power modes and transitions directly to ACTIVE mode. |
| 3 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 2:0 | mode | R/W | 0 | **Operating Mode**<br>Configures the current operating mode for the device.<br>0b000: ACTIVE mode<br>0b011: Shutdown Mode<br>0b100: BACKUP Low Power Mode<br>*Note: All other values are Reserved for Future Use.* |

*Table 3-12: Peripheral Clock Divisor Register*

| Peripheral Clocks Divisor Register | | | GCR_PCLK_DIV | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 15:14 | aondiv | R/W | 0 | **Always-on Domain (AoD) Clock Divider**<br>Configures the frequency of the Always On Domain clock as shown in the following equation.<br>$f_{aod\_clk} = \frac{f_{HCLK}}{(4 \times 2^{aondiv})}$<br>*Note: aondiv valid values are 0, 1, 2 and 3.* |
| 13:10 | adcfrq | R/W | 0 | **ADC Clock Divider**<br>Configures the frequency of the ADC peripheral from the PCLK.<br>0x0 – 0x1: Invalid<br>0x2 – 0xF: $f_{adc\_clk} = \frac{f_{PCLK}}{adcfrq}$ |
| 9:8 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| Peripheral Clocks Divisor Register | | | GCR_PCLK_DIV | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7 | sdhcfrq | R/W | 0 | **SDHC Clock Frequency**<br>Configures the frequency of the SDHC as a divisor of the 120MHz high-speed oscillator.<br>0: $f_{SDHC\_CLK} = \frac{120MHz}{2}$<br>1: $f_{SDHC\_CLK} = \frac{120MHz}{4}$ |
| 6:4 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 3:0 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

*Table 3-13: Peripheral Clock Disable Register 0*

| Peripheral Clocks Disable 0 | | | GCR_PCLK_DIS0 | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31 | spixipm | R/W | 0 | **XSPI master Clock Disable**<br>Write 0 to enable or 1 to disable.<br>0: Enabled<br>1: Disabled |
| 30 | spixipf | R/W | 0 | **SPI-XIPF Clock Disable**<br>Write 0 to enable or 1 to disable.<br>0: Enabled<br>1: Disabled |
| 29 | pt | R/W | 0 | **Pulse Train Clock Disable**<br>Write 0 to enable or 1 to disable.<br>0: Enabled<br>1: Disabled |
| 28 | i2c1 | R/W | 0 | **I2C1 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>0: Enabled<br>1: Disabled |
| 27:26 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 25 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 24 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 23 | adc | R/W | 0 | **Analog to Digital Converter Clock Disable**<br>Write 0 to enable or 1 to disable.<br>0: Enabled<br>1: Disabled |
| 22:21 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |

| Peripheral Clocks Disable 0 | | | GCR_PCLK_DIS0 | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 20 | timer5 | R/W | 0 | **Timer5 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>0: Enabled<br>1: Disabled |
| 19 | timer4 | R/W | 0 | **Timer4 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>0: Enabled<br>1: Disabled |
| 18 | timer3 | R/W | 0 | **Timer3 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>0: Enabled<br>1: Disabled |
| 17 | timer2 | R/W | 0 | **Timer2 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>0: Enabled<br>1: Disabled |
| 16 | timer1 | R/W | 0 | **Timer1 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 15 | timer0 | R/W | 0 | **Timer0 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 14 | crypto | R/W | 0 | **Crypto Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 13 | i2c0 | R/W | 0 | **I2C0 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 12:11 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 10 | uart1 | R/W | 0 | **UART1 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 9 | uart0 | R/W | 0 | **UART0 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>0: Enabled<br>1: Disabled |
| 8 | spi2 | R/W | 0 | **SPI2 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |

| Peripheral Clocks Disable 0 | | | GCR_PCLK_DIS0 | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7 | spi1 | R/W | 0 | **SPI1 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 6 | spi0 | R/W | 0 | **SPI0 Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 5 | dma | R/W | 0 | **Standard DMA Clock Disable**<br>Write 0 to enable the Standard DMA peripheral clock or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 4 | tft | R/W | 0 | **TFT Controller Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 3 | usb | R/W | 0 | **USB Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 2 | gpio2 | R/W | 0 | **GPIO2 Port and Pad Logic Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 1 | gpio1 | R/W | 0 | **GPIO1 Port and Pad Logic Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |
| 0 | gpio0 | R/W | 0 | **GPIO0 Port and Pad Logic Clock Disable**<br>Write 0 to enable or 1 to disable.<br>1: Disabled<br>0: Enabled |

*Table 3-14: Memory Clock Control Register*

| Memory Clock Control | | | GCR_MEM_CLK | [0x0028] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:30 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 29 | romls | R/W | 0 | **ROM Light Sleep Enable**<br>Write 1 to put the ROM into Light Sleep low power state. Write 0 to enable the ROM for operation. |
| 28 | usbls | R/W | 0 | **USB FIFO Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access. |

| Memory Clock Control | | | GCR_MEM_CLK | [0x0028] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 27 | cryptols | R/W | 0 | **Crypto RAM Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access. |
| 26 | scachels | R/W | 0 | **Internal RAM Cache Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access. |
| 25 | icachexipls | R/W | 0 | **SPI-XIPF Instruction Cache RAM Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access.<br>*Note that the SPI-XIPF shares its cache with the HyperBus/Xccela interface.* |
| 24 | icachels | R/W | 0 | **Internal Flash ICache Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access. |
| 23 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 22 | sysram6ls | R/W | 0 | **System RAM 6 (0x200C 0000 - 0x200F FFFF) Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access. |
| 21 | sysram5ls | R/W | 0 | **System RAM 5 (0x2008 0000 - 0x200B FFFF) Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access. |
| 20 | sysram4ls | R/W | 0 | **System RAM 4 (0x2004 0000 - 0x2007 FFFF) Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access. |
| 19 | sysram3ls | R/W | 0 | **System RAM 3 (0x2002 0000 - 0x2003 FFFF) Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access. |
| 18 | sysram2ls | R/W | 0 | **System RAM 2 (0x2001 8000 - 0x2001 FFFF) Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access. |
| 17 | sysram1ls | R/W | 0 | **System RAM 1 (0x2000 8000 - 0x2001 7FFF) Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access. |

| Memory Clock Control | | | GCR_MEM_CLK | [0x0028] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 16 | sysram0ls | R/W | 0 | **System RAM 0 (0x2000 0000 - 0x2000 7FFF) Light Sleep Enable**<br>Write 1 to enter Light Sleep low power state. Data is unavailable for read/write operations in light sleep mode but is retained. Write 0 put the RAM into active mode for read and write access.<br>*Note: To put RAM in a shutdown mode that removes all power from the RAM and reset the RAM contents, use the LP_MEM_PWR register.* |
| 15:3 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 2:0 | fws | R/W | 0b101 | **Program Flash Wait States**<br>Number of wait-state cycles per Flash code read access.<br>0: Invalid<br>1 – 7: Number of Flash code access wait states<br>*Note: For the 40MHz clock and slower, minimum wait state is 1.*<br>*Note: For the 120MHz clock option, minimum wait state setting is 3.*<br>*Note: For the 96MHz clock option, the minimum wait state setting is 2.* |

*Table 3-15: Memory Zeroization Control Register*

| Memory Zeroization Control Register | | | GCR_MEM_ZERO | [0x002C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:15 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 14 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 13 | usbfifoz | R/W1 | 0 | **USB FIFO Zeroization**<br>Write 1 to clear the RAM to 0. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize memory |
| 12 | cryptoz | R/W1 | 0 | **Crypto MAA Memory Zeroization**<br>Write 1 to clear the Crypto MAA RAM. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize memory |
| 11 | scachetagz | R/W1 | 0 | **External Memory Controller Cache (EMCC) Tag Zeroization**<br>Write 1 to clear the EMCC tag RAM. The bit is set to 0 when the operation is complete.<br>0: Operation complete or not active<br>1: Zeroize EMCC Tag memory<br>*Note: This cache, the EMCC, is shared between the SPIXR and the HyperBus/Xccela interface.* |

| Memory Zeroization Control Register | | | GCR_MEM_ZERO | [0x002C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 10 | scachedataz | R/W1 | 0 | **External Memory Controller Cache (EMCC) Data Zeroization**<br>Write 1 to clear the EMCC Data RAM to 0. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize memory<br>*Note: This cache, the EMCC, is shared between the SPIXR and the HyperBus/Xccela interface.* |
| 9 | icachexipz | R/W1 | 0 | **ICC1 (SPI-XIPF) Cache Zeroization**<br>Write 1 to clear the ICC1 16KB cache RAM to 0. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize the ICC1 cache memory |
| 8 | icachez | R/W1 | 0 | **ICC0 Cache Zeroization**<br>Write 1 to clear the ICC0 16KB cache RAM to 0. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize the ICC0 16KB cache memory |
| 7 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 6 | sram6z | R/W1 | 0 | **Data RAM 6 (0x200C 0000 – 0x200F FFFF) Zeroization**<br>Write 1 to clear the RAM to 0. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize memory |
| 5 | sram5z | R/W1 | 0 | **Data RAM 5 (0x2008 0000 – 0x200B FFFF) Zeroization**<br>Write 1 to clear the RAM to 0. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize memory |
| 4 | sram4z | R/W1 | 0 | **Data RAM 4 (0x2004 0000 – 0x2007 FFFF) Zeroization**<br>Write 1 to clear the RAM to 0. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize memory |
| 3 | sram3z | R/W1 | 0 | **Data RAM 3 (0x2002 0000 – 0x2003 FFFF) Zeroization**<br>Write 1 to clear the RAM to 0. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize memory |
| 2 | sram2 | R/W1 | 0 | **Data RAM 2 (0x2001 8000 – 0x2001 FFFF) Zeroization**<br>Write 1 to clear the RAM to 0. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize memory |
| 1 | sram1z | R/W1 | 0 | **Data RAM 1 (0x2000 8000 – 0x2001 7FFF) Zeroization**<br>Write 1 to clear the RAM to 0. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize memory |
| 0 | sram0z | R/W1 | 0 | **Data RAM 0 (0x2000 0000 – 0x2000 7FFF) Zeroization**<br>Write 1 to clear the RAM to 0. The bit is set to 0 when the operation is complete.<br>0: Operation complete<br>1: Zeroize memory |

*Table 3-16: System Status Flag Register*

| System Status Flag Register | | GCR_SYS_STAT | | [0x0040] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:6 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 5 | scmemf | R | 0 | **HyperBus/Xccela Cache Memory Error Status Flag**<br>Indicates a memory fault has occurred in the cache while receiving data from the HyperBus/Xccela interface.<br>0 = Normal operation<br>1 = HyperBus/Xccela cache memory fault |
| 4:2 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | codeinterr | R | 0 | **Flash SPI-XIPF Code Integrity Error Status Flag**<br>This indicates a code integrity error has occurred in the Flash SPI-XIPF interface.<br>0 = Normal Operation<br>1 = SPI-XIPF code integrity error |
| 0 | icelock | R | 0 | **Arm ICE Lock Status Flag**<br>0: Arm ICE is unlocked (enabled)<br>1: Arm ICE is locked (disabled) |

*Table 3-17: Reset Register 1*

| Reset Register 1 | | GCR_RST1 | | [0x0044] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:17 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 16 | sema | R/W1O | 0 | **Semaphore Block Reset**<br>Write 1 to reset the peripheral state and reset the peripheral registers. When complete this field will read 0.<br>1: Write 1 to reset the peripheral. If this field reads 1, the peripheral is actively being reset by hardware.<br>0: Peripheral is reset if this field was previously written to 1, otherwise, not actively being reset. |
| 15 | xipr | R/W1O | 0 | **SPIXR Reset**<br>Write 1 to reset the SPI XIP RAM peripheral and reset the peripheral to the POR state. When complete this field is automatically cleared by hardware.<br>1: Write 1 to reset the peripheral. If this field reads 1, the peripheral is actively being reset by hardware.<br>0: Peripheral is reset if this field was previously written to 1, otherwise, not actively being reset. |
| 14:11 | Reserved | RO | - | **Reserved for Future Use**<br>Do not modify this field. |

| Reset Register 1 | | | GCR_RST1 | [0x0044] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 10 | I²S | R/W1 | 0 | **I²S (SPIMSS) Reset**<br>Write 1 to reset the peripheral state and reset the peripheral registers. When complete this field will read 0.<br>1: Write 1 to reset the peripheral. If this field reads 1, the peripheral is actively being reset by hardware.<br>0: Peripheral is reset if this field was previously written to 1, otherwise, not actively being reset. |
| 9 | spi3 | R/W1 | 0 | **SPI3 Reset**<br>Write 1 to reset the peripheral state and reset the peripheral registers. When complete this field will read 0.<br>1: Write 1 to reset the peripheral. If this field reads 1, the peripheral is actively being reset by hardware.<br>0: Peripheral is reset if this field was previously written to 1, otherwise, not actively being reset. |
| 8 | wdt1 | R/W1 | 0 | **Watchdog Timer 1 Reset**<br>Write 1 to reset the peripheral state and reset the peripheral registers. When complete this field will read 0.<br>1: Write 1 to reset the peripheral. If this field reads 1, the peripheral is actively being reset by hardware.<br>0: Peripheral is reset if this field was previously written to 1, otherwise, not actively being reset. |
| 7 | owire | R/W1 | 0 | **One-Wire Reset**<br>Write 1 to reset the peripheral state and reset the peripheral registers. When complete this field will read 0.<br>1: Write 1 to reset the peripheral. If this field reads 1, the peripheral is actively being reset by hardware.<br>0: Peripheral is reset if this field was previously written to 1, otherwise, not actively being reset. |
| 6 | sdhc | R/W1 | 0 | **SDHC Reset**<br>Write 1 to reset the peripheral state and reset the peripheral registers. When complete this field will read 0.<br>1: Write 1 to reset the peripheral. If this field reads 1, the peripheral is actively being reset by hardware.<br>0: Peripheral is reset if this field was previously written to 1, otherwise, not actively being reset. |
| 5 | gpio3 | R/W1 | 0 | **GPIO3 Reset**<br>Write 1 to reset the peripheral state and reset the peripheral registers. When complete this field will read 0.<br>1: Write 1 to reset the peripheral. If this field reads 1, the peripheral is actively being reset by hardware.<br>0: Peripheral is reset if this field was previously written to 1, otherwise, not actively being reset. |
| 4 | xspim | R/W1 | 0 | **XSPI Master Reset**<br>Write 1 to reset the peripheral state and reset the peripheral registers. When complete this field will read 0.<br>1: Write 1 to reset the peripheral. If this field reads 1, the peripheral is actively being reset by hardware.<br>0: Peripheral is reset if this field was previously written to 1, otherwise, not actively being reset. |

| Reset Register 1 | | | GCR_RST1 | [0x0044] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 3 | spixip | R/W1 | 0 | **SPI-XIPF Reset**<br>Write 1 to reset the peripheral state and reset the peripheral registers. When complete this field will read 0. |
| 2 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | pt | R/W1 | 0 | **Pulse Train Reset**<br>Write 1 to reset the peripheral state and reset the peripheral registers. When complete this field will read 0. |
| 0 | i2c1 | R/W1 | 0 | **I2C1 Reset**<br>Write 1 to reset the peripheral state and reset the peripheral registers. When complete this field will read 0. |

*Table 3-18: Peripheral Clock Disable Register 1*

| Peripheral Clock Disable Register 1 | | | GCR_PCLK_DIS1 | [0x0048] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:21 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 20 | spixipr | R/W | 0 | **SPIXR RAM Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 19:16 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | $I^2S$ | R/W | 0 | **$I^2S$ (SPIMSS) Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 14 | spi3 | R/W | 0 | **SPI3 Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 13 | ow | R/W | 0 | **One-Wire Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |

| Peripheral Clock Disable Register 1 | | GCR_PCLK_DIS1 | | [0x0048] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 12 | icachexipf | R/W | 0 | **SPI-XIPF Flash Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 11 | icache | R/W | 0 | **Flash Instruction Cache Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 10 | sdhc | R/W | 0 | **SDHC Controller Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 9 | smphr | R/W | 0 | **Semaphore Block Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 8 | sdma | R/W | 0 | **Smart DMA Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 7 | scache | R/W | 0 | **System Cache Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 6 | gpio3 | R/W | 0 | **GPIO3 Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 5 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| Peripheral Clock Disable Register 1 | | | GCR_PCLK_DIS1 | [0x0048] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4 | hbc | R/W | 0 | **HyperBus/Xccela Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 3 | sflc | R/W | 0 | **Secure Flash Controller Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 2 | trng | R/W | 0 | **TRNG Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 1 | uart2 | R/W | 0 | **UART2 Clock Disable**<br>Write 1 to disable the clock to the corresponding peripheral. Disabling a clock peripheral makes the peripheral non-functional while also saving power. Reads and writes to peripheral registers are disabled. Peripheral register states are retained.<br>1: Clock disabled to peripheral.<br>0: Clock enabled to peripheral. |
| 0 | - | R/W | - | **Reserved for Future Use**<br>Do not modify this field. |

*Table 3-19: Event Enable Register*

| Event Enable Register | | | GCR_EVENT_EN | [0x004C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:3 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 2 | txevent | R/W | 0 | **Transmit Event (TXEV) Pin Enable**<br>When set, an SEV (Send Event) instruction outputs a single-cycle pulse on output pin TXEV.<br>TXEV is Alternate Function 1 (AF1) on pin GPIO0.25 on this device. For proper operation, set *txevent* = 1 only when this pin is configured for AF1.<br>1: Transmit Event enabled on Send Event (SEV) instruction.<br>0: Transmit Event disabled. |
| 1 | rxevent | R/W | 0 | **Receive Event (RXEV) Pin Enable**<br>When set, a low to high transition on external input pin RXEV generates a Receive Event to wakeup the device from a low power mode entered with a WFE instruction. RXEV is AF1 on pin GPIO0.24 on this device. To properly enable the RXEV external wakeup function, this pin must only be configured for GPIO or AF1 when *rxevent* = 1.<br>1: A Receive Event (RXEV) is generated when an external event is triggered.<br>0: A Receive Event (RXEV) will not be generated and is disabled. |

| Event Enable Register | | GCR_EVENT_EN | | [0x004C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | dmaevent | R/W | 0 | **DMA CTZ Event Wake-Up Enable**<br>When set, when a DMA block transfer is completed and the DMA counter *DMAn_CNT.cnt* = 0, a CTZ DMA event occurs which generates an RXEV to wake-up the device from a low power mode entered with a WFE instruction.<br>1: DMA CTZ Event Wake-Up Enabled<br>0: A DMA CTZ Event will not wake-up the device. |

*Table 3-20: Revision Register*

| Revision Register | | GCR_REV | | [0x0050] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 15:0 | revision | R | - | **Maxim Integrated Chip Revision**<br>Returns the chip revision id. |

*Table 3-21: System Status Interrupt Enable Register*

| System Status Interrupt Enable | | GCR_SYS_STAT_IE | | [0x0054] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:6 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 5 | scmfie | R/W | 0 | **HyperBus/Xccela Cache Memory Fault Interrupt Enable**<br>When set, generates an interrupt if hardware detects an error in the HyperBus/Xccela code.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 4:2 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | cieie | R/W | 0 | **SPI-XIPF Code Integrity Error Interrupt Enable**<br>When set, generates an interrupt if hardware detects an error in the SPI-XIPF.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 0 | iceulie | R/W | 0 | **Arm ICE Unlocked Interrupt Enable**<br>When set, generates an interrupt if the Arm ICE is unlocked.<br>0: Interrupt disabled<br>1: Interrupt enabled |

## 3.15 Function Control Registers

Refer to *Table 2-1* for the Function Control Register's Base Peripheral Address.

*Table 3-22: Function Control Registers*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0800] | *GCR_FCR* | R/W | Function Control Register |
| [0x0804] | *GCR_AUTO_CAL* | R/W | Autocalibration Register |
| [0x0808] | *GCR_AUTO_CAL_TRIM* | R/W | Autocalibration Trim Control Register |
| [0x080C] | *GCR_AUTO_CAL_CNT* | R/W | Autocalibration Count Register |
| [0x5810] | *AOD_HYP_CLK_CN* | R/W | HyperBus/Xccela Clock Control Register |

## 3.16 Function Control Register Details

*Table 3-23: Function Control Register 0*

| Function Control Register 0 | GCR_FCR | [0x0800] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:24 | - | RO | - | **Reserved for Future Use**<br>Do Not Modify this Field. |
| 23 | i2c1_scl_filter_en | R/W | 0 | **I2C1 SCL Filter Enable**<br>0: Filter disabled<br>1: Filter enabled |
| 22 | i2c1_sda_filter_en | R/W | 0 | **I2C1 SDA Filter Enable**<br>0: Filter disabled<br>1: Filter enabled |
| 21 | i2c0_scl_filter_en | R/W | 0 | **I2C0 SCL Filter Enable**<br>0: Filter disabled<br>1: Filter enabled |
| 20 | i2c0_sda_filter_en | R/W | 0 | **I2C0 SDA Filter Enable**<br>0: Filter disabled<br>1: Filter enabled |
| 19:18 | - | R/W | - | **Reserved for Future Use**<br>Do Not Modify this Field. |
| 17 | - | R/W | 1 | **Reserved for Future Use**<br>Do Not Modify this Field. |
| 16 | usb_clk_sel | R/W | 0 | **USB Reference Clock Source Select**<br>This selects the clock source for the USB Hi-Speed Interface.<br>0: High speed 120MHz oscillator<br>1: External clock input<br>*See the GPIO chapter for the external clock input pin* |
| 15:6 | - | R/W | - | **Reserved for Future Use**<br>Do Not Modify this Field. |
| 5:0 | - | R/W | 0x2A | **Reserved for Future Use**<br>Always write this field to 0x2A. Do not modify this field. |

*Table 3-24: Autocalibration Function Control Register 1*

| Autocalibration Function Control Register 1 | | | GCR_AUTO_CAL | [0x0804] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:20 | - | R/W | - | **Reserved for Future Use**<br>Do Not Modify this Field. |
| 19:8 | mu | R/W | 0 | **Auto Calibration Loop Gain** |
| 4 | atomic_cal_en | R/W1 | 0 | **Atomic Mode Start**<br>Write 1 to start atomic mode calibration. The calibration runs for the number of cycles indicated in *GCR_AUTO_CAL_CNT.loop_cnt* and then stops.<br>This bit is automatically cleared by hardware when the atomic mode calibration is completed.<br>0: Atomic mode calibration complete or not started<br>1: Perform atomic mode calibration |
| 3 | inv_gain | R/W | 0 | **Invert Gain** |
| 2 | ld_trim | R/W | 0 | **Load Initial Trim**<br>Write 1 to load the trim setting from *GCR_AUTO_CAL_TRIM.initial_trim* into the oscillator trim register. |
| 1 | auto_cal_start | R/W | 0 | **Auto-Calibration Start**<br>Write 1 to begin auto-calibration. |
| 0 | auto_cal_trim_en | R/W | 0 | **120MHz High Speed Oscillator Auto-Calibration Enable**<br>0: Use factory trim settings for the oscillator<br>1: Use the trim setting in *GCR_AUTO_CAL_TRIM.initial_trim* |

*Table 3-25: Autocalibration Function Control Register 2*

| Autocalibration Function Control Register 2 | | | GCR_AUTO_CAL_TRIM | [0x0808] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:20 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 28:20 | max_trim | RO | See description | **Maximum Trim Limit**<br>This is the upper limit of the trim value for the auto-calibration routine. This is a factory set field that is used by auto-calibration to ensure the calculated trim is valid for the device. |
| 19 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 18:10 | min_trim | RO | See description | **Minimum Trim Limit**<br>This is the lower limit of the trim value for the auto-calibration routine. This is a factory set field that is used by auto-calibration to ensure the calculated trim is valid for the device. |
| 9 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |

| Autocalibration Function Control Register 2 | | | GCR_AUTO_CAL_TRIM | [0x0808] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 8:0 | initial_trim | RO | See description | **Initial Trim**<br>Initial trim value for the device set during production test. This factory set field is used by auto-calibration to ensure the auto-calibration calculated trim is valid for the device. |

*Table 3-26: Autocalibration Function Control Register 3*

| Autocalibration Function Control Register 3 | | | GCR_AUTO_CAL_CNT | [0x080C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:8 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 7:0 | loop_cnt | R/W | 0 | **Auto-calibration Loop Counter Setting**<br>Set this field to the number of loops desired for the atomic mode calibration routine to execute for setting the trim value. |

*Table 3-27: HyperBus/Xccela Clock Control Register*

| HyperBus/Xccela Clock Control Register | | | AOD_HYP_CLK_CN | [0x5810] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:9 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 8 | rds_dll_en | R/W | 0 | **HyperBus Read Data Strobe Delay Lock Loop (DLL) Enable**<br>0: Disabled<br>1: Enabled<br>When the HyperBus is reading from an external device, RWDS acts as a Read Data Strobe (RDS).<br>*Note: Only reset on a POR.* |
| 7:4 | hyp_clkn_drv | R/W | 8 | **HyperBus/Xccela CK# Drive Setting**<br>*Note: Only reset on a POR.* |
| 3:0 | hyp_clk_drv | R/W | 8 | **HyperBus/Xccela CK Drive Setting**<br>*Note: Only reset on a POR.* |

## 3.17 AES Key Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the AES Key Registers' Base Peripheral Address.

*Table 3-28: AES Key Register Addresses and Descriptions*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *AES_KEY0* | R/W | 128-bit AES Key Register 0 |
| [0x0080] | *AES_KEY1* | R/W | 128-bit AES Key Register 1 |
| [0x0100] | *AES_KEY2* | R/W | 128-bit AES Key Register 2 |
| [0x0180] | *AES_KEY3* | R/W | 128-bit AES Key Register 3 |

## 3.18 AES Key Register Details

*Table 3-29: AES Key 0 and 1 Registers*

| AES Key 0 | | AES_KEY0 | | [0x0000] |
|---|---|---|---|---|
| **AES Key 1** | | **AES_KEY1** | | **[0x0080]** |
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 127:0 | aes_key | R/W | 0 | **AES 128-bit Key Registers**<br>These two registers make up the 256-bit AES key, with the most significant bits in *AES_KEY1* and the least significant bits in *AES_KEY0*.<br>*This register is reset only on AoD Reset.* |

*Table 3-30: AES Key 2 and 3 Registers*

| AES Key 2 | | AES_KEY2 | | [0x0100] |
|---|---|---|---|---|
| **AES Key 3** | | **AES_KEY3** | | **[0x0180]** |
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 127:0 | aes_key | R/W | - | **AES 128-bit Key Registers**<br>Each of these registers are loaded at system initialization with user-defined 128-bit keys.<br>Refer to the secure bootloader section in the TPU supplement for more information. |

## 3.19 Power Supply Monitoring

This device has a power monitor that monitors the external power supplies relative to the on-chip bandgap voltage. The following power supplies are monitored:

- VCORE ($V_{CORE}$) CPU Supply Voltage
- VDDIO ($V_{DDIO}$) GPIO Supply Voltage
- VDDIOH ($V_{DDIOH}$) GPIO High Supply Voltage
- VDDA ($V_{DDA}$) Analog Supply Voltage
- VRTC ($V_{RTC}$) AoD Voltage
- VDDB ($V_{DDB}$) USB Supply Voltage

Each of these supplies has a dedicated power monitor setting in the Low Power Control Register, *LP_CTRL*. When the corresponding power monitor is enabled, the input voltage pin is constantly monitored. If the voltage drops below the trigger threshold, all registers and peripherals in that power domain are reset. This improves reliability and safety by guarding against a low voltage condition corrupting the contents of the registers and the device state. Disabling a power monitor risks data corruption of internal registers and corruption of the device state should the input voltage drop below the safe minimum value.

VCORE, VDDA, and VRTC have power failure monitors. When enabled, if that power supply drops below the power fail reset voltage the entire device goes into a Power-On Reset.

See the MAX32650–MAX32652 data sheet for the trigger threshold values and power fail reset voltages. When any power supply monitor is tripped, a Power Fail Warning Interrupt is triggered.

## 3.20 AOD Low Power Control Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the AoD Low Power Control (LP_) Register's Base Peripheral Address.

Note: These registers are in the Always-on Domain and are only reset on a Power-On Reset.

*Table 3-31: Always-on-Domain Power Control Registers, Offsets, and Descriptions*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *LP_CTRL* | R/W | Low Power Voltage Control Register |
| [0x0004] | *LP_GPIO0_WK_FL* | R/W | GPIO0 Wakeup Enable Register |
| [0x0008] | *LP_GPIO0_WK_EN* | R/W | GPIO0 Wakeup Flags Register |
| [0x000C] | *LP_GPIO1_WK_FL* | R/W | GPIO1 Wakeup Status Flags |
| [0x0010] | *LP_GPIO1_WK_EN* | R/W | GPIO1 Wakeup Enable |
| [0x0014] | *LP_GPIO2_WK_FL* | R/W | GPIO2 Wakeup Status Flags |
| [0x0018] | *LP_GPIO2_WK_EN* | R/W | GPIO2 Wakeup Enable |
| [0x001C] | *LP_GPIO3_WK_FL* | R/W | GPIO3 Wakeup Status Flags |
| [0x0020] | *LP_GPIO3_WK_EN* | R/W | GPIO3 Wakeup Enable |
| [0x0030] | *LP_USB_WK_FL* | R/W | USB Wakeup Status Flags |
| [0x0034] | *LP_USB_WK_EN* | R/W | USB Wakeup Enable |
| [0x0040] | *LP_MEM_PWR* | R/W | RAM Shut Down Control |

## 3.21 AOD Low Power Control Register Details

*Table 3-32: Low Power Voltage Control Register*

| Low Power Voltage Control Register | | LP_CTRL | [0x0000] |
|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:28 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 27 | vddbmd | R/W | 0 | **VDDB ($V_{DDB}$) USB Supply Power Monitor Disable**<br>When enabled, if the voltage drops below the trigger threshold the USB enters POR. If disabled, the USB operates below the power monitor threshold potentially causing integrity issues in the USB domain.<br>Write 1 to disable. |
| 26:25 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 24 | vddiohmd | R/W | 0 | **VDDIOH ($V_{DDIOH}$) GPIO High Supply Power Monitor Disable**<br>When enabled, if the voltage drops below the trigger threshold the GPIO pad logic enters a POR. |
| 23 | vddiomd | R/W | 0 | **VDDIO ($V_{DDIO}$) GPIO Supply Power Monitor Disable**<br>When enabled, if the voltage drops below the trigger threshold the GPIO pad logic enters a POR.<br>Write 1 to disable. |

| Low Power Voltage Control Register | LP_CTRL | [0x0000] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 22 | vddamd | R/W | 0 | **VDDA ($V_{DDA}$) Analog Supply Power Monitor Disable**<br>Write 1 to disable the corresponding power monitor. With a power monitor enabled, if the voltage drops below the trigger threshold all registers in that power domain are reset. With a power monitor disabled, if the voltage drops below the trigger threshold the integrity of registers in that power domain is unknown.<br>Refer to the VDDA Power Fail Reset Voltage in the device data sheet for voltage level. |
| 21 | vrtcmd | R/W | 0 | **VRTC ($V_{RTC}$) AoD Power Monitor Disable**<br>Write 1 to disable the corresponding power monitor. With a power monitor enabled, if the voltage drops below the trigger threshold all registers in that power domain are reset. With a power monitor disabled, if the voltage drops below the trigger threshold the integrity of registers in that power domain is unknown.<br>Refer to the VRTC Power Fail Reset Voltage in the device data sheet for voltage level. |
| 20 | vcoremd | R/W | 0 | **VCORE ($V_{CORE}$) CPU Supply Power Monitor Disable**<br>Write 1 to disable the corresponding power monitor. With a power monitor enabled, if the voltage drops below the trigger threshold all registers in that power domain are reset. With a power monitor disabled, if the voltage drops below the trigger threshold the integrity of registers in that power domain is unknown.<br>Refer to the VCORE Power Fail Reset Voltage in the device data sheet for voltage level. |
| 19:13 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 12 | porvcoremd | R/W | 1 | **VCORE ($V_{CORE}$) POR Monitor for *DEEPSLEEP* and *BACKUP* Disable**<br>Write 1 to disable the power failure monitor. With the power failure monitor enabled, if the voltage drops below the trigger voltage the device enters a *Power-On Reset*. |
| 11 | bgoff | R/W | 1 | ***DEEPSLEEP* and *BACKUP* Modes Bandgap Off**<br>0: System Bandgap is always on<br>1: System Bandgap is off in DEEPSLEEP and BACKUP modes |
| 10 | fwkm | R/W | 0 | ***DEEPSLEEP* Mode Fast Wakeup Enable**<br>Set to 1 to enable fast wakeup from DEEPSLEEP mode. |
| 9 | bkgrnd | R/W | 0 | ***BACKGROUND* Mode Enable**<br>Write 1 to put device into BACKGROUND mode when DEEPSLEEP is enabled. |
| 8 | rregen | R/W | 0 | ***BACKUP* Mode RAM Retention Regulator Enable**<br>0: RAM Retention regulator disabled. Data RAM retention is enabled using VCORE.<br>1: RAM Retention regulator enabled. RAM retention in BACKUP mode is configured with the ramret field. |
| 7:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1:0 | ramret | R/W | 0b00 | **BACKUP Mode Data RAM Retention**<br>0b00: RAM retention in BACKUP mode disabled<br>0b01: Data RAM 0 (0x2000 0000 - 0x2000 7FFF) retention<br>0b1x: Data RAM 0 & 1 (0x2000 8000 - 0x2001 7FFF) retention<br>*Note: If LP_CTRL.rregen = 0 all data RAM retention is enabled in BACKUP mode and this field is Don't Care.* |

*Table 3-33: Low Power GPIO Wakeup Interrupt Enable Registers*

| | | |
|---|---|---|
| **GPIO0 Wakeup Interrupt Enable** | **LP_GPIO0_WK_EN** | **[0x0008]** |
| **GPIO1 Wakeup Interrupt Enable** | **LP_GPIO1_WK_EN** | **[0x0010]** |
| **GPIO2 Wakeup Interrupt Enable** | **LP_GPIO2_WK_EN** | **[0x0018]** |
| **GPIO3 Wakeup Interrupt Enable** | **LP_GPIO3_WK_EN** | **[0x0020]** |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | wakeen | R/W | 0 | **GPIO Pin Wakeup Interrupt Enable**<br>Write 1 to any bit to enable the corresponding pin on the 32-bit GPIO port to generate an interrupt to wakeup the device from any low power mode to ACTIVE mode.<br>A wakeup occurs on any low-to-high or high-to-low transition on the corresponding pin.<br>*Note: To enable the device to wakeup from a low power mode on a GPIO pin transition, first set the "GPIO Wakeup enable" register bit GCR_PMR.gpiowken = 1.* |

*Table 3-34: Low Power GPIO Wakeup Flag Registers*

| | | |
|---|---|---|
| **Low Power GPIO0 Wakeup Flags** | **LP_GPIO0_WK_FL** | **[0x0004]** |
| **Low Power GPIO1 Wakeup Flags** | **LP_GPIO1_WK_FL** | **[0x000C]** |
| **Low Power GPIO2 Wakeup Flags** | **LP_GPIO2_WK_FL** | **[0x0014]** |
| **Low Power GPIO3 Wakeup Flags** | **LP_GPIO3_WK_FL** | **[0x001C]** |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | wakest | R/W1C | 0 | **GPIO Pin Wakeup Status Flag**<br>When a GPIO pin transitions from low-to-high or high-to-low, the corresponding bit is set.<br>If the corresponding interrupt enable bit is set in LP_GPIO0_WK_EN, LP_GPIO1_WK_EN, LP_GPIO2_WK_EN, or LP_GPIO3_WK_EN, an interrupt is generated to wakeup the device from any low power mode to ACTIVE mode.<br>Write 1 to clear. Writing 0 has no effect.<br>*Note: To enable the device to wakeup from a low power mode on a GPIO pin transition, first set the "GPIO Wakeup enable" register bit GCR_PMR.gpiowken=1.* |

*Table 3-35: USB Wakeup Status Register*

| | | |
|---|---|---|
| **Low Power USB Wakeup Flag Register** | **LP_USB_WK_FL** | **[0x0030]** |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:3 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |

| Low Power USB Wakeup Flag Register | | | LP_USB_WK_FL | [0x0030] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 2 | usbvbuswkst | R/W1C | 0 | **USB $V_{BUS}$ State Change Detect Flag**<br>0: Normal operation<br>1: The USB has been powered on or off by plugging or unplugging an external USB Host.<br>*Note: If the corresponding bit in LP_USB_WK_EN register is set, the event generates an interrupt to wakeup the device from a low power mode.* |
| 1:0 | usblswkst | R/W1C | 0 | **USB Line State Change Detect Status Flag**<br>If one or both USB differential pair D+/D- pins change state, one or both of this field's bits are correspondingly set.<br>usblswkst[0] corresponds to D+<br>usblswkst[1] corresponds to D-<br>*Note: If the corresponding bit in LP_USB_WK_EN register is set, the event generates an interrupt to wakeup the device from a low power mode.* |

*Table 3-36: Low Power USB Wakeup Enable Register*

| Low Power USB Wakeup Enable | | | LP_USB_WK_EN | [0x0034] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:3 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 2 | usbvbuswken | R/W | 0 | **USB VBUS State Change Detect Interrupt Enable**<br>Write 1 to enable an interrupt and wakeup the device from any low power mode when *LP_USB_WK_FL.usbvbuswkst* = 1. |
| 1:0 | usblswken | R/W | 0 | **USB Line State Change Detect Interrupt Enable**<br>Write 0b11 to enable an interrupt and wakeup the device from any low power mode when *LP_USB_WK_FL.usblswkst* does not equal 0. |

*Table 3-37: Low Power RAM Power Control Register*

| Low Power RAM Power Control | | | LP_MEM_PWR | [0x0040] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:13 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 12 | romsd | R/W | 0 | **ROM Shut Down**<br>Write 1 to shut down the power to the ROM. |
| 11 | usbfifosd | R/W | 0 | **USB FIFO Shut Down**<br>Write 1 to shut off power to the USB FIFO.<br>*Note: When this field is set, the contents of the USB FIFO are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings.* |
| 10 | cryptosd | R/W | 0 | **Crypto MAA RAM Shut Down**<br>Write 1 to shut off power to the Crypto-MAA RAM.<br>*Note: When this field is set, the contents of the Crypto MAA RAM are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings.* |

| Low Power RAM Power Control | | | LP_MEM_PWR | [0x0040] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 9 | scachesd | R/W | 0 | **External Memory Controller Cache (EMCC) RAM Shut Down**<br>Write 1 to shut off power to the External Memory Cache 16KB RAM.<br>*Note: Setting this field to 1 does not bypass the EMCC's line buffer.*<br>*Note: When this field is set, the contents of the External Memory Cache RAM are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings.*<br>*Note: The EMC is shared between the SPIXR and the HyperBus/Xccela interface.* |
| 8 | icachexipsd | R/W | 0 | **ICC1 SPIXR Cache RAM Shut Down**<br>Write 1 to shut off power to the SPI-XIPF Cache RAM.<br>*Note: When this field is set, the contents of ICC1, the SPI-XIPF Cache RAM, are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings.* |
| 7 | icachesd | R/W | 0 | **Internal Flash Cache RAM Shut Down**<br>Write 1 to shut off power to the Internal Flash Memory Cache RAM.<br>*Note: When this field is set, the contents of the Internal Flash Memory Cache RAM are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings* |
| 6 | sram6sd | R/W | 0 | **System RAM 6 (0x200C0000 – 0x200FFFFF) Shut Down**<br>Write 1 to shut off power to System RAM 6.<br>*Note: When this field is set, the contents of the System RAM 6 are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings* |
| 5 | sram5sd | R/W | 0 | **System RAM 5 (0x20080000 – 0x200BFFFF) Shut Down**<br>Write 1 to shut off power to System RAM 5.<br>*Note: When this field is set, the contents of the System RAM 5 are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings* |
| 4 | sram4sd | R/W | 0 | **System RAM 4 (0x20040000 – 0x2007FFFF) Shut Down**<br>Write 1 to shut off power to System RAM 5.<br>*Note: When this field is set, the contents of the System RAM 4 are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings* |
| 3 | sram3sd | R/W | 0 | **System RAM 3 (0x20020000 – 0x2003FFFF) Shut Down**<br>Write 1 to shut off power to System RAM 3.<br>*Note: When this field is set, the contents of the System RAM 3 are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings* |
| 2 | sram2sd | R/W | 0 | **System RAM 2 (0x20018000 – 0x2001FFFF) Shut Down**<br>Write 1 to shut off power to System RAM 2.<br>*Note: When this field is set, the contents of the System RAM 2 are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings* |
| 1 | sram1sd | R/W | 0 | **System RAM 1 (0x20008000 – 0x20017FFF) Shut Down**<br>Write 1 to shut off power to System RAM 1.<br>*Note: When this field is set, the contents of the System RAM 1 are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings* |
| 0 | sram0sd | R/W | 0 | **System RAM 0 (0x20000000 – 0x20007FFF) Shut Down**<br>Write 1 to shut off power to System RAM 0.<br>*Note: When this field is set, the contents of the System RAM 0 are destroyed. Refer to GCR_MEM_CLK register for retention mode power settings* |

# 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 MAX32650 interrupt vector table and describes each exception and interrupt.

## 4.1 Features

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

## 4.2 Interrupt Vector Table

*Table 4-1* lists the interrupt and exception table for the MAX32650. There are 80 interrupt entries for the MAX32650, 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 96. The Interrupt Vector Table alignment must be on a 128-word boundary, the next power of 2 greater than the 96-total interrupt vector table entries.

*Table 4-1: MAX32650 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 for Future Use |
| 11 | [0x002C] | SVC_Handler | Supervisor Call Exception |
| 12 | [0x0030] | DebugMon_Handler | Debug Monitor Exception |
| 13 | [0x0034] | - | Reserved for Future Use |
| 14 | [0x0038] | PendSV_Handler | Request Pending for System Service |
| 15 | [0x003C] | SysTick_Handler | System Tick Timer |
| 16 | [0x0040] | SysFault_IRQHandler | System Fault interrupt |
| 17 | [0x0044] | WDT0_IRQHandler | Watchdog Timer 0 Interrupt |
| 18 | [0x0048] | USB_IRQHandler | USB Interrupt |
| 19 | [0x004C] | RTC_IRQHandler | Real-Time Clock Interrupt |
| 20 | [0x0050] | - | Reserved for Future Use |
| 21 | [0x0054] | TMR0_IRQHandler | Timer 0 Interrupt |
| 22 | [0x0058] | TMR1_IRQHandler | Timer 1 Interrupt |

| Exception (Interrupt) Number | Offset | Name | Description |
|---|---|---|---|
| 23 | [0x005C] | TMR2_IRQHandler | Timer 2 Interrupt |
| 24 | [0x0060] | TMR3_IRQHandler | Timer 3 Interrupt |
| 25 | [0x0064] | TMR4_IRQHandler | Timer 4 Interrupt |
| 26 | [0x0068] | TMR5_IRQHandler | Timer 5 Interrupt |
| 27 | [0x006C] | - | Reserved for Future Use |
| 28 | [0x0070] | CLCD_IRQHandler | CLCD Controller Interrupt |
| 29 | [0x0074] | I2C0_IRQHandler | I2C Port 0 Interrupt |
| 30 | [0x0078] | UART0_IRQHandler | UART Port 0 Interrupt |
| 31 | [0x007C] | UART1_IRQHandler | UART Port 1 Interrupt |
| 32 | [0x0080] | SPI0_IRQHandler | SPI Port 0 Interrupt |
| 33 | [0x0084] | SPI1_IRQHandler | SPI Port 1 Interrupt |
| 34 | [0x0088] | SPI2_IRQHandler | SPI Port 2 Interrupt |
| 35 | [0x008C] | - | Reserved for Future Use |
| 36 | [0x0090] | ADC_IRQHandler | ADC Interrupt |
| 37:38 | [0x0094]:[0x098] | - | Reserved for Future Use |
| 39 | [0x009C] | FLC_IRQHandler | Flash Controller Interrupt |
| 40 | [0x00A0] | GPIO0_IRQHandler | GPIO Port 0 Interrupt |
| 41 | [0x00A4] | GPIO1_IRQHandler | GPIO Port 1 Interrupt |
| 42 | [0x00A8] | GPIO2_IRQHandler | GPIO Port 2 Interrupt |
| 43 | [0x00AC] | - | Reserved for Future Use |
| 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 for Future Use |
| 50 | 0x00C8 | UART2_IRQHandler | UART Port 2 Interrupt |
| 51 | 0x00CC | - | Reserved for Future Use |
| 52 | 0x00D0 | I2C1_IRQHandler | I2C Port 1 Interrupt |
| 53 | 0x00D4 | - | Reserved for Future Use |
| 54 | 0x00D8 | SPIXC_IRQHandler | SPI XIP Interrupt |
| 55:69 | [0x00DC]: [0x0114] | - | Reserved for Future Use |
| 70 | [0x0118] | GPIOWAKE_IRQHandler | GPIO or USB Wakeup Interrupt |
| 71 | [0x011C] | - | Reserved for Future Use |
| 72 | [0x0120] | SPI3_IRQHandler | SPI Port 3 Interrupt |
| 73 | [0x0124] | WDT1_IRQHandler | Watchdog Timer 1 Interrupt |

| Exception (Interrupt) Number | Offset | Name | Description |
|---|---|---|---|
| 74 | [0x0128] | GPIO3_IRQHandler | GPIO Port 3 Interrupt |
| 75 | [0x012C] | PT_IRQHandler | Pulse Train Interrupt |
| 76 | [0x0130] | SDMA_IRQHandler | Smart DMA Interrupt |
| 77 | [0x0134] | HPB_IRQHandler | HyperBus Interrupt |
| 78:81 | [0x0138]: [0x0144] | - | Reserved for Future Use |
| 82 | [0x0148] | SDHC_IRQHandler | SDHC Interrupt |
| 83 | [0x014C] | OWM_IRQHandler | 1-Wire Master Interrupt |
| 84 | [0x0150] | DMA4_IRQHandler | DMA4 Interrupt |
| 85 | [0x0154] | DMA5_IRQHandler | DMA5 Interrupt |
| 86 | [0x0158] | DMA6_IRQHandler | DMA6 Interrupt |
| 87 | [0x015C] | DMA7_IRQHandler | DMA7 Interrupt |
| 88 | [0x0160] | DMA8_IRQHandler | DMA8 Interrupt |
| 89 | [0x0164] | DMA9_IRQHandler | DMA9 Interrupt |
| 90 | [0x0168] | DMA10_IRQHandler | DMA10 Interrupt |
| 91 | [0x016C] | DMA11_IRQHandler | DMA11 Interrupt |
| 92 | [0x0170] | DMA12_IRQHandler | DMA12 Interrupt |
| 93 | [0x0174] | DMA13_IRQHandler | DMA13 Interrupt |
| 94 | [0x0178] | DMA14_IRQHandler | DMA14 Interrupt |
| 95 | [0x017C] | DMA15_IRQHandler | DMA15 Interrupt |
| 96 | [0x0180] | USBDMA_IRQHandler | USB DMA Interrupt |

# 5 General-Purpose I/O and Alternate Function Pins

General-purpose I/O (GPIO) pins share both a firmware-controlled I/O mode and up to two peripheral alternate functions (AFs). Each pin is individually enabled for GPIO or peripheral alternate function 1 (AF1) or alternate function 2 (AF2). Configuring a pin for an alternate function supersedes its use as a firmware-controlled I/O, however the input data is always readable via the GPIO input register if the GPIO input is enabled. Multiplexing between the alternate functions and the I/O function is often static in an application; set at initialization and dedicated as either an alternate function or an I/O. If needed, dynamic multiplexing between AF1, AF2 and I/O mode is supported if the application manages the AFs and I/O to ensure each is set up properly when switching from a peripheral to the I/O function. Refer to the data sheet electrical characteristics table for information on the GPIO pin behavior based on the configurations described in this document.

## 5.1 Features

Each port pin is independently configurable[1] by the application code. The features for each GPIO pin include:

- Output modes: push-pull or open drain with weak or strong pullup or pulldown
- Input modes: high-impedance, weak or strong pullup, or weak or strong pulldown
- Output data from GPIO register (GPIO0_OUT, GPIO1_OUT, GPIO2_OUT, and GPIO3_OUT) or peripheral (AF1 or AF2)
- Input data readable via the GPIO input registers (GPIO0_IN, GPIO1_IN, GPIO2_IN, GPIO3_IN) and, if enabled, the peripheral register interface
- Bit set and clear registers for efficient bit-wise write access to the pins and configuration registers
- Peripheral alternate function selection (up to two per pin)
- Wake from low-power modes using edge triggered inputs
- GPIO voltage supply selection ($V_{DDIO}$ or $V_{DDIOH}$)[2]
- Output drive strength selection of 1, 2, 4 or 8.
- Selectable weak (1MΩ) or strong (25KΩ) internal pullup or pulldown resistors.
- Optional interrupt generation selectable between:
- Level triggered (input low or input high) or
- Edge triggered (rising edge, falling edge or both).
- All GPIO pins default to input mode with weak-pullup during power-on-reset events.
- JTAG remains enabled and selected as the alternate function during all reset events (GPIO0[26:29])

## 5.2 General Description

The MAX32650**–MAX32652** includes four total GPIO ports; port 0 (P0), port 1 (P1), port 2 (P2) and port 3 (P3). P0, P1 and P2 support up to 32 GPIO pins each (P0[0:31], P1[0:31] and P2[0:31]) and P3 supports up to 10 GPIO pins (P3[0:9]). Each of the four ports maps to a GPIO register set with P0 mapped to GPIO0, P1 mapped to GPIO1, P2, mapped to GPIO2 and P3 mapped to GPIO3. Each GPIO port supports an identical register set for operation and control. For simplicity, this chapter refers to the GPIO registers generically using the nomenclature GPIOn, where n is either 0, 1, 2, or 3.

[1] Subject to hardware restrictions listed in the data sheet and register descriptions.
[2] GPIO which support HyperBus as AF1 or AF2 (P1[21:18], P1.[16:11], P3.0) are tied to the $V_{DDIO}$ supply and do not support $V_{DDIOH}$ supply selection.

A dedicated interrupt vector is assigned to each GPIO port and is detailed in the section *GPIO Interrupt Vectors*. The GPIO pins and the peripheral alternate functions available are package specific. *Table 5-1*, *Table 5-2* and *Table 5-3*, below, show the GPIO and the assigned AF1 and AF2 for the 140-WLP, 144-TQFP and 96-WLP packages of the MAX32650**–MAX32652**.

*Table 5-1: GPIO Port, Pin Name and Alternate Function Matrix, 140 WLP*

| 140-WLP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO0[0] | P0.0 | PT3 | SPIXF_SDIO2[3] |
| GPIO0[1] | P0.1 | SPIXR_SDIO0[3] | - |
| GPIO0[2] | P0.2 | SPIXR_SDIO2[3] | - |
| GPIO0[3] | P0.3 | SPIXR_SCK[3] | - |
| GPIO0[4] | P0.4 | SPIXR_SDIO3[3] | - |
| GPIO0[5] | P0.5 | SPIXR_SDIO1[3] | - |
| GPIO0[6] | P0.6 | SPIXR_SS0[3] | - |
| GPIO0[7] | P0.7 | SPIXF_SS0[3] | - |
| GPIO0[8] | P0.8 | SPIXF_SCK[3] | - |
| GPIO0[9] | P0.9 | SPIXF_SDIO1[3] | - |
| GPIO0[10] | P0.10 | SPIXF_SDIO0[3] | - |
| GPIO0[11] | P0.11 | SPIXF_SDIO2[3] | - |
| GPIO0[12] | P0.12 | SPIXF_SDIO3[3] | - |
| GPIO0[13] | P0.13 | SPI3_SS1 | CLCD_G0 |
| GPIO0[14] | P0.14 | SPI3_SS2 | CLCD_G1 |
| GPIO0[15] | P0.15 | SPI3_SDIO3 | CLCD_G2 |
| GPIO0[16] | P0.16 | SPI3_SCK | CLCD_G3 |
| GPIO0[17] | P0.17 | SPI3_SDIO2 | CLCD_G4 |
| GPIO0[18] | P0.18 | SPI3_SS3 | CLCD_G5 |
| GPIO0[19] | P0.19 | SPI3_SS0 | CLCD_G6 |
| GPIO0[20] | P0.20 | SPI3_SDIO1 | CLCD_G7 |
| GPIO0[21] | P0.21 | SPI3_SDIO0 | — |
| GPIO0[22] | P0.22 | SPI0_SS0 | CLCD_VDEN |
| GPIO0[23] | P0.23 | PT15 | CLCD_CLK |
| GPIO0[24] | P0.24 | RXEV | CLCD_HSYNC |
| GPIO0[25] | P0.25 | TXEV | CLCD_B0 |
| GPIO0[26] | P0.26 | TDI | TDI |
| GPIO0[27] | P0.27 | TDO | TDO |
| GPIO0[28] | P0.28 | TMS (SWDIO)[4] | TMS (SWDIO)[4] |
| GPIO0[29] | P0.29 | TCK (SWDCLK)[4] | TCK (SWDCLK)[4] |
| GPIO0[30] | P0.30 | — | CLCD_B0 |

[3] This signal is available on multiple pins as either AF1 or AF2 for the peripheral.
[4] Single wire debug when enabled.

| 140-WLP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO0[31] | P0.31 | 32KCAL | SDHC_CDN |
| GPIO1[0] | P1.0 | SDHC_CMD | SPIXF_SDIO3[3] |
| GPIO1[1] | P1.1 | SDHC_DAT2 | SPIXF_SDIO1[3] |
| GPIO1[2] | P1.2 | SDHC_WP | SPIXF_SS0[3] |
| GPIO1[3] | P1.3 | SDHC_DAT3 | CLCD_CLK |
| GPIO1[4] | P1.4 | SDHC_DAT0 | SPIXF_SDIO0[3] |
| GPIO1[5] | P1.5 | SDHC_CLK | SPIXF_SCK[3] |
| GPIO1[6] | P1.6 | SDHC_DAT1 | PT0 |
| GPIO1[7] | P1.7 | UART2_CTS | PT1 |
| GPIO1[8] | P1.8 | UART2_RTS | PT2 |
| GPIO1[9] | P1.9 | UART2_RX | PT3 |
| GPIO1[10] | P1.10 | UART2_TX | PT4 |
| GPIO1[11] | P1.11 | HYP_CS0N | SPIXR_SDIO0[3] |
| GPIO1[12] | P1.12 | HYP_D0 | SPIXR_SDIO1[3] |
| GPIO1[13] | P1.13 | HYP_D4 | SPIXR_SS0[3] |
| GPIO1[14] | P1.14 | HYP_RWDS | PT5 |
| GPIO1[15] | P1.15 | HYP_D1 | SPIXR_SDIO2[3] |
| GPIO1[16] | P1.16 | HYP_D5 | SPIXR_SCK[3] |
| GPIO1[17] | P1.17 | PT9 | - |
| GPIO1[18] | P1.18 | HYP_D6 | PT6 |
| GPIO1[19] | P1.19 | HYP_D2 | PT7 |
| GPIO1[20] | P1.20 | HYP_D3 | CLCD_HSYNC |
| GPIO1[21] | P1.21 | HYP_D7 | PT8 |
| GPIO1[22] | P1.22[5] | - | - |
| GPIO1[23] | P1.23 | SPI1_SS0 | CLCD_B1 |
| GPIO1[24] | P1.24 | SPI1_SS2 | CLCD_B2 |
| GPIO1[25] | P1.25 | SPI1_SS1 | CLCD_B3 |
| GPIO1[26] | P1.26 | SPI1_SCK | CLCD_B4 |
| GPIO1[27] | P1.27 | SPI1_SS3 | CLCD_B5 |
| GPIO1[28] | P1.28 | SPI1_MISO | CLCD_B6 |
| GPIO1[29] | P1.29 | SPI1_MOSI | CLCD_B7 |
| GPIO1[30] | P1.30 | OWM_PUPEN | CLCD_R0 |
| GPIO1[31] | P1.31 | OWM_IO | CLCD_R1 |

[5] This pin is not pinned out in the 140-WLP.

| 140-WLP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO2[0] | P2.0 | SPI2_SS2 | PT9 |
| GPIO2[1] | P2.1 | SPI2_SS1 | PT10 |
| GPIO2[2] | P2.2 | SPI2_SCK (I2S_BCLK)† | CLCD_LEND |
| GPIO2[3] | P2.3 | SPI2_MISO (I2S_SDI)† | CLCD_PWREN |
| GPIO2[4] | P2.4 | SPI2_MOSI (I2S_SDO)† | - |
| GPIO2[5] | P2.5 | SPI2_SS0 (I2S_LRCLK)† | PT11 |
| GPIO2[6] | P2.6 | SPI2_SS3 | CLCD_VSYNC |
| GPIO2[7] | P2.7 | I2C0_SDA | - |
| GPIO2[8] | P2.8 | I2C0_SCL | - |
| GPIO2[9] | P2.9 | UART0_CTS | PT12 |
| GPIO2[10] | P2.10 | UART0_RTS | PT14 |
| GPIO2[11] | P2.11 | UART0_RX | PT13 |
| GPIO2[12] | P2.12 | UART0_TX | PT15 |
| GPIO2[13] | P2.13 | UART1_CTS | CLCD_R2 |
| GPIO2[14] | P2.14 | UART1_RX | CLCD_R3 |
| GPIO2[15] | P2.15 | UART1_RTS | CLCD_R4 |
| GPIO2[16] | P2.16 | UART1_TX | CLCD_R5 |
| GPIO2[17] | P2.17 | I2C1_SDA | CLCD_R6 |
| GPIO2[18] | P2.18 | I2C1_SCL | CLCD_R7 |
| GPIO2[19] | P2.19 | PT4 | - |
| GPIO2[20] | P2.20 | PT5 | - |
| GPIO2[21] | P2.21 | PT7 | - |
| GPIO2[22] | P2.22 | PT8 | - |
| GPIO2[23] | P2.23 | PT6 | SPIXR_SDIO3[3] |
| GPIO2[24] | P2.24 | PT10 | - |
| GPIO2[25] | P2.25 | PT11 | - |
| GPIO2[26] | P2.26 | PT12 | - |
| GPIO2[27] | P2.27 | PT13 | - |
| GPIO2[28] | P2.28 | PT14 | - |
| GPIO2[29] | P2.29 | PT0 | - |
| GPIO2[30] | P2.30 | PT1 | - |
| GPIO2[31] | P2.31 | PT2 | - |
| GPIO3[0] | P3.0 | PDOWN | HYP_CS1N |
| GPIO3[1] | P3.1 | SPI0_MISO | - |

| 140-WLP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO3[2] | P3.2 | SPI0_MOSI | - |
| GPIO3[3] | P3.3 | SPI0_SCK | - |
| GPIO3[4] | P3.4 | TMR0 | - |
| GPIO3[5] | P3.5 | TMR2 | - |
| GPIO3[6] | P3.6 | TMR4 | - |
| GPIO3[7] | P3.7 | TMR1 | - |
| GPIO3[8] | P3.8 | TMR3 | - |
| GPIO3[9] | P3.9 | TMR5 | - |

*Table 5-2: GPIO Port, Pin Name and Alternate Function Matrix, 144 TQFP*

| 144-Pin TQFP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO0[0] | P0.0[6] | - | - |
| GPIO0[1] | P0.1 | SPIXR_SDIO0[7] | - |
| GPIO0[2] | P0.2 | SPIXR_SDIO2[7] | - |
| GPIO0[3] | P0.3 | SPIXR_SCK[7] | - |
| GPIO0[4] | P0.4 | SPIXR_SDIO3[7] | - |
| GPIO0[5] | P0.5 | SPIXR_SDIO1[7] | - |
| GPIO0[6] | P0.6 | SPIXR_SS0[7] | - |
| GPIO0[7] | P0.7 | SPIXF_SS0[7] | - |
| GPIO0[8] | P0.8 | SPIXF_SCK[7] | - |
| GPIO0[9] | P0.9 | SPIXF_SDIO1[7] | - |
| GPIO0[10] | P0.10 | SPIXF_SDIO0[7] | - |
| GPIO0[11] | P0.11 | SPIXF_SDIO2[7] | - |
| GPIO0[12] | P0.12 | SPIXF_SDIO3[7] | - |
| GPIO0[13] | P0.13 | SPI3_SS1 | CLCD_G0 |
| GPIO0[14] | P0.14 | SPI3_SS2 | CLCD_G1 |
| GPIO0[15] | P0.15 | SPI3_SDIO3 | CLCD_G2 |
| GPIO0[16] | P0.16 | SPI3_SCK | CLCD_G3 |
| GPIO0[17] | P0.17 | SPI3_SDIO2 | CLCD_G4 |
| GPIO0[18] | P0.18 | SPI3_SS3 | CLCD_G5 |
| GPIO0[19] | P0.19 | SPI3_SS0 | CLCD_G6 |

[6] This pin is not available in the 144-pin TQFP package.
[7] This signal is available on multiple pins as either AF1 or AF2 for the peripheral.

| 144-Pin TQFP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO0[20] | P0.20 | SPI3_SDIO1 | CLCD_G7 |
| GPIO0[21] | P0.21 | SPI3_SDIO0 | - |
| GPIO0[22] | P0.22 | SPI0_SS0 | CLCD_VDEN |
| GPIO0[23] | P0.23 | PT15 | CLCD_CLK |
| GPIO0[24] | P0.24 | RXEV | CLCD_HSYNC |
| GPIO0[25] | P0.25 | TXEV | CLCD_B0 |
| GPIO0[26] | P0.26 | TDI | - |
| GPIO0[27] | P0.27 | TDO | - |
| GPIO0[28] | P0.28 | TMS | - |
| GPIO0[29] | P0.29 | TCK | - |
| GPIO0[30] | P0.30 | - | CLCD_B0 |
| GPIO0[31] | P0.31 | 32KCAL | SDHC_CDN |
| GPIO1[0] | P1.0 | SDHC_CMD | SPIXF_SDIO3[7] |
| GPIO1[1] | P1.1 | SDHC_DAT2 | SPIXF_SDIO1[7] |
| GPIO1[2] | P1.2 | SDHC_WP | SPIXF_SS0[7] |
| GPIO1[3] | P1.3 | SDHC_DAT3 | CLCD_CLK |
| GPIO1[4] | P1.4 | SDHC_DAT0 | SPIXF_SDIO0[7] |
| GPIO1[5] | P1.5 | SDHC_CLK | SPIXF_SCK[7] |
| GPIO1[6] | P1.6 | SDHC_DAT1 | PT0 |
| GPIO1[7] | P1.7 | UART2_CTS | PT1 |
| GPIO1[8] | P1.8 | UART2_RTS | PT2 |
| GPIO1[9] | P1.9 | UART2_RX | PT3 |
| GPIO1[10] | P1.10 | UART2_TX | PT4 |
| GPIO1[11] | P1.11 | HYP_CS0 | SPIXR_SDIO0[7] |
| GPIO1[12] | P1.12 | HYP_D0 | SPIXR_SDIO1[7] |
| GPIO1[13] | P1.13 | HYP_D4 | SPIXR_SS0[7] |
| GPIO1[14] | P1.14 | HYP_RWDS | PT5 |
| GPIO1[15] | P1.15 | HYP_D1 | SPIXR_SDIO2[7] |
| GPIO1[16] | P1.16 | HYP_D5 | SPIXR_SCK[7] |
| GPIO1[17] | P1.17 | PT9 | - |
| GPIO1[18] | P1.18 | HYP_D6 | PT6 |
| GPIO1[19] | P1.19 | HYP_D2 | PT7 |
| GPIO1[20] | P1.20 | HYP_D3 | CLCD_HSYNC |
| GPIO1[21] | P1.21 | HYP_D7 | PT8 |

| 144-Pin TQFP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO1[22] | P1.22[6] | - | - |
| GPIO1[23] | P1.23 | SPI1_SS0 | CLCD_B1 |
| GPIO1[24] | P1.24 | SPI1_SS2 | CLCD_B2 |
| GPIO1[25] | P1.25 | SPI1_SS1 | CLCD_B3 |
| GPIO1[26] | P1.26 | SPI1_SCK | CLCD_B4 |
| GPIO1[27] | P1.27 | SPI1_SS3 | CLCD_B5 |
| GPIO1[28] | P1.28 | SPI1_MISO | CLCD_B6 |
| GPIO1[29] | P1.29 | SPI1_MOSI | CLCD_B7 |
| GPIO1[30] | P1.30 | OWM_PUPEN | CLCD_R0 |
| GPIO1[31] | P1.31 | OWM_IO | CLCD_R1 |
| GPIO2[0] | P2.0 | SPI2_SS2 | PT9 |
| GPIO2[1] | P2.1 | SPI2_SS1 | PT10 |
| GPIO2[2] | P2.2 | SPI2_SCK (I2S_BCLK)[8] | CLCD_LEND |
| GPIO2[3] | P2.3 | SPI2_MISO (I2S_LRCLK)[9] | CLCD_PWREN |
| GPIO2[4] | P2.4 | SPI2_MOSI (I2S_SD)[10] | - |
| GPIO2[5] | P2.5 | SPI2_SS0 | PT11 |
| GPIO2[6] | P2.6 | SPI2_SS3 | CLCD_VSYNC |
| GPIO2[7] | P2.7 | I2C0_SDA | - |
| GPIO2[8] | P2.8 | I2C0_SCL | - |
| GPIO2[9] | P2.9 | UART0_CTS | PT12 |
| GPIO2[10] | P2.10 | UART0_RTS | PT14 |
| GPIO2[11] | P2.11 | UART0_RX | PT13 |
| GPIO2[12] | P2.12 | UART0_TX | PT15 |
| GPIO2[13] | P2.13 | UART1_CTS | CLCD_R2 |
| GPIO2[14] | P2.14 | UART1_RX | CLCD_R3 |
| GPIO2[15] | P2.15 | UART1_RTS | CLCD_R4 |
| GPIO2[16] | P2.16 | UART1_TX | CLCD_R5 |
| GPIO2[17] | P2.17 | I2C1_SDA | CLCD_R6 |
| GPIO2[18] | P2.18 | I2C1_SCL | CLCD_R7 |
| GPIO2[19] | P2.19[6] | - | - |
| GPIO2[20] | P2.20[6] | - | - |
| GPIO2[21] | P2.21[6] | - | - |

[8] When the I2S peripheral is enabled, Alternate Function 1 for this pin is I2S_BCLK and SPI2_SCK is not available.
[9] When the I2S peripheral is enabled, Alternate Function 1 for this pin is I2S_LRCLK and SPI2_MISO is not available.
[10] When the I2S peripheral is enabled, Alternate Function 1 for this pin is I2S_SD and SPI2_MOSI is not available.

| 144-Pin TQFP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO2[22] | P2.22[6] | - | - |
| GPIO2[23] | P2.23 | PT6 | SPIXR_SDIO3[7] |
| GPIO2[24] | P2.24[6] | - | - |
| GPIO2[25] | P2.25 | PT11 | - |
| GPIO2[26] | P2.26 | PT12 | - |
| GPIO2[27] | P2.27[6] | - | - |
| GPIO2[28] | P2.28 | PT14 | - |
| GPIO2[29] | P2.29[6] | - | - |
| GPIO2[30] | P2.30 | PT1 | - |
| GPIO2[31] | P2.31[6] | - | - |
| GPIO3[0] | P3.0 | PDOWN | HYP_CS1 |
| GPIO3[1] | P3.1 | SPI0_MISO | - |
| GPIO3[2] | P3.2 | SPI0_MOSI | - |
| GPIO3[3] | P3.3 | SPI0_SCK | - |
| GPIO3[4] | P3.4 | TMR0 | - |
| GPIO3[5] | P3.5 | TMR2 | - |
| GPIO3[6] | P3.6 | TMR4 | - |
| GPIO3[7] | P3.7 | TMR1 | - |
| GPIO3[8] | P3.8 | TMR3 | - |
| GPIO3[9] | P3.9 | TMR5 | - |

*Table 5-3: GPIO Port, Pin Name and Alternate Function Matrix, 96 WLP*

| 96-Pin WLP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO0[0] | P0.0[11] | - | - |
| GPIO0[1] | P0.1[11] | - | - |
| GPIO0[2] | P0.2[11] | - | - |
| GPIO0[3] | P0.3[11] | - | - |
| GPIO0[4] | P0.4[11] | - | - |
| GPIO0[5] | P0.5[11] | - | - |
| GPIO0[6] | P0.6[11] | - | - |
| GPIO0[7] | P0.7[11] | - | - |
| GPIO0[8] | P0.8[11] | - | - |

[11] This pin is not available in the 96-WLP.

| 96-Pin WLP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO0[9] | P0.9[11] | - | - |
| GPIO0[10] | P0.10[11] | - | - |
| GPIO0[11] | P0.11 | SPIXF_SDIO2[12] | P0.11 |
| GPIO0[12] | P0.12[11] | - | - |
| GPIO0[13] | P0.13 | SPI3_SS1 | CLCD_G0 |
| GPIO0[14] | P0.14 | SPI3_SS2 | CLCD_G1 |
| GPIO0[15] | P0.15 | SPI3_SDIO3 | CLCD_G2 |
| GPIO0[16] | P0.16 | SPI3_SCK | CLCD_G3 |
| GPIO0[17] | P0.17 | SPI3_SDIO2 | CLCD_G4 |
| GPIO0[18] | P0.18 | SPI3_SS3 | CLCD_G5 |
| GPIO0[19] | P0.19 | SPI3_SS0 | CLCD_G6 |
| GPIO0[20] | P0.20 | SPI3_SDIO1 | CLCD_G7 |
| GPIO0[21] | P0.21 | SPI3_SDIO0 | - |
| GPIO0[22] | P0.22 | SPI0_SCK | CLCD_VDEN |
| GPIO0[23] | P0.23[11] | - | - |
| GPIO0[24] | P0.24 | RXEV | - |
| GPIO0[25] | P0.25[11] | - | - |
| GPIO0[26] | P0.26 | TDI | - |
| GPIO0[27] | P0.27 | TDO | - |
| GPIO0[28] | P0.28 | TMS | - |
| GPIO0[29] | P0.29 | TCK | - |
| GPIO0[30] | P0.30 | - | CLCD_B0 |
| GPIO0[31] | P0.31[11] | - | - |
| GPIO1[0] | P1.0 | SDHC_CMD | SPIXF_SDIO3[12] |
| GPIO1[1] | P1.1 | SDHC_DAT2 | SPIXF_SDIO1[12] |
| GPIO1[2] | P1.2 | SDHC_WP | SPIXF_SS0[12] |
| GPIO1[3] | P1.3 | SDHC_DAT3 | CLCD_CLK |
| GPIO1[4] | P1.4 | SDHC_DAT0 | SPIXF_SDIO0[12] |
| GPIO1[5] | P1.5 | SDHC_CLK | SPIXF_SCK[12] |
| GPIO1[6] | P1.6 | SDHC_DAT1 | PT0 |
| GPIO1[7] | P1.7[11] | - | - |
| GPIO1[8] | P1.8 | UART2_RTS | PT2 |
| GPIO1[9] | P1.9 | UART2_RX | PT3 |

[12] This signal is available on multiple pins as either AF1 or AF2 for the peripheral.

| 96-Pin WLP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO1[10] | P1.10 | UART2_TX | PT4 |
| GPIO1[11] | P1.11 | - | SPIXR_SDIO0[12] |
| GPIO1[12] | P1.12 | - | SPIXR_SDIO1[12] |
| GPIO1[13] | P1.13 | - | SPIXR_SS0[12] |
| GPIO1[14] | P1.14 | - | PT5 |
| GPIO1[15] | P1.15 | - | SPIXR_SDIO2[12] |
| GPIO1[16] | P1.16 | - | SPIXR_SCK[12] |
| GPIO1[17] | P1.17[11] | - | - |
| GPIO1[18] | P1.18 | - | PT6 |
| GPIO1[19] | P1.19 | - | PT7 |
| GPIO1[20] | P1.20 | - | CLCD_HSYNC |
| GPIO1[21] | P1.21 | - | PT8 |
| GPIO1[22] | P1.22[11] | - | - |
| GPIO1[23] | P1.23 | SPI1_SS0 | CLCD_B1 |
| GPIO1[24] | P1.24 | SPI1_SS2 | CLCD_B2 |
| GPIO1[25] | P1.25 | SPI1_SS1 | CLCD_B3 |
| GPIO1[26] | P1.26 | SPI1_SCK | CLCD_B4 |
| GPIO1[27] | P1.27 | SPI1_SS3 | CLCD_B5 |
| GPIO1[28] | P1.28 | SPI1_MISO | CLCD_B6 |
| GPIO1[29] | P1.29 | SPI1_MOSI | CLCD_B7 |
| GPIO1[30] | P1.30 | OWM_PUPEN | CLCD_R0 |
| GPIO1[31] | P1.31 | OWM_IO | CLCD_R1 |
| GPIO2[0] | P2.0 | SPI2_SS2 | PT9 |
| GPIO2[1] | P2.1[11] | - | - |
| GPIO2[2] | P2.2 | SPI2_SCK (I2S_BCLK)[13] | CLCD_LEND |
| GPIO2[3] | P2.3 | SPI2_MISO (I2S_LRCLK)[14] | CLCD_PWREN |
| GPIO2[4] | P2.4 | SPI2_MOSI (I2S_SD)[15] | - |
| GPIO2[5] | P2.5 | SPI2_SS0 | PT11 |
| GPIO2[6] | P2.6 | SPI2_SS3 | CLCD_VSYNC |
| GPIO2[7] | P2.7[11] | - | - |
| GPIO2[8] | P2.8[11] | - | - |
| GPIO2[9] | P2.9 | UART0_CTS | PT12 |

[13] When the I2S peripheral is enabled, Alternate Function 1 for this pin is I2S_BCLK and SPI2_SCK is not available.
[14] When the I2S peripheral is enabled, Alternate Function 1 for this pin is I2S_LRCLK and SPI2_MISO is not available.
[15] When the I2S peripheral is enabled, Alternate Function 1 for this pin is I2S_SD and SPI2_MOSI is not available.

| 96-Pin WLP | | | |
|---|---|---|---|
| **GPIO Port*[pin]*** | **GPIO** | **ALTERNATE FUNCTION 1** | **ALTERNATE FUNCTION 2** |
| GPIO2[10] | P2.10[11] | - | - |
| GPIO2[11] | P2.11 | UART0_RX | PT13 |
| GPIO2[12] | P2.12 | UART0_TX | PT15 |
| GPIO2[13] | P2.13 | UART1_CTS | CLCD_R2 |
| GPIO2[14] | P2.14 | UART1_RX | CLCD_R3 |
| GPIO2[15] | P2.15 | UART1_RTS | CLCD_R4 |
| GPIO2[16] | P2.16 | UART1_TX | CLCD_R5 |
| GPIO2[17] | P2.17 | I2C1_SDA | CLCD_R6 |
| GPIO2[18] | P2.1[8] | I2C1_SCL | CLCD_R7 |
| GPIO2[19] | P2.19[11] | - | - |
| GPIO2[20] | P2.20[11] | - | - |
| GPIO2[21] | P2.21[11] | - | - |
| GPIO2[22] | P2.22[11] | - | - |
| GPIO2[23] | P2.23 | - | SPIXR_SDIO3[12] |
| GPIO2[24] | P2.24[11] | - | - |
| GPIO2[25] | P2.25[11] | - | - |
| GPIO2[26] | P2.26[11] | - | - |
| GPIO2[27] | P2.27[11] | - | - |
| GPIO2[28] | P2.28[11] | - | - |
| GPIO2[29] | P2.29[11] | - | - |
| GPIO2[30] | P2.30[11] | - | - |
| GPIO2[31] | P2.31[11] | - | - |
| GPIO3[0] | P3.0[11] | - | - |
| GPIO3[1] | P3.1[11] | - | - |
| GPIO3[2] | P3.2[11] | - | - |
| GPIO3[3] | P3.3[11] | - | - |
| GPIO3[4] | P3.4 | TMR0 | - |
| GPIO3[5] | P3.5 | TMR2 | - |
| GPIO3[6] | P3.6 | TMR4 | - |
| GPIO3[7] | P3.7 | TMR1 | - |
| GPIO3[8] | P3.8 | TMR3 | - |
| GPIO3[9] | P3.9 | TMR5 | - |

## 5.3 GPIO

During a power-on-reset event, each GPIO defaults to input mode with the weak pullup resistor enabled. Following the reset or initial power-up, all GPIO are configured as follows (pin is any specific bit or bits in the GPIO register, GPIO0[0:31], GPIO1[0:31], GPIO2[0:31] and GPIO3[0:9]):

1. GPIO enabled (*GPIOn_EN[pin]* = 1), alternate function disabled except for the JTAG pins on port 0.
2. Input mode enabled (*GPIOn_IN_EN[pin]* = 1).
3. High impedance mode enabled (*GPIOn_PDPU_SEL0[pin]* = 0, *GPIOn_PDPU_SEL1[pin]* = 0), pullup and pulldown disabled.
4. Output mode disabled (*GPIOn_OUT_EN[pin]* = 0)
5. Interrupt disabled (*GPIOn_INT_EN[pin]* = 0)

*Note: The JTAG port, GPIO0[26:29], are an exception to these rules, and are always enabled as AF1 (GPIOn_AF_SEL[26:29] = 0) with the Peripheral Alternate Function mode enabled (GPIOn_EN[26:29] = 0) after a reset. To use the JTAG pins in I/O mode, set the GPIOn_EN[26:29] bits to 1.*

*Note: On parts without a JTAG debug port, the JTAG port is still available for boundary scan testing, however, the JTAG debug port is hardware disabled. To use the JTAG pins in I/O mode, set the GPIO0_EN[26:29] bits to 1 and clear the JTAG tap enabled field (GCR_SCON.bstapen = 0).*

### 5.3.1 *Input mode configuration*

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

1. Set the pin for I/O mode (*GPIOn_EN[pin]* = 1).
2. Enable the input buffer connection to the GPIO pin by setting *GPIOn_IN_EN[pin]* to 1.
3. Optionally configure the pin supply rail as $V_{DDIOH}$ by setting the *GPIOn_VSSEL[pin]* to 1. GPIO1[11:16], GPIO1[18:21], and GPIO3[0] do not support supply rail selection and are tied to $V_{DDIO}$.
4. Configure the pin for pullup, pulldown, or high-impedance mode using the *GPIOn_PDPU_SEL1[pin]* and *GPIOn_PDPU_SEL0[pin]* bits. Refer to *Table 5-4* for details.
5. If the input pin is set to pullup or pulldown, select the strength of the resistor by setting or clearing the *GPIOn_PSSEL[pin]* bit. Refer to *Table 5-4* for details.
6. Read the input state of the pin using the *GPIOn_IN[pin]* field.

### 5.3.2 *Output Mode Configuration*

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

1. Set the pin for I/O mode (*GPIOn_EN[pin]* = 1).
2. Enable the output buffer for the pin by setting *GPIOn_OUT_EN[pin]* to 1.
3. Set the output drive strength using the *GPIOn_DS_SEL1[pin]* and *GPIOn_DS_SEL0[pin]* bits. Refer to *GPIO Drive Strength Selection* for modes. Refer to the MAX32650–MAX32652 data sheet for the electrical characteristics for the drive strength selection modes.
4. Set the output high or low using the *GPIOn_OUT[pin]* bit.

### 5.3.3 *Alternate Function Configuration*

Perform the following steps to configure a pin for a peripheral alternate function mode:

1. Set the pin for alternate function mode (*GPIOn_EN[pin]* = 0).
2. Select the alternate function option, either AF1 (*GPIOn_AF_SEL[pin]* = 0) or AF2 (*GPIOn_AF_SEL[pin]* = 1).

*Note: Each Alternate Function for a given peripheral is independently selectable. Mixing functions assigned to AF1 and AF2 is supported if the necessary peripheral pins are all available.*

## 5.4 Input Modes and Pulldown/Pullup Strength Selection

Each GPIO pin supports selection between high-impedance input mode, weak or strong pullup mode, or weak or strong pulldown mode when the pin is set for I/O mode (*GPIOn_EN[pin]* = 1) and the input register is enabled (*GPIOn_IN_EN[pin]* = 1). The input mode selection uses *GPIOn_PDPU_SEL1[pin]* and *GPIOn_PDPU_SEL0[pin]* to select the input mode. The selection between weak or strong pulldown/pullup is set using the *GPIOn_PSSEL[pin]*. *Table 5-4* details the combination of these three bits and the resulting input modes available.

*Table 5-4: Input Mode Configuration*

| GPIOn_PDPU_SEL1*[pin]* | GPIOn_PDPU_SEL0*[pin]* | GPIOn_PSSEL*[pin]* | Input Mode |
|---|---|---|---|
| 0 | 0 | NA | High-impedance |
| 0 | 1 | 0 | Weak Pull-up |
| 0 | 1 | 1 | Strong Pull-up |
| 1 | 0 | 0 | Weak Pull-down |
| 1 | 0 | 1 | Strong Pull-down |
| 1 | 1 | NA | Reserved for Future Use |

## 5.5 Configuring GPIO (External) Interrupts

Each GPIO supports external interrupt events when the GPIO is configured for I/O mode and the input mode is enabled. If the GPIO is configured as a peripheral alternate function, the interrupts are peripheral controlled. GPIO interrupts can be enabled for any number of GPIO on each GPIO port. The following procedure details the steps for enabling Active mode interrupt events for a GPIO pin:

1. Disable interrupts by setting the *GPIOn_INT_EN[pin]* field to 0. This 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[pin]* bit.
3. Set *GPIOn_INT_MODE[pin]* to select either level (0) or edge triggered (1) interrupts.
4. For level triggered interrupts, the interrupt triggers on an input high (*GPIOn_INT_POL[pin]* = 0) or input low level.
5. For edge triggered interrupts, the interrupt triggers on a transition from low to high(*GPIOn_INT_POL[pin]* = 0) or high to low (*GPIOn_INT_POL[pin]* = 1).
6. Optionally set *GPIOn_INT_DUAL_EDGE[pin]* to 1 to trigger on both the rising and falling edges of the input signal.
7. Set *GPIOn_INT_EN[pin]* to 1 to enable the interrupt for the pin.

### 5.5.1 *GPIO Interrupt Vectors*

Each GPIO port is assigned a dedicated interrupt vector as shown in the following table.

*Table 5-5: GPIO Port Interrupt Vector Mapping*

| GPIO Interrupt Source | GPIO Interrupt Status Register | Device Specific Interrupt Vector Number | GPIO Interrupt Vector |
|---|---|---|---|
| GPIO0[0:31] | GPIO0_INT_STAT | 40 | GPIO0_IRQHandler |
| GPIO1[0:31] | GPIO1_INT_STAT | 41 | GPIO1_IRQHandler |
| GPIO2[0:31] | GPIO2_INT_STAT | 42 | GPIO2_IRQHandler |
| GPIO3[0:9] | GPIO3_INT_STAT | 43 | GPIO3_IRQHandler |

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. Signal an end-of-interrupt to the interrupt controller by writing to the End-of-Interrupt register.
5. Return from the interrupt vector handler.

### 5.5.2 *Using GPIO for Wakeup from Low Power Modes*

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

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

*Table 5-6: GPIO Wakeup Interrupt Vector*

| GPIO Wake Interrupt Source | GPIO Wake Interrupt Status Register | Device Specific Interrupt Vector Number | GPIO Wakeup Interrupt Vector |
|---|---|---|---|
| GPIO0[0:31] | GPIO0_INT_STAT | 70 | GPIOWAKE_IRQHandler |
| GPIO1[0:31] | GPIO1_INT_STAT | 70 | GPIOWAKE_IRQHandler |
| GPIO2[0:31] | GPIO2_INT_STAT | 70 | GPIOWAKE_IRQHandler |
| GPIO3[0:9] | GPIO3_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. Set the polarity (rising or falling edge) by writing to the *GPIOn_INT_POL[pin]* field. The wakeup function relies on the rising and falling edge detectors, which operate asynchronously and do not require an active system clock. You can also use the dual-edge mode to accomplish this.
2. Clear pending interrupt flags by writing to *GPIOn_INT_CLR[pin]*.
3. Activate the GPIO wakeup function by writing 1 to *GPIOn_WAKE_EN[pin]*.
4. Configure the power manager to use the GPIO as a wakeup source by setting the *GCR_PMR.gpiowken* field to 1.

## 5.6 GPIO Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the GPIOn Base Peripheral Offset Address.

*Table 5-7: GPIO Registers*

| Offset | Register Name | Description |
|---|---|---|
| [0x0000] | *GPIOn_EN* | Enable register |
| [0x0004] | *GPIOn_EN_SET* | Atomic set for *GPIOn_EN* register |
| [0x0008] | *GPIOn_EN_CLR* | Atomic clear for *GPIOn_EN* register |
| [0x000C] | *GPIOn_OUT_EN* | GPIOn Output enable register |
| [0x0010] | *GPIOn_OUT_EN_SET* | Atomic set for *GPIOn_OUT_EN* register |
| [0x0014] | *GPIOn_OUT_EN_CLR* | Atomic clear for *GPIOn_OUT_EN* register |

| Offset | Register Name | Description |
|---|---|---|
| [0x0018] | *GPIOn_OUT* | GPIOn Output register |
| [0x001C] | *GPIOn_OUT_SET* | Atomic set for *GPIOn_OUT* register |
| [0x0020] | *GPIOn_OUT_CLR* | Atomic clear for *GPIOn_OUT* register |
| [0x0024] | *GPIOn_IN* | GPIOn Input register |
| [0x0028] | *GPIOn_INT_MODE* | GPIOn Interrupt mode register |
| [0x002C] | *GPIOn_INT_POL* | GPIOn Interrupt polarity register |
| [0x0030] | *GPIOn_IN_EN* | GPIOn Input enable register |
| [0x0034] | *GPIOn_INT_EN* | GPIOn Interrupt enable register |
| [0x0038] | *GPIOn_INT_EN_SET* | Atomic set for *GPIOn_INT_EN* register |
| [0x003C] | *GPIOn_INT_EN_CLR* | Atomic clear for *GPIOn_INT_EN* register |
| [0x0040] | *GPIOn_INT_STAT* | GPIOn Interrupt status register |
| [0x0048] | *GPIOn_INT_CLR* | Atomic clear for *GPIOn_INT_STAT* register |
| [0x004C] | *GPIOn_WAKE_EN* | GPIOn Wake from DEEPSLEEP enable register |
| [0x0050] | *GPIOn_WAKE_EN_SET* | Atomic set for *GPIOn_WAKE_EN* register |
| [0x0054] | *GPIOn_WAKE_EN_CLR* | Atomic clear for *GPIOn_WAKE_EN* register |
| [0x005C] | *GPIOn_INT_DUAL_EDGE* | GPIOn Interrupt dual edge register |
| [0x0060] | *GPIOn_PDPU_SEL0* | GPIOn Input mode selection register 0 |
| [0x0064] | *GPIOn_PDPU_SEL1* | GPIOn Input mode selection register 1 |
| [0x0068] | *GPIOn_AF_SEL* | GPIOn Alternate function select register |
| [0x006C] | *GPIOn_AF_SEL_SET* | Atomic set for *GPIOn_AF_SEL* register |
| [0x0070] | *GPIOn_AF_SEL_CLR* | Atomic clear for *GPIOn_AF_SEL* register |
| [0x00B0] | *GPIOn_DS_SEL0* | GPIOn Drive strength selection register 0 |
| [0x00B4] | *GPIOn_DS_SEL1* | GPIOn Drive strength selection register 1 |
| [0x00B8] | *GPIOn_PSSEL* | GPIOn Pulldown/Pullup strength select register |
| [0x00C0] | *GPIO0_VSSEL* | GPIO0 Voltage select register |
| [0x00C0] | *GPIO1_VSSEL* | GPIO1 Voltage select register |
| [0x00C0] | *GPIO2_VSSEL* | GPIO2 Voltage select register |
| [0x00C0] | *GPIO3_VSSEL* | GPIO3 Voltage select register |

## 5.7 GPIO Register Details

*Table 5-8: GPIO Port 0 Enable Register*

| GPIO Port 0 Enable Register | | | GPIO0_EN | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:30 | - | R/W | 1 | **GPIO Enable**<br>Each bit in this register controls the GPIO enable and alternate function enable for a pin on this GPIO port. Writing a bit to 0 enables the alternate function selected in the *GPIOn_AF_SEL* register for the pin.<br>0: Alternate function enabled<br>1: GPIO enabled (default)<br>*Note: This bit's setting does not affect input and interrupt functionality of the associated pin.* |
| 29 | | R/W | 0 | **GPIO Enable**<br>If JTAG debug is available on the part, this pin defaults to the JTAG alternate function (TCK/SWCLK) on all forms of reset.<br>0: Alternate function JTAG TCK/SWCLK enabled (default).<br>1: GPIO enabled |
| 28 | - | R/W | 0 | **GPIO Enable**<br>If JTAG debug is available on the part, this pin defaults to the JTAG alternate function (TMS/SWDIO) on all forms of reset.<br>0: Alternate function JTAG TMS/SWDIO enabled (default).<br>1: GPIO enabled |
| 27 | - | R/W | 0 | **GPIO Enable**<br>If JTAG debug is available on the part, this pin defaults to the JTAG alternate function (TDO) on all forms of reset.<br>0: Alternate function JTAG TDO enabled (default).<br>1: GPIO enabled |
| 26 | - | R/W | 0 | **GPIO Enable**<br>If JTAG debug is available on the part, this pin defaults to the JTAG alternate function (TDI) on all forms of reset.<br>0: Alternate function JTAG TDI enabled (default).<br>1: GPIO enabled |
| 25:0 | - | R/W | 1 | **GPIO Enable**<br>Each bit in this register controls the GPIO enable and alternate function enable for a pin on this GPIO port. Writing a bit to 0 enables the alternate function selected in the *GPIOn_AF_SEL* register for the pin.<br>0: Alternate function enabled<br>1: GPIO enabled (default)<br>*Note: This bit's setting does not affect input and interrupt functionality of the associated pin.* |

*Table 5-9: GPIO Port 1 to Port 3 Enable Registers*

| GPIO Port Enable Register | | | GPIOn_EN | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 1 | **GPIO Enable**<br>Each bit in this register controls the GPIO enable and alternate function enable for a pin on this GPIO port. Writing a bit to 0 enables the alternate function selected in the *GPIOn_AF_SEL* register for the pin.<br>0: Alternate function enabled<br>1: GPIO enabled (default)<br>*Note: This bit's setting does not affect input and interrupt functionality of the associated pin.* |

*Table 5-10: GPIO Port 0 to Port 3 Enable Atomic Set Registers*

| GPIO Port Enable Atomic Set Register | | | GPIOn_EN_SET | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Enable Atomic Set**<br>Writing 1 to one or more bits in this register sets the corresponding bits in the *GPIOn_EN* register.<br>0: No effect on corresponding bit in *GPIOn_EN* register.<br>1: Set corresponding bit in *GPIOn_EN* register. |

*Table 5-11: GPIO Port 0 to Port 3 Enable Atomic Clear Registers*

| GPIO Port Enable Atomic Clear Register | | | GPIOn_EN_CLR | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Enable Atomic Clear**<br>Writing 1 to one or more bits in this register clears the corresponding bits in the GPIO0_EN register.<br>1: No effect on corresponding bit in *GPIOn_EN* register.<br>0: Clear corresponding bit in *GPIOn_EN* register. |

*Table 5-12: GPIO Port 0 Output Enable Register*

| GPIO Port 0 Output Enable Register | | | GPIO0_OUT_EN | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:28 | - | R/W | 0 | **GPIO Output Enable**<br>Set bit to 1 to enable the output driver for the corresponding GPIO pin. This bit is ignored if the corresponding bit in the *GPIOn_EN* register is not set.<br>0: Pin is set to input mode; output driver disabled (default).<br>1: Pin is set to output mode<br>*Note: This bit is ignored if the corresponding bit position in the GPIOn_EN register is not set.* |

| GPIO Port 0 Output Enable Register | | | GPIO0_OUT_EN | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 27 | - | R/W | 1 | **GPIO Output Enable**<br>This pin defaults to output enabled if JTAG debug is available on the part and maps to alternate function JTAG TDO.<br>0: Pin is set to input mode; output driver disabled.<br>1: Pin is set to output mode (default). |
| 26:0 | - | R/W | 0 | **GPIO Output Enable**<br>Set bit to 1 to enable the output driver for the corresponding GPIO pin. This bit is ignored if the corresponding bit in the GPIO0_EN register is not set.<br>0: Pin is set to input mode; output driver disabled (default).<br>1: Pin is set to output mode<br>*Note: This bit is ignored if the corresponding bit position in the GPIOn_EN register is not set.* |

*Table 5-13: GPIO Port 1 to Port 3 Output Enable Registers*

| GPIO Port Output Enable Register | | | GPIOn_OUT_EN | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Output Enable**<br>Set bit to 1 to enable the output driver for the corresponding GPIO pin. This bit is ignored if the corresponding bit in the GPIOn_EN register is not set.<br>0: Pin is set to input mode; output driver disabled.<br>1: Pin is set to output mode<br>*Note: This bit is ignored if the corresponding bit position in the GPIOn_EN register is not set.* |

*Table 5-14: GPIO Port 0 to Port 3 Output Enable Atomic Set Registers*

| GPIO Port Output Enable Atomic Set Register | | | GPIOn_OUT_EN_SET | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Output Enable Atomic Set**<br>Writing 1 to one or more bits in this register sets the corresponding bits in the GPIOn_OUT_EN register.<br>0: No effect on corresponding bit in GPIOn_OUT_EN register.<br>1: Set corresponding bit in GPIOn_OUT_EN register.<br>*Note: This bit is ignored if the corresponding bit position in the GPIOn_EN register is not set.* |

*Table 5-15: GPIO Port 0 to Port 3 Output Enable Atomic Clear Registers*

| GPIO Port Output Enable Atomic Clear Register | | GPIOn_OUT_EN_CLR | | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Output Enable Atomic Clear**<br>Writing 1 to one or more bits in this register clears the corresponding bits in the GPIOn_OUT_EN register.<br>0: No effect on corresponding bit in GPIOn_OUT_EN register.<br>1: Clear corresponding bit in *GPIOn_OUT_EN* register.<br>*Note: This bit is ignored if the corresponding bit position in the GPIOn_EN register is not set.* |

*Table 5-16: GPIO Port 0 to Port 3 Output Registers*

| GPIO Port Output Register | | GPIOn_OUT | | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Output**<br>Set the corresponding output pin high or low.<br>0: Drive the corresponding output pin low (logic 0).<br>1: Drive the corresponding output pin high (logic 1).<br>*Note: This bit is ignored if the corresponding bit position in the GPIOn_EN register and GPIOn_OUT_EN register is not set.* |

*Table 5-17: GPIO Port 0 to Port 3 Output Atomic Set Registers*

| GPIO Port Output Atomic Set Register | | GPIOn_OUT_SET | | [0x001C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Output Atomic Set**<br>Writing 1 to one or more bits in this register sets the corresponding bits in the *GPIOn_OUT* register.<br>0: No effect on corresponding bit in *GPIOn_OUT* register.<br>1: Set corresponding bit in *GPIOn_OUT* register.<br>*Note: This bit is ignored if the corresponding bit position in the GPIOn_EN register and GPIOn_OUT_EN register is not set.* |

*Table 5-18: GPIO Port 0 to Port 3 Output Atomic Clear Registers*

| GPIO Port Output Atomic Clear Register | | GPIOn_OUT_CLR | | [0x0020] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Output Atomic Clear**<br>Writing 1 to one or more bits in this register clears the corresponding bits in the *GPIOn_OUT* register.<br>0: No effect on corresponding bit in *GPIOn_OUT* register.<br>1: Clear corresponding bit in *GPIOn_OUT* register.<br>*Note: This bit is ignored if the corresponding bit position in the GPIOn_EN register and GPIOn_OUT_EN register is not set.* |

*Table 5-19: GPIO Port 0 to Port 3 Input Registers*

| GPIO Port Input Register | | GPIOn_IN | | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | RO | - | **GPIO Input**<br>Read the state of the corresponding input pin. The corresponding pin must be enabled as an input in the *GPIOn_IN_EN* register or the input does not return the value of the pin. The input state is always readable for a pin regardless of the pin's configuration as an output or alternate function.<br>0: Input pin low (logic 0).<br>1: Input pin high (logic 1).<br>*Note: This bit is ignored if the corresponding bit position in the GPIOn_EN register and GPIOn_OUT_EN register is not set.* |

*Table 5-20: GPIO Port 0 to Port 3 Interrupt Mode Registers*

| GPIO Port Interrupt Mode Register | | GPIOn_INT_MODE | | [0x0028] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Interrupt Mode**<br>Interrupt mode selection bit for the corresponding GPIO pin.<br>0: Level triggered interrupt for corresponding GPIO pin.<br>1: Edge triggered interrupt for corresponding GPIO pin.<br>*Note: This bit has no effect unless the corresponding bit in the GPIOn_INT_EN register is set.* |

*Table 5-21: GPIO Port 0 to Port 3 Interrupt Polarity Registers*

| GPIO Port Interrupt Polarity Register | | GPIOn_INT_POL | | [0x002C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Interrupt Polarity**<br>Interrupt polarity selection bit for the corresponding GPIO pin.<br>Level triggered mode (*GPIOn_INT_MODE* = 0):<br>0: Input low (logic 0) triggers interrupt.<br>1: Input high (logic 1) triggers interrupt.<br>Edge triggered mode (*GPIOn_INT_MODE* = 1):<br>0: Falling edge triggers interrupt<br>1: Rising edge triggers interrupt.<br>*Note: This bit has no effect unless the corresponding bit in the GPIOn_INT_EN register is set.* |

*Table 5-22: GPIO Port 0 to Port 3 Input Enable Registers*

| GPIO Port Input Enable Register | | | GPIOn_IN_EN | [0x0030] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 1 | **GPIO Input Enable**<br>Connects the corresponding input pad to the specified input pin for reading the pin state using the *GPIOn_IN* register.<br>0: Input not connected.<br>1: Input pin connected to the pad for reading via *GPIOn_IN* register. |

*Table 5-23: GPIO Port 0 to Port 3 Interrupt Enable Registers*

| GPIO Port Interrupt Enable Register | | | GPIOn_INT_EN | [0x0034] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Interrupt Enable**<br>Enable or Disable the interrupt for the corresponding GPIO pin.<br>0: GPIO interrupt disabled.<br>1: GPIO interrupt enabled.<br>*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 5-24: GPIO Port 0 to Port 3 Interrupt Enable Atomic Set Registers*

| GPIO Port Interrupt Enable Atomic Set Register | | | GPIOn_INT_EN_SET | [0x0038] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Interrupt Enable Atomic Set**<br>Writing 1 to one or more bits in this register sets the corresponding bits in the *GPIOn_INT_EN* register.<br>0: No effect on corresponding bit in *GPIOn_INT_EN* register.<br>1: Set corresponding bit in *GPIOn_INT_EN* register. |

*Table 5-25: GPIO Port 0 to Port 3 Interrupt Enable Atomic Clear Registers*

| GPIO Port Interrupt Enable Atomic Clear Register | | | GPIOn_INT_EN_CLR | [0x003C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Interrupt Enable Atomic Clear**<br>Writing 1 to one or more bits in this register clears the corresponding bits in the GPIOn_INT_EN register.<br>1: No effect on corresponding bit in *GPIOn_INT_EN* register.<br>0: Clear corresponding bit in *GPIOn_INT_EN* register. |

*Table 5-26: GPIO Port 0 to Port 3 Interrupt Status Registers*

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

*Table 5-27: GPIO Port 0 to Port 3 Interrupt Clear Registers*

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

*Table 5-28: GPIO Port 0 to Port 3 Wakeup Enable Registers*

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

*Table 5-29: GPIO Port 0 to Port 3 Wakeup Enable Atomic Set Registers*

| GPIO Port Wakeup Enable Atomic Set Register | | | GPIOn_WAKE_EN_SET | [0x0050] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Wakeup Enable Atomic Set**<br>Writing 1 to one or more bits in this register sets the corresponding bits in the *GPIOn_WAKE_EN* register.<br>0: No effect on corresponding bit in *GPIOn_WAKE_EN* register.<br>1: Set corresponding bit in *GPIOn_WAKE_EN* register. |

*Table 5-30: GPIO Port 0 to Port 3 Wakeup Enable Clear Registers*

| GPIO Port Wakeup Enable Atomic Clear Register | | | GPIOn_WAKE_EN_CLR | [0x0054] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Wakeup Enable Atomic Clear**<br>Writing 1 to one or more bits in this register clears the corresponding bits in the *GPIOn_WAKE_EN* register.<br>1: No effect on corresponding bit in *GPIOn_WAKE_EN* register.<br>0: Clear corresponding bit in *GPIOn_WAKE_EN* register. |

*Table 5-31: GPIO Port 0 to Port 3 Interrupt Dual Edge Mode Registers*

| GPIO Port Interrupt Dual Edge Mode Register | | | GPIOn_INT_DUAL_EDGE | [0x005C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Interrupt Dual-Edge Mode Select**<br>Setting this bit selects dual edge mode (rising and falling edge triggered) interrupts if the associated *GPIOn_INT_MODE* bit is set to edge triggered. When dual edge mode is set, and the interrupt mode is edge-triggered, the associated polarity (*GPIOn_INT_POL*) setting has no effect.<br>0: No effect on interrupt generation.<br>1: Enable dual edge mode interrupts. |

*Table 5-32: GPIO Port 0 to Port 3 Pullup Pulldown Selection 0 Registers*

| GPIO Port Pullup Pulldown Selection 0 Register | | | GPIOn_PDPU_SEL0 | [0x0060] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Pullup Pulldown Selection 0**<br>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 descripted in detail in Input Modes and Pulldown/Pullup Strength Selection. |

*Table 5-33: GPIO Port 0 to Port 3 Pullup Pulldown Selection 1 Registers*

| GPIO Port Pullup Pulldown Selection 1 Register | | | GPIOn_PDPU_SEL1 | [0x0064] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Pullup Pulldown Selection 1**<br>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 descripted in detail in the section *Input Modes and Pulldown/Pullup Strength Selection*. |

*Table 5-34: GPIO Port 0 to Port 3 Alternate Function Select Registers*

| GPIO Port AF Select Register | | | GPIOn_AF_SEL | [0x0068] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Alternate Select Function**<br>When the *GPIOn_EN[pin]* is set to 0 (Alternate Function Enabled), this bit selects either Alternate Function 1 or Alternate Function 2 for the pin. Refer to *Table 5-1* and *Table 5-3* above for the package specific alternate functions (AF1 and AF2) available.<br>0: Alternate Function 1 selected.<br>1: Alternate Function 2 selected. |

*Table 5-35: GPIO Port 0 to Port 3 Alternate Function Select Atomic Set Registers*

| GPIO Port AF Select Atomic Set Register | | | GPIOn_AF_SEL_SET | [0x006C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Alternate Function Select Atomic Set**<br>Writing 1 to one or more bits in this register sets the corresponding bits in the *GPIOn_AF_SEL* register.<br>0: No effect on corresponding bit in *GPIOn_AF_SEL* register.<br>1: Set corresponding bit in *GPIOn_AF_SEL* register. |

*Table 5-36: GPIO Port 0 to Port 3 Alternate Function Select Clear Registers*

| GPIO Port AF Select Atomic Clear Register | | | GPIOn_AF_SEL_CLR | [0x0070] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Alternate Function Select Atomic Clear**<br>Writing 1 to one or more bits in this register clears the corresponding bits in the *GPIOn_AF_SEL* register.<br>1: No effect on corresponding bit in *GPIOn_AF_SEL* register.<br>0: Clear corresponding bit in *GPIOn_AF_SEL* register. |

*Table 5-37: GPIO Port 0 to Port 3 Drive Strength Selection 0 Registers*

| GPIO Port Drive Strength Selection 0 Register | | | GPIOn_DS_SEL0 | [0x00B0] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Drive Strength Selection 0**<br>The output drive strength supports four modes. The mode selection is set using the combination of the *GPIOn_DS_SEL1* and *GPIOn_DS_SEL0* bits for the associated GPIO pin. |

| GPIOn_DS_SEL1 | GPIOn_DS_SEL0 | Drive Strength |
|---|---|---|
| 0 | 0 | 1x |
| 0 | 1 | 2x |
| 1 | 0 | 4x |
| 1 | 1 | 8x |

*Table 5-38: GPIO Port 0 to Port 3 Drive Strength Selection 1 Registers*

| GPIO Port Drive Strength Selection 1 Register | | | GPIOn_DS_SEL1 | [0x00B4] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Drive Strength Selection 1**<br>Refer to *GPIO Drive Strength Selection* for details. |

*Table 5-39: GPIO Port 0 to Port 3 Pulldown/Pullup Select Registers*

| GPIO Port Pulldown/Pullup Select Register | | | GPIOn_PSSEL | [0x00B8] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Pulldown/Pullup Select**<br>Selects a weak Pulldown/Pullup resistor if set to 0 and a strong Pulldown/Pullup resistor if set to 1.<br>0: Weak (1MΩ) Pulldown/Pullup resistor for input pin.<br>1: Strong (25KΩ) Pulldown/Pullup resistor for input pin.<br>*Note: Refer to the MAX32650–MAX32652 data sheet for specific electrical characteristics of the Pulldown/Pullup resistances.* |

*Table 5-40: GPIO Port 0 Supply Voltage Select Register*

| GPIO Port 0 Voltage Select Register | | | GPIO0_VSSEL | [0x00C0] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Supply Voltage Select**<br>0: $V_{DDIO}$ set as the pin's supply.<br>1: $V_{DDIOH}$ set as the pin's supply. |

*Table 5-41: GPIO Port 1 Supply Voltage Select Register*

| GPIO Port 1 Voltage Select Register | | | GPIO1_VSSEL | [0x00C0] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:22 | - | R/W | 0 | **GPIO Supply Voltage Select**<br>Select the Supply Voltage for the pin. Set to 1 to select $V_{DDIOH}$.<br>0: $V_{DDIO}$ set as the pin's supply.<br>1: $V_{DDIOH}$ set as the pin's supply. |
| 21:18 | - | RO | 0 | **GPIO Supply Voltage Select**<br>GPIO P1[21:18] pins are tied to the $V_{DDIO}$ supply and cannot be changed to $V_{DDIOH}$.<br>0: $V_{DDIO}$ is always used for the pin's supply.<br>1: Invalid |
| 17 | - | R/W | 0 | **GPIO Supply Voltage Select**<br>Select the Supply Voltage for the pin. Set to 1 to select $V_{DDIOH}$.<br>0: $V_{DDIO}$ set as the pin's supply.<br>1: $V_{DDIOH}$ set as the pin's supply. |
| 16:11 | - | RO | 0 | **GPIO Supply Voltage Select**<br>GPIO1[16:11] are tied to the $V_{DDIO}$ supply and cannot be set to $V_{DDIOH}$ for the supply selection.<br>0: $V_{DDIO}$ is always used for the pin's supply.<br>1: Invalid |
| 10:0 | - | R/W | 0 | **GPIO Supply Voltage Select**<br>Select the Supply Voltage for the pin. Set to 1 to select $V_{DDIOH}$.<br>0: $V_{DDIO}$ set as the pin's supply.<br>1: $V_{DDIOH}$ set as the pin's supply. |

*Table 5-42: GPIO Port 2 Supply Voltage Select Register*

| GPIO Port 2 Voltage Select Register | | | GPIO2_VSSEL | [0x00C0] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | R/W | 0 | **GPIO Supply Voltage Select**<br>Select the Supply Voltage for the pin. Set to 1 to select $V_{DDIOH}$.<br>0: $V_{DDIO}$ set as the pin's supply.<br>1: $V_{DDIOH}$ set as the pin's supply. |

*Table 5-43: GPIO Port 3 Supply Voltage Select Register*

| GPIO Port 3 Supply Voltage Select Register | | | GPIO3_VSSEL | [0x00C0] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:10 | - | R | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 9:1 | - | R/W | 0 | **GPIO Supply Voltage Select**<br>Select the Supply Voltage for the pin. Set to 1 to select $V_{DDIOH}$.<br>0: $V_{DDIO}$ set as the pin's supply.<br>1: $V_{DDIOH}$ set as the pin's supply. |

| GPIO Port 3 Supply Voltage Select Register | | | GPIO3_VSSEL | [0x00C0] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | - | RO | 0 | **GPIO Supply Voltage Select**<br>GPIO3[0] (P3.0) is tied to the $V_{DDIO}$ supply and cannot be set to $V_{DDIOH}$ for the supply selection.<br>0: $V_{DDIO}$ is always used for the pin's supply.<br>1: Invalid |

# 6 Flash Controller

The MAX32650–MAX32652 Flash Controller is a peripheral that manages read, write, and erase accesses to the internal flash.

*Features*

- Up to 3 MB total internal flash memory
  - 192 pages
  - 16,384 bytes per page
  - 4096 words by 128 bits per page
- 128-bit data reads
- 32-bit or 128-bit write support
- Page erase and mass erase support
- Write Protection

## 6.1 Overview

The MAX32650–MAX32652 contains 3MB of internal flash memory for storing user application and data. The internal flash memory is programmable via the JTAG debug interface (in-system) or directly with user application code (in-application).

The flash is organized as an array of pages. Each page is 4,096 words by 128 bits, or 16,384 bytes per page. *Table 6-1*, *below*, shows the start address and end address for the internal flash memory. The internal flash memory is mapped with a start address of 0x1000 0000 and an end address of 0x102F FFFF for a total of 3MB.

*Table 6-1: Internal Flash Memory Organization*

| Page Number | Size | Start Address | End Address |
|---|---|---|---|
| 1 | 16,384 Bytes | 0x1000 0000 | 0x1000 3FFF |
| 2 | 16,384 Bytes | 0x1000 4000 | 0x1000 7FFF |
| 3 | 16,384 Bytes | 0x1000 8000 | 0x1000 BFFF |
| 4 | 16,384 Bytes | 0x1000 C000 | 0x1000 FFFF |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| . | . | . | . |
| 191 | 16,384 Bytes | 0x102F 8000 | 0x102F BFFF |
| 192 | 16,384 Bytes | 0x102F C000 | 0x102F 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.

### 6.2.1 *Clock Configuration*

The Flash Controller requires a 1MHz peripheral clock for operation. The input clock to the Flash Controller block is the system clock, $f_{SYSCLK}$. Use the Flash Controller clock divisor to generate $f_{FLCCLK} = 1MHz$, as shown in *Equation 10-2 below*. For the 120MHz Relaxation Oscillator as the system clock, the *FLC_CLKDIV.clkdiv* should be set to 120 (0xC0).

*Equation 6-1: Flash Controller Clock Frequency*

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

### 6.2.2 Lock Protection

The Flash Controller provides a locking mechanism to prevent accidental writes and erases. All writes and erase operations require the *FLC_CTRL.unlock* field be set to 0x2 prior to starting the operation. Writing any other value to this field, *FLC_CTRL.unlock*, results in the flash remaining locked.

*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, FLC_INTR.access_fail, to indicate an access violation occurred.*

### 6.2.3 Flash Write Width

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

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

*Table 6-2: Valid Addresses for 32-bit and 128-bit Internal Flash Writes*

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

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

1. If desired, enable flash controller interrupts by setting the *FLC_INTR.access_error_ie* and *FLC_INTR.done_ie* bits.
2. Set the write field, *FLC_CTRL.width*, as described in *Flash Write Width*.
3. Set the *FLC_ADDR* register to a valid target address. Reference *Table 6-2*.
4. Set the data register or registers.
   a. For 32-bit write width, set *FLC_DATA0* to the data to write.
   b. For 128-bit write width, set *FLC_DATA3*, *FLC_DATA2*, *FLC_DATA1*, and *FLC_DATA0* to the data to write. *FLC_DATA3* is the most significant word and *FLC_DATA0* is the least significant word.
5. Set *FLC_CTRL.unlock* to 0x2 to unlock the internal flash.
6. Read the *FLC_CTRL.busy* bit until it returns 0.
7. Start the flash write, set *FLC_CTRL.write* to 1 and this field is automatically cleared by the Flash Controller when the write operation is finished.
8. *FLC_INTR.done* is set by hardware when the write completes and if an error occurred, the *FLC_INTR.access_fail* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.

### 6.2.5 Page Erase

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

1. If desired, enable flash controller interrupts by setting the *FLC_INTR.access_error_ie* and *FLC_INTR.done_ie* bits.
2. Set the *FLC_ADDR* register to a page address to erase. *FLC_ADDR*[11:0] are ignored by the Flash Controller to ensure the address is page aligned. Refer to *Table 6-3* for the valid page aligned addresses for the internal flash memory.
3. Set *FLC_CTRL.unlock* to 0x2 to unlock the internal flash.
4. Read the *FLC_CTRL.busy* bit until it returns 0.
5. Set *FLC_CTRL.erase_code* to 0x55 for page erase.
6. Set *FLC_CTRL.page_erase* to 1 to start the page erase operation.
7. The *FLC_CTRL.busy* bit is set by the flash controller while the page erase is in progress and the *FLC_CTRL.page_erase* and *FLC_CTRL.busy* are cleared by the flash controller when the page erase is complete.
8. *FLC_INTR.done* is set by hardware when the page erase completes and if an error occurred, the *FLC_INTR.access_fail* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.

*Table 6-3: Page Boundary Address Range for Page Erase Operations*

| | FLC_ADDR[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **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** |
| **Page Aligned Address** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | x | x | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 6.2.6 Mass Erase

Mass erase clears the internal flash memory. This operation requires the JTAG debug port to be enabled to perform the operation. If the JTAG debug port is not enabled a mass erase operation cannot be performed. Perform the following steps to mass erase the internal flash:

1. Set *FLC_CTRL.unlock* to 0x2 to unlock the internal flash.
2. Read the *FLC_CTRL.busy* bit until it returns 0.
3. Set *FLC_CTRL.erase_code* to 0xAA for mass erase.
4. Set *FLC_CTRL.mass_erase* to 1 to start the mass erase operation.
5. The *FLC_CTRL.busy* bit is set by the flash controller while the mass erase is in progress and the *FLC_CTRL.mass_erase* and *FLC_CTRL.busy* are cleared by the flash controller when the mass erase is complete.
6. *FLC_INTR.done* is set by the flash controller when the mass erase completes and if an error occurred, the *FLC_INTR.access_fail* flag is set. These bits generate a flash IRQ if the interrupt enable bits are set.

*Note: Mass erase requires the JTAG debug port to be enabled, if the JTAG debug port is disabled on the device an access fail error is generated (*FLC_INTR.access_fail *= 1).*

## 6.3 Flash Controller Registers

Refer to *Table 2-1* for the SDHC Base Peripheral Address.

*Table 6-4: Flash Controller Registers*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *FLC_ADDR* | R/W | Flash Controller Address Pointer Register |
| [0x0004] | *FLC_CLKDIV* | R/W | Flash Controller Clock Divisor Register |
| [0x0008] | *FLC_CTRL* | R/W | Flash Controller Control Register |

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0024] | *FLC_INTR* | R/W | Flash Controller Interrupt Register |
| [0x0030] | *FLC_DATA0* | R/W | Flash Controller Data Register 0 |
| [0x0034] | *FLC_DATA1* | R/W | Flash Controller Data Register 1 |
| [0x0038] | *FLC_DATA2* | R/W | Flash Controller Data Register 2 |
| [0x003C] | *FLC_DATA3* | R/W | Flash Controller Data Register 3 |

## 6.4 Flash Controller Register Details

*Table 6-3. Flash Controller Address Pointer Register*

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

*Table 6-4. Flash Controller Clock Divisor Register*

| Flash Controller Clock Divisor Register | | | FLC_CLKDIV | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:8 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 7:0 | clkdiv | R/W | 0xC0 | **Flash Controller Clock Divisor**<br>The system clock is divided by the value in this field to generate the FLC peripheral clock, $f_{FLCCLK}$. The FLC peripheral clock must equal 1MHz. The default on all forms of reset is 120 (0xC0), resulting in $f_{FLCCLK}$ = 1MHz. |

*Table 6-5. Flash Controller Control Register*

| Flash Controller Control Register | | | FLC_CTRL | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:28 | unlock_code | R/W | 0 | **Flash Unlock**<br>Write the unlock code, 0x2, prior to any flash write or erase operation to unlock the Flash. Writing any other value to this field locks the internal flash.<br>0x2: Flash unlock code |
| 27:25 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |

| Flash Controller Control Register | | FLC_CTRL | | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 24 | busy | RO | 0 | **Flash Busy Flag**<br>When this field is set, writes to all flash registers except the *FLC_INTR* register are ignored by the Flash Controller.<br>*Note: If the Flash Controller is busy (FLC_CTRL.busy = 1), reads, writes and erase operations are not allowed and result in an access failure (FLC_CTRL.access_fail = 1).*<br>0: Flash idle<br>1: Flash busy |
| 23:16 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 15:8 | erase_code | R/W | 0 | **Erase Code**<br>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.<br>This field is automatically cleared after the erase operation is complete.<br>0x00: Erase disabled.<br>0x55: Page erase code.<br>0xAA: Enable mass erase via the JTAG debug port. |
| 7:5 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 4 | width | R/W | 0 | **Data Width Select**<br>This field sets the data width of a write to the flash page. The Flash Controller supports either 32-bit writes, or 128-bit writes.<br>0: 128-bit transactions (*FLC_DATA3* - *FLC_DATA0*)<br>1: 32-bit transactions (*FLC_DATA0* only) |
| 3 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 2 | page_erase | R/W1O | 0 | **Page Erase**<br>Write a 1 to this field to initiate a page erase at the address in *FLC_ADDR.addr*. The flash must be unlocked prior to attempting a page erase, see *FLC_CTRL.unlock* for details.<br>The Flash Controller hardware clears this bit when a page erase operation is complete.<br>0: No page erase operation in process or page erase is complete.<br>1: Write a 1 to initiate a page erase. If this field reads 1, a page erase operation is in progress.<br>*Note: This field is protected and cannot be set to 0 by application code.* |
| 1 | mass_erase | R/W1O | 0 | **Mass Erase**<br>Write a 1 to this field to initiate a mass erase of the internal flash memory. The flash must be unlocked prior to attempting a mass erase, see *FLC_CTRL.unlock* for details.<br>The Flash Controller hardware clears this bit when the mass erase operation completes.<br>0: No operation<br>1: Initiate mass erase<br>*Note: This field is protected and cannot be set to 0 by application code.* |

| Flash Controller Control Register | | | FLC_CTRL | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | write | R/W1O | 0 | **Write**<br>If this field reads 0, no write operation is pending for the flash. To initiate a write operation, set this bit to 1 and the Flash Controller will write to the address set in the *FLC_ADDR* register.<br>0: No write operation in process or write operation complete.<br>1: Write 1 to initiate a write operation. If this field reads 1, a write operation is in progress.<br>*Note: This field is protected and cannot be set to 0 by application code.* |

*Table 6-5: Flash Controller Interrupt Register*

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

*Table 6-6: Flash Controller Data Register 0*

| Flash Controller Data Register 0 | | | FLC_DATA0 | [0x0030] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | data0 | R/W | 0 | **Flash Data 0**<br>Flash data for bits 31:0. |

*Table 6-7: Flash Controller Data Register 1*

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

*Table 6-8: Flash Controller Data Register 2*

| Flash Controller Data Register 2 | | | FLC_DATA2 | [0x0038] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | data2 | R/W | 0 | **Flash Data 1**<br>Flash data for bits 95:64 |

*Table 6-9: Flash Controller Data Register 3*

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

# 7 External Memory

## 7.1 Overview

External memory can be accessed via multiple interfaces. There a four external memory interfaces, three of which are backed by 16KB of cache:

- SPI Execute-in-Place FLASH (SPIXF)
  - 16KB Dedicated Cache
- SPI Execute-in-Place RAM (SPIXR)
  - 16KB cache multiplexed with the HyperBus/Xccela Bus interface
- HyperBus/Xccela Bus
  - 16KB cache multiplexed with the SPIXR interface
- SD/SDIO/SDHC/MMC

## 7.2 SPI Execute-in-Place Flash

The SPI Execute in Place Flash interface consists of a Master Controller block and Master block. The features of the SPIXF interface include:

- Up to 60MHz operation in mode 0 and 3
- Four wire mode for single-bit slave device communication
- Dual and Quad I/O supported
- Programmable SCK frequency and duty cycle
- SS assertion and de-assertion timing with respect to the leading and trailing SCK edge
- Configurable command, address, dummy, and data fields to support a variety of SPI flashes

The SPIXF Master Controller allows the CPU to transparently execute instructions stored in an external SPI flash. Instructions fetched using the SPIXF Master Controller are cached just like instructions fetched from internal program memory. You can also use the SPIXF Master Controller to access large amounts of external static data that would otherwise reside in internal data memory. This device provides support for a wide variety of read commands that balance the need for backward comparability and high performance.

The SPIXF Master Controller provides a highly-configurable, flexible, and efficient interface that supports the programming and configuration of external SPI flash devices.

Prior to using the SPI flash device, you must configure the SPIXF Master Controller interface.

To prevent disclosure of intellectual property, code and data can optionally be stored in external Flash in an encrypted form using the SPIXF Master Controller. Generation of the encrypted data can be done via user firmware or with the cryptographic accelerator. The SPIXF Master Controller can transparently decrypt this information in real time using the AES-128 algorithm in ECB mode.

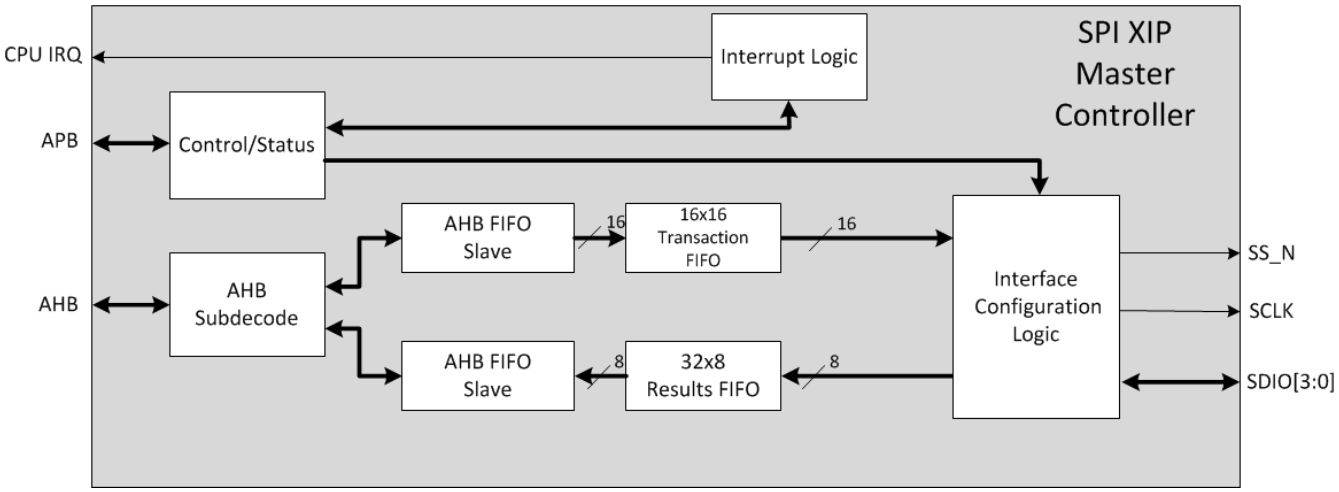
### 7.2.1 *SPIXF Master Controller*

The SPIXF Master Controller is used to program data to an external SPI flash and for configuring the external SPI flash interface.

The SPIXF Master Controller block shown in *Figure 7-1* consists of transmit and receive shift registers (supported by FIFOs) and a control unit. Communication and interface configuration are set up using the APB registers. It contains one 16×16 FIFO (Transaction FIFO) to support the transmit direction and one 32×8 FIFO (Results FIFO) to support the receive direction. These FIFOs are accessible to firmware using an AHB interface to support high-speed data transfers. New data is moved automatically from the Transmit FIFO into the shift register at the start of every new SPI transfer as long as there is data in the Transmit FIFO. At the end of every SPI transfer, data is moved from the shift register into the Receive FIFO. Status flags and interrupts are available to monitor the data levels in these FIFOs. You can use the Transaction FIFO to configure the SPI interface using transaction-header entries.

When a SPI transfer occurs, a multi-byte (selectable from 1 to 1024bytes) packet is shifted out if the Transaction FIFO has configured the device to transmit using the Transaction FIFO header entry. The most significant bit is sent first. If the Transaction FIFO configures the device to receive, the device receives data most significant bit first, and places each byte received into the Results FIFO.

*Figure 7-1. Simplified Block Diagram*




## 7.2.2 *SPI Pin Configuration*

The SPIXF Master Controller shares pins with the SPIXF Master so that the SPI flash is configured for code execution or data transfer. See the SPI Pins Configuration section of the SPIXF Master for more information.

### *7.2.2.1 Configuration Modes Overview*

Once the main SPIXF Master Controller clock is set up, the remainder of the configuration and operation for this block is mapped into three categories:

1. Static configuration: Performed during SPI initial setup, when the communicate port is disabled, or both.
   a. *SPIXFC_GEN_CTRL* register: SCK Feedback mode, enable Transmit and Receive FIFOs
   b. *SPIXFC_SS_POL* register: Slave Select signal polarity
   c. *SPIXFC_CFG* register: Active slave, SPI clock priority and phase that is, mode), clock and slave select timing.
2. Dynamic configuration: Configuration required to communicate with a specific slave device, which may take place while the communication port is enabled but the slave select is de-asserted.
   a. *SPIXFC_CFG* register: SPI page transfer size if using pages. See header information in *Table 7-1*.
3. Interrupt servicing: Status and control used by an application to efficiently service the SPI data transfer.
   a. *SPIXFC_FIFO_CTRL* register: Transaction and Results FIFO monitoring levels
   b. *SPIXFC_INT_FL* register: Interrupt flag bits
   c. *SPIXFC_INT_EN* register: Interrupt enable bits

#### 7.2.2.2 SPI Master Controller Transaction

Once the SPI master is configured to communicate to a specific slave, SPI transactions are initiated by writing to the SPI Transaction FIFO mapped into the AHB system address map at 0x400BC000. The FIFO is 16-bits wide and expects a 16-bit header followed by an optional payload padded out to a word boundary. If the transaction generates results data, this data is pushed into the SPI. Results FIFO is mapped into the AHB system address map at 0x400BC004. This FIFO is 8-bits wide and is zero-padded to a byte boundary at the completion of a SPI transaction.

The format of the header is shown in *Table 7-1*. The width field allows for initial communication to the external SPI flash in its default IO configuration and may be changed to improve throughput once the external SPI flash is reconfigured. The Size Units and Size fields allow for maximum flexibility in sending commands or programming data to the external SPI flash.

A complete access sequence to a SPI device is made up of one or more transactions. In some cases, the slave select signal remains asserted across several transactions. In other cases, the access sequence defined by the slave device might require de-assertion of the slave selection in the middle of the access sequence. In general, any part of the access sequence that requires a change in direction, width, or timing, requires another transaction. Interrupt logic is provided to allow efficient servicing of the SPI Master functionality by firmware.

Interrupts are grouped into two categories:

1. Keeping the Transaction FIFO full.
2. Keeping the Results FIFO empty.

Programmable levels in the FIFO allow interrupt events to issue if the Transaction FIFO falls below a certain level, or if the results FIFO fills above a certain level.

*Table 7-1: SPI Header Format*

| Name | Bits | Description | Settings |
|---|---|---|---|
| Direction | 1:0 | Defines direction of information transfer. For headers that do not define a transmission (that is, direction = None or Rx), no payload is required. Conversely, headers that do not define a reception (that is,. direction = None or Tx), result in no data pushing into the Results FIFO. | 0: None<br>1: Tx<br>2: Rx<br>3: Both |
| Size Units | 3:2 | Defines units to use when interpreting the size field. | 0: Bits Note: Bit transactions are available only for Tx (that is, direction = 1 transactions).<br>1: Bytes<br>2: Pages (See the *SPIXFC_CFG.pgsz* field for page size definition) |
| Size | 8:4 | Size of transaction in terms of units. | 0: 32<br>1: 1<br>2: 2<br>…<br>15: 15 |
| Width | 10:9 | Number of SDIO I/O to use for the transaction. This has no effect on the size of the transaction, just the time required to complete it.<br>Note: Dual and Quad I/O support are parameterized features. If an unimplemented width is defined, operation reverts to a 1-bit wide mode. | 0: Single I/O mode<br>1: Dual I/O mode<br>2: Quad I/O mode<br>3: Invalid |
| RFU | 12:11 | ***Reserved for Future Use.*** This header field should always be set to 0b00. | |

| Name | Bits | Description | Settings |
|---|---|---|---|
| De-assert SS | 13 | When asserted, de-assert Slave Select at the completion of this transaction. | |
| Header Type | 15:14 | | Must be 00 |

### 7.2.2.3 Sample SPIXF Master Controller Example

Here is an example how to set up the Master Controller:

1. Configure the SPIXF Master Controller mode, number of bytes per page (see *SPIXFC_CFG.pgsz and Note below*), SCK high and low values, and Slave Select (SS) active timing and inactive timing.
   a. Example:
   ***SPI Mode 0, page size = 1(four bytes), SCK high and low clocks = 1, SS Active = 2, and SS Inactive = 3***
   *SPIXFC_CFG* = `0x0091101`
2. Configure the Almost Empty and Almost Full levels for monitoring the FIFOs.
   a. Example:
   ***Almost Empty = 8, Almost Full = 12***
   `SPIXFC_FIFO_CTRL = 0x0C08`
3. Enable the Transaction and Results FIFOs as well as the feedback clock (*SPIXFC_GEN_CTRL* = `0x1006`).
4. Write to the Transaction FIFO to send a command to configure the SPI flash for configuration or programming. More than one command may be loaded to the FIFO.
5. Initialize the SPIXF Master Controller interrupt flags by clearing the *SPIXFC_INT_FL* register.
6. Set the interrupt enables for Transaction FIFO stalled and almost empty to make sure that the Transaction FIFO does not stall the AHB bus.
7. Write to the SPIXF Master Controller enable bit to start the transmission (*SPIXFC_GEN_CTRL* = 0x1007).
8. Monitor the results stalled interrupt status flag to know when data is available in the Results FIFO if data is received by this command. Reading the Results FIFO when the results stalled interrupt status flag is not set results in indeterminate data from the FIFO but does not stall the AHB bus.
9. Poll *SPIXFC_INT_FL.txrdy* for 1 to monitor completion of command.
10. Iterate step 6 through step 9 to monitor multiple commands to the SPI flash if necessary.

*Note: Page size is used if enabled by the SPIXF Master Controller header to configure the SPIXF Master Controller for larger transaction packet sizes (not to be confused with the SPI flash page size).*

Multiple headers and payloads are written to the Transaction FIFO for consecutive execution. As an example, complete the following steps to set up the external SPI flash using the SPIXF Master Controller:

1. Configure the SPIXF Master Controller so it can communicate with the default configuration of the external SPI flash chosen using the appropriate register and header settings.
2. Write the header and initial payload to the Transaction FIFO to send configuration of the data with (single/dual/quad) to the external SPI flash. This might require multiple commands to write status registers or to send specific commands.
3. Write header and payload to the Transaction FIFO to complete subsequent commands (read/write external SPI flash registers and program external SPI flash) using the new external SPI flash IO configuration.
4. Enable the SPIXF Master Controller to send commands to the external SPI flash.

### 7.2.2.4 Clock Phase and Polarity Control

The SPIXF Master Controller and the SPIXF Master support configuration of SCK phase and polarity:

- Clock polarity (CLKPOL) selects an active low/high clock and has no effect on the transfer format
- Clock phase (PHASE) selects one of two different transfer formats

The master always places data on the MOSI line a half cycle before the SCK edge for the slave to latch the data.

*Table 7-2* details the SCK phase and polarity combinations supported.

*Table 7-2: Clock Polarity and Phase Combinations*

| PHASE | CLKPOL | SCK Transmit Edge | SCK Receive Edge | SCK Idle State |
|---|---|---|---|---|
| 0 | 0 | Falling | Rising | Low |
| 1 | 1 | Falling | Rising | High |

*Note: Do not change the clock phase and polarity control while executing or reading from SPIXF space. This configuration should ideally be done prior to SPIXF transactions and remain unchanged while reading or executing from SPIXF space. If the clock phase and polarity need to be changed after the SPIXF slave select is low, the user must not be executing from SPIXF space, and the SPIXF block should be reset by setting GCR_RST1.spixip = 1.*

#### 7.2.2.5 Serial Clock Configuration

The output clock speed and pulse width can be controlled with the *SPIXFC_CFG.hiclk* and *SPIXFC_CFG.loclk* register fields as shown in *Figure 7-2*. These four-bit fields define clock pulses that synchronize the data transmission generated by the master.

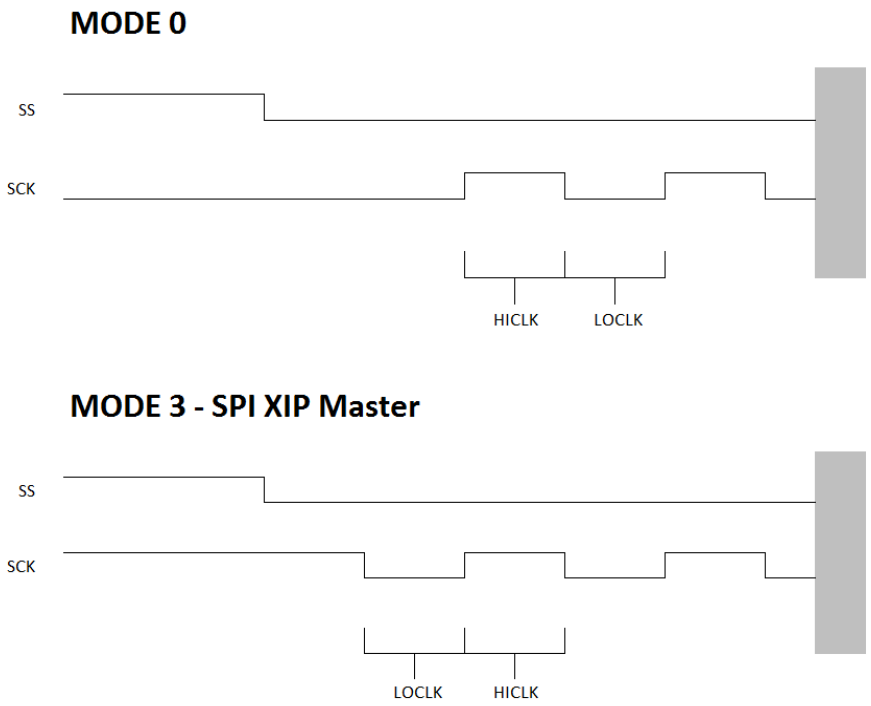
#### 7.2.2.6 Slave Select Transaction Delay Configuration

The transaction delay and slave select timing with respect to the active or inactive slave select edge are determined by a combination of the following register fields:

- *SPIXFC_CFG.ssact*
- *SPIXFC_CFG.ssiact*
- *SPIXFC_CFG.hiclk*
- *SPIXFC_CFG.loclk*
- *SPIXFC_CFG.mode*
- *SPIXFC_SP_CTRL.sckinh3* (if in mode 3)

Automatic slave selection de-assert for the SPIXF Master Controller occurs when the Transaction Header Deassert Slave Select field is set. The Slave Select is automatically de-asserted if the SPIXF Master Controller is disabled (*SPIXFC_GEN_CTRL.enable* = 0) or *GCR_RST1.spixip* = 1, resetting this peripheral.

*Figure 7-2. SPIXF Mode*




#### *7.2.2.7 Slave Select*

The SPIXF Master Controller operates with one slave device. A dedicated select pin for slave #0 is provided and controlled by hardware. Both execute-in-place and data storage are supported on slave #0.

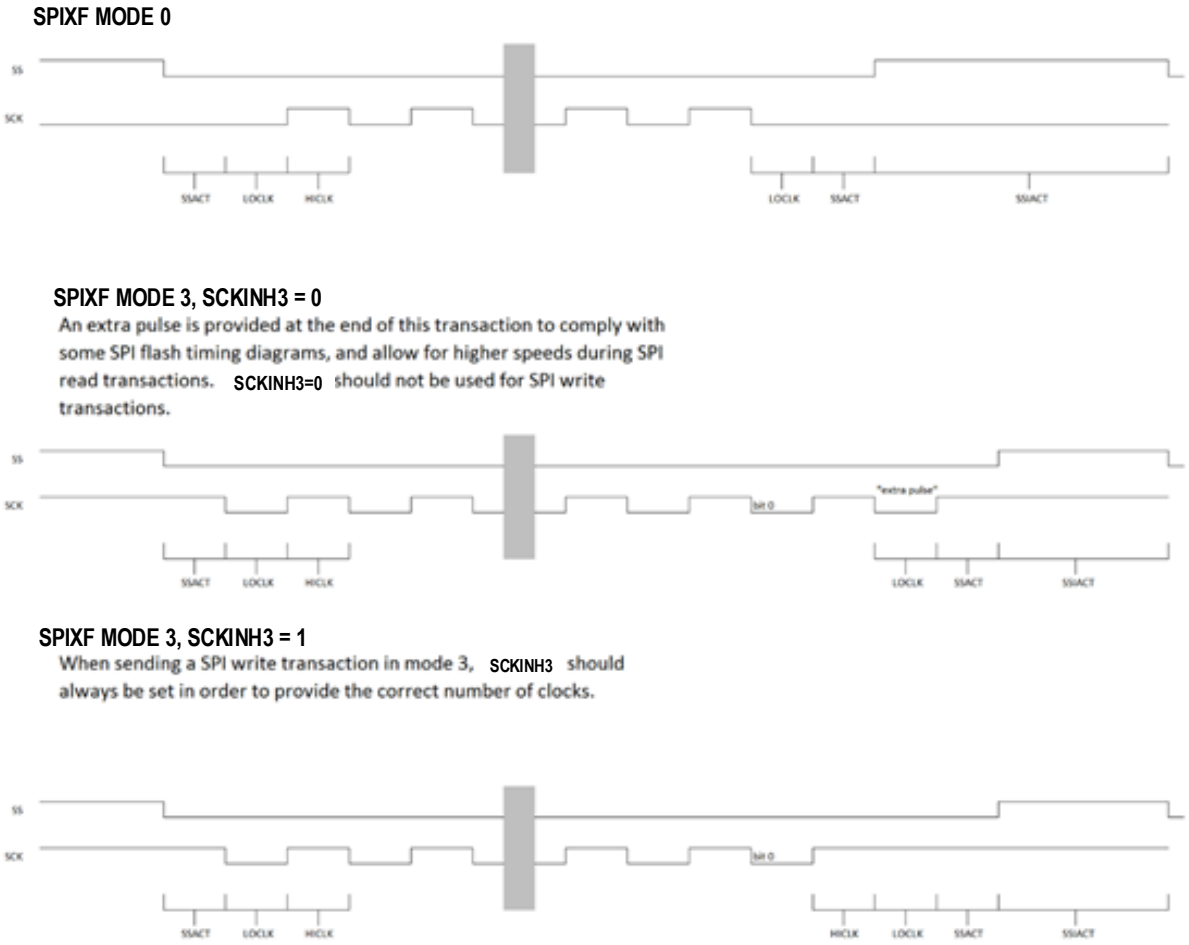
#### *7.2.2.8 Interrupts*

Interrupt logic is provided to allow efficient servicing of the SPIXF Master Controller by firmware. You can group interrupts into the following two categories:

- Keeping the Transaction FIFO full
- Keeping the Results FIFO empty

Programmable levels in the FIFO_CTRL register allow interrupt events to be issued if the Transaction FIFO falls below a certain level or if the Results FIFO fills above a certain level. See the FIFO_CTRL register description for more information.

*Figure 7-3. SPIXF Transaction Delay*




### 7.2.2.9 External SPI Flash Encryption

The user may optionally store encrypted data or code in the external SPI flash. Encryption of the SPI flash data is achieved using the cryptographic accelerator to encrypt the data and the SPIXF Master Controller to write the data. Data should be encrypted using AES-128, ECB mode.

Also, the following cryptographic accelerator control bits should be set when encrypting the SPIXF address space:

- CRYPTO_CTRL.*bis*
- CRYPTO_CTRL.*bso*

Setting CIPHER_CTRL.*src* = 0b11 selects the key stored for MDU use in memory locations 0x4000 5080 to 0x4000 508F.

The data must be pre-processed with an address mask. 128 bits plain data blocks are XORed (^) with a 128-bit address mask to avoid patterns in encrypted data. The address mask, addr_mask below, results in 128-bit aligned addressing by masking off the lower four bits of the input address (addr_in) as follows:

```
addr_mask = addr_in & 0xFFFF FFF0
```

For encryption, the data stored in the SPI flash, data_out below, is calculated as follows:

```
data_out = AES(data_in ^ ((addr_mask << 96) | ((addr_mask+4) << 64) | \
                     ((addr_mask+8) << 32) | (addr_mask+12)))
```

where:

```
data_in = word0:word1:word2:word3 (big endian format)
```

When using the cryptographic accelerator, the input data should be loaded as follows:

```
crypto_din0 = word0 ^ (addr_mask)
crypto_din1 = word1 ^ (addr_mask+4)
crypto_din2 = word2 ^ (addr_mask+8)
crypto_din3 = word3 ^ (addr_mask+12)
```

Once the encrypted data is available (either via FIFO or via Crypto Data Output Registers [3:0]), this data may be written to SPI flash using the SPIXF Master Controller.

The available output bytes from the cryptographic accelerator should be written to SPIXF flash space as shown in *Table 7-3*

*Table 7-3: Encrypted Data Write Order to SPIX Flash Memory*

| Least Significant Word | | | Most Significant Word |
|---|---|---|---|
| crypto_dout0 | crypto_dout1 | crypto_dout2 | crypto_dout3 |

### 7.2.2.10 SPIXF Master Controller Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the SPIXF Base Peripheral Offset Address.

*Table 7-4. SPIXF Master Controller Register Offsets, Names, Access and Description*

| Offset | Register | Access | Description |
|---|---|---|---|
| [0x0000] | *SPIXFC_CFG* | RW | SPIXF Controller Configuration Register |
| [0x0004] | *SPIXFC_SS_POL* | RW | SPIXF Controller Slave Select Polarity Register |
| [0x0008] | *SPIXFC_GEN_CTRL* | RW | SPIXF Controller General Controller Register |
| [0x000C] | *SPIXFC_FIFO_CTRL* | RW | SPIXF Controller FIFO Control and Status Register |
| [0x0010] | *SPIXFC_SP_CTRL* | RW | SPIXF Controller Special Control Register |
| [0x0014] | *SPIXFC_INT_FL* | RW | SPIXF Controller Interrupt Status Register |
| [0x0018] | *SPIXFC_INT_EN* | RW | SPIXF Controller Interrupt Enable Register |

### 7.2.2.11 SPIXF Master Controller Register Details

*Table 7-5. SPIXF Controller Configuration Register*

| SPIXF Controller Configuration Register | | | SPIXFC_CFG | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| SPIXF Controller Configuration Register | SPIXFC_CFG | [0x0000] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 23:20 | iosmpl | R/W | 0 | **Sample Delay**<br>Defines additional delay in SPI clock periods to wait before sampling SDIO input. This value must be less than or equal to the value set for HICLK (for SPI modes 0 and 3). This value applies only in non-clock feedback mode (*SPIXFC_GEN_CTRL*.sckfb = 0).<br>0b0000: No Delay<br>0b0001: 1 SPI Clock delay<br>0b1111: 15 SPI Clock delay |
| 19:18 | inact | R/W | 0 | **Slave Select Inactive Stretch**<br>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).<br>Refer to section *Slave Select Transaction Delay Configuration* for detailed information.<br>0b00: 4 system clocks<br>0b01: 6 system clocks<br>0b10: 8 system clocks<br>0b11: 12 system clocks |
| 17:16 | ssact | R/W | 0 | **Slave Select Holdoff**<br>Controls the delay from assertion of slave select to the start of SCK pulse and the delay from the end of SCK pulses to de-assertion of slave select.<br>Refer to section *Slave Select Transaction Delay Configuration* for detailed information.<br>0b00: 0 system clocks<br>0b01: 2 system clocks<br>0b10: 4 system clocks<br>0b11: 8 system clocks |
| 15:12 | loclk | R/W | 0 | **SCK Low Clocks**<br>Number of system clocks that SCK is held low when SCK pulses are generated<br>0: 16 system clocks<br>1: 1 system clock<br>2: 2 system clocks<br>3: 3 system clocks<br>…<br>All other values: This value defines the number of system clock that SCK is held low. |
| 11:8 | hiclk | R/W | 0 | **SCK High Clocks**<br>Number of system clocks that SCK is held high when SCK pulses are generated.<br>00: 16 system clocks.<br>All other values: This value defines the number of system clock that SCK is held high. |
| 7:6 | pgsz | R/W | 0 | **Page Size**<br>Defines the number of bytes per page for transactions that define transfers in terms of pages.<br>00: 4 bytes<br>01: 8 bytes<br>10: 16 bytes<br>11: 32 bytes |

| SPIXF Controller Configuration Register | | | SPIXFC_CFG | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 5:4 | mode | R/W | 0 | **SPI Mode.**<br>Defines the SPI mode.<br>00: SPI Mode 0. Clock Polarity = 0, Clock Phase = 0<br>01: Invalid<br>10: Invalid<br>11: SPI Mode 3. Clock Polarity = 1, Clock Phase = 1 |
| 3 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 2:0 | ssel | R/W | 0 | **Slave Select.**<br>Only Slave 0 is supported<br>0b000: Slave 0 is selected<br>0b001-0b111: Invalid |

*Table 7-6. SPIXF Controller Slave Select Polarity Register*

| SPIXF Controller Slave Select Polarity Register | | | SPIXFC_SS_POL | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:1 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 0 | sspol_0 | R/W | 0 | **Slave Select 0 Polarity**<br>0: Active Low<br>1: Active High |

*Table 7-7. SPIXF Controller General Control Register*

| SPIXF Controller General Control Register | | | SPIXFC_GEN_CTRL | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 25 | sckfbinv | R/W | 0 | **SCK Inversion**<br>0: Use SCK as feedback clock<br>1: Use inverted SCK as feedback clock |
| 24 | sckfb | R/W | 0 | **Enable SCK Feedback mode**<br>0: Disable SCK feedback mode.<br>1: Enable SCK feedback mode. |
| 23 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 22 | smplss | R/W | 0 | **Simple Mode Slave Select**<br>0: No action<br>1: Deassert Slave Select when simple = 1 |

| SPIXF Controller General Control Register | SPIXFC_GEN_CTRL | [0x0008] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 21 | simplerx | R/W | 0 | **Simple Receive Enable**<br>Setting this bit to a 1 initiates a SPI transaction as defined in the Receive-Only Transaction Header when in Simple Mode.<br>0: No action<br>1: Initiate SPI transaction |
| 20 | simple | R/W | 0 | **Simple Mode Enable**<br>0: Simple Mode disabled<br>1: Simple Mode enabled |
| 19:16 | bbdatoe | R/W | 0 | **Bit Bang SDIO Output Enable**<br>Enable output of SDIO0-3 in Bit-Bang mode.<br>bit3 = SDIO[3]<br>bit2 = SDIO[2]<br>bit1 = SDIO[1]<br>bit0 = SDIO[0] |
| 15:12 | bbdat | R/W | 0 | **SDIO Drive value in Bit-Bang mode**<br>Defines the output state of the SDIO outputs when in Bit-Bang mode (SPIXFC_GEN_CTRL.bbmode=1)<br>bit[3]: SDIO[3]<br>bit[2]: SDIO[2]<br>bit[1]: SDIO[1]<br>bit[0]: SDIO[0] |
| 11:8 | sdatain | R/W | - | **SDIO Input Data Value**<br>Returns the state of the SDIO Input values. Writes to this field have no effect.<br>bit3: SDIO[3]<br>bit2: SDIO[2]<br>bit 1: SDIO[1]<br>bit 0: SDIO[0] |
| 7 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 6 | sckdr | R/W | 0 | **SCK Drive and State**<br>This bit reflects the state of the SCK. When in Bit-Bang mode (SPIXFC_GEN_CTRL.*bbmode* = 1), this bit is written to control the output state of the SCK.<br>0: SCK is 0.<br>1: SCK is 1. |
| 5 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 4 | ssdr | R/W | 0 | **Slave Select Drive and State**<br>This bit reflects the state of the slave select. This accounts for the polarity as defined in the *SPIXFC_SS_POL* register. When in Bit-Bang mode, this bit is written to control the output state of the slave select.<br>0: Selected Slave Select Output is 0<br>1: Selected Slave Select Output is 1. |
| 3 | bbmode | R/W | 0 | **Bit-Bang Mode**<br>0: Disable Bit-Bang mode<br>1: Enable Bit-Bang mode |

| SPIXF Controller General Control Register | | SPIXFC_GEN_CTRL | | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 2 | rfifoen | R/W | 0 | **Results FIFO Enable**<br>Setting this bit enables the Results FIFO. Clearing this bit disables the Results FIFO and places it into a reset state.<br>0: Disable result FIFO.<br>1: Enable result FIFO. |
| 1 | tfifoen | R/W | 0 | **Transaction FIFO Enable**<br>Setting this bit to 1 enables the Transaction FIFO. Clearing this bit disables the Transaction FIFO and places it into reset state.<br>0: Disable Transaction FIFO.<br>1: Enable Transaction FIFO. |
| 0 | enable | R/W | 0 | **SPI Master enable**<br>Setting this bit to 1 enables SPI Master for processing transactions. Clearing this bit disables the SPI Master and puts the block into reset state.<br>0: Disable SPI Master, putting it into a reset state.<br>1: Enable SPI Master for processing transactions. |

*Table 7-8. SPIXF Controller FIFO Control and Status Register*

| SPIXF Controller FIFO Control and Status Register | | SPIXFC_FIFO_CTRL | | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:30 | - | R/W | 00 | **Reserved for Future Use**<br>Do not modify this field. |
| 29:24 | rfifocnt | R/W | 0 | **Results FIFO Entry Count**<br>Current number of used entries (bytes) in Results FIFO. Writes to this field are ignored. |
| 23:21 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 20:16 | rfifolvl | R/W | 0 | **Results FIFO Almost Full Level**<br>The Almost Full flag is asserted when the number of used FIFO entries (bytes) exceed this value. FIFO depth is 32 bytes. |
| 15:13 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 12:8 | tfifocnt | R/W | 0 | **Transaction FIFO Entry Count**<br>Current number of used entries (words) in the Transaction FIFO. Writes to this field are ignored. |
| 7:4 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3:0 | tfifolvl | R/W | 0 | **Transaction FIFO Almost Empty Level**<br>The Almost Empty flag is asserted when the number of unused FIFO entries in words exceeds this value. FIFO depth is 16 words. |

*Table 7-9. SPIXF Controller Special Control Register*

| SPIXF Controller Special Control Register | | | SPIXFC_SP_CTRL | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:17 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 16 | sckinh3 | R/W | 0 | **SCK Inhibit mode 3**<br>In SPI mode 3, some SPI flash read timing diagrams show the last SCK going low prior to de-assertion. The default is to support this additional falling edge of the clock. When this bit is set, and the device is in SPI mode 3, the SPI clock is held high while slave select is de-asserted. This is to support some SPI flash write timing diagrams.<br>0: Allow trailing SCK low pulse prior to slave select de-assertion.<br>1: Inhibit trailing SCK low pulse prior to slave select de-assertion. |
| 15:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11:8 | sdiooe | R/W | 0 | **SDIO Output Enable Sample Mode**<br>Defines whether the output is enabled for each SDIO pin.<br>Bit 11: SDIO[3]<br>Bit 10: SDIO[2]<br>Bit 9: SDIO[1]<br>Bit 8: SDIO[0]<br>0: SDIO output disabled.<br>1: SDIO output enabled. |
| 7:4 | sdioout | R/W | 0 | **SDIO Output Value Sample Mode**<br>Defines the values for the SDIO outputs when in Sample Mode (SPIXFC_SP_CTRL.*sampl*=1).<br>Bit 7: SDIO[3]<br>Bit 6: SDIO[2]<br>Bit 5: SDIO[1]<br>Bit 4: SDIO[0] |
| 3:1 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field |
| 0 | sampl | R/W | 0 | **SDIO Sample Mode Enable**<br>Setting this bit to a 1 enables the ability to drive SDIO outputs prior to the assertion of Slave Select. This bit must only be set when the SPIXF bus is idle and the transaction FIFO is empty. This bit is automatically cleared by hardware after the next slave select assertion.<br>0: Sample Mode disabled<br>1: Sample mode enabled |

*Table 7-10. SPIXF Controller Interrupt Status Register*

| SPIXF Controller Interrupt Status Register | | | SPIXFC_INT_FL | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:6 | - | R/W1C | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| SPIXF Controller Interrupt Status Register | | | SPIXFC_INT_FL | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 5 | rfifoaf | R/W1C | 0 | **Results FIFO Almost Full Flag.**<br>This flag is set by hardware when the Results FIFO is almost full as defined by *rfifolvl*.<br>0: Results FIFO level below the Almost Full level<br>1: Results FIFO level at almost full level. |
| 4 | tfifoae | R/W1C | 1 | **Transaction FIFO Almost Empty Flag.**<br>This flag is set by hardware when the Transaction FIFO is almost empty as defined by *tfifolvl*. This does not depend on block enable or the slave select value.<br>0: Transaction FIFO not Almost Empty<br>1: Transaction FIFO Almost Empty. |
| 3 | rdone | R/W1C | 0 | **Results Done Interrupt Status.**<br>This flag is set by hardware when the Results FIFO is not empty, and the slave select is deasserted.<br>0: Results FIFO ready<br>1: Results FIFO Not ready. |
| 2 | trdy | R/W1C | 0 | **Transaction Ready Interrupt Status.**<br>This flag is set by hardware when the Transaction FIFO is empty, and the slave select is deasserted.<br>0: FIFO Transaction not ready<br>1: FIFO Transaction is ready. |
| 1 | rstall | R/W1C | 0 | **Results Stalled Interrupt Flag.**<br>This flag is set by hardware when the Results FIFO is full, and the selected slave select is asserted.<br>0: Normal FIFO operation.<br>1: Stalled FIFO. |
| 0 | tstall | R/W1C | 0 | **Transaction Stalled Interrupt Flag.**<br>This flag is set by hardware when the Transaction FIFO is empty, and the selected slave select is asserted.<br>0: Normal FIFO Transaction.<br>1: Stalled FIFO Transaction. |

*Table 7-11. SPIXF Controller Interrupt Enable Register*

| SPIXF Controller Interrupt Enable Register | | | SPIXFC_INT_EN | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:6 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 5 | rfifoafie | R/W | 0 | **Results FIFO Almost Full Interrupt Enable.**<br>Setting this bit enables interrupt generation when the *SPIXFC_INT_FL.rfifoaf* flag is set. Clearing this bit means that no interrupt is generated.<br>0: Disable Results FIFO Almost Full Interrupt<br>1: Enable Results FIFO Almost Full Interrupt. |

| SPIXF Controller Interrupt Enable Register | | | SPIXFC_INT_EN | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4 | tfifoaeie | R/W | 1 | **Transaction FIFO Almost Empty Interrupt Enable.**<br>Setting this bit enables interrupt generation when the *SPIXFC_INT_FL.tfifoae* flag is set. Clearing this bit means that no interrupt is generated.<br>0: Disable Transaction FIFO Almost Empty Interrupt.<br>1: Enable Transaction FIFO Almost Empty Interrupt. |
| 3 | rdoneie | R/W | 0 | **Results Done Interrupt Enable.**<br>Setting this bit enables interrupt generation when the *SPIXFC_INT_FL.rdone* flag is set. Clearing this bit means that no interrupt is generated.<br>0: Disable Results Done Interrupt.<br>1: Enable Results Done Interrupt. |
| 2 | trdyie | R/W | 0 | **Transaction Ready Interrupt Enable.**<br>Setting this bit enables interrupt generation when the *SPIXFC_INT_FL.trdy* flag is set. Clearing this bit means that no interrupt is generated.<br>0: Disable Transaction Ready Interrupt.<br>1: Enable Transaction Ready Interrupt. |
| 1 | rstallie | R/W | 0 | **Results Stalled Interrupt Enable.**<br>Setting this bit enables the Results Stalled Interrupt. Clearing this bit means that no interrupt is generated.<br>0: Disable Results Stalled Interrupt.<br>1: Enable Results Stalled Interrupt. |
| 0 | tstallie | R/W | 0 | **Transaction Stalled Interrupt Enable.**<br>Setting this bit enables interrupt generation when the *SPIXFC_INT_FL.tstall* flag is set. Clearing this bit means that no interrupt is generated.<br>0: Disable Transaction Stalled Interrupt.<br>1: Enable Transaction Stalled Interrupt. |

### 7.2.2.12 SPIXF Master Controller FIFO Registers

Refer to *Table 2-2: AHB Peripheral Base Address Map* for the SPIXF Master Controller FIFO Base Peripheral Offset Address.

*Table 7-12. SPIXF Master Controller FIFO Register Offsets, Names, Access and Description*

| Offset | Register | Access | Description |
|---|---|---|---|
| [0x0000] | *SPIXFC_FIFO_TX* | WO | SPIXF Master Controller TX FIFO Register |
| [0x0004] | *SPIXFC_FIFO_RX* | RO | SPIXF Master Controller RX FIFO Register |

### 7.2.2.13 SPIXF Master Controller FIFO Register Details

*Table 7-13. SPIXF Master Controller TX FIFO Register*

| SPIXF Master Controller TX FIFO Register | | | SPIXFC_FIFO_TX | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | data | WO | 0 | **TX FIFO**<br>Writes to this register are put into the TX FIFO for the SPIXF Master Controller. |

*Table 7-14. SPIXF Master Controller TX FIFO Register*

| SPIXF Master Controller RX FIFO Register | | | SPIXFC_FIFO_RX | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | data | RO | 0 | **RX FIFO**<br>Reads from this register return the data from the SPIXF Master Controller RX FIFO. |

### 7.2.3 *SPIXF Master*

The SPIXF Master block consists of two interfaces.

One interface is an AHB slave interface that is driven by a 16KB Unified Instruction and Constant cache to support cache operation. The AHB slave supports either instruction execution or fetching of data from external SPI flash. This interface is accessible to firmware using an AHB interface to support high-speed data transfer. The address for SPI flash access is determined by the AHB access and is mapped from address 0x0800 0000 to 0x0FFF FFFF for a total addressable space of 128MB.

The other interface is an APB slave that is used for control. Communication and interface configuration are set using the APB registers. The command and mode/dummy information required by external SPI flashes are configured with APB register settings.

The command used to transfer SPI flash data is configured using firmware. Then, the access to SPI flash space (either code execution or data) may be performed by firmware. The AHB transaction initiated by the firmware provides address and other transaction critical parameters to control the data transfer from the external SPI flash.

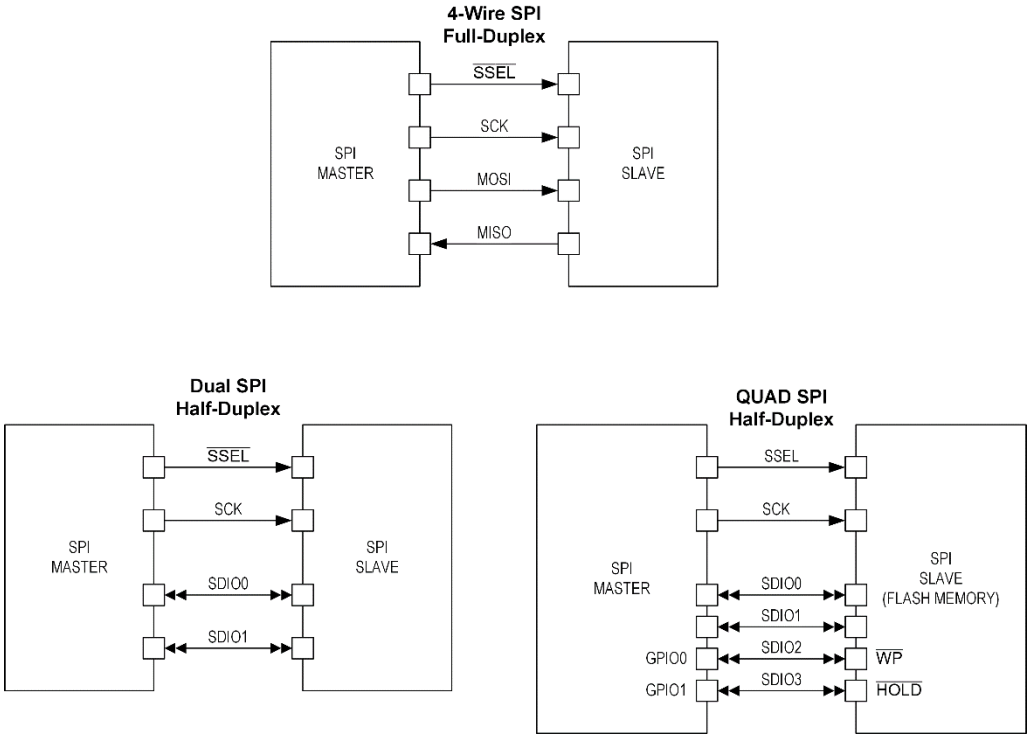
You should exercise care in choosing the correct configuration and command to support the speed of data transfer. The SPIXF Master provides SCK periods as fast as the AHB clock speed divided by two. The external SPI flash configuration to support data transfer rates must be performed by the SPIXF Master Controller.

#### *7.2.3.1 SPIXF Pin Configuration*

The SPIXF Master and SPIXF Master Controller use a highly-configurable, flexible, and efficient interface supporting single, dual, or quad I/O. Dedicated pins are provided to support high-speed communication. The following pin configurations are supported and shown in Figure 7-4:

- Four-wire SPI: SS, SCK, MOSI on SDIO0, and MISO on SDIO1
- Dual SPI: SS, SCK, SDIO0, and SDIO1
- Quad SPI: SS, SCK, SDIO0, SDIO1, SDIO2, and SDIO3

*Figure 7-4. Supported SPI configuration*



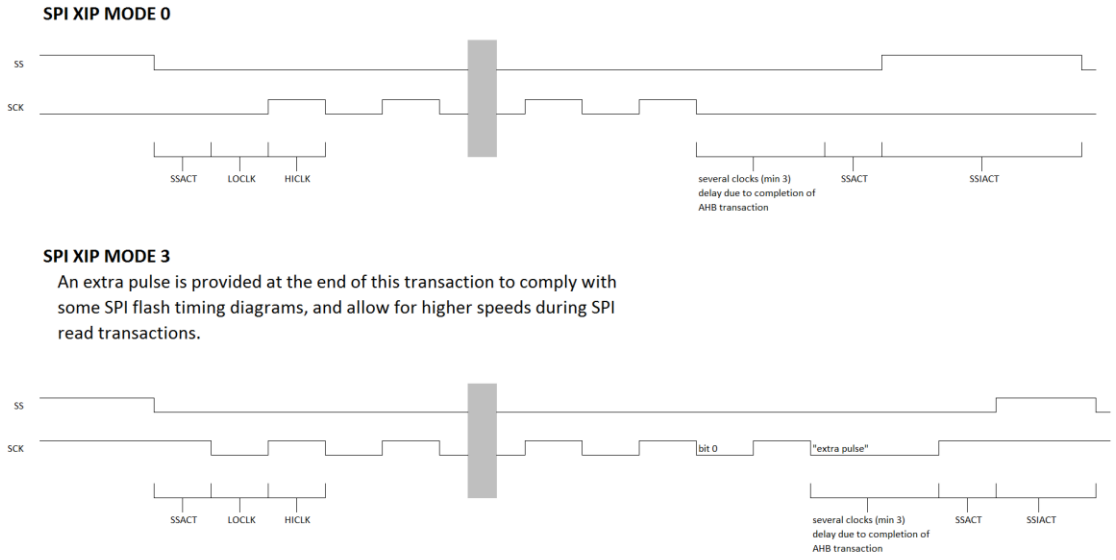

### 7.2.3.2 Slave-Select Transaction Delay Configuration

The transaction delay and slave-select timing with respect to the active or inactive slave-select edge is determined by a combination of the following register fields:

- *SPIXF_CFG.ssact*
- *SPIXF_CFG.ssinact*
- *SPIXF_CFG.hiclk*
- *SPIXF_CFG.loclk*
- *SPIXF_CFG.mode*

Automatic slave-select de-assertion only occurs when the next flash address fetched is not contiguous to the current flash address that is being read or used for execution. The SPIXF does not automatically de-assert slave selection under any other circumstance including data read or execution of areas outside of the SPIXF space. For these cases, manual control of the slave select is provided. Invoke manual control only when running from internal memory. You can de-assert slave-select safely by setting GCR_RSTR1.*spixip*. This resets the SPIXF block (including turning off decryption if previously enabled) and causes the slave select to de-assert. The SPIXF block requires reconfiguration prior to subsequent access to external SPI flash space either for execution or data reads.

*Figure 7-5. SPIXF Delay Configuration*




### *7.2.3.3 SPIXF Read Sequence Configuration and Control*

Assertion of SPIXF slave select followed by the read command, then the read address. After the read address is sent 0 or more clocks are generated (called dummy bytes or mode clocks) to allow the flash to access the data being addressed. The remainder of the SPI access is read data. Sequential bytes are read until the de-assertion of SPIXF slave select.

Depending on the read command and the SPI flash configuration, the read command is sent over 1, 2, or 4 bits per clock. The same is true for the address, data, and mode/dummy clocks. Also, configure the device to eliminate the sending of the read command once the command is sent to the SPI flash device. This is enabled and disabled through special data sent during the mode or dummy period between address and read data.

#### 7.2.3.4 Sample SPIXF Master Configuration - Execute Code

Complete the following steps to execute the SPIXF Master Configuration sample:

1. Turn on ICache XIP Clock (*GCR_PCLK_DIS1*.*icachexipf* = 1).
   a. The cache can be put into different power states. See *GCR_MEM_CLK* for options.
2. Configure the SPIXF Master mode, slave select polarity, slave number, and slave select timing.
   a. Example:
      i. SPI Mode 0
      ii. Slave select high
      iii. Slave #0
      iv. 1 SPI clock per 2 AHB clocks
      v. *SPIXF_CFG* = 0x1104.
3. Configure the command value, the command, address, data width, and whether the address is three- or four-byte mode.
   a. Example Read command:
      i. command value = 0x03
      ii. command width = single data I/O
      iii. address bit = single data I/O,
      iv. data width = single data I/O
      v. 3-byte address mode
      vi. *SPIXF_FETCH_CTRL* = 0x0003.
4. Configure the SPIXF mode/dummy field and the data for the mode/dummy field
   a. Example:
      i. Mode clocks = 0 (no dummy field)
      ii. *SPIXF_MODE_CTRL* = 0x0
5. Enable the SPIXF feedback clock control.
   a. Example:
      i. SPIXF feedback clock enabled using non-inverted serial feedback clock
      ii. *SPIXF_FB_CTRL* = 0x0001.
6. Jump to the start of application code in SPI flash space.
   a. Example pseudo code:
      i. jump_to_external_flash = (void) (0x08000001)
      ii. jump_to_external_flash()

#### 7.2.3.5 Clock Phase and Polarity Control

The SPIXF Master's clock phase and polarity should match the configuration set up by the SPIXF Master Controller. For more information about clock phase and polarity control please, See (link to the section on SPIXF Clock Phase and Polarity Control).

#### 7.2.3.6 Serial Clock Feedback Mode

The SPIXF Master supports high-speed transfer up to 60MHz using the Serial Feedback Clock Mode (*SPIXF_FB_CTRL* .*fbmd*=1). The master output clock is routed back into the digital logic to sample incoming data from the slave. Configuring the SPIXF in serial feedback clock mode uses the slave output data stream sampled using the SCK feedback clock.

This allows for automatic alignment of the master clock to the input slave data allowing for faster speeds. The Serial Feedback Mode should not be changed while the SPIXF slave select is low. This configuration should be done prior to SPIXF transactions and remain unchanged while reading or executing from SPIXF space. If the Serial Feedback Mode needs to be changed after the SPIXF slave select is low, the user must not be executing from SPIXF space, and the SPIXF block should be reset by setting *GCR_RST1*.*spixip* = 1.

#### 7.2.3.7 External SPI Flash Decryption

If data in the SPI flash is encrypted when written, it might be transparently decrypted on read back using either code execution or data reads. Decryption is not enabled by default. Setting *SPIXF_SEC_CTRL.decen* = 1 enables the Memory Decryption Unit (MDU). The MDU uses an AES-128 algorithm in ECB mode. This key is written by the user to the register file locations 0x4000 5020 to 0x4000 502F, which is automatically used by the MDU for decryption.

See *SPIXF Master Controller* for information about data encryption for SPIX flash.

#### 7.2.3.8 SPIXF Master Registers

Reserved register bits should only be written as 0.

*Table 7-15. SPIXF Master Register Addresses (Base ADDR = 0x4002 6000)*

| Offset | Register | Access | Description |
|---|---|---|---|
| [0x0000] | *SPIXF_CFG* | R/W | SPIXF Configuration Register |
| [0x0004] | *SPIXF_FETCH_CTRL* | R/W | SPIXF Fetch Control Register |
| [0x0008] | *SPIXF_MODE_CTRL* | R/W | SPIXF Mode Control Register |
| [0x000C] | *SPIXF_MODE_DATA* | R/W | SPIXF Mode Data Register |
| [0x0010] | *SPIXF_FB_CTRL* | R/W | SPIXF SCK Feedback Control Register |
| [0x001C] | *SPIXF_IO_CTRL* | R/W | SPIXF I/O Control Register |
| [0x0020] | *SPIXF_SEC_CTRL* | R/W | SPIXF Memory Security Register |
| [0x0024] | *SPIXF_BUS_IDLE* | R/W | SPIXF Bus Idle Detection |

#### 7.2.3.9 SPIXF Master Register Details

*Table 7-16. SPIXF Configuration Register*

| SPIXF Configuration Register | | | SPIXF_CFG | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:20 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 19:18 | ssiact | R/W | 0 | **Slave Select Inactive Timing**<br>Controls delay from de-assertion of slave select to re-assertion of slave select for another SPI transaction. See *7.2.2.6*, *above*, for details on slave select transaction delay configuration.<br>0b00: 1 system clocks<br>0b01: 3 system clocks<br>0b10: 5 system clocks<br>0b11: 9 system clocks |
| 17:16 | ssact | R/W | 0 | **Slave Select Active Timing**<br>Controls delay from assertion of slave select to start of the SCK pulse and delay from the end of SCK pulses to de-assertion of slave select. See *7.2.2.6*, *above*, for details on slave select transaction delay configuration.<br>0b00: 0 system clocks<br>0b01: 2 system clocks<br>0b10: 4 system clocks<br>0b11: 8 system clocks |

| SPIXF Configuration Register | | | SPIXF_CFG | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:12 | hiclk | R/W | 0 | **SCK High Clocks**<br>Number of system clocks that SCK is held high when SCK pulses are generated.<br>0: Invalid<br>All other values: The number of system clocks that SCK is held high. |
| 11:8 | loclk | R/W | 0 | **SCK Low Clocks**<br>Number of system clocks that SCK is held low when SCK pulses are generated.<br>0: Invalid<br>All other values: The number of system clocks that SCK is held low. |
| 7 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 6:4 | ssel | R/W | 0 | **Slave Select**<br>Only valid value is zero |
| 3 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 2 | sspol | R/W | 1 | **Slave Select Polarity**<br>This bit controls the polarity of the slave select.<br>0: Slave Select active high<br>1: Slave Select active low |
| 1:0 | mode | R/W | 0 | **SPI mode**<br>Set this field to the required SPI mode.<br>0b00: SPI mode 0<br>0b01: Reserved<br>0b10: Reserved<br>0b11: SPI mode 3 |

*Table 7-17. SPIXF Fetch Control Register*

| SPIXF Fetch Control Register | | | SPIXF_FETCH_CTRL | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:17 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 16 | addr4 | R/W | 0 | **Four-Byte Address mode**<br>Enables 4-byte Flash Address mode. Defaults to value as defined by parameter in instantiation. User can override.<br>0: 3-byte address mode<br>1: 4-byte address mode |
| 15:14 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 13:12 | data_width | R/W | 0 | **Data Width**<br>Number of data I/O used to receive data.<br>0b00: Single SDIO<br>0b01: Dual SDIO<br>0b10: Quad SDIO<br>0b11: Reserved |

| SPIXF Fetch Control Register | | | SPIXF_FETCH_CTRL | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 11:10 | addr_width | R/W | 0 | **Address Width**<br>Number of data I/O used to send address and mode/dummy clocks.<br>0b00: Single SDIO<br>0b01: Dual SDIO<br>0b10: Quad SDIO<br>0b11: Reserved |
| 9:8 | cmdwth | R/W | 0 | **Command Width**<br>Number of data I/O used to send commands.<br>0b00: Single SDIO<br>0b01: Dual SDIO<br>0b10: Quad SDIO<br>0b11: Reserved |
| 7:0 | cmdval | R/W | 0 | **Command Value**<br>Command value sent to target to initiate fetching from SPI flash. |

*Table 7-18. SPIXF Mode Control Register*

| SPIXF Mode Control Register | | | SPIXF_MODE_CTRL | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:10 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 9 | mode_send | R/W | 0 | **Mode Send**<br>Setting this field ensures that the next SPI flash transaction will send the mode byte as defined in the *SPIXF_MODE_DATA*.*mddata* field.<br>When this field is set, the next SPI flash read operation exits continuous mode cleanly. This field and the *SPIXF_MODE_CTRL*.*nocmd* field is automatically cleared by hardware after the next SPI transaction.<br>0: No Action.<br>1: Send Mode Byte on next transaction |
| 8 | nocmd | R/W | 0 | **No Command Mode**<br>Read command sent only once after this bit is set.<br>0: Send read command every time SPI transaction is initiated.<br>1: Send read command on first transaction only and not on subsequent transactions. |
| 7:4 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3:0 | mdclk | R/W | 0 | **Mode Clocks**<br>Number of SPI clocks needed during the mode/dummy phase of fetch. |

*Table 7-19. SPIXF Mode Data Register*

| SPIXF Mode Data Register | | | SPIXF_MODE_DATA | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | mdoe | R/W | 0 | **Mode Output Enable**<br>Output enable state for each corresponding data bit in *SPIXF_MODE_DATA.mddata*.<br>0: Output enable off, I/O is tristate stated.<br>1: Output enable on, I/O is driving *SPIXF_MODE_DATA.mddata*. |
| 15:0 | mddata | R/W | 0 | **Mode Data**<br>Specifies the data to send with the dummy/mode clocks. |

*Table 7-20. SPIXF SCK Feedback Control Register*

| SPIXF SCK Feedback Control Register | | | SPIXF_FB_CTRL | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | fbinv | R/W | 0 | **SCK feedback Clock inversion.**<br>0: Non-inverted SCK is used for feedback clock<br>1: Inverted SCK is used for feedback clock |
| 0 | fbmd | R/W | 0 | **SCK Feedback Mode Enable. Enable SCK feedback mode**<br>0: Disable SCK feedback mode<br>1: Enable SCK feedback mode |

*Table 7-21. SPIXF I/O Control Register*

| SPIXF I/O Control Register | | | SPIXF_IO_CTRL | [0x001C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:5 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 4:3 | pupdctrl | R/W | 1 | **IO Pullup/Pulldown Control**<br>These bits control the pullups and pulldowns associated with all SPIXF SDIO pins.<br>0b00: tristate<br>0b01: pull-up<br>0b10: pull-down<br>0b11: pull-up |
| 2 | sdio_ds | R/W | 1 | **SDIO Drive Strength**<br>This bit controls the drive strength of all SDIO pins.<br>0: Low Drive Strength.<br>1: Hi Drive Strength. |
| 1 | ss_ds | R/W | 1 | **Slave Select Drive Strength**<br>This bit controls the drive strength on the dedicated slave select pin.<br>0: Low Drive Strength.<br>1: Hi Drive Strength. |

| SPIXF I/O Control Register | | | SPIXF_IO_CTRL | [0x001C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | sck_ds | R/W | 1 | **SCK Drive Strength**<br>This bit controls the drive strength on the SCK pin.<br>0: Low Drive Strength.<br>1: Hi Drive Strength. |

*Table 7-22. SPIXF Memory Security Control Register*

| SPIXF Memory Security Control Register | | | SPIXF_SEC_CTRL | [0x0020] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | auth_disable | R/W | 0 | **Integrity Enable.**<br>0: Integrity checking enabled.<br>1: Integrity checking disabled. |
| 0 | dec_en | R/W | 0 | **Decryption Enable.**<br>0: Disable decryption of SPIXF data.<br>1: Enable decryption of SPIXF data. |

*Table 7-23. SPIXF Bus Idle Detection*

| SPIXF Bus Idle Detection | | | SPIXF_BUS_IDLE | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15:0 | busidle | R/W | 0 | **Bus Idle Timer Limit**<br>A 16-bit timer will be triggered for each external access. The timer will be restarted if another access is performed before the timer expires. When the timer expires, slave select will be deactivated.<br>This register contains the limit (expiration) value for the timer. A value of 0 will disable the bus idle detection and non-zero values enable bus idle detection. This feature is useful when fetching code out of I-cache, ROM or in SLEEP and DEEPSLEEP modes. When this number is too small, Slave Select will be deactivated on every access, which may reduce current consumption, but decreases performance. |

## 7.3 SPI Execute-in-Place RAM

The SPI Execute-in-Place RAM Master Controller is an instantiation of the Quad SPI Interface with the following features:

- Four SPI modes (mode 0, 1, 2, and 3)
- Master mode only support
- Wakeup from low power modes based on configurable Transmit and Receive FIFO Levels
- Dual SPI Mode with two bidirectional serial data I/O (SDIO) lines
- High Performance Quad SPI Mode with four bidirectional SDIO lines
- Up to two Slave Select (SS) control lines with programmable polarity
- Programmable Serial Clock (SCK) frequency and duty cycle with 60MHz maximum
- 32-byte Transmit FIFO, 32-byte Receive FIFO with DMA support backed by a 16KB Data Cache

The SPIXR Master Controller allows the CPU to transparently execute instructions stored in an external SPI SRAM device. Instructions fetched using the SPIXR Master Controller are cached just like instructions fetched from internal program memory. You can also use the SPIXR Master Controller to access large amounts of external static data that would otherwise reside in internal data memory. This device provides support for a wide variety of read commands that balance the need for backward comparability and high performance.

The SPIXR Master Controller provides a highly-configurable, flexible, and efficient interface that supports the programming and configuration of external SPI SRAM devices.

Prior to using the SPI SRAM device, you must configure the SPIXR Master Controller interface.

The SPIXR Master Controller block consists of two interfaces. One interface is an AHB slave interface that is driven by a 16KB Unified Instruction and Constant cache to support cache operation. The AHB slave supports either instruction execution or fetching of data from external SPI SRAM. This interface is accessible to firmware using an AHB interface to support high-speed data transfer. The address for SPI SRAM access is determined by the AHB access and is mapped from address 0x8000 0000 to 0xFFFF FFFF for a total addressable space of 512MB.

The other interface is an APB slave that is used for control. Communication and interface configuration are set using the APB registers. The command and mode/dummy information required by external SPI SRAM devices are configured with APB register settings.

The command used to transfer SPI SRAM data is configured using firmware. Then, the access to SPI SRAM space (either code execution or data) may be performed by firmware. The AHB transaction initiated by the firmware provides address and other transaction critical parameters to control the data transfer from the external SPI SRAM.

You should exercise care in choosing the correct configuration and command to support the speed of data transfer. The SPIXR Master Controller provides SCK periods as fast as the AHB clock speed divided by two. The external SPI SRAM configuration to support data transfer rates must be performed by the SPIXR Master Controller.

### 7.3.1 *SPIXR Master Controller Registers*

Refer to *Table 2-1: APB Peripheral Base Address Map* for the SPIXR Base Peripheral Address

*Table 7-24. SPIXR Master Controller Register Offsets, Names, Access and Descriptions*

| Offset | Register | Access | Description |
|---|---|---|---|
| [0x0000] | *SPIXR_DATA* | R/W | SPIXR FIFO Data Register |
| [0x0004] | *SPIXR_CTRL1* | R/W | SPIXR Master Signals Control Register |

| Offset | Register | Access | Description |
|---|---|---|---|
| [0x0008] | *SPIXR_CTRL2* | R/W | SPIXR Transmit Packet Size Register |
| [0x000C] | *SPIXR_CTRL3* | R/W | SPIXR Static Configuration Register |
| [0x0010] | *SPIXR_SS_TIME* | R/W | SPIXR Slave Select Timing Register |
| [0x0014] | *SPIXR_BRG_CTRL* | R/W | SPIXR Master Baud Rate Register |
| [0x001C] | *SPIXR_DMA* | R/W | SPIXR DMA Control Register |
| [0x0020] | *SPIXR_INT_FL* | R/W1C | SPIXR Interrupt Status Flags Register |
| [0x0024] | *SPIXR_INT_EN* | R/W | SPIXR Interrupt Enable Register |
| [0x0028] | *SPIXR_WAKE* | R/W1C | SPIXR Wakeup Status Flags Register |
| [0x002C] | *SPIXR_WAKE_E* | R/W | SPIXR Wakeup Enable Register |
| [0x0030] | *SPIXR_STAT* | RO | SPIXR Active Status Register |
| [0x0034] | *SPIXR_XMEM_CTRL* | R/W | SPIXR XMEM Control Register |

### 7.3.2 SPIXR Register Details

*Table 7-25. SPIXR FIFO Data Register*

| SPIXR FIFO Data Register | | | SPIXR_DATA | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | data | R/W | 0 | **SPIXR FIFO Data**<br>FIFO data for the SPIXR. |

*Table 7-26. SPIXR Master Signals Control Register*

| SPIXR Master Signals Control Register | | | SPIXR_CTRL1 | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 23:16 | ss | R/W | 0 | **Master Slave Select**<br>This field selects the slave select pin for the SPIXR interface.<br>0: The slave select pin is not selected for the SPIXR.<br>1: The SPIXR slave select pin is used for the SPIXR.<br>*Note: This field must be set to 1 for SPIXR operation.* |
| 15:9 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 8 | ss_ctrl | R/W | 0 | **Master Slave Select Control**<br>Setting this field to 1 leaves the Slave Select signal asserted at the end of the transmission. This enables multiple transmissions to occur without the Slave Select signal being deasserted. At the completion of all transmissions with the SPIXR device, this field must be set to 0 to deassert the Slave Select line.<br>0: Slave Select is deasserted at the end of a transmission<br>1: Slave Select stays asserted at the end of a transmission |
| 7:6 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| SPIXR Master Signals Control Register | | | SPIXR_CTRL1 | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 5 | start | R/W1AC | 0 | **Master Start Data Transmission**<br>Set this field to 1 to start the transaction with the slave device. Hardware automatically clears this field after the transaction is started.<br>0: No SPIXR data transmission is in process.<br>1: Master initiates a data transmission.<br>*Note: Ensure that all pending transactions are complete before writing a 1.*<br>***Warning**: If the transmit FIFO is enabled, there must be at least one byte in the TX FIFO before setting this bit.* |
| 4 | ss_io | R/W | 0 | **Master Slave Select Signal Direction**<br>This field must be set to 0 for SPIXR operation.<br>0: Slave Select is an output<br>*Note: The SPIXR only operates as a SPI master in single master mode. Writing 1 to this field is invalid.* |
| 3:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | master | R/W | 0 | **SPIXR Master Mode Enable**<br>This field must be set to 1 to use the SPIXR peripheral.<br>1: Master Mode<br>*Note: The SPIXR peripheral only operates in Master Mode. Writing 0 to this field is invalid.* |
| 0 | enable | R/W | 0 | **SPIXR Enable/Disable**<br>Set this field to 1 to enable the SPIXR peripheral.<br>0: SPIXR is disabled.<br>1: SPIXR is enabled.<br>*Note: Setting this field to 0 disables the SPIXR but maintains all registers and the FIFO data.* |

*Table 7-27. SPIXR Transmit Packet Size Register*

| SPIXR Transmit Packet Size Register | | | SPIXR_CTRL2 | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | rx_num_char | R/W | 0 | **Number of characters to receive in RX FIFO**<br>The number of characters in the RX FIFO.<br>*Note: This field is only used if the SPIXR is configured for Three-Wire SPI operation, SPIXR_CTRL3.three_wire = 1.* |
| 15:0 | tx_num_char | R/W | 0 | **Number of characters to transmit from TX FIFO**<br>The number of characters in the TX FIFO.<br>*Note: If the SPIXR is set to Four-wire mode, SPIXR_CTRL3.three_wire = 0, this field represents both the RX and TX FIFO character count.* |

*Table 7-28. SPIXR Static Configuration Register*

| SPIXR Static Configuration Register | | | SPIXR_CTRL3 | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:27 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 16 | sspol | R/W | 0 | **Slave Select Polarity**<br>0: SS is active low<br>1: SS is active high |
| 15 | three_wire | R/W | 0 | **Three-Wire Mode Enable**<br>0: Four-wire mode enabled (Single Mode only)<br>1: Three-wire mode enabled |
| 14 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 13:12 | data_width | R/W | 0 | **SPIXR Data Width**<br>Sets the number of data lines (SDIO pins) for communication.<br>0: 1-data pin (Single Mode)<br>1: 2-data pins (Dual Mode)<br>2: 4-data pins (Quad Mode)<br>3: Reserved for Future Use |
| 11:8 | numbits | R/W | 0 | **Number of Bits per Character**<br>Sets the number of bits per character for an SPIXR transaction. |
| 7:5 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 4 | sck_inv | R/W | 0 | **SCK Inverted**<br>This field must always be set to 0 for SPIXR operation. SCK inversion for a specific mode is not supported by the SPIXR peripheral. Use the *SPIXR_CTRL3*.*cpol* field to set the polarity of the clock for a given mode.<br>0: Normal SCK output.<br>1: Invalid, not supported. |
| 3:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | cpol | R/W | 0 | **Clock Polarity**<br>Sets the SCK clock polarity for the supported modes.<br>0: Normal clock. Use when in SPI Mode 0 and Mode 1<br>1: Inverted clock. Use when in SPI Mode 2 and Mode 3<br>*Note: This field is set depending on the SPI Mode configuration.* |
| 0 | cpha | R/W | 0 | **Clock Phase**<br>Sets the SPIXR SCK clock phase.<br>0: Data sampled on clock rising edge. Use when in SPI Mode 0 and Mode 2<br>1: Data sampled on clock falling edge. Use when in SPI Mode 1 and Mode 3<br>*Note: This field must be set based on the SPI Mode configuration.* |

*Table 7-29. SPIXR Slave Select Timing Register*

| SPIXR Slave Select Timing Register | | | SPIXR_SS_TIME | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 23:16 | ssinact | R/W | 0 | **SS Inactive Clock Delay**<br>This is the time SS is inactive, and the bus is inactive between character transmission.<br>It is the number of system clock cycles from the time a character is transmitted, and SS is inactive to the time SS is active and a new character is transmitted. |
| 15:8 | ssact2 | R/W | 0 | **Slave Select Active After Last SCK**<br>Number of system clock cycles that SS is active from the last SCK edge to when SS is inactive. |
| 7:0 | ssact1 | R/W | 0 | **Slave Select Active Before SCK**<br>Number of system clock cycles between the time SS is asserted until the first SCK edge. |

*Table 7-30. SPIXR Master Baud Rate Generator*

| SPIXR Master Baud Rate Generator Register | | | SPIXR_BRG_CTRL | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:20 | - | R/W | | **Reserved for Future Use**<br>Do not modify this field. |

| SPIXR Master Baud Rate Generator Register | SPIXR_BRG_CTRL | [0x0014] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 19:16 | scale | R/W | 0 | **System Clock to SPIXR Clock Scale Factor**<br>Scales the system clock by 2scale to generate the internal SPIXR peripheral clock.<br>$f_{SPIXR\_CLK} = \frac{f_{SYS\_CLK}}{2^{scale}}$ |
| 15:8 | hi | R/W | 0 | **SCK Hi Clock Cycles Control**<br>Setting this field to 0 disables the high duty cycle control for SCK.<br>Setting this field to any non-zero value sets the high cycle time to:<br>$SCK\_HIGH = hi \times SPIXR\_CLK$<br>*Note: If SPIXR_BRG_CTRL.scale = 0, SPIXR_BRG_CTRL.hi = 0, and SPIXR_BRG_CTRL.lo = 0, character sizes of 2 and 10 bits are not supported.* |
| 7:0 | lo | R/W | 0 | **SCK Low Clock Cycles Control**<br>Setting this field to 0 disables the low duty cycle control for SCK.<br>Setting this field to any non-zero value sets the high cycle time to:<br>$SCK\_LOW = lo \times SPIXR\_CLK$<br>*Note: If SPIXR_BRG_CTRL.scale = 0, SPIXR_BRG_CTRL.hi = 0, and SPIXR_BRG_CTRL.lo = 0, character sizes of 2 and 10 bits are not supported.* |

| scale | $f_{SPIXR\_CLK}$ |
|---|---|
| 0 | $f_{SYS\_CLK}$ |
| 1 | $\frac{f_{SYS\_CLK}}{2^1}$ |
| 2 | $\frac{f_{SYS\_CLK}}{2^2}$ |
| 3 | $\frac{f_{SYS\_CLK}}{2^3}$ |
| 4 | $\frac{f_{SYS\_CLK}}{2^4}$ |
| 5 | $\frac{f_{SYS\_CLK}}{2^5}$ |
| 6 | $\frac{f_{SYS\_CLK}}{2^6}$ |
| 7 | $\frac{f_{SYS\_CLK}}{2^7}$ |
| 8 | $\frac{f_{SYS\_CLK}}{2^8}$ |
| 9 - 15 | Reserved for Future Use |

*Table 7-31. SPIXR DMA Control Register*

| SPIXR DMA Control Register | | | | SPIXR_DMA | [0x001C] |
|---|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** | |
| 31 | rx_dma_en | R/W | 0 | **RX DMA Enable**<br>Enable or disable the RX DMA.<br>0: RX DMA is disabled. Any pending DMA requests are cleared<br>1: RX DMA is enabled | |
| 30 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. | |
| 29:24 | rx_fifo_cnt | RO | 0 | **Number of Bytes in the RX FIFO**<br>Reading this field returns the number of bytes currently in the RX FIFO | |
| 23 | rx_fifo_clear | R/W1O | 0 | **Clear the RX FIFO**<br>Set this field to clear the RX FIFO and all related RX FIFO flags in the *SPIXR_INT_FL* register. When cleared, the *SPIXR_INT_FL.rx_fifo_empty* flag is set by hardware.<br>1: Clear the RX FIFO and any pending RX FIFO flags in *SPIXR_INT_FL*. This should be done when the RX FIFO is inactive.<br>*Note: Writing 0 has no effect.* | |
| 22 | rx_fifo_en | R/W | 0 | **RX FIFO Enabled**<br>Set this field to 1 to enable the RX FIFO.<br>0: RX FIFO disabled<br>1: RX FIFO enabled | |
| 21 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. | |
| 20:16 | rx_fifo_level | R/W | 0 | **RX FIFO Threshold Level**<br>When the RX FIFO has more than this field, a DMA request is triggered, and the *SPIXR_INT_FL.rx_level* interrupt flag is set. Valid values are 0x00 to 0x1E.<br>0x1F is not a valid value. | |
| 15 | tx_dma_en | R/W | 0 | **TX DMA Enable**<br>0: TX DMA is disabled. Any pending DMA requests are cleared.<br>1: TX DMA is enabled | |
| 14 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. | |
| 13:8 | tx_fifo_cnt | RO | 0 | **Number of Bytes in the TX FIFO**<br>Read returns the number of bytes currently in the TX FIFO | |
| 7 | tx_fifo_clear | WO | 0 | **Clear the TX FIFO**<br>Set this field to 1 to clear the TX FIFO and all TX FIFO related flags in the *SPIXR_INT_FL* register. When the TX FIFO is cleared, the *SPIXR_INT_FL.tx_fifo_empty* flag is set by hardware.<br>1: Clear the TX FIFO and any pending TX FIFO flags in *SPIXR_INT_FL*. This should be done when the TX FIFO is inactive.<br>*Note: Writing a 0 has no effect.* | |
| 6 | tx_fifo_en | R/W | 0 | **TX FIFO Enabled**<br>Set to 1 to enable the TX FIFO.<br>0: TX FIFO disabled<br>1: TX FIFO enabled | |
| 5 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. | |

| SPIXR DMA Control Register | | | SPIXR_DMA | [0x001C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4:0 | tx_fifo_level | R/W | 0x10 | **TX FIFO Threshold Level**<br>When the TX FIFO has fewer than this field, a DMA request is triggered and the *SPIXR_INT_FL.tx_level* interrupt flag is set. |

*For all read-only fields, writes have no effect.*

*Table 7-32. SPIXR Interrupt Status Flag Register*

| SPIXR Interrupt Status Flag Register | | | SPIXR_INT_FL | [0x0020] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W1C | | **Reserved for Future Use** |
| 15 | rx_und | R/W1C | | **RX FIFO Underrun Flag**<br>Set when a read is attempted from an empty RX FIFO. |
| 14 | rx_ovr | R/W1C | | **RX FIFO Overrun Flag**<br>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 |
| 13 | tx_und | R/W1C | 0 | **TX FIFO Underrun Flag**<br>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**<br>Set when a write is attempted to a full TX FIFO. |
| 11 | m_done | R/W1C | 0 | **Master Data Transmission Done Flag**<br>Set if SPI is in Master Mode, and all transactions have completed. |
| 10 | - | R/W1C | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 9 | abort | R/W1C | 0 | **Slave Mode Transaction Abort Detected Flag**<br>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**<br>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/W1C | 0 | **Reserved for Future Use**<br>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**<br>Set when the RX FIFO is full. |
| 2 | rx_level | R/W1C | 0 | **RX FIFO Threshold Level Crossed Flag**<br>Set when the RX FIFO exceeds the value in *SPIXR_DMA.rx_fifo_level*. |
| 1 | tx_empty | R/W1C | 1 | **TX FIFO Empty Flag**<br>Set when the TX FIFO is empty. |

| SPIXR Interrupt Status Flag Register | | | SPIXR_INT_FL | [0x0020] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | tx_level | R/W1C | 0 | **TX FIFO Threshold Level Crossed Flag**<br>Set when the TX FIFO is less than the value in *SPIXR_DMA.tx_fifo_level*. |

*Table 7-33. SPIXR Interrupt Enable Register*

| SPIXR Interrupt Enable Register | | | SPIXR_INT_EN | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | rx_und | R/W | 0 | **RX FIFO Underrun Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 14 | rx_ovr | R/W | 0 | **RX FIFO Overrun Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 13 | tx_und | R/W | 0 | **TX FIFO Underrun Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 12 | tx_ovr | R/W | 0 | **TX FIFO Overrun Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 11 | m_done | R/W | 0 | **Master Data Transmission Done Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 10 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 9 | abort | R/W | 0 | **Slave Mode Transaction Abort Detected Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 8 | fault | R/W | 0 | **Multi-Master Fault Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 7:6 | - | R/W | 0 | **Reserved for Future Use**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 5 | ssd | R/W | 0 | **Slave Select Deasserted Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 4 | ssa | R/W | 0 | **Slave Select Asserted Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 3 | rx_full | R/W | 0 | **RX FIFO Full Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |

| SPIXR Interrupt Enable Register | | | SPIXR_INT_EN | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 2 | rx_level | R/W | 0 | **RX FIFO Threshold Level Crossed Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 1 | tx_empty | R/W | 1 | **TX FIFO Empty Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |
| 0 | tx_level | R/W | 0 | **TX FIFO Threshold Level Crossed Interrupt Enable**<br>1: Interrupt enabled<br>0: Interrupt disabled |

*Table 7-34. SPIXR Wakeup Flag Register*

| SPIXR Wakeup Flag Register | | | SPIXR_WAKE_FL | [0x0028] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:4 | - | R/W1C | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3 | rx_full | R/W1C | 0 | **Wake on RX FIFO Full Flag**<br>If set, the RX FIFO Full condition caused the wake event. |
| 2 | rx_level | R/W1C | 0 | **Wake on RX FIFO Threshold Level Crossed Flag**<br>If set, the RX FIFO Threshold Level Crossed condition caused the wake event. |
| 1 | tx_empty | R/W1C | 0 | **Wake on TX FIFO Empty Flag**<br>If set, the TX FIFO empty condition caused the wake event. |
| 0 | tx_level | R/W1C | 0 | **Wake on TX FIFO Threshold Level Crossed Flag**<br>If set, the TX FIFO threshold level crossed caused the wake event. |

*Table 7-35. SPIXR Wakeup Enable Register*

| SPIXR Wakeup Enable Register | | | SPIXR_WAKE_EN | [0x002C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:4 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3 | rx_full | R/W | 0 | **Wake on RX FIFO Full Enable**<br>Set to 1 to wake up the device when this RX FIFO is full.<br>0: Wakeup Disabled for this condition.<br>1: Wakeup Enabled for this condition. |
| 2 | rx_level | R/W | 0 | **Wake on RX FIFO Threshold Level Crossed Enable**<br>Set to 1 to wake up the device when this RX FIFO is full.<br>0: Wakeup Disabled for this condition.<br>1: Wakeup Enabled for this condition. |
| 1 | tx_empty | R/W | 0 | **Wake on TX FIFO Empty Enable**<br>Set to 1 to wake up the device when this RX FIFO is full.<br>0: Wakeup Disabled for this condition.<br>1: Wakeup Enabled for this condition. |

| SPIXR Wakeup Enable Register | | | SPIXR_WAKE_EN | [0x002C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | tx_level | R/W | 0 | **Wake on TX FIFO Threshold Level Crossed Enable**<br>Set to 1 to wake up the device when this RX FIFO is full.<br>0: Wakeup Disabled for this condition.<br>1: Wakeup Enabled for this condition. |

*Table 7-36. SPIXR Active Status Register*

| SPIXR Active Status Register | | | SPIXR_STAT | [0x0030] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:1 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 0 | busy | RO | 0 | **SPI Active Status**<br>This field returns the status of the SPIXR communications. Hardware sets and clears this field automatically when SPI communications are active or complete.<br>0: SPI is not active. Cleared when the last character is sent.<br>1: SPI is active. Set when transmit starts. |

*Table 7-37. SPIXR External Memory Control Register*

| SPIXR External Memory Control Register | | | SPIXR_XMEM_CTRL | [0x0034] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31 | xmem_en | R/W | | **Enable External Memory**<br>0: XMEM disabled<br>1: XMEM enabled |
| 30:24 | - | R/W | | **Reserved for Future Use**<br>Do not modify this field. |
| 23:16 | xmem_dclks | R/W | | **Number of dummy characters between address phase and read data from the external memory.**<br>0: no delay between address and read data<br>1:255 delay number of characters |
| 15:8 | xmem_wr_cmd | R/W | | **Write command to be received at the external memory**<br>Vendor specific value |
| 7:0 | xmem_rd_cmd | R/W | | **Read command to be received at the external memory**<br>Vendor specific value |

## 7.4 External Memory Cache Controller (EMCC)

The External Memory Cache Controller is a AHB block that has multiple interfaces. The address and data interface is connected to the AHB and the EMCC register interface is connected via the APB.

The EMCC is a 16KB 2-way set-associative, LRU replacement policy, write-through implementation used for caching instructions and data from an external SPI-XiP RAM or HyperBus/Xccela Bus device. The EMCC includes tag RAM, cache RAM and a line fill buffer as shown in *Figure 3-6*. Write allocate and critical word first are options controlled by the application. Each cache line is 256-bits wide with the lower 5-bits of the address used as the cache line index. The EMCC uses tag cache RAM with 8-bits of the address index and a 5-bit line offset to access the tag cache RAM. 16-bits of the address are stored in tag RAM for hit/miss checking enabling the EMCC to access up to 512MB of external memory. If using the HyperBus/Xccela Bus interface, the EMCC interfaces to the address range of 0x6000 0000 to 0x7FFF FFFF for a maximum of 512MB memory. When using the SPI XiP RAM interface, the EMCC interfaces to the address range of 0x8000 0000 to 0x9FFF FFFF for a maximum of 512MB external.

### 7.4.1 *Features*

- 2-way set associative, LRU (Least-Recently Used) replacement policy
- Write-no-allocate with option to Write-allocate
- Write-through
- Read critical word first and streaming
- 512MB addressable range
- 16KB size

### 7.4.2 *Enabling the EMCC*

Enable the EMCC as follows:

1. Set the *GCR_SCON*.*dcache_dis* field to 0.
2. Set the *EMCC_CACHE_CTRL*.*enable* field to 1.

Once enabled, the cache is empty and begins filling when a read from or write to (if write allocate is enabled) the external memory is performed.

After a Power-On-Reset event, the cache tag RAM is cleared by hardware ensuring that no corrupted data is accessed from the initial cache read.

### 7.4.3 *Disabling the EMCC*

Disabling the EMCC cache automatically invalidates the cache contents. All access to the external memory while the EMCC cache is disabled are performed using the Line Buffer.

Disable the EMCC by setting *EMCC_CACHE_CTRL*.*enable* to 0.

The EMCC cache and Line Buffer can both be bypassed by setting *GCR_SCON*.*dcache_dis* to 1. Bypassing the EMCC enables direct access to the external memory from the application firmware.

### 7.4.4 *EMCC Registers*

Refer to *Table 2-1: APB Peripheral Base Address Map* for the EMCC Base Peripheral Address.

*Table 7-38: External Memory Cache Controller Register Addresses and Descriptions*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *EMCC_CACHE_ID* | RO | Cache ID Register |
| [0x0004] | *EMCC_MEM_SIZE* | RO | Cache Memory Size Register |
| [0x0100] | *EMCC_CACHE_CTRL* | R/W | Cache Control Register |
| [0x0700] | *EMCC_INVALIDATE* | WO | Invalidate Register |

### 7.4.5 *EMCC Register Details*

*Table 7-39: EMCC Cache ID Register*

| **EMCC Cache ID Register** | | | **EMCC_CACHE_ID** | **[0x0000]** |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | RO | - | **Reserved for Future Use**<br>Do not modify this field. |
| 15:10 | cchid | RO | - | **Cache ID**<br>Returns the Cache ID for this Cache instance. |
| 9:6 | partnum | RO | - | **Cache Part Number**<br>Returns the part number indicator for this Cache instance. |
| 5:0 | relnum | RO | - | **Cache Release Number**<br>Returns the release number for this Cache instance. |

*Table 7-40: EMCC Memory Size Register*

| **EMCC Memory Size Register** | | | **EMCC_MEM_SIZE** | **[0x0004]** |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | memsz | RO | - | **Addressable Memory Size**<br>Indicates the size of addressable memory by this cache controller instance in 128KB units. |
| 15:0 | cchsz | RO | - | **Cache Size**<br>Returns the size of the cache RAM memory in 1KB units.<br>16: 16KB Cache RAM |

*Table 7-41: EMCC Cache Control Register*

| **EMCC Cache Control Register** | | | **EMCC_CACHE_CTRL** | **[0x0100]** |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | - | **Reserved for Future Use**<br>Do not modify this field. |

| EMCC Cache Control Register | | | EMCC_CACHE_CTRL | [0x0100] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 16 | ready | RO | - | **Ready**<br>This field is cleared by hardware anytime the cache as a whole is invalidated (including a Power-On Reset event). Hardware automatically sets this field to 1 when the invalidate operation is complete and the cache is ready.<br>0: Cache Invalidate in process.<br>1: Cache is ready.<br>*Note: While this field reads 0, the cache is bypassed and reads come directly from the line fill buffer.* |
| 15:3 | - | R/W | - | **Reserved for Future Use**<br>Do not modify this field. |
| 2 | cwfst_dis | R/W | 0 | **Critical Word First (CWF) Disable**<br>Setting this field to 1 disables Critical Word First operation. When CWF is disabled, the cache fills the cache line before sending the data to the Arm Cortex core. When CWF is enabled, any data fetch that results in a cache miss immediately sends the data read to the Arm Cortex core prior to filling the cache line.<br>0: Enable Critical Word First.<br>1: Critical Word First Disabled.<br>*Note: This field is only writable when the EMCC is disabled (EMCC_CACHE_CTRL.enable = 0).* |
| 1 | write_alloc | R/W | 0 | **Write Allocate Enable**<br>Set this field to enable write allocate for the cache. When this is enabled, writes to the memory update the external memory and the cache line associated with the write is filled from the external memory. Disabling write allocate, default mode, performs a write to the external memory on any write operation, but the associated cache line is not refilled. When disabled, writes to successive memory locations are more efficient.<br>0: Write allocate disabled (default)<br>1: Write allocate enabled.<br>*Note: The EMCC is a write-through cache resulting in any write to the external memory performing an immediate write to the external device.* |
| 0 | enable | R/W | 0 | **Enable**<br>Set this field to 1 to enable the cache. Setting this field to 0 automatically invalidates the cache contents. When this cache is disabled, reads are handled by the line fill buffer.<br>0: Disable Cache<br>1: Enable Cache |

*Table 7-42: EMCC Invalidate Register*

| EMCC Invalidate Register | | | EMCC_INVALIDATE | [0x0700] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | - | WO | - | **Invalidate**<br>Any write to this register of any value invalidates the cache. |

## 7.5 Secure Digital Host Controller

### 7.5.1 *Overview*

The Secure Digital Host Controller (SDHC) provides an interface between the AHB and Embedded MultiMediaCards (e.MMCs), Secure Digital I/O (SDIO) cards, Standard Capacity SD Memory Cards and High-Capacity SD Memory Cards. The SDHC handles the SDIO/SD protocol at the transmission level, packing data, adding cyclic redundancy check (CRC), Start/End bit, and checking for transaction format correctness. Details of the SD communication and protocol are not part of the scope of this document. The MAX32650–MAX32652 SDHC only supports a single embedded SD card.

SD memory card and SDIO card specifications are available at *https://www.sdcard.org*.

The e.MMC specifications are available from JEDEC at *http://www.jedec.org*.

### 7.5.2 *Features*

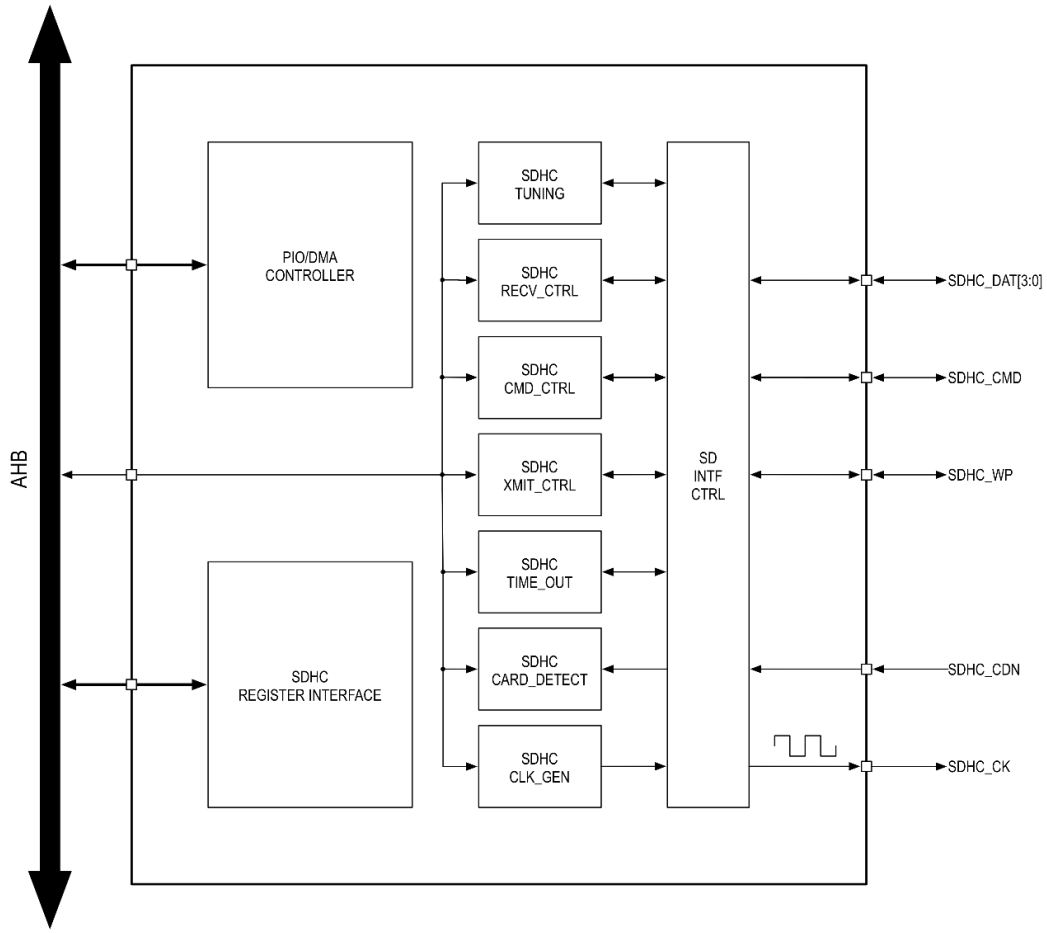
*Compliance*

- SD Host Controller Standard Specification Version 3.00
- SDIO Card Specification Version 3.0
- SD Memory Card Specification Version 3.01
- SD Memory Card Security Specification version 1.01
- e.MMC Specification version 4.51

*SD/SDIO Card Interface*

- Supports SDR50 with SDHC clock of up to 60MHz (30MB/sec)
- Supports DDR50 with SDHC clock of up to30MHz (30MB/sec)
- Designed to work with I/O cards, Read-Only cards, and Read/Write cards
- 1-bit and 4-bit data transfers in SD modes and SPI mode
- Double buffer for transfers configurable from 512B to 1KB.
- Auto Command (AutoCMD12 or AutoCMD23) support
- Multi-block transfers
- Variable-length data transfers
- Default and high-speed mode transfers
- Card insertion/removal events
- Read Wait Control, Suspend/Resume operation
- CRC7 for command and CRC16 for data integrity
- Single Operation DMA (SDMA) for data transfer
- Advanced DMA (ADMA) support

*Figure 7-6: SDHC Block Diagram*




### 7.5.3 *Signals and Pins*

The MAX32650–MAX32652 SDHC pin mapping for the 144-pin TQFP and 96-WLP mapped to the SD Host Controller Standard Specification Version 3.0 are shown in *Table 7-43*, *below*.

*Table 7-43: SDHC Alternate Function Mapping to SDHC Specification Pin Names*

| MAX32650–MAX32652 Alternate Function | MAX32650–MAX32652 Alternate Function Number | MAX32650–MAX32652 144-TQFP Pin Name | MAX32650–MAX32652 96-WLP Pin Name | SDHC Specification Pin Name | Direction | Signal Description |
|---|---|---|---|---|---|---|
| SDHC_CDN | AF2 | P0.31 | NA | SDCD# | I | Card present, active low. |
| SDHC_CLK | AF1 | P1.5 | P1.5 | SDCLK | O | SD clock signal. |
| SDHC_WP | AF1 | P1.2 | P1.2 | SDWP | I | Write protect signal, active high. |

| MAX32650–MAX32652 Alternate Function | MAX32650–MAX32652 Alternate Function Number | MAX32650–MAX32652 144-TQFP Pin Name | MAX32650–MAX32652 96-WLP Pin Name | SDHC Specification Pin Name | Direction | Signal Description |
|---|---|---|---|---|---|---|
| SDHC_CMD | AF1 | P1.0 | P1.0 | CMD | I/O | SD bus command signal. |
| SDHC_DAT0 | AF1 | P1.4 | P1.4 | DAT[0] | I/O | SD data bus bit 0. |
| SDHC_DAT1 | AF1 | P1.6 | P1.6 | DAT[1] | I/O | SD data bus bit 1. |
| SDHC_DAT2 | AF1 | P1.1 | P1.1 | DAT[2] | I/O | SD data bus bit 2. |
| SDHC_DAT3 | AF1 | P1.3 | P1.3 | DAT[3] | I/O | SD data bus bit 3. |

Perform the following steps to configure the GPIO for SDHC peripheral usage:

1. If using the 144-pin TQFP, set GPIO0_EN[31] to 0 and set GPIO0_AF_SEL[31] to 1, enabling P0.31 for alternate function and setting it to AF2.
2. Enable the alternate function for pins P1.0 through P1.6 by setting GPIO1_EN[0:6] to 0.
3. Set Alternate Function 1 by setting GPIO1_AF_SEL[0:6] to 0.

### 7.5.4 SDHC Peripheral Clock Selection

The input clock to the SDHC peripheral is driven by the high speed system oscillator always, 120MHz. This 120MHz input clock is either divided by 2 (default) or by 4 to drive the SDHC peripheral. Set the SDHC peripheral clock divisor using the *GCR_PCLK_DIV.sdhcfrq* bit as shown

*Equation 7-1: SDHC Peripheral Clock*

$$f_{SDHC\_CLK} = \frac{120MHz}{2^{GCR\_PCLK\_DIV.sdhcfrq}}$$

### 7.5.5 Usage

Communication over the SD bus is based on command and data bit streams/blocks that are initiated by a start bit and terminated by a stop bit.

- **Command**: A command is a token that starts an operation and is sent by the SDHC to the card in the embedded card slot. A command is transferred serially using the *SDHC_CMD* pin.
- **Response**: A response is a token sent from the card to the SDHC in response to a previously received command and is transferred serially using the *SDHC_CMD* pin.
- **Data**: You can transfer data from the card to the SDHC or vice versa using the SDHC_DAT[3:0] pins.

*Figure 7-7*, *Figure 7-8*, and *Figure 7-9* show the basic types of SD operations as described in the Physical Layer Simplified Specification Version 6.00 from the SD Card Association.

*Figure 7-7: SD Bus Protocol - No Response and No Data Operations*

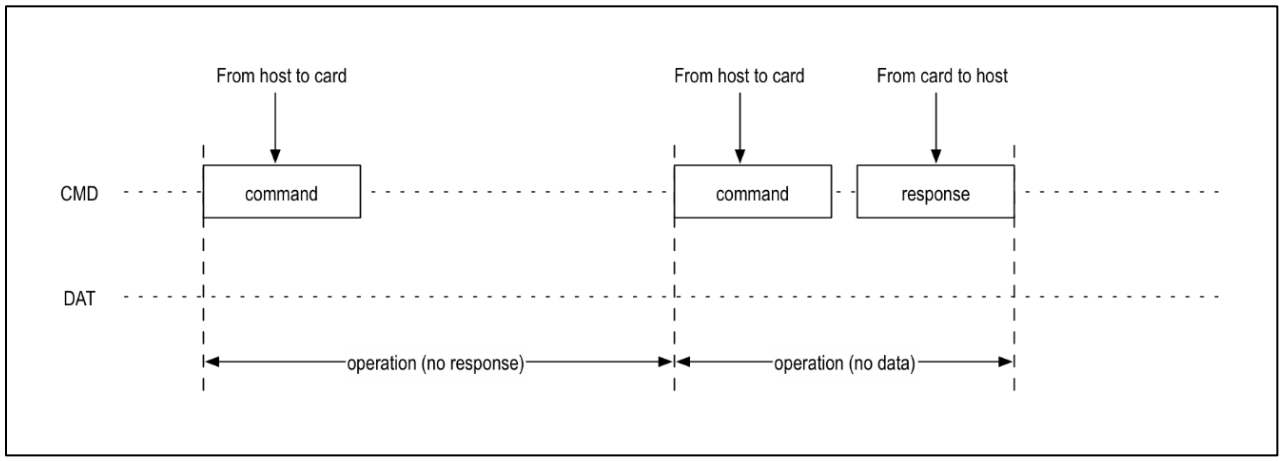



*Figure 7-8: SD Bus Protocol - Multi-Block Read Operation*

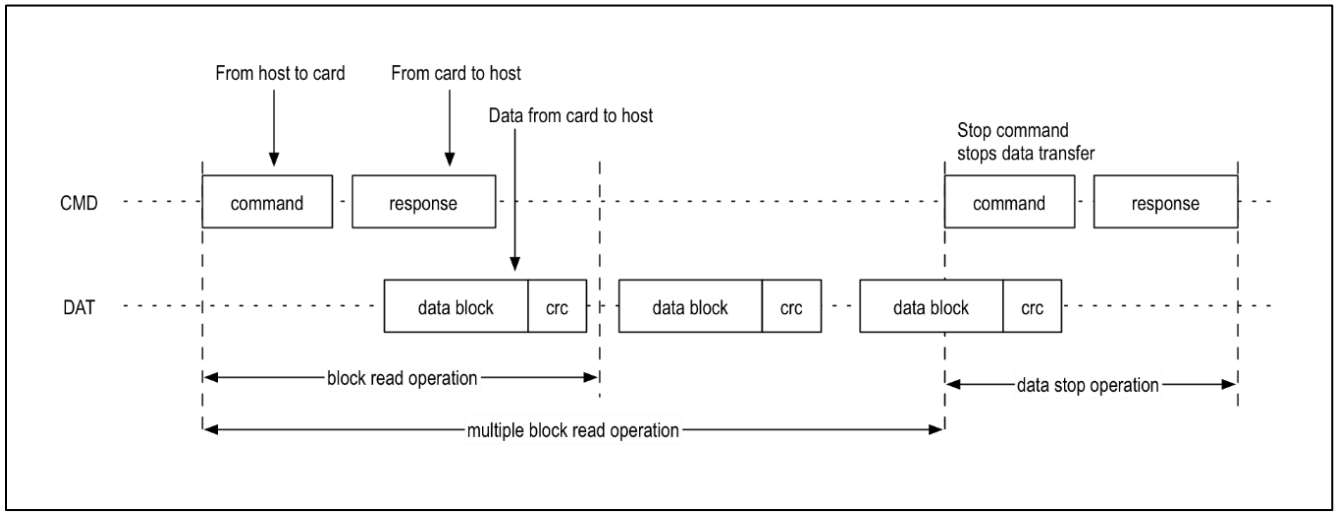
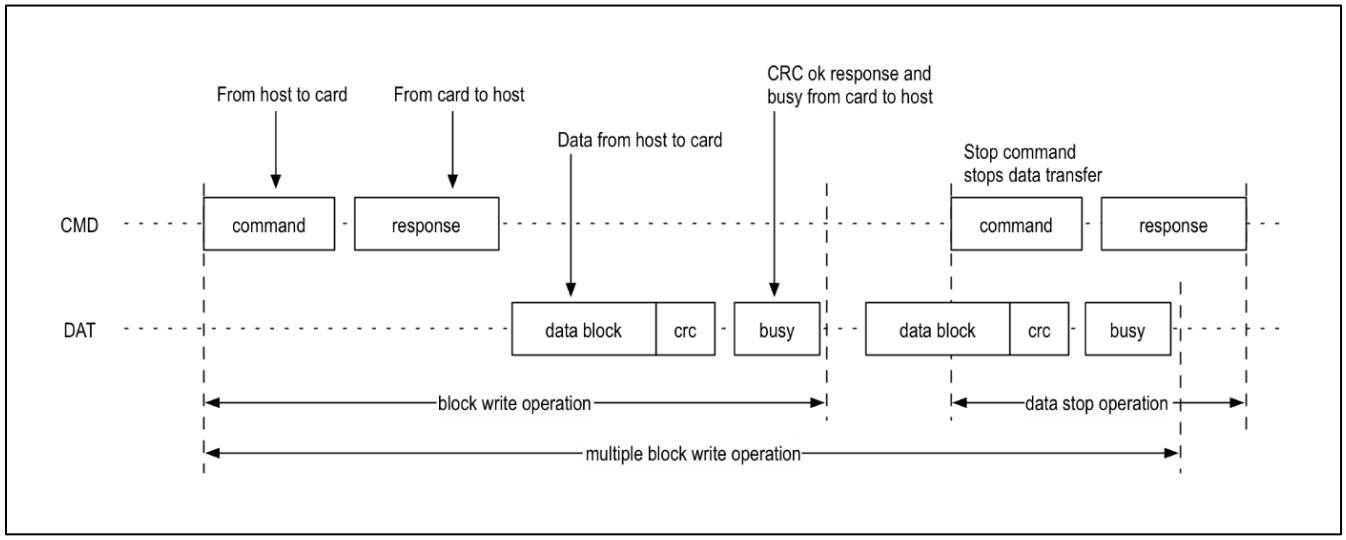


*Figure 7-9: SD Bus Protocol - Multi Block Write Operation*




### 7.5.6 SD Command Generation

*Table 7-44* shows the registers required for three transaction types: SDMA generated transactions, ADMA generated transactions, and CPU transactions (includes data transfers and Non-DAT transfers). When initiating a transaction, you should program the registers sequentially starting with the *SDHC_SDMA* register and finishing with the *SDHC_CMD* register. When the upper byte of the *SDHC_CMD* register is written, it triggers the SDHC to issue the SD command.

*Table 7-44: Registers Used to Generate SD Commands*

| Register | SDMA Command | ADMA Command | CPU Data Transfer | Non-DAT (No Data) Transfer |
|---|---|---|---|---|
| SDMA System Address / Argument 2<br>*SDHC_SDMA* | Yes/No | No/Auto CMD23 | No/AutoCMD23 | No/No |
| Block Size<br>*SDHC_BLK_SIZE* | Yes | Yes | Yes | No (Protected) |
| Block Count<br>*SDHC_BLK_CNT* | Yes | Yes | Yes | No (Protected) |
| Argument 2<br>*SDHC_SDMA* | Yes | Yes | Yes | No (Protected) |
| Command<br>*SDHC_CMD* | Yes | Yes | Yes | Yes |

### 7.5.7 SDHC Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the SDHC Base Peripheral Address

*Table 7-45: SDHC Register Offsets, Names and Descriptions*

| Offset | Register Name | Description |
|---|---|---|
| [0x0000] | *SDHC_SDMA* | SDMA System Address / Argument 2 |
| [0x0004] | *SDHC_BLK_SIZE* | Block Size register |
| [0x0006] | *SDHC_BLK_CNT* | Block Count register |

| Offset | Register Name | Description |
|---|---|---|
| [0x0008] | *SDHC_ARG_1* | Argument 1 register |
| [0x000C] | *SDHC_TRANS* | Transfer Mode register |
| [0x000E] | *SDHC_CMD* | Command register |
| [0x0010] | *SDHC_RESP_0* | Response register 0 |
| [0x0012] | *SDHC_RESP_1* | Response register 1 |
| [0x0014] | *SDHC_RESP_2* | Response register 2 |
| [0x0016] | *SDHC_RESP_3* | Response register 3 |
| [0x0018] | *SDHC_RESP_4* | Response register 4 |
| [0x001A] | *SDHC_RESP_5* | Response register 5 |
| [0x001C] | *SDHC_RESP_6* | Response register 6 |
| [0x001E] | *SDHC_RESP_7* | Response register 7 |
| [0x0020] | *SDHC_BUFFER* | Buffer Data Port register |
| [0x0024] | *SDHC_PRESENT* | Present State register |
| [0x0028] | *SDHC_HOST_CN_1* | Host Control 1 register |
| [0x0029] | *SDHC_PWR* | Power Control register |
| [0x002A] | *SDHC_BLK_GAP* | Block Gap Control register |
| [0x002B] | *SDHC_WAKEUP* | Wakeup Control register |
| [0x002C] | *SDHC_CLK_CN* | Clock Control register |
| [0x002E] | *SDHC_TO* | Timeout Control register |
| [0x002F] | *SDHC_SW_RESET* | Software Reset register |
| [0x0030] | *SDHC_INT_STAT* | Normal Interrupt Status register |
| [0x0032] | *SDHC_ER_INT_STAT* | Error Interrupt Status register |
| [0x0034] | *SDHC_INT_EN* | Normal Interrupt Status Enable register |
| [0x0036] | *SDHC_ER_INT_EN* | Error Interrupt Status Enable register |
| [0x0038] | *SDHC_INT_SIGNAL* | Normal Interrupt Signal Enable register |
| [0x003A] | *SDHC_ER_INT_SIGNAL* | Error Interrupt Signal Enable register |
| [0x003C] | *SDHC_AUTO_CMD_ER* | Auto CMD Error Status register |
| [0x003E] | *SDHC_HOST_CN_2* | Host Control 2 register |
| [0x0040] | *SDHC_CFG_0* | Capabilities register 0 |
| [0x0044] | *SDHC_CFG_1* | Capabilities register 1 |
| [0x0048] | *SDHC_MAX_CURR_CFG* | Maximum Current Capabilities register |
| [0x0050] | *SDHC_FORCE_CMD* | Force Event Register for Auto CMD Error Status |
| [0x0052] | *SDHC_FORCE_EVENT_INT_STAT* | Force Event Register for Error Interrupt Status |
| [0x0054] | *SDHC_ADMA_ER* | ADMA Error Status register |
| [0x0058] | *SDHC_ADMA_ADDR_0* | ADMA System Address register 0 |
| [0x005C] | *SDHC_ADMA_ADDR_1* | ADMA System Address register 1 |

| Offset | Register Name | Description |
|---|---|---|
| [0x0060] | *SDHC_PRESET_0* | Preset Value for Initialization |
| [0x0062] | *SDHC_PRESET_1* | Preset Value for Default Speed |
| [0x0064] | *SDHC_PRESET_2* | Preset Value for High Speed |
| [0x0066] | *SDHC_PRESET_3* | Preset Value for SDR12 |
| [0x0068] | *SDHC_PRESET_4* | Preset Value for SDR25 |
| [0x006A] | *SDHC_PRESET_5* | Preset Value for SDR50 |
| [0x006C] | *SDHC_PRESET_6* | Preset Value for SDR104 |
| [0x006E] | *SDHC_PRESET_7* | Preset Value for DDR50 |
| [0x00FC] | *SDHC_SLOT_INT* | Slot Interrupt Status register |
| [0x00FE] | *SDHC_HOST_CN_VER* | Host Controller Version register |

### 7.5.8 SDHC Register Details

*Table 7-46: SDHC SDMA System Address / Argument Register*

| SDMA System Address / Argument 2 Register | | | SDHC_SDMA | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | addr | R/W | 0 | **SDMA System Address**<br>This register is the address of the buffer used for a SDMA transfer. You must set this register to a valid data buffer address prior to starting an SDMA transfer. A SDHC DMA interrupt (*SDHC_INT_STAT*.dma = 1) is generated if the total size of the SDMA transfer exceeds the Host SDMA Buffer Size (*SDHC_BLK_SIZE.host_buf*). The card driver must update the SDMA System Address (*SDHC_SDMA*) with the address of the next data to transfer and clear the SDHC DMA interrupt flag prior to the transfer resuming.<br>When the SDMA transfer is complete, this register contains the address of the next contiguous data address.<br>When resuming a SDMA transfer, using the Resume command or by setting the *SDHC_BLK_GAP.gap_cont* bit to 1, the SDHC resumes using the address in this register for the data to transfer.<br>Reading this register during a SDMA transfer might return an invalid value unless the transfer is paused as the result of a SDHC DMA interrupt. This field is not used for ADMA transfers.<br>**Argument 2**<br>This register is used with Auto CMD23 to set a 32-bit block count value to the argument of CMD23 while executing Auto CMD23.<br>If Auto CMD23 is used with ADMA, then the full 32-bit block count value is used. If Auto CMD23 is used without AMDA, the available block count value is limited by the *SDHC_BLK_GAP* register to 65,535 blocks. |

*Table 7-47: SDHC SDMA Block Size Register*

| SDMA Block Size Register | | | SDHC_BLK_SIZE | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:15 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| SDMA Block Size Register | SDHC_BLK_SIZE | [0x0004] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 14:12 | host_buf | R/W | | **Host SDMA Buffer Size**<br>This field specifies the size of the contiguous buffer in the system memory for SDMA transfers. SDMA transfers larger than this buffer generates a SDHC DMA interrupt (*SDHC_INT_STAT.dma*) when the transfer reaches the *host_buf* size boundary. The SDMA transfer pauses until the card driver updates the SDMA System Address (*SDHC_SDMA*) register with the next buffer address to transfer and clears the SDHC DMA interrupt flag. When the SDMA transfer is complete, a SDHC transfer complete interrupt (*SDHC_INT_STAT.trans_comp* = 1) is generated. The SDHC DMA interrupt flag is not set when the SDMA transfer completes.<br>**_host_buf_ Value** / **Host SDMA Buffer Size (KB)**<br>0b000 — 4<br>0b001 — 8<br>0b010 — 16<br>0b011 — 32<br>0b100 — 64<br>0b101 — 128<br>0b110 — 256<br>0b111 — 512<br>*Note: This field is used for SDMA transfers only.* |
| 11:0 | trans | R/W | 0x0200 | **Data Transfer Block Size**<br>Sets the block size of data transfers for CMD17, CMD18, CMD24, CMD25, and CMD53. You can set values ranging from 1 up to the maximum buffer size. Setting this field to 0 indicates there is no data to transfer.<br>During a transfer, reading this field might return an invalid value, and writes to this field are ignored.<br>**trans Value** / **Block Size in Bytes**<br>0x0800 — 2,048<br>0x07FF — 2,047<br>... — ...<br>0x200 — 512<br>0x01FF — 511<br>... — ...<br>0x0004 — 4<br>0x0003 — 3<br>0x0002 — 2<br>0x0001 — 1<br>0x0000 — No data transfer |

*Table 7-48: SDHC SDMA Block Count Register*

| SDMA Block Count Register | | | SDHC_BLK_CNT | [0x0006] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15:0 | trans | R/W | 0x0200 | **Current Transfer Block Count**<br>Set to the total number of blocks to transfer prior to a block transfer operation. Set the Block Count Enable (*SDHC_TRANS.blk_cnt_en*) bit to 1 for a block transfer. If Block Count Enable is clear, then this field is unused.<br>When set to 1, the value in this register is the total number of blocks to transfer. After each block transfer, this register is decremented by 1, and stops when the count reaches 0.<br>Reads from this register are only valid when no transactions are active. A setting of 0 results in no blocks transferred.<br>When a Suspend command is complete, the number of remaining blocks to transfer is contained in this field.<br>Before issuing a Resume command, the card driver must restore the previously-saved block count to this field. |

| ***trans* Value** | **Block Count** |
|---|---|
| 0xFFFF | 65,535 |
| 0xFFFE | 65,534 |
| …. | …. |
| 0x0002 | 2 |
| 0x0001 | 1 |
| 0x0000 | Stop count or no block transfer |

*Table 7-49: SDHC SDMA Argument 1 Register*

| SDMA Argument 1 Register | | | SDHC_ARG_1 | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | cmd | R/W | 0 | **SD Command Argument 1**<br>The SD Command Argument 1 is specified as bit [39:8] of the Command-Format in the Physical Layer Specification. |

*Table 7-50: SDHC SDMA Transfer Mode Register*

| SDMA Transfer Mode Register | | | SDHC_TRANS | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:6 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| SDMA Transfer Mode Register | | SDHC_TRANS | | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 5 | multi | R/W | 0 | **Multi/Single Block Select**<br>Used for DAT line transfers and multiple-block commands. For all other commands, set this bit to 0.<br>1: Multiple-block or DAT line transfer<br>0: Single Block<br>*Note: The SDHC_BLK_CNT register is ignored if this field is set to 0.* |
| 4 | read_write | R/W | 0 | **Data Transfer Direction Select**<br>Sets the direction for DAT line data transfers. Set to 1 to transfer data from the SD card to the SDHC (Read). For all other commands, set this bit to 0 (Write).<br>1: Read (from card to host)<br>0: Write (from host to card) |
| 3:2 | auto_cmd_en | R/W | 0 | **Auto CMD Enable / Function Selection**<br>0b00: Auto Command Disabled<br>0b01: Auto CMD12 Enable<br>0b10: Auto CMD23 Enable<br>0b11: Reserved for Future Use<br>**Auto CMD12 Enable**<br>When auto_cmd_en is set to 1, the SDHC issues CMD12 automatically after completion of the last block transfer. If an error occurs from Auto CMD12, then the error is saved to the SDHC_AUTO_CMD_ER register.<br>*Note: Do not set to 1 if an Auto CMD12 is not required.*<br>**Auto CMD23 Enable**<br>When this bit field is set to 0b10, the Host Controller issues a CMD23 automatically before issuing the command specified in the SDHC_CMD (Command) register. The following conditions are required to use Auto CMD23:<br>• Auto CMD23 support (Host Controller Version is 3.00 or later)<br>• A memory card that supports CMD23 (SCR[33] = 1)<br>• If using DMA, ADMA mode only<br>• Only when CMD18 or CMD25 is issued<br>You can use Auto CMD23 with or without ADMA. By writing to the Command register, the SDHC issues a CMD23 first, and then issues the command specified by the Command Index (*SDHC_CMD.idx*) in the Command register. If response errors are detected from CMD23, then the second command is not issued. A CMD23 error is indicated in the Auto CMD Error Status register (SDHC_AUTO_CMD_ER).<br>The 32-bit block count value for CMD23 is set to the SDMA System Address / Argument 2 register (SDHC_SDMA).<br>*Note: The SDHC does not check the command index.* |
| 1 | blk_cnt_en | R/W | 0 | **Block Count Enable**<br>Set to enable the Block Count register (SDHC_BLK_CNT) for multiple block transfers. When this bit is 0, the Block Count register (SDHC_BLK_CNT) is disabled, which is useful if executing an infinite transfer.<br>1: Enable SDHC_BLK_CNT register<br>0: Disable SDHC_BLK_CNT register |
| 0 | dma_en | R/W | 0 | **DMA Enable**<br>Enables DMA functionality per the Capabilities register.<br>If this bit is set to 1, a DMA operation begins when the card driver writes to the upper byte of the Command register (SDHC_CMD).<br>1: DMA mode is enabled as specified in the SDHC_HOST_CN_1.*dma_select* field.<br>0: DMA mode disabled. |

*Table 7-51: Summary of how register settings determine type of data transfer*

| Multi/Single Block Select *SDHC_TRANS.multi* | Block Count Enable *SDHC_TRANS.blk_cnt_en* | Block Count *SDHC_BLK_CNT.trans* | Function |
|---|---|---|---|
| 0 | N.A. | N.A. | Single transfer |
| 1 | 0 | N.A. | Infinite transfer |
| 1 | 1 | ≠0 | Multiple transfer |
| 1 | 1 | 0 | Stop Multiple transfer |

*Table 7-52: SDHC Command Register*

| Command Register | | | SDHC_CMD | [0x000E] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:6 | - | R/W | NA | **Reserved for Future Use**<br>Do not modify this field. |
| 13:8 | idx | R/W | 0 | **Command Index**<br>Valid command number (CMD0-63, ACMD0-63) per the SD Physical Specification and SDIO Card Specification. |
| 7:6 | type | R/W | 0 | **Command Type**<br>The following table lists the values for this field, the type of command, and provides notes about what the command type is typically used for:<br>*type* Value / Command Type / Notes<br>0b11 / Abort / CMD12, CMD52 for writing I/O Abort in CCCR.<br>0b10 / Resume / CMD52 for writing Function Select in CCCR.<br>0b01 / Suspend / CMD52 for writing Bus Suspend in CCCR.<br>0b00 / Normal / Other commands |
| 5 | data_pres_sel | R/W | 0 | **Data Present Select**<br>1: Set to indicate data is present and transferable using the DAT line.<br>0: Commands that only use the CMD line (for example, CMD52), commands with no data transfer but are using the busy signal on SDHC_DAT[0], or a Resume command. |
| 4 | idx_chk_en | R/W | 0 | **Command Index Check Enable**<br>1: SDHC checks the index field in the response and sets a Command Index Error if it does not match the value in the *SDHC_CMD.idx* field.<br>0: Index of response is not checked. |
| 3 | crc_chk_en | R/W | 0 | **Command CRC Check Enable**<br>1: SDHC verifies the CRC field in the response, and if an error is detected, it is reported as a Command CRC Error.<br>0: CRC not checked by hardware. |
| 2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| Command Register | | | SDHC_CMD | [0x000E] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 1:0 | resp_type | R/W | 0 | **Response Type Select**<br>0b00: No Response<br>0b01: Response Length 136<br>0b10: Response Length 48<br>0b11: Response Length 48, and check if busy after response |

*Table 7-53: Relationship between Parameters and the Name of Response Type*

| Response Type<br>*SDHC_CMD.resp_type* | Index Check Enable<br>*SDHC_CMD.idx_chk_en* | CRC Check Enable<br>*SDHC_CMD.crc_chk_en* | Name of Response Type |
|---|---|---|---|
| 0b00 | 0 | 0 | No Response |
| 0b01 | 0 | 1 | R2 |
| 0b10 | 0 | 0 | R3, R4 |
| 0b10 | 1 | 1 | R1, R5, R6, R7 |
| 0b11 | 1 | 1 | R1b, R5b |

*Table 7-54: SDHC Response 0 Register*

| Response 0 Register | | | SDHC_RESP_0 | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | cmd_resp | RO | 0 | **Response Register 0**<br>Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 7-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32650–MAX32652. *Table 7-63* shows the SD types of response mapped to the MAX32650–MAX32652 Response registers. |

*Table 7-55: SDHC Response 1 Register*

| Response 1 Register | | | SDHC_RESP_1 | [0x0012] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | cmd_resp | RO | 0 | **Response Register 1**<br>Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 7-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32650–MAX32652. *Table 7-63* shows the SD types of response mapped to the MAX32650–MAX32652 Response registers. |

*Table 7-56: SDHC Response 2 Register*

| Response 2 Register | | | SDHC_RESP_2 | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | cmd_resp | RO | 0 | **Response Register 2**<br>Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 7-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32650–MAX32652. Table 7-63 shows the SD types of response mapped to the MAX32650–MAX32652 Response registers. |

*Table 7-57: SDHC Response 3 Register*

| Response 3 Register | | | SDHC_RESP_3 | [0x0016] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | cmd_resp | RO | 0 | **Response Register 3**<br>Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 7-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32650–MAX32652. *Table 7-63* shows the SD types of response mapped to the MAX32650–MAX32652 Response registers. |

*Table 7-58: SDHC Response 4 Register*

| Response 4 Register | | | SDHC_RESP_4 | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | cmd_resp | RO | 0 | **Response Register 4**<br>Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 7-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32650–MAX32652. *Table 7-63* shows the SD types of response mapped to the MAX32650–MAX32652 Response registers. |

*Table 7-59: SDHC Response 5 Register*

| Response 5 Register | | | SDHC_RESP_5 | [0x001A] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | cmd_resp | RO | 0 | **Response Register 5**<br>Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 7-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32650–MAX32652. *Table 7-63* shows the SD types of response mapped to the MAX32650–MAX32652 Response registers. |

*Table 7-60: SDHC Response 6 Register*

| Response 6 Register | | SDHC_RESP_6 | | [0x001C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | cmd_resp | RO | 0 | **Response Register 6**<br>Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 7-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32650–MAX32652. *Table 7-63* shows the SD types of response mapped to the MAX32650–MAX32652 Response registers. |

*Table 7-61: SDHC Response 7 Register*

| Response 7 Register | | SDHC_RESP_7 | | [0x001E] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | cmd_resp | RO | 0 | **Response Register 7**<br>Response 7 to Response 0 registers are referenced as a contiguous, single register in the SD Host Controller Spec V3.0. *Table 7-62* shows the mapping from the Response Registers to the SD Host Controller Standard Specification REP[127:0] notation for the MAX32650–MAX32652. *Table 7-63* shows the SD types of response mapped to the MAX32650–MAX32652 Response registers. |

*Table 7-62: SDHC Response Register Mapping to SD Host Controller Response Register Convention*

| Register | Register Name | Register Offset | SDHC REP[] Bit Mapping |
|---|---|---|---|
| *SDHC_RESP_0* | Response 0 | 0x10 | REP[15:0] |
| *SDHC_RESP_1* | Response 1 | 0x12 | REP[31:16] |
| *SDHC_RESP_2* | Response 2 | 0x14 | REP[47:32] |
| *SDHC_RESP_3* | Response 3 | 0x16 | REP[63:48] |
| *SDHC_RESP_4* | Response 4 | 0x18 | REP[79:64] |
| *SDHC_RESP_5* | Response 5 | 0x1A | REP[95:80] |
| *SDHC_RESP_6* | Response 6 | 0x1C | REP[111:96] |
| *SDHC_RESP_7* | Response 7 | 0x1E | REP[127:112] |

*Table 7-63: Kind of SD Card Response Mapping to SDHC Response Registers*

| Kind of Response | Meaning of Response | REP[] Specification Mapping | SDHC Response Register MSW | SDHC Response Register LSW |
|---|---|---|---|---|
| R1, R1b (normal response) | Card Status | REP[31:0] | *SDHC_RESP_1* | *SDHC_RESP_0* |
| R1b (Auto CMD12 response) | Card Status for Auto CMD12 | REP[127:96] | *SDHC_RESP_7* | *SDHC_RESP_6* |
| R1 (Auto CMD23 response) | Card Status for Auto CMD23 | REP[127:96] | *SDHC_RESP_7* | *SDHC_RESP_6* |
| R2 (CID, CSD register) | CID or CSD reg. incl. | REP [119:0] | *SDHC_RESP_7*[7:0] | *SDHC_RESP_0* |
| R3 (OCR register) | OCR register for memory | REP [31:0] | *SDHC_RESP_1* | *SDHC_RESP_0* |
| R4 (OCR register) | OCR register for I/O, etc | REP [31:0] | *SDHC_RESP_1* | *SDHC_RESP_0* |
| R5, R5b | SDIO response | REP [31:0] | *SDHC_RESP_1* | *SDHC_RESP_0* |
| R6 (Published RCA response) | Newly published RCA[31:16], etc | REP [31:0] | *SDHC_RESP_1* | *SDHC_RESP_0* |

*Table 7-64: SDHC Buffer Data Port Register*

| Buffer Data Port Register | | SDHC_BUFFER | | [0x0020] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | data | R/W | 0 | **Buffer Data**<br>Pointer to the SDHC internal data buffer. |

*Table 7-65: SDHC Present State Register*

| Present State Register | | SDHC_PRESENT | | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:25 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 24 | cmd_signal_level | RO | - | **CMD Line Signal Level**<br>Indicates the CMD line level for error recovery and debugging. |
| 23:20 | dat_signal_level | RO | - | **SDHC_DAT[3:0] Line Signal Level**<br>Indicates the DAT line level for error recovery and debugging. Use to detect the busy signal level as indicated on SDHC_DAT[0]. |
| 19 | wp | RO | - | **Write Protect Switch Pin Level**<br>The write protect switch is supported for memory and combo cards. This bit reflects the state of the SDHC_WP pin.<br>1: Write enabled (SDHC_WP = 1)<br>0: Write protected (SDHC_WP = 0) |
| 18 | card_detect | RO | - | **Card Detect Pin Level**<br>This bit reflects the inverted state of the SDHC_CDN pin. Debouncing is not performed on this bit. When Card State Stable is set to 1, this bit might be valid, but is not guaranteed. To use this bit, the card driver must debounce the bit.<br>1: Card present (SDHC_CDN = 0)<br>0: No card present (SDHC_CDN = 1) |
| 17 | card_state | RO | - | **Card State Stable**<br>Used for debugging only. If this bit reads 0, the SDHC_CDN pin level is not stable. If this bit reads 1, the SDHC_CDN pin level is stable.<br>1: No card or card inserted<br>0: Reset or debouncing<br>*Note: This bit is not valid unless the SDHC_PRESENT.card_inserted bit reads 1.* |
| 16 | card_inserted | RO | - | **Card Inserted**<br>Indicates if a card is inserted. This signal is debounced by the SDHC hardware. A change in state from 0 to 1 on this bit generates an SDHC_IRQ with the *SDHC_INT_STAT.card_insertion* flag set. Conversely, a transition of this bit from a 1 to a 0 generates an SDHC_IRQ interrupt with the *SDHC_INT_STAT.card_removal* field set.<br>1: Card Inserted<br>0: Reset, debouncing, or no card inserted |
| 15:12 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| Present State Register | | | SDHC_PRESENT | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 11 | buffer_read | RO | 0 | **Buffer Read Enable**<br>If this bit reads 1, then data is available in the buffer for non-DMA transfers. This bit is cleared when all available block data is read from the buffer. This bit transitions from 0 to 1 when block data is ready in the buffer resulting in a SDHC_IRQ interrupt, if enabled, with the *SDHC_INT_STAT.buffer_rd_ready* flag set.<br>1: Read data available<br>0: No data to read |
| 10 | buffer_write | RO | 0 | **Buffer Write Enable**<br>If this bit reads 1, then space is available in the buffer for write data. This bit is cleared when no space is available in the buffer. This bit transitions from a 0 to a 1 when top-of-block data is written to the buffer, resulting in a SDHC_IRQ interrupt, if enabled, with the *SDHC_INT_STAT.buffer_wr_ready* flag set.<br>1: Space available in the buffer for write data<br>0: No space available in the buffer for write data |
| 9 | read_transfer | RO | 0 | **Read Transfer Active**<br>Indicates completion of a read transfer.<br>This bit is set to 1 for either of the following conditions:<br>1) After the end bit of a Read command.<br>2) When a read operation is restarted by setting the *SDHC_BLK_GAP.cont* bit (Continue Request).<br>This bit is set to 0 for either of the following conditions:<br>1) The last data block as specified by the block length is transferred to the SDHC.<br>2) When all valid data blocks are transferred to the system, and no current block transfers are sent because the Stop At Block Gap Request register field is set to 1.<br>A SDHC_IRQ interrupt is generated, if enabled.<br>1: Transferring data<br>0: No valid data |
| 8 | write_transfer | RO | 0 | **Write Transfer Active**<br>This bit is set to 1 for either of the following conditions:<br>3) After the end bit of the Write command.<br>4) When a write operation is restarted by setting the *SDHC_BLK_GAP.cont* bit to 1.<br>This bit is cleared to 0 for either of the following conditions:<br>5) After getting the CRC status of the last data block transfer as specified by the transfer count, single and multiple block, *SDHC_BLK_CNT* register.<br>6) After getting the CRC status of any block where data transmission is stopped by a Stop At Block Gap Request (*SDHC_BLK_GAP.stop*).<br>When *SDHC_BLK_GAP.stop* (stop at block gap request) is set, a change in *write_transfer* from 1 to 0 causes an SDHC_IRQ interrupt, if enabled, with the *SDHC_INT_STAT.blk_gap_event* flag set to 1. The *blk_gap_event* field indicates to the card driver that a non-DAT command can be issued during an active write.<br>1: Transferring data<br>0: No valid data for transfer |
| 7:4 | - | RO | NA | **Reserved for Future Use**<br>Do not modify this field. |

| Present State Register | | | SDHC_PRESENT | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 3 | retuning | RO | 0 | **Re-Tuning Request**<br>If this field reads 1, a retuning request was received from the external device.<br>0: Re-tuning request has not been received.<br>1: Re-tuning request received. |
| 2 | dat_line_active | RO | 0 | **DAT Line Active**<br>A value of 1 indicates one or more DAT lines (SDHC_DAT[3:0]) are in use on the SD Bus.<br>0: No SD Bus DAT lines in use.<br>1: 1 or more DAT lines are in use. |
| 1 | dat | RO | 0 | **Command Inhibit (DAT)**<br>This bit is set if DAT Line Active or the Read Transfer Active bits are set. A SDHC_IRQ interrupt is generated, if enabled, when this bit transitions from a 1 to a 0 with the *SDHC_INT_STAT.trans_comp* flag set. The card driver can save registers in the range of 0x000 to 0x00D for a suspend transaction after the *SDHC_INT_STAT.trans_comp* interrupt event.<br>1: Command that uses DAT line cannot be issued.<br>0: Command that uses DAT line can be issued. |
| 0 | cmd_comp | RO | 0 | **Command Inhibit (CMD)**<br>If this bit reads 0, the CMD line is not in use. This bit is set to 1 by the SDHC immediately after the *SDHC_CMD* register is written, and the bit is cleared to 0 when the Command Response is received. Auto CMD12 and Auto CMD23 consist of two responses, and this bit is not cleared until the read/write portion of the sequence is complete.<br>1: Command cannot be issued.<br>0: Can issue command using only CMD line. |

*Table 7-66: SDHC Host Control 1 Register*

| Host Control 1 Register | | | SDHC_HOST_CN_1 | [0x0028] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7 | card_detect_signal | R/W | 0 | **Card Detect Signal Selection**<br>1: The Card Detect Test Level is selected (for test purposes)<br>0: SDHC_CDN is used for card detection (normal operation)<br>*Note: Disable the Card Detect Interrupt when changing this bit.* |
| 6 | card_detect_test | R/W | - | **Card Detect Test Level**<br>This bit is enabled when the Card Detect Signal Selection, *SDHC_HOST_CN_1*. card_detect_signal, field is set to 1.<br>1: Card Inserted<br>0: No card inserted |
| 5 | ext_data_transfer_width | R/W | 0 | **Extended Data Transfer Width**<br>Extended data transfer width is not supported on the MAX32650–MAX32652. Always reads 0.<br>0: Bus width is selected by SHDC_HOST_CN_1.*data_transfer_width* field |

| Host Control 1 Register | | | SDHC_HOST_CN_1 | [0x0028] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4:3 | dma_select | R/W | 0 | **DMA Select**<br>Sets the DMA mode.<br>0b00: SDMA mode<br>0b01: Reserved<br>0b10: 32-bit address ADMA2 mode<br>0b11: Reserved |
| 2 | hs_en | R/W | 0 | **High Speed Enable**<br>1: High-speed mode<br>0: Normal-speed mode |
| 1 | data_transfer_width | R/W | 0 | **Data Transfer Width**<br>Sets the data transfer width of the SDHC.<br>1: 4-bit mode<br>0: 1-bit mode |
| 0 | led_cn | R/W | 0 | **LED Control**<br>1: LED on<br>0: LED off |

*Table 7-67: SDHC Power Control Register*

| Power Control Register | | | SDHC_PWR | [0x0029] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:4 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3:1 | bus_volt_sel | R/W | 6 | **SD Bus Voltage Select**<br>Sets the voltage level for the SD card. Validate the setting against the Capabilities Register (*SDHC_CFG_0*).<br>7: 3.3V typical<br>6: 3.0V typical<br>5: 1.8V typical<br>4: Reserved for Future Use.<br>3: Reserved for Future Use.<br>2: Reserved for Future Use.<br>1: Reserved for Future Use.<br>0: Reserved for Future Use. |
| 0 | bus_power | R/W | 0 | **SD Bus Power**<br>Before setting this bit, configure the SDHC_PWR.*bus_volt_sel* field. If no card is detected, then this bit is automatically set to 0 by the SDHC.<br>1: Power Enabled<br>0: Power Disabled |

*Table 7-68: SDHC Block Gap Control Register*

| Block Gap Control Register | | | SDHC_BLK_GAP | [0x002A] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:4 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| Block Gap Control Register | | SDHC_BLK_GAP | | [0x002A] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 3 | intr | R/W | 0 | **Interrupt at Block Gap**<br>Setting this bit to 1 enables interrupt detection at the block gap for a multiple block transfer.<br>1: Enabled<br>0: Disabled<br>*Note: This bit is only valid if SDHC_PWR.data_transfer_width=1 (4-bit mode).* |
| 2 | read_wait | R/W | 0 | **Read Wait Control**<br>If the card supports read wait (optional for SDIO cards), setting this bit enables use of the read wait protocol to stop reading data using the SDHC_DAT[2] line. If the card does not support read wait, the SDHC stops the SD Clock to hold read data, preventing command generation. When a card is inserted, the card driver must set this field based on the CCCR of the SDIO card inserted.<br>Suspend/Resume is not supported when this bit is set to 0.<br>1: Enable Read Wait Control<br>0: Disable Read Wait Control<br>*Note: If the SDIO card does not support read wait, then you must not set this bit to 1. Setting it to 1 when read wait is not supported might cause a SDHC_DAT line conflict.* |
| 1 | cont | R/W | 0 | **Continue Request**<br>This bit is used to restart a transaction that was stopped using the Stop At Block Gap Request (*SDHC_BLK_GAP.stop*). To cancel a stop at the block gap, set *SDHC_BLK_GAP.stop* to 0, and set this bit, *SDHC_BLK_GAP.cont*, to 1 to restart the transfer.<br>This bit is automatically cleared by hardware for either of the following conditions:<br>• During a read transaction, the DAT Line Active changes from 0 to 1 as the write transaction restarts.<br>• During a write transaction, the Write Transfer Active changes from 0 to 1 as the write transaction restarts.<br>1: Restart<br>0: No effect |

| Block Gap Control Register | | SDHC_BLK_GAP | | [0x002A] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | stop | R/W | 0 | **Stop At Block Gap Request**<br>Setting this bit stops executing read and write transactions at the next block gap for non-DMA, SDMA, and ADMA transfers. This bit must remain set to 1 until the *SDHC_INT_STAT.trans_comp* bit is set to 1.<br>For write transfers where the card driver writes data to the Buffer Data Port Register (*SDHC_BUFFER*), the card driver must set this bit after all block data is written.<br>1: Stop<br>0: Transfer<br>This bit affects the following fields:<br>• Read Transfer Active, *SDHC_PRESENT.read_transfer*<br>• Write Transfer Active, *SDHC_PRESENT.write_transfer*<br>• SDHC_DAT Line Active, *SDHC_PRESENT.dat_line_active*<br>• Command Inhibit (DAT), *SDHC_PRESENT.dat*<br>*Note: If this bit is set to 1, the card driver must not write data to the Buffer Data Port Register (SDHC_BUFFER).*<br>*Note: Clearing both the SDHC_BLK_GAP.stop and SDHC_BLK_GAP.cont fields does not cause a transaction to restart.*<br>*Note: You can set this bit to 1 regardless of whether the card inserted supports Read Wait Control. The SDHC stops the card through Read Wait Control or by stopping the SD clock.* |

*Table 7-69: SDHC Wakeup Control Register*

| Wakeup Control Register | | SDHC_WAKEUP | | [0x002B] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:3 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 2 | card_rem | R/W | 0 | **Wakeup Event Enable on SD Card Removal**<br>Enable wakeup event interrupt when the *SDHC_INT_STAT.card_removal* flag occurs.<br>1: Enable Interrupt<br>0: Disable Interrupt |
| 1 | card_ins | R/W | 0 | **Wakeup Event Enable on SD Card Insertion**<br>Enable wakeup event interrupt when the *SDHC_INT_STAT.card_inserted* flag occurs.<br>1: Enable Interrupt<br>0: Disable Interrupt |
| 0 | card_int | R/W | 0 | **Wakeup Event Enable On Card Interrupt**<br>Enable wakeup event interrupt when the *SDHC_INT_STAT.card_intr* flag occurs. |

*Table 7-70: SDHC Clock Control Register*

| Clock Control Register | SDHC_CLK_CN | [0x002C] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 15:8 | sdclk_freq_sel | R/W | 0 | **SDCLK Frequency Select**<br>Selects the SD Clock Frequency output on the SDHC_CLK pin.<br>The SD Clock Frequency Select is a total of 10bits. The divisors shown below consist of the upper_sdclk_freq_sel bits as bits 9:8, and the sdclk_freq_sel bits as bits 7:0 of the divisor.<br>*upper_sdclk_freq_sel* / *sdclk_freq_sel* / SDCLK Divisor (N)<br>0b11 / 0b11111111 / 1023<br>0b11 / 0b00000000 / 768<br>… / … / …<br>0b10 / 0b01010101 / 597<br>…. / …. / ….<br>…. / …. / N<br>…. / …. / ….<br>0b00 / 0b00000010 / 2<br>0b00 / 0b00000001 / 1<br>0b00 / 0b00000000 / 0 (MAX)<br>Setting upper_sdclk_freq_sel and sdclk_freq_sel to 0 results in the maximum SDCLK frequency of $f_{SDHC\_CLK\_FRQ}$. All other settings for upper_sdclk_freq_sel and sdclk_freq_sel follow the equation below:<br>$SDHC\_CLK = f_{SDHC\_CLK\_FRQ} / (2 \times N)$<br>*Note: The SD Clock Enable must be disabled (SDHC_CLK_CN.sd_clk_en = 0) prior to modification of this field.* |
| 7:6 | upper_sdclk_freq_sel | R/W | 0 | **Upper Bits of SDCLK Frequency Select**<br>Bits 9 and 8 of the 10-bit SDCLK frequency select. Refer to the *SDHC_CLK_CN.sdclk_freq_sel* field for details about the clock select calculation.<br>*Note: The SD Clock Enable must be disabled (SDHC_CLK_CN.sd_clk_en = 0) prior to modification of this field.* |
| 5 | clk_gen_sel | RO | 0 | **Clock Generator Select**<br>Reads 0 indicating Divided Clock mode only for SD Clock Frequency generation.<br>0: Divided clock mode |
| 4:3 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 2 | sd_clk_en | R/W | 0 | **SD Clock Enable**<br>Enable/disable SD Clock generation.<br>1: Enable the SD Clock and output on the SDHC_CLK pin.<br>0: SD Clock is disabled.<br>*Note: This bit is cleared by the SDHC if the card-inserted field in the Present State register is cleared.*<br>*Note: The internal_clk_en bit must be set to 1, and the internal_clk_stable bit must read 1 prior to setting this bit to 1.* |

| Clock Control Register | | SDHC_CLK_CN | | [0x002C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 1 | internal_clk_stable | RO | 0 | **Internal Clock Stable**<br>This bit is set to 1 when the internal clock is stable.<br>*Note: The internal clock must be enabled (SDHC_CLK_CN.internal_clk_en = 1) before this field is used.* |
| 0 | internal_clk_en | R/W | 0 | **Internal Clock Enable**<br>Enable the internal clock.<br>*Note: This bit must be set, and the internal_clk_stable bit must read 1 prior to setting the SD Clock Enable (SDHC_CLK_CN.sd_clk_en) bit.*<br>*Note: This bit is set to 0 by the SDHC if waiting for a wakeup interrupt.* |

*Table 7-71: SDHC Timeout Control Register*

| Timeout Control Register | | SDHC_TO | | [0x002E] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:4 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3:0 | data_count_value | R/W | 0 | **Data Timeout Counter Value**<br>Determines the interval for DAT line timeout detection. The timeout clock frequency is generated by dividing PCLK by the value calculated using this register. See Capabilities 0 Register (SDHC_CFG_0) for the definition of TMCLK.<br>The calculation for Data Timeout is shown in the following equation:<br>$Data\ Timeout = TMCLK \times 2^{(13+data\_count\_value)}$<br><br>**Setting** / **Data Timeout**<br>0b1111 – Reserved<br>0b1110 – $TMCLK \times 2^{(27)}$<br>0b1101 – $TMCLK \times 2^{(26)}$<br>… – …<br>0b0010 – $TMCLK \times 2^{(15)}$<br>0b0001 – $TMCLK \times 2^{(14)}$<br>0b0000 – $TMCLK \times 2^{(13)}$<br><br>*Note: Disable the Data Timeout Error Status Enable in the Error Interrupt Status Enable register (SDHC_ER_INT_EN.data_to).* |

*Table 7-72: SDHC Software Reset Register*

| Software Reset Register | | SDHC_SW_RESET | | [0x002F] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:3 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

<table>
<tr><th colspan="3">Software Reset Register</th><th>SDHC_SW_RESET</th><th>[0x002F]</th></tr>
<tr><th>Bits</th><th>Name</th><th>Access</th><th>Reset</th><th>Description</th></tr>
<tr><td>2</td><td>reset_dat</td><td>RWAC</td><td>0</td><td>Software Reset for DAT Line<br>1: Reset<br>0: Ready<br>The following registers and fields are cleared/initialized when this bit is set:
<table>
<tr><th>Register</th><th>Field</th></tr>
<tr><td>SDHC_BUFFER</td><td>data</td></tr>
<tr><td rowspan="8">SDHC_PRESENT</td><td>buffer_read</td></tr>
<tr><td>buffer_write</td></tr>
<tr><td>read_transfer</td></tr>
<tr><td>write_transfer</td></tr>
<tr><td>dat_line_active</td></tr>
<tr><td>dat</td></tr>
<tr><td>cmd</td></tr>
<tr><td rowspan="2">SDHC_BLK_GAP</td><td>cont</td></tr>
<tr><td>stop</td></tr>
<tr><td rowspan="6">SDHC_INT_STAT</td><td>buff_rd_ready</td></tr>
<tr><td>buff_wr_ready</td></tr>
<tr><td>dma</td></tr>
<tr><td>blk_gap_event</td></tr>
<tr><td>trans_comp</td></tr>
</table>
Note: After setting this bit to 1, the Card Driver must poll this bit until it reads 0 to determine reset completion.</td></tr>
<tr><td>1</td><td>reset_cmd</td><td>RWAC</td><td>0</td><td>Software Reset for CMD Line<br>1: Reset<br>0: Ready<br>The following registers and fields are cleared by setting this bit.
<table>
<tr><th>Register</th><th>Field</th></tr>
<tr><td>SDHC_PRESENT</td><td>cmd</td></tr>
<tr><td>SDHC_INT_STAT</td><td>cmd_comp</td></tr>
</table>
Note: After setting this bit to 1, the card driver must poll this bit for 0 to determine when the reset is complete.</td></tr>
<tr><td>0</td><td>reset_all</td><td>RWAC</td><td>0</td><td>Software Reset for All<br>Reset the SDHC except for the card detection interface. All registers are reset to their Reset/POR state.<br>1: Reset<br>0: Ready<br>Note: After the Card Driver sets this bit to 1, the Card Driver should poll this bit until it reads 0 to determine when the SDHC completes the reset all request.</td></tr>
</table>

### 7.5.8.1 Normal Interrupt Status Register

The Normal Interrupt Status Enable affects reads of this register, but Normal Interrupt Signal Enable does not. An interrupt is generated when the Normal Interrupt Signal Enable is enabled, and at least one of the status bits is set to 1. Writing 1 to a bit of the RW1C attribute clears it. Writing 0 keeps the bit unchanged. Writing 1 to a bit of the ROC attribute keeps the bit unchanged. You can clear more than one status with a single register write. The Card Interrupt (*SDHC_INT_STAT.card_intr*) is cleared when the card stops asserting the interrupt after the Card Driver services the interrupt condition.

*Table 7-73: SDHC Normal Interrupt Status Register*

| **Normal Interrupt Status Register** | | | **SDHC_INT_STAT** | **[0x0030]** |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15 | err_intr | ROC | 0 | **Error Interrupt**<br>If any of the bits in the Error Interrupt Status register are set, then this bit is set. Therefore, the Host Driver can efficiently test for an error by checking this bit first. This bit is read only.<br>1: Error<br>0: No Error |
| 14:13 | - | ROC | 0 | **Reserved for Future Use** |
| 12 | retuning | ROC | 0 | **Re-Tuning Event**<br>This status is set if the Re-Tuning Request bit in the Present State register changes from 0 to 1. The SDHC requests the Host Driver to perform re-tuning for the next data transfer. However, you can complete the current data transfer (not large block count) without re-tuning.<br>1: Perform re-tuning before the next data transfer<br>0: Re-tuning is not required |
| 11:9 | - | ROC | 0 | **Reserved for Future Use** |
| 8 | card_intr | ROC | 0 | **Card Interrupt**<br>In one-bit mode, the SDHC detects the Card Interrupt without the SD Clock to support wakeup. In four-bit mode, the card interrupt signal is sampled during the interrupt cycle resulting in a delay between the interrupt signal from the memory card and the interrupt signal to the host driver.<br>1: Generate Card Interrupt<br>0: No Card Interrupt<br>*Note: Writing a 1 to this bit does not clear this bit. It is cleared by resetting the SDHC_INT_EN.card_int flag.* |
| 7 | card_removal | RW1C | 0 | **Card Removal**<br>Set if the Card Inserted field in the Present State register (*SDHC_PRESENT.card_inserted*) changes from 1 to 0.<br>1: Card removed<br>0: Card state stable or hardware debouncing |
| 6 | card_insertion | RW1C | 0 | **Card Inserted**<br>Set if the Card Inserted field in the Present State register (*SDHC_PRESENT.card_inserted*) changes from 0 to 1.<br>1: Card inserted<br>0: Card state stable or hardware debouncing |

| Normal Interrupt Status Register | | SDHC_INT_STAT | [0x0030] |
|---|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 5 | buff_rd_ready | RW1C | 0 | **Buffer Read Ready**<br>Set if the Buffer Read Enable field in the Present State register (*SDHC_PRESENT.buff_rd_ready*) changes from 0 to 1.<br>1: Ready to read buffer<br>0: Not ready to read buffer<br>*Note: This field is set to 1 for every CMD19 execution while performing a tuning procedure (SDHC_HOST_CN_2.execute = 1).* |
| 4 | buff_wr_ready | RW1C | 0 | **Buffer Write Ready**<br>Set if the Buffer Write Enable field in the Present State register (*SDHC_PRESENT.buff_wr_ready*) changes from 0 to 1.<br>1: Ready to write buffer<br>0: Not ready to write buffer |
| 3 | dma | RW1C | 0 | **DMA Interrupt**<br>Set when the SDHC encounters the DMA buffer boundary set in the *SDHC_BLK_SIZE*.trans field during a SDMA transfer. The Card Driver must update the *SDHC_SDMA* register with the address of the next block to transfer before the SDHC continues the transfer.<br>1: SDHC DMA Interrupt is generated<br>0: No SDHC DMA Interrupt |
| 2 | blk_gap_event | RW1C | 0 | **Block Gap Event**<br>If the Stop at Block Gap Request field is set in the Block Gap Control register (*SDHC_BLK_GAP*.stop), this bit is set when a read or write transaction is stopped at a block gap. If Stop at Block Gap Request is not set to 1, then this bit is not meaningless.<br>1: Transaction stopped at block gap<br>0: No Block Gap Event |
| 1 | trans_comp | RW1C | 0 | **Transfer Complete**<br>Set when a read/write transfer and a command with busy is complete. This bit has higher priority than Data Timeout Error. If both bits are set to 1, execution of a command is complete. See *Table 7-74* for Transfer Complete and Data Timeout Error priority and meaning.<br>1: Command execution is complete<br>0: Not complete<br>*Note: This field is not set while performing a tuning procedure (SDHC_HOST_CN_2.execute = 1).* |
| 0 | cmd_cmp | RW1C | 0 | **Command Complete**<br>Set when the end bit of the command response is received. Auto CMD12 and Auto CMD23 consist of two responses. This flag is not set by the card's response to the CMD12 or CMD23, but by the card's response to the read or write command you send to complete the Auto CMD12 or Auto CMD23. Refer to Command Inhibit (CMD) in the Present State (*SDHC_PRESENT.cmd_comp*) register for how to control this bit. *Table 7-75* illustrates the relationship between Command Complete and Command Timeout Error bits. If both bits are set, then the response was not received within 64 SD clock cycles.<br>1: Command execution is complete<br>0: Not complete |

*Table 7-74: Transfer Complete and Data Timeout Error Priority and Status*

| Transfer Complete *SDHC_INT_STAT.trans_comp* | Data Timeout Error *SDHC_ER_INT_STAT.data_to* | Status |
|---|---|---|
| 0 | 0 | Interrupted by another event |
| 0 | 1 | Timeout occurred during transfer |
| 1 | N.A. | Command execution complete |

*Table 7-75: Command Complete and Command Timeout Error Priority and Status*

| Transfer Complete *SDHC_INT_STAT.cmd_comp* | Data Time Error *SDHC_ER_INT_STAT.cmd_to* | Status |
|---|---|---|
| 0 | 0 | Interrupted by another event. |
| N.A. | 1 | Response not received within 64 SD Clock cycles. |
| 1 | 0 | Response received. |

### 7.5.8.2 Error Interrupt Status Register

The interrupts defined in this register are enabled by the corresponding fields in the Error Interrupt Status Enable (*SDHC_ER_INT_EN*) register. Setting any field in the *SDHC_ER_INT_SIGNAL* register enables SDHC error interrupt generation using the SDHC_IRQ. The interrupt occurs when any field in the *SDHC_ER_INT_STAT* register is set to 1.

*Table 7-76: SDHC Error Interrupt Status Register*

| Error Interrupt Status Register | SDHC_ER_INT_STAT | [0x0032] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 15:13 | - | R/W1C | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 12 | dma | R/W1C | 0 | **DMA Error**<br>Error in SDMA transaction<br>1: Error<br>0: No error |
| 11:10 | - | R/W1C | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 9 | adma | R/W1C | 0 | **ADMA Error**<br>Set when the SDHC detects an error during an ADMA data transfer. The state of the ADMA when the error occurs is saved in the ADMA Error Status (*SDHC_ADMA_ER*) register.<br>This bit is also set if the SDHC detects invalid descriptor data. If the *SDHC_ADMA_ER* register indicates an ADMA Error State, then an invalid descriptor was detected.<br>1: Error<br>0: No error |

| Error Interrupt Status Register | SDHC_ER_INT_STAT | [0x0032] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 8 | auto_cmd_12 | R/W1C | 0 | **Auto CMD Error**<br>Auto CMD12 and Auto CMD23 use this error status. This bit is set when detecting that one of the bits D00-D04 in Auto CMD Error Status (*SDHC_AUTO_CMD_ER*) register changed from a 0 to a 1.<br>1: Error<br>0: No error<br>*Note: For Auto CMD12, this bit is set to 1 not only when an error occurs in Auto CMD12, but also when Auto CMD12 is not executed due to a previous command error.* |
| 7 | current_limit | R/W1C | 0 | **Current Limit Error**<br>Not supported on MAX32650–MAX32652 |
| 6 | data_end_bit | R/W1C | 0 | **Data End Bit Error**<br>Set if a 0 is detected at the end bit position of read data that uses the DAT line or the end-bit position of the CRC status.<br>1: Error<br>0: No error |
| 5 | data_crc | R/W1C | 0 | **Data CRC Error**<br>Set when a CRC error is detected when receiving read data that uses the DAT line or when detecting a Write CRC status with a value other than 010.<br>1: Error<br>0: No error |
| 4 | data_to | R/W1C | 0 | **Data Timeout Error**<br>Set for any of the following timeout conditions:<br>• Busy timeout for R1b and R5b response types. See Table 7-53 for more information about response types.<br>• Busy timeout after Write CRC status<br>• Write CRC status Timeout<br>• Read Data Timeout<br>1: Error<br>0: No error |
| 3 | cmd_idx | R/W1C | 0 | **Command Index Error**<br>Set if a Command Index error is detected in the Command Response.<br>1: Error<br>0: No error |
| 2 | cmd_end_bit | R/W1C | 0 | **Command End Bit Error**<br>Set if the end bit of a Command Response is 0.<br>1: Error<br>0: No error |

| Error Interrupt Status Register | SDHC_ER_INT_STAT | [0x0032] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 1 | cmd_crc | R/W1C | 0 | **Command CRC Error**<br>Set for the following cases:<br>7) If a response is returned, and the Command Timeout Error is set to 0, then this error flag is set if a CRT error is detected in the Command Response.<br>8) The SDHC detects a CMD-line conflict by monitoring the *SDHC_CMD* line when a command is issued. The SDHC sets the Command Timeout Error flag if a CMD line conflict is detected. A CMD line conflict indicates the CMD line was driven to 1, and the SDHC detected a 0 on the CMD line on the next SDCLK.<br>1: Error<br>0: No error |
| 0 | cmd_to | R/W1C | 0 | **Command Timeout Error**<br>Set if there is not response within 64 SDCLK cycles from the end bit of a command.<br>1: Error<br>0: No error<br>*Note: If both the* SDHC_ER_INT_STAT.cmd_crc *and* SDHC_ER_INT_STAT.cmd_to *flags are set, then the SDHC detected a CMD-line conflict. See* SDHC_ER_INT_STAT.cmd_crc *for more information about a CMD-line conflict.* |

*Table 7-77: SDHC Normal Interrupt Status Register*

| Normal Interrupt Status Enable Register | SDHC_INT_EN | [0x0034] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 15:13 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 12 | retuning | R/W | | **Re-Tuning Event Status Enable**<br>1: Enabled<br>0: Disabled |
| 11:9 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 8 | card_int | R/W | 0 | **Card Interrupt Status Enable**<br>Set to enable card-interrupt detection. The Card Driver should clear this bit prior to servicing a card interrupt status event and re-enable this bit after all interrupts from the card are serviced.<br>1: Enabled<br>0: Disabled |
| 7 | card_removal | R/W | 0 | **Card Removal Status Enable**<br>Set to enable card removal event.<br>1: Enabled<br>0: Disabled |
| 6 | card_insert | R/W | 0 | **Card Insertion Status Enable**<br>Set to enable card insertion event.<br>1: Enabled<br>0: Disabled |

| Normal Interrupt Status Enable Register | | | SDHC_INT_EN | [0x0034] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 5 | buffer_rd | R/W | 0 | **Buffer Read Ready Status Enable**<br>Set to enable Buffer Read Ready status.<br>1: Enabled<br>0: Disabled |
| 4 | buffer_wr | R/W | 0 | **Buffer Write Ready Status Enable**<br>Set to enable Buffer Write Ready status.<br>1: Enabled<br>0: Disabled |
| 3 | dma | R/W | 0 | **DMA Interrupt Status Enable**<br>Set to enable DMA status.<br>1: Enabled<br>0: Disabled |
| 2 | blk_gap | R/W | 0 | **Block Gap Event Status Enable**<br>Set to enable Block Gap status.<br>1: Enabled<br>0: Disabled |
| 1 | trans_comp | R/W | 0 | **Transfer Complete Status Enable**<br>Set to enable Transfer Complete status.<br>1: Enabled<br>0: Disabled |
| 0 | cmd_comp | R/W | 0 | **Command Complete Status Enable**<br>Set to enable Command Complete status.<br>1: Enabled<br>0: Disabled |

*Table 7-78: SDHC Error Interrupt Status Enable Register*

| Error Interrupt Status Enable Register | | | SDHC_ER_INT_EN | [0x0036] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:13 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 12 | vendor | R/W | 0 | **Target Response Error/Host Error Status Enable**<br>Set to enable Target Response/Host Error status interrupts.<br>1: Enabled<br>0: Disabled |
| 11 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 10 | tuning | R/W | 0 | **Tuning Error Status Interrupt Enable**<br>1: Enabled<br>0: Disabled |
| 9 | adma | R/W | 0 | **ADMA Error Status Interrupt Enable**<br>1: Enabled<br>0: Disabled |

| Error Interrupt Status Enable Register | | | SDHC_ER_INT_EN | [0x0036] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 8 | auto_cmd_12 | R/W | 0 | **Auto CMD12 Error Status Interrupt Enable**<br>1: Enabled<br>0: Disabled |
| 7 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 6 | data_end_bit | R/W | 0 | **Data End Bit Error Status Interrupt Enable**<br>1: Enabled<br>0: Disabled |
| 5 | data_crc | R/W | 0 | **Data CRC Error Status Interrupt Enable**<br>1: Enabled<br>0: Disabled |
| 4 | data_to | R/W | 0 | **Data Timeout Error Status Interrupt Enable**<br>1: Enabled<br>0: Disabled |
| 3 | cmd_idx | R/W | 0 | **Command Index Error Status Interrupt Enable**<br>1: Enabled<br>0: Disabled |
| 2 | cmd_end_bit | R/W | 0 | **Command End Bit Error Status Interrupt Enable**<br>1: Enabled<br>0: Disabled |
| 1 | cmd_crc | R/W | 0 | **Command CRC Error Status Interrupt Enable**<br>1: Enabled<br>0: Disabled |
| 0 | cmd_to | R/W | 0 | **Command Timeout Error Status Interrupt Enable**<br>1: Enabled<br>0: Disabled |

*Table 7-79: SDHC Normal Interrupt Signal Enable Register*

| Normal Interrupt Signal Enable Register | | | SDHC_INT_SIGNAL | [0x0038] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:13 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 12 | retuning | R/W | 0 | **Re-Tuning Event Signal Enable**<br>1: Enabled<br>0: Disabled |
| 11:9 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 8 | card_int | R/W | 0 | **Card Interrupt Signal Enable**<br>1: Enabled<br>0: Disabled |
| 7 | card_removal | R/W | 0 | **Card Removal Signal Enable**<br>1: Enabled<br>0: Disabled |

| Normal Interrupt Signal Enable Register | | SDHC_INT_SIGNAL | | [0x0038] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 6 | card_insert | R/W | 0 | **Card Insertion Signal Enable**<br>1: Enabled<br>0: Disabled |
| 5 | buffer_rd | R/W | 0 | **Buffer Read Ready Signal Enable**<br>1: Enabled<br>0: Disabled |
| 4 | buffer_wr | R/W | 0 | **Buffer Write Ready Signal Enable**<br>1: Enabled<br>0: Disabled |
| 3 | dma | R/W | 0 | **DMA Interrupt Signal Enable**<br>1: Enabled<br>0: Disabled |
| 2 | blk_gap | R/W | 0 | **Block Gap Signal Enable**<br>1: Enabled<br>0: Disabled |
| 1 | trans_comp | R/W | 0 | **Transfer Complete Signal Enable**<br>1: Enabled<br>0: Disabled |
| 0 | cmd_comp | R/W | 0 | **Command Complete Signal Enable**<br>1: Enabled<br>0: Disabled |

*Table 7-80: SDHC Error Interrupt Signal Enable Register*

| Error Interrupt Signal Enable Register | | SDHC_ER_INT_SIGNAL | | [0x003A] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:13 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 12 | tar_resp | R/W | 0 | **Target Response Error Signal Enable**<br>1: Enabled<br>0: Disabled |
| 11 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 10 | tuning | R/W | 0 | **Tuning Error Signal Enable**<br>1: Enabled<br>0: Disabled |
| 9 | adma | R/W | 0 | **ADMA Error Signal Enable**<br>1: Enabled<br>0: Disabled |
| 8 | auto_cmd_12 | R/W | 0 | **Auto CMD12 Error Signal Enable**<br>1: Enabled<br>0: Disabled |

| Error Interrupt Signal Enable Register | | | SDHC_ER_INT_SIGNAL | [0x003A] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7 | current_limit | R/W | 0 | **Current Limit Error Signal Enable**<br>1: Enabled<br>0: Disabled |
| 6 | data_end_bit | R/W | 0 | **Data End Bit Error Signal Enable**<br>1: Enabled<br>0: Disabled |
| 5 | data_crc | R/W | 0 | **Data CRC Error Signal Enable**<br>1: Enabled<br>0: Disabled |
| 4 | data_to | R/W | 0 | **Data Timeout Error Signal Enable**<br>1: Enabled<br>0: Disabled |
| 3 | cmd_idx | R/W | 0 | **Command Index Error Signal Enable**<br>1: Enabled<br>0: Disabled |
| 2 | cmd_end_bit | R/W | 0 | **Command End Bit Error Signal Enable**<br>1: Enabled<br>0: Disabled |
| 1 | cmd_crc | R/W | 0 | **Command CRC Error Signal Enable**<br>1: Enabled<br>0: Disabled |
| 0 | cmd_to | R/W | 0 | **Command Timeout Error Signal Enable**<br>1: Enabled<br>0: Disabled |

### 7.5.8.3 Auto CMD Error Status Register

This register is used to indicate response errors for Auto CMD12 and Auto CMD23. The contents of this register are only valid when the Auto CMD Error is set (*SDHC_ER_INT_STAT.auto_cmd_12*). For Auto CMD23 errors, the error code is stored in *SDHC_AUTO_CMD_ER*[4:1].

*Table 7-81: SDHC Auto CMD Error Status Register*

| Auto CMD Error Status Register | | | SDHC_AUTO_CMD_ER | [0x003C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:8 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7 | not_issued | ROC | 0 | **Command Not Issued by Auto CMD12 Error**<br>1: Command not issued due to Auto CMD12 error as indicated in bits 4:1 of this register.<br>0: Auto CMD Error issued by Auto CMD23 |
| 6:4 | - | ROC | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| Auto CMD Error Status Register | | | SDHC_AUTO_CMD_ER | [0x003C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4 | index | ROC | 0 | **Auto CMD Index Error**<br>Command Index error occurred in response to a command.<br>1: Command Index Error<br>0: No Error |
| 3 | end_bit | ROC | 0 | **Auto CMD End Bit Error**<br>Set if the end bit of the Command Response is 0.<br>1: End Bit Error<br>0: No Error |
| 2 | crc | ROC | 0 | **Auto CMD CRC Error**<br>Set if CRC error in command response.<br>1: CRC Error<br>0: No Error<br>*Note: If both SDHC_AUTO_CMD_ER.crc and SDHC_AUTO_CMD_ER.to are set, then a CMD-line conflict occurred.* |
| 1 | to | ROC | 0 | **Auto CMD Timeout Error**<br>Set if no response is returned within 64 SDCLK cycles from the end bit of the command. If set, then ignore bits 4:2 of this register.<br>1: Timeout Error<br>0: No Error<br>*Note: If both SDHC_AUTO_CMD_ER.crc and SDHC_AUTO_CMD_ER.to are set, then a CMD-line conflict occurred.* |
| 0 | not_execute | ROC | 0 | **Auto CMD12 Not Executed Error**<br>Auto CMD12 was not issued to stop a multi-block memory transfer due to an error with a prior command.<br>1: Not Executed<br>0: No Error or error generated by Auto CMD23 |

*Table 7-82: SDHC Host Control 2 Register*

| Host Control 2 Register | | | SDHC_HOST_CN_2 | [0x003E] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15 | preset_val_en | R/W | | **Preset Value Enable**<br>When set to 0, the following fields must be set by the Card Driver:<br>• SDCLK Frequency Select (*SDHC_CLK_CN.sdclk_freq_sel*)<br>• Clock Generator Select (*SDHC_CLK_CN.clk_gen_sel*)<br>• Driver Strength Select (*SDHC_HOST_CN_2.driver_strength*)<br>If set to 1, the Host Controller hardware sets the above fields based on the values in the Preset Value registers.<br>0: Card Driver must set the SDCLK Frequency Select, Clock Generator Select and Driver Strength Select fields.<br>1: The Host Controller hardware sets the above fields using the Preset Value register settings. |
| 14 | asynch_int | R/W | 0 | **Asynchronous Interrupt Enable**<br>Always reads 0. Asynchronous Interrupt Enable is not supported by the MAX32650–MAX32652. Writes to this field have no effect. |
| 13:8 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| Host Control 2 Register | | | SDHC_HOST_CN_2 | [0x003E] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7 | sampling_clk | R/W | 0 | **Sampling Clock Select**<br>This field is automatically set by hardware when Execute Tuning (*SDHC_HOST_CN_2*. execute) is cleared.<br>0: The fixed clock is used to sample data<br>1: The tuned clock is used to sample data<br>*Note: The Card Driver cannot write 1 to this bit. Writing this bit to 0 can only be done if the Host Controller is not receiving a response or reading a data block.* |
| 6 | execute | R/WAC | 0 | **Execute Tuning**<br>Setting this bit to 1 starts the tuning procedure and the bit is automatically cleared when the Host Controller completes the tuning procedure. Writing a 0 to this bit when it is set to 1 aborts the tuning procedure.<br>1: Execute tuning<br>0: Tuning complete or not tuned |
| 5:4 | driver_strength | R/W | 0 | **Driver Strength Select**<br>If using 3.3V signaling, this field is ignored. For 1.8V signaling, the output driver strength is set using this field.<br>If *SDHC_HOST_CN_2.preset_val_en* = 0, this field is controlled by the Host Driver. If *SDHC_HOST_CN_2.preset_val_en* = 1, this field is automatically set by the hardware using the Preset Value registers.<br>00b: Driver Type B is selected<br>01b: Driver Type A is selected<br>10b: Driver Type C is selected<br>11b: Driver Type D is selected |
| 3 | 1_8v_signal | R/W | 0 | **1.8V Signaling Enable**<br>If the card inserted supports UHS-I, this bit can be set to 1. No matter the value set, 3.3V is used for the card's supply.<br>1: 1.8V signaling<br>0: 3.3V signaling |
| 2:0 | uhs | R/W | 0 | **UHS Mode Select**<br>Used to select the UHS-I mode. This field is only used if 1.8V signaling is set to 1 (*SDHC_HOST_CN_2.1_8v_signal* = 1).<br>000b: SDR12<br>001b: SDR25<br>010b: SDR50<br>011b: SDR104 (Not supported)<br>100b: DDR50<br>101b – 111b: Reserved for Future Use |

*Table 7-83: SDHC Capabilities Register 0*

| Capabilities Register 0 | | | SDHC_CFG_0 | [0x0040] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:30 | slot_type | RO | 0b00 | **Slot Type**<br>0b00: Support for a single slot with support for a removable card |
| 29 | async_int | RO | 1 | **Asynchronous Interrupt Support**<br>1: Asynchronous Interrupt Supported |

| Capabilities Register 0 | | | SDHC_CFG_0 | [0x0040] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 28 | 64_bit_sys_bus | RO | 0 | **64-bit System Bus Support**<br>0: 64-bit system bus not supported |
| 27 | - | RO | 0 | **Reserved for Future Use** |
| 26 | 1_8v | RO | 1 | **Voltage Support 1.8V**<br>1: 1.8V supported |
| 25 | 3_0v | RP | 1 | **Voltage Support 3.0V**<br>1: 3.0V supported |
| 24 | 3_3v | RO | 1 | **Voltage Support 3.3V**<br>1: 3.3V supported |
| 23 | suspend | RO | 1 | **Suspend/Resume Support**<br>1: Suspend / Resume functionality is supported |
| 22 | sdma | RO | 1 | **SDMA Support**<br>SDMA is supported and can transfer data between system memory and the SDHC directly.<br>1: SDMA supported |
| 21 | hs | RO | 1 | **High Speed Support**<br>The SDHC supports High Speed mode with $f_{PCLK}$=120MHz/2.<br>1: High speed mode supported |
| 20 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 19 | adma2 | RO | 1 | **ADMA2 Support**<br>The SDHC supports ADMA2.<br>1: ADMA2 supported |
| 18 | 8_bit | RO | 0 | **8-bit Support for Embedded Device**<br>The SDHC supports 8-bit bus width mode.<br>0: 8-bit Bus width not supported |
| 17:16 | max_blk_len | RO | 0b10 | **Max Block Length**<br>This value indicates the maximum block size that the Host Driver can read and write to the buffer in the SDHC without wait cycles. The transfer block length is always 512 bytes for SD memory cards regardless of this field.<br>0b10: 2048 bytes |
| 15:8 | clk_freq | RO | 0x00 | **Base Clock Frequency for SD Clock**<br>0x00: Get information using another method |
| 7 | clk_unit | RO | 1 | **Timeout Clock Unit**<br>1: MHz base clock unit |
| 6 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 5:0 | clk_freq | RO | 0x01 | **Timeout Clock Frequency**<br>The base clock frequency used to detect Data Timeout errors. The Timeout Clock Unit defines the units of this field's value.<br>1: 1 [MHz] |

*Table 7-84: SDHC Capabilities Register 1*

| Capabilities Register 1 | | | SDHC_CFG_1 | [0x0044] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | - | RO | 1 | **Reserved for Future Use**<br>Do not modify this field. |
| 23:16 | clk_multi | RO | 0 | **Clock Multiplier**<br>Always reads 0x00.<br>0: Programmable clock generation is not supported. |
| 15:14 | retuning | RO | 0 | **Re-Tuning Modes**<br>Always reads 0b00. The SDHC supports Mode 1 Re-Tuning only with timer controlled by the host driver and a maximum of 4MB data length. |
| 13 | tuning_sdr50 | RO | 0 | **Use Tuning for SDR50**<br>1: Tuning required for SDR50<br>0: SDR50 does not require tuning |
| 12 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11:8 | timer_cnt_tuning | RO | 0 | **Timer Count for Re-Tuning**<br>0x0: Re-Tuning Timer disabled<br>0x1: 1 second<br>0x2: 2 seconds<br>0x3: 4 seconds<br>0x4: 8 seconds<br>….: ……<br>n: $2^{(n-1)}$ $seconds$<br>….: ……<br>0xB: 1024 seconds<br>0xC: Reserved<br>0xD: Reserved<br>0xE: Reserved<br>0xF: Get information from another source |
| 7 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 6 | driver_d | RO | 1 | **Driver Type D Support**<br>1: Driver Type D is supported |
| 5 | driver_c | RO | 1 | **Driver Type C Support**<br>1: Driver Type C is supported |
| 4 | driver_a | RO | 1 | **Driver Type A Support**<br>1: Driver Type A is supported |
| 3 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 2 | ddr50 | RO | 1 | **DDR50 Support**<br>1: DDR50 is support |
| 1 | sdr104 | RO | 1 | **SRD104**<br>1: SDR104 is supported |
| 0 | sdr50 | RO | 1 | **SDR50**<br>1: SDR50 is supported |

*Table 7-85: SDHC Maximum Current Capabilities Register*

| Maximum Current Capabilities Register | | | SDHC_MAX_CURR_CFG | [0x0048] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 23:16 | 1_8v | RO | 0 | **Maximum Current for 1.8V**<br>0x00: System dependent |
| 15:8 | 3_0v | RO | 0 | **Maximum Current for 3.0V**<br>0x00: System dependent |
| 7:0 | 3_3v | RO | 0 | **Maximum Current for 3.3V**<br>0x00: System dependent |

*Table 7-86: SDHC Force Event Register for Auto CMD Error Status Register*

| Force Event Register for Auto CMD Error Status | | | SDHC_FORCE_CMD | [0x0050] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:8 | - | WO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7 | not_issued | WO | 0 | **Force Event for Command Not Issued By Auto CMD12 Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 6:5 | - | WO | 0 | **Reserved for Future Use**<br>Do not modify field. |
| 4 | index | WO | 0 | **Force Event for Auto CMD Index Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 3 | end_bit | WO | 0 | **Force Event for Auto CMD End Bit Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 2 | crc | WO | 0 | **Force Event for Auto CMD CRC Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 1 | to | WO | 0 | **Force Event for Auto CMD Timeout Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 0 | not_excu | WO | 0 | **Force Event for Auto CMD12 Not Executed**<br>1: Interrupt is generated<br>0: No interrupt generated |

*Table 7-87: SDHC Force Event Register for Error Interrupt Status*

| Force Event Register for Error Interrupt Status | | SDHC_FORCE_EVENT_INT_STAT | | [0x0052] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:12 | stat_vendor | R/W | 0 | **Force Event for Vendor Specific Error Status**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 11:10 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 9 | adma | R/W | | **Force Event for ADMA Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 8 | auto_cmd | R/W | 0 | **Force Event for Auto CMD Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 7 | curr_limit | R/W | 0 | **Force Event for Current Limit Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 6 | data_end_bit | R/W | 0 | **Force Event for Data End Bit Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 5 | data_crc | R/W | 0 | **Force Event for Data CRC Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 4 | data_to | R/W | 0 | **Force Event for Data Timeout Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 3 | cmd_index | R/W | 0 | **Force Event for Command Index Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 2 | cmd_end_bit | R/W | 0 | **Force Event for Command End Bit Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 1 | cmd_crc | R/W | 0 | **Force Event for Command CRC Error**<br>1: Interrupt is generated<br>0: No interrupt generated |
| 0 | cmd_to | R/W | 0 | **Force Event for Command Timeout Error**<br>1: Interrupt is generated<br>0: No interrupt generated |

*Table 7-88: SDHC ADMA Error Status Register*

| ADMA Error Status Register | | SDHC_ADMA_ER | | [0x0054] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:3 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| ADMA Error Status Register | | | SDHC_ADMA_ER | [0x0054] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 2 | len_mismatch | ROC | 0 | **ADMA Length Mismatch Error**<br>This error occurs in the following two cases:<br>1) When setting Block Count Enable, the total data length specified by the Descriptor Table is different from that specified by the Block Count and Block Length fields.<br>2) Total data length is not divisible by the Block Length field.<br>1: Error<br>0: No Error |
| 1:0 | state | ROC | 0b00 | **ADMA Error State**<br>The state of the ADMA when the error condition occurred. Only valid during data transfer for ADMA.<br>The following table shows the possible state values, the associated ADMA Error State, and the contents of the *SDHC_SDMA* register.<br>**state** / **ADMA Error State when the error occurred** / **SYS_SDR register contents**<br>0b00 / ST_STOP (Stop DMA) / Points next to the error descriptor<br>0b01 / ST_FDS (Fetch Descriptor) / Points to the error descriptor<br>0b10 / N.A. / N.A.<br>0b11 / ST_TFR (Transfer Data) / Points next to the error descriptor<br>*Note: 0b10 is not a valid error state and is never set.* |

*Table 7-89: SDHC ADMA System Address Register 0*

| ADMA System Address Register 0 | | SDHC_ADMA_ADDR_0 | | [0x0058] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | addr | R/W | 0 | **ADMA System Address 0**<br>Holds the byte address of the executing command for the Descriptor Table. The Host Driver must set this address, made up of <*SDHC_ADMA_ADDR_1*>:<*SDHC_ADMA_ADDR_0*>, to the start address of the Descriptor Table. The ADMA increments this register address when fetching a descriptor line to point to the next address. When an ADMA Error Interrupt occurs, this register holds a valid descriptor address depending on the ADMA state. The following table shows the 64-bit System Address for ADMA using <*SDHC_ADMA_ADDR_1*>:<*SDHC_ADMA_ADDR_0*>. |

| *SDHC_ADMA_ADDR_1* | *SDHC_ADMA_ADDR_0* | 64-bit System Address |
|---|---|---|
| 0x0000 0000 | 0x0000 0000 | 0x00000000_00000000 |
| 0x0000 0000 | 0x0000 0004 | 0x00000000_00000004 |
| 0x0000 0000 | 0x0000 0008 | 0x00000000_00000008 |
| 0x0000 0000 | 0x0000 000C | 0x00000000_0000000C |
| ……. | ……. | ……. |
| 0xFFFF FFFF | 0xFFFF FFFC | 0xFFFFFFFF_FFFFFFFC |

*Note: The Host Driver must program the Descriptor Table on 32-bit boundaries and set the 32-bit boundary address to this register. ADMA2 ignores the lower two bits of this register, assuming it to be 00b.*

*Table 7-90: SDHC ADMA System Address Register 1*

| ADMA System Address Register 1 | | SDHC_ADMA_ADDR_1 | | [0x005C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | addr | R/W | 0 | **ADMA System Address 1**<br>Most-significant double word for 64-bit ADMA address. See *SDHC_ADMA_ADDR_0* for details. |

### 7.5.8.4 Preset Value Registers

All Preset Value registers (*SDHC_PRESET_0* to *SDHC_PRESET_7*) contain the same fields as described in the *SDHC_PRESET_0* register. One of the Preset Value registers is automatically selected by the SDHC based on the selected bus-speed mode

*Table 7-91* shows a group of preset values per card or device. One of the Preset Value registers (*SDHC_PRESET_1* – *SDHC_PRESET_7*) is selected by the SDHC hardware based on the Selected Bus Speed mode. *Table 7-92* defines the conditions to select one of the Preset Value registers.

*Table 7-91: Preset Value Register Example*

| Offset | Preset Value Registers | Signal Voltage |
|---|---|---|
| [0x0060] | Preset Value for Initialization | 3.3V or 1.8V |
| [0x0062] | Preset Value for Default Speed | 3.3V |

| Offset | Preset Value Registers | Signal Voltage |
|---|---|---|
| [0x0064] | Preset Value for High Speed | 3.3V |
| [0x0066] | Preset Value for SDR12 | 1.8V |
| [0x0068] | Preset Value for SDR25 | 1.8V |
| [0x006A] | Preset Value for SDR50 | 1.8V |
| [0x006C] | Preset Value for SDR104 | 1.8V |
| [0x006E] | Preset Value for DDR50 | 1.8V |

*Table 7-92: Preset Value Register Selection Conditions*

| Selected Bus Speed Mode | 1.8V Signaling Enable *SDHC_HOST_CN_2*. 1_8v_signal | High Speed Enable *SDHC_HOST_CN_1*. hs_en | UHS-I Mode Selection *SDHC_HOST_CN_2*.uhs |
|---|---|---|---|
| Default Speed | 0 | 0 | N.A. |
| High Speed | 0 | 1 | N.A. |
| SDR12 | 1 | N.A. | 0b000 |
| SDR25 | 1 | N.A. | 0b001 |
| SDR50 | 1 | N.A. | 0b010 |
| SDR104 | 1 | N.A. | 0b011 |
| DDR50 | 1 | N.A. | 0b100 |
| Reserved | 1 | N.A. | 0b101 to 0b111 |

*Table 7-93: SDHC Preset Value 0 to Preset Value 7 Registers*

| | | |
|---|---|---|
| **Preset Value 0 for Initialization** | **SDHC_PRESET_0** | **[0x0060]** |
| **Preset Value 1 for Initialization** | **SDHC_PRESET_1** | **[0x0062]** |
| **Preset Value 2 for Initialization** | **SDHC_PRESET_2** | **[0x0064]** |
| **Preset Value 3 for Initialization** | **SDHC_PRESET_3** | **[0x0066]** |
| **Preset Value 4 for Initialization** | **SDHC_PRESET_4** | **[0x0068]** |
| **Preset Value 5 for Initialization** | **SDHC_PRESET_5** | **[0x006A]** |
| **Preset Value 6 for Initialization** | **SDHC_PRESET_6** | **[0x006C]** |
| **Preset Value 7 for Initialization** | **SDHC_PRESET_7** | **[0x006E]** |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 15:14 | driver_strength | RO | 1 | **Driver Strength Select Value**<br>Driver strength is supported by 1.8V signaling bus speed modes. This field is not used for 3.3V signaling.<br>0b00: Driver Type B is selected<br>0b01: Driver Type A is selected<br>0b10: Driver Type C is selected<br>0b11: Driver Type D is selected |
| 13:11 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| Preset Value 0 for Initialization | SDHC_PRESET_0 | [0x0060] |
|---|---|---|
| Preset Value 1 for Initialization | SDHC_PRESET_1 | [0x0062] |
| Preset Value 2 for Initialization | SDHC_PRESET_2 | [0x0064] |
| Preset Value 3 for Initialization | SDHC_PRESET_3 | [0x0066] |
| Preset Value 4 for Initialization | SDHC_PRESET_4 | [0x0068] |
| Preset Value 5 for Initialization | SDHC_PRESET_5 | [0x006A] |
| Preset Value 6 for Initialization | SDHC_PRESET_6 | [0x006C] |
| Preset Value 7 for Initialization | SDHC_PRESET_7 | [0x006E] |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 10 | clk_gen | RO | 0 | **Clock Generator Select Value**<br>0: Programmable clock generator is not supported |
| 9:0 | sdclk_freq | RO | - | **SDCLK Frequency Select Value**<br>10-bit preset value to set the SDCLK Frequency Select field in the Clock Control register (*SDHC_CLK_CN.upper_sdclk_freq_sel* and *SDHC_CLK_CN.sdclk_freq_sel*) |

*Table 7-94: SDHC Slot Interrupt Status Register*

| Slot Interrupt Status Register | SDHC_SLOT_INT | [0x00FC] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 15:8 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7:1 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 0 | int_signals | RO | 0 | **Interrupt Signals**<br>Indicates the logical OR of Interrupt Signal and Wakeup Signal for the single slot. Only one slot is defined for the MAX32650–MAX32652, slot 0. Reset by POR and by software reset for all (*SDHC_SW_RESET.reset_all*). |

*Table 7-95: SDHC Host Controller Version Register*

| Host Controller Version Register | SDHC_HOST_CN_VER | [0x00FE] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 15:8 | vend_ver | RO | - | **Vendor Version**<br>This status is reserved for the vendor version number. The Host Driver should not use this status. |
| 7:0 | spec_ver | RO | 0x02 | **Specification Version Number**<br>This status indicates the Host Controller Specification Version.<br>0x02: SD Host Specification Version 3.00 |

## 7.6 HyperBus/Xccela High Speed Memory Controller Interface

The HyperBus and Xccela Memory Controller (HBMC) interface is a high-speed, low-pin count interface for connecting the MAX32650–MAX32652 to one or more compatible external memory devices. The external HyperBus or Xccela Bus memory device is mapped into the MAX32650–MAX32652's memory space enabling direct code execution, data storage or both.

The HyperBus and Xccela Bus interface is a Double Data Rate (DDR) interface, using both the rising and falling edges of the clock to transfer data. The HBMC is connected to the APB and the peripheral clock is set using the PCLK with a maximum data transfer rate of 120Mbps in DDR mode.

Data is transferred over a high-speed 8-bit data bus. Two chip select lines are supported by the HBMC with only one chip select line active at a time. Data is latched for reads and writes using a Read/Write data strobe signal. HyperBus transfers are clocked using a differential clock while Xccela bus transfers use a single clock.

Features of the HyperBus/Xccela Memory Controller Interface include:

- Master/Slave System
- 120Mbytes/sec maximum data transfer rate
- Double Data Rate (DDR) – two data transfers per clock cycle
- Transparent bus operation to the processor
- 16KByte Write-through cache
- Two chip selects for two memory ports
- Each port supports memories up to 512MBytes
- Can address two external memories, one at a time
- Interfaces to HyperFlash, HyperRAM, and Xccela PSRAM
- Zero wait state burst mode operation
- Low power Half Sleep mode
- Puts the external memory device into low power mode while retaining memory contents
- Configurable timing parameters

All bus transactions are either a read or a write. Bus transactions, as well as read and write destinations and any required latency, are all slave device dependent. Read and write destinations are either in the memory space or the configuration register space of the target memory device. Refer to the slave device data sheet for specific timing diagrams, latency, and read/write memory addresses.

HyperBus memory devices are 16-bit word addressed so all addresses must be on 16-bit boundaries. Xccela memory devices can be byte-addressed. For both protocols all 16-bit and 32-bit transfers must be on aligned boundaries.

### 7.6.1 HyperBus/Xccela Signal Descriptions

*Table 7-96: HyperBus, Xccela Bus Pin Mapping and Signal Descriptions*

| Function | HyperBus Pin Name | Xccela Pin Name | MAX32650–MAX32652 Pin Name | Direction | Description |
|---|---|---|---|---|---|
| Chip Select | CS0# | CE0# | GPIO1.11 | Output | Each external device requires a dedicated chip select line. The MAX32650–MAX32652 initiates a bus transaction by transitioning a chip select line from a high state (deselected) to a low state (selected). When the MAX32650–MAX32652 completes the bus transaction, the HBMC drives the chip select line to the deselected state (high) indicating to the external device the end of the transaction. Only one chip select can be active at any given time.<br>***Note: CS#1 requires an external pull-up to the supply powering the HyperBus/Xccela Bus if it is used as a Chip Select line.*** |
| | CS1# | CE1# | GPIO3.0 | | |
| Clock | CK | CK | HYP_CLKB | Output | All information on the bus is transferred with respect to the edges of the clock. CK# is not used for Xccela Bus or 3V HyperBus devices. |
| | CK# | not used | HYP_CLK | | |
| Data I/O | DQ0 | DQ0 | GPIO1.12 | Input/Output | Data Bus Inputs/Outputs |
| | DQ1 | DQ1 | GPIO1.15 | | |
| | DQ2 | DQ2 | GPIO1.19 | | |
| | DQ3 | DQ3 | GPIO1.20 | | |
| | DQ4 | DQ4 | GPIO1.13 | | |
| | DQ5 | DQ5 | GPIO1.16 | | |
| | DQ6 | DQ6 | GPIO1.18 | | |
| | DQ7 | DQ7 | GPIO1.21 | | |
| R/W Data Strobe | RWDS | DQS | GPIO1.14 | Input/Output | RWDS is controlled by the Master or Slave device that is reading data. Data being read is edge aligned with this signal. This signal is held high during any read/write latency periods. |
| Hardware Reset | RESET# | RESET# | GPIO1.10 | Output | The HBMC drives this line low to reset the external device, placing it in Standby Mode. The HBMC sets the DQ[7:0] pins into High-Z input mode.. |

### 7.6.2 Related Specifications

Refer to the HyperBus Specification for details for the HyperBus protocol including timing diagrams, AC and DC characteristics, and electrical specifications. Refer to the HyperRAM or HyperFlash data sheet of the specific device being used for timing and electrical specifications. Refer to the Xccela PSRAM data sheet of the device being used for information on the Xccela protocol including timing diagrams, AC and DC characteristics, and electrical specifications.

Most compatible memory devices have on-chip registers for configuring the device. HyperBus refers to these registers as Configuration Registers, while Xccela devices refer to these as Mode Registers. This HyperBus/Xccela interface uses the HyperBus terminology for registers and pin names.

### 7.6.3 Reading and Writing to a Slave Device from Firmware

Reading and writing with external memory devices is done through two memory-mapped regions of this microcontroller's memory. Configure the memory-mapped addresses using the Memory Base Address registers, where HBMC_MBR0 configures the memory mapped region for Port0 and HBMC_MBR1 configures the memory mapped region for Port1. The allocated memory region for the HBMC on this microcontroller is between 0x6000 0000 and 0x7FFF FFFF. The recommended values are as follows:

HBMC_MBR0 = 0x6000 0000
HBMC_MBR1 = 0x7000 0000

The address on the target memory device is the offset from the above base addresses, so if firmware writes a value to RAM location 0x6000 0008, that value is written to address 0x0008 in the target memory device on Port0. If firmware reads memory location 0x7000 2000, then the value stored in address 0x2000 in the target memory device on Port1 is read back.

These memory-mapped regions are treated as device RAM and executable code space.

Reads and writes to both the external device's memory and configuration registers are all done with these memory-mapped regions. Data transfer operations are as easy as reading and writing to microcontroller RAM. This makes the behavior of the HBMC bus interface transparent to firmware. Bit banding is not supported.

Once the memory base addresses are configured, reads and writes to the memory mapped regions is device dependent. Refer to the memory device data sheet to determine how to use the address space.

From firmware, for both HyperBus and Xccela memory devices, all firmware read and write operations can be 16-bits or 32-bits on aligned on 16-bit or 32-bit boundaries, respectively. Xccela devices also support firmware single byte reads and writes on byte boundaries.

On the external bus interface, all HyperBus reads and writes are 16-bit bus transactions. All Xccela read bus transactions are 16-bit while all write bus transactions are 32-bit. RWDS is low for the active bytes, while any dummy bytes are masked with RWDS pulled high. This behavior is transparent to firmware.

For example, if firmware requests a byte write to an Xccela memory device, the external bus performs a 32-bit write on a 32-bit boundary. RWDS is low for the active byte and masks the three dummy bytes with RWDS high. This behavior is transparent to firmware.

For HyperBus devices, Configuration Register space must be accessed on 16-bit boundaries. Xccela devices can access Mode Registers on byte, 16-bit, or 32-bit boundaries.

### 7.6.4 Data Cache

Between the memory mapped regions and the HyperBus/Xccela external memory is a 16KByte data cache. When activated, the data cache fills its lines by performing INCR8/WRAP8 32-bit read bursts to the HyperBus controller. Configure external memory devices to have a WRAP burst length of 32 bytes.

When accessing a Configuration Register (CR) in an external memory device the data cache invalidated and disabled. While HyperFlash memory does not support configuration registers, HyperFlash does support the Status Register Read Command. This command requires the application firmware invalidate the data cache and then disable the data cache prior to the sending the Status Register Read Command. Refer to the Data Cache Chapter for details on invalidating and disabling the data cache.

### 7.6.5 *HyperBus/Xccela Memory Transfers*

Using either a HyperBus or Xccela Bus memory requires configuration of the HBMC to communicate with the external memory device. The HBMC supports a maximum of two devices connected via the external I/O pins

This HyperBus/Xccela Interface supports three memory types:

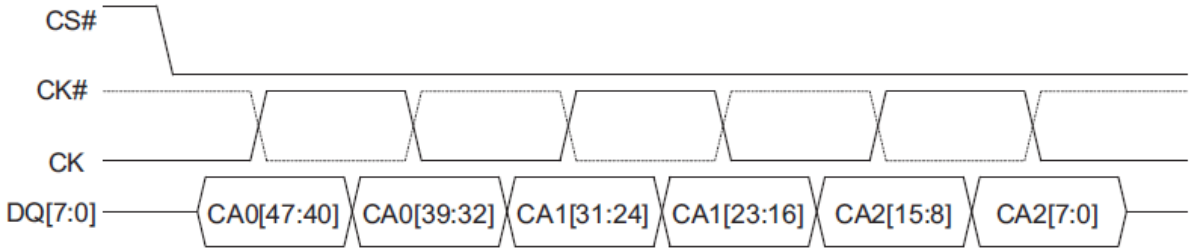
- HyperFlash
- HyperRAM
- Xccela PSRAM

Configure the bus timing for each external memory device. These parameters are in registers HBMC_MTR0 and HBMC_MTR1 and should match the same timing parameters specified in the data sheet of the memory device being accessed. By default, Xccela PSRAM is accessed with a variable read latency but for devices that support it, fixed read latency can be configured.

HyperBus and Xccela bus are similar except that HyperBus uses a differential clock (CK and CK#) while Xccela only uses one clock (CK). While there are differences in signal names, for simplicity only the HyperBus signal names are used for this interface. Both protocols use Double Data Rate (DDR) timing, so data is transferred on the 8-bit data bus on both rising and falling edges of the clock.

All bus transactions are either a read or write to the target memory. All bus transactions first start with one of the active-low chip selects (CS0# or CS1#) going low while CK is low. The first three clock cycles (six clock edges) transfer six bytes of Command/Address information. Because HyperBus is word-oriented the data is organized as three words designated CA0, CA1, and CA2. Xccela is byte-oriented so the data is organized as one 2-byte instruction INST, and four address bytes A0, A1, A2, and A3.

*Figure 7-10: HyperBus Command-Address Sequence*




These three words, CA0, CA1 and CA2, define the transaction type and contain the following information:

- Transaction type
  - Read or Write
- Target
  - Slave address space
  - Configuration register space
- Burst type
  - Linear or burst.
- Target row and column as determined by the memory controller
- Global Reset (Xccela only)

Register space is used to access Device Identification (ID) and Configuration Registers (CR). These identify the characteristics of the accessed device and determine the slave-specific behavior of read and write transfers on the HyperBus interface. To write to configuration registers, first set register bit *HBMC_MCR0.crt* = 1 or *HBMC_MCR1.crt* = 1. After the configuration registers are written, set these bits back to 0 to access memory.

### 7.6.6 External Memory Reset

The external memory devices can be reset by firmware. The external reset pin, RESET#, is on GPIO1.10. To generate a reset, firmware must pull this GPIO pin low, then high.

If the external memory is in a reset state during a read operation, the read-only status bit *HBMC_STATUS.rrstoerr* is set. If the external memory is in a reset state during a write operation, the read-only status bit *HBMC_STATUS.wrstoerr* is set. In both cases an interrupt is generated.

### 7.6.7 HyperBus/Xccela Interrupts

One interrupt is supported that occurs on any bus error. The interrupt is enabled when *HBMC_INTEN.errinte* = 1. When an AHB bus error occurs, *HBMC_INTFL.errints* is set.

### 7.6.8 HyperBus/Xccela Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the HyperBus (HBMC_) Base Peripheral Address

*Table 7-97: HyperBus Register Names, Offsets, Access and Descriptions*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *HBMC_STATUS* | RO | HBMC Status Register |
| [0x0004] | *HBMC_INTEN* | R/W | HBMC Interrupt Enable Control |
| [0x0008] | *HBMC_INTFL* | R/W1C | HBMC Interrupt Status Flags |
| [0x0010] | *HBMC_MBR0* | R/W | HBMC CS0# Memory Base Address Register |
| [0x0014] | *HBMC_MBR1* | R/W | HBMC CS1# Memory Base Address Register |
| [0x0020] | *HBMC_MCR0* | R/W | HBMC CS0# Memory Configuration Register |
| [0x0024] | *HBMC_MCR1* | R/W | HBMC CS1# Memory Configuration Register |
| [0x0030] | *HBMC_MTR0* | R/W | HBMC CS0# Memory Timing Register |
| [0x0034] | *HBMC_MTR1* | R/W | HBMC CS1# Memory Timing Register |

*Table 7-98: HBMC Status Register*

| HBMC Status Register | | | HBMC_STATUS | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:27 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 26 | wrstoerr | R/W | 0 | **Reset During Write Error**<br>If this field is set a reset during write error orrcured. When set to 1 by hardware, the *HBMC_INTFL.wrstoerr* interrupt flag is also set to 1. An HBMC IRQ is generated if the HBMC error interrupt enable is set to 1 (*HBMC_INTEN.errinte* = 1).<br>0: Normal Operation<br>1: The memory controller was reset during the write. |

| HBMC Status Register | | | HBMC_STATUS | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 25 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 24 | wdecerr | R/W | 0 | **Write Address Error**<br>If this field is set a write address error orrcured. When set to 1 by hardware, the *HBMC_INTFL*.*wdecerr* interrupt flag is set to 1. An HBMC IRQ is generated if the HBMC error interrupt enable is set to 1 (*HBMC_INTEN*.*errinte* = 1).<br><br>0: Normal operation.<br>1: The write address to the external memory is invalid. |
| 23:17 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 16 | wact | R/W | 0 | **Write Transaction in Progress**<br>This field is set if a write transaction is in progress. This field is cleared by hardware when the transaction completes.<br>0: No write in progress.<br>1: Write Transaction in progress. |
| 15:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11 | rdsstall | R/W | 0 | **Read Data Stall**<br>0: Normal Operation<br>1: Read transaction is stalled because RDS is low (stalled). |
| 10 | rrstoerr | R/W | 0 | **Reset During Read Error**<br>If this field is set a reset orrcured during a read. When set to 1 by hardware, the *HBMC_INTFL*.*rrstoerr* is also set to 1. An HBMC IRQ is generated if the HBMC error interrupt enable flag is set (*HBMC_INTEN*.*errinte* = 1).<br>0: Normal Operation<br>1: Error - Memory controller is under reset during the current read operation |
| 9 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 8 | rdecerr | R/W | 0 | **Read Address Error**<br>0: Normal operation<br>1: Error - external read address not responding |
| 7:1 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 0 | ract | R/W | 0 | **Read Transaction in Progress**<br>0: No read in progress.<br>1: Read transaction in progress. |

*Table 7-99: HBMC Interrupt Enable Control Register*

| HBMC Interrupt Enable Control | | | HBMC_INTEN | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| HBMC Interrupt Enable Control | | | HBMC_INTEN | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 1 | errinte | R/W | 0 | **Error Interrupt Enable**<br>0: No interrupt<br>1: Generate an interrupt when an error occurs |
| 0 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

*Table 7-100: HBMC Interrupt Status Flags Register*

| HBMC Interrupt Status Flags | | | HBMC_INTFL | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | errinte | R/W1C | 0 | **Error Interrupt Status Flag**<br>Write a 1 to clear this bit. |
| 0 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

*Table 7-101: HBMC CS0# Memory Base Address Register*

| HBMC CS0# Memory Base Address Register | | | HBMC_MBR0 | [0x0010] |
|---|---|---|---|---|
| HBMC CS1# Memory Base Address Register | | | HBMC_MBR1 | [0x0014] |
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | addr | R/W | 0 | **Memory Base Address**<br>This sets the base address of the addressable memory region where the port is mapped. Each address space is 512Mbytes. |
| 23:0 | addr | RO | 0 | **Address Low**<br>Always reads 0 |

This is the base address of the addressable memory region in this microcontroller's RAM. Because the addressable memory is mapped in 16M boundaries, the lower 24 bits are fixed to 0.

The value for HBMC_MBR0 must be less than HBMC_MBR1, or CS1# becomes inactive and the Port1 memory region is inaccessible.

The allocated memory region for the HBMC on this microcontroller is between 0x6000 0000 and 0x7FFFFFFF. The recommended values are as follows:

```
HBMC_MBR0 = 0x6000 0000
HBMC_MBR1 = 0x7000 0000
```

The address on the target memory device is the offset from the base address, so with the recommended values, if 0x12345678 is written to RAM address 0x6000 0008, then 0x12345678 is written to offset address 0x0008 in the target memory device on Port0. The interface signals to Port0, including CS0# and RWDS, are all handled automatically.

Bit banding is not supported in the HyperBus/Xccela addressable memory.

Reading and writing to the memory address space is device dependent. Refer to the slave device data sheet to determine how to use the memory space address map for that device.

CS0# is the Port0 memory region chip select.

CS1# is the Port1 memory region chip select. If CS#1 is used, an external pull-up resistor must be connected and tied to the supply used for the HyperBus/Xcella Bus peripheral.

*Table 7-102: HBMC Memory Configuration 0 Registers*

| **HBMC Memory Configuration Register 0** | **HBMC_MCR0** | **[0x0020]** |
|---|---|---|
| **HBMC Memory Configuration Register 1** | **HBMC_MCR1** | **[0x0024]** |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31 | maxlen_en | R/W | 0 | **Maximum CS# Length Enable**<br>0: No configurable CS# low time<br>1: CS# low time is configured using the field maxlen. |
| 30:27 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 26:18 | maxlen | R/W | 0 | **Maximum Read/Write**<br>Set this field to the CS# low time in terms of clock cycles.<br>The number of clock cycles total:<br>$N_{CLKS} = maxlen + 1$<br>Number of bytes transferred:<br>$N_T = 2 \times (maxlen + 1)$<br>*Note: This field is only valid when HBMC_MCR0.maxlen_en = 1.* |
| 17:8 | - | None | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7 | hse | R/1 | 0 | **Xccela Half Sleep Exit**<br>When half sleep exit is enabled, the CS# line is held low for ten clock cycles.<br>This bit is automatically cleared by hardware when a Half Sleep Exit completes.<br>0: Half Sleep Exit disabled<br>1: Half sleep exit enabled<br>*Note: This field is only used if the memory device type is either HyperFlash or HyperRAM (HBMC_MCR0.dev_type = 0 or HBMC_MCR0.dev_type = 2).* |
| 6 | read_latency_en | R/W | 0 | **Xccela Fixed Read Latency Enable**<br>Set this bit to enable Xccela bus Fixed Read Latency. Set this field to match the Latency Type configuration in the target PSRAM.<br>0: Variable read latency.<br>1: Fixed read latency.<br>*Note: This field is only used if the memory device type is Xccela PSRAM (HBMC_MCR0.dev_type = 1).* |
| 5 | crt | R/W | 0 | **Configuration Register Target Select**<br>For HyperRAM and Xccela Bus devices, this field selects between read/write target being the devices memory map or configuration register space.<br>For HyperFlash set this field to 0.<br>0: Access Memory space.<br>1: Access Configuration Register (CR) space. The data cache must be disabled/invalidated in this mode. |

| HBMC Memory Configuration Register 0 | HBMC_MCR0 | [0x0020] |
|---|---|---|
| HBMC Memory Configuration Register 1 | HBMC_MCR1 | [0x0024] |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 4:3 | dev_type | R/W | 0 | **Memory Device Type**<br>0: HyperFlash<br>1: Xccela PSRAM<br>2: HyperRAM<br>3: Reserved |
| 2:0 | - | R/W | 3 | **Reserved for Future Use**<br>Do not modify this field. |

*Table 7-103: HBMC Memory Timing Register 0*

| HBMC Memory Timing Register 0 | HBMC_MTR0 | [0x0030] |
|---|---|---|
| HBMC Memory Timing Register 1 | HBMC_MTR1 | [0x0034] |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:28 | rcshi | R/W | 0 | **Read Chip Select High Between Operations ($t_{CSHI}$)**<br>This bit sets the CS# high time, in clock cycles, between read operations as shown in the following equation:<br>$t_{CSHI} = t_{HBMC\_CLK} \times (rcshi + 1.5)$ |
| 27:24 | wcshi | R/W | 0 | **Write Chip Select High Between Operations ($t_{CSHI}$)**<br>This bit sets the CS# high time, in clock cycles, between write operations.<br>$t_{CSHI} = t_{HBMC\_CLK} \times (wcshi + 1.5)$<br>*Note: This field should be set to the same value as the rcshi field.* |
| 23:20 | rcss | R/W | 0 | **Read Chip Select Setup Time to Next CK Rising Edge ($t_{RCSS}$)**<br>This bit indicates CS# latency, in clock cycles, for read operations. It adds additional clock cycles after CS# goes low.<br>$t_{RCSS} = t_{HBMC\_CLK} \times (rcss + 1.0)$ |
| 19:16 | wcss | R/W | 0 | **Write Chip Select Setup Time to Next CK Rising Edge $t_{WCSS}$)**<br>This bit indicates CS# latency, in clock cycles, for write operations. The value set in this field adds additional clock cycles after the CS# line goes low.<br>$t_{WCSS} = t_{HBMC\_CLK} \times (wcss + 1.5)$ |
| 15:12 | rcsh | R/W | 0 | **Read Chip Select Hold After CK Falling Edge ($t_{RCSH}$)**<br>This field sets the CS# hold time, in clock cycles, between the completion of a read and when the CS# de-assertion.<br>$t_{RCSH} = t_{HBMC\_CLK} \times (rcsh + 1.0)$ |
| 11:8 | wcsh | R/W | 0 | **Write Chip Select Hold after CK falling edge ($t_{WCSH}$)**<br>This bit indicates the CS# hold time, in clock cycles, for the end of the write to the CS# de-assertion.<br>$t_{WCSH} = t_{HBMC\_CLK} \times (wcsh + 1.0)$ |
| 7:4 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| HBMC Memory Timing Register 0 | HBMC_MTR0 | [0x0030] |
|---|---|---|
| HBMC Memory Timing Register 1 | HBMC_MTR1 | [0x0034] |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 3:0 | latency | R/W | 1 | **RAM Latency Clock Cycles**<br>***HyperRAM:***<br>Set this field to the external HyperRAM Read Latency Configuration Register value.<br>***Xccela Bus:***<br>For Xccela Bus PSRAM, refer to *Table 7-104*, *below*.<br>0: 5 clock cycles<br>1: 6 clock cycles<br>14: 3 clock cycles<br>15: 4 clock cycles<br>All other values are reserved for future use.<br>*Note: This field is ignored when using HyperFlash devices (HBMC_MCR0.dev_type=0).* |

*Table 7-104: Latency Value Mapped to HyperRAM and Xccela PSRAM Latency Cycles*

| *HBMC_MTR0*<br>*HBMC_MTR1*<br>latency | HyperRAM R/W Latency<br>Clock Cycles | Xccela PSRAM Write Latency Clock Cycles | Xccela PSRAM Read Latency Clock Cycles<br>HBMC_MCRn.frl=0 | Xccela PSRAM Read Latency Clock Cycles<br>HBMC_MCRn.frl=1 |
|---|---|---|---|---|
| 0xE | 3 | 3 | 3 | 6 |
| 0xF | 4 | 4 | 4 | 8 |
| 0x0 | 5 | 5 | 5 | 10 |
| 0x1 | 6 | 6 | 6 | 12 |

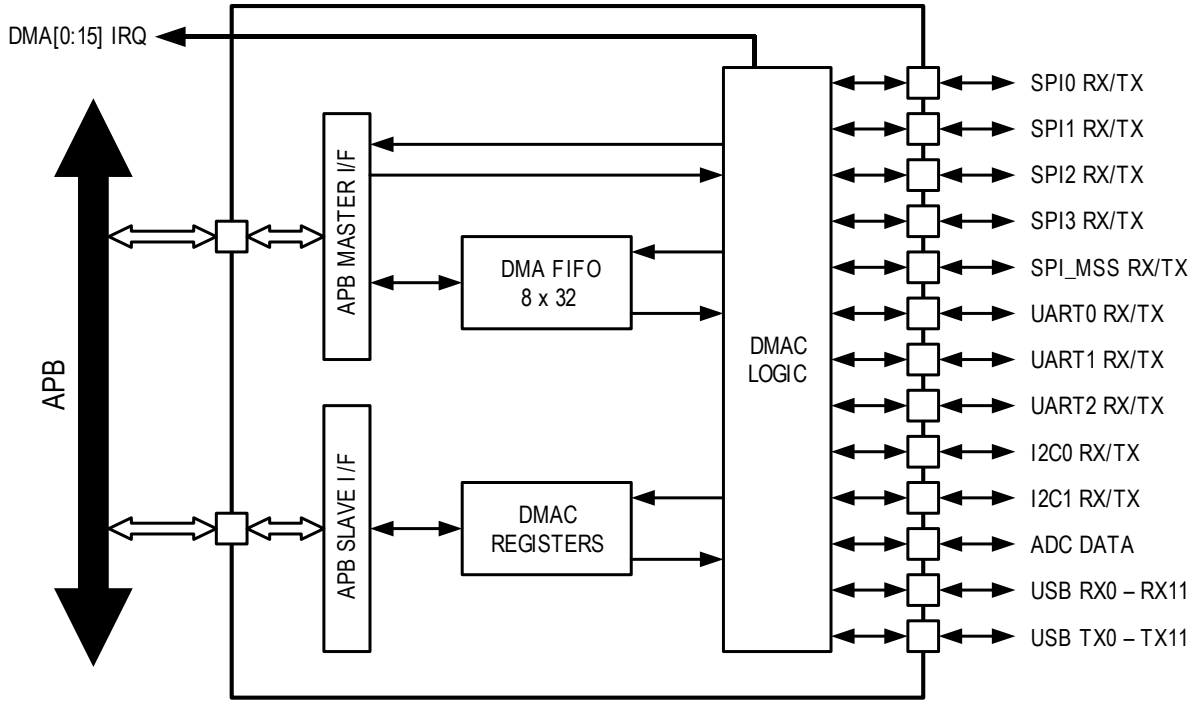
# 8 Standard DMA Controller

The Standard Direct Memory Access controller (DMAC) is a hardware feature that moves data blocks from peripheral to memory, memory to peripheral, and memory to memory. This hardware-based data movement reduces the processor load significantly.

*Note: The Standard DMA controller does not support internal Flash memory for memory source or destination.*

*Figure 8-1* provides a high-level overview of the major DMAC components.

*Figure 8-1: DMAC Block Diagram*




All direct memory access (DMA) transactions consist of an advanced high-performance bus (AHB) burst read from the source into the DMA FIFO followed by an AHB burst write from the DMA FIFO to the destination.

## 8.1 DMA channel operation

The DMA controller has 16 channels. Each channel is governed by the registers shown in *Table 8-1*.

*Table 8-1: DMA Channel Registers*

| Register | Description |
|---|---|
| *DMAn_DST* | Destination register |
| *DMAn_CFG* | Configuration register |
| *DMAn_ST* | Status register |

| Register | Description |
|---|---|
| *DMAn_SRC* | Source register |
| *DMAn_CNT* | Count register |

In addition, each channel has a set of reload registers shown in *Table 8-2* that are used to chain DMA buffers when a count-to-zero (CTZ) condition occurs:

*Table 8-2: Channel Reload Registers*

| Register | Description |
|---|---|
| *DMAn_DST_RLD* | Destination reload register |
| *DMAn_SRC_RLD* | Source reload register |
| *DMAn_CNT_RLD* | Count reload register |

Using these eight registers provides each channel with the following features:

- Full 32-bit source and destination addresses with 24-bit (16 Mbytes) address increment capability
- Up to 16 Mbytes for each DMA buffer
- Programmable burst size
- Programmable priority
- Interrupt upon CTZ
- Abort on error

## 8.2 DMA Channel Arbitration and DMA Bursts

DMAC contains an internal arbiter that allows enabled channels to access the AHB and move data. A DMA channel is enabled using the *DMAn_CFG*.*chen* bit.

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

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

Whenever the *DMAn_ST*.*ch_st* bit transitions from 1 to 0, the corresponding *DMAn_CFG*.*chen* bit is also cleared. During an AHB read/write burst, attempting to disable an active channel is delayed until burst completion.

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).

The arbiter grants requests to a single channel at a time. 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.

When a channel's request is granted, it runs a DMA transfer. Once the DMA transfer completes, the channel relinquishes its grant.

Only an error condition can interrupt an ongoing data transfer.

*DMAn_CFG*.*reqsel* determines which request is used to initiate a DMA burst. In the case of a memory-to-memory transfer, the channel is treated as always requesting DMA access. The *DMAn_CFG*.*priority* field determines the DMA channel priority.

## 8.3 DMA Source and Destination Addressing

For memory addresses, the *DMAn_SRC* and *DMAn_DST* registers are used to program the addresses of the source and destination. For peripherals, however, the address is fixed based on the settings of the *DMAn_CFG*.*reqsel* bit.

*Table 8-3* shows how the source and destination addresses as well as the address increment controls are constructed based on the *DMAn_CFG*.*reqsel* bit (shown in the Request Select column).

"Programmable" in the **SRCINC** or **DSTINC** columns indicates that the bits are programmable and set according to the *DMAn_CFG*.*srcinc* and the *DMAn_CFG*.*dstinc* bits, respectively. If there is a 0 in the column, then the bit is forced to 0.

*Table 8-3: Source and Destination Address Definition*

| Request Select | Transfer | Source Address | SRCINC | Destination Address | DSTINC |
|---|---|---|---|---|---|
| 0x0 | Mem-to-Mem | *DMAn_SRC* | Programmable | *DMAn_DST* | Programmable |
| 0x1 | SPI0 RX | *DMAn_SRC* | 0 | *DMAn_DST* | Programmable |
| 0x2 | SPI1 RX | *DMAn_SRC* | 0 | *DMAn_DST* | Programmable |
| 0x3 | SPI2 RX | *DMAn_SRC* | 0 | *DMAn_DST* | Programmable |
| 0x4 | UART0 RX | *DMAn_SRC* | 0 | *DMAn_DST* | Programmable |
| 0x5 | UART1 RX | *DMAn_SRC* | 0 | *DMAn_DST* | Programmable |
| 0x7 | I2C0 RX | *DMAn_SRC* | 0 | *DMAn_DST* | Programmable |
| 0x8 | I2C1 RX | *DMAn_SRC* | 0 | *DMAn_DST* | Programmable |
| 0x9 | ADC | *DMAn_SRC* | 0 | *DMAn_DST* | Programmable |
| 0xE | UART2 RX | *DMAn_SRC* | 0 | *DMAn_DST* | Programmable |
| 0xF | SPI3 RX | *DMAn_SRC* | 0 | *DMAn_DST* | Programmable |
| 0x10 | SPI_MSS RX | *DMAn_SRC* | 0 | *DMAn_DST* | Programmable |
| 0x11 | USB RX1 | USB OUT Endpoint 1 | 0 | *DMAn_DST* | Programmable |
| 0x12 | USB RX2 | USB OUT Endpoint 2 | 0 | *DMAn_DST* | Programmable |
| 0x13 | USB RX3 | USB OUT Endpoint 3 | 0 | *DMAn_DST* | Programmable |
| 0x14 | USB RX4 | USB OUT Endpoint 4 | 0 | *DMAn_DST* | Programmable |
| 0x15 | USB RX5 | USB OUT Endpoint 5 | 0 | *DMAn_DST* | Programmable |
| 0x16 | USB RX6 | USB OUT Endpoint 6 | 0 | *DMAn_DST* | Programmable |
| 0x17 | USB RX7 | USB OUT Endpoint 7 | 0 | *DMAn_DST* | Programmable |
| 0x18 | USB RX8 | USB OUT Endpoint 8 | 0 | *DMAn_DST* | Programmable |
| 0x19 | USB RX9 | USB OUT Endpoint 9 | 0 | *DMAn_DST* | Programmable |
| 0x1A | USB RX10 | USB OUT Endpoint 10 | 0 | *DMAn_DST* | Programmable |
| 0x1B | USB RX11 | USB OUT Endpoint 11 | 0 | *DMAn_DST* | Programmable |
| 0x21 | SPI0 TX | *DMAn_SRC* | Programmable | *DMAn_DST* | 0 |
| 0x22 | SPI1 TX | *DMAn_SRC* | Programmable | *DMAn_DST* | 0 |
| 0x23 | SPI2 TX | *DMAn_SRC* | Programmable | *DMAn_DST* | 0 |
| 0x24 | UART0 TX | *DMAn_SRC* | Programmable | *DMAn_DST* | 0 |
| 0x25 | UART1 TX | *DMAn_SRC* | Programmable | *DMAn_DST* | 0 |
| 0x27 | I2C0 TX | *DMAn_SRC* | Programmable | *DMAn_DST* | 0 |
| 0x28 | I2C1 TX | *DMAn_SRC* | Programmable | *DMAn_DST* | 0 |
| 0x2E | UART2 TX | *DMAn_SRC* | Programmable | *DMAn_DST* | 0 |

| Request Select | Transfer | Source Address | SRCINC | Destination Address | DSTINC |
|---|---|---|---|---|---|
| 0x2F | SPI3 TX | *DMAn_SRC* | Programmable | *DMAn_DST* | 0 |
| 0x30 | SPI_MSS TX | *DMAn_SRC* | Programmable | *DMAn_DST* | 0 |
| 0x31 | USB TX1 | *DMAn_SRC* | Programmable | USB IN Endpoint 1 | 0 |
| 0x32 | USB TX2 | *DMAn_SRC* | Programmable | USB IN Endpoint 2 | 0 |
| 0x33 | USB TX3 | *DMAn_SRC* | Programmable | USB IN Endpoint 3 | 0 |
| 0x34 | USB TX4 | *DMAn_SRC* | Programmable | USB IN Endpoint 4 | 0 |
| 0x35 | USB TX5 | *DMAn_SRC* | Programmable | USB IN Endpoint 5 | 0 |
| 0x36 | USB TX6 | *DMAn_SRC* | Programmable | USB IN Endpoint 6 | 0 |
| 0x37 | USB TX7 | *DMAn_SRC* | Programmable | USB IN Endpoint 7 | 0 |
| 0x38 | USB TX8 | *DMAn_SRC* | Programmable | USB IN Endpoint 8 | 0 |
| 0x39 | USB TX9 | *DMAn_SRC* | Programmable | USB IN Endpoint 9 | 0 |
| 0x3A | USB TX10 | *DMAn_SRC* | Programmable | USB IN Endpoint 10 | 0 |
| 0x3B | USB TX11 | *DMAn_SRC* | Programmable | USB IN Endpoint 11 | 0 |

## 8.4 Data Movement from Source to DMA FIFO

*Table 8-4* shows the register and bit fields used to control the movement of data into DMA FIFO. The source is a peripheral or memory.

*Table 8-4: Data movement from source to DMA FIFO*

| Register/Bit Field | Description | Comments |
|---|---|---|
| *DMAn_SRC* | Source address | If the increment enable is set, this increments on every read cycle of the burst. |
| *DMAn_CNT* | Number of bytes to transfer before a CTZ condition occurs | This register is decremented on each read of the burst. |
| *DMAn_CFG.brst* | Burst size (1-32) | This determines the maximum number of bytes moved during the burst read. |
| *DMAn_CFG.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 *DMAn_CNT* is not great enough to supply all the needed bytes. |
| *DMAn_CFG.srcinc* | Source increment enable | Increments *DMAn_SRC*. |

## 8.5 Data Movement from the DMA FIFO to Destination

*Table 8-5* shows the register and bit fields used to control the burst movement of data out of the DMA FIFO. The destination is a peripheral or memory.

*Table 8-5: Data movement from the DMA FIFO to destination*

| Register/Bit Field | Description | Comments |
|---|---|---|
| *DMAn_DST* | Destination address | If the increment enable is set, this increments on every write cycle of the burst. |
| *DMAn_CFG.brst* | Burst size (1-32) | This determines the maximum number of bytes moved during a single AHB read/write burst. |
| *DMAn_CFG.dstwd* | Destination width | This determines the maximum data width used during each write of the AHB burst (one byte, two bytes, or four bytes). |

| Register/Bit Field | Description | Comments |
|---|---|---|
| *DMAn_CFG.dstinc* | Destination increment enable | Increments *DMAn_DST*. |

## 8.6 Count-To-Zero Condition

When an AHB channel burst completes, DMAC checks whether *DMAn_CNT* is decremented to 0. If it is, then a CTZ condition exists.

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

5. If *DMAn_CFG*.*rlden* = 1, then the *DMAn_SRC*, *DMAn_DST*, and *DMAn_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.
6. If *DMAn_CFG*.*rlden* = 0, then the channel is disabled, and the *DMAn_ST*.*ch_st* bit is cleared.

## 8.7 Chaining Buffers

Use reload registers to chain buffers. Chaining buffers reduces the DMA ISR response time and allows DMA to service requests without intermediate processing from the CPU.

Configure the following registers to configure a channel for chaining:

- *DMAn_CFG*
- *DMAn_SRC*
- *DMAn_DST*
- *DMAn_CNT*
- *DMAn_SRC_RLD*
- *DMAn_DST_RLD*
- *DMAn_CNT_RLD*

When the *DMAn_CNT_RLD* register is written, the *DMAn_CNT_RLD*.*rlden* bit must not be set. In addition, any writes to the *DMAn_CFG* register prior to initialization must not set the *DMAn_CFG*.*chen* and *DMAn_CFG*.*rlden* bits. After all registers are initialized, the last operation involves writing to the *DMAn_CFG*.*chen* and *DMAn_CFG*.*rlden* bits. This starts the DMA.

Set the *DMAn_CFG*.*ctzien* bit in the register to receive an interrupt after each buffer is accessed. In addition, set the *DMAn_CFG*.*chdien* bit to provide an interrupt in case of a bus error.

*Caution: Setting the* DMAn_CFG*.chen and the* DMAn_CFG*.rlden bits separately risks a race condition. The condition occurs between a DMA completion interrupt service routine initializing the reload registers for the third buffer before the software initialization of these registers for the second buffer.*

When the first DMA transfer completes (based on the *DMAn_CNT*.*cnt* bit value), a CTZ interrupt occurs, and the *DMAn_SRC*, *DMAn_DST*, and *DMAn_CNT* registers are reloaded from the corresponding reload registers.

The *DMAn_ST* register indicates that the reload and CTZ events occurred. In this case, *DMAn_ST*.*ch_st* = 1 indicating that the DMA is now busy with the second DMA transfer defined in the reload registers. If DMAn_ST.*ch_st* = 0, then the initial and second DMA transfers have completed. If there are additional buffers to chain, the interrupt service routine initializes the *DMAn_SRC_RLD*, *DMAn_DST_RLD*, and *DMAn_CNT_RLD* registers and sets the *DMAn_CNT_RLD*.*rlden* bit. The interrupt service routine does not write to the *DMAn_CFG*, *DMAn_SRC*, *DMAn_DST*, and *DMAn_CNT* registers, just the reload registers.

To prevent improper operation, program the address bits before setting the *DMAn_CFG*.*chen* and *DMAn_CNT_RLD*.*rlden* bits.

## 8.8 DMA Interrupts

Enable interrupts for each channel by setting *DMA_CN.chien*. When an interrupt is pending, the corresponding *DMA_INT.ipend* = 1. The *DMA_INT.ipend* field is read-only, to clear the interrupt use the *DMAn_ST* register and write a 1 to the field that indicates the cause of the interrupt.

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

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

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

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

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

## 8.9 Channel Timeouts

Each channel can optionally generate an interrupt when its associated request line is inactive for a given time period. An example use of this feature is to determine an idle UART receive channel. Each channel has a dedicated 10-bit timer allowing use of a different timeout value.

## 8.10 10-bit Timer

Use the settings in the *DMAn_CFG* register to control each channel's 10-bit timer. Scale the input clock for the timer using the *DMAn_CFG.pssel* field. The options available are shown in the following table.

*Table 8-6: DMA Channel Timer Frequency Selection*

| ***DMAn_CFG pssel*** | **DMA Channel Timer Frequency** |
|---|---|
| 0 | $f_{TIMER} = \frac{f_{HCLK}}{256}$ |
| 1 | $f_{TIMER} = \frac{f_{HCLK}}{64K}$ |
| 2 | $f_{TIMER} = \frac{f_{HCLK}}{16M}$ |

*Note: HCLK is the AHB interface clock that enables the memory system to run at a different frequency than the system clock, the cache controller, and the event monitor.*

The *DMAn_CFG*.*tosel* field sets the time the 10-bit timer counts until generating an interrupt.

The 10-bit timer resets whenever any of the following conditions occur:

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

To disable the 10-bit timer, set the *DMAn_CFG*.*pssel* field to 0.

Normally, the 10-bit timer starts as soon as the channel is enabled and the *DMAn_CFG*.*pssel* field are non-zero. However, if *DMAn_CFG*.*reqwait* = 1, then the timer starts counting only after the first DMA request is received from the peripheral.

To calculate the timeout period, use *Equation 8-1*, *below*.

*Equation 8-1: Timeout Equation for Standard DMA*

$$T_{timeout} = T_{HCLK} \times N_{pssel} \times N_{tosel}$$

For example, if $T_{HCLK}$ = 1/90MHz, $N_{pssel}$ = 0x2 ⇒ 65536 timer prescalar, and $N_{tosel}$ = 0x3 ⇒ 32 clocks, then the timeout calculation is:

*Equation 8-2:Standard DMA Timeout Example Calculation*

$$T_{timeout} = \left(\frac{1}{90{,}000{,}000}\right) \times 65{,}536 \times 32 = 23.3ms$$

## 8.11 Channel and Register Access Restrictions

Writing to any register while a channel is disabled is supported, but there are certain restrictions when a channel is enabled. The *DMAn_ST*.*ch_st* bit indicates whether the channel is enabled or not.

Because an active channel might be in the middle of an AHB read/write burst, do not write to the *DMAn_SRC*, *DMAn_DST*, or *DMAn_CNT* registers while a channel is active (*DMAn_ST*.*ch_st* = 1).

To disable any DMA channel, clear the *DMA_CN*.*chien* bit. Then, poll the *DMAn_ST*.*ch_st* bit to verify that the channel is disabled.

## 8.12 Memory-to-Memory DMA

Memory-to-memory transfers are completed 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.

## 8.13 Standard DMA Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the Standard DMA (DMA_) Base Peripheral Address

*Table 8-7: Standard DMA Registers, Offsets, Access and Descriptions*

| Offset | Register | Access | Description |
|---|---|---|---|
| [0x0000] | *DMA_CN* | R/W | DMA Control register |
| [0x0004] | *DMA_INT* | RO | DMA Interrupt Status register |

## 8.14 Standard DMA Register Details

*Table 8-8: DMA Control Register*

| DMA Control Register | | | DMA_CN | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15:0 | chien | R/W | 0 | **Channel Interrupt Enable**<br>Each bit in this field enables the corresponding channel interrupt.<br>0: Channel interrupt disabled<br>1: Channel interrupt enabled |

*Table 8-9: DMA Interrupt Register*

| DMA Interrupt Register | | | DMA_INT | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15:0 | ipend | RO | 0 | **Channel Interrupt**<br>Each bit in this field represents an interrupt for the corresponding channel. To clear an interrupt, clear the corresponding active interrupt bit in the *DMAn_ST* register. An interrupt bit in this field is set only if the corresponding interrupt enable field is set in the *DMA_CN* register.<br>0: No interrupt<br>1: Interrupt pending |

## 8.15 Standard DMA Channel Register Offsets

*Table 8-10: Standard DMA Channel 0 to Channel 15 Offsets*

| Offset | DMA Channel | Access | Description |
|---|---|---|---|
| [0x0100] | DMA0 | R/W | DMA Channel 0 |
| [0x0120] | DMA1 | R/W | DMA Channel 1 |
| [0x0140] | DMA2 | R/W | DMA Channel 2 |
| [0x0160] | DMA3 | R/W | DMA Channel 3 |
| [0x0180] | DMA4 | R/W | DMA Channel 4 |
| [0x0200] | DMA5 | R/W | DMA Channel 5 |
| [0x0220] | DMA6 | R/W | DMA Channel 6 |
| [0x0240] | DMA7 | R/W | DMA Channel 7 |
| [0x0260] | DMA8 | R/W | DMA Channel 8 |
| [0x0280] | DMA9 | R/W | DMA Channel 9 |
| [0x0300] | DMA10 | R/W | DMA Channel 10 |
| [0x0320] | DMA11 | R/W | DMA Channel 11 |

| Offset | DMA Channel | Access | Description |
|---|---|---|---|
| [0x0340] | DMA12 | R/W | DMA Channel 12 |
| [0x0360] | DMA13 | R/W | DMA Channel 13 |
| [0x0380] | DMA14 | R/W | DMA Channel 14 |
| [0x0400] | DMA15 | R/W | DMA Channel 15 |

## 8.16 Standard DMA Channel Registers

Each DMA channel has a set of associated Configuration Registers. *Table 8-11* shows the addresses of these associated registers with respect to the channel base address. Because the registers are identical for all channels, only registers associated with DMA Channel 0 are shown in *Table 8-11*. The base address for channel 0 is 0x4002 8100 from *Table 8-7.*

*Table 8-11: DMAn Channel Registers, Offsets, Access and Descriptions*

| Offset | Register | Access | Description |
|---|---|---|---|
| [0x0000] | *DMAn_CFG* | R/W | DMAn Channel Configuration register |
| [0x0004] | *DMAn_ST* | R/W | DMAn Channel Status register |
| [0x0008] | *DMAn_SRC* | R/W | DMAn Channel Source register |
| [0x000C] | *DMAn_DST* | R/W | DMAn Channel Destination register |
| [0x0010] | *DMAn_CNT* | R/W | DMAn Channel Count register |
| [0x0014] | *DMAn_SRC_RLD* | R/W | DMAn Channel Source Reload register |
| [0x0018] | *DMAn_DST_RLD* | R/W | DMAn Channel Destination Reload register |
| [0x001C] | *DMAn_CNT_RLD* | R/W | DMAn Channel Count Reload register |

## 8.17 Standard DMA Channel Register Details

*Table 8-12: DMA Configuration Register*

| DMA Configuration Register | | | DMAn_CFG | [0x0100] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31 | ctzien | R/W | 0 | **CTZ Interrupt Enable**<br>When enabled, the *DMA_INT.ipend* bit is set to 1 whenever a CTZ event occurs.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 30 | chdien | R/W | 0 | **Channel Disable Interrupt Enable**<br>When enabled, the *DMA_INT.ipend* bit is set to 1 whenever the *DMAn_ST.ch_st* bit changes from 1 to 0.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 29 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| DMA Configuration Register | | | DMAn_CFG | [0x0100] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 28:24 | brst | R/W | 0 | **Burst Size**<br>The number of bytes transferred into and out of the DMA FIFO in a single burst.<br>0b00000: 1 byte<br>0b00001: 2 bytes<br>0b00010: 3 bytes<br>...<br>0b11111: 32 bytes |
| 23 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 22 | distinc | R/W | 0 | **Destination Increment Enable**<br>This bit enables the automatic increment of the *DMAn_DST* register upon every AHB transaction. This bit is forced to 0 for a DMA transmit to peripherals.<br>0: Increment disabled<br>1: Increment enabled |
| 21:20 | dstwd | R/W | 0 | **Destination Width**<br>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).<br>0: One byte<br>1: Two bytes<br>2: Four bytes<br>3: Reserved (Byte width if set) |
| 19 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 18 | srinc | R/W | 0 | **Source Increment Enable**<br>This bit enables the automatic increment of the *DMAn_SRC* register upon every AHB transaction. This bit is forced to 0 for a DMA receive from peripherals.<br>0: Increment disabled<br>1: Increment enabled |
| 17:16 | srcwd | R/W | 0 | **Source Width**<br>Indicates the width of each AHB transaction from the source peripheral or memory. The actual width might be less than this if the *DMAn_CNT* register indicates a smaller value.<br>0: One byte<br>1: Two bytes<br>2: Four bytes<br>3: Reserved (1 byte wide if set) |
| 15:14 | pssel | R/W | 0 | **Pre-Scale Select**<br>Selects the Pre-Scale divider for the timer clock input.<br>0: Timer disabled.<br>1: $\frac{f_{HCLK}}{256}$<br>2: $\frac{f_{HCLK}}{64K}$<br>3: $\frac{f_{HCLK}}{16M}$ |

| DMA Configuration Register | | | DMAn_CFG | [0x0100] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 13:11 | tosel | R/W | 0 | **Timeout Select**<br>Selects the number of pre-scaled clocks seen by the channel timer before a timeout condition is generated for this channel.<br>0: 3-4<br>1: 7-8<br>2: 15-16<br>3: 31-32<br>4: 63-64<br>5: 127-128<br>6: 255-256<br>7: 511-512 |
| 10 | reqwait | R/W | 0 | **Request Wait Enable**<br>When enabled, delay the timeout timer start until after the first DMA transaction occurs.<br>0: Start timer normally<br>1: Delay timer start |
| 9:4 | reqsel | R/W | 0 | **Request Select**<br>Select DMA request line for this channel. If memory to memory is selected, then the channel operates as if the request is always active. |
| 3:2 | pri | R/W | 0 | **DMA priority**<br>0: Highest priority<br>3: Lowest priority |
| 1 | rlden | R/W | 0 | **Reload Enable**<br>Setting this bit to 1 allows reloading the *DMAn_SRC*, *DMAn_DST*, and *DMAn_CNT* registers with their corresponding reload registers upon CTZ.<br>*Note: This bit is also writeable in the DMAn_CNT_RLD register.* |
| 0 | chen | R/W | 0 | **Channel Enable**<br>This bit is automatically cleared when *DMAn_ST.ch_st* changes from 1 to 0.<br>0: Disable this channel<br>1: Enable this channel |

*Table 8-13: DMA Status Register*

| DMA Status Register | | | DMAn_ST | [0x0104] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:7 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 6 | to_st | R/W1C | 0 | **Timeout Status**<br>Timeout status field. Write 1 to clear.<br>0: No time out<br>1: A time out has occurred |
| 5 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| DMA Status Register | | | DMAn_ST | [0x0104] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4 | bus_err | R/W1C | 0 | **Bus Error**<br>If this bit reads 1, an AHB abort occurred and the channel was disabled by hardware. Write 1 to clear.<br>0: No error found<br>1: An AHB bus error occurred |
| 3 | rld_st | R/W1C | 0 | **Reload Status**<br>Reload status field. Write 1 to clear.<br>0: Reload has not occurred.<br>1: Reload occurred. |
| 2 | ctz_st | R/W1C | 0 | **CTZ Status**<br>Read:<br>0: CTZ has not occurred<br>1: CTZ has occurred<br>Write:<br>0: No effect<br>1: Write 1 to clear |
| 1 | ipend | RO | 0 | **Channel Interrupt.**<br>Channel interrupt pending status.<br>0: No interrupt<br>1: Interrupt pending |
| 0 | ch_st | RO | 0 | **Channel Status**<br>This bit is used to indicate when it is safe to change the configuration, address, and count registers for the channel.<br>Whenever this bit is cleared by hardware, the *DMAn_CFG*.*chen* bit is also cleared.<br>0: Channel disabled<br>1: Channel enabled |

*Table 8-14: DMA Source Register*

| DMA Source Register | | | DMAn_SRC | [0x0108] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | src | R/W | 0 | **Source Device Address**<br>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.<br>If *DMAn_CFG*.*srcinc* = 1, then this register is incremented on each AHB transfer cycle by one, two, or four bytes depending on the data width.<br>If *DMAn_CFG*.*srcinc* = 0, this register remains constant.<br>If a CTZ condition occurs while *DMAn_CFG*.*rlden* = 1, then this register is reloaded with the contents of the *DMAn_SRC_RLD* register. |

*Table 8-15: DMA Destination Register*

| DMA Destination Register | | | DMAn_DST | [0x010C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | dst | R/W | 0 | **Destination Device Address**<br>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.<br>If *DMAn_CFG.dstinc* = 1, then this register is incremented on every AHB transfer cycle by one, two, or four bytes depending on the data width.<br>If a CTZ condition occurs while *DMAn_CFG.rlden* = 1, then this register is reloaded with the contents of the *DMAn_DST_RLD* register. |

*Table 8-16: DMA Count Register*

| DMA Count Register | | | DMAn_CNT | [0x0110] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 23:0 | cnt | R/W | 0 | **DMA Counter**<br>Load this register with the number of bytes to transfer. This field (*cnt*) 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.<br>If a CTZ condition occurs while *DMAn_CFG.rlden* = 1, then this register is reloaded with the contents of the *DMAn_CNT_RLD* register.<br>0x0: 0 Bytes<br>0x1: 1 Byte<br>0x2: 2 Bytes<br>…<br>0xFFFFFF: 16,777,215 Bytes |

*Table 8-17: DMA Source Reload Register*

| DMA Source Reload Register | | | DMAn_SRC_RLD | [0x0114] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 30:0 | src_rld | R/W | 0 | **Source Address Reload Value**<br>If *DMAn_CFG.rlden* = 1, then the value of this register is loaded into *DMAn_SRC* upon a CTZ condition. |

*Table 8-18: DMA Destination Reload Register*

| DMA Destination Reload Register | | | DMAn_DST_RLD | [0x0118] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| DMA Destination Reload Register | | | DMAn_DST_RLD | [0x0118] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 30:0 | dst_rld | R/W | 0 | **Destination Address Reload Value**<br>If *DMAn_CFG.rlden* = 1, then the value of this register is loaded into *DMAn_DST* upon a CTZ condition. |

*Table 8-19: DMA Count Reload Register*

| DMA Count Reload Register | | | DMAn_CNT_RLD | [0x011C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31 | rlden | R/W | 0 | **Reload Enable.**<br>Enables automatic loading of the *DMAn_SRC*, *DMAn_DST*, and *DMAn_CNT* registers when a CTZ event occurs. Set this bit after the address reload registers are programmed.<br>*Note: This bit is automatically cleared to 0 when reload occurs.*<br>*Note: This bit is also seen in the DMAn_CFG register.*<br>0: Reload disabled<br>1: Reload enabled |
| 30:24 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 23:0 | cnt_rld | R/W | 0 | **Count Reload Value.**<br>If *DMAn_CNT_RLD.rlden* = 1, then the value of this register is loaded into *DMAn_CNT* upon a CTZ condition. |

# 9 CRC Engine

## 9.1 Overview

Cyclic Redundancy Checks (CRC) are commonly used for error detection. An n-bit CRC is capable of detecting the following types of errors:

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

Overall, all except 1 out of $2^n$ errors are detected:

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

The hardware accelerator calculates the CRC of a block of data. Data is written to the Crypto data register.

The starting initial CRC value is typically preset to all ones. If the starting initial value is preset to all zeros and an initial stream of all zeros is processed as the data, the CRC does not change.

Historically, CRCs were calculated on serial bit streams. Most serial bit streams were sent least significant bit first. The CRC was calculated as each bit was transmitted or received. This resulted in the CRC being calculated on the least significant bit of the data first.

The CRC is typically appended to the end of the data. If the receiver calculates the CRC on both the data and received CRC, the result should be all zeros if the data and CRC were received error free. Most implementations do not like to check against an all zero checksum. Therefore, most implementations invert the CRC before transmitting it. By inverting the CRC on the transmitting end, the resulting CRC on the receiving end should be a constant. The specific constant is dependent upon the CRC polynomial. This works because the non-inverted CRC calculated at the end of the data XOR'd with the received inverted CRC is all ones ($CRC \oplus \sim CRC = 1s$). Shifting all ones through the polynomial results in the same constant for each message, the constant is dependent upon the polynomial.

Because the receiving end calculates a new CRC on both the data and received CRC, the you must send the received CRC in the correct order, so the highest-order term of the CRC is shifted through the generator first. Because data is typically shifted through the generator LSB first, this means the CRC is reversed bit-wise, with the highest-order term of the remainder in the LSB position. Software CRC algorithms typically handle this by calculating everything backwards. They reverse the polynomial and do right shifts on the data. The resulting CRC ends up being bit swapped and in the correct format.

The CRYPTO_CRC register is preset to all ones if the crypto block is reset. The initial CRC state is written to any value. The final inversion must be done by software if required.

The CRC generator has a programmable polynomial up to 32-bits. The polynomial should be written to the *CRC_POLY* register. The largest term $x^n$ defines the length of the CRC. When calculating the CRC on data LSB first, the polynomial should be reversed so that the coefficient of the highest power term is in the LSB position. The largest term $x^n$ is implied (always one) and should be omitted when writing to the *CRC_POLY* register. This is necessary because the polynomial is always one bit larger than the resulting CRC, so a 32-bit CRC has a polynomial with 33 terms ($x^0$ … $x^{32}$).

CRC polynomials with good error detection properties should be irreducible (the polynomial should not be factorable). Therefore, the constant term $x^0$ or 1 should always be present, otherwise the polynomial would be factorable by $x$. If the constant term $x^0$ or 1 were not present, the resulting CRC would be cyclic with a subgroup smaller than $x^n$. The effective length of the CRC would be the difference between the highest and lowest order terms. Therefore, the highest and lowest order terms $x^n$and $x^0$ should always appear in the polynomial.

When found in literature, sometimes the LSB or MSB of the polynomial is omitted when the polynomial is written in binary. It is more common to see CRC polynomials with the MSB implied because that is the bit that is shifted off, XOR'd with the data, and tested to see if the result is set. Some literature assumes the reader knows that an *n*-bit CRC must have the $x^n$ term set, or else it would be a smaller length CRC.

Some common CRC polynomials and their check constants are shown in the table below. The polynomial register resets to the 32-bit CRC polynomial used by Ethernet, PPP and file compression utilities such as zip or gzip.

*Table 9-1: Common CRC Polynomials*

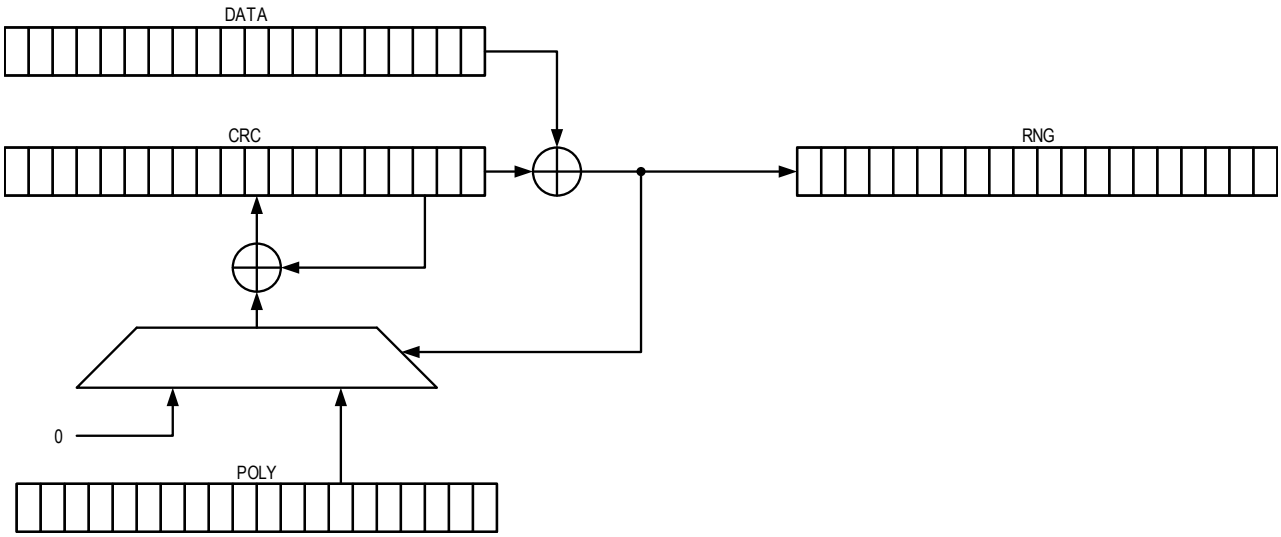
| Algorithm | Polynomial Expression | Order | Polynomial (*CRC_POLY*) | Check |
|---|---|---|---|---|
| CRC-32 Ethernet | $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+x^0$ | LSB | 0xEDB8 8320 | 0xDEBB 20E3 |
| CRC-CCITT | $x^{16}+x^{12}+x^5+x^0$ | LSB | 0x0000 8408 | 0x0000 F0B8 |
| CRC-16 | $x^{16}+x^{15}+x^2+x^0$ | LSB | 0x0000 A001 | 0x0000 B001 |
| USB Data | $x^{16}+x^{15}+x^2+x^0$ | MSB | 0x8005 0000 | 0x800D 0000 |
| Parity | $x^1+x^0$ | LSB | 0x0000 0001 | |

By default, the CRC accelerator does right shifts and calculates the CRC on the LSB of the data first. The CRC can be calculated on the MSB of the data first by setting the bit-swap control bit to 1(*CRC_CTRL*.*msb* = 1). To calculate the CRC MSB first, you must left justify the polynomial in the *CRC_POLY* register. The hardware implies the MSB of the polynomial just as it did when shifting the LSB first. The LSB of the polynomial should be set, this defines the length of the CRC. The initial state of the CRC should also be left justified. When the CRC calculation is complete, it is necessary to right shift the CRC to right justify it if the polynomial is less than 32-bits.

### 9.1.1 *Linear Feedback Shift Register*

Linear Feedback Shift Registers (LFSR) are commonly used to implement Pseudo Random Number Generators (PRNG). A LFSR polynomial can be written to the *CRC_POLY* register to generate pseudo-random data. The starting state or seed for the pseudo random sequence should be written to the CRC register. The lockup state of all zeros is detected, and the LFSR substitutes the value 1 to prevent lockup.

*Figure 9-1: Galois Field CRC and LFSR Architecture*




Different polynomials generate different sequences of random data. Ideally, an n-bit polynomial generates a random sequence of $2^n - 1$ bits. Not all polynomials are maximal length. Some repeat before the theoretical maximum length of $2^n - 1$. There are thousands of different maximal length 32-bit LFSR polynomials. You can use any length of an LFSR polynomial up to 32bits. Some tables of maximal length LFSR polynomials omit the MSB $(x^n)$ term or the LSB $(x^0 = 1)$ term. Fibonacci LFSRs feedback the XOR of all the taps to the constant term $x^0 = 1$. It is often implied when listing the taps but must be present when writing the polynomial to the *CRC_POLY* register.

The crypto accelerator automatically generates the next sequence of 32bits whenever the CRYPTO_LFSR register is read. If the PRNG control bit is set, the incoming data is forced to zero. You can use the DMA to quickly fill a block of memory with pseudo-random data.

## 9.2 CRC Registers

Refer to *Table 2-1* for the CRC Base Peripheral Address.

*Table 9-2: CRC Registers*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *CRYPTO_CTRL* | R/W | Crypto Control Register |
| [0x000C] | *CRC_CTRL* | R/W | CRC Control Register |
| [0x0010] | *CRC_DMA_SRC* | R/W | CRC DMA Source Address Register |
| [0x0014] | *CRC_DMA_DST* | R/W | CRC DMA Destination Address Register |
| [0x0018] | *CRC_DMA_CNT* | R/W | CRC DMA Byte Count Register |
| [0x0020] | *CRC_DATA_IN0* | R/W | CRC Data Input Register 0 (Bits 31:0) |
| [0x0024] | *CRC_DATA_IN1* | R/W | CRC Data Input Register 0 (Bits 63:32) |
| [0x0028] | *CRC_DATA_IN2* | R/W | CRC Data Input Register 0 (Bits 95:64) |
| [0x002C] | *CRC_DATA_IN3* | R/W | CRC Data Input Register 0 (Bits 127:96) |
| [0x0030] | *CRC_DATA_OUT0* | R/W | CRC Data Output Register 0 (Bits 31:0) |

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0034] | *CRC_DATA_OUT1* | R/W | CRC Data Output Register 0 (Bits 63:32) |
| [0x0038] | *CRC_DATA_OUT2* | R/W | CRC Data Output Register 0 (Bits 95:64) |
| [0x003C] | *CRC_DATA_OUT3* | R/W | CRC Data Output Register 0 (Bits 127:96) |
| [0x0040] | *CRC_POLY* | R/W | CRC Polynomial Register |
| [0x0044] | *CRC_VAL* | R/W | CRC Value Register |
| [0x0048] | *CRC_PRNG* | R/W | CRC Pseudo-random Number Register |

## 9.3 CRC Register Details

*Table 9-3. Flash Controller Address Pointer Register*

| Crypto Control Register | | | CRYPTO_CTRL | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31 | done | RO | 0 | **Done**<br>Done bit indicator.<br>0: CRC not done.<br>1: CRC done. |
| 30 | rdy | R0 | 1 | **Ready**<br>CRC engine ready for operation.<br>0: CRC engine busy<br>1: CRC engine ready for operation |
| 29 | err | RO | 0 | **Error Flag**<br>If this field reads 1 and error occurred.<br>0: No error condition.<br>1: Error occurred. |
| 28:25 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 24 | dma_done | R/W | 0 | **DMA Complete Flag**<br>This field is set to 1 when a DMA read/write operation is complete. Set this bit to 0 prior to starting a DMA CRC operation. |
| 23:16 | - | R/O | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | dmadne_msk | R/W | 0 | **DMA Done Flag Mask for DMA Operation**<br>This field sets the behavior of the *CRYPTO_CTRL*.*done* flag. Setting this field to 1 results in a *CRYPTO_CTRL*.*dma_done* complete to set the *CRYPTO_CTRL*.*done* field. The setting for this field does not affect the actual behavior of the *CRYPTO_CTRL*.*dma_done* flag.<br>0: DMA Complete Flag is not used for setting *CRYPTO_CTRL*.*done* field.<br>1: DMA Complete condition sets the *CRYPTO_CTRL*.*done* field. |

| Crypto Control Register | | | CRYPTO_CTRL | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 14 | flag_mode | R/W | 0 | **Done Flag Mode**<br>This field configures the access behavior of the *CRYPTO_CTRL.dma_done* field. When this field is set to 1, the *CRYPTO_CTRL.dma_done* field is Write 1 to Clear. When this field is 0, the *CRYPTO_CTRL.dma_done* field is unrestricted read/write.<br>0: *CRYPTO_CTRL.dma_done* field is unrestricted read/write access.<br>1: *CRYPTO_CTRL.dma_done* field is Write 1 to Clear.<br>*Note: This field is only reset on a Power-On Reset.* |
| 13:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11:10 | rdsrc | R/W | 0 | **Read FIFO Source Select**<br>This field selects the source of the read FIFO. The default is for the Read FIFO to not use DMA as it's source buffer.<br>0b00: DMA disabled<br>0b01: DMA or Direct via *CRC_DATA_IN3:CRC_DATA_IN0* registers<br>0b10: *CRC_PRNG*<br>0b11: Reserved for Future Use |
| 9:8 | wrsrc | R/W | 0 | **Write FIFO Source Select**<br>This field selects the source of the Write FIFO. The default is for the Write FIFO to not use DMA as it's source buffer.<br>0b00: DMA disabled<br>0b01: DMA or direct via *CRC_DATA_OUT3: CRC_DATA_OUT0* registers.<br>0b10: *CRC_PRNG*<br>0b11: Reserved for Future Use |
| 7:6 | 0 | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 5 | bsi | R/W | 0 | **Byte Swap Input**<br>Enables input word byte swapping if set.<br>0: Input bytes are not byte-swapped.<br>1: Input bytes are byte swapped.<br>*Note: Byte swap only occurs on full words.* |
| 4 | bso | R/W | 0 | **Byte Swap Output**<br>Enable output word byte swapping if set.<br>0: Output bytes are not byte-swapped.<br>1: Output bytes are byte swapped.<br>*Note: Byte swap only occurs on full words.* |
| 3 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 2 | src | R/W | 0 | **Source Select**<br>Selects the CRC generator input source.<br>0: CRC uses input FIFO.<br>1: CRC uses output FIFO. |
| 1 | int | R/W | 0 | **Interrupt Enable**<br>Set this field to 1 to generate a CRC/Crypto IRQ when an interrupt flag is set (done or error).<br>0: Interrupt disabled<br>1: Interrupt enabled |

| Crypto Control Register | | | CRYPTO_CTRL | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | rst | R/W | 0 | **Reset**<br>Set this field to reset the CRC including the FIFOs. The *CRYPTO_CTRL* register is not reset.<br>0: Reset not active.<br>1: Reset the CRC Engine. |

*Table 9-3: CRC Control Register*

| CRC Control Register | | | CRC_CTRL | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | msb | R/W | 0 | **MSB Select**<br>Set the order of calculating the CRC on the input data.<br>0: LSB data first<br>1: MSB data first |
| 0 | 0 | R/W | 0 | **CRC Enable**<br>Enable the CRC function.<br>0: CRC function disabled<br>1: CRC function enable |

*Table 9-4: CRC DMA Source Register*

| CRC DMA Source Register | | | CRC_DMA_SRC | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | src_addr | R/W | 0 | **DMA Source Address**<br>Set this field to the address of the DMA source input data. |

*Table 9-5: CRC DMA Destination Register*

| CRC DMA Destination Register | | | CRC_DMA_DST | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | dst_addr | R/W | 0 | **DMA Destination Address**<br>Set this field to the address for the CRC DMA output data. |

*Table 9-6: CRC DMA Count Register*

| CRC DMA Count Register | | | CRC_DMA_CNT | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | count | R/W | 0 | **DMA Byte Count**<br>Set this field to the number of bytes of input data to the CRC engine. |

*Table 9-7: CRC Data Input Registers*

| | | |
|---|---|---|
| **CRC Data Input Register 0** | **CRC_DATA_IN0** | **[0x0020]** |
| **CRC Data Input Register 1** | **CRC_DATA_IN1** | **[0x0024]** |
| **CRC Data Input Register 2** | **CRC_DATA_IN2** | **[0x0028]** |
| **CRC Data Input Register 3** | **CRC_DATA_IN3** | **[0x002C]** |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | data | R/W | 0 | **CRC Data Input**<br>Writes to this register bypass the DMA interface and put data directly into the CRC FIFO. Data written to a register is placed into the FIFO in the order it is written. *CRC_DATA_IN3*:*CRC_DATA_IN0* represent 128 bits with *CRC_DATA_IN3* mapped to bits 127:96, *CRC_DATA_IN2* mapped to bits 95:64, *CRC_DATA_IN1* mapped to bits 63:32 and *CRC_DATA_IN0* mapped to bits 31:0.<br>*CRC_DATA_IN3*:*CRC_DATA_IN0* occupy four successive words to allow the use of multi-store instructions. Writes to any location are supported and writes are loaded into the CRC FIFO in the order they are written.<br>*Note: The CRC_DATA_INn registers are affected by the input endian swap bit, CRYPTO_CTRL.bsi.* |

*Table 9-8: CRC Data Output Registers*

| | | |
|---|---|---|
| **CRC Data Output Register 0** | **CRC_DATA_OUT0** | **[0x0030]** |
| **CRC Data Output Register 1** | **CRC_DATA_OUT1** | **[0x0034]** |
| **CRC Data Output Register 2** | **CRC_DATA_OUT2** | **[0x0038]** |
| **CRC Data Output Register 3** | **CRC_DATA_OUT3** | **[0x003C]** |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | data | R/W | 0 | **CRC Data Output**<br>The output from the CRC is written to the *CRC_DATA_OUT3:CRC_DATA_OUT0* registers representing a total of 128 bits. *CRC_DATA_OUT3* maps to bits 127:96, *CRC_DATA_OUT2* maps to bits 95:64, *CRC_DATA_OUT1* maps to bits 63:31, and *CRC_DATA_OUT0* represents output bits 31:0.<br>*Note: The CRC_DATA_OUTn registers are affected by the output endian swap bit, CRYPTO_CTRL.bso.* |

*Table 9-9: CRC Polynomial Register*

| | | |
|---|---|---|
| **CRC Polynomial Register** | **CRC_POLY** | **[0x0040]** |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | src_addr | R/W | 0 | **CRC Polynomial**<br>The polynomial used for Galois Field calculations (CRC or LFSR) is written to this register.<br>*Note: This register is affected by the MSB control bit, CRC_CTRL.msb.* |

*Table 9-10: CRC Value Register*

| CRC Value Register | | | CRC_VAL | [0x0044] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | val | R/W | 0 | **CRC Value**<br>This is the state for the Galois Field. Output of the CRC calculation or the current state of the LFSR.<br>*Note: This register is affected by the MSB control bit, CRC_CTRL.msb.* |

*Table 9-11: CRC Pseudo-Random Number Generator Register*

| CRC PRNG Register | | | CRC_PRNG | [0x0048] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | prng | R/W | 0 | **CRC Pseudo Random Value**<br>Output of the Galois Field shift register. This register holds the resulting pseudo-random number.<br>*Note: This register is affected by the MSB control bit, CRC_CTRL.msb.* |

# 10 Analog to Digital Converter

The Analog to Digital Converter (ADC) on the MAX32650–MAX32652 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 the external analog input signals (AIN0, AIN1, AIN2, and AIN3) or the internal power supply inputs. The 10-bit ADC conversions are stored as a 16-bit value selectable as most-significant bit (MSB) or least-significant bit (LSB) aligned

## 10.1 Features

- 8MHz maximum ADC clock rate
- Two reference sources, an internal 1.22V bandgap or the VDDA analog supply
- Fixed 10-bit word conversion time of 1024 ADC clock cycles
- AIN0 and AIN1 input range up to 5.5V
- AIN2 and AIN3 input range up to VDDA
- 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-bit output either MSB or LSB aligned

## 10.2 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 10-1*. Details of selecting the ADC clock frequency, $f_{adcclk}$, are covered in the Clock Configuration section.

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

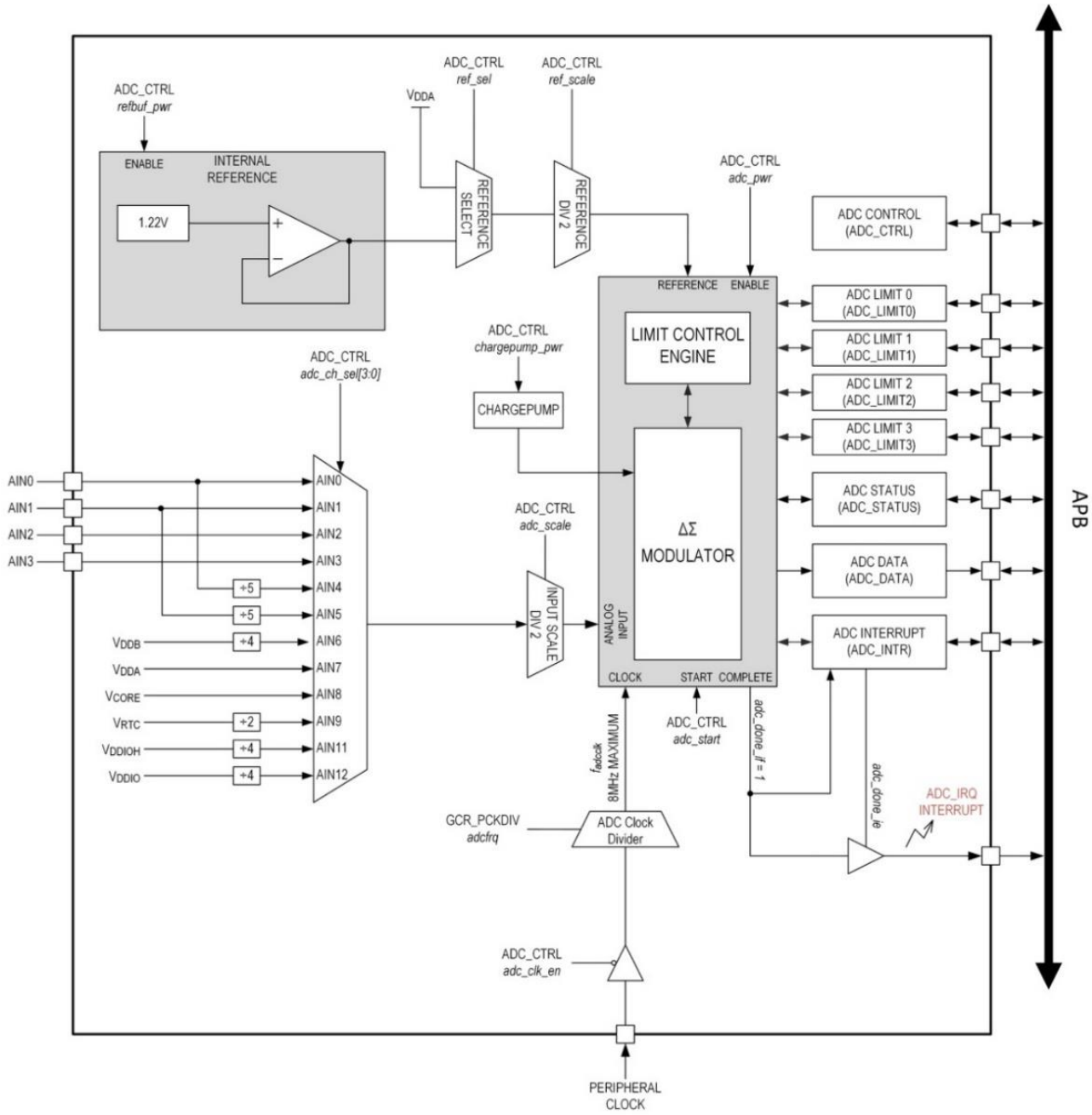
$$t_{adc\_sample} = 1024 \times \left(\frac{1}{f_{adcclk}}\right)$$

ADC offset and gain errors are factory trimmed and automatically loaded into the ADC controller during system power-up. Gain error is trimmed to null out the total errors of the ADC and internal reference.

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 – AN3). 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 10-1: Analog to Digital Converter Block Diagram*




## 10.3 Clock Configuration

The ADC clock, *adcclk*, is controlled by the GCR_PCKDIV.adcfrq register field. Configure this field a for the target ADC sample frequency. The maximum clock supported by the ADC is 8MHz. The divisor selection, *GCR_PCLK_DIV.adcfrq*, for the ADC depends on the peripheral clock. *Equation 10-2* shows the calculation for the ADC clock frequency, where $f_{PCLK} = {f_{SYSCLK}}/{2}$.

*Equation 10-2: ADC Clock Frequency*

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

The GCR_PCKDIV.adcfrq register field setting must result in a value for $f_{adcclk} \leq 8MHz$ as shown in *Table 10-1* with the System Clock set as the 120MHz Relaxation Oscillator.

*Table 10-1: ADC Clock Frequency and ADC Conversion Time ($f_{SYSCLK} = 120MHz$, $f_{PCLK} = 60MHz$)*

| 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 | 7,500,000 | 137 |
| 0x9 | 6,666,667 | 154 |
| 0xA | 6,000,000 | 171 |
| 0xB | 5,454,545 | 188 |
| 0xC | 5,000,000 | 205 |
| 0xD | 4,615,385 | 222 |
| 0xE | 4,285714 | 239 |
| 0xF | 4,000,000 | 256 |

## 10.4 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 the Global Control Register field, *GCR_PCLK_DIV.adcfrq*. Refer to *Clock Configuration* for details.
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:
   a. Internal 1.22V bandgap reference (*ADC_CTRL.ref_sel* = 0).
   b. $V_{DDA}$ reference (*ADC_CTRL.ref_sel* = 1).
7. Complete the following steps to enable power:
   a. Set *ADC_CTRL.pwr* to 1 to turn on the ADC.
   b. Set *ADC_CTRL.refbuf_pwr* to 1 to turn on the internal reference buffer If using the internal bandgap reference.
   c. Set *ADC_CTRL.chargepump_pwr* to 1 to turn on the ADC charge pump. Note: The ADC charge pump takes approximately 10µs to fully charge and stabilize.
   d. Wait until hardware sets the *ADC_INTR.ref_ready_if* bit to 1 indicating the charge pump is fully stabilized.
   e. Clear the ADC reference ready interrupt flag by writing 1 to *ADC_INTR.ref_ready_if*.
   f. Optionally disable the ADC reference ready interrupt (*ADC_INTR.ref_ready_ie* = 0).

## 10.5 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. Refer to 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. Refer to Table 10-3 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). Refer to 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*.

## 10.6 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, except AIN7 and AIN8, supports the default of no scaling of the input (*ADC_CTRL.input_scale* = 0) and no scaling of the reference (*ADC_CTRL.ref_scale* = 0). For AIN7 and AIN8, the ADC scale (*ADC_CTRL.input_scale*) must be set to 1 and the reference scale (*ADC_CTRL.ref_scale*) must be set to 0. The following sections describe the scale options for each of the ADC input channels.

### 10.6.1 *AIN0 – AIN3 Scale Limitations*

The external inputs, AIN0 through AIN3, support scaling of the input by 50%, the reference by 50%, or both by 50%. The scale settings for the given input signal and reference must satisfy the following equation to be valid:

*Equation 10-3: Input and Reference Scale Requirements Equation*

$$\frac{AIN}{2^{adc\_scale}} < \frac{V_{REF}}{2^{ref\_scale}}$$

### 10.6.2 *AIN7 – AIN8 Scale Limitations*

Analog input channels AIN7 and AIN8 must use the following settings for the ADC scale and reference scale for all measurements of these channels.

- *ADC_CTRL.input_scale* = 1
- *ADC_CTRL.ref_scale* = 0

### 10.6.3 *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 10-2*, *below*.

*Table 10-2: Input and Reference Scale Support by ADC Input Channel*

| ADC Channel | ADC_CTRL input_scale | ADC_CTRL ref_scale |
|---|---|---|
| AIN4 | 0 | 0 |
| | 1 | 1 |
| AIN5 | 0 | 0 |
| | 1 | 1 |
| AIN6 | 0 | 0 |
| | 1 | 1 |
| AIN9 | 0 | 0 |
| | 1 | 1 |
| AIN10 | NA | NA |
| AIN11 | 0 | 0 |
| | 1 | 1 |
| AIN12 | 0 | 0 |
| | 1 | 1 |

### 10.6.4 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 10-3* shows the ADC data alignment based on the value of the ADC_CTRL.*data_align* bit.

*Table 10-3: 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 |

### 10.6.5 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, $V_{REF}$ = 1.22V; otherwise $V_{REF}$ = VDDA.

*Equation 10-4: ADC Data Calculation for AIN0 – AIN3*

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

*Note: Must satisfy Equation 10-3.*

*Equation 10-5: ADC Data Equation for AIN4 and AIN5*

$$ADC\_DATA = round\left\{\left(\frac{\left(\frac{\frac{AIN}{5}}{2^{adc\_scale}}\right)}{\left(\frac{V_{REF}}{2^{ref\_SCALE}}\right)}\right) \times (2^{10} - 1)\right\}$$

*Note: Refer to Table 10-2 for limitations.*

*Equation 10-6: ADC Data Calculation AIN6*

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

*Note: Refer to Table 10-2 for limitations.*

*Equation 10-7: ADC Data Calculation for AIN7*

$$ADC\_DATA = round\left\{\left(\frac{\left(\frac{V_{DDA}}{2}\right)}{V_{REF}}\right) \times (2^{10} - 1)\right\}$$

*Note: input_scale = 1 and ref_scale = 0.*

*Equation 10-8: ADC Data Calculation for AIN8*

$$ADC\_DATA = round\left\{\left(\frac{\left(\frac{V_{CORE}}{2}\right)}{V_{REF}}\right) \times (2^{10} - 1)\right\}$$

*Note: input_scale = 1 and ref_scale = 0.*

*Equation 10-9: ADC Data Calculation for AIN9*

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

*Note: Refer to Table 10-2 for limitations.*

*Equation 10-10: ADC Data Calculation for AIN11*

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

*Note: Refer to Table 10-2 for limitations.*

*Equation 10-11: ADC Data Calculation for AIN12*

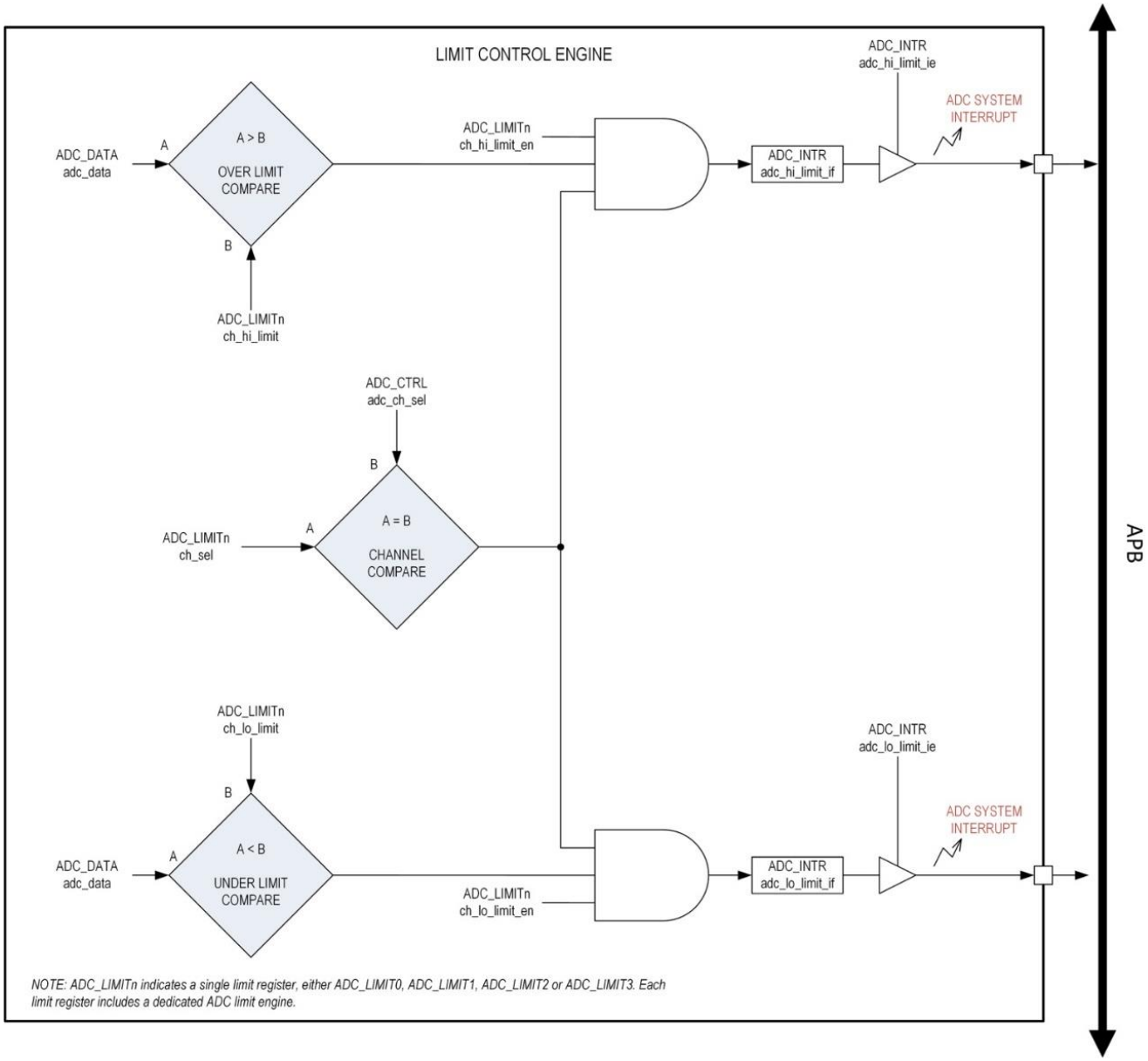
$$ADC\_DATA = round\left\{\left(\frac{\left(\frac{V_{DDIOH}}{4}\right)}{\left(\frac{V_{REF}}{2^{ref\_scale}}\right)}\right) \times (2^{10} - 1)\right\}$$

*Note: Refer to Table 10-2 for limitations.*

### 10.6.6 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 10-2*.

*Figure 10-2: 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.

### 10.6.7 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.chargepump_pwr* to 0, disabling the ADC charge pump.
4. Set *ADC_CTRL.clk_en* to 0, disabling the ADC internal clock.

## 10.7 ADC Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the ADC Base Peripheral Address.

Address assignments for the ADC registers are shown in *Table 10-4*. Detailed descriptions of each ADC register and the fields within each register are shown in the following tables. Do not modify bits and fields marked as *Reserved for Future Use*.

*Table 10-4. ADC Registers, Offsets and Descriptions*

| Offset | Register 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 |

## 10.8 ADC Register Details

*Table 10-5: ADC Control Register*

| ADC Control Register | ADC_CTRL | [0x0000] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:18 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 17 | data_align | R/W | 0 | **ADC Data Alignment**<br>Selects the alignment of the 16-bit data conversion stored in the DATA register.<br>0: Data is LSB justified in 16-bit DATA register. DATA[15:10] = 0.<br>1: Data is MSB justified in 16-bit DATA register. DATA[5:0] = 0. |
| 16 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15:12 | ch_sel | R/W | 0 | **ADC Channel Select**<br>Selects the active channel for the next ADC conversion. |

| ch_sel | ADC Input Channel | Input |
|---|---|---|
| 0x0 | AIN0 | AIN0 |
| 0x1 | AIN1 | AIN1 |
| 0x2 | AIN2 | AIN2 |
| 0x3 | AIN3 | AIN3 |
| 0x4 | AIN4 | AIN0 / 5 |
| 0x5 | AIN5 | AIN1 / 5 |
| 0x6 | AIN6 | $V_{DDB}$ / 4 |
| 0x7 | AIN7 | $V_{DDA}$ |
| 0x8 | AIN8 | $V_{CORE}$ |
| 0x9 | AIN9 | $V_{RTC}$ / 2 |
| 0xA | AIN10 | Reserved for Future Use |
| 0xB | AIN11 | $V_{DDIO}$ / 4 |
| 0xC | AIN12 | $V_{DDIOH}$ / 4 |

| | | | | |
|---|---|---|---|---|
| 11 | clk_en | R/W | 0 | **ADC Clock Enable**<br>Enables the ADC internal clock.<br>0: ADC internal clock disabled<br>1: ADC internal clock enabled |
| 10 | ref_sel | R/W | 0 | **ADC Reference Select**<br>0: Internal bandgap reference is used for the ADC reference<br>1: VDDA is used for the ADC reference |
| 9 | input_scale | R/W | 0 | **ADC Input Scale**<br>Scales ADC input by 50 percent.<br>0: ADC input is not scaled<br>1: ADC input is scaled by ½.<br>*Note: Refer to Reference Scaling and Input Scaling for valid settings for each ADC input.* |
| 8 | ref_scale | R/W | 0 | **Reference Scale**<br>Scales the reference by 50 percent.<br>0: Reference is not scaled.<br>1: Reference is scaled by ½.<br>*Note: Refer to Reference Scaling and Input Scaling for valid settings for each ADC input* |
| 7:5 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| ADC Control Register | | | ADC_CTRL | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4 | chgpump_pwr | R/W | 0 | **ADC Charge Pump**<br>Setting this bit to 1 turns on the ADC charge pump required to perform ADC conversions.<br>0: Charge pump is powered off<br>1: Charge pump is powered on |
| 3 | refbuf_pwr | R/W | 0 | **Reference Buffer Power Enable**<br>Enables the reference buffer power for the internal bandgap reference.<br>0: Reference buffer powered off<br>1: Reference buffer powered on |
| 2 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | pwr | R/W | 0 | **ADC Power Enable**<br>0: ADC is powered off<br>1: ADC is powered on |
| 0 | start | R/W | 0 | **Start ADC Conversion**<br>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.<br>0: ADC inactive or data conversion complete.<br>1: Start ADC conversion and remains set until complete. |

*Table 10-6: ADC Status Register*

| ADC Status Register | | | ADC_STATUS | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:4 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3 | overflow | RO | 0 | **ADC Overflow Flag**<br>If this bit is set, the last conversion performed by the ADC resulted in an overflow condition.<br>0: No overflow on last conversion<br>1: Overflow on last conversion |
| 2 | pwr_up_active | RO | 0 | **ADC Power-Up State**<br>This field is set to 1 when the ADC charge pump is powering up.<br>0: AFE is not in power-up delay.<br>1: AFE is currently in the power-up delay state. |
| 1 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 0 | active | RO | 0 | **ADC Conversion in Progress**<br>0: ADC is idle<br>1: ADC conversion is in progress |

*Table 10-7: ADC Data Register*

| ADC Data Register | | | ADC_DATA | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | data | RO | 0 | **ADC Data**<br>This field contains the ADC conversion output data. *Data Conversion Output Alignment* section for details. |

*Table 10-8: ADC Interrupt Control Register*

| ADC Interrupt Control Register | | | ADC_INTR | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:23 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 22 | pending | RO | 0 | **ADC Interrupt Pending**<br>0: No ADC interrupt is pending for an enabled interrupt condition.<br>1: At least one ADC interrupt is pending, and the corresponding interrupt enable bit is set. |
| 21 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 20 | overflow_if | R/W1C | 0 | **ADC Overflow Interrupt Flag**<br>1: The last conversion resulted in an overflow |
| 19 | lo_limit_if | R/W1C | 0 | **ADC Low Limit Interrupt Flag**<br>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**<br>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**<br>1: The ADC reference is ready for use. |
| 16 | done_if | R/W1C | 0 | **ADC Conversion Complete Interrupt Flag**<br>Set by the ADC hardware when an ADC conversion is complete.<br>1: ADC conversion complete |
| 15:5 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 4 | overflow_ie | R/W | 0 | **ADC Overflow Interrupt Enable**<br>0: ADC overflow interrupt disabled.<br>1: Enables interrupt assertion when hardware sets *ADC_INTR.overflow_if*. |
| 3 | lo_limit_ie | R/W | 0 | **ADC Low Limit Interrupt Enable**<br>0: ADC low-limit interrupt disabled.<br>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**<br>0: ADC high-limit interrupt disabled.<br>1: Enables interrupt assertion when hardware sets *ADC_INTR.lo_limit_if*. |
| 1 | ref_ready_ie | R/W | 0 | **ADC Reference Ready Interrupt Enable**<br>0: ADC reference ready interrupt disabled.<br>1: Enables interrupt assertion when hardware sets *ADC_INTR.ref_ready_if*. |
| 0 | done_ie | R/W | 0 | **ADC Conversion Complete**<br>0: ADC reference ready interrupt disabled.<br>1: Enables interrupt assertion when hardware sets *ADC_INTR.done_if*. |

*Table 10-9: ADC Limit 0 to 3 Registers*

| ADC Limit 0 Register | | | ADC_LIMIT0 | [0x0010] |
|---|---|---|---|---|
| ADC Limit 1 Register | | | ADC_LIMIT1 | [0x0014] |
| ADC Limit 2 Register | | | ADC_LIMIT2 | [0x0018] |
| ADC Limit 3 Register | | | ADC_LIMIT3 | [0x001C] |
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:30 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

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

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 29 | ch_hi_limit_en | R/W | 0 | **High Limit Monitoring Enable**<br>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).<br>1: The high-limit comparison for the ch_sel channel is active.<br>0: The high-limit comparison is not enabled. |
| 28 | ch_lo_limit_en | R/W | 0 | **Low Limit Monitoring Enable**<br>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).<br>1: The low-limit comparison for the ch_sel channel is active.<br>0: The low-limit comparison is not enabled. |
| 27:24 | ch_sel | R/W | 0 | **ADC Channel for Limit Monitoring**<br>Sets the ADC input channel for high- and low-limit thresholds. Refer to ADC_CTRL.*ch_sel* for valid values for this field. |
| 23:22 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 21:12 | ch_hi_limit | R/W | 0x3FF | **High Limit Threshold**<br>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.<br>Valid values for this field are 0x000 to 0x3FF. |
| 11:10 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 9:0 | ch_lo_limit | R/W | 0 | **Low Limit Threshold**<br>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.<br>Valid values for this field are 0x000 to 0x3FF. |

# 11 Color LCD-TFT Controller

The Color LCD-TFT (CLCD) Controller provides a standard RGB (Red, Green, Blue) parallel interface and controls for horizontal and vertical synchronization, data enable, display enable and pixel clock for Liquid Crystal Displays (LCD) including Thin-Film-Transistor (TFT) panels with up to a 24-bit bus interface and Color STN panels with an 8-bit bus interface. The panel resolution is programmable from 320 × 200 to 4096 × 4096.

## 11.1 Features

*General*

- 24-bit RGB parallel data output, 8 bits per pixel (bpp)
- Programmable horizontal front porch (HFP) and horizontal back porch (HBP)
- Programmable horizontal synchronization pulse (HYSNC) width
- Programmable vertical front porch (VFP) and vertical back porch (VBP)
- Programmable vertical synchronization pulse (VSYNC) width
- Programmable number of bits per pixel (bpp)
- Programmable number of pixels per line
- Programmable number of display lines
- Polarity control for display control pins for horizontal sync, vertical sync, and data enable
- Programmable LCD clock output
- Selectable Endian format
- Little Endian Byte, Little Endian Pixel (LBLP)
- Big Endian Byte, Big Endian Pixel (BBBP)
- Little Endian Byte, Big Endian Pixel (LBBP)
- Interrupt Generation at start of vertical events including VSYNC, Vertical Back Porch (VBP), Active Video (AV), and Vertical Front Porch (VFP)

*TFT Panel Support*

- 1 bpp, 2 colors
- 2 bpp, 4 colors
- 4 bpp, 16 colors
- 8 bpp, 256 colors
- 16 bpp direct RGB565
- 16 bpp direct RGB555 with intensity bit
- 24 bpp direct RGB888

*Color STN Support*

- 1 bpp, 2 colors
- 2 bpp, 4 colors
- 4 bpp, 16 colors
- 8 bpp, 256 colors
- RGB555 with 16 bpp, 1 unused bit
- Typical Panel Resolutions
  - 320 × 200, 320 × 240
  - 640 × 200, 640 × 240, 640 × 480
  - 800 × 600
  - 1024 × 768
  - 2048 × 2048
  - 4096 × 4096

## 11.2 Functional Overview

*Figure 11-1*, *below*, shows the block diagram of the CLCD controller, depicting the flow of data to a color TFT display and a color STN panel. The CLCD performs translation of pixel-coded data into LCD format and generates the timing to drive single or dual panel displays in monochrome or color. The CLCD controller drives an external display by performing the following steps:

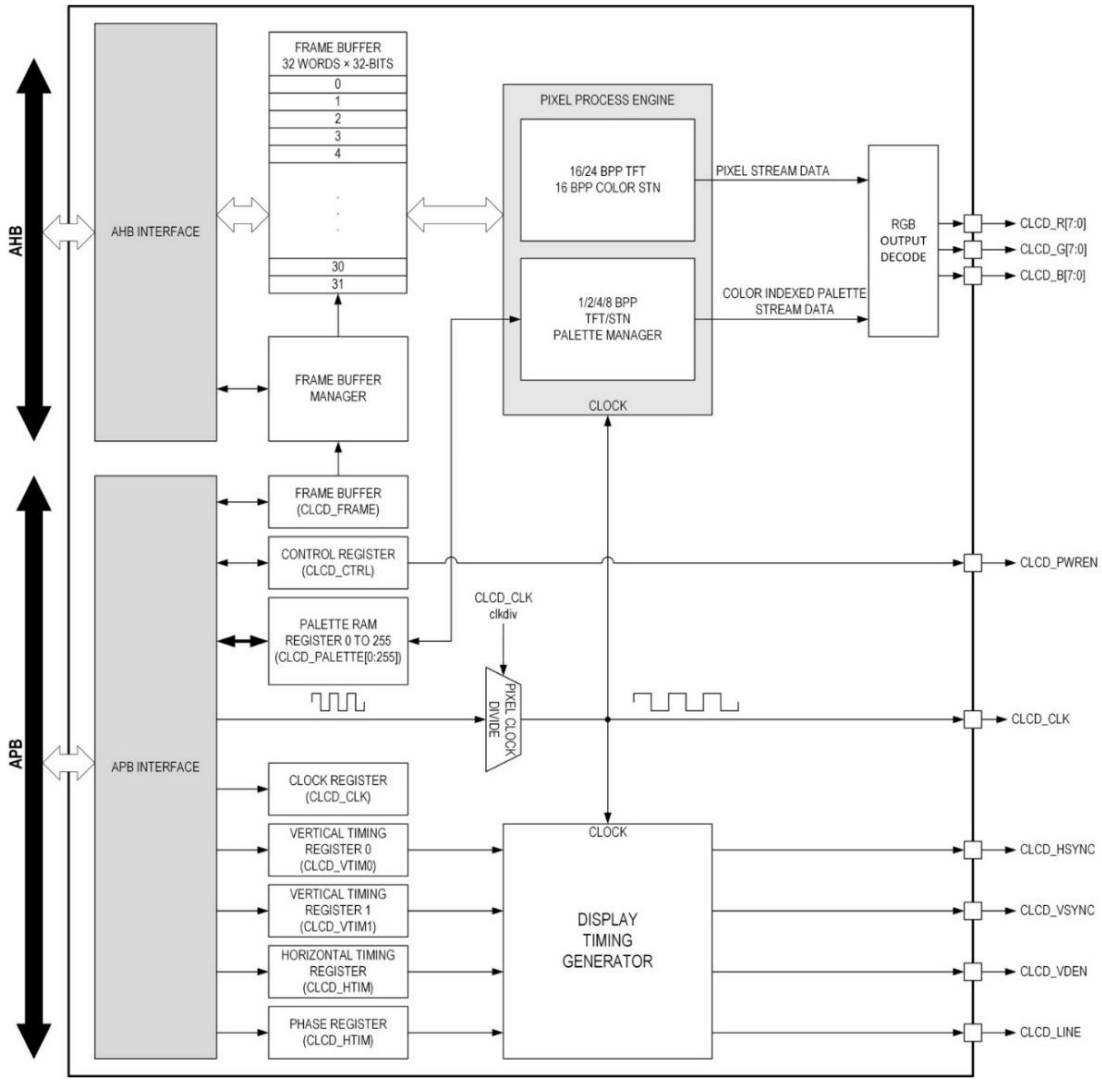
*TFT Displays with 16/24 bpp*

- Packets of pixel coded data are loaded by the CLCD's AMBA AHB master interface and stored in an internal 32-bit wide DMA FIFO.
- The Pixel Process Engine directly streams the DMA FIFO data as a pixel data stream to the panel interface pins.

*Color STN panels and TFT Displays with 1/2/4/8 bpp*

- The Pixel Process Engine uses the DMA FIFO data as an index into a color palette organized as 256 registers representing a single set of RGB color data.
- The indexed color pallet entry is sent to the panel interface pins to drive the indexed color to the panel interface pins.

*Figure 11-1: Color LCD Block Diagram*




## 11.2.1 *AHB Master Interface and DMA FIFO Operation*

The AMBA master interface performs following functions

### *11.2.1.1 Frame Buffer Memory Addressing*

The CLCD's AHB master interface burst copies frame data from external memory to the CLCD Frame FIFO (32 words x 32-bits) and increments the frame buffer address pointer to the next frame data address memory location.

When the CLCD Frame FIFO falls below the burst threshold (*CLCD_CTRL*.*burst*), the CLCD's AHB master state machine burst fills the CLCD Frame FIFO and increments the frame address value to the next frame data address memory location.

### *11.2.1.2 FIFO full, empty conditions, write address generation*

The AMBA AHB master writes the frame buffer data in bursts of either 4, 8 or 16 words into the internal 32 word FIFO.

The PPE unloads the DMA FIFO using the APB slave interface, freeing the AHB until the FIFO falls below the burst fill level.

If the LCD frame is not complete and the DMA FIFO cannot burst fill the FIFO, a buffer underflow interrupt occurs notifying the application of the error.

#### 11.2.1.3 APB Slave Interface and Registers

The CLCD registers including the color palate registers are mapped to the AMBA APB interface enabling the application code to directly modify the CLCD configuration and color palette.

## 11.3 Signals and Pins

The MAX32650–MAX32652 CLCD pin mapping for the 144-pin TQFP and 96-WLP.

*Table 11-1: CLCD Pins and Signal Description*

| Alternate Function | Alternate Function Number | 144-TQFP Pin Name | Signal Description |
|---|---|---|---|
| CLCD_PWREN | AF2 | P2.3 | Display power output |
| CLCD_CLK | AF2 | P0.23 | CLCD clock output |
| | AF2 | P1.3 | |
| CLCD_HSYNC | AF2 | P0.24 | Horizontal Synchronization |
| | AF2 | P1.20 | |
| CLCD_VSYNC | AF2 | P2.6 | Vertical Synchronization |
| CLCD_VDEN | AF2 | P0.22 | Data Enable |
| CLCD_LEND | AF2 | P2.2 | Line End |
| CLCD_R0 | AF2 | P1.30 | Red data bit 0 |
| CLCD_R1 | AF2 | P1.31 | Red data bit 1 |
| CLCD_R2 | AF2 | P2.13 | Red data bit 2 |
| CLCD_R3 | AF2 | P2.14 | Red data bit 3 |
| CLCD_R4 | AF2 | P2.15 | Red data bit 4 |
| CLCD_R5 | AF2 | P2.16 | Red data bit 5 |
| CLCD_R6 | AF2 | P2.17 | Red data bit 6 |
| CLCD_R7 | AF2 | P2.18 | Red data bit 7 |
| CLCD_G0 | AF2 | P0.13 | Green data bit 0 |
| CLCD_G1 | AF2 | P1.14 | Green data bit 1 |
| CLCD_G2 | AF2 | P2.15 | Green data bit 2 |
| CLCD_G3 | AF2 | P2.16 | Green data bit 3 |
| CLCD_G4 | AF2 | P2.17 | Green data bit 4 |
| CLCD_G5 | AF2 | P2.18 | Green data bit 5 |
| CLCD_G6 | AF2 | P2.19 | Green data bit 6 |
| CLCD_G7 | AF2 | P2.20 | Green data bit 7 |
| CLCD_B0 | AF2 | P0.30 | Blue data bit 0 |
| CLCD_B1 | AF2 | P1.23 | Blue data bit 1 |
| CLCD_B2 | AF2 | P1.24 | Blue data bit 2 |
| CLCD_B3 | AF2 | P1.25 | Blue data bit 3 |
| CLCD_B4 | AF2 | P1.26 | Blue data bit 4 |
| CLCD_B5 | AF2 | P1.27 | Blue data bit 5 |
| CLCD_B6 | AF2 | P1.28 | Blue data bit 6 |
| CLCD_B7 | AF2 | P1.29 | Blue data bit 7 |

Perform the following steps to configure the GPIO for CLCD peripheral usage:

*144-TQFP Package*

1. Determine which of the GPIO port pins to use for the CLCD_CLK (P0.23 or P1.3) output pin and perform the following:
   a. Enable the alternate function for the CLCD_CLK signal output by setting either GPIO0[23] to 0 or GPIO1[3] to 0.
   b. Enable Alternate Function 2 for the same output pin by setting GPIO0_AF_SEL[23] to 1 or GPIO1_AF_SEL[3] to 1.
2. Determine which of the GPIO port pins to use for the CLCD_HSYNC (P0.24 or P1.20) output pin and perform the following:
   a. Enable the alternate function for the HSYNC signal output by setting either GPIO0[24] to 0 or GPIO1[20] to 0.
   b. Enable alternate function 2 for the same output pin by setting either GPIO0_AF_SEL[24] to 1 or GPIO1_AF_SEL[20] to 1.
3. Enable the CLCD_VSYNC alternate function by setting GPIO2[6] to 0 and selecting AF2 by setting GPIO2_AF_SEL[6] to 1.
4. Enable the CLCD_VDEN alternate function by setting GPIO0[22] to 0 and selecting AF2 by setting GPIO0_AF_SEL[22] to 1.
5. Optionally enable the CLCD_LEND alternate function by setting GPIO2[2] to 0 and selecting AF2 by setting GPIO2_AF_SEL[2] to 1.
6. Similarly enable each of the data output pins for the red, green and blue output signals.

*Note: If using a Color STN display, the 8-bit data bus is output on the CLCD_R[7:0] pins and the CLCD_G[7:0] and CLCD_B[7:0] pins are not used.*

## 11.4 Pixel Process Engine

The Pixel Process Engine (PPE) unpacks the 32-bit wide LCD data from frame buffer and streams the data directly to the display panel, the grey scaler or the color palette map. The CLCD controller supports three types of endianness configurable to match the external display as shown below:

- Little Endian Byte, Little Endian Pixel (LBLP)
- Big Endian Byte, Big Endian Pixel (BBBP)
- Little Endian Byte, Big Endian Pixel (LBBP)

*Table 11-2* shows the data mapping for LBLP mode, *Table 11-3* shows the data mapping for BBBP and

*Table 11-4* shows the data mapping for LBBP. The CLCD Frame Buffer registers point to the frame data to map to the display. The data should be constructed by the application code to match the format required by the display.

*Table 11-2: CLCD Data Format Little Endian Byte, Little Endian Pixel (LBLP)*

| | DMA FIFO OUPUT BITS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **bpp** | **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** |
| **1** | p31 | p30 | p29 | p28 | p27 | p26 | p25 | p24 | p23 | p22 | p21 | p20 | p19 | p18 | p17 | p16 | p15 | p14 | p13 | p12 | p11 | p10 | p9 | p8 | p7 | p6 | p5 | p4 | p3 | p2 | p1 | p0 |
| **2** | p15 | | p14 | | p13 | | p12 | | p11 | | p10 | | p9 | | p8 | | p7 | | p6 | | p5 | | p4 | | p3 | | p2 | | p1 | | p0 | |
| | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **4** | p7 | | | | p6 | | | | p5 | | | | p4 | | | | p3 | | | | p2 | | | | p1 | | | | p0 | | | |
| | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |
| **8** | p3 | | | | | | | | p2 | | | | | | | | p1 | | | | | | | | p0 | | | | | | | |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **16** | p1 | | | | | | | | | | | | | | | | p0 | | | | | | | | | | | | | | | |
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **24** | p0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | 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 |
| **compact** | 1⅓-pixel data | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 11-3: CLCD Data Format Big Endian Byte, Big Endian Pixel (BBBP)*

| | DMA FIFO OUPUT BITS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **bpp** | **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** |
| **1** | p0 | p1 | p2 | p3 | p4 | p5 | p6 | p7 | p8 | p9 | p10 | p11 | p12 | p13 | p14 | p15 | p16 | p17 | p18 | p19 | p20 | p21 | p22 | p23 | p24 | p25 | p26 | p27 | p28 | p29 | p30 | p31 |
| **2** | p0 | | p1 | | p2 | | p3 | | p4 | | p5 | | p6 | | p7 | | p8 | | p9 | | p10 | | p11 | | p12 | | p13 | | p14 | | p15 | |
| | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **4** | p0 | | | | p1 | | | | p2 | | | | p3 | | | | p4 | | | | p5 | | | | p6 | | | | p7 | | | |
| | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |
| **8** | p3 | | | | | | | | p2 | | | | | | | | p1 | | | | | | | | p0 | | | | | | | |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **16** | p0 | | | | | | | | | | | | | | | | p1 | | | | | | | | | | | | | | | |
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **24** | p0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | 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 |
| **compact** | 1⅓-pixel data | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 11-4: CLCD Data Format Little Endian Byte, Big Endian Pixel (LBBP)*

| | DMA FIFO OUPUT BITS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **bpp** | **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** |
| **1** | p24 | p25 | p26 | p27 | p28 | p29 | p30 | p31 | p16 | p17 | p18 | p19 | p20 | p21 | p22 | p23 | p8 | p9 | p10 | p11 | p12 | p13 | p14 | p15 | p0 | p1 | p2 | p3 | p4 | p5 | p6 | p7 |
| **2** | p12 | | p13 | | p14 | | p15 | | p8 | | p9 | | p10 | | p11 | | p4 | | p5 | | p6 | | p7 | | p0 | | p1 | | p2 | | p3 | |
| | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **4** | p6 | | | | p7 | | | | p4 | | | | p5 | | | | p2 | | | | p3 | | | | p0 | | | | p1 | | | |
| | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 |
| **8** | p3 | | | | | | | | p2 | | | | | | | | p1 | | | | | | | | p0 | | | | | | | |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **16** | p1 | | | | | | | | | | | | | | | | p0 | | | | | | | | | | | | | | | |
| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **24** | p0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | 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 |
| **compact** | 1⅓-pixel data | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

## 11.5 Palette RAM

The CLCD includes 768 bytes of palette RAM organized as 256 entries of 24-bits each. The palette RAM is 768 bytes in size and structured as 256 x 24 bits. Each of the 256 entries represents one color in the color palette and maps to the *CLCD_PALETTE_RAM[0:255]* registers. *Table 11-5* shows the bit representation of each 24-bit color palette entry in the corresponding CLCD_PALETTE_RAMn register.

*Table 11-5 Palette RAM Data Format for CLCD_PALETTE_RAM[0:255] Registers*

| | **CLCD_PALLETTE_RAMn Bit Positions 31:0** | | | |
|---|---|---|---|---|
| | **31:24** | **23:16** | **15:8** | **7:0** |
| **Color Data** | Unused | Blue | Green | Red |

The palette data feeds the gray scaler in STN mode or the panel data interface mux in TFT mode when bpp is 1, 2, 4, or 8. The palette RAM is not enabled for color resolutions of 16 bpp or 24 bpp for both display types STN, TFT. For the case that bpp is 16 or 24, the palette RAM is bypassed, and the output of the pixel stream engine is used as the panel data for TFT. For the case that bpp is 16 the display type is STN, palette RAM is bypassed, and the output of the pixel stream engine is fed to the gray scale logic. For mono STN mode, only the red palette field bits [4:1] (serving as a 4-bit gray level code) are used. In STN color mode with color resolution 8 or lower bpp the green and blue [4:1] are also used.

## 11.6 8-Bit Color STN Output Format

In TFT mode, the PPE bypasses the palette engine and converts the frame buffer data to a serialized data stream driven out the display port pins (CLCD_R[0:7], CLCD_G[0:7], CLCD_B[0:7]).

In 8-bit Color STN mode the CLCD controller outputs 8-bit multiplexed pixel data each CLCD_CLK cycle. Red, green and blue pixel data bits are shifted into internal registers. When enough data is collected, the PPE constructs a multiplexed 8-bit RGB pixel to output to the STN display. *Table 11-6* shows the 8-bit multiplexed output for a Color STN display. Rn (Gn or Bn) denotes the RED (GREEN or BLUE) data of pixel number n.

*Table 11-6 STN Data Output Format per Clock Cycle*

| | Output Data Bits | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **CLCD_CLK Period** | **p0** | **p1** | **p2** | **p3** | **p4** | **p5** | **p6** | **p7** |
| **1** | R1 | G1 | B1 | R2 | G2 | B2 | R3 | G3 |
| **2** | B3 | R4 | G4 | B4 | R5 | G5 | B5 | R6 |
| **3** | G6 | B6 | R7 | G7 | B7 | R8 | G8 | B8 |
| **…** | … | | | | | | | |

In STN single panel mode, output pins CLCD_R[7:0] are used for the 8-bit interface. In STN dual panel mode, CLCD_R[7:0] are used for the upper panel, and CLCD_G[7:0] are used for the lower panel. *Table 11-7* shows which output pins are used to supply the pixel data for supported operation modes. D[7:0] are the output pins for the 8-bit Color STN display interface.

*Table 11-7 LCD Panel Signals*

| | STN Panel | TFT | | |
|---|---|---|---|---|
| **Output Pins** | **8-bit Color** | **24 bpp** | **1/2/4/8/16 bpp** | **16 bit (5:6:5 mode)** |
| CLCD_R[7] | D [0] | Red[7] | Red[4] | Red[4] |
| CLCD_R[6] | D[1] | Red[6] | Red[3] | Red[3] |
| CLCD_R[5] | D[2] | Red[5] | Red[2] | Red[2] |
| CLCD_R[4] | D[3] | Red[4] | Red[1] | Red[1] |
| CLCD_R[3] | D[4] | Red[3] | Red[0] | Red[0] |
| CLCD_R[2] | D[5] | Red[2] | - | - |
| CLCD_R[1] | D[6] | Red[1] | - | - |
| CLCD_R[0] | D[7] | Red[0] | - | - |
| CLCD_G[7] | - | Green[7] | Green[4] | Green[5] |
| CLCD_G[6] | - | Green[6] | Green[3] | Green[4] |
| CLCD_G[5] | - | Green[5] | Green[2] | Green[3] |
| CLCD_G[4] | - | Green[4] | Green[1] | Green[2] |
| CLCD_G[3] | - | Green[3] | Green[0] | Green[1] |
| CLCD_G[2] | - | Green[2] | - | Green[0] |
| CLCD_G[1] | - | Green[1] | - | - |
| CLCD_G[0] | - | Green[0] | - | - |
| CLCD_B[7] | - | Blue[7] | Blue[4] | Blue[4] |
| CLCD_B[6] | - | Blue[6] | Blue[3] | Blue[3] |
| CLCD_B[5] | - | Blue[5] | Blue[2] | Blue[2] |
| CLCD_B[4] | - | Blue[4] | Blue[1] | Blue[1] |
| CLCD_B[3] | - | Blue[3] | Blue[0] | Blue[0] |
| CLCD_B[2] | - | Blue[2] | Intensity | - |
| CLCD_B[1] | - | Blue[1] | - | - |
| CLCD_B[0] | - | Blue[0] | - | - |

## 11.7 Panel/Pixel Clock Generation

The output of the clock generator block is the panel clock. This is a divided down version of the input APB clock. The clock divide value can be programmed and is stored in the LCD clock control register. Since the color STN displays address 8 segments per pixel or 8/3 pixels are addressed per pixel clock. The pixel clock needs be 1/8 of the system clock and in that case every 3rd system clock pixel data is driven. Likewise, the following division ratios are used for STN 4-bit interfaces. For calculation of the required pixel clock refer to section Pixel/Panel Clock Frequency Calculation.

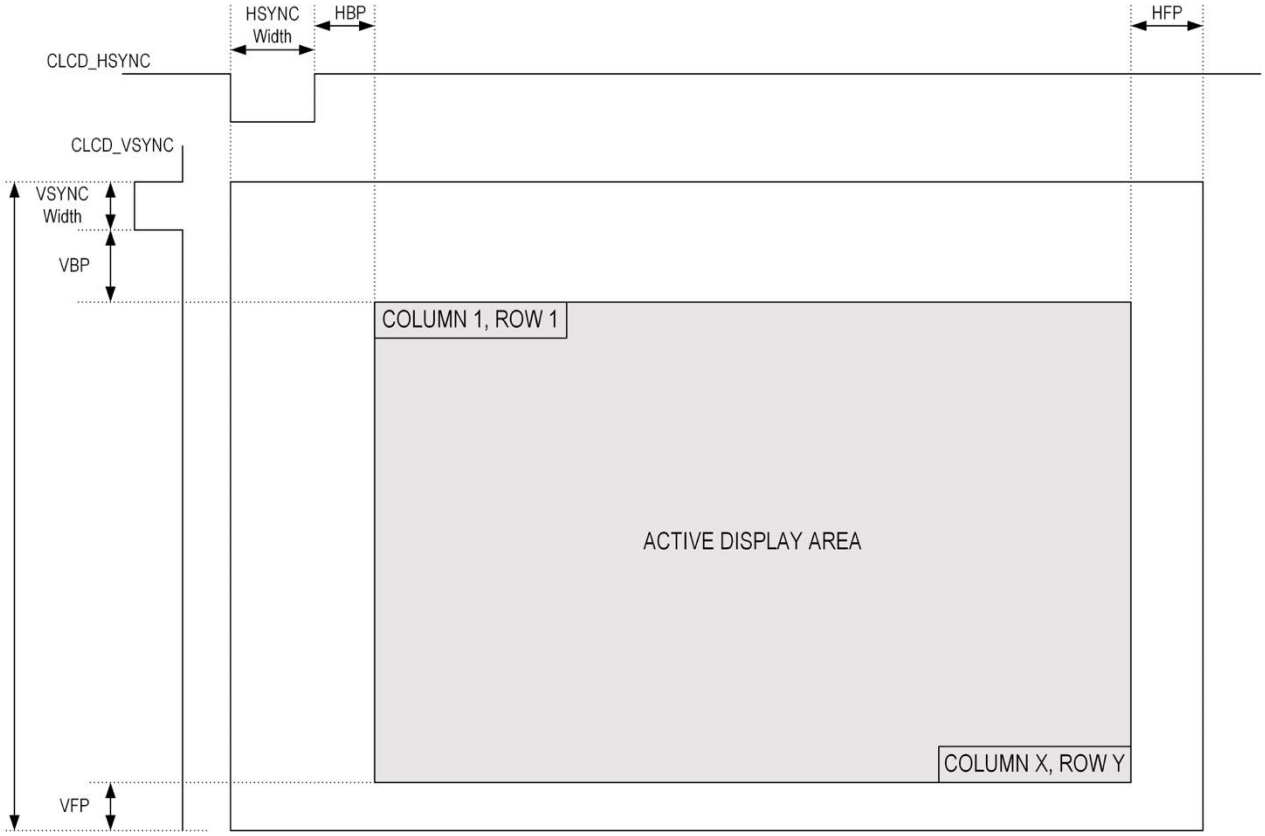
*Table 11-8 PCLK to PIXEL Clock Divide Ratios*

| Display Type | Pixels per Clock | Divide Ratio |
|---|---|---|
| TFT | 1 | 1(min) |
| STN 8-bit color interface | 2⅔ | 8 |
| STN 4-bit color interface | 1⅓ | 4 |

All the logic of the color LCD controller is synchronized with PCLK, except for the AMBA AHB interface, which is synchronized with HCLK.

## 11.8 LCD Panel Timing Generation

The primary function of the LCD timing block is to control the frame buffer reading sequence, synchronize the PPE with the output timing requirement, and generate the horizontal and vertical timing panel signals (HSYNC, VSYNC, LEND, and VDEN).



## 11.9 Interrupt Operation

The LCD controller provides four individually mask able interrupts:

- DMA FIFO underflow: an underflow interrupt is asserted if an attempt is made to read the frame buffer FIFO when it is empty
- Address ready signification: an interrupt is set when the current base address registers have been loaded to the AMBA AHB master state machine, signifying that a new next address can be loaded to the registers.
- Vertical status: an interrupt is asserted when the specified vertical state is reached. The vertical state is selected via the LCD control register.
- Bus error: an interrupt is asserted if an error occurs during an AHB data transfer

## 11.10 TFT Controller Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the CLCD Base Peripheral Address.

Address assignments for the LCD Interface registers are outlined in *Table 11-9*. Reserved register bits should only be written as 0.

*Table 11-9 LCD Interface Register Offsets, Names and Descriptions*

| Offset | Register Name | Description |
|---|---|---|
| [0x0000] | *CLCD_CLK_CTRL* | CLCD Clock Register |
| [0x0004] | *CLCD_VTIM_0* | CLCD Vertical Timing 0 Register |
| [0x0008] | *CLCD_VTIM_1* | CLCD Vertical Timing 1 Register |
| [0x000C] | *CLCD_HTIM* | CLCD Horizontal Timing Register |
| [0x0010] | *CLCD_CTRL* | CLCD Control Register |
| [0x0018] | *CLCD_FRBUF* | CLCD Frame Buffer 0 Register |
| [0x0020] | *CLCD_INT_EN* | CLCD Interrupt Mask Register |
| [0x0024] | *CLCD_INT_STAT* | CLCD Status Register |
| [0x0400] - [0x07FC] | *CLCD_PALETTE_RAM[0:255]* | CLCD Palette RAM Registers 0 to 255 (CLCD_PALETTE_RAM[0:255] |

## 11.11 TFT Controller Register Details

*Table 11-10: CLCD Clock Register*

| **CLCD Clock Register** | | | **CLCD_CLK_CTRL** | **[0x0000]** |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:21 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 20 | clk_active | R/W | 0 | **STN Clock Active Select**<br>If the display type is Color STN 8-bit, refer to *CLCD_CTRL.disptype*, this bit selects if the CLCD_CLK output is active always or only during data output to the display. Refer to the Color STN display data sheet to determine the specific display's requirement.<br>0: CLCD_CLK output is always active.<br>1: CLCD_CLK output is only active during data output.<br>*Note: Always set this bit to 0 for TFT displays.* |

| CLCD Clock Register | | | CLCD_CLK_CTRL | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 19 | clk_edge_sel | R/W | 0 | **Clock Edge Selection**<br>This field controls the clock edge that is used by the LCD panel to sample the data and signal lines. When set to 1, the CLCD_R[7:0], CLCD_G[7:0], and CLCD_B[7:0] data is valid on the falling edge of the CLCD_CLK. When set to 0, the data on the same pins is valid on the rising edge of the CLCD_CLK signal.<br>0: Data valid on rising edge of the clock output signal (CLCD_CLK).<br>1: Data valid on falling edge of the clock output signal (CLCD_CLK). |
| 18 | hsync_pol | R/W | 0 | **CLCD_HSYNC Polarity Selection**<br>This field sets the polarity of the hortizal sync signal output pin (CLCD_HSYNC). Setting this field to 0 sets CLCD_HSYNC active low and setting it to 1 results in an active high signal output.<br>0: CLCD_HSYNC output is active low.<br>1: CLCD_HSYNC output is active high. |
| 17 | vsync_pol | R/W | 0 | **CLCD_VSYNC Polarity Selection**<br>This field sets the polarity of the vertical sync signal output pin (CLCD_VSYNC). Setting this field to 0 sets CLCD_VSYNC active low and setting it to 1 results in an active high signal output.<br>0: CLCD_VSYNC output is active low.<br>1: CLCD_VSYNC output is active high. |
| 16 | vden_pol | R/W | 0 | **CLCD_VDEN Polarity Selection**<br>This field sets the polarity of the video enable signal output pin (CLCD_VDEN). Setting this field to 0 sets CLCD_VDEN active low and setting it to 1 results in an active high signal output.<br>0: CLCD_VDEN output is active low.<br>1: CLCD_VDEN output is active high. |
| 15:8 | stn_ac_bias | R/W | 0 | **AC Bias Frequency Control**<br>This field sets the AC Bias Frequency output on the CLCD_VDEN pin for Color STN display mode. Set the value of this field to the required number of line clocks (CLCD_VSYNC pulses) before the AC Bias output (CLCD_VDEN pin) changes state. The AC Bias is always the value set in this field plus 1.<br>0: 1 line<br>1: 2 lines<br>2: 3 lines<br>…<br>254: 255 lines<br>255: 256 lines<br>*Note: This field is ignored if the display type is set to TFT (CLCD_CTRL.disptype = 0x8)* |
| 7:0 | lcd_clkdiv | R/W | 0 | **CLCD Clock Divisor**<br>This field sets the CLCD clock and the CLCD_CLK output pin clock divisor. The CLCD clock frequency is $f_{LCD\_CLK} = \frac{f_{PCLK}}{(lcd\_clkdiv+1)}$. |

*Table 11-11: CLCD Vertical Timing Register 0*

| CLCD Vertical Timing Register 0 | | | CLCD_VTIM_0 | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| CLCD Vertical Timing Register 0 | | | CLCD_VTIM_0 | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 23:16 | vbp_width | R/W | 0 | **Vertical Back Porch (VBP) Control**<br>This field sets the number of lines for the VBP from 0 lines to 255 lines.<br>0: 0 lines<br>1: 1 lines<br>2: 2 lines<br>…<br>254: 254 lines<br>255: 255 lines |
| 15:12 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11:0 | vlines | R/W | 0 | **Number of Vertical Lines**<br>This field sets the number of vertical lines for the display from 1 to 4096 lines.<br>0: 1 line<br>1: 2 lines<br>2: 3 lines<br>4: 4 lines<br>…<br>4095: 4096 lines |

*Table 11-12: CLCD Vertical Timing Register 1*

| CLCD Vertical Timing Register 1 | | | CLCD_VTIM_1 | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 23:16 | vfp_width | R/W | 0 | **Vertical Front Porch (VFP) Control**<br>This field sets the number of lines for the VFP from 0 lines to 255 lines.<br>0: 0 lines<br>1: 1 lines<br>2: 2 lines<br>…<br>254: 254 lines<br>255: 255 lines |
| 15:8 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7:0 | vsync_width | R/W | 0 | **VSYNC Width Control**<br>This field sets the width of the VSYNC signal output from 1 to 256 lines.<br>0: 1 line<br>1: 2 lines<br>2: 3 lines<br>4: 4 lines<br>…<br>255: 255 lines |

*Table 11-13: CLCD Horizontal Timing Register*

| CLCD Horizontal Timing Register | | | CLCD_HTIM | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | hbp_width | R/W | 0 | **Horizontal Back Porch (HBP) Width**<br>This field sets the number of lines for the HBP from 1 CLCD_CLK to 256 CLCD_CLKs.<br>0: 1 CLCD_CLK<br>1: 2 CLCD_CLKs<br>2: 3 CLCD_CLKs<br>…<br>254: 255 CLCD_CLKs<br>255: 256 CLCD_CLKs<br>*Note: The following equation must be met to allow the minimum propagation time from the frame buffer memory to the CLCD output pins.*<br>$hpb\_width \geq \left(\frac{4}{CLCD\_CLK\_CTRL.clkdiv + 1}\right) - 1$ |
| 23:16 | hsize_index | R/W | 0 | **Horizontal Front Porch (HFP) Width Control**<br>This field sets the horizontal size of the display from 16 pixels to 4096 pixels using the equation: size $= (\text{hsize} + 1) * 16$.<br>0: 16 pixels<br>1: 32 pixels<br>2: 48 pixels<br>3: 64 pixels<br>4: 128 pixels<br>…<br>254: 4080 pixels<br>255: 4096 pixels |
| 15:8 | hfp_width | R/W | 0 | **Horizontal Front Porch (HFP) Width**<br>This field sets the number of lines for the HFP from 0 CLCD_CLKs to 255 CLCD_CLKs.<br>0: 0 lines<br>1: 1 lines<br>2: 2 lines<br>…<br>254: 254 lines<br>255: 255 lines |
| 7:0 | hsync_width | R/W | 0 | **Horizontal Sync (HSYNC) Width Control**<br>This field sets the width of the HSYNC signal output from 1 CLCD_CLK to 256 CLCD_CLKs.<br>0: 1 CLCD_CLK<br>1: 2 CLCD_CLKs<br>2: 3 CLCD_CLKs<br>4: 4 CLCD_CLKs<br>…<br>255: 255 lines |

*Table 11-14: CLCD Control Register*

| CLCD Control Register | | | CLCD_CTRL | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:23 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 22 | pwr_enable | R/W | 0 | **Display Power Enable**<br>Enables power to the display using the CLCD_PWREN output pin.<br>0: Power enable pin (CLCD_PWREN) is set low<br>1: Power enable pin (CLCD_PWREN) is set high |

| CLCD Control Register | | | CLCD_CTRL | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 21 | lend_pol | R/W | 0 | **CLCD_LEND Polarity Selection**<br>This field sets the polarity of the line end signal output pin (CLCD_LEND). Setting this field to 0 sets CLCD_LEND active low and setting it to 1 results in an active high signal output.<br>0: CLCD_LEND output is active low.<br>1: CLCD_LEND output is active high. |
| 20:19 | burst_size | R/W | 0 | **FIFO Burst and Threshold**<br>Set this field to the required burst and threshold level for the Frame Buffer load from the AHB.<br>0: 4 32-bit words<br>1: 8 32-bit words<br>2: 16 32-bit words<br>3: 16 32-bit words |
| 18:16 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | compact_24b | R/W | 0 | **TFT Compact 24-bit Mode**<br>This field selects compact 24-bit mode if the display type is set to TFT and 24bpp. When this field is set to 1, each word in the Frame Buffer holds 1⅓ pixels of data, using all 32-bits in the Frame Buffer entry. When this field is 0, each frame buffer entry contains 24-bits of data in the lower 24-bits of the word and the upper 8-bits are unused.<br>0: 1 pixel per frame buffer entry<br>1: 1⅓ pixels per frame buffer entry |
| 14 | – | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 13:12 | endian | R/W | 0 | **Endian Mode for Word and Pixel**<br>This field selects the endian mode for bytes and words of CLCD frame data.<br>0: Little endian Byte, Littel endian Pixel (LBLP)<br>1: Big endian Byte, Big endian Pixel (BBBP)<br>2: Little endian Byte, Big endian Pixel (LBBP)<br>3: Reserved for Future Use |
| 11 | mode565 | R/W | 0 | **Mode 565 Select**<br>This field selects the mode for either RGB565 or BGR556 for the frame data.<br>0: BGR556<br>1: RGB565 |
| 10 | bpp | R/W | 0 | **Bits per Pixel Select**<br>This field sets the bits per pixel (bpp) for the CLCD display.<br>0: 1 bpp<br>1: 2 bpp<br>2: 4 bpp<br>3: 8 bpp<br>4: 16 bpp<br>5: 24 bpp<br>6: Reserved for Future Use<br>7: Reserved for Future Use |
| 7:4 | disptype | R/W | 0 | **Display Type Selection**<br>This field selects between TFT or 8-bit Color STN display types. Application software must set this field after any form of reset.<br>4: 8-bit Color STN<br>8: TFT<br>All other values are Reserved for Future Use |
| 3 | – | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| CLCD Control Register | | | CLCD_CTRL | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 2:1 | vci_sel | R/W | 0 | **Vertical Compare Interrupt (VCI) Source Select**<br>This field allows the selection of the interrupt source for Vertical Compare interrupts.<br>0: VCI on start of VSYNC<br>1: VCI on start of VBP<br>2: VCI on start of VDEN (active video)<br>3: VCI on start of VFP |
| 0 | lcd_enable | R/W | 0 | **CLCD Enable**<br>This field enables the CLCD to begin displaying the frame data using the CLCD output pins. After setting this bit to 1, the CLCD controller clears the *CLCD_INT_STAT.lcd_idle* flag indicating it is actively driving the frame data to the display.<br>0: Set to 0 to disable the CLCD active driving of an external display.<br>1: Set to 1 to start the CLCD actively driving the configured display.<br>*Note: Ensure that CLCD_INT_STAT.lcd_idle = 1 prior to setting the lcd_enable bit to avoid corrupting any active frame.* |

*Table 11-15: CLCD Frame Buffer Register*

| CLCD Frame Buffer Register | | | CLCD_FRBUF | [0x001C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | frame_addr | R/W | 0 | **Frame Buffer Start Address**<br>Set this field to the beginning of the frame buffer data to display. The frame management begins loading the internal frame buffer FIFO starting at this address. When the frame manager is complete, and this register is available to the application again, the *CLCD_INT_STAT.addr_ready* flag is set to 1 by hardware. The application should not modify this register until the hardware has set the flag.<br>*Note: Frame Buffer data must be word aligned (bits 0 and 1 are always 0).* |

*Table 11-16: CLCD Interrupt Enable Register*

| CLCD Interrupt Enable Register | | | CLCD_INT_EN | [0x0020] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31: 4 | – | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3 | bus_error_ie | R/W | 0 | **Bus Error Interrupt Enable**<br>Set this bit to 1 to enable IRQ events for AHB errors.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 2 | vci_ie | R/W | 0 | **Vertical Compare Interrupt Enable**<br>Set this bit to 1 to enable IRQ events for the event selected by the *CLCD_CTRL.vci_sel* field.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 1 | addr_rdy_ie | R/W | 0 | **Frame Buffer Register Free Interrupt Enable**<br>Set this field to enable IRQ events for the frame buffer register free events.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 0 | underflow_ie | R/W | 0 | **Frame Buffer FIFO Underflow Interrupt Enable**<br>Set this field to 1 to enable the underflow interrupt.<br>0: Interrupt disabled<br>1: Interrupt enabled |

*Table 11-17: CLCD Interrupt Status Register*

| CLCD Interrupt Status Register | | | CLCD_INT_STAT | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:9 | – | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 8 | clcd_idle | RO | 1 | **CLCD Idle Flag**<br>After the *CLCD_CTRL.lcd_enable* bit is set to 1 by the application, this bit is cleared by hardware to indicate the CLCD controller is sending the frame data to the display. The CLCD controller hardware clears this bit when the frame is complete, or an error condition occurs.<br>0: CLCD controller is displaying the current frame.<br>1: CLCD controller is idle. |
| 3 | bus_error_ie | R/W | 0 | **Bus Error Interrupt Enable**<br>Set this bit to 1 to enable IRQ events for AHB errors.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 2 | vci_ie | R/W | 0 | **Vertical Compare Interrupt Enable**<br>Set this bit to 1 to enable IRQ events for the event selected by the *CLCD_CTRL.vci_sel* field.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 1 | addr_rdy_ie | R/W | 0 | **Frame Buffer Register Free Interrupt Enable**<br>Set this field to enable IRQ events for the frame buffer register free events.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 0 | underflow_ie | R/W | 0 | **Frame Buffer FIFO Underflow Interrupt Enable**<br>Set this field to 1 to enable the underflow interrupt.<br>0: Interrupt disabled<br>1: Interrupt enabled |

*Table 11-18: CLCD Palette RAM Registers 0 to 255*

| CLCD Palette RAM Registers 0 – 255 | | | CLCD_PALETTE_RAM[0:255] | [0x0400:0x07FC] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31: 24 | – | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 23:16 | blue | R/W | 0 | **Blue Data for Pallet Entry**<br>Write this field with 8 bits of blue data for the specific CLCD_PALETTE_RAM[n] register. Each CLCD_PALETTE_RAM[n] register holds 1 RGB color representation. |
| 15:8 | green | R/W | 0 | **Green Data for Pallet Entry**<br>Write this field with 8 bits of green data for the specific CLCD_PALETTE_RAM[n] register. Each CLCD_PALETTE_RAM[n] register holds 1 RGB color representation. |
| 7:0 | red | R/W | 0 | **Red Data for Pallet Entry**<br>Write this field with 8 bits of green data for the specific CLCD_PALETTE_RAM[n] register. Each CLCD_PALETTE_RAM[n] register holds 1 RGB color representation. |

# 12 UART

The MAX32650–MAX32652 microcontroller provides three industry-standard UART ports which can communicate with external devices using standard serial communications protocols. The UARTs are full-duplex Universal Asynchronous Receiver/Transmitter (UART) serial ports. Each UART instance, UART0, UART1, and UART2, supports identical functionality and registers unless specifically noted otherwise. For simplicity, the UARTs are referenced in the documentation as UARTn where n = 0, 1, or 2 The registers for each UART are documented showing an offset address, which is identical for each UART instance. Access a specific UART's control register using the UART's control register offset adding it to the specific UART's base peripheral address. For example, to access the Interrupt Flag Register for UART 2, use UART2's base peripheral address of 0x4004 4000 and add the offset of [0x0010] for the Interrupt Flag Register, resulting in UART2_INT_FL register address of 0x4004 4010.

*Features:*

- Flexible baud rate generation up to 4 Mbps with ±2% accuracy
- 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 32-bytes deep transmit and receive FIFOs
- Flexible interrupt conditions
- Hardware flow control for RTS and CTS
- Null modem support
- Break generation and detection
- Wakeup from DEEPSLEEP on UART edge with no character loss
- RX Timeout detection

## 12.1 UART Frame Characters

Character sizes of 5 to 8 bits are supported. The field *UARTn_CTRL0.charsize* is used to select the character size.

Stop bit support includes 1, 1.5, and 2 stop bits selected with the register field *UARTn_CTRL0*.

Parity support includes even, odd, mark, space or none. For no parity, set field *UARTn_CTRL0.parity_en* to 0. For all other parity options, select one of the four parity options using the *UARTn_CTRL0.parity_mode* field and enable parity (*UARTn_CTRL0.parity_en*=1). Parity can be based on the number of 1 bits or 0 bits in the receive characters as set in the register bit *UARTn_CTRL0.parity_lvl*.

Break frames are transmitted by setting the field *UARTn_CTRL0.break* to 1. A break sets all bits in the frame to 0.

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

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

## 12.2 UART Interrupts

Interrupts can be generated for the following conditions:

- Tshe Transmit FIFO level is less than or equal to the set transmit threshold.
- The Receive FIFO level is greater than or equal to the set receive threshold.
- The Receive FIFO is overrun, which means the Receive FIFO is full but is still receiving data
- Any CTS state change. During Hardware Flow Control, this interrupt is generated either because:
  - CTS is deasserted, which tells the UART to pause transmitting data
  - CTS is asserted, which tells the UART to resume transmitting data
- A Receive Parity Error occurred
- A Receive Frame Error occurred, which means START or STOP bits were not detected
- A Receive Timeout condition occurred, which means the RX FIFO has not received a character for a set time
- First and Last BREAK characters

## 12.3 Alternate Bit Rate Clock Source

The MAX32650 includes a dedicated 7.3728MHz clock generator, which can be used for the UART bit rate clock generator if the selected System Clock does not meet the bit rate requirements of the application. In practice, the 7.3728MHz clock is ideal for use during low power mode where the Peripheral Clock is turned off for power conservation. The 7.3728MHz clock can be enabled during low power modes enabling the microcontroller to send and receive data while in low power mode.

The UART always uses the Peripheral Clock for register access and logic operation.

The UART bit rate clock is set using the *UARTn_CTRL0.clksel* bit. The UART defaults to PCLK for the bit rate generator clock source. Setting *UARTn_CTRL0.clksel* to 1 selects the 7.3728MHz clock for the bit rate clock source.

## 12.4 UART Bit Rate Calculation

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

- *UARTn_BAUD0.clk_div*: Selects the bit rate clock divisor.
- *UARTn_BAUD0.ibaud*: Sets the integer portion of the bit rate divisor.
- *UARTn_BAUD1.dbaud*: Sets the decimal portion of the bit rate divisor.

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

*Equation 12-1: UART Bit Rate Divisor Equation*

$$DIV = \frac{f_{UART\_BIT\_RATE\_CLK}}{(Clock\ Divider \times Target\ Bit\ Rate)}$$

*Note:* UARTn_BAUD0.clkdiv *should be set to the highest value that results in* $\lfloor DIV \rfloor \geq 1$ *to achieve the highest accuracy for the target bit rate.*

*Equation 12-2: Bit Rate Integer Calculation*

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

*Equation 12-3: Bit Rate Remainder Calculation*

$$UARTn\_BAUD.dbaud = (DIV - UARTn\_BAUD.ibaud) \times 128$$

Example Baud Rate Calculation:

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

$$f_{BIT\_RATE\_CLK} = f_{PCLK} = 60\ MHz$$

$$DIV = \frac{60{,}000{,}000}{(Clock\ Divider\ \times 1{,}843{,}200)}, where\ Clock\ Divider = 2^{(7-clkdiv)}$$

*Table 12-1: Example Baud Rate Calculation Results, Target Bit Rate = 1.8Mbps*

| UARTn_BAUD0 clkdiv | Clock Divider | DIV |
|---|---|---|
| 4 | 8 | 4.069 |
| 3 | 16 | 2.035 |
| 2 | 32 | 1.017 |
| 1 | 64 | 0.509 |
| 0 | 128 | .254 |

*Table 12-1*, *above*, shows the resulting DIV for each of the *UARTn_BAUD0.clkdiv* field settings. With the Clock Divider set to 8, 16 or 32, the resulting DIV value is greater than 1. Setting the clock divider to 32 will generate the most accurate target bit rate because it is the largest value that results in DIV ≥ 1. Using 32 for Clock Divider, *UARTn_BAUD0.clkdiv* = 2), *UARTn_BAUD0.ibaud* is 1, which is the integer portion of the 1.017 DIV calculation. The *UARTn_BAUD1.dbaud* field calculation based on *UARTn_BAUD0.clkdiv* = 2, *UARTn_BAUD0.ibaud* = 1 and DIV = 1.017 is:

$$UARTn\_BAUD1.dbaud = (1.107 - 1) \times 2$$

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

- *UARTn_BAUD0.clkdiv* = 2
- *UARTn_BAUD0.ibaud* = 1
- *UARTn_BAUD1.dbaud* = 2

## 12.5 UART DMA Using the TX and RX FIFOs

Each UART has 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*. The RX FIFO DMA channel and TX FIFO DMA channels operate independently, and each can be enabled or disabled individually. Enable the RX FIFO DMA channel by setting *UARTn_DMA.rxdma_en* to 1 and enable the TX FIFO DMA channel by setting the *UARTn_DMA.txdma_en* to 1. DMA transfers are automatically triggered based on the number of bytes in the RX or TX FIFO as described in the following two sections.

### 12.5.1 *RX FIFO DMA Operation*

*UARTn_DMA.rxdma_lvl* configures the number of entries in the RX FIFO that triggers a DMA transfer from the RX FIFO to system RAM. If the number of entries in the RX FIFO is more than the configured value, a DMA transfer is triggered from the RX FIFO to system RAM. If *UARTn_DMA.rxdma_lvl* = 0 then a transfer is triggered when there is one byte in the FIFO.

*Note: The RX DMA level must be set to a value less than 32 to avoid an RX FIFO overrun condition that results in loss of received data.*

### 12.5.2 TX FIFO DMA Operation

*UARTn_DMA.txdma_lvl* sets the number of entries (level) in the TX FIFO that will trigger a DMA transfer from system RAM to the TX FIFO. If the number of entries (level) in the TX FIFO falls below this value a TX DMA transfer is automatically triggered from System RAM to the TX FIFO.

*Note: Set the TX DMA level (UARTn_DMA.txdma_lvl) greater than 1 to avoid stalling the UART transfer.*

## 12.6 Flushing the UART FIFOs

The FIFOs can be flushed independently by setting *UARTn_CTRL0.rxflush* to 1 for the RX FIFO and *UARTn_CTRL0.txflush*. to 1 for the TX FIFO. The TX FIFO and RX FIFO are automatically flushed if the UART is disabled by clearing the *UARTn_CTRL0.enable* field (*UARTn_CTRL0.enable* = 0).

## 12.7 Hardware Flow Control

When hardware flow control is enabled, the CTS (Clear-to-send) and RTS (Request-to-Send) external signals are directly managed by hardware without CPU intervention. RTS and CTS are active when flow control is enabled by setting the register bit *UARTn_CTRL0.flowctl*=1. The polarity of the CTS/RTS signals are configured with register bit *UARTn_CTRL0.flowpol* and can be active low or active high.

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

If this UART is receiving data, and the RX FIFO reaches the level set in the 6-bit register field *UARTn_CTRL1.rts_fifo_lvl*, then the RTS signal of this UART is deasserted, informing the transmitting UART to stop sending data to this UART to prevent data overflow. Transmission resumes when the level of the RX FIFO drops below *UARTn_CTRL1.rts_fifo_lvl*, which automatically asserts RTS.

## 12.8 UART Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the UARTn Base Peripheral Address.

*Table 12-2: UART Register Offsets, Names, Access and Descriptions*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *UARTn_CTRL0* | R/W | UARTn Control 0 Register |
| [0x0004] | *UARTn_CTRL1* | R/W | UARTn Control 1 Register |
| [0x0008] | *UARTn_STAT* | RO | UARTn Status Register |
| [0x000C] | *UARTn_INT_EN* | R/W | UARTn Interrupt Enable Register |
| [0x0010] | *UARTn_INT_FL* | R/W1C | UARTn Interrupt Flag Register |
| [0x0014] | *UARTn_BAUD0* | R/W | UARTn Baud Rate Integer Register |
| [0x0018] | *UARTn_BAUD1* | R/W | UARTn Baud Rate Decimal Register |
| [0x001C] | *UARTn_FIFO* | R/W | UARTn FIFO Read/Write Register |
| [0x0020] | *UARTn_DMA* | R/W | UARTn DMA Configuration Register |
| [0x0024] | *UARTn_TXFIFO* | RO | UARTn TX FIFO Register |

## 12.9 UART Register Details

*Table 12-3: UART Control 0 Register*

| UART Control 0 Register | | | UARTn_CTRL0 | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 23:16 | to_cnt | R/W | 0 | **RX Timeout Frame Count**<br>If the RX FIFO contains data, a RX Timeout condition occurs if the time for the number of frames in this register passes without the FIFO receiving any new data. If a timeout occurs, the hardware sets the receive timeout flag to 1 (*UARTn_INT_FL.rxto* = 1). |
| 15 | clk_sel | R/W | 0 | **Bit Rate Clock Source Select**<br>Selects the bit rate clock, $f_{UART\_BIT\_RATE\_CLK}$<br>0: Peripheral Clock, $f_{UART\_BIT\_RATE\_CLK} = f_{PCLK}$<br>1: 7.3728MHz internal bit rate clock, $f_{UART\_BIT\_RATE\_CLK} = 7.3728MHz$. |
| 14 | break | R/W | 0 | **Transmit BREAK Frame**<br>Set this field to 1 to send a BREAK frame. A BREAK frame transmits a character with all bits set to 0.<br>0: Normal UART operation.<br>1: Transmit BREAK frame. |
| 13 | nullmod | | 0 | **Null Modem Support**<br>0: Normal operation for RTS/CTS and TXD/RXD<br>1: Null Modem Mode: RTS/CTS swapped, TXD/RXD swapped |
| 12 | flowpol | R/W | 0 | **RTS/CTS Polarity**<br>0: RTS/CTS asserted is 0<br>1: RTS/CTS asserted is 1 |
| 11 | flow | R/W | 0 | **Hardware Flow Control Enable**<br>0: Hardware flow control disabled.<br>1: Hardware RTS/CTS flow control enabled. |
| 10 | stop | R/W | 0 | **STOP Bit Mode Select**<br>0: 1 STOP bit.<br>1: 1.5 STOP bits for 5-bit character size or 2 STOP bits for all other character sizes |
| 9:8 | size | R/W | 0 | **Character Size**<br>Set the number of data bits per frame.<br>0: 5 data bits<br>1: 6 data bits<br>2: 7 data bits<br>3: 8 data bits |
| 7 | bitacc | R/W | 0 | **Frame or Bit Accuracy Select**<br>This field selects between either Frame Accuracy or Bit Accuracy for transmitting data.<br>**Frame Accuracy:** Individual frame bit durations may be varied by hardware to meet the target frame period.<br>**Bit accuracy:** Bit width is fixed by hardware. The frame accuracy of data transmitted may be reduced if bit accuracy is prioritized.<br>0: Frame accuracy.<br>1: Bit accuracy.<br>*Note: A frame includes the start, stop, all data bits, and parity bit/bits for the character being transmitted.* |
| 6 | rxflush | R/W1O | 0 | **Receive FIFO Flush**<br>Write 1 to flush the receive FIFO<br>Cleared to 0 by hardware when flush is completed |

| UART Control 0 Register | UARTn_CTRL0 | [0x0000] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 5 | txflush | R/W1O | 0 | **Transmit FIFO Flush**<br>Write 1 to flush the Transmit FIFO<br>Cleared to 0 by hardware when flush is completed |
| 4 | parity_lvl | R/W | 0 | **Parity Level Select**<br>0: Parity is based on number of 0 bits in the character.<br>1: Parity is based on number of 1 bits in the character. |
| 3:2 | parity_mode | R/W | 0 | **Parity Mode Select**<br>0: Even parity<br>1: Odd Parity<br>2: Mark parity<br>3: Space parity |
| 1 | parity_en | R/W | 0 | **Parity Enable**<br>0: No parity<br>1: Parity enabled as charsize+1 bit |
| 0 | enable | R/W | 0 | **UART Enable**<br>0: UART disabled. FIFOs are flushed, bit rate generator is off.<br>1: UART Enabled, bit rate generator is active. |

*Table 12-4: UART Control 1 Register*

| UART Control 1 Register 1 | UARTn_CTRL1 | [0x0004] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:22 | 0 | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 21:16 | rts_fifo_lvl | R/W | 0 | **RTS RX FIFO Threshold Level**<br>When the RX FIFO level is equal to or greater than this value, assert RTS output signal to inform the transmitting UART to stop sending data to this UART. Valid values are 1 to 32. |
| 15:14 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 13:8 | tx_fifo_lvl | R/W | 0 | **TX FIFO Threshold Level**<br>When the TX FIFO level is less than or equal to this value, set *UARTn_INT_FL.tx_fifo_lvl* interrupt flag. Valid values are 1 to 32. Set this field greater than 1 to avoid a stall condition when transmitting UART data. |
| 7:6 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 5:0 | rx_fifo_lvl | R/W | 0 | **RX FIFO Threshold Level**<br>When the RX FIFO level is equal to or greater than this value, the hardware sets the *UARTn_INT_FL.rx_fifo_lvl* interrupt flag is set. Valid values are 1 to 32. Set this field to less than 32 to avoid a RX FIFO overrun condition. |

*Table 12-5: UART Status Register*

| UART Status Register | UARTn_STAT | [0x0008] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:25 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| UART Status Register | UARTn_STAT | [0x0008] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 24 | rx_to | RO | 0 | **RX Timeout**<br>This field is set to 1 when a receive timeout occurs. This field is set by hardware when the condition occurs and is automatically cleared when the condition is no longer valid. |
| 23:22 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 21:16 | tx_num | RO | 0 | **Number of Bytes in the TX FIFO**<br>Read this field to determine the number of bytes in the transmit FIFO. |
| 15:14 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 13:8 | rx_num | RO | 0 | **Number of Byes in RX FIFO**<br>Read this field to determine the number of bytes in the receive FIFO. |
| 7 | tx_full | RO | 0 | **TX FIFO Full Status Flag**<br>This field reads 1 when the TX FIFO is full. This field is set by hardware when the condition occurs and is automatically cleared when the condition is no longer valid.<br>0: TX FIFO is not full.<br>1: TX FIFO is full. |
| 6 | tx_empty | RO | 1 | **TX FIFO Empty Flag**<br>This field reads 1 when the TX FIFO is empty. This field is set by hardware when the condition occurs and is automatically cleared when the condition is no longer valid.<br>0: TX FIFO is not empty, tx_num > 0.<br>1: TX FIFO is empty. |
| 5 | rx_full | RO | 0 | **RX FIFO Full Flag**<br>This field reads 1 when then RX FIFO is full. This field is set by hardware when the condition occurs and is automatically cleared when the condition is no longer valid.<br>0: RX FIFO is not full.<br>1: RX FIFO is full. |
| 4 | rx_empty | RO | 1 | **RX FIFO Empty Flag**<br>This flag reads 1 when the RX FIFO is empty. |
| 3 | break | RO | 0 | **Break Flag**<br>This field is set when a break condition occurs.<br>0: BREAK not received.<br>1: BREAK condition received. |
| 2 | parity | RO | 0 | **Parity Bit State**<br>This field returns the state of the parity bit.<br>0: Parity bit is 0.<br>1: Parity bit is 1. |
| 1 | rx_busy | RO | 0 | **RX Busy**<br>This field reads 1 when the UART is receiving data.<br>0: UART is not actively receiving data.<br>1: UART is actively receiving data. |
| 0 | tx_busy | RO | 0 | **TX Busy**<br>This field reads 1 when the UART is transmitting data.<br>0: UART is not actively transmitting data.<br>1: UART is transmitting data. |

*Table 12-6: UART Interrupt Enable Register*

| UART Interrupt Enable Register | | | UARTn_INT_EN | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:10 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 9 | last_break | R/W | 0 | **Last Break Interrupt Enable**<br>When the UART receives a series of BREAK frames, this enables an interrupt when the last BREAK frame is received. |
| 8 | rx_to | R/W | 0 | **RX Timeout Interrupt Enable**<br>Enable the receive timeout interrupt.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 7 | break | R/W | 0 | **Received BREAK Interrupt Enable**<br>Enables the BREAK interrupt for the first BREAK received on the UART.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 6 | tx_fifo_lvl | R/W | 0 | **TX FIFO Threshold Level Interrupt Enable**<br>Enables the tx_fifo_lvl interrupt when the number of entries in the TX FIFO is equal or less than the value set in *UARTn_CTRL1.tx_fifo_lvl*.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 5 | tx_fifo_ae | R/W | 0 | **TX FIFO One Byte Remaining Interrupt Enable**<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 4 | rx_fifo_lvl | R/W | 0 | **RX FIFO Threshold Level Interrupt Enable**<br>Enables an interrupt when the number of entries in the RX FIFO is greater than or equal to *UARTn_CTRL1.rx_fifo_lvl*.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 3 | rx_overrun | R/W | 0 | **RX FIFO Overrun Interrupt Enable**<br>Enables an interrupt when a write is made to a full RX FIFO.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 2 | cts | R/W | 0 | **CTS State Change Interrupt Enable**<br>Enable the CTS level change interrupt event. This is often referred to as Modem Status Interrupt.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 1 | rx_parity_error | R/W | 0 | **RX Parity Error Interrupt Enable**<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 0 | rx_frame_error | R/W | 0 | **RX Frame Error Interrupt Enable**<br>0: Interrupt disabled<br>1: Interrupt enabled |

*Table 12-7: UART Interrupt Flags Register*

| UART Interrupt Flags Register | | | UARTn_INT_FL | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:10 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| UART Interrupt Flags Register | UARTn_INT_FL | [0x0010] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 9 | last_break | R/W1C | 0 | **Last Break Interrupt Flag**<br>When the UART receives a series of BREAK frames, this flag is set when the last BREAK frame is received. Write 1 to clear this field.<br>0: Last BREAK condition has not occurred.<br>1: Last BREAK condition has occurred. |
| 8 | rx_to | R/W1C | 0 | **Receive Frame Timeout Interrupt Flag**<br>This field is set when a receive frame timeout occurs. Write 1 to clear this field.<br>0: Receive frame timeout has not occurred.<br>1: A receive frame timeout was detected by the UART. |
| 7 | break | R/W1C | 0 | **Received Break Interrupt Flag**<br>When the UART receives a series of BREAK frames, this flag is set when the first BREAK frame is received. Write 1 to clear this field.<br>0: Break condition not occurred<br>1: Break condition occurred. |
| 6 | tx_fifo_lvl | R/W1C | 0 | **Transmit FIFO Threshold Interrupt Flag**<br>Set when number of entries in in the Transmit FIFO is less than or equal to the Transmit FIFO level set in *UARTn_CTRL1.tx_fifo_lvl*.<br>Write 1 to clear.<br>0: The TX FIFO level is below the threshold set in *UARTn_CTRL1.tx_fifo_lvl*.<br>1: The TX FIFO level is equal to or greater than the *UARTn_CTRL1.tx_fifo_lvl*. |
| 5 | tx_fifo_ae | R/W1C | 0 | **Transmit FIFO Almost Empty Interrupt Flag**<br>This field is set when there is one byte remaining in the Transmit FIFO.<br>Write 1 to clear.<br>0: TX FIFO level is greater than 1.<br>1: TX FIFO is Almost Empty. |
| 4 | rx_fifo_lvl | R/W1C | 0 | **RX FIFO Threshold Interrupt Flag**<br>Set when number of entries in the RX FIFO is equal to or greater than the RX FIFO threshold level as set in the *UARTn_CTRL1.rx_fifo_lvl* field. Data must be read from the RX FIFO to reduce the level below the threshold to guarantee this interrupt does not occur again after clearing the flag. Write 1 to clear this field.<br>0: The number of bytes in the RX FIFO is below the threshold level.<br>1: The number of bytes in the RX FIFO is equal to or greater than the threshold level. |
| 3 | rx_ovr | R/W1C | 0 | **RX FIFO Overrun Interrupt Flag**<br>This field is set if the receive FIFO is full and an additional byte is received resulting in a FIFO overrun condition. If this field is set at least one byte of received data has been lost. Write 1 to clear.<br>0: RX FIFO overrun has not occurred.<br>1: RX FIFO overrun occurred. |
| 2 | cts | R/W1C | 0 | **CTS Interrupt Flag**<br>Also called Modem Status Interrupt |
| 1 | parity | R/W1C | 0 | **Receive Parity Error Status Flag**<br>Set if a parity error is detected. This flag applies to data received only. Write 1 to clear.<br>0: Parity error has not been detected.<br>1: Parity error detected. |
| 0 | frame | R/W1C | 0 | **Frame Error Status Flag**<br>Set if a frame error occurs while receiving data. Write 1 to clear.<br>0: Frame error not occurred.<br>1: Frame error occurred. |

*Table 12-8: UART Rate Integer Register*

| UART Baud Rate Integer Register | UARTn_BAUD0 | [0x0014] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:17 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 18:16 | clkdiv | R/W | 0 | **Bit Rate Clock Divisor**<br>This field is used to divide the bit rate clock by the selected Clock Divider value.<br>*Note: Refer to the UART Bit Rate Calculation section for details of determining this field's value for a given UART bit rate.* |
| 15:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11:0 | ibaud | R/W | 0 | **Integer Portion of Baud Rate Divisor**<br>This field contains the integer value of the bit rate divisor. Refer to the *UART Bit Rate Calculation* section for details of determining this field's value for a given UART bit rate. |

| *clkdiv* | Clock Divider Value |
|---|---|
| 0 | 128 |
| 1 | 64 |
| 2 | 32 |
| 3 | 16 |
| 4 | 8 |
| 5, 6, 7 | Reserved for Future Use |

*Table 12-9: UART Baud Rate Decimal Register*

| UART Baud Rate Decimal Register | UARTn_BAUD1 | [0x0018] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11:0 | dbaud | R/W | 0 | **Decimal Portion of Baud Rate Divisor**<br>This field contains the remainder portion of the bit rate divisor. Refer to the *UART Bit Rate Calculation* section for details of determining this field's value for a given UART bit rate. |

*Table 12-10: UART FIFO Register*

| UART FIFO Register | UARTn_FIFO | [0x001C] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:8 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7:0 | fifo | R/W | N/A | **UART FIFO Register**<br>Reading this field reads data from the RX FIFO and writes to this field write to the TX FIFO. |

*Table 12-11: UART DMA Configuration Register*

| UART DMA Configuration Register | | | UARTn_DMA | [0x0020] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:22 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 21:16 | rxdma_lvl | R/W | 0 | **RX FIFO Level DMA Trigger**<br>If the RX FIFO level is greater than this value, the DMA channel transfers data from the RX FIFO into memory. DMA transfers continue until the RX FIFO is empty. To avoid an RX FIFO overrun, do not set this value to 32.<br>Values above 32 are reserved for future use. |
| 15:14 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 13:8 | txdma_lvl | R/W | 0 | **TX FIFO Level DMA Trigger**<br>If the TX FIFO level is less than this value, the DMA channel transfers data from memory into the TX FIFO. DMA transfers continue until the TX FIFO is full. To avoid stalling a UART transmission, do not set this value to 1 or 0.<br>*Note: Values above 32 are Reserved for Future Use.* |
| 7:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | rxdma_en | R/W | 0 | **RX FIFO DMA Channel Enable**<br>0: RX DMA is disabled<br>1: RX DMA is enabled |
| 0 | txdma_en | R/W | 0 | **TX FIFO DMA Channel Enable**<br>0: TX DMA is disabled<br>1: TX DMA is enabled |

*Table 12-12: UART TX FIFO Data Output Register*

| UART TX FIFO Data Output Register | | | UARTn_TXFIFO | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:8 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7:0 | data | RO | 0 | **TX FIFO Data Output Peek Register**<br>Reads from this register return the next character available for transmission at the end of the TX FIFO. If no data is available, reads of this field return 0.<br>Reads from this register do not affect the TX FIFO state.<br>0: No data available in the TX FIFO<br>1-15: Number of bytes in the TX FIFO. |

# 13 I²C Master/Slave Serial Communications Peripheral

The microcontroller integrates two I²C peripherals, designated I2C0_ and I2C1_. The registers for each of the instances are identical with the same offset addresses for each register. For simplicity, I2Cn is used throughout this section to refer to both I²C ports. Each I²C port support both Master and Slave modes. The I²C peripherals support standard-mode and fast-mode I²C standards.

The I²C bus is a standardized two-wire, bidirectional serial bus. It uses only two bus lines, a Serial Data Access (SDA) line for data, and a Serial Clock line (SCL) for the clock. SDA and SCL idle high with external pullup resistors. They are pulled low by open-drain drivers in the peripherals. Internal pullup circuits in the I/O pins can keep SDA and SDL at a logic high state when all devices are idle, but external pullup resistors are highly recommended for all but the simplest, lowest-capacitance systems.

An I²C master owns the I²C bus for the duration of a transfer, which means it drives the SCL pin and generates START and STOP signals. In slave mode, the device relies on an external master to generate the clock on SCL. An I2C slave responds to data and commands only when an I²C master device addresses it.

For detailed information on I²C bus operation, see Maxim Application Note 4024: SPI/I²C Bus Lines Control Multiple Peripherals.

## 13.1 I²C Master/Slave Features

Each I²C Master/Slave is compliant with the I²C Bus Specification with these features:

- I²C bus specification version 2.1 compliant (100kHz and 400kHz)
- Programmable for both Standard Mode (100 kHz) and Fast Mode (400kHz) data rates
- Multi-master capable, including support for arbitration and clock synchronization
- Supports 7- and 10-bit addressing
- Supports RESTART condition
- Supports clock stretching
- Support for 7- and 10-bit device addressing
- Transfer status interrupts and flags
- DMA data transfer support
- I²C timing parameters fully controllable via firmware
- Glitch filter and Schmitt trigger hysteresis on SDA and SCL
- Control, status, and interrupt events are available for maximum flexibility

Each I²C bus port has a FIFO that supports the following features:

- Independent 8 byte receive FIFO and an 8-byte transmit FIFO
- Preloading the TX FIFO
- Programmable threshold TX and RX interrupts

## 13.2 I²C Bus Speeds

The I²C peripherals support two I²C clock frequencies: 100kHz Standard mode and 400kHz Fast Mode. All modes are downward compatible and operate at a lower bus speed as necessary.

## 13.3 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. The data rate is not fixed and can dynamically operate up to 100kHz in Standard Mode and up to 400kHz in Fast Mode.

Each transfer is initiated when the bus master sends a START or repeated START condition followed by the address of the slave peripheral. Information is sent most significant bit (MSB) first. Following the slave address, the master exchanges data with the addressed slave. The master can transmit data to the slave (a 'write' operation) or receive data from the slave (a 'read' operation). 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.

## 13.4 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.

## 13.5 I²C Master/Slave Overview

I²C transmit and receive data transfer operations are initiated by first loading the data to be sent in the I²C FIFO by writing data to the *I2Cn_FIFO* register. Once the transaction has completed, the data received can be read from the FIFO by reading data from the *I2Cn_FIFO* register. If a slave sends a NACK in response to a write operation, the I²C master generates an interrupt to the core. The I2C port 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 stretches the clock to allow time to read bytes from the receive FIFO or load bytes into the transmit FIFO.

## 13.6 Slave Addressing

The first byte transmitted after a START condition is the slave address byte. If seven-bit addressing is used, the address byte consists of seven address bits and one R/W bit.

The I²C implementation used in this device supports both 7-bit and 10-bit addressing. However, some addresses are reserved for special purposes: for example, 0b0000 0000 is a general call address to every slave on the bus, and 0b0000 0001 is a START byte for slower microcontrollers. If the master sends address 0b1111 1xx1, then it is requesting the device ID of a slave. If the address byte starts with 0b1111 0xxx, then the master is initiating 10-bit addressing mode where xxx are the most significant bits of the 10-bit address.

All addresses that start with 0b0000 xxxx or 0b1111 1xxx are reserved by the I²C specification for special purposes and should not be used for slave addresses.

## 13.7 Acknowledge and Not Acknowledge

An acknowledge bit (ACK) is generated by the receiver, whether I2C master or slave, after every byte received. 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 allows SDA to float high during the acknowledge time slot. The I2C master can then either generate a STOP condition to abort the transfer, or it can generate a repeated START condition (that is, send a START condition without an intervening STOP condition) to start a new transfer.

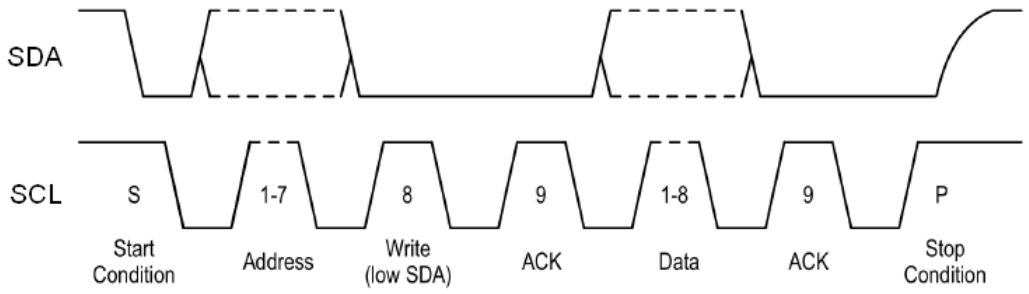
A receiver can generate a NACK after a byte transfer if any of the following conditions occur:

- No receiver is present on the bus with the transmitted address. In that case, no device 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 I²C master has requested data from a slave, it signals the slave to stop transmitting by sending a NACK following the last byte it requires.

## 13.8 Bit Transfer Process

Both SDA and SCL circuits are open-drain, bidirectional circuits. Each has an external pullup resistor that ensures each circuit is high when idle. The I²C specification states that during data transfer, the SDA line can change state only when SCL is low, and that SDA is stable and able to be read when SCL is high as shown in *Figure 13-1*, *below*.

*Figure 13-1: I²C Write Data Transfer*




The process for an I²C data transfer is as follows:

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

## 13.9 SCL and SDA Bus Drivers

The I²C bus expects SCL and SDA to be 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 pull-up resistor. This should only be used in systems where no I²C slave device can hold SCL low. Push-pull operation is enabled by setting *I2Cn_CTRL0.sclppm* to 1. (SDA always operates in open-drain mode.)

### 13.9.1 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 bring set, only from generating an interrupt request.*

It is recommended that 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.

### 13.9.2 SCL Clock Configurations

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

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

*Figure 13-2: I²C Specification Min and Max Clock Parameters*

| Parameter | Standard Mode | | Fast Mode | |
|---|---|---|---|---|
| | Min | Max | Min | Max |
| SCL Clock Freq. | 0 | 100 kHz | 0 | 400 kHz |
| I2C Hold Time | 4.0 µs | - | 0.6 µs | - |
| SCL Hi | 4.0 µs | - | 0.6 µs | - |
| SCL Lo | 4.7 µs | - | 1.3 µs | - |
| TRC Rise Time | - | 1000 ns | 20 ns | 300 ns |

### 13.9.3 Clock Synchronization

The I²C specification allows for more than one bus master. When more than one master is on the same bus, clock synchronization between different master's clocks is necessary. The I²C Masters support automatic clock synchronization

and are compliant with the clock synchronization requirements of the I²C specification. Clock synchronization is automatic, and no additional programming is required.

### 13.9.4 *Transmit and Receive FIFOs*

Each I²C master/slave has one 8-byte deep transmit FIFO (TX FIFO) and one 8-byte deep receive FIFO (RX FIFO) that reduces processor overhead. To further speed transfers, the DMA can read and write to each FIFO. When the DMA is used to read and write to the FIFOs, no additional I²C configuration is required and interrupts are still sent to the core. See the DMA section for more details.

When the receive FIFO is enabled, received bytes are automatically written to it. If the receive FIFO is full, no more data is written and any newly received bytes are lost.

When the transmit FIFO is enabled, either user firmware or the DMA can provide data to be transmitted. The oldest byte in the FIFO is sent out over SDA only when an ACK signal is received from an addressed slave.

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

## 13.10 Clock Stretching

If a slave cannot receive or transmit a complete byte of data, it can force the master into a wait state by clock stretching. Clock stretching is when a slave holds SCL low after an ACK is on the bus. When the slave is ready, it releases the SCL line from low and then resumes the data transfer.

These I²C ports can hold SCL low in both master and slave modes after an ACK bit transmission. However, the term "clock stretching" as defined in the I²C Bus Specification only applies if performed by a slave device. If a master holds the SCL line low, the master is technically varying the clock speed. The master can vary the clock speed from DC (0 Hz) up to the maximum $f_{SCL}$. For simplicity, this section describes situations where either an external slave or external master holds SCL low.

For clock stretching, SCL is held low after an ACK bit and before the first data bit. This is often done when a receiver cannot receive more data (usually from a full RX FIFO), or a transmitter needs to send more data but is not ready (usually from an empty TX FIFO).

However, during Interactive Receive Mode (IRXM), the receiver begins clock stretching before the ACK bit, allowing firmware time to decide whether to send an ACK or NACK. If operating in IRXM (*I2Cn_CTRL0.irxm*=1) as a slave with clock stretching disabled (*I2Cn_CTRL0.sclstrd*=1), SCL is not held low. Thus, the burden is on firmware to respond quickly enough to meet the data setup timing requirements as a late ACK could cause a transition on SDA while SCL is high, resulting in an unwanted STOP or RESTART. For these reasons, it is not recommended to use interactive receive mode with slave clock stretching disabled.

For a transmit operation as either master or slave, when the TX FIFO is empty after the last byte is shifted out, SCL is automatically held low until data is written to the TX FIFO. Master transmitters can stop clock stretching in this situation to end the transaction by sending a START or RESTART condition. When a slave transmitter sees an external master end the transaction by sending a NACK, it can then release SDA.

## 13.11 I²C Bus Timeout

The Timeout register bit field *I2Cn_TIMEOUT.to* is used to detect if a bus error has occurred. The Timeout register configures the timeout value from the following equation:

*Equation 13-1: I²C Timeout Maximum*

$$t_{TIMEOUT} \leq \left(\frac{1}{f_{I2C\_CLK}}\right) \times \left((I2Cn\_TIMEOUT.to \times 8) + 3\right)$$

Because of clock synchronization the timeout is guaranteed to meet the following minimum time:

*Equation 13-2: I²C Timeout Minimum*

$$t_{TIMEOUT} \leq \left(\frac{1}{f_{I2C\_CLK}}\right) \times \left((I2Cn\_TIMEOUT.to \times 8) + 2\right)$$

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 SCL is driven low by this I²C and resets when SCL is released.

The timeout counter only monitors if the I²C port is driving SCL line low. It does not monitor if external I²C device is holding it low. The I2Cn peripheral does not monitor the status of the SDA line.

If the timeout timer expires a bus error condition has occurred and the I2Cn peripheral releases both the SCL and SDA lines and sets the timeout error interrupt flag to 1 (*I2Cn_INT_FL0.toeri* = 1).

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

## 13.12 I²C Addressing

After a START or RESTART condition, an address byte is transmitted where the first seven bits are the address, and the last bit indicates to the slave if the operation is a read or a write.

*Table 13-1: I²C Address Byte 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_MSTR_MODE.sea*= 0, then write the address to the TX FIFO formatted as follows where An is address A6:A0.

Master Writing to Slave : 7-bit address : [A6A5A4A3A2A1A0 0]
Master Reading from Slave : 7-bit address : [A6A5A4A3A2A1A0 1]

In 10-bit addressing mode (*I2Cn_MSTR_MODE.sea* = 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.

If the RX FIFO is not empty and this I²C is asked to receive data, this I²C sends a NACK and so does not participate in the transaction. The setting of the Do Not Respond bit (*I2Cn_RX_CTRL0.dnr*) controls when the NACK is sent: *dnr*=1 sends a NACK on the first address byte and generates an interrupt by setting status flag *I2Cn_INT_FL0.dnreri*; *dnr*=0 sends an ACK on the address bytes but NACKs any following data bytes.

If the TX FIFO is not ready (*I2Cn_TX_CTRL1.txrdy* = 0) and the controller is asked to send data, it sends a NACK during the first address byte. The setting of the Do Not Respond bit does not affect this, since this is the only opportunity to send a NACK for a read transaction.

## 13.13 I²C TX FIFO and RX FIFO Management

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. Reads from an empty RX FIFO returns 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

Both the RX FIFO and TX FIFO are flushed when the I²C port is disabled by clearing *I2Cn_CTRL0.i2cen*=0.

The TX FIFO and RX FIFO can be flushed by setting the Transmit FIFO Flush bit (*I2Cn_TX_CTRL0.txfsh*=1) or the Receive FIFO Flush bit (*I2Cn_RX_CTRL0.rxfsh*=1), respectively. In addition, the TX FIFO is automatically flushed and locked out from SW writes under the following conditions:

- General Call Address match and TX FIFO Preloading is disabled
- Slave Address match and TX FIFO Preloading is disabled
- Operating as a slave transmitter, and a NACK is received.
- Any of the following interrupts: Arbitration Error, Timeout Error, Master Mode Address NACK, Data NACK Error, Start Error, and STOP Condition Detected.

When the above conditions occur, the TX FIFO is flushed so stale data is not unintentionally transmitted. In addition, the Transmit Lockout Flag is set (*I2Cn_INT_FL0.txloi*=1) and writes to the TX FIFO are ignored until firmware acknowledges the external event by clearing *I2Cn_INT_FL0.txloi*.

Flushing the TX FIFO on Slave Address Match or General Call Match can be disabled using the Transmit FIFO Preload bit (*I2Cn_TX_CTRL0.txpreld*). Setting this bit allows applications to preload the Transmit FIFO prior to a Slave Address Match. This can be combined with Slave Clock Stretching disabled (*I2Cn_CTRL0.sclstrd* = 0) to maximize the chance of completing a transmit operation without a transmit underflow error.

## 13.14 Interactive Receive Mode

In some situations, this I²C might want to inspect and respond to each byte of received data. In this case, Interactive Receive Mode can be used. Interactive Receive Mode is enabled by setting *I2Cn_CTRL0.irxm*=1. If Interactive Receive Mode is enabled, it must occur before any I²C transfer is initiated.

When Interactive Receive Mode (IRXM) is enabled, after every data byte received this I²C automatically holds SCL low before the ACK bit, and after the 8th SCL falling edge sets the IRXM Interrupt Status Flag (*I2Cn_INT_FL0.irxmi*=1). Firmware can then read the received data and generate appropriate response based on the active low bit *I2Cn_CTRL0.ack*. If *ack*=1, this I²C acknowledges with a NACK (leaving SDA high). If ack=0, then this I²C acknowledges with an ACK (pulling SDA low).

After deciding on the ACK/NACK response, write a 1 to clear *I2Cn_INT_FL0.irxmi* to 0. This releases SCL and sends an *I2Cn_CTRL0.ack* value onto SDA. For both master and slave operations, SCL is released only after the necessary SCL low time requirement has been satisfied, to conform with tsu;dat timing.

While this I²C is waiting for *I2Cn_INT_FL0.irxmi* to be cleared, 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.

During Interactive Receive Mode, 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*. Otherwise, firmware would read 0xFF from an empty RX FIFO.

*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.*

## 13.15 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.txth*) and RX FIFO (*I2Cn_RX_CTRL0.rxth*) threshold levels.

When the TX FIFO byte count (*I2Cn_TX_CTRL1.txfifo*) is less than or equal to the TX FIFO Threshold Level *I2Cn_TX_CTRL0.txth*, 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:

DMA burst size = 8 - *I2Cn_TX_CTRL0.txth*, where 0 <= *I2Cn_TX_CTRL0.txth* <= 7.

Applications trying to avoid transmit underflow and/or clock stretching should use a smaller burst size and higher txth. This fills up the FIFO more often at the expense of more internal bus traffic.

When the RX FIFO count (*I2Cn_RX_CTRL1.rxfifo*) is greater than or equal to the RX FIFO Threshold Level *I2Cn_RX_CTRL0.rxth*, 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:

DMA burst size = *I2Cn_RX_CTRL0.rxth*, where 1 <= *I2Cn_RX_CTRL0.rxth* <= 8.

Applications trying to avoid receive overflow and/or clock stretching should use a smaller burst size and a lower *I2Cn_RX_CTRL0.rxth*. This results in reading the FIFO more frequently in the application but reduces the internal bus traffic. For I²C receive operations, the length of the DMA transaction (in bytes) must be an integer multiple of *I2Cn_RX_CTRL0.rxth*. 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 the DMA burst size = *I2Cn_RX_CTRL0.rxth* = 1.

To enable DMA transfers, enable the TX DMA channel (*I2Cn_DMA.txen*) and/or the RX DMA channel (*I2Cn_DMA.rxen*). Refer to the DMA chapter for more information on configuring the DMA.

## 13.16 I²C Master Mode Transmit Operation

The peripheral operates in master mode when Master Mode Enable *I2Cn_CTRL0.mst*=1. To initiate a transfer, the master generates a START condition by setting *I2Cn_MSTR_MODE.start*=1. If the bus is busy, it does not generate a START condition until the bus is available.

A master can communicate with two 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_MSTR_MODE.restart*=1. If a transaction is in progress, the master finishes the transaction before generating a RESTART. The controller then transmits the slave address stored in the TX FIFO. The *I2Cn_MSTR_MODE.restart* bit is

automatically cleared to 0 as soon as the master begins a RESTART condition. The reception of a STOP condition clears any pending RESTART.

*I2Cn_MSTR_MODE.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_MSTR_MODE.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_MSTR_MODE.stop* bit is cleared and ignored.

A slave cannot generate START, RESTART, or STOP conditions. Therefore, when Master Mode is disabled, the *I2Cn_MSTR_MODE.start*, *I2Cn_MSTR_MODE.restart*, and *I2Cn_MSTR_MODE.stop* bits are all cleared to 0.

Once a transfer has started by setting *I2Cn_MSTR_MODE.start* =1, settings should not be changed, or unpredictable behavior will occur.

## 13.17 I²C Master Mode Transmit Bus Arbitration

The I²C protocol supports multiple masters on the same bus. When the bus is free, it is possible that two masters might try to initiate communication at the same time. This is a valid bus condition. If this occurs, 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 is called bus arbitration.

To determine which master wins the arbitration, each master compares the data being transmitted on SDA to the value observed on SDA. If the master attempting to transmit a 1 on SDA (that is, the master wants SDA to float) senses a 0 instead, that master concludes that it has lost arbitration because another master is transmitting a 0 onto SDA. It then cedes the bus by switching off its SDA driver.

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, leaving the data on SDA intact.

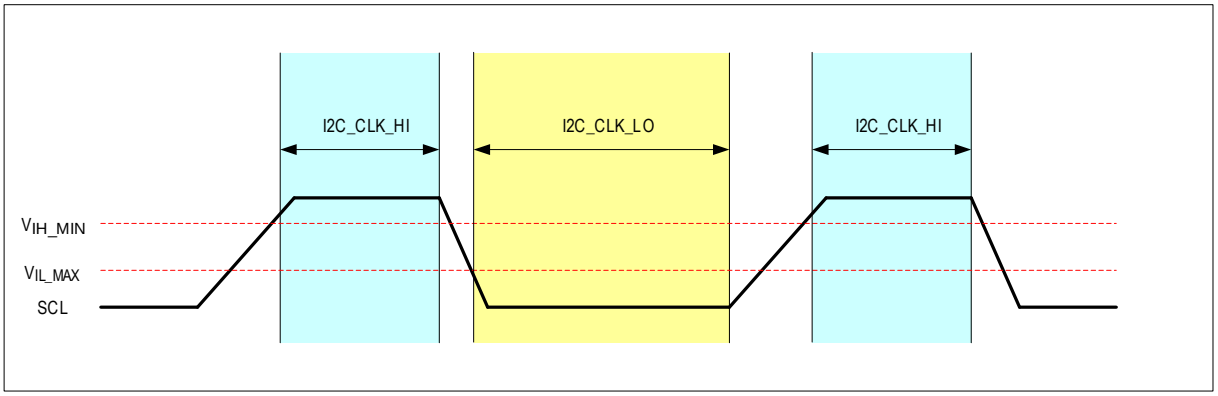
Once a master has lost arbitration it stops generating SCL, sets *I2Cn_INT_FL0.areri*, and clears *I2Cn_MSTR_MODE.start*, *I2Cn_MSTR_MODE.restart*, and *I2Cn_MSTR_MODE.stop* to 0.

The I²C master peripheral is compliant with the bus arbitration requirements of the I²C specification. I²C bus arbitration is automatic, and no additional programming is required.

## 13.18 SCL Clock Generation

The master generates the I²C clock on the SCL line. The I²C peripheral clock, $f_{I2C\_CLK}$ is equal to $f_{PCLK}$

*Figure 13-3: I²C Clock Period*




The SCL high time is configured in the I²C Clock High Time register *I2Cn_CLK_HI.clk_hi*. The SCL low time is configured in the I²C Clock Low Time register *I2Cn_CLK_LO.clk_lo* as shown in *Equation 13-3* and *Equation 13-4*.

*Equation 13-3: I²C Clock High Time Calculation*

$$t_{SCL\_HI} = t_{PCLK} \times I2Cn\_CLK\_HI.clk\_hi$$

*Equation 13-4: I²C Clock Low Time Calculation*

$$t_{SCL\_LO} = t_{PCLK} \times I2Cn\_CLK\_LO.clk\_lo$$

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.ckl*, 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.

## 13.19 TX FIFO Preloading

There may be situations where, when operating as a slave, firmware wants to preload the TX FIFO prior to a transmission, such as when clock stretching is disabled. Firmware may also want to respond to an external master requesting data by sending a NACK until the requested data is ready to transmit, rather than sending an ACK and then holding the bus low while the data is prepared. By default, however, Address Match and General Call Match clear the TX FIFO preventing firmware from preloading data into the TX FIFO. Firmware can change this behavior by enabling TX FIFO Preloading.

When TX FIFO Preloading is enabled, the application firmware controls ACKs to the external master using the TX Ready (*I2Cn_TX*_CTRL1.*txrdy*) bit. When *I2Cn_TX*_CTRL1.*txrdy* is set to 0, hardware automatically NACKs all read transactions from the Master. Setting *I2Cn_TX*_CTRL1.*txrdy* 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 must be complete prior to setting the *I2Cn_TX*_CTRL1.*txrdy* field to 1.

The required steps for implementing TX FIFO Preloading in an application are as follow:

1. Set *I2Cn_TX*_CTRL1.*txrdy* to 0
2. Enable TX FIFO Preloading by setting *I2Cn_TXCTRL0.txpreld* to 1.
3. If the TX FIFO Lockout Flag (*I2Cn_INT*_FL0.*txloi*) is set to 1, write 1 to clear the flag and enable writes to the TX FIFO.
4. Enable DMA or Interrupts if required.
5. Load the TX FIFO with the data to send when the Master sends the next read request.
6. Set *I2Cn_TX*_CTRL1.*txrdy* to 1 to automatically let the hardware send the preloaded FIFO on the next read from a Master.
7. *I2Cn_TX*_CTRL1.*txrdy* is cleared by hardware when a read occurs, and data is transmitted from the TX FIFO. Once cleared, the application firmware may repeat the Preloading process or disable TX FIFO Preloading.

*Note: The TX FIFO Lockout flag is set if an error condition occurs while TX FIFO Preloading is enabled.*

## 13.20 Master Mode Receiver Operation

When in Master Mode, starting a Master Receiver operation begins with the following sequence:

1. Write the number of data bytes to be received to *I2Cn_RX_CTRL1.rxcnt*.
2. Write the Slave Address to the TX FIFO with the R/W bit set to 1 in the byte written to the TX FIFO.
3. Send a START condition by setting *I2Cn_MSTR_MODE.start* = 1
4. Slave address is automatically pushed out of the TX FIFO
5. This I²C receives an ACK from the slave, setting *I2Cn_INT_FL0.adracki* = 1
6. This I²C receives data from the slave and automatically replies with an ACK to each.
7. Once *rxcnt* data bytes have been received, this I2C sends a NACK to the slave and sets the Transfer Done Interrupt Status Flag *I2Cn_INT_FL0.donei*
8. If *I2Cn_MSTR_MODE.restart* or *I2Cn_MSTR_MODE.stop* is set, then the I²C peripheral sends a repeated START or STOP, respectively.

## 13.21 I²C Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the I2C0 and I2C1 Register Base Addresses.

*Table 13-2: I2C Registers*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *I2Cn_CTRL0* | R/W | I²C Control 0 Register |
| [0x0004] | *I2Cn_STAT* | RO | I²C Status Register |
| [0x0008] | *I2Cn_INT_FL0* | R/W1C | I²C Interrupt Flags 0 Register |
| [0x000C] | *I2Cn_INT_EN0* | R/W | I²C Interrupt Enable 0 Register |
| [0x0010] | *I2Cn_INT_FL1* | R/W1C | I²C Interrupts Flags 1 Register |
| [0x0014] | *I2Cn_INT_EN1* | R/W | I²C Interrupts Enable 1 Register |
| [0x0018] | *I2Cn_FIFO_LEN* | RO | I²C FIFO Length Register |
| [0x001C] | *I2Cn_RX_CTRL0* | R/W | I²C Receive Control 0 Register |
| [0x0020] | *I2Cn_RX_CTRL1* | R/W | I²C Receive Control 1 Register |
| [0x0024] | *I2Cn_TX_CTRL0* | R/W | I²C Transmit Control 0 Register |
| [0x0028] | *I2Cn_TX_CTRL1* | R/W | I²C Transmit Control 1 Register |
| [0x002C] | *I2Cn_FIFO* | R/W | I²C Transmit and Receive FIFO Register |
| [0x0030] | *I2Cn_MSTR_MODE* | R/W | I²C Master Mode Register |
| [0x0034] | *I2Cn_CLK_LO* | R/W | I²C Clock Low Time Register |
| [0x0038] | *I2Cn_CLK_HI* | R/W | I²C Clock High Time Register |

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0040] | *I2Cn_TIMEOUT* | R/W | I²C Timeout Register |
| [0x0044] | *I2Cn_SLV_ADDR* | R/W | I²C Slave Address Register |
| [0x0048] | *I2Cn_DMA* | R/W | I²C DMA Enable Register |

## 13.22 I²C Register Details

*Table 13-3: I²C Control 0 Register*

| I²C Control 0 Register | | I2Cn_CTRL0 | | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | hsmode | R/W | - | **High Speed Mode**<br>This field must always be set to 0. High speed mode is not supported. |
| 14 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 13 | scl_ppm | R/W | 0 | **SCL Push-Pull Mode Enable**<br>Setting this field enables push-pull mode for the SCL hardware pin. This field should not be set unless any external slave device will never actively drive SCL low.<br>0: SCL operates in standard I²C open-drain mode<br>1: SCL operates in push-pull mode without the need for a pull-up resistor. Only recommended when in Master mode and external slaves will not drive SCL low. |
| 12 | scl_strd | R/W | 0 | **SCL Clock Stretch Control**<br>0: Enable Slave clock stretching<br>1: Disable Slave clock stretching |
| 11 | read | R | 0 | **Read/Write Bit Status**<br>Returns the logic level of the R/W bit on a received address match (*I2Cn_INT_FL0.ami* = 1) or general call match (*I2Cn_INT_FL0.gci* = 1). This bit is valid for three SCL clock cycles after the address match status flag is set. |
| 10 | swoe | R/W | 0 | **Software output Enabled**<br>When set, pins SDA and SCL are directly controlled by the fields *I2Cn_CTRL0.sdao* and *I2Cn_CTRL0.sclo*, rather than the I²C controller. Setting this field to 1 enables software bit bang control of I²C.<br>0: The I²C controller manages the SDA and SCL pins in hardware.<br>1: SDA and SCL are controller by firmware using the *I2Cn_CTRL0.sdao* and *I2Cn_CTRL0.sclo* fields. |
| 9 | sda | R | - | **SDA Status**<br>Returns the current logic level of the SDA pin.<br>0: SDA pin is logic low.<br>1: SDA pin is logic high. |
| 8 | scl | R | - | **SCL Status**<br>Returns the current logic level of the SCL hardware pin.<br>0: SCL pin is logic low.<br>1: SCL pin is logic high. |

| I²C Control 0 Register | | | I2Cn_CTRL0 | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7 | sdao | R/W | 0 | **SDA Pin Control**<br>Set the state of the SDA hardware pin (actively pull low or float).<br>0: Pull SDA Low<br>1: Release SDA<br>*Note: Only valid when I2Cn_CTRL0.swoe=1* |
| 6 | sclo | R/W | 0 | **SCL Pin Control**<br>Set the state of the SCL hardware pin (actively pull low or float).<br>0: Pull SCL low<br>1: Release SCL<br>*Note: Only valid when I2Cn_CTRL0.swoe=1* |
| 5 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 4 | ack | R/W | 0 | **Interactive Receive Mode (IRM) Acknowledge**<br>If IRM is enabled (*I2Cn_CTRL0.irxm* = 1), this field determines if the hardware sends an ACK or a NACK to an IRM transaction.<br>0: Respond to IRM with ACK<br>1: Respond to IRM with NACK |
| 3 | irxm | R/W | 0 | **Interactive Receive Mode (IRXM)**<br>When receiving data, allows for an Interactive Receive Mode (IRM) interrupt event after each received byte of data. The I²C peripheral hardware can be enabled to send either an ACK or NACK for IRM. See *Interactive Receive Mode* section for detailed information.<br>0: Disable Interactive Receive Mode<br>1: Enable Interactive Receive Mode<br>*Note: Only set this field when the I²C bus is inactive.* |
| 2 | gcen | R/W | 0 | **General Call Address Enable**<br>Set this field to 1 to enable General Call Address Acknowledgement.<br>0: Ignore General Call Address<br>1: Acknowledge General Call Address |
| 1 | mst | R/W | 0 | **Master Mode Enable**<br>Setting this field to 1 enables Master mode operation for the I²C peripheral. Setting this field to 0 enables the I²C peripheral for Slave mode operation.<br>0: Slave mode enabled<br>1: Master mode enabled |
| 0 | i2cen | R/W | 0 | **I²C Enable**<br>Set this field to 1 to enable the I²C peripheral.<br>0: I²C peripheral disabled<br>1: I²C peripheral enabled |

*Table 13-4: I²C Status Register*

| I²C Status Register | | | I2Cn_STAT | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:6 | - | R/W | — | **Reserved for Future Use**<br>Do not modify this field. |

| I²C Status Register | | | I2Cn_STAT | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 5 | ckmd | RO | 0 | **SCL Drive Status**<br>This field indicates if an external device is behaving as a master by actively driving the SCL line.<br>0: External device not driving SCL<br>1: External device is a Master actively driving the SCL pin |
| 4 | txf | RO | 0 | **TX FIFO Full**<br>When set, the TX FIFO is full.<br>0: TX FIFO is not full<br>1: TX FIFO full |
| 3 | txe | RO | 1 | **TX FIFO Empty**<br>If set, the TX FIFO is empty.<br>0: TX FIFO is not empty<br>1: TX FIFO is empty |
| 2 | rxf | RO | 0 | **RX FIFO Full**<br>If set, the RX FIFO is full.<br>0: RX FIFO not full<br>1: RX FIFO Full |
| 1 | rxe | RO | 1 | **RX FIFO Empty**<br>If set, the RX FIFO is empty.<br>0: RX FIFO is not empty<br>1: RX FIFO is empty |
| 0 | busy | RO | 0 | **Bus Busy**<br>If set, the I²C bus is active.<br>0: Bus is idle<br>1: Bus is busy |

*Table 13-5: I²C Interrupt Flag 0 Register*

| I²C Interrupt Flag 0 Register | | | I2Cn_INT_FL0 | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | txloi | R/W1C | 0 | **TX FIFO Locked Interrupt Flag**<br>If set, the TX FIFO is locked and writes to the TX FIFO are ignored. This field is set to 1 by hardware to prevent stale data from being transmitted from the TX FIFO. When set, the TX FIFO is automatically flushed. Writes to the TX FIFO are ignored until this flag is cleared. Write 1 to clear.<br>0: TX FIFO not locked.<br>1: TX FIFO is locked and all writes to the TX FIFO are ignored. |
| 14 | stoperi | R/W1C | 0 | **Out of Sequence STOP Interrupt Flag**<br>This flag is set if a STOP condition occurs on the I²C Bus out of expected sequence. Write 1 to clear this field. Writing 0 has no effect.<br>0: Error condition has not occurred.<br>1: Out of sequence STOP condition occurred. |

| I²C Interrupt Flag 0 Register | | | I2Cn_INT_FL0 | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 13 | strteri | R/W1C | 0 | **Out of Sequence START Interrupt Flag**<br>This flag is set if a START condition occurs on the I²C Bus out of expected sequence. Write 1 to clear this field. Writing 0 has no effect.<br>0: Error condition has not occurred.<br>1: Out of sequence START condition occurred. |
| 12 | dnreri | R/W1C | 0 | **Slave Mode Do Not Respond Interrupt Flag**<br>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.<br>0: Error condition has not occurred.<br>1: I²C address match has occurred and either the TX or RX FIFO is not configured. |
| 11 | dateri | R/W1C | 0 | **Master Mode: Data NACK from External Slave Interrupt Flag**<br>This flag is set by hardware if a NACK is received from a slave on the I²C bus. This flag is only valid if the I2Cn peripheral is configured for Master Mode operation. Write 1 to clear. Write 0 has no effect.<br>0: Error condition has not occurred.<br>1: Data NACK received from a slave. |
| 10 | adreri | R/W1C | 0 | **Master Mode: Address NACK from Slave Error Flag**<br>This flag is set by hardware if an Address NACK is received from a slave on the I²C bus. This flag is only valid if the I2Cn peripheral is configured for Master Mode operation. Write 1 to clear. Write 0 has no effect.<br>0: Error condition has not occurred.<br>1: Address NACK received from a slave. |
| 9 | toeri | R/ W1C | 0 | **Timeout Error Interrupt Flag**<br>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.<br>0: Timeout error has not occurred.<br>1: Timeout error occurred. |
| 8 | arberi | R/ W1C | 0 | **Master Mode Arbitration Lost Interrupt Flag**<br>Write 1 to clear. Write 0 has no effect.<br>0: Condition has not occurred.<br>1: Condition occurred. |
| 7 | adracki | R/ W1C | 0 | **Master Mode: Address ACK from External Slave Interrupt Flag**<br>This field is set when a slave address ACK is received. Write 1 to clear. Write 0 has no effect.<br>0: Condition has not occurred.<br>1: The slave device ACK for the address was received. |
| 6 | stopi | R/ W1C | 0 | **Slave Mode: STOP Condition Interrupt Flag**<br>This flag is set by hardware when a STOP condition is detected on the I2C bus. Write 1 to clear. Write 0 has no effect.<br>0: Stop condition has not occurred<br>1: Stop condition occurred |
| 5 | txthi | RO | 1 | **TX FIFO Threshold Level Interrupt Flag**<br>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.<br>0: TX FIFO contains more bytes than the TX threshold level.<br>1: TX FIFO contains TX threshold level or fewer bytes (Default). |

| I²C Interrupt Flag 0 Register | | I2Cn_INT_FL0 | | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4 | rxthi | RO | 1 | **RX FIFO Threshold Level Interrupt Flag**<br>This field is set by hardware if the number of bytes in the Receive FIFO is greater than or equal top the Receive FIFO threshold level. This field is automatically cleared when the RX FIFO contains fewer bytes than the RX threshold setting.<br>0: RX FIFO contains fewer bytes than the RX threshold level.<br>1: RX FIFO contains at least RX threshold level of bytes (Default). |
| 3 | ami | R/W1C | 0 | **Slave Mode: Address Match Status Interrupt Flag**<br>In Slave Mode operation, a slave mode address match condition has occurred. Write 1 to clear. Writing 0 has no effect.<br>0: Slave address match has not occurred.<br>1: Slave address match occurred. |
| 2 | gci | R/W1C | 0 | **Slave Mode: General Call Address Match Received Interrupt Flag**<br>In Slave Mode operation, a general call address match condition has occurred. Write 1 to clear. Writing 0 has no effect.<br>0: General call address match has not occurred.<br>1: General call address match occurred. |
| 1 | irxmi | R/W1C | 0 | **Interactive Receive Mode Interrupt Flag**<br>Write 1 to clear. Writing 0 is ignored.<br>0: Interrupt condition has not occurred.<br>1: Interrupt condition occurred. |
| 0 | donei | R/W1C | 0 | **Transfer Complete Interrupt Flag**<br>This flag is set for both Master and Slave mode for both transmit and receive operations on the SCL falling edge after an ACK is received or sent. Write 1 to clear. Writing 0 has no effect.<br>0: Transfer is not complete.<br>1: Transfer complete. |

*Table 13-6: I²C Interrupt Enable 0 Register*

| I²C Interrupt Enable 0 Register | | I2Cn_INT_EN0 | | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | txloie | R/W | 0 | **TX FIFO Locked Out Interrupt Enable**<br>Set this field to enable events for TX FIFO lock events.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 14 | stoperie | R/W | 0 | **Out of Sequence STOP condition detected Interrupt Enable**<br>Set this field to enable events for an out of sequence STOP condition event.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 13 | strterie | R/W | 0 | **Out of Sequence START condition detected Interrupt Enable**<br>Set this field to enable events for an out of sequence START condition event.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |

| I²C Interrupt Enable 0 Register | | | I2Cn_INT_EN0 | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 12 | dnrerie | R/W | 0 | **Slave Mode Do Not Respond Interrupt Enable**<br>Set this field to enable events in Slave Mode when the Do Not Respond condition occurs.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 11 | daterie | R/W | 0 | **Master Mode Received Data NACK from Slave Interrupt Enable**<br>Set this field to enable events for Master Mode external device data NACK events.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 10 | adrerie | R/W | 0 | **Master Mode Received Address NACK from Slave Interrupt Enable**<br>Set this field to enable events for Master Mode slave device address NACK events.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 9 | toerie | R/W | 0 | **Timeout Error Interrupt Enable**<br>Set this field to enable events for a timeout error interrupt event.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 8 | arberie | R/W | 0 | **Master Mode Arbitration Lost Interrupt Enable**<br>Set this field to enable events in Master Mode for arbitration lost events.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 7 | adrackie | R/W | 0 | **Received Address ACK from Slave Interrupt Enable**<br>Set this field to enable events for Master Mode slave device address ACK events.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 6 | stopie | R/W | 0 | **STOP Condition Detected Interrupt Enable**<br>Set this field to enable interrupt events for STOP conditions.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 5 | txthie | R/W | 0 | **TX FIFO Threshold Level Interrupt Enable**<br>Set this field to enable interrupt events when a TX FIFO threshold event occurs.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 4 | rxthie | R/W | 0 | **RX FIFO Threshold Level Interrupt Enable**<br>Set this field to enable interrupt events when an RX FIFO threshold event occurs.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 3 | amie | R/W | 0 | **Slave Mode Incoming Address Match Interrupt Enable**<br>Set this field to enable the slave mode address match interrupt event.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 2 | gcie | R/W | 0 | **Slave Mode General Call Address Match Received Interrupt Enable**<br>Set this field to enable the slave mode general call address match received interrupt event.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |

| I²C Interrupt Enable 0 Register | | I2Cn_INT_EN0 | | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 1 | irxmie | R/W | 0 | **Interactive Receive Interrupt Enable**<br>Set this field to enable the interactive receive interrupt event.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 0 | doneie | R/W | 0 | **Transfer Complete Interrupt Enable**<br>Set this field to enable the transfer complete interrupt event.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |

*Table 13-7: I²C Interrupt Flag 1 Register*

| I²C Interrupt Status Flags 1 Register | | I2Cn_INT_FL1 | | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:2 | - | R/W | — | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | txufi | R/W1C | 0 | **Slave Mode: TX FIFO Underflow Status Flag**<br>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.<br>0: Slave mode TX FIFO underflow condition has not occurred.<br>1: Slave mode TX FIFO underflow condition occurred. |
| 0 | rxofi | R/W1C | 0 | **Slave Mode: RX FIFO Overflow Status Flag**<br>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.<br>0: Slave mode RX FIFO overflow event has not occurred.<br>1: Slave mode RX FIFO overflow condition occurred (data lost). |

*Table 13-8: I²C Interrupt Enable 1 Register*

| I²C Interrupt Enable 1 Register | | I2Cn_INT_EN1 | | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | txufie | R/W | 0 | **Slave Mode TX FIFO Underflow Interrupt Enable**<br>In slave mode operation, set this field to enable the TX FIFO underflow interrupt.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 0 | rxofie | R/W | 0 | **Slave Mode RX FIFO Overflow Interrupt Enable**<br>In slave mode operation, set this field to enable the RX FIFO overflow interrupt.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |

*Table 13-9: I²C FIFO Length Register*

| I²C FIFO Length Register | | I2Cn_FIFO_LEN | | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15:8 | txlen | RO | 8 | **TX FIFO Length**<br>Returns the length of the TX FIFO.<br>8: 8-byte TX FIFO. |
| 7:0 | rxlen | RO | 8 | **RX FIFO Length**<br>Returns the length of the RX FIFO.<br>8: 8-byte RX FIFO. |

*Table 13-10: I²C Receive Control 0 Register*

| I²C Receive Control Register 0 | | I2Cn_RX_CTRL0 | | [0x001C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11:8 | rxth | R/W | 0 | **RX FIFO Threshold Level**<br>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.rxthi* bit indicating an RX FIFO threshold level event.<br>0: 0 bytes or more in the RX FIFO causes a threshold event.<br>1: 1+ bytes in the RX FIFO triggers a receive threshold event (recommended minimum value).<br>…<br>8: RX FIFO threshold event only occurs when the RX FIFO is full. |
| 7 | rxfsh | R/W1O | 0 | **Flush RX FIFO**<br>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.<br>0: RX FIFO flush complete or not active.<br>1: Flush the RX FIFO |
| 6:1 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 0 | dnr | R/W | 0 | **Do Not Respond**<br>Slave mode operation only.<br>0: If the RX FIFO contains data and an external master requests a WRITE transaction, respond to an address match with an ACK but NACK the subsequent data byte(s). (No additional data is written into the RX FIFO.)<br>1: If the RX FIFO contains data and a master requests a WRITE transaction, do not respond to an address match and send a NACK instead. |

*Table 13-11: I²C Receive Control 1 Register*

| I²C Receive Control 1 Register | | I2Cn_RX_CTRL1 | | [0x0020] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11:8 | rxfifo | R | 0 | **RX FIFO Byte Count Status**<br>Returns the number of bytes currently in the RX FIFO.<br>0: No data in the RX FIFO.<br>...<br>8: 8 bytes in the RX FIFO (max value). |
| 7:0 | rxcnt | R/W | 1 | **RX FIFO Transaction Byte Count Configuration**<br>When in Master Mode, write the number of bytes to be received in a transaction from 1 to 256. 0x00 represents 256.<br>0: 256 byte receive transaction.<br>1: 1 byte receive transaction.<br>2: 2 byte receive transaction.<br>…<br>255: 255 byte receive transaction. |

*Table 13-12: I²C Transmit Control 0 Register*

| I²C Transmit Control Register 0 | | I2Cn_TX_CTRL0 | | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11:8 | txth | R/W | 0 | **TX FIFO Threshold Level**<br>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.txthi* is set indicating a TX FIFO Threshold Event occurred.<br>0: 0 bytes remaining in the TX FIFO triggers a TX FIFO threshold event.<br>1: 1 byte or less remaining in the TX FIFO triggers a TX FIFO threshold event (recommended minimum value).<br>…<br>7: 7 or fewer bytes remaining in the TX FIFO triggers a TX FIFO threshold event |
| 7 | txfsh | R/W1O | 0 | **TX FIFO Flush**<br>Write this field to 1 to initiate a TX FIFO flush, clearing all remaining data from the transmit FIFO.<br>0: TX FIFO flush is complete or not active.<br>1: Flush the TX FIFO<br>*Note: Hardware automatically clears this bit to 0 after it is written to 1 when the flush is completed.*<br>If *I2Cn_INT_FL0.txloi* = 1, then *I2Cn_TX_CTRL0.txfsh* = 1. |
| 6:1 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| I²C Transmit Control Register 0 | | I2Cn_TX_CTRL0 | | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | txpreld | R/W | 0 | **TX FIFO Preload Mode Enable**<br>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.txloi.*<br>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.txloi.* This allows firmware to preload data into the TX FIFO. The status of the I2C is controllable at *I2Cn_TX_CTRL1.txrdy.* |

*Table 13-13: I²C Transmit Control 1 Register*

| I²C Transmit Control Register 1 | | I2Cn_TX_CTRL1 | | [0x0028] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify. |
| 11:8 | txfifo | R | 0x0 | **Transmit FIFO Byte Count Status**<br>Contains the number of bytes in the TX FIFO |
| 7:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify. |
| 1 | txlast | R/1 | 0 | **Slave Mode Transmit Last**<br>This bit decides what to do if the I²C is in Slave Mode, is transmitting data to a Master, and the TX FIFO is empty.<br>0: Hold SCL low. This pauses transmission until data is written to the TX FIFO.<br>1: End transaction by releasing SCL.<br>Cleared on a STOP/RESTART condition, or if *I2Cn_INT_FL0txloi*=1 (transmit FIFO locked for writing). |
| 0 | txrdy | R/1 | 1 | **Transmit FIFO Preload Ready Status**<br>When TX FIFO Preload Mode is enabled, *I2Cn_TX_CTRL0.txpreld* = 1, this bit is automatically cleared to 0. While this bit is 0, if the I²C hardware receives a slave address match a NACK is sent. Once the I²C hardware is ready (firmware has preloaded the TX FIFO, configured the DMA, etc.) application firmware must set this bit to 1 so the I²C hardware will send an ACK on a slave address match.<br>When TX FIFO Preload Mode is disabled, *I2Cn_TX_CTRL0.txpreld* = 1, this bit is forced to 1 and the I²C hardware behaves normally. |

*Table 13-14: I²C Data Register*

| I²C Data Register | | I2Cn_FIFO | | [0x002C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:8 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7:0 | data | R/W | 0xFF | **I2C FIFO Data Register**<br>Reads from this register pops data off the RX FIFO.<br>Writes to this register pushes data onto the TX FIFO.<br>Reading from an empty RX FIFO returns 0xFF.<br>Writes to a full TX FIFO are ignored. |

*Table 13-15: I²C Master Mode Control Register*

| I²C Master Mode Control Register | | | I2Cn_MSTR_MODE | [0x0030] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:8 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7 | sea | R/W | 0 | **Slave Extended Addressing**<br>0: Send a 7-bit address to the slave<br>1: Send a 10-bit address to the slave |
| 6:3 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 2 | stop | R/W1O | 0 | **Send STOP Condition**<br>1: Send a STOP Condition<br>*Note: This bit is automatically cleared by hardware when the STOP condition begins.* |
| 1 | restart | R/W1O | 0 | **Send Repeated START Condition**<br>After sending data to a slave, instead of sending a STOP condition the master may send another START to retain control of the bus.<br>1: Send a Repeated START<br>*Note: This bit is automatically cleared by hardware when the repeated START condition begins.* |
| 0 | start | R/W1O | 0 | **Start Master Mode Transfer**<br>1: Start Master Mode Transfer<br>*Note: This bit is automatically cleared by hardware when the transfer is completed or aborted.* |

*Table 13-16: I²C SCL Low Control Register*

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

*Table 13-17: I²C SCL High Control Register*

| I²C Clock High Control Register | | | I2Cn_CLK_HI | [0x0038] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:9 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| I²C Clock High Control Register | | I2Cn_CLK_HI | | [0x0038] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 8:0 | scl_hi | R/W | 1 | **Clock High Time**<br>In Master Mode, this configures the SCL high time.<br>$t_{SCL\_HI} = {}^{1}/_{f_{I2C\_CLK}} \times (scl\_hi + 1)$<br>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 irxmi during Interactive Receive Mode.<br>*Note: 0 is not a valid setting for this field.* |

*Table 13-18: I²C Timeout Register*

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

*Table 13-19: I²C Slave Address Register*

| I²C Slave Address Register | | I2Cn_SLV_ADDR | | [0x0044] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field |
| 15 | ea | R/W | 0 | **Slave Mode Extended Address Select**<br>In Slave Mode operation this field sets the slave address as either 7-bit or 10-bit.<br>0: 7-bit addressing.<br>1: 10-bit addressing. |
| 14:10 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 9:0 | sla | R/W | 0 | **Slave Mode Slave Address**<br>In Slave Mode Operation, (*I2Cn_CTRL0.mstr* = 0), set this field to the slave address for the I2Cn port.<br>*Note: I2Cn_SLV_ADDR.ea controls if this field is a 7-bit or 10-bit address.* |

*Table 13-20: I²C DMA Register*

| I²C DMA Register | | | I2Cn_DMA | [0x0048] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | rxen | R/W | 0 | **RX DMA Channel Enable**<br>0: Disable RX DMA channel<br>1: Enable RX DMA channel |
| 0 | txen | R/W | 0 | **TX DMA Channel Enable**<br>0: Disable TX DMA channel<br>1: Enable TX DMA channel |

# 14 Pulse Train Engine

The Pulse Train Engine includes 16 independent pulse train engines, designated PT0 to PT15. Each pulse train engine can either operate in Square Wave mode which generates a continuous 50% duty-cycle square wave, or Pulse Train mode which generates a continuous programmed bit pattern from 2- to 32-bits in length. Pulse train engines are used independently or may be synchronized together to generate signals in unison. The frequency of each generated output can be set separately based on a divisor of the Peripheral Clock.

## 14.1 Pulse Train Engine Features

The Pulse Train Engine supports up to 16 independent pulse train outputs with individually programmable modes, patterns and output enables. The PTE uses the Peripheral Clock (PCLK) for the PTE clock, $f_{PTE\_CLK} = f_{PCLK}$, ensuring all pulse train outputs use the same clock source.

- Independent or synchronous pulse train output operation
- Atomic Enable and Atomic Disable
  - Synchronous enable or disable of pulse train output(s) without modification to non-intended pulse train outputs
- Multiple Output Modes:
  - Square Wave Output mode generates a repeating square wave (50% duty cycle)
  - Pattern Output mode for generating a customizable output wave based on a programmable bit pattern from 2 to 32 output cycles
- Global Clock for all generated outputs
- Individual rate configuration for each pulse train output
- Configuration registers are modifiable while the pulse train engine is running
- Pulse train outputs can be halted and resumed at the same point

## 14.2 Engine

The Pulse Train Engine uses the Peripheral Clock as the peripheral input clock, $f_{PTE\_CLK} = f_{PCLK}$. Each pulse train output is individually configurable and independently controlled.

The following sections describe the available configuration options for each individual pulse train output.

### 14.2.1 *Pulse Train Output Modes*

Each pulse train output supports the following modes:

- Pulse Train Mode
- Bit Patter Length
- Square Wave Mode

#### *14.2.1.1 Pulse Train Mode*

When Pulse Train x (PTn) is configured in Pulse Train mode, the configuration also includes the bit length (up to 32-bits) of the custom Pulse Train. This is configured using the 5-bit field *PTn_RATE_LENGTH.mode*.

```
PTn_RATE_LENGTH.mode = 1    (PTn configured in Square Wave mode)
PTn_RATE_LENGTH.mode > 1    (PTn configured in Pulse Train mode. The value of mode is the
```

```
pattern bit length.)
PTn_RATE_LENGTH.mode =0    (PTn bit length configured for Pulse Train mode, 32-bit pattern)
```

If in Pulse Train Mode, Set the Bit Pattern

If an output is set to Pulse Train mode, then configure a custom bit pattern from 2-bits to 32-bits in length in the 32-bit register *PTn_TRAIN*. The pattern is shifted out least significant bit (LSB) first. If the output is configured in Square Wave mode, then the *PTn_TRAIN* register is ignored.

*Equation 14-1: Pulse Train Mode Output Function*

$$PTn\_TRAIN = [Bit\ pattern\ for\ PTn]$$

Synchronize Two or More Outputs, if Needed

The write-only register *PTG_RESYNC* "PT Global Resync" allows two or more outputs to be reset and synchronized. Write to any bit in *PTG_RESYNC* to simultaneously reset any outputs in Pulse Train mode to the beginning of the pattern (the LSB) set in the *PTn_TRAIN* bit-pattern register, and reset the output to 0 for outputs in Square Wave mode.

#### 14.2.1.2 Pulse Train Loop Mode

By default, a pulse train engine runs indefinitely until it is disabled by firmware.

A pulse train engine can be configured to repeat its pattern a specified number of times, called Loop mode. To select Loop mode, write a non-zero value to the 16-bit field *PTn_LOOP.count*. When the pulse train engine is enabled, this field decrements by 1 each time a complete pattern is shifted through the output pin. When the count reaches 0, the output is halted, and the corresponding flag in the *PTG_INTFL* register is set.

#### 14.2.1.3 Pulse Train Loop Delay

If the pulse train is configured in Loop mode, a delay can be inserted after each repeated output pattern. To enable a delay, write the 12-bit field *PTn_LOOP.delay* with the number of Peripheral Clock cycles to delay between the most significant bit (MSB) of the last pattern to the least-significant bit (LSB) of the next pattern. During this delay, the output is held at the MSB of the last pattern. If the loop counter has not reached 0, then it is decremented when the next pattern starts.

#### 14.2.1.4 Pulse Train Automatic Restart Mode

When an engine in Pulse Train mode is in Loop mode and stops when the loop count reaches 0, this is called a Stop Event. A Stop Event can optionally trigger one or more pulse trains to restart from the beginning. This is called Automatic Restart mode. While only pulse train engines operating in Pulse Train mode can operate in Loop mode and can optionally restart a pulse train engine, Automatic Restart mode can trigger pulse train engines operating in Pulse Train mode or in Square Wave mode.

If a running pulse train engine is triggered by another pulse train's Stop Event, Automatic Restart restarts the running pulse train engine from the beginning of its pattern. If a pulse train engine is triggered by another pulse train's Stop Event, and it is not running, Automatic Restart sets the enable bit to 1, and starts the pulse train engine.

The settings for this mode are contained in the *PTn_RESTART* register for each pulse train engine. Note that the configuration for automatic restart is set using the pulse engine(s) triggered by the automatic restart, not the pulse train engine(s) that trigger the automatic restart. For example, the PT8_RESTART register configures which pulse train engine triggers PT8 to restart.

Each pulse train engine can be configured to perform an Automatic Restart when it detects a Stop Event from one or two pulse trains.

- If *PTn_RESTART.on_pt_n_loop_exit* = 1, then pulse train engine n automatically restarts when it detects a Stop Event from pulse train x, where x is the value in the 5-bit field *PTn_RESTART.pt_n_*select.
- If *PTn_RESTART.on_pt_*y_loop*_exit* = 1, then pulse train engine n automatically restarts when it detects a Stop Event from pulse train y, where y is the value in 5-bit field *PTn_RESTART.pt_y_select*.

A pulse train engine can be configured to restart on its own Stop Event, allowing the pulse train to run indefinitely.

Each individual pulse train can be configured for:

- No Automatic Restart
- Automatic Restart triggered by a stop event from pulse train x only
- Automatic Restart triggered by a stop event from pulse train y only
- Automatic Restart triggered by a stop event from both pulse train x and pulse train y

## 14.3 Enabling and Disabling a Pulse Train Output

The *PTG_ENABLE* register is used to enable and disable each of the individual pulse train outputs. Enable a given pulse train output by setting the respective bit in the *PTG_ENABLE* register. Halt a pulse train output by clearing the respective bit in the *PTG_ENABLE* register.

*Note: Prior to changing a pulse train output's configuration the corresponding pulse train output should be halted to prevent unexpected behavior.*

## 14.4 Atomic Pulse Train Output Enable and Disable

Deterministic enable and disable operation is critical for pulse train output that must be synchronized in an application. The *PTG_ENABLE* register does not perform atomic access directly. Atomic operations are supported using the registers *PTG_SAFE_EN*, *PTG_SAFE_DIS*.

For most pulse train peripherals, enabling and disabling individual pulse trains is performed by setting and clearing bits in the global enable/disable register, which for this peripheral is *PTG_ENABLE*. For most Arm Cortex-M microcontrollers, this is usually done by bit banding. Because bit banding performs a read, modify, write (RMW), some pulse trains could start and end during the RMW operation, often with unpredictable results.

To ensure safe and predictable operation, two additional registers are used to enable and disable the outputs.

### 14.4.1 *Pulse Train Atomic Enable*

*PTG_SAFE_EN* "Global Safe Enable" is a write-only register. To safely enable outputs without a RMW, write a 32-bit value to this register with a 1 in the bit positions corresponding to the pulse train engines to be enabled. This immediately sets to 1 the corresponding bits in the *PTG_ENABLE* register to 1, which enables the corresponding pulse train engine. Writing a 0 to any bit position in the *PTG_SAFE_EN* register has no effect on the state of the corresponding pulse train enable bit. If the corresponding pulse train engine is already enabled and running, writing a 1 to that bit position in the *PTG_SAFE_EN* register has no effect.

### 14.4.2 *Pulse Train Atomic Disable*

*PTG_SAFE_DIS* "Global Safe Disable" is a write-only register for disabling a pulse train engine without performing a RMW. To safely disable pulse train engines, write a 32-bit value to this register with a 1 in the bit positions corresponding to the pulse train engines to be disabled. This immediately clears to 0 the corresponding bits in *PTG_ENABLE* which disables the corresponding pulse train engines. Writing a 0 to any bit position in the *PTG_SAFE_DIS* register has no effect on the state of the corresponding pulse train enable bit.

Bit banding is not supported for the *PTG_ENABLE*, *PTG_SAFE_EN*, and *PTG_SAFE_DIS* registers and can have unpredictable results.

## 14.5 Pulse Train Halt and Disable

Once a pulse train engine is enabled and running, it continues to run until one of the following events stops the output:

The corresponding enable bit in the *PTG_ENABLE* register is cleared to 0 to halt the output.

A 1 is written to the corresponding disable bit in the *PTG_SAFE_DIS* register to halt the output.

The corresponding resync bit in the *PTG_RESYNC* register is cleared to 0 to halt and reset the output.

*PTn_LOOP* was initialized to a non-zero value, and the loop count has reached 0 (this has no effect in Square Wave mode; it only applies to Pulse Train mode).

When a pulse train is halted, the corresponding enable bit in *PTG_ENABLE* is automatically cleared to 0.

## 14.6 Pulse Train Interrupts

Each pulse train can generate an interrupt only if it is configured in Pulse Train mode, and the loop counter *PTG_SAFE_DIS* was initialized to a non-zero number. When *PTG_SAFE_DIS* counts down to 0, the corresponding status flag in the *PTG_INTFL* register is set. If the corresponding interrupt enable bit in the *PTG_INTEN* register is set, the event also generates an interrupt.

## 14.7 Pulse Train Engine Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the Pulse Train Engine (PTE) Base Peripheral Address.

If *PTn_LOOP.count* loop counter is set to a non-zero number, when the loop counter counts down to zero then the pulse train engine stops, and the corresponding enable bit is cleared.

*Table 14-1: Pulse Train Engine Registers*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *PTG_ENABLE* | R/W | PT Global Enable/Disable Control |
| [0x0004] | *PTG_RESYNC* | W | PT Global Resync |
| [0x0008] | *PTG_INTFL* | R/1 | PT Stopped Global Status Flags |
| [0x000C] | *PTG_INTEN* | R/W | PT Global Interrupt Enable |
| [0x0010] | *PTG_SAFE_EN* | W | PT Global Safe Enable |
| [0x0014] | *PTG_SAFE_DIS* | W | PT Global Safe Disable |
| [0x0020] | *PTn_RATE_LENGTH* | R/W | PT0 Configuration |
| [0x0024] | *PTn_TRAIN* | R/W | PT0 Pulse Train Mode Bit Pattern |
| [0x0028] | *PTn_LOOP* | R/W | PT0 Loop Control |
| [0x002C] | *PTn_RESTART* | R/W | PT0 Automatic Restart |

## 14.8 Pulse Train Engine Register Details

*Table 14-2: Pulse Train Engine Global Enable/Disable Register*

| PT Global Enable/Disable Control | | PTG_ENABLE | | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| PT Global Enable/Disable Control | | | PTG_ENABLE | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15 | enable_pt15 | R/W | 0 | **Enable/Disable Control for PT15**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 14 | enable_pt14 | R/W | 0 | **Enable/Disable Control for PT14**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 13 | enable_pt13 | R/W | 0 | **Enable/Disable Control for PT13**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event. |
| 12 | enable_pt12 | R/W | 0 | **Enable/Disable Control for PT12**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 11 | enable_pt11 | R/W | 0 | **Enable/Disable Control for PT11**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event. |
| 10 | enable_pt10 | R/W | 0 | **Enable/Disable Control for PT10**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 9 | enable_pt9 | R/W | 0 | **Enable/Disable Control for PT9**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 8 | enable_pt8 | R/W | 0 | **Enable/Disable Control for PT8**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 7 | enable_pt7 | R/W | 0 | **Enable/Disable Control for PT7**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |

| PT Global Enable/Disable Control | | | PTG_ENABLE | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 6 | enable_pt6 | R/W | 0 | **Enable/Disable Control for PT6**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 5 | enable_pt5 | R/W | 0 | **Enable/Disable Control for PT5**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 4 | enable_pt4 | R/W | 0 | **Enable/Disable Control for PT4**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 3 | enable_pt3 | R/W | 0 | **Enable/Disable Control for PT3**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 2 | enable_pt2 | R/W | 0 | **Enable/Disable Control for PT2**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 1 | enable_pt1 | R/W | 0 | **Enable/Disable Control for PT1**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |
| 0 | enable_pt0 | R/W | 0 | **Enable/Disable Control for PT0**<br>1: Enable Pulse Train<br>0: Disable Pulse Train<br>*Note: Manually disabling an active pulse train immediately halts the PT output and does not generate a Stop Event.* |

*Table 14-3: Pulse Train Engine Resync Register*

| PT Resync Register | | | PTG_RESYNC | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | WO | - | **Reserved for Future Use**<br>Do not modify this field. |

| PT Resync Register | | | PTG_RESYNC | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15 | pt15 | WO | - | **Resync Control for PT15**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 14 | pt14 | WO | - | **Resync Control for PT14**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 13 | pt13 | WO | - | **Resync Control for PT13**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 12 | pt12 | WO | - | **Resync Control for PT12**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 11 | pt11 | WO | - | **Resync Control for PT11**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 10 | pt10 | WO | - | **Resync Control for PT10**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |

| PT Resync Register | | | PTG_RESYNC | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 9 | pt9 | WO | - | **Resync Control for PT9**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 8 | pt8 | WO | - | **Resync Control for PT8**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 7 | pt7 | WO | - | **Resync Control for PT7**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 6 | pt6 | WO | - | **Resync Control for PT6**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 5 | pt5 | WO | - | **Resync Control for PT5**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 4 | pt4 | WO | - | **Resync Control for PT4**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |

| PT Resync Register | | | PTG_RESYNC | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 3 | pt3 | WO | - | **Resync Control for PT3**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 2 | pt2 | WO | - | **Resync Control for PT2**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 1 | pt1 | WO | - | **Resync Control for PT1**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |
| 0 | pt0 | WO | - | **Resync Control for PT0**<br>Write 1 to reset the output of the Pulse Train. For Pulse Train mode the output is restarted to the beginning of the output pattern. For Square Wave mode the output is reset to 0.<br>Setting multiple bits simultaneously in this register synchronizes the set outputs.<br>1: Reset/Restart the Pulse Train<br>0: No effect<br>*Note: Writing 1 has no effect if the corresponding Pulse Train is disabled.* |

*Table 14-4:Pulse Train Engine Stopped Interrupt Flag Register*

| PT Stopped Interrupt Flag Register | | | PTG_INTFL | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | pt15 | R/W1C | 0 | **PT15 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |

| PT Stopped Interrupt Flag Register | | PTG_INTFL | | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 14 | pt14 | R/W1C | 0 | **PT14 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 13 | pt13 | R/W1C | 0 | **PT13 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 12 | pt12 | R/W1C | 0 | **PT12 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 11 | pt11 | R/W1C | 0 | **PT11 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 10 | pt10 | R/W1C | 0 | **PT10 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 9 | pt9 | R/W1C | 0 | **PT9 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 8 | pt8 | R/W1C | 0 | **PT8 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 7 | pt7 | R/W1C | 0 | **PT7 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 6 | pt6 | R/W1C | 0 | **PT6 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |

| PT Stopped Interrupt Flag Register | | | PTG_INTFL | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 5 | pt5 | R/W1C | 0 | **PT5 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 4 | pt4 | R/W1C | 0 | **PT4 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 3 | pt3 | R/W1C | 0 | **PT3 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 2 | pt2 | R/W1C | 0 | **PT2 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 1 | pt1 | R/W1C | 0 | **PT1 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |
| 0 | pt0 | R/W1C | 0 | **PT0 Stopped Status Flag**<br>This bit is set to 1 by hardware when the corresponding Pulse Train is in Pulse Train Mode and the loop counter reaches 0. In Square Wave mode, this field is not used. Write 1 to clear.<br>1: Pulse Train is stopped. |

*Table 14-5: Pulse Train Engine Interrupt Enable Register*

| PT Interrupt Enable Register | | | PTG_INTEN | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | pt15 | R/W | 0 | **PT15 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 14 | pt14 | R/W | 0 | **PT14 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |

| PT Interrupt Enable Register | | PTG_INTEN | | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 13 | pt13 | R/W | 0 | **PT13 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 12 | pt12 | R/W | 0 | **PT12 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 11 | pt11 | R/W | 0 | **PT11 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 10 | pt10 | R/W | 0 | **PT10 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 9 | pt9 | R/W | 0 | **PT9 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 8 | pt8 | R/W | 0 | **PT8 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 7 | pt7 | R/W | 0 | **PT7 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 6 | pt6 | R/W | 0 | **PT6 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 5 | pt5 | R/W | 0 | **PT5 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |

| PT Interrupt Enable Register | | | PTG_INTEN | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4 | pt4 | R/W | 0 | **PT4 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 3 | pt3 | R/W | 0 | **PT3 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 2 | pt2 | R/W | 0 | **PT2 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 1 | pt1 | R/W | 0 | **PT1 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |
| 0 | pt0 | R/W | 0 | **PT0 Interrupt Enable**<br>Write 1 to enable the interrupt for the corresponding PT when the flag is set in the *PTG_INTFL* register.<br>1: Interrupt is enabled.<br>0: Interrupt is disabled. |

### 14.8.2 *Pulse Train Engine Safe Enable Register*

A 32-bit value written to this register is used to perform an immediate binary OR with the contents of *PTG_ENABLE*. The result is immediately stored in the *PTG_ENABLE*.

*Table 14-6: Pulse Train Engine Safe Enable Register*

| Pulse Train Engine Safe Enable Register | | | PTG_SAFE_EN | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | W0 | - | **Reserved for Future Use**<br>Always write 0. |
| 15 | safeen_pt15 | WO | - | **Safe Enable Control for PT15**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 14 | safeen_pt14 | WO | - | **Safe Enable Control for PT14**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |

| Pulse Train Engine Safe Enable Register | PTG_SAFE_EN | [0x0010] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 13 | safeen_pt13 | WO | - | **Safe Enable Control for PT1**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 12 | safeen_pt12 | WO | - | **Safe Enable Control for PT12**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 11 | safeen_pt11 | WO | - | **Safe Enable Control for PT11**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 10 | safeen_pt10 | WO | - | **Safe Enable Control for PT10**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 9 | safeen_pt9 | WO | - | **Safe Enable Control for PT9**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 8 | safeen_pt8 | WO | - | **Safe Enable Control for PT8**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 7 | safeen_pt7 | WO | - | **Safe Enable Control for PT7**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 6 | safeen_pt6 | WO | - | **Safe Enable Control for PT6**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 5 | safeen_pt5 | WO | - | **Safe Enable Control for PT5**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 4 | safeen_pt4 | WO | - | **Safe Enable Control for PT4**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 3 | safeen_pt3 | WO | - | **Safe Enable Control for PT3**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |

| Pulse Train Engine Safe Enable Register | PTG_SAFE_EN | [0x0010] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 2 | safeen_pt2 | WO | - | **Safe Enable Control for PT2**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 1 | safeen_pt1 | WO | - | **Safe Enable Control for PT1**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |
| 0 | safeen_pt0 | WO | - | **Safe Enable Control for PT0**<br>Writing a 1 sets the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Enable corresponding Pulse Train<br>0: No effect |

### 14.8.3 *Pulse Train Engine Safe Disable Register*

A 32-bit value written to this register is used to immediately disable any corresponding pulse train in the *PTG_ENABLE* register. The result is immediately stored in *PTG_ENABLE*. Setting a field to 1 disables the corresponding pulse train immediately.

*Table 14-7: Pulse Train Engine Safe Disable Register*

| Pulse Train Engine Safe Disable Register | PTG_SAFE_DIS | [0x0014] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:16 | - | WO | - | **Reserved for Future Use**<br>Always write 0. |
| 15 | safedis_pt15 | WO | - | **Safe Disable Control for PT15**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 14 | safedis_pt14 | WO | - | **Safe Disable Control for PT14**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 13 | safedis_pt13 | WO | - | **Safe Disable Control for PT13**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 12 | safedis_pt12 | WO | - | **Safe Disable Control for PT12**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 11 | safedis_pt11 | WO | - | **Safe Disable Control for PT11**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |

| Pulse Train Engine Safe Disable Register | PTG_SAFE_DIS | [0x0014] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 10 | safedis_pt10 | WO | - | **Safe Disable Control for PT10**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 9 | safedis_pt9 | WO | - | **Safe Disable Control for PT9**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 8 | safedis_pt8 | WO | - | **Safe Disable Control for PT8**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 7 | safedis_pt7 | WO | - | **Safe Disable Control for PT7**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 6 | safedis_pt6 | WO | - | **Safe Disable Control for PT5**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 5 | safedis_pt5 | WO | - | **Safe Disable Control for PT4**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 4 | safedis_pt4 | WO | - | **Safe Disable Control for PT3**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 3 | safedis_pt3 | WO | - | **Safe Disable Control for PT2**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 2 | safedis_pt2 | WO | - | **Safe Disable Control for PT1**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 1 | safedis_pt1 | WO | - | **Safe Disable Control for PT0**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |
| 0 | safedis_pt0 | WO | - | **Safe Disable Control for PT0**<br>Writing a 1 clears the corresponding enable bit in the *PTG_ENABLE* register.<br>1: Disable corresponding Pulse Train<br>0: No effect |

*Table 14-8: Pulse Train Engine Configuration Register*

| Pulse Train Configuration Register | | | PTn_RATE_LENGTH | [0x0020] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:27 | mode | R/W | 0b00001 | **Square Wave or Pulse Train Output Mode**<br>Sets either Pulse Train mode with length, or Square Wave mode.<br>0: Pulse Train mode, 32-bits long<br>1: Square Wave mode<br>2: Pulse Train mode, 2-bits long<br>3: Pulse Train mode, 3-bits long<br>… etc …<br>31: Pulse Train mode, 31-bits long<br>*Note: If this field is set to 1, Square Wave mode, the PTn_LENGTH register is not used.* |
| 26:0 | rate_control | R/W | 0 | **Pulse Train Enable and Rate Control**<br>Defines the rate at which the output for PTn changes state by setting the divisor of the PT Clock, where:<br>$f_{PTn} = \frac{f_{PTE\_CLK}}{rate\_control}$<br>0: Output halted<br>All other values use the equation above. |

*Table 14-9: Pulse Train Mode Bit Pattern Register*

| Pulse Train Mode Bit Pattern | | | PTn_TRAIN | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | ptn_train | R/W | 0 | **Pulse Train Mode Bit Pattern**<br>Write the repeating bit pattern that is shifted out, LSB first, when configured in Pulse Train mode. Set the bit pattern length with the *PTn_RATE_LENGTH.mode* field.<br>*Note: This register is ignored in Square Wave mode.*<br>*Note: 0 and 1 are invalid values for this register.* |

*Table 14-10: Pulse Train n Loop Configuration Register*

| Pulse Train Loop Configuration | | | PTn_LOOP | [0x0028] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:28 | - | R/OW | - | **Reserved for Future Use**<br>Do not modify this field. |
| 27:16 | delay | R/W | 0 | **Pulse Train Delay Between Loops**<br>Sets the delay, in number of Peripheral Clock cycles, that the output pauses between loops. The bitfield count is decremented after the delay.<br>If firmware writes a 0 to bitfield count, this field is ignored. |

| Pulse Train Loop Configuration | | | PTn_LOOP | [0x0028] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | count | R/W | 0 | **Pulse Train Loop Countdown**<br>Sets the number of times a pulse train pattern is repeated until it automatically stops.<br>Reading this field returns the number of loops remaining.<br>When this field counts down to zero, the corresponding *PTG_INTFL* flag is set.<br>Write 0 to have the pulse train pattern repeat indefinitely.<br>Ignored in Square Wave mode.<br>If *PTn_LOOP.count* loop counter is set to a non-zero number, when the loop counter counts down to zero then the pulse train engine stops, and the corresponding enable bit is cleared. |

*Table 14-11: Pulse Train n Automatic Restart Configuration Register*

| Pulse Train Automatic Restart Configuration | | | PTn_RESTART | [0x002C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:28 | - | R/W | - | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | on_pt_y_loop_exit | R/W | 0 | **Automatic Restart for PTn on PYy Stop**<br>Enable Automatic Restart for this Pulse Train on a PTy Stop Event<br>1: When PTy has a Stop Event, automatically restart this pulse train from the beginning of its pattern.<br>0: Disable Automatic Restart |
| 14:11 | - | R/W | - | **Reserved for Future Use**<br>Do not modify this field. |
| 12:8 | pt_y_select | R/W | 0 | **Select PTy**<br>Write the Pulse Train number representing Pty. This engine must be in Pulse Train mode.<br>0: PT0<br>1: PT1<br>2: PT2<br>…<br>14: PT14<br>15: PT15<br>*Note: Values above 15 are invalid and ignored.* |
| 7 | on_pt_n_loop_exit | R/W | 0 | **Enable Automatic Restart for this Pulse Train on a PTn Stop Event**<br>1: When PTn has a Stop Event, automatically restart this Pulse Train from the beginning of its pattern.<br>0: Disable Automatic Restart |
| 6:5 | - | R/W | - | **Reserved for Future Use**<br>Do not modify this field. |

| Pulse Train Automatic Restart Configuration | | | PTn_RESTART | [0x002C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4:0 | pt_n_select | R/W | 0 | **Select PTn**<br>Write the Pulse Train number representing PTn. This engine must be in Pulse Train mode.<br>0: PT0<br>1: PT1<br>2: PT2<br>…<br>14: PT14<br>15: PT15<br>*Note: Values above 15 are invalid and ignored.* |

# 15 Timers

The MAX32650–MAX32652 contains six 32-bit, reloadable timers. Each timer provides multiple operating modes:

- 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.

## 15.1 Features

- 32-bit reload counter
- Programmable prescaler with values from 1 to 4096
- Non-overlapping PWM output generation with configurable off-time
- Capture, compare, and capture/compare capability
- Timer pin available 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 interrupt

## 15.2 Basic Operation

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

Each timer mode has a user-configurable timer period, 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 *TMRn_CNT* 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 *TMRn_CNT* to 0x0000 0001 at the end of a timer period, but *TMRn_CNT* is set to 0x0000 0000 following a system reset. This means the first timer period following a system reset will be one timer clock longer than subsequent timer periods if *TMRn_CNT* is not initialized to 0x0000 0001 during the timer configuration step.

Clocking of timer functions is driven by the timer clock frequency, $f_{CNT\_CLK}$. The timer clock frequency is a user-configurable, division of the system peripheral clock, PCLK. Each timer has an independent prescaler, allowing timers to run at different frequencies. The prescaler can be set from 1 to 4096 using the *TMRn_CN.pres3*:*TMRn_CN.pres* fields. Unless otherwise mentioned *Equation 15-1* is used to calculate the timer clock frequency.

*Equation 15-1: Timer Peripheral Clock Equation*

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

Application firmware writes to the timer registers and external events on timer pins are asynchronous events to the slower timer clock frequency. 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 a delay of up to $0.5 \times f_{CNT\_CLK}$ timer clocks before being recognized.

## 15.3 Timer Pin Functionality

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

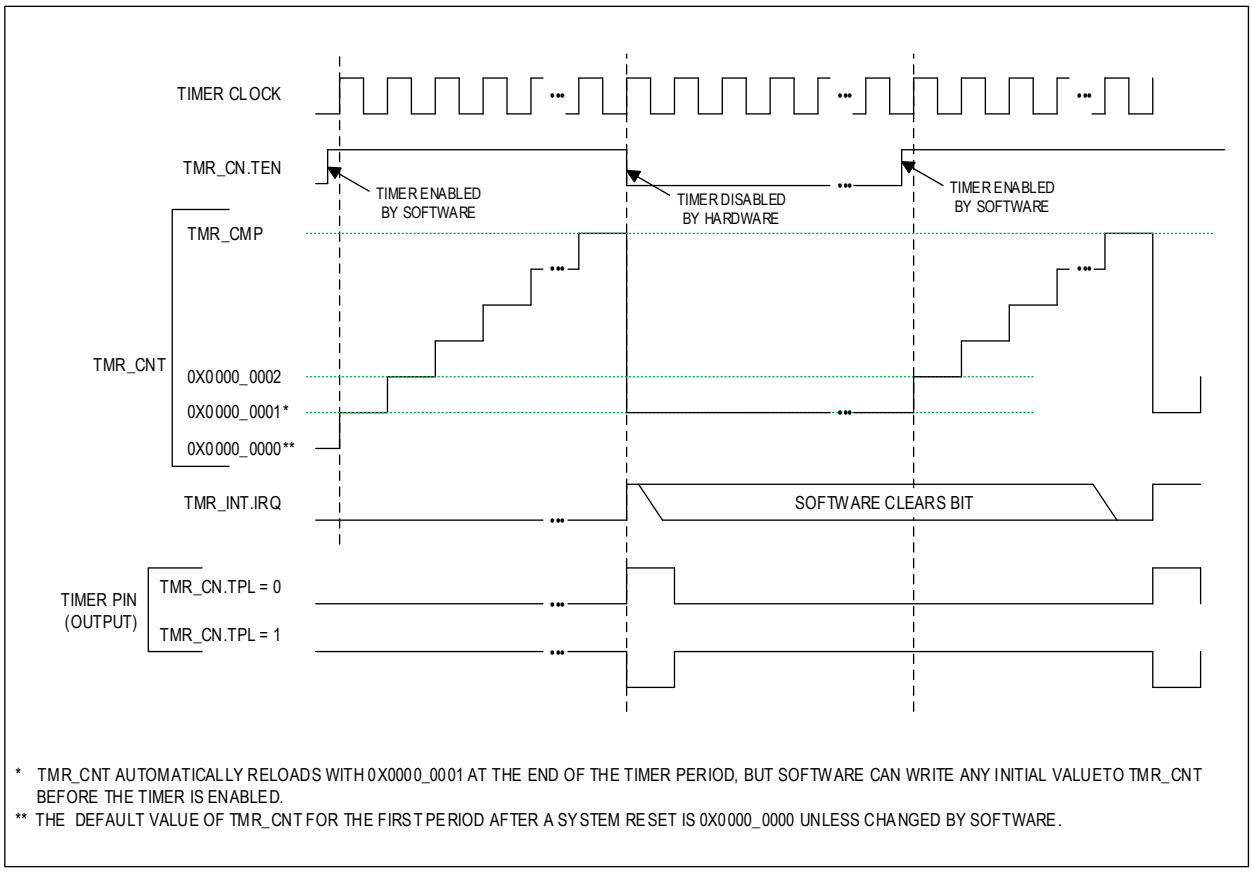
When the timer pin alternate function is enabled, the timer pin will have the same electrical characteristics, such as pullup/pulldown strength, drive strength, etc. as the GPIO mode settings for that pin. When configured as an output, the corresponding bit in the GPIO_OUT register should be configured to match the inactive state of the timer pin for that mode. Configure the timer pin characteristic before enabling the timer. Consult the GPIO section for details on how to configure the GPIO alternate functions for the desired pin

Each timer has a dedicated interrupt flag, *TMRn_INT.irq*, which is set at the end of a timer period. If enabled, an interrupt is generated. Write any value to *TMRn_INT.irq* to clear the interrupt flag.

## 15.4 One-Shot Mode (000b)

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

*Figure 15-1: One-Shot Mode Diagram*




### 15.4.1 *One-Shot Mode Timer Period*

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

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

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

### 15.4.2 One-Shot Mode Configuration

Configure the timer for One-Shot mode by doing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 000b to select One-shot mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. If using the timer pin:
   a. Configure the pin as a timer output and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired (inactive) state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write an initial value to *TMRn_CNT*, if desired. This effects only the first period; subsequent timer periods always reset *TMRn_CNT*= 0x0000 0001.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* = 1 to enable the timer.

Calculate the timer period using *Equation 15-2*, *below*.

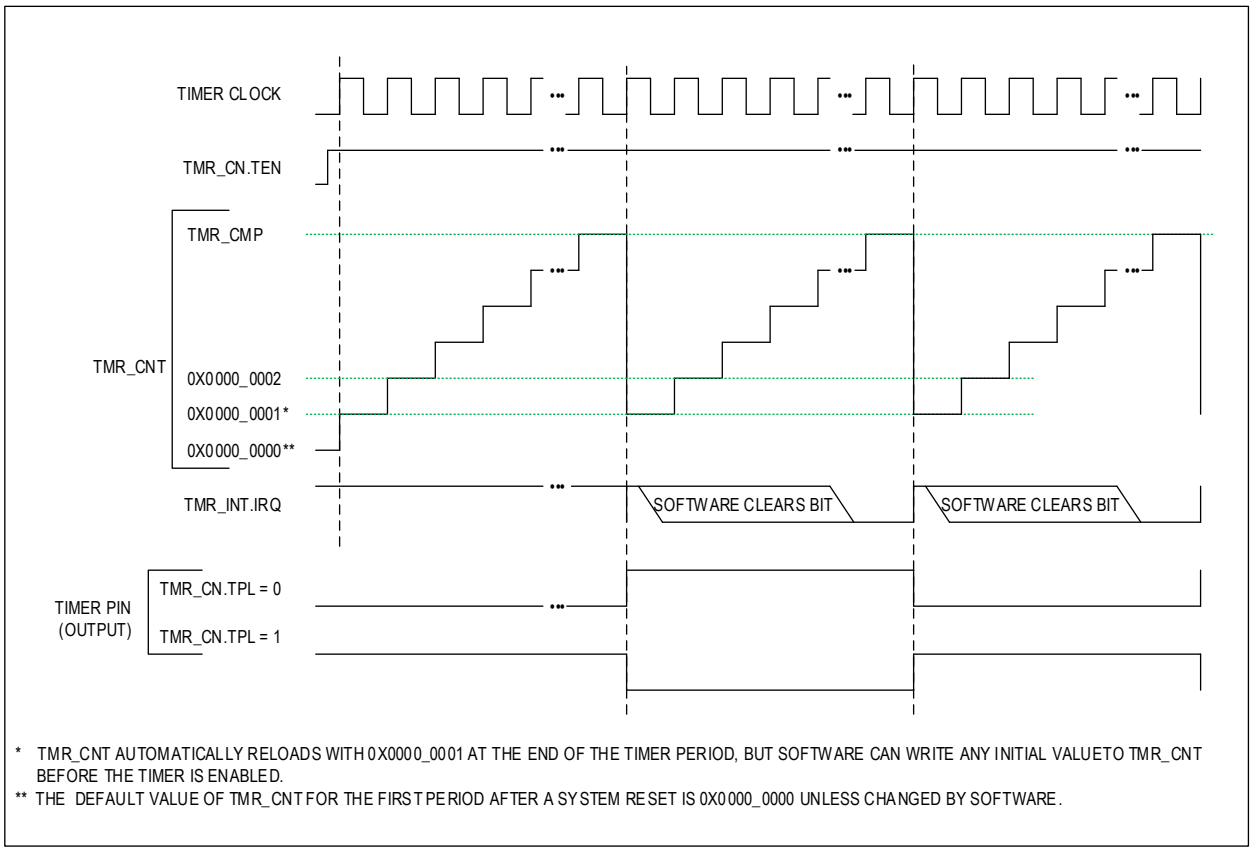
*Equation 15-2: One-shot Mode Timer Period*

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

## 15.5 Continuous Mode (001b)

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

*Figure 15-2: Continuous Mode Diagram*




### 15.5.1 *Continuous Mode Timer Period*

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

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

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

### 15.5.2 *Continuous Mode Configuration*

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

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 001b to select Continuous mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency, $f_{CNT\_CLK}$.
4. If using the timer pin:
   a. Configure the pin as a timer output and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired inactive state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write an initial value to *TMRn_CNT*, if desired. The initial value is only used for the first period; subsequent timer periods always reset the *TMRn_CNT* register to 1.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* to 1 to enable the timer.

The Continuous Mode Timer Period is calculated using *Equation 15-3*.

*Equation 15-3: Continuous Mode Timer Period*

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

## 15.6 Counter Mode (010b)

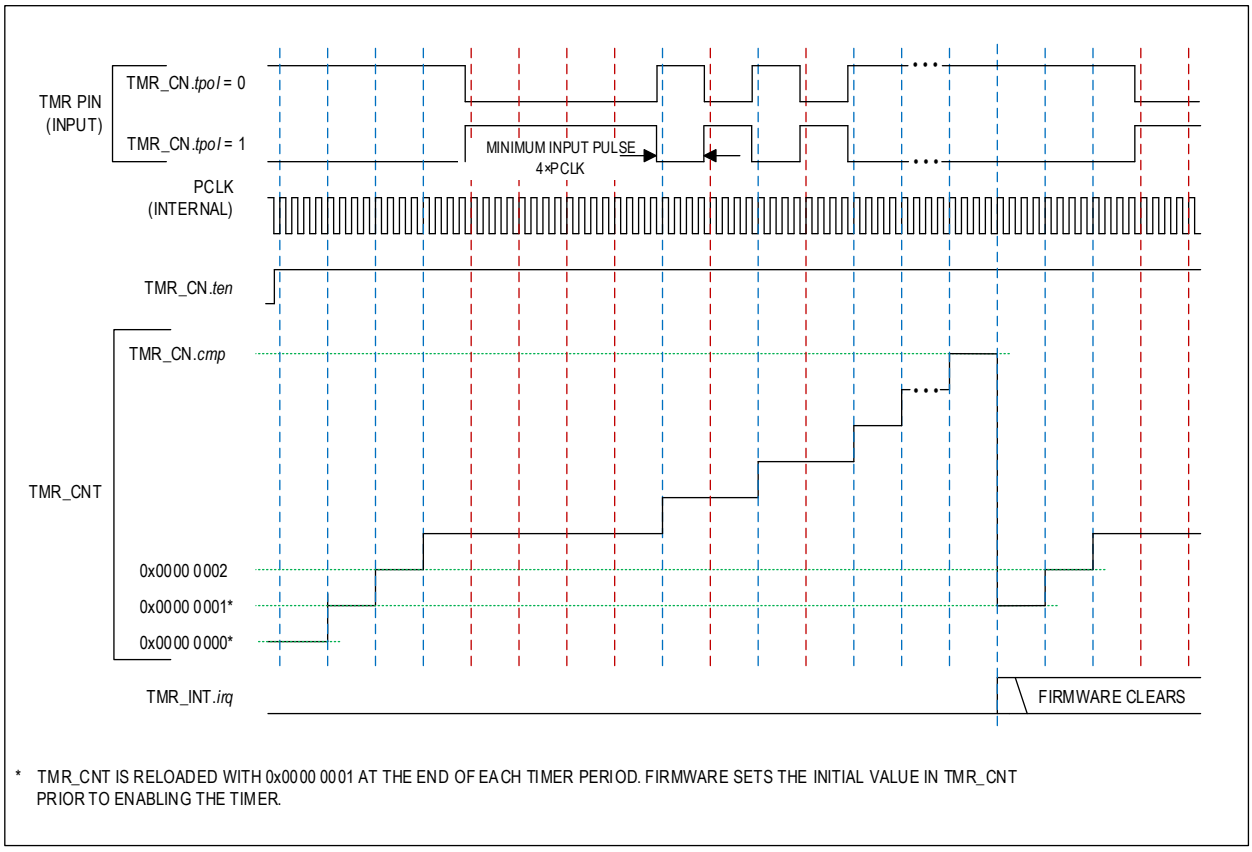
In Counter mode, the timer peripheral increments *TMRn_CNT* when a transition occurs on the timer pin. When *TMRn_CNT* = *TMRn_CMP*, the interrupt bit is set and the *TMRn_CNT* register is set to 0x0000 0001 and continues incrementing. Configure the timer to increment on either the rising edge or the 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% of the PCLK frequency as shown in *Equation 15-4*, *below*.

*Equation 15-4: Counter Mode Maximum Clock Frequency*

$$f_{CTR\_CLK} \leq \frac{f_{PCLK}\ (Hz)}{4}$$

*Figure 15-3: Counter Mode Diagram*




### 15.6.1 *Counter Mode Timer Period*

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

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

1. *TMRn_CNT* is reset to 0x0000 0001. The timer stays enabled and continues incrementing on selected transitions of the timer pin.
2. The timer interrupt bit, *TMRn_INT.irq*, is set on the rising edge of PCKL following the condition of *TMRn_CNT* = *TMRn_CMP*. An interrupt is generated if enabled.

### 15.6.2 *Counter Mode Configuration*

Configure the timer for Counter mode by doing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 010b to select Counter mode.
3. Configure the timer pin:
   a. Configure the pin as a timer input and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired initial (inactive) state.
4. If using the timer interrupt, enable the interrupt and set the interrupt priority.
5. Write an initial value to *TMRn_CNT*, if desired. The initial value is only used for the first timer period; subsequent timer periods always reset *TMRn_CNT* to 1.
6. Write the compare value to *TMRn_CMP*.
7. Set *TMRn_CN.ten* = 1 to enable the timer.

In Counter mode, the number of timer input transitions since starting the timer is calculated using *Equation 15-5*, *below*.

*Equation 15-5: Counter Mode Timer Input Transitions*

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

## 15.7 PWM Mode (011b)

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* register. At the end of the cycle where the *TMRn_CNT* value matches the *TMRn_PWM* value, the timer's output toggles state. The timer continues counting until it reaches the *TMRn_CMP* value.

### 15.7.1 *PWM Mode Timer Period*

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

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

1. The *TMRn_CNT* is reset to 0x0000 0001, and the timer resumes counting.
2. The timer output signal is toggled.
3. The timer interrupt bit, *TMRn_INT.irq*, is set on the rising edge of PCLK. An interrupt is generated if enabled.

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

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

### 15.7.2 *PWM Mode Configuration*

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

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 011b to select PWM mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. Configure the timer pin:
5. Configure the pin as a timer input and configure the electrical characteristics as needed.
6. Set *TMRn_CN.tpol* to match the desired initial (inactive) state.
   a. Set *TMRn_CN.tpol* to select the initial logic level (high or low) and PWM transition state for the timer's output.
   b. Set *TMRn_CNT* to the starting count, typically 0x0000 0001. The initial *TMRn_CNT* value only effects the initial period in PWM mode with subsequent periods always setting *TMRn_CNT* to 0x0000 0001.
   c. Set the *TMRn_PWM* value to the transition period count.
7. Set the *TMRn_CMP* value for the PWM second transition period. Note: *TMRn_CMP* must be greater than the *TMRn_PWM* value.
8. Optionally, use the NVIC to enable the timer's interrupt and set the timer's interrupt priority.
9. Set *TMRn_CN.ten* to 1 to enable the timer and start the PWM.

Use *Equation 15-6*, *below*, to calculate the PWM period.

*Equation 15-6: Timer PWM Period*

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

If an initial starting value other than 0x0000 0001 is loaded into the *TMRn_CNT* register, use the One-Shot mode equation, *Equation 15-2*, to determine the initial PWM period.

If *TMRn_CN.tpol* is 0, the ratio of the PWM output high time to the total period is calculated using *Equation 15-7*, *below*.

*Equation 15-7: Timer PWM Output High Time Ratio with Polarity 0*

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

If *TMRn_CN.tpol* is set to 1, the ratio of the PWM output high time to the total period is calculated using *Equation 15-8*.

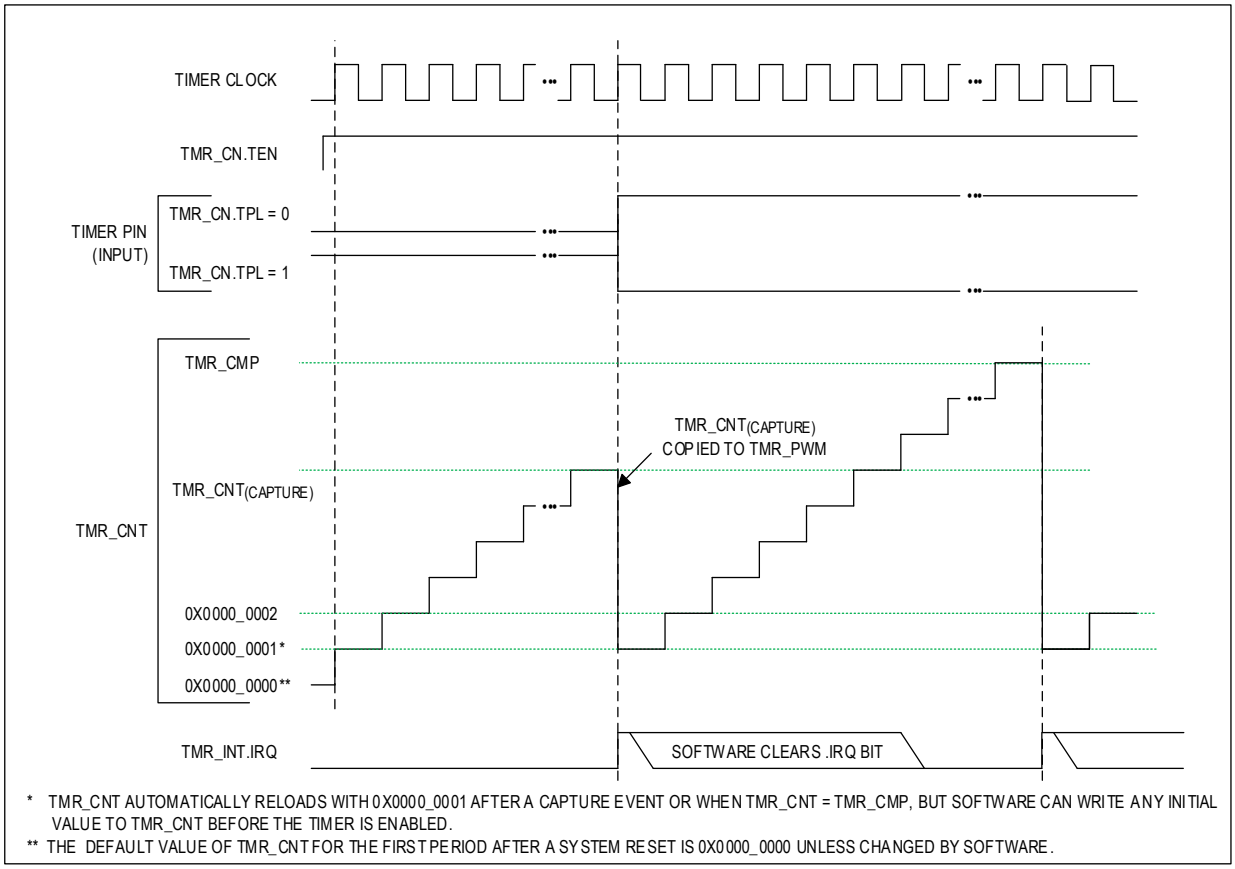
*Equation 15-8: Timer PWM Output High Time Ratio with Polarity 1*

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

## 15.8 Capture Mode (100b)

Capture mode is most often used to measure the time between events. The timer increments from an initial value until an edge transition occurs on the timer pin. This triggers the 'capture' event which copies *TMRn_CNT* to the *TMRn_PWM* register, resets *TMRn_CNT* to 1, and then continues incrementing the *TMRn_CNT* value. If the timer pin does transition before the *TMRn_CNT* = *TMRn_CMP* condition, a 'rollover' event occurs. When the 'rollover' event occurs, the timer resets *TMRn_CNT* to 1 and then continues incrementing the *TMRn_CNT* value. Either a 'capture' event or a 'rollover' event sets the timer interrupt flag indicating an event occured.

*Figure 15-4: Capture Mode Diagram*




### 15.8.1 *Capture Mode Timer Period*

Two timer period events are possible in Capture Mode:

- Capture Event
- Rollover Event

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

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

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

1. The value in *TMRn_CNT* is reset to 0x0000 00001. The timer remains enabled and continues incrementing.
2. The timer interrupt bit *TMRn_INT.irq* will be set. An interrupt is generated if enabled.

### 15.8.2 *Capture Mode Configuration*

Configure the timer for Capture mode by doing the following:

1. Disable the timer by setting *TMRn_CN.ten* to 0.
2. Select Counter mode by setting *TMRn_CN.tmode* to 010b.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. If using the timer pin:
   a. Configure the pin as a timer output and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired (inactive) state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write the initial value to *TMRn_CNT*. This effects only the first period; subsequent periods always begin with 0x0000 0001.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* = 1 to enable the timer.

Use *Equation 15-9*, *below*, The timer period is calculated using the following equation:

*Equation 15-9: Capture Mode Elapsed Time Calculation in Seconds*

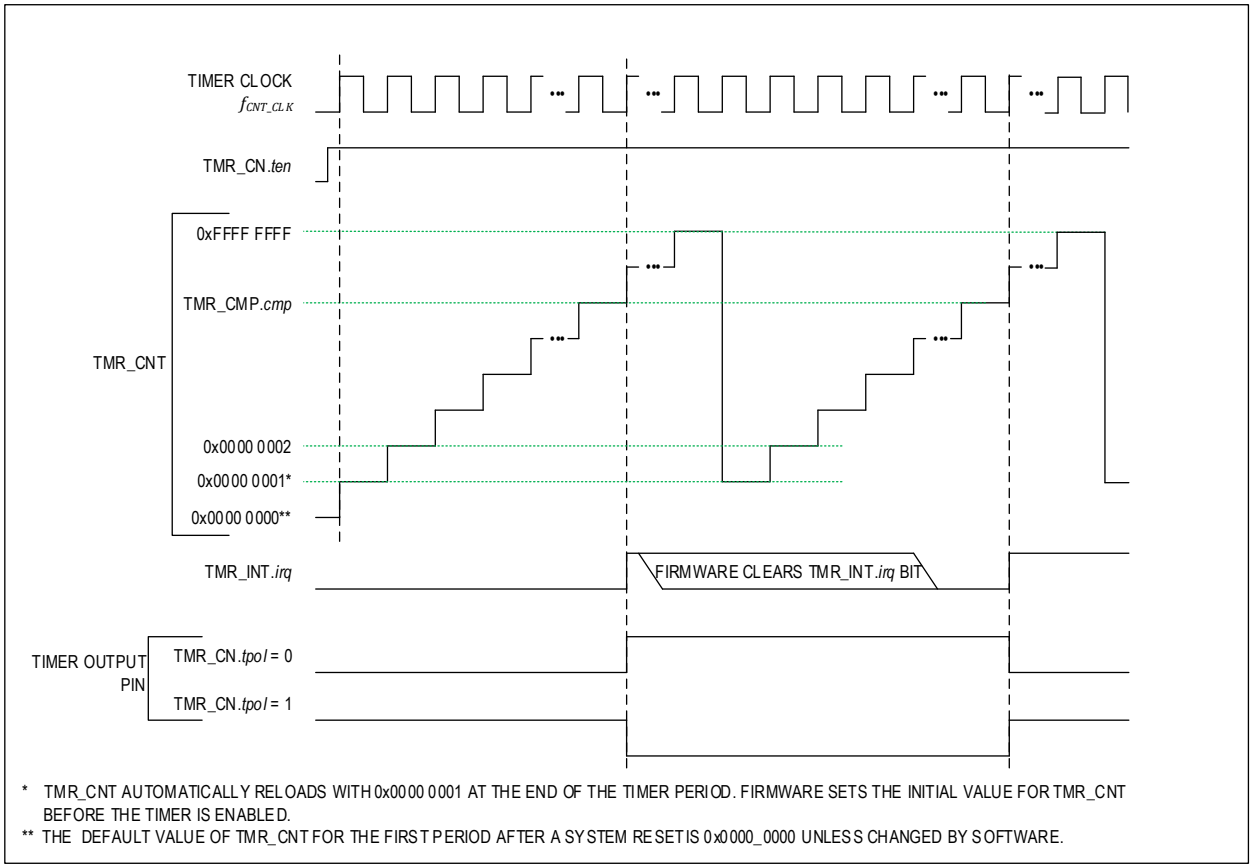
$$Capture\ elapsed\ time\ in\ seconds = \frac{TMR\_PWM - TMR\_CNT_{INITIAL\_VALUE}}{f_{CNT\_CLK}}$$

*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.*

## 15.9 Compare Mode (101b)

In Compare mode the timer peripheral increments continually, allowing the timer to be a programmable 32-bit programmable period timer. 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.

*Figure 15-5: Counter Mode Diagram*




### 15.9.1 *Compare Mode Timer Period*

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

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

1. The timer remains enabled and continues incrementing. Unlike other modes, *TMRn_CNT* is not reset to 0x0000 0001 at the end of the timer period.
2. If the timer output is enabled, then the timer pin toggles state (low to high or high to low).
3. The timer interrupt bit *TMRn_INT.irq* will be set. An interrupt is generated if enabled.

### 15.9.2 *Compare Mode Configuration*

Configure the timer for Compare mode by doing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 011b to select Compare mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. If using the timer pin:
   a. Configure the pin for the timer output alternate function and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired (inactive) state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write the initial value to *TMRn_CNT*. This effects only the first period as the counter increments continuously, rolling over to 0x0000 0000 and continuing.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* = 1 to enable the timer.

Use *Equation 15-10*, *below*, to calculate the Compare Mode timer period.

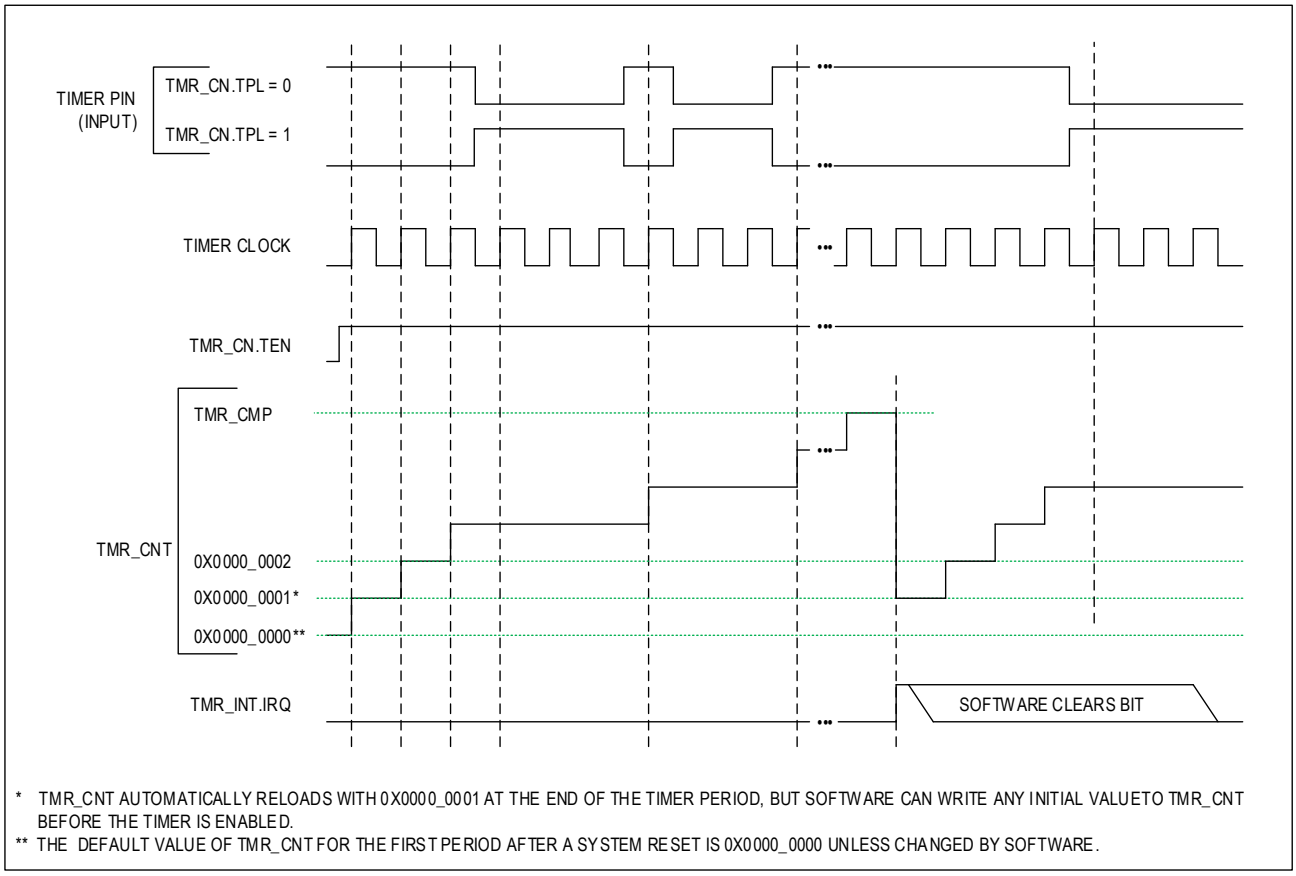
*Equation 15-10: Compare Mode Timer Period*

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

## 15.10 Gated Mode (110b)

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

*Figure 15-6: Gated Mode Diagram*




### 15.10.1 *Gated Mode Timer Period*

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

1. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing.
2. The timer interrupt bit *TMRn_INT.irq* will be set. An interrupt will be generated if enabled.

### 15.10.2 *Gated Mode Configuration*

Configure the timer for Gated mode by doing the following:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 110b to select Gated mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. Configure the timer pin:
   a. Configure the pin as a timer input and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to match the desired initial (inactive) state.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write an initial value to *TMRn_CNT*, if desired. This effects only the first period; subsequent timer periods always reset *TMRn_CNT* = 0x0000 0001.
7. Write the compare value to *TMRn_CMP*.
8. Set *TMRn_CN.ten* = 1 to enable the timer.

## 15.11 Capture/Compare Mode (111b)

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

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

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

### 15.11.1 *Capture/Compare Timer Period*

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

The timer peripheral automatically performs the following actions at the end of the timer period depending on what caused the period to end.

**Timer Pin Transition**

1. The value in *TMRn_CNT* is copied to *TMRn_PWM*.
2. *TMRn_CNT* is reset to 0x0000 0001. The timer remains enabled and continues incrementing.
3. The timer interrupt bit, *TMRn_INT.irq*, is set. If the timer's interrupt is enabled a Timer IRQ is generated automatically.

**Rollover**

4. The timer interrupt bit *TMRn_INT.irq* will be set.
5. An interrupt is generated if enabled.

### 15.11.2 *Capture/Compare Configuration*

Perform the following steps to configure the timer for Capture/Compare mode:

1. Set *TMRn_CN.ten* = 0 to disable the timer.
2. Set *TMRn_CN.tmode* to 0b111 to select Capture/Compare mode.
3. Set *TMRn_CN.pres3*:*TMRn_CN.pres* to set the prescaler that determines the timer frequency.
4. Configure the timer pin:
   a. Configure the pin as a timer input and configure the electrical characteristics as needed.
   b. Set *TMRn_CN.tpol* to select the positive edge (*TMRn_CN.tpol* = 0) or negative edge (*TMRn_CN.tpol* = 0) transition causes the capture event.
5. If using the timer interrupt, enable the interrupt and set the interrupt priority.
6. Write an initial value to *TMRn_CNT*, if desired. This effects only the first timer period; subsequent timer periods always reset *TMRn_CNT* to 1.
7. Enable the timer, set *TMRn_CN.ten* to 1. Counting starts after the first transition of the timer's input signal.

*Note: The first transition of the timer's input pin does not generate an interrupt.*

Calculate the elapsed time from start to the capture event using *Equation 15-11*, *below*.

*Equation 15-11: Capture Mode Elapsed Time*

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

## 15.12 Timer Registers

*Table 15-1* shows the address offset for each timer register from the Timer's Base Peripheral Address. All Timer instances use an identical register set. Register names for a specific instance are defined by appending the instance number to the peripheral name. For example, the Timer Count Register for Timer 0 is TMR0_CNT while the Timer Count Register for Timer 1 is TMR1_CNT. The MAX32650–MAX32652 include six timer instances, defined as TMR0, TMR1, TMR2, TMR3, TMR4 and TMR5.

Refer to *Table 2-1: APB Peripheral Base Address Map* for the Timer 0 (TMR0_) to Timer 5 (TMR5_) Base Peripheral Address.

*Table 15-1: Timer Register Offset, Names, Access and Descriptions*

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

## 15.13 Timer Register Details

*Table 15-2: Timer Count Registers*

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

*Table 15-3: Timer Compare Registers*

| Timer Compare Register | | | TMRn_CMP | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | compare | R/W | 0 | **Timer Compare Value**<br>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. Refer to the timer mode's detailed configuration section for *compare* usage and meaning. |

*Table 15-4: Timer PWM Register*

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

*Table 15-5: Timer Interrupt Registers*

| Timer Interrupt Register | | | TMRn_INT | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:1 | - | R | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 0 | irq | RW | 0 | **Timer Interrupt**<br>If set, this field indicates a timer interrupt condition occurred.<br>Writing any value to this bit clears the timer's interrupt.<br>0: Timer interrupt is not active.<br>1: Timer interrupt occurred. |

*Table 15-6: Timer Control Registers*

| Timer Control Register | | | TMRn_CN | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| | - | R | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 12 | pwmckbd | R/W | 1 | **PWM Output $\phi_{A'}$ Disable**<br>1: Disable PWM Output $\phi_{A'}$<br>0: Enable PWM Output $\phi_{A'}$ |
| 11 | nollpol | R/W | 0 | **PWM Output $\phi_{A'}$ Polarity Bit**<br>1: Output $\phi_{A'}$ inverted<br>0: Output $\phi_{A'}$ non-inverted |
| 10 | nolhpol | R/W | 0 | **PWM Output $\phi_A$ Polarity Bit**<br>1: Output $\phi_A$ inverted<br>0: Output $\phi_A$ non-inverted |
| 9 | pwmsync | R/W | 0 | **PWM Synchronization Mode**<br>1: PWM synchronization mode enabled<br>0: PWM synchronization mode disabled |
| 8 | pres3 | R/W | 0 | **Timer Prescale Select MSB**<br>See *TMRn_CN.pres* for details on this field's usage.. |
| 7 | ten | R/W | 0 | **Timer Enable**<br>1: Timer enabled<br>0: Timer disabled |
| 6 | tpol | R/W | 0 | **Timer Polarity**<br>Selects the polarity of the timer's input and output signal. This setting is not used if the GPIO Port Pin for the timer's alternate function is not set in the GPIO. The *tpol* field meaning is determined by the specific mode of the timer. Refer to the mode's detailed configuration section for *tpol* usage. |

| Timer Control Register | TMRn_CN | [0x0010] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 5:3 | pres | R/W | 0 | **Timer Prescaler Select**<br>Sets the timer's prescaler value. The prescaler divides the PCLK input to the timer and sets the timer's count clock, $f_{CNT\_CLK} = {PCLK\ (Hz)}/{prescaler}$. The timer's prescaler setting is a 4-bit value with pres3 as the most significant bit and pres as the three least significant bits. The table below shows the prescaler values based on *pres3*:*pres*. |

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

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

*Table 15-7: Timer Non-Overlapping Compare Registers*

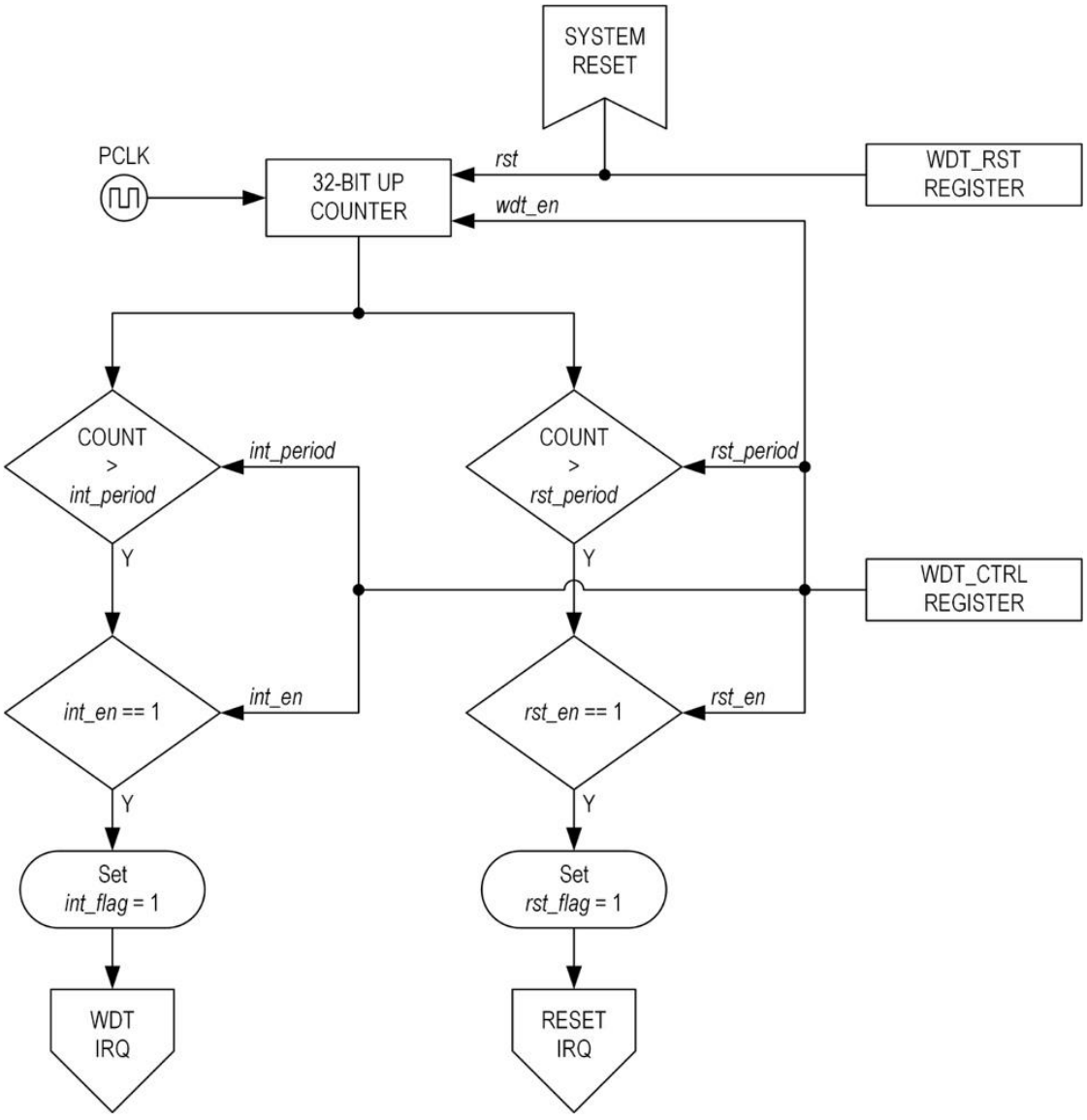
| Timer Non-Overlapping Compare Register | | | TMRn_NOLCMP | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15:8 | nolhcmp | R/W | 0 | **Non-Overlapping High Compare**<br>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 $\phi_A$. |
| 7:0 | nollcmp | R/W | 0 | **Non-Overlapping Low Compare**<br>The 8-bit timer count value of non-overlapping time between the falling edge of PWM output $\phi_A$ and next rising edge of PWM output $\phi_{A'}$. |

# 16 Watchdog Timer (WDT)

The watchdog timer protects against corrupt or unreliable software, power faults, and other system-level problems, which may place the microcontroller into an improper operating state. When the application is executing properly, application software periodically resets the watchdog counter. If the watchdog timer interrupt is enabled and the software does not reset the counter within the interrupt time period (*WDTn_CTRL.int_period*), the watchdog timer generates a watchdog timer interrupt. If the watchdog timer reset is enabled and the software does not reset the counter within the reset time period (*WDTn_CTRL.rst_period*), the watchdog timer generates a system reset.

*Figure 16-1* shows the block diagram of the watchdog timers.

*Figure 16-1: Watchdog Timer Block Diagram*



## 16.1 Features

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

## 16.2 Usage

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

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

## 16.3 Interrupt and Reset Period Timeout Configuration

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

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

The interrupt and reset period timeouts are calculated using *Equation 16-1* and *Equation 16-2* respectively, where $f_{PCLK} = {}^{f_{SYSCLK}}/_{2}$. *Table 16-1* shows example interrupt period timeout calculations for several *WDTn_CTRL.int_period* settings with the System Clock set as the 120MHz Relaxation Oscillator.

*Equation 16-1: Watchdog Timer Interrupt Period*

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

*Equation 16-2. Watchdog Timer Reset Period*

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

*Table 16-1: Watchdog Timer Interrupt Period*

| *WDTn_CTRL int_period* | $T_{INT\_PERIOD}$ (seconds) |
|---|---|
| 15 | 0.001 |
| 14 | 0.002 |
| 13 | 0.004 |
| 12 | 0.009 |

| *WDTn_CTRL int_period* | $T_{INT\_PERIOD}$ (seconds) |
|---|---|
| 11 | 0.018 |
| 10 | 0.035 |
| 9 | 0.070 |
| 8 | 0.140 |
| 7 | 0.280 |
| 6 | 0.560 |
| 5 | 1.12 |
| 4 | 2.24 |
| 3 | 4.47 |
| 2 | 8.95 |
| 1 | 17.9 |
| 0 | Disabled |

## 16.4 Enabling the Watchdog Timer

The watchdog timers are free running and require a protected sequence of writes to enable the watchdog timers to prevent an unintended reset during the enable process.

### 16.4.1 *Enable sequence*

1. Write *WDTn_RST.wdt_rst*: 0x000000A5
2. Write *WDTn_RST.wdt_rst*: 0x0000005A
3. Set *WDTn_CTRL.wdt_en* to 1

## 16.5 Disabling the Watchdog Timer

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

### 16.5.1 *Manual Disable*

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

### 16.5.2 *Automatic Disable*

A power-on-reset (POR) event automatically disables the watchdog timers by setting *WDTn_CTRL.wdt_en* to 0.

*Note: The watchdog timers remain enabled during all other types of reset.*

## 16.6 Resetting the Watchdog Timer

To prevent a watchdog interrupt or a watchdog reset or both, application software must write the reset sequence, shown below, to the *WDTn_RST* register prior to an interrupt or reset timeout occurring.

### 16.6.1 *Reset Sequence*

1. Write *WDTn_RST*: 0x000000A5
2. Write *WDTn_RST*: 0x0000005A

## 16.7 Detection of a Watchdog Reset Event

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

## 16.8 Watchdog Timer Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the Watchdog Timer's Base Peripheral Address.

*Table 16-2: Watchdog Timer Register Offsets, Names and Descriptions*

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

## 16.9 Watchdog Timer Register Details

*Table 16-3: Watchdog Timer Control Register*

| Watchdog Timer Control Register | | | WDTn_CTRL | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31 | rst_flag | R/W | See description | **Reset Flag**<br>If set a watchdog system reset occurred.<br>0: Watchdog did not cause reset event.<br>1: Watchdog reset occurred. |
| 30:12 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11 | rst_en | R/W | 0 | **Reset Enable**<br>Enable/Disable system reset if the *rst_period* expires. Only reset by power on reset.<br>0: Disabled<br>1: Enabled |
| 10 | int_en | R/W | 0 | **Interrupt Enable**<br>Enable or Disable the watchdog interrupt.<br>0: Disabled<br>1: Enabled |
| 9 | int_flag | R/W1C | 0 | **Interrupt Flag**<br>If set, the watchdog interrupt period has occurred.<br>0: IRQ not pending<br>1: Interrupt period expired. Generates and IRQ if *int_en*=1. |
| 8 | wdt_en | R/W | 0 | **Enable**<br>Enable or disable the watchdog timer. Only reset by a power on reset. To enable the watchdog timer, the following sequence of writes must be performed.<br>3) Write *WDTn_RST*: 0x0000 00A5<br>4) Write *WDTn_RST*: 0x0000 005A<br>5) Write *wdt_en*: 0x1<br>0: Disabled<br>1: Enabled<br>*Note: This field is only reset by a Power-On Reset. All other types of reset do not effect this field.* |

| Watchdog Timer Control Register | | WDTn_CTRL | | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:4 | rst_period | R/W | 0 | **Reset Period**<br>Sets the number of PCLK cycles until a system reset occurs if the watchdog timer is not reset.<br>0xF: $2^{16} \times t_{PCLK}$<br>0xE: $2^{17} \times t_{PCLK}$<br>0xD: $2^{18} \times t_{PCLK}$<br>0xC: $2^{19} \times t_{PCLK}$<br>0xB: $2^{20} \times t_{PCLK}$<br>0xA: $2^{21} \times t_{PCLK}$<br>0x9: $2^{22} \times t_{PCLK}$<br>0x8: $2^{23} \times t_{PCLK}$<br>0x7: $2^{24} \times t_{PCLK}$<br>0x6: $2^{25} \times t_{PCLK}$<br>0x5: $2^{26} \times t_{PCLK}$<br>0x4: $2^{27} \times t_{PCLK}$<br>0x3: $2^{28} \times t_{PCLK}$<br>0x2: $2^{29} \times t_{PCLK}$<br>0x1: $2^{30} \times t_{PCLK}$<br>0x0: $2^{31} \times t_{PCLK}$ |
| 3:0 | int_period | R/W | 0 | **Interrupt Period**<br>Sets the number of PCLK cycles until a watchdog timer interrupt is generated.<br>0xF: $2^{16} \times t_{PCLK}$<br>0xE: $2^{17} \times t_{PCLK}$<br>0xD: $2^{18} \times t_{PCLK}$<br>0xC: $2^{19} \times t_{PCLK}$<br>0xB: $2^{20} \times t_{PCLK}$<br>0xA: $2^{21} \times t_{PCLK}$<br>0x9: $2^{22} \times t_{PCLK}$<br>0x8: $2^{23} \times t_{PCLK}$<br>0x7: $2^{24} \times t_{PCLK}$<br>0x6: $2^{25} \times t_{PCLK}$<br>0x5: $2^{26} \times t_{PCLK}$<br>0x4: $2^{27} \times t_{PCLK}$<br>0x3: $2^{28} \times t_{PCLK}$<br>0x2: $2^{29} \times t_{PCLK}$<br>0x1: $2^{30} \times t_{PCLK}$<br>0x0: $2^{31} \times t_{PCLK}$ |

*Table 16-4: Watchdog Timer Reset Register*

| Watchdog Timer Reset Register | | WDTn_RST | | [0x0004] |
|---|---|---|---|---|
| **Register Field** | **Bits** | **Access** | **Reset** | **Description** |
| - | 31:8 | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| Watchdog Timer Reset Register | | WDTn_RST | | [0x0004] |
|---|---|---|---|---|
| **Register Field** | **Bits** | **Access** | **Reset** | **Description** |
| wdt_rst | 7:0 | R/W | 0 | **Reset Register**<br>Writing the watchdog counter reset sequence to this register resets the watchdog counter. The following is the required reset sequence to reset the watchdog and prevent a watchdog timer interrupt or watchdog system reset.<br>• Write *WDTn_RST*: 0x000000A5<br>• Write *WDTn_RST*: 0x0000005A |

# 17 1-Wire Master

The MAX32650–MAX32652 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.

## 17.1 Features

- Flexible, 1-Wire timing generation (required 1 MHz 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 simplifying 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 conditions interrupts.

For more information about the Maxim 1-Wire protocol and supporting devices, refer to the following resources:

- AN937*: The Book of iButton® Standards*
  - *http://www.maximintegrated.com/en/app-notes/index.mvp/id/937*
- *AN1796: Overview of 1-Wire Technology and Its Use*
  - *http://www.maximintegrated.com/en/app-notes/index.mvp/id/1796*
- *AN187: 1-Wire Search Algorithm*
  - *http://www.maximintegrated.com/en/app-notes/index.mvp/id/187*

## 17.2 Pins and Configuration

The MAX32650–MAX32652 OWM pin mapping for the 140-pin WLP, the 144-pin TQFP and 96-WLP are shown in *Table 17-1*.

*Table 17-1: OWM Pin to Alternate Function Mapping*

| MAX32650–MAX32652 Alternate Function | MAX32650–MAX32652 Alternate Function Number | MAX32650–MAX32652 144-TQFP Pin Name | Direction | Signal Description |
|---|---|---|---|---|
| OWM_PUPEN | AF1 | P1.30 | O | Pull-Up Enable Output |
| OWM_IO | AF1 | P1.31 | I/O | 1-Wire I/O |

### 17.2.1 Pin Configuration

Perform the following steps to configure the GPIO for OWM peripheral usage:

1. Enable the alternate function mode for pins P1.30 and P1.31 by setting GPIO1_EN[30:31] to 0.
2. Set alternate function 1 (AF1) by setting GPIO1_AF_SEL[30:31] to 0.

### 17.2.2 I/O

The IO signal is a bidirectional I/O that is used to directly drive the external 1-Wire bus. As described in the 1-Wire interface specification, 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.

### 17.2.3 Pullup Enable

The pullup enable (PUE) 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.

## 17.3 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 17-1, below, where $f_{PCLK} = {f_{SYSCLK}}/{2}$.

*Equation 17-1: OWM 1MHz Clock Frequency*

$$f_{owmclk} = 1MHz = \frac{f_{PCLK}}{OWM\_CLK\_DIV\_1US.divisor}$$

If the system clock is set to 120MHz, $f_{PCLK} = 60\text{MHz}$, the *OWM_CLK_DIV_1US.divisor* field should be set to 60 as shown in Equation 17-2, below.

*Equation 17-2: OWM Clock Divisor for $f_{SYSCLK} = 120MHz$*

$$OWM\_CLK\_DIV\_1US.divisor = \frac{60MHz}{1MHz} = 60$$

## 17.4 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.

### 17.4.1 Networking Layers

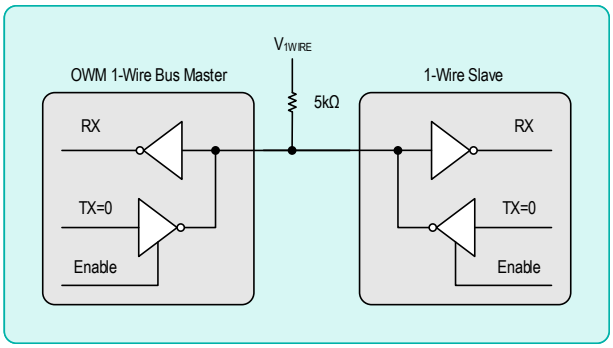
In the *Book of iButton Standards*, the 1-Wire communication protocol is described in terms of the standard ISO Open System Interconnection (OSI) layered network model. Network layers that apply to this description are the Physical, Link, Network, and Transport layers. The Presentation layer would correspond to higher-level application software functions (such as library layers) that implement communication protocols using the 1-Wire layers as a foundation.

### 17.4.2 Bus Interface (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 17-1 and avoids bus contention if more than one device attempts to drive the 1-Wire bus at the same time.

*Figure 17-1: 1-Wire Signal Interface*



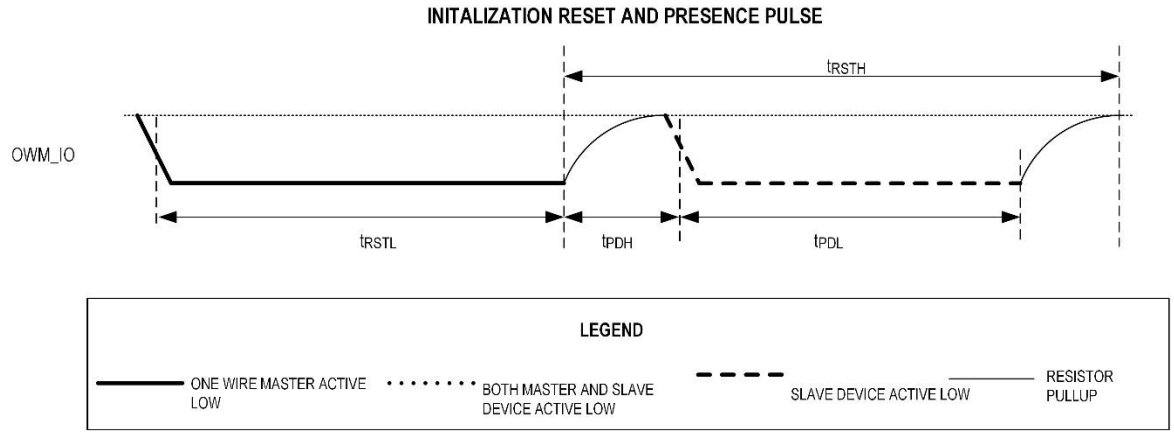

### 17.4.3 Reset, Presence Detect, and Data Transfer (Link Layer)

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 iButtons 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 MAX32650–MAX32652.

The OWM begins each communication sequence by sending a reset pulse as shown in Figure 17-2. 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 17-2: 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 no 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.

## 17.5 Read and Write Time Slots

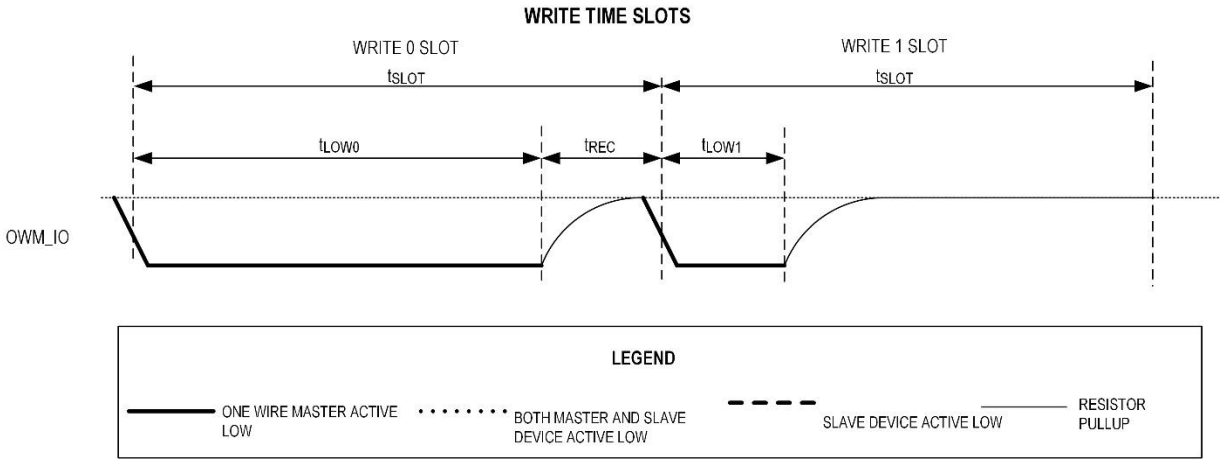
### 17.5.1 *OWM Write Time Slot*

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 17-3*, the time slots for writing a zero bit and writing a one 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 17-3: 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.

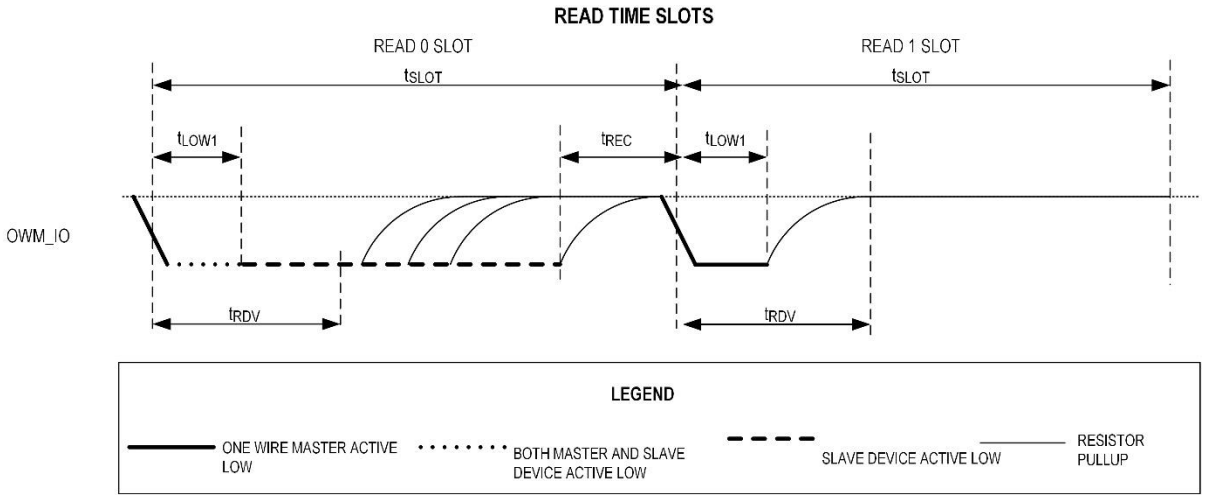
### 17.5.2 *OWM Read Time Slot*

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 17-3*. 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 17-4* 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 17-4: 1-Wire Read Tme Slot*



## 17.6 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.

### 17.6.1 *ROM Commands (Network Layer)*

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 17-2*, *below*.

*Table 17-2: 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.

*Figure 17-5* 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. Table 17-3 provides a detailed description of each of the ROM ID fields.

*Figure 17-5: 1-Wire ROM ID FIelds*

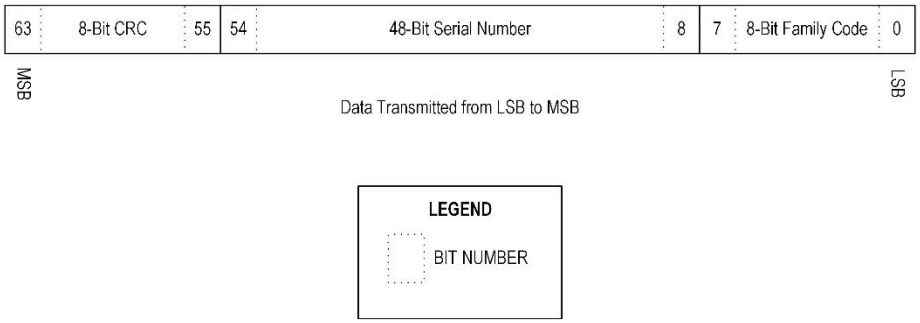


*Table 17-3: 1-Wire Slave Device ROM ID Field*

| 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 has to 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.

### 17.6.2 *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.

### 17.6.3 *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 proceed 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.

### 17.6.4 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.

### 17.6.5 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.

### 17.6.6 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 four-bit processing stage is complete, the return value loaded into *OWM_DATA.tx_rx* consists of eight 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 two bits read were both zero), or that no slaves responded (the two bits read were both 1). If the discrepancy bit is set to 0, then the two 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.

### 17.6.7 *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.

## 17.7 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

### 17.7.1 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.

### 17.7.2 1-Wire Data Writes

#### *17.7.2.1 Writing a Single Bit to the 1-Wire Bus*

To transmit a single bit of data on 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 of 8 bits.
2. Write the data bit to be transmitted (0 or 1) to *OWM_DATA.tx_rx*. 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 transmission 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.

#### *17.7.2.2 Writing an 8-Bit Byte to the 1-Wire Bus*

To transmit an 8-bit byte of data on the 1-Wire bus, complete the following steps:

1. Set *OWM_CFG.single_bit_mode* to 0. This setting causes the OWM to transmit/receive data 8 bits at a time. The least significant bit (LSB) of the data is always transmitted first.
2. Write the 8-bit value to be transmitted to *OWM_DATA.tx_rx*. Writing to the *OWM_DATA* register initiates the transmission of the 8-bit value on the 1-Wire bus.
3. 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.

## 17.8 1-Wire Data Reads

### 17.8.1 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, a 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.

### 17.8.2 *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).

1. Set *OWM_CFG.single_bit_mode* to 0. This setting causes the OWM to transmit/receive in the default 8-bit mode.
2. Write *OWM_DATA.tx_rx* to 0x0FFh.
3. 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.
4. 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, a OWM module interrupt occurs.
5. 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.

## 17.9 OWM Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the OWM Base Peripheral Address.

*Table 17-4: OWM Register Offsets, Names, Access and Descriptions*

| Offset | Register | Access | Description |
|---|---|---|---|
| [0x0000] | *OWM_CFG* | R/W | OWM Configuration Register |
| [0x0004] | *OWM_CLK_DIV_1US* | R/W | OWM Clock Divisor Register |
| [0x0008] | *OWM_CTRL_STAT* | See Description | OWM Control/Status Register |
| [0x000C] | *OWM_DATA* | R/W | OWM Data Buffer Register |
| [0x0010] | *OWM_INTFL* | R/W1C | OWM Interrupt Flags Register |
| [0x0014] | *OWM_INTEN* | R/W | OWM Interrupt Enable Register |

## 17.10 OWM Register Details

*Table 17-5: OWM Configuration Register*

| OWM Configuration Register | | OWM_CFG | | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:8 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7 | int_pullup_enable | R/W | 0 | **Internal Pullup Enable**<br>Set this field to enable the internal pullup resistor.<br>0: Internal pullup disabled<br>1: Internal pullup enabled |
| 6 | overdrive | R/W | 0 | **Overdrive Enable**<br>Set this field to 1 to enable overdrive mode for 1-Wire communications. Clearing this field sets 1-Wire communications to standard speed.<br>0: Overdrive mode disabled, standard speed mode.<br>1: Overdrive mode enabled. |
| 5 | single_bit_mode | R/W | 0 | **Bit Mode Enable**<br>When set to 1, only a single bit at a time is transmitted and received (LSB of *OWM_DATA*) rather than the whole byte.<br>0: Byte mode enabled, single bit mode disabled.<br>1: Single bit mode enabled, byte mode disabled. |
| 4 | ext_pullup_enable | R/W | 0 | **External Pullup Enable**<br>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.<br>0: External pullup pin is not driven to high.<br>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**<br>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**<br>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.<br>0: Bit-bang mode disabled.<br>1: Bit-bang mode enabled. |
| 1 | force_pres_det | R/W | 1 | **Presence Detect Force**<br>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.<br>0: OWM_IO pin floats during presence detection portion of 1-Wire reset.<br>1: OWM_IO pin is driven low during presence detection portion of 1-Wire reset. |

| OWM Configuration Register | | OWM_CFG | | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | long_line_mode | R/W | 0 | **Long Line Mode Enable**<br>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.<br>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.<br>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.<br>0: Standard operation for lines less than 40 meters.<br>1: Long line mode enabled, see description above. |

*Table 17-6: OWM Clock Divisor Register*

| OWM Clock Divisor Register | | OWM_CLK_DIV_1US | | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:8 | - | R | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7:0 | divisor | R/W | 0 | **OWM Clock Divisor**<br>Divisor for the OWM peripheral clock. The target is to achieve a 1MHz clock. Refer to the clock configuration section for details.<br>0x00: OWM clock disabled.<br>0x01: $f_{owmclk} = \frac{f_{PCLK}}{1}$<br>0x02: $f_{owmclk} = \frac{f_{PCLK}}{2}$<br>....<br>0xFF: $f_{owmclk} = \frac{f_{PCLK}}{255}$ |

*Table 17-7: OWM Control/Status Register*

| OWM Control/Status Register | | OWM_CTRL_STAT | | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:5 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7 | presence_detect | RO | 0 | **Presence Detect Flag**<br>Set to 1 when a presence pulse is detected from one or more slaves during the 1-Wire reset sequence.<br>0: No presence detect pulse during previous 1-Wire reset sequence.<br>1: Presence detect pulse on bus during previous 1-Wire reset sequence. |
| 4 | od_spec_mode | RO | 0 | **Overdrive Spec Mode**<br>Returns the version of the overdrive spec. |
| 3 | ow_input | RO | - | **OWM_IN State**<br>Returns the current logic level on the OWM_IO pin.<br>0: OWM_IO pin is low.<br>1: OWM_IO pin is high. |

| OWM Control/Status Register | | OWM_CTRL_STAT | | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 2 | bit_bang_oe | R/W | 0 | **OWM Bit Bang Output**<br>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.<br>0: OWM_IO pin floating.<br>1: Drive OWM_IO pin to low state. |
| 1 | sra_mode | R/W | 0 | **Search ROM Accelerator Enable**<br>Enable Search ROM Accelerator mode. This mode is used to identify slaves and their addresses that are attached to the 1-Wire bus.<br>0: Search ROM accelerator mode disabled.<br>1: Search ROM accelerator mode enabled. |
| 0 | start_ow_reset | R/W | 0 | **Start 1-Wire Reset Pulse**<br>Write 1 to start a 1-Wire reset sequence. Automatically cleared by the OWM hardware when the reset sequence is complete.<br>0: 1-Wire reset sequence complete or inactive.<br>1: Start a 1-Wire reset sequence. |

*Table 17-8: OWM Data Register*

| OWM Data Register | | OWM_DATA | | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:8 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7:0 | tx_rx | R/W | 0 | **OWM Data Field**<br>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 17-9: OWM Interrupt Flag Register*

| OWM Interrupt Flags Register | | OWM_INTFL | | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:5 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 4 | line_low | R/W1C | 0 | **Line Low Flag**<br>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**<br>The OWM hardware detected a short on the OWM_IO pin. Write 1 to clear this flag. |
| 2 | rx_data_ready | R/W1C | 0 | **RX Data Ready**<br>Data received from the 1-Wire bus and is available in the *OWM_DATA.tx_rx* field. Write 1 to clear this flag.<br>0: RX data not available.<br>1: Data received and is available in the *OWM_DATA.tx_rx* field. |

| OWM Interrupt Flags Register | | OWM_INTFL | | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 1 | tx_data_empty | R/W1C | 0 | **TX Empty**<br>The OWM hardware automatically sets this interrupt flag when the data transmit is complete. Write 1 to clear this flag.<br>0: Either no data was sent or the data in the *OWM_DATA*.*tx_rx* field has not completed transmission.<br>1: Data in the *OWM_DATA*.*tx_rx* field was transmitted. |
| 0 | ow_reset_done | R/W1C | 0 | **Reset Complete**<br>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.<br>0: 1-Wire reset sequence not complete or bus idle.<br>1: 1-Wire reset sequence complete. |

*Table 17-10: OWM Interrupt Enable Register*

| OWM Interrupt Enable Register | | OWM_INTEN | | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:5 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 4 | line_low | R/W | 0 | **Line Low Interrupt Enable**<br>I/O pin low detected interrupt enable.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 3 | line_short | R/W | 0 | **Line Short Interrupt Enable**<br>I/O pin short detected interrupt enable.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 2 | rx_data_ready | R/W | 0 | **Receive Data Ready Interrupt Enable**<br>RX data ready interrupt enable.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 1 | tx_data_empty | R/W | 0 | **Transmit Data Empty Interrupt Enable**<br>TX data empty interrupt enable.<br>0: Interrupt disabled<br>1: Interrupt enabled |
| 0 | ow_reset_done | R/W | 0 | **1-Wire Reset Sequence Complete Interrupt Enable**<br>1-Wire reset sequence completed.<br>0: Interrupt disabled<br>1: Interrupt enabled |

# 18 USBHS 2.0 Hi-Speed Host Interface with PHY

The microcontroller includes one Universal Serial Bus (USB) Host communications peripheral with a USB physical interface (PHY). The USB Host is USB 2.0 Hi-Speed (USBHS) compliant, capable of transfers at 480Mbits/sec. It supports Host mode with 12 USB buffers called Endpoints.

The following features are supported:

- USB Device Mode
- USB 2.0 Full Speed (FS) 12Mbps transfers
- USB 2.0 Hi-Speed (HS) 480Mbps transfers
- Bulk transfers
- Isochronous transfers
- 11 Endpoints plus Endpoint 0, each with dedicated FIFOs
- Packet splitting and combining
- High bandwidth IN and OUT Isochronous endpoints

Each endpoint has an associated FIFO with the following sizes:

- Endpoint 0 FIFO: 64 bytes deep
- Endpoints 1 through 7 FIFOs: 512 bytes deep
- Endpoints 8 and 9 FIFOs: 2048 bytes deep
- Endpoints 10 and 11 FIFOs: 4096 bytes deep

Supported interrupts include:

- Interrupts for each IN endpoint from Endpoint 0 to Endpoint 11
- Interrupts for each OUT endpoint from Endpoint 1 to Endpoint 11
- Start of Frame (SOF)
- RESET bus state
- RESUME bus state
- SUSPEND Mode bus state
- STALL sent
- Control byte received
- Control transfer ended early
- Packet transmitted
- Packet received
- Data underrun
- Data overrun
- Invalid token received
- Empty data packet sent

This chapter includes a simplified description of USB bus states. Refer to the USB specification for a complete description of USB operation.

The USB Device hardware behavior is controlled by the internal Serial Interface Engine (SIE). The SIE is a small control processor that manages the USB port's behavior. When referring to behavior of the USB hardware, it is the SIE doing the work.

## 18.1 USBHS Bus Signals

A USB cable connects a USB Host, which controls the transfer, and a USB Device, which is controlled by the Host. The USBHS Peripheral is a USB Device. A USB cable has four conductors (three hardware signals plus ground). These signals can be duplicated more than once in a physical USB cable. The signals in a USB cable are as follows:

- **D+ (DPLUS):** Positive line of the differential data pair.
- **D- (DMINUS):** Negative line of the differential data pair.
- **VBUS:** Bus voltage supplied by Host.
- **Ground**

When a USB Device is attached to a USB Host, the USB Host identifies the speed of the USB Device by the presence of a pull-up resistor on either D+ or D-. The Host then begins the Enumeration sequence. The Enumeration sequence allows the Host to identify the type and characteristics of the Device attached to the Host so that it can load the proper drivers for the Device. Enumeration always uses Endpoint 0. The Host requests and reads from the Device the contents of the USB Endpoint Descriptor Table that tells the Host everything it needs to know about the capabilities of the USB Device. The Host then assigns the USB Device an address, which firmware writes to the *USBHS_FADDR.faddr* bit field.

*Table 18-1* shows the USB Bus states seen by the Host indicated by the differential pair.

*Table 18-1: USB Bus States indicated by the differential pair (D+, D-)*

| Bus State | Condition | D+ | D- | Notes |
|---|---|---|---|---|
| Differential 1 | Host or Device is driving the bus | Hi | Lo | |
| Differential 0 | Host or Device is driving the bus | Lo | Hi | |
| Single Ended Zero (SE0) | Cable Detached | Lo | Lo | No Device plugged in |
| Single Ended One (SE1) | Illegal State | Hi | Hi | Illegal state. This state should never occur on a properly configured USB bus |
| IDLE State, Full Speed | No Host or Device is driving the bus | Hi | Lo | USB Device is Full Speed. No activity on bus |
| IDLE State, Low Speed | No Host or Device is driving the bus | Lo | Hi | USB Device is Low Speed. No activity on bus |
| DISCONNECT | Device wants to disconnect from Host | Lo | Lo | Held for 2.5µsec or longer |
| RESET | Host is initiating communication with a Device | Lo | Lo | Held for 10msec or longer |

USB communication is based on the above basic conditions, which are used to generate the following states:

- **Data J State** – Same as IDLE state but bus is actively driven by either the Host or the Device.
- **Data K State** – Opposite of J state. Bus is actively driven by the Host or the Device.
- **RESUME** – Data K State. Tells Device to exit SUSPEND mode.
- **START OF PACKET (SOP)** – Bus switches from IDLE to K state.
- **END OF PACKET (EOP)** – SE0 for two bit periods, then J state for one bit period.
- **KEEP ALIVE Signal** – EOP sent every 1 millisecond.

## 18.2 USBHS Device Endpoints

Each USB Device supports one or more Endpoints. Endpoints serve as a source or destination for data and are supported by memory buffers. This USB controller supports 12 Endpoints, each with its own set of descriptors and data buffers. These Endpoints are referenced as Endpoint 0 through Endpoint 11.

Endpoints support four different types of data transfers:

- **Control Endpoint** – Always uses Endpoint 0, this endpoint is used by the USB Host to setup the USB Device for the USB Device to receive operational status from the USB Host.
- **Interrupt Endpoints** – Used to send and receive non-time critical data to and from a USB Device. An application example is a USB keyboard or a USB mouse.
- **Isochronous Endpoints** – Used to send or receive real-time data that requires a guaranteed bandwidth to or from a Host. An application example is a video camera used for real-time video streaming.
- **Bulk Endpoints** – Used to send and receive high-volume data that does not require real-time processing. An application example is a USB flash drive which transfers high volume data.

The USBHS supports Control, Interrupt, Bulk, and Isochronous Endpoints. Per the USB specification, Endpoint 0 is dedicated to Control Transfers only.

Endpoint directions are always defined from a USB Host to a USB Device. OUT Endpoint 1 refers to a Device Endpoint holding data sent out from a USB Host to a USB Device. IN Endpoint 2 refers to a Device Endpoint holding data sent from a USB Device to a USB Host.

Each USBHS Data Endpoint supports the following features:

- Single or double buffered
- Programmable and flexible interrupts
- Ability to send a STALL packet to the Host to indicate an error with the data
- Ability to automatically send an ACK packet to the Host to acknowledge a successful data transfer
- Ability to send a NYET (Not Yet) packet to the Host for Hi-Speed transfers to indicate it is not yet ready to receive more data
- Configurable response to Status Stage of Control transfer

## 18.3 USBHS Reset

When a RESET state is detected on the bus, the USBHS performs the following actions:

- Sets *USBHS_FADDR.addr* = 0
- Sets *USBHS_INDEX* = 0
- All endpoint FIFOs are flushed
- All control and status registers are reset
- The USB PHY is electrically disconnected from the bus
- All endpoint interrupts are enabled
- Generates a USB Reset IRQ

## 18.4 USBHS SUSPEND and RESUME States

When the USBHS sees no activity on the bus for 3ms, and if Suspend Mode is allowed (*USBHS_POWER.suspendm* = 1), then the USBHS goes into low-power SUSPEND mode. The Suspend status flag is set (*USBHS_INTSIGFL.suspend* = 1), and a Suspend interrupt is generated if enabled (*USBHS_INTSIGEN.suspend* = 1).

Firmware can exit Suspend mode by sending a RESUME state on the bus by setting the bit field *USBHS_POWER.resume* = 1. Firmware must leave this bit set between 2ms and 15ms with 10ms being the optimal time after which firmware must clear the resume bit field.

If the external Host generates a RESUME state on the bus, a RESUME interrupt is generated. A RESUME interrupt is not generated if the RESUME state on the bus is caused by firmware setting the *USBHS_POWER.resume* bit.

## 18.5 Packet Size

For all transfers the packet size is specified in the *USBHS_INMAXP* register for IN Endpoints, and the *USBHS_OUTMAXP* register for OUT Endpoints. These registers specify the size of the entire transactions.

## 18.6 Endpoint 0 Control Transactions

Endpoint 0 (EP0) is the main control endpoint and handles all USB Standard Device Requests for control transfers. There are three types of Standard Device Requests:

1. In Zero Data Requests, all the information for the request is included in an 8-byte command.
2. In Write Requests, the command from the USBHS is followed by additional data.
3. In Read Requests, the USBHS is communicating with a USB Device that is required to send data back to the Host.

### 18.6.1 Endpoint 0 Error Handling

The USBHS can detect and generate interrupts for control transfers errors. It sends a STALL packet on the bus and generates an interrupt if the incorrect amount of data is transferred over the bus. This can happen under the following conditions:

1. The Host sends more data during the OUT Data phase of a write request than the amount specified in the command. This condition is detected when the Host sends an OUT token after the Data End bit *USBHS_CSR0.dataend* is set by firmware.
2. The Host requests more data during the IN Data phase of a read request than the amount specified in the command. This condition is detected when the Host sends an IN token after the DataEnd bit *USBHS_CSR0.dataend* is set by firmware.
3. The Host sends more data in an OUT data packet than the amount specified in the *USBHS_OUTMAXP* register.
4. The Host sends a non-zero length DATA1 packet during the STATUS phase of a read request.

An error occurs if a control transaction ends prematurely. This can happen if the USB Host enters the STATUS phase before all data has been transferred. This can also occur if a USB Host transmits a new SETUP packet before finishing the current control transaction. In both cases, the *USBHS_CSR0.setupend* bit is set, which generates an Endpoint 0 interrupt.

If the *USBHS_CSR0.outpktrdy* bit is set, this indicates that the Host has sent another SETUP packet. Firmware should then process the command in that packet.

## 18.7 Bulk Endpoints Operation and Options

### 18.7.1 Bulk IN Endpoints

A Bulk IN endpoint is used to transfer high-volume data that does not require real-time processing. Five features are available for use with a Bulk IN endpoint as shown in *Table 18-2*, *below*.

*Table 18-2: USB Bulk IN Endpoints Options*

| Bulk IN Endpoint Option | Description |
|---|---|
| Double Packet Buffering | When the value written to the *USBHS_INMAXP* register is less than or equal to half the size of the FIFO allocated to the endpoint, and double packet buffering is allowed (*USBHS_INCSRU.dpktbufdis* = 0), double packet buffering is enabled. This allows up to two packets to be stored in the FIFO for transmission to the Host. |
| DMA Transfers | If the DMA is enabled for a Bulk IN endpoint (*USBHS_INCSRU.dmareqenab* = 1), a DMA request is generated whenever the endpoint can accept another packet in its FIFO. The DMA request is terminated when the entire packet is loaded into the DMA or when the *USBHS_INCSRL.inpktrdy* bit field is set indicating a packet in the FIFO was transmitted.<br>For Bulk IN endpoints, the DMA is set to DMA Request Mode 1 where a DMA request is generated each time a packet is received. The DMA completes burst transfers based on the maximum packet size for the endpoint: 512 bytes for Hi-Speed and 64 bytes for full speed. DMA burst transfers continue until the entire data block is transferred. |
| AutoSet | When the AutoSet feature is enabled (*USBHS_INCSRU.autoset* = 1) for a Bulk IN endpoint, the In Packet Ready bit field *USBHS_INCSRL.inpktrdy* is automatically set when a packet of *USBHS_INMAXP* bytes is loaded into the FIFO. |
| Automatic Packet Splitting | For some USB transfers, it might be necessary to write larger amounts of data to an endpoint than you can transfer in a single USB operation. For these transfers, the USBHS supports split transactions where large data packets that are written to Bulk endpoints are split into multiple smaller packets. The necessary packet size information is set in the *USBHS_INCSRU* register. |
| Error Handling | A STALL packet is used to indicate that an endpoint has had an error. To shut down the Bulk IN endpoint transfer, set the *USBHS_INCSRL.sendstall* bit field. When the USBHS receives the next IN token, it then sends a STALL to the Host, sets the *USBHS_INCSRL.sentstall* bit field, and generates an interrupt. |

### 18.7.2 Bulk OUT Endpoints

A Bulk OUT endpoint is used to transfer non-periodic data from the Host to the function controller. Five optional features are available for use with a Bulk OUT endpoint.

| Bulk OUT Endpoint Option | Description |
|---|---|
| Double Packet Buffering | When the value written to the *USBHS_OUTMAXP* register is less than or equal to half the size of the FIFO allocated to the endpoint, and double packet buffering is allowed (*USBHS_OUTCSRU.dpktbufdis* = 0), double packet buffering is enabled. This allows storage of up to two packets in the FIFO for transmission to the Host. |
| DMA Transfers | DMA transfers for an OUT Endpoint depend on the DMA Request Mode selected with the *USBHS_OUTCSRU.dmareqmode* bit field.<br>In DMA Request Mode 0, a DMA request is generated when a data packet is available in the OUT Endpoint FIFO. The DMA request is terminated when the last byte of the data packet is read from the OUT FIFO, or when the *USBHS_OUTCSRL.outpktrdy* bit is cleared indicating the OUT FIFO is empty.<br>In DMA Request Mode 1, the DMA request line only goes high when the packet received is of the maximum packet size set in the *USBHS_OUTMAXP* register. If the packet received is of some other size, a DMA request is not generated, leaving the packet in the FIFO with the *USBHS_OUTCSRL.outpktrdy* bit still set. |
| AutoClear | When the AutoClear feature is enabled (*USBHS_OUTCSRU.autoclear* = 1), the *USBHS_OUTCSRL.outpktrdy* bit is automatically cleared when a packet of *USBHS_OUTMAXP* bytes is unloaded from the FIFO. |
| Automatic Packet Combining | For some USB transfers, it might be necessary to receive larger amounts of data from an endpoint than can be received in a single USB operation. For these transfers, the USBHS supports automatically combining packets received by split transactions, where large data packets received by Bulk endpoints had been split into multiple smaller packets. The necessary packet size information is set in the *USBHS_OUTMAXP* register. |
| Error Handling | A STALL packet is used to indicate that an endpoint has an error. To shut down the Bulk OUT endpoint transfer, set *USBHS_OUTCSRL.sendstall* = 1. When the USBHS receives the next packet, it then sends a STALL to the Host, sets the *USBHS_OUTCSRL.sentstall* bit, and generates an interrupt. |

## 18.8 Interrupt Endpoints

### 18.8.1 Interrupt IN Endpoints

Interrupt IN endpoints use the same protocols as Bulk IN endpoints and are used the same way. Although DMA can be used, there is little benefit as Interrupt endpoints transfer all their data in a single packet.

One feature supported by Interrupt IN Endpoints and not Bulk IN Endpoints is continuous toggle of the Data-Toggle bit. This feature is enabled by setting bit *USBHS_INCSRU.frcdatatog* = 1. When continuous toggling of the Data-Toggle bit is enabled, USBHS always considers a transmitted Interrupt packet as successfully sent and toggles Data-Toggle regardless of whether an ACK was received from the Host.

### 18.8.2 Interrupt OUT Endpoints

Interrupt OUT Endpoints use almost the same protocols as Bulk OUT endpoints and are used in the same way. Although DMA can be used, there is little benefit as Interrupt endpoints receive all their data in a single packet.

One feature not supported by Interrupt OUT endpoints that is supported by Bulk OUT endpoints is PING flow control. Because of this, Interrupt OUT endpoints cannot respond with NYET (Not Yet) handshakes. Instead, they can only respond with ACK, NAK, or STALL.

## 18.9 Isochronous Endpoints

### 18.9.1 *Isochronous IN Endpoints*

An Isochronous IN endpoint is used to transfer time-sensitive but loss-tolerant data from a USB Device to a USB Host. Five optional features are available for use with an Isochronous IN endpoint as shown in *Table 18-3*, *below*.

*Table 18-3: USB Isochronous IN Endpoint Options*

| Isochronous IN Endpoint Option | Description |
|---|---|
| Double Packet Buffering | When the value written to the *USBHS_INMAXP* register is less than or equal to half the size of the FIFO allocated to the endpoint, and double packet buffering is allowed (*USBHS_INCSRU.dpktbufdis* = 0), double packet buffering is enabled. This allows the storage of up to two packets in the FIFO for transmission to the Host. This is recommended to avoid data underrun. |
| DMA Transfers | If the DMA is enabled for an endpoint (*USBHS_INCSRU.dmareqenab* = 1), a DMA request is generated whenever the endpoint can accept another full packet in its FIFO. However, there is little benefit with Isochronous endpoints because the packets transferred are often not the maximum packet size. In this situation, the *USBHS_INCSRL.underrun* bit would have to be checked for under-run errors after each packet. |
| AutoSet | When the AutoSet feature is enabled (*USBHS_INCSRU.autoset* = 1), the *USBHS_INCSRL.inpktrdy* bit is automatically set when a packet of *USBHS_INMAXP* bytes is loaded into the FIFO. However, there is little benefit with Isochronous endpoints because the packets transferred are often not the maximum packet size. In this situation, the *USBHS_INCSRL.underrun* bit would have to be checked for under-run errors after each packet. |
| Error Handling | If an Isochronous IN Endpoint receives an IN Token while the IN FIFO is empty, it creates an under-run condition. This automatically sets the *USBHS_INCSRL.underrun* bit and results in the USBHS sending a null packet to the USB Host.<br>If firmware is loading the IN Endpoint FIFO one packet per frame, it should check that there is room in the IN FIFO by making sure the *USBHS_INCSRL.inpktrdy* bit is cleared before loading the next packet. If this bit is set, it indicates that a data packet is still in the FIFO and has not been sent, possibly from a corrupt IN Token. This error condition must be handled by firmware, for example, firmware might flush the unsent packet, or skip the current packet. |
| Error Handling – High Bandwidth Isochronous IN Endpoints only | High bandwidth Isochronous IN endpoints can transfer three 1024 byte packets in one payload. To the USB bus, it appears to be a single packet of 3072 bytes with a data transfer rate of up to 24Mbytes/sec. If a high-bandwidth isochronous data transfer is split into more than one packet but has not received enough IN tokens from the Host to send all the packets, an error condition exists. In this case, the Incomplete Split Transfer Error Status bit *USBHS_INCSRL.incomPTn*, is automatically set. This also automatically flushes the remainder of the packet from the IN FIFO. If a second packet is in the IN FIFO, it is not flushed. Because the packet was lost, the *USBHS_INCSRL.inpktrdy* bit is cleared. |

### 18.9.2 *Isochronous OUT Endpoints*

An Isochronous OUT endpoint is used to transfer time-sensitive but loss-tolerant data from the Host to the function controller. Five optional features are available for use with an Isochronous OUT endpoint as shown in *Table 18-4*, *below*.

*Table 18-4: USB Isochronous OUT Endpoint Options*

| Isochronous OUT Endpoint Option | Description |
|---|---|
| Double Packet Buffering | When the value written to the *USBHS_OUTMAXP* register is less than or equal to half the size of the FIFO allocated to the endpoint, and double packet buffering is allowed (*USBHS_OUTCSRU.dpktbufdis* = 0), double packet buffering is enabled. This allows the storage of up to two packets in the FIFO for transmission to the Host. Double packet buffering is recommended for isochronous OUT endpoints to avoid data over-run errors. |
| DMA Transfers | If the DMA is enabled for an endpoint, a DMA request is generated whenever the endpoint can accept another full packet in its FIFO. However, there is little benefit with Isochronous endpoints because the packets transferred are often not the maximum packet size. In this situation, the *USBHS_INCSRL.underrun* bit would need to be checked for under-run errors after each packet. |
| AutoClear | When the AutoClear feature is enabled (*USBHS_OUTCSRU.autoclear* = 1), the *USBHS_OUTCSRL.outpktrdy* bit is automatically cleared when a packet of *USBHS_OUTMAXP* bytes is unloaded from the FIFO. However, there is little benefit with Isochronous endpoints because the packets transferred are often not the maximum packet size. In this situation, the *USBHS_INCSRL.underrun* bit would need to be checked for under-run errors after each packet. |
| Error Handling | If a packet is received from a USB Host, but the OUT FIFO is full, it creates an overrun error condition. The register bit *USBHS_OUTCSRL.overrun* is automatically set. This error condition usually means that firmware is not unloading the OUT FIFO fast enough. This error condition must be handled by firmware.<br>If a received packet has a CRC error the packet is stored in the OUT FIFO, and both the *USBHS_OUTCSRL.dataerror* bit and the *USBHS_OUTCSRL.outpktrdy* bit are set. This error condition must be handled by firmware. |
| Error Handling – High Bandwidth Isochronous OUT Endpoints only | High-bandwidth Isochronous OUT endpoints can transfer three 1024-byte packets in one payload. To the USB bus, it appears to be a single packet of 3072 bytes. If a high-bandwidth isochronous data transmission is split into more than one packet, but if less than the expected number of packets is received by the OUT endpoint, an error condition exists. In this case, the Incomplete Isochronous Packet Received Error Status bit *USBHS_OUTCSRU.incomprx* is automatically set to indicate that the data received in the OUT FIFO is incomplete.<br>If a packet of the wrong data type is received during a high-bandwidth Isochronous OUT transaction, then the PID Error Status bit *USBHS_OUTCSRU.piderror* is automatically set. |

## 18.10 USBHS Device Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the USBHS Base Peripheral Address.

*Table 18-5: USBHS Device Register Offsets, Names, Access and Descriptions*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *USBHS_FADDR* | R/W | USBHS Device Address Register |
| [0x0001] | *USBHS_POWER* | R/W | USBHS Power Management |
| [0x0002] | *USBHS_INTRINFL* | RO | USBHS IN Endpoints Interrupt Status Flags |
| [0x0004] | *USBHS_INTROUTFL* | RO | USBHS OUT Endpoints Interrupt Status Flags |
| [0x0006] | *USBHS_INTRINEN* | R/W | USBHS IN Endpoints Interrupt Enable |
| [0x0008] | *USBHS_INTROUTEN* | R/W | USBHS OUT Endpoints Interrupt Enable |
| [0x000A] | *USBHS_INTSIGFL* | RO | USBHS Signaling Interrupt Status Flags |
| [0x000B] | *USBHS_INTSIGEN* | R/W | USBHS Signaling Interrupt Enable |

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x000C] | *USBHS_FRAME* | RO | USBHS Frame Number |
| [0x000E] | *USBHS_INDEX* | R/W | USBHS Endpoint and Status Register Index Select |
| [0x000F] | *USBHS_TESTMODE* | R/W | USBHS Test Mode Register |
| [0x0010] | *USBHS_INMAXP* | R/W | USBHS IN Endpoints Maximum Packet Size |
| [0x0012] | *USBHS_CSR0* | R/W | USBHS EP0 Control Status Register (*USBHS_INDEX* = 0) |
| [0x0012] | *USBHS_INCSRL* | R/W | USBHS IN Endpoints Lower Control Register (*USBHS_INDEX* != 0) |
| [0x0013] | *USBHS_INCSRU* | R/W | USBHS IN Endpoints Upper Control & Status Register |
| [0x0014] | *USBHS_OUTMAXP* | R/W | USBHS OUT Endpoints Maximum Packet Sizes |
| [0x0016] | *USBHS_OUTCSRL* | R/W | USBHS OUT Endpoints Lower Control Status Register |
| [0x0017] | *USBHS_OUTCSRU* | R/W | USBHS OUT Endpoints Upper Control Status Register |
| [0x0018] | *USBHS_COUNT0* | RO | USBHS Endpoint 0 IN FIFO Byte Count |
| [0x0018] | *USBHS_OUTCOUNT* | RO | USBHS Endpoints OUT FIFO Byte Count |
| [0x0020] | *USBHS_FIFO0* | R/W | USBHS FIFO for Endpoint 0 |
| [0x0024] | *USBHS_FIFO1* | R/W | USBHS FIFO for Endpoint 1 |
| [0x0028] | *USBHS_FIFO2* | R/W | USBHS FIFO for Endpoint 2 |
| [0x002C] | *USBHS_FIFO3* | R/W | USBHS FIFO for Endpoint 3 |
| [0x0030] | *USBHS_FIFO4* | R/W | USB HS FIFO for Endpoint 4 |
| [0x0034] | *USBHS_FIFO5* | R/W | USBHS FIFO for Endpoint 5 |
| [0x0038] | *USBHS_FIFO6* | R/W | USBHS FIFO for Endpoint 6 |
| [0x003C] | *USBHS_FIFO7* | R/W | USBHS FIFO for Endpoint 7 |
| [0x0040] | *USBHS_FIFO8* | R/W | USBHS FIFO for Endpoint 8 |
| [0x0044] | *USBHS_FIFO9* | R/W | USBHS FIFO for Endpoint 9 |
| [0x0048] | *USBHS_FIFO10* | R/W | USBHS FIFO for Endpoint 10 |
| [0x004C] | *USBHS_FIFO11* | R/W | USBHS FIFO for Endpoint 11 |
| [0x0078] | *USBHS_EPINFO* | RO | USBHS Endpoints Count Info |
| [0x0079] | *USBHS_RAMINFO* | RO | USBHS RAM and DMA Info |
| [0x007A] | *USBHS_SOFTRESET* | R/W1C | USBHS Soft Reset Control |
| [0x007B] | *USBHS_EARLYDMA* | R/W | USBHS Early DMA |
| [0x0080] | *USBHS_CTUCH* | R/W | USBHS Hi-Speed Chirp Timeout |
| [0x0082] | *USBHS_CTHSRTN* | R/W | USBHS Hi-Speed RESUME Delay |

## 18.11 USBHS Device Register Details

*Table 18-6: USBHS Device Address Register*

| USBHS Device Address Register | | | USBHS_FADDR | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7 | update | RO | 0 | **Read USBHS Device Update Status**<br>0: The Device address in the bit field addr is presently used.<br>1: New address written to the bit field addr is pending. New address takes effect at the end of the current transfer. |

| USBHS Device Address Register | | | USBHS_FADDR | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 6:0 | faddr | R/W | 0 | **USBHS Device Address**<br>This is the USB Device address specified by the external USB Host during the enumeration process. It must be written with the address value contained in the SET_ADDRESS Device request when received during a Control Transaction. |

*Table 18-7: USBHS Power Management Register*

| USBHS Power Management | | | USBHS_POWER | [0x0001] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7 | iso_update | R/W | 0 | **Isochronous Update**<br>1: If an SOF token is received from the Host and a packet is in the IN FIFO (*USBHS_INCSRL.inpktrdy*= 1), then send the packet. However, if an IN token is received from the Host before an SOF token, then send a zero-length data packet.<br>*Note: This register is only applicable in Isochronous Mode and ignored in all other modes.* |
| 6 | softconn | R/W | 0 | **Soft Connect/Disconnect PHY**<br>0: The USB D+/D- lines of the PHY are tri-stated, and this USB is electrically disconnected from the USB bus.<br>1: The USB D+/D- lines of the PHY are enabled. |
| 5 | hs_enable | R/W | 1 | **Enable Hi-Speed (HS) Mode**<br>0: USB remains in Full Speed Mode even if connected to a USB HS port.<br>1: USB always negotiates for HS mode on the bus. |
| 4 | hs_mode | RO | 0 | **Read Hi-Speed Mode Status Flag**<br>0 = USB in Full Speed Mode.<br>1 = USB in Hi-Speed Mode. |
| 3 | power_reset | RO | 0 | **Read RESET Mode Status Flag**<br>0 = Normal operation.<br>1 = RESET state is on the bus. |
| 2 | resume | R/W | 0 | **Generate RESUME State**<br>Set to generate a RESUME State on the bus. Once set, it should be left set for at least 10ms and no more than 15ms, then cleared. |
| 1 | suspend | RO | 0 | **Read Suspend Mode Status**<br>0 = Normal operation.<br>1 = USBHS is in Suspend Mode.<br>*Note: Automatically cleared when a Suspend Mode interrupt occurs, or if the resume bit (above) is set to 1.* |
| 0 | suspendm | R/W | 0 | **Suspend Mode Enable**<br>0: Suspend Mode disabled. USB will not enter Suspend Mode.<br>1: Suspend Mode allowed. If no activity is detected on the bus for more than 3.0ms, this USB enters low power Suspend Mode. |

*Table 18-8: USBHS IN Endpoints Interrupt Flags Register*

| USBHS IN Endpoints Interrupt Flags | | USBHS_INTRINFL | | [0x0002] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 16:12 | - | ROC | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11 | ep11_in | ROC | 0 | **IN EP11 Interrupt Status Flag**<br>0: IN Endpoint 11 not active.<br>1: IN Endpoint 11 occurred. |
| 10 | ep10_in | ROC | 0 | **IN EP10 Interrupt Status Flag**<br>0: IN Endpoint 10 not active.<br>1: IN Endpoint 11 occurred. |
| 9 | ep9_in | ROC | 0 | **IN EP9 Interrupt Status Flag**<br>0: IN Endpoint 9 not active.<br>1: IN Endpoint 9 occurred. |
| 8 | ep8_in | ROC | 0 | **IN EP8 Interrupt Status Flag**<br>0: IN Endpoint 8 not active.<br>1: IN Endpoint 8 occurred. |
| 7 | ep7_in | ROC | 0 | **IN EP7 Interrupt Status Flag**<br>0: IN Endpoint 7 not active.<br>1: IN Endpoint 7 occurred. |
| 6 | ep6_in | ROC | 0 | **IN EP6 Interrupt Status Flag**<br>0: IN Endpoint 6 not active.<br>1: IN Endpoint 6 occurred. |
| 5 | ep5_in | ROC | 0 | **IN EP5 Interrupt Status Flag**<br>0: IN Endpoint 5 not active.<br>1: IN Endpoint 5 occurred. |
| 4 | ep4_in | ROC | 0 | **IN EP4 Interrupt Status Flag**<br>0: IN Endpoint 4 not active.<br>1: IN Endpoint 4 occurred. |
| 3 | ep3_in | ROC | 0 | **IN EP3 Interrupt Status Flag**<br>0: IN Endpoint 3 not active.<br>1: IN Endpoint 3 occurred. |
| 2 | ep2_in | ROC | 0 | **IN EP2 Interrupt Status Flag**<br>0: IN Endpoint 2 not active.<br>1: IN Endpoint 2 occurred. |
| 1 | ep1_in | ROC | 0 | **IN EP1 Interrupt Status Flag**<br>0: IN Endpoint 1 not active.<br>1: IN Endpoint 1 occurred. |
| 0 | ep0 | ROC | 0 | **EP0 Interrupt Status Flag**<br>0: In Endpoint 0 not active.<br>1: In Endpoint 0 occurred. |

*Table 18-9: USBHS OUT Endpoints Interrupt Flags Register*

| USBHS OUT Endpoints Interrupt Flags | | USBHS_INTROUTFL | | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 16:12 | - | ROC | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11 | ep11_out | ROC | 0 | **OUT EP11 Interrupt Status Flag**<br>0: OUT Endpoint 11 interrupt not active.<br>1: OUT Endpoint 1 1interrupt active. |
| 10 | ep10_out | ROC | 0 | **OUT EP10 Interrupt Status Flag**<br>0: OUT Endpoint 10 interrupt not active.<br>1: OUT Endpoint 10 interrupt active. |
| 9 | ep9_out | ROC | 0 | **OUT EP9 Interrupt Status Flag**<br>0: OUT Endpoint 9 interrupt not active.<br>1: OUT Endpoint 9 interrupt active. |
| 8 | ep8_out | ROC | 0 | **OUT EP8 Interrupt Status Flag**<br>0: OUT Endpoint 8 interrupt not active.<br>1: OUT Endpoint 8 interrupt active. |
| 7 | ep7_out | ROC | 0 | **OUT EP7 Interrupt Status Flag**<br>0: OUT Endpoint 7 interrupt not active.<br>1: OUT Endpoint 7 interrupt active. |
| 6 | ep6_out | ROC | 0 | **OUT EP6 Interrupt Status Flag**<br>0: OUT Endpoint 6 interrupt not active.<br>1: OUT Endpoint 6 interrupt active. |
| 5 | ep5_out | ROC | 0 | **OUT EP5 Interrupt Status Flag**<br>0: OUT Endpoint 5 interrupt not active.<br>1: OUT Endpoint 5 interrupt active. |
| 4 | ep4_out | ROC | 0 | **OUT EP4 Interrupt Status Flag**<br>0: OUT Endpoint 4 interrupt not active.<br>1: OUT Endpoint 4 interrupt active. |
| 3 | ep3_out | ROC | 0 | **OUT EP3 Interrupt Status Flag**<br>0: OUT Endpoint 3 interrupt not active.<br>1: OUT Endpoint 3 interrupt active. |
| 2 | ep2_out | ROC | 0 | **OUT EP2 Interrupt Status Flag**<br>0: OUT Endpoint 2 interrupt not active.<br>1: OUT Endpoint 2 interrupt active. |
| 1 | ep1_out | ROC | 0 | **OUT EP1 Interrupt Status Flag**<br>0: OUT Endpoint 1 interrupt not active.<br>1: Interrupt occured. |
| 0 | - | ROC | 0 | **Reserved for Future Use**<br>Do not modify this field. |

*Table 18-10: USBHS IN Endpoints Interrupt Enable Register*

| USBHS IN Endpoints Interrupt Enable | | USBHS_INTRINEN | | [0x0006] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 16:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11 | ep11_in | R/W | 0 | **IN EP11 Interrupt Enable**<br>Set to 1 to enable the interrupt for IN Endpoint 11.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 10 | ep10_in | R/W | 0 | **IN EP10 Interrupt Enable**<br>Set to 1 to enable the the interrupt IN Endpoint 10.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 9 | ep9_in | R/W | 0 | **IN EP9 Interrupt Enable**<br>Set to 1 to enable the the interrupt IN Endpoint 9.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 8 | ep8_in | R/W | 0 | **IN EP8 Interrupt Enable**<br>Set to 1 to enable the the interrupt IN Endpoint 8.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 7 | ep7_in | R/W | 0 | **IN EP7 Interrupt Enable**<br>Set to 1 to enable the the interrupt IN Endpoint 7.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 6 | ep6_in | R/W | 0 | **IN EP6 Interrupt Enable**<br>Set to 1 to enable the the interrupt IN Endpoint 6.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 5 | ep5_in | R/W | 0 | **IN EP5 Interrupt Enable**<br>Set to 1 to enable the the interrupt IN Endpoint 5.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 4 | ep4_in | R/W | 0 | **IN EP4 Interrupt Enable**<br>Set to 1 to enable the the interrupt IN Endpoint 4.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 3 | ep3_in | R/W | 0 | **IN EP3 Interrupt Enable**<br>Set to 1 to enable the the interrupt IN Endpoint 3.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 2 | ep2_in | R/W | 0 | **IN EP2 Interrupt Enable**<br>Set to 1 to enable the the interrupt IN Endpoint 2.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 1 | ep1_in | R/W | 0 | **IN EP1 Interrupt Enable**<br>Set to 1 to enable the the interrupt IN Endpoint 1.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |

| USBHS IN Endpoints Interrupt Enable | | | USBHS_INTRINEN | [0x0006] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | ep0 | R/W | 0 | **EP0 Interrupt Enable**<br>Set to 1 to enable the the interrupt IN Endpoint 0.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |

*Table 18-11: USBHS OUT Endpoints Interrupt Enable Register*

| USBHS OUT Endpoints Interrupt Enable | | | USBHS_INTROUTEN | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 16:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 11 | ep11_out | R/W | 1 | **OUT EP11 Interrupt Enable**<br>Set to 1 to enable the interrupt for OUT Endpoint 11.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 10 | ep10_out | R/W | 1 | **OUT EP10 Interrupt Enable**<br>Set to 1 to enable the interrupt for OUT Endpoint 10.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 9 | ep9_out | R/W | 1 | **OUT EP9 Interrupt Enable**<br>Set to 1 to enable the interrupt for OUT Endpoint 9.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 8 | ep8_out | R/W | 1 | **OUT EP8 Interrupt Enable**<br>Set to 1 to enable the interrupt for OUT Endpoint 8.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 7 | ep7_out | R/W | 1 | **OUT EP7 Interrupt Enable**<br>Set to 1 to enable the interrupt for OUT Endpoint 7.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 6 | ep6_out | R/W | 1 | **OUT EP6 Interrupt Enable**<br>Set to 1 to enable the interrupt for OUT Endpoint 6.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 5 | ep5_out | R/W | 1 | **OUT EP5 Interrupt Enable**<br>Set to 1 to enable the interrupt for OUT Endpoint 5.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 4 | ep4_out | R/W | 1 | **OUT EP4 Interrupt Enable**<br>Set to 1 to enable the interrupt for OUT Endpoint 4.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |

| USBHS OUT Endpoints Interrupt Enable | | | USBHS_INTROUTEN | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 3 | ep3_out | R/W | 1 | **OUT EP3 Interrupt Enable**<br>Set to 1 to enable the interrupt for OUT Endpoint 3.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 2 | ep2_out | R/W | 1 | **OUT EP2 Interrupt Enable**<br>Set to 1 to enable the interrupt for OUT Endpoint 2.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 1 | ep1_out | R/W | 1 | **OUT EP1 Interrupt Enable**<br>Set to 1 to enable the interrupt for OUT Endpoint 1.<br>0: Interrupt disabled.<br>1: Interrupt enabled. |
| 0 | - | R | 0 | **Reserved for Future Use**<br>Do not modify this field. |

*Table 18-12: USBHS Signaling Interrupt Status Flag Register*

| USBHS Signaling Interrupt Status Flags | | | USBHS_INTSIGFL | [0x000A] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 8:4 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3 | sof | R | 0 | **Start Of Frame Detected Status Flag** |
| 2 | reset | R | 0 | **RESET State Detected Status Flag** |
| 1 | resume | R | 0 | **RESUME State Detected Status Flag**<br>Set when a RESUME state is detected while in SUSPEND Mode. |
| 0 | suspend | R | 0 | **Suspend Mode Status Flag**<br>Reads 1 when suspend mode is active. |

*Table 18-13: USBHS Signaling Interrupt Enable Register*

| USBHS Signaling Interrupt Enable | | | USBHS_INTSIGEN | [0x000B] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 8:4 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3 | sof | R/W | 0 | **Start Of Frame (SOF) Detected Interrupt Enable**<br>0: Interrupt event disabled.<br>1: Interrupt event enabled. |
| 2 | reset | R/W | 1 | **RESET State Detected Interrupt Enable**<br>0: Interrupt event disabled.<br>1: Interrupt event enabled. |
| 1 | resume | R/W | 1 | **RESUME State Detected Interrupt Enable**<br>0: Interrupt event disabled.<br>1: Interrupt event enabled. |

| USBHS Signaling Interrupt Enable | | | USBHS_INTSIGEN | [0x000B] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | suspend | R/W | 0 | **Suspend Mode Interrupt Enable**<br>0: Interrupt event disabled.<br>1: Interrupt event enabled. |

*Table 18-14: USBHS Frame Number Register*

| USBHS Frame Number | | | USBHS_FRAME | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:11 | framenum | R | 0 | **Frame Number**<br>Always reads 0 |
| 10:0 | framenum | R | 0 | **Read Last Received Frame Number**<br>This is the 11-bit frame number received in the SOF packet. |

*Table 18-15: USBHS Register Index Select Register*

| USBHS Register Index Select | | | USBHS_INDEX | [0x000E] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:5 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 4:0 | index | R/W | 0x0 | **Index Register Access Selector**<br>Each IN and OUT endpoint has memory-mapped control and status registers in addresses from 0x400B 1010 to 0x400B 1018. Only one endpoint's registers are addressable in the memory map at time. This bit field selects which endpoint's registers are present in the memory map where:<br>0x0: Endpoint 0 IN/OUT status registers addressable.<br>0x1: Endpoint 1 IN/OUT status registers addressable.<br>0x2: Endpoint 2 IN/OUT status registers addressable.<br>…<br>0xB: Endpoint 11 IN/OUT status registers addressable. |

*Table 18-16: USBHS Test Mode Register*

| USBHS Test Mode Register | | | USBHS_TESTMODE | [0x000F] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 8:6 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 5 | force_fs | R/W | 0 | **Force FS Mode**<br>When the USBHS receives a RESET from the Host, the USBHS is forced into Full Speed mode. |
| 4 | force_hs | R/W | 0 | **Force HS Mode**<br>When the USBHS receives a RESET from the Host, the USBHS is forced into Hi-Speed mode. |

| USBHS Test Mode Register | | | USBHS_TESTMODE | [0x000F] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 3 | test_packet | R/W | 0 | **Test Packet Mode**<br>To enter this test mode, firmware must write the standard 53-byte test packet to the Endpoint 0 FIFO, then set *USBHS_INCSRL.inpktrdy* = 1, then set this bit. The DATA0 PID is automatically added to the head of the packet and the CRC to the end of the packet. The USBHS will continue to send the test packet until this bit is cleared. |
| 2 | test_k | R/W | 0 | **HS Data K State Test Mode**<br>The USBHS transmits a continuous Data K State |
| 1 | test_j | R/W | 0 | **HS Data J State Test Mode**<br>The USBHS transmits a continuous Data J State |
| 0 | test_se0_nak | R/W | 0 | **SE0 NAK Test Mode**<br>The USBHS responds to any valid IN token with a NAK |

### 18.11.2 *Endpoint Register Access Control*

Each IN and OUT endpoint from endpoint 0x1 to 0xB uses memory mapped access to the registers in *Table 18-17*, *below*. Selecting a specific endpoint, using the *USBHS_INDEX* register, maps each of the registers in *Table 18-17* to the selected endpoint.

*Table 18-17: USB Memory Mapped Register Access for Endpoints 1 to 11*

| Endpoint Register |
|---|
| *USBHS_INMAXP* |
| *USBHS_INCSRL* |
| *USBHS_INCSRU* |
| *USBHS_OUTMAXP* |
| *USBHS_OUTCSRL* |
| *USBHS_OUTCSRU* |
| *USBHS_OUTCOUNT* |

### 18.11.3 USBHS IN Endpoint Maximum Packet Size Registers

Endpoint 1 to 11 have a memory mapped version of this register selected using the *USBHS_INDEX* register.

*Table 18-18: USBHS IN Endpoint Maximum Packet Size Register*

| USBHS IN Endpoint Maximum Packet Size | | | USBHS_INMAXP | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:11 | numpackminus1 | R/W | 0 | **Number of Split Packets - 1**<br>Defines the maximum number of packets minus 1 that a USB payload can be split into. This must be an exact multiple of *maxpacketsize*.<br>The total number of bytes transferred in the payload is:<br>$N_{BYTES\_TRANSFERED} = maxpacketsize \times (numpackminus1 + 1)$<br>This must match the *wmaxpacketsize* field of the Standard Endpoint Descriptor for the associated endpoint.<br>For HS High Bandwidth Isochronous endpoints, the multiplier can only be 2 or 3, so this field can only be 0x01 or 0x02.<br>For Bulk endpoints, the max multiplier is 32, so the maximum value for this register is 31 (0x1F).<br>*Note: Only applicable for High-Speed (HS), High-Bandwidth Isochronous endpoints and Bulk endpoints. Ignored in all other cases.* |
| 10:0 | maxpacketsize | R/W | 0 | **Maximum Packet Size in a Single Transaction**<br>This is the maximum packet size, in bytes, that is transmitted for each microframe. The maximum value is 1024, subject to the limitations for the endpoint type set in the *USB 2.0 Specification, Chapter 9.*<br>Bulk Transfers: the USB specification requires this to be 8, 16, 32, or 64. HS Bulk transfer also supports 512. |

### 18.11.4 USBHS IN Endpoint Lower Control & Status Registers

Endpoint 1 to 11 have a memory mapped version of this register selected using the *USBHS_INDEX* register.

*Table 18-19: USBHS IN Endpoint Lower Control & Status Register*

| USBHS IN Endpoint Lower Control and Status | | | USBHS_INCSRL | [0x0012] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7 | incompt | R/W0C | 0 | **Read Incomplete Split Transfer Error Status**<br>**High-bandwidth Isochronous transfers:**<br>Automatically set when a payload is split into multiple packets but insufficient IN tokens were received to send all packets. The current packet is flushed from the IN FIFO. Write a 0 to clear.<br>**Bulk and Interrupt Transfers:**<br>Ignored |
| 6 | clrdatatog | R/W1O | 0 | **Clear IN Endpoint Data Toggle**<br>1: Clear the IN endpoint data-toggle to 0.<br>*Note: Automatically cleared after set.* |
| 5 | sentstall | R/W0C | 0 | **Read STALL Handshake Sent Status**<br>Automatically set when a STALL handshake is transmitted, at which time the IN FIFO is flushed, and *USBHS_INCSRL.inpktrdy* is cleared.<br>*Note: Write a 0 to clear.* |

| USBHS IN Endpoint Lower Control and Status | | | | USBHS_INCSRL | [0x0012] |
|---|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** | |
| 4 | sendstall | R/W | 0 | **Send STALL Handshake**<br>1: Respond to an IN token with a STALL handshake.<br>0: Terminate STALL handshake<br>*Note: Ignored for Isochronous transfers.* | |
| 3 | flushfifo | R/W1O | 0 | **Flush Next Packet from IN FIFO**<br>1: Flush the next packet to be transmitted from the IN FIFO. This also clears the bit field *USBHS_INCSRL.inpktrdy*. This must only be set when *USBHS_INCSRL.inpktrdy* = 1, or FIFO data corruption might occur.<br>*Note: If the IN FIFO contains two packets set the flushfifo field twice to clear both packets.*<br>*Note: Automatically cleared when the packet is flushed.* | |
| 2 | underrun | R/W0C | 0 | **Read IN FIFO Underrun Error Status**<br>**Isochronous Mode:** Automatically set if the IN FIFO is empty (*inpktrdy*=0), an IN token has been received, and a zero-length data packet has been sent.<br>**Bulk or Interrupt Modes:** Automatically set when an IN token is received, and a NAK is sent.<br>*Note: Write a 0 to clear.* | |
| 1 | fifonotempty | R/W0C | 0 | **Read FIFO Not Empty Status**<br>Automatically set when there is at least one packet in the IN FIFO.<br>*Note: Write a 0 to clear.* | |
| 0 | inpktrdy | R/W1O | 0 | **IN Packet Ready**<br>1: Write a 1 after writing a data packet to the IN FIFO.<br>Automatically cleared when the data packet is transmitted. If double-buffering is enabled then this field is automatically cleared when there is space for a second packet in the FIFO.<br>*Note: This bit field is also controlled by USBHS_INCSRU.autoset.* | |

## 18.11.5 *USBHS Endpoint 0 Control Status Register*

*Table 18-20: USBHS Endpoint 0 Control Status Register*

| USBHS Endpoint 0 Control Status | | | | USBHS_CSR0 | [0x0012] |
|---|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** | |
| 7 | servicedsetupend | R/W1C | 0 | **Clear EP0 Setup End Bit**<br>Write a 1 to clear the *setupend* bit.<br>*Note: Automatically cleared after being set.* | |
| 6 | servicedoutpktrdy | R/W1C | 0 | **Clear EP0 Out Packet Ready Bit**<br>Write a 1 to clear the *outpktrdy* bit (below).<br>*Automatically cleared after being set.* | |
| 5 | sendstall | R/W1O | 0 | **Send EP0 STALL Handshake**<br>Write a 1 to this bit to terminate the current Control Transaction by sending a STALL handshake.<br>Automatically cleared after being set.<br>*Note: This behavior is different from the sendstall bits associated with IN/OUT endpoints.* | |

| USBHS Endpoint 0 Control Status | | USBHS_CSR0 | | [0x0012] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4 | setupend | RO | 0 | **Read Setup End Status**<br>Automatically set when a Control Transaction ends before the dataend bit has been set by firmware.<br>An interrupt is generated when this bit is set.<br>Write a 1 to servicedsetupend (above) to clear. |
| 3 | dataend | R/W1O | 0 | **Control Transaction Data End**<br>Write a 1 to this bit after firmware completes any of the following three transactions:<br>6) Set *inpktrdy* = 1 for the last data packet.<br>9) Set *inpktrdy* = 1 for a zero-length data packet.<br>10) Clear *outpktrdy* = 0 after unloading the last data packet.<br>*Note: Automatically cleared.* |
| 2 | sentstall | R/W0C | 0 | **Read EP0 STALL Handshake Sent Status**<br>Automatically set when a STALL handshake is transmitted.<br>Write a 0 to clear. |
| 1 | inpktrdy | R/W1O | 0 | **EP0 IN Packet Ready**<br>Set this bit to indicate a packet is ready to transmit from the IN FIFO. Hardware automatically clears this bit when the packet transmit is complete.<br>0: Packet was transmitted or no packet transmit pending. Read only.<br>1: Write a 1 after writing a data packet to the IN FIFO to indicate the EP0 IN packet is ready.<br>*Note: An interrupt is generated when this bit is cleared.* |
| 0 | outpktrdy | RO | 0 | **EP0 OUT Packet Ready Status**<br>Automatically set when a data packet is received in the OUT FIFO.<br>An interrupt is generated when this bit is set.<br>Write a 1 to the *servicedoutpktrdy* bit (above) to clear after the packet is unloaded from the OUT FIFO. |

### 18.11.6 *USBHS IN Endpoint Upper Control Registers*

Endpoint 1 to 11 have a memory mapped version of this register selected using the *USBHS_INDEX* register.

*Table 18-21: USBHS IN Endpoint Upper Control Register*

| USBHS IN Endpoint Upper Control | | USBHS_INCSRU | | [0x0013] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7 | autoset | R/W | 0 | **Auto Set inpktrdy**<br>0: *USBHS_INCSRL.inpktrdy* must be set by firmware<br>1: *USBHS_INCSRL.inpktrdy* is automatically set when data that is of the maximum packet size specified in the *USBHS_INMAXP* register is loaded into the IN FIFO. |
| 6 | iso | R/W | 0 | **Isochronous Transfer Enable**<br>0: Enable IN Bulk and IN Interrupt transfers<br>1: Enable IN Isochronous transfers |
| 5 | mode | R/W | 0 | **Endpoint Direction Mode**<br>0: Endpoint direction is OUT<br>1: Endpoint direction is IN<br>*Note: Ignored if endpoint is not used for both IN and OUT transactions.* |

| USBHS IN Endpoint Upper Control | | USBHS_INCSRU | | [0x0013] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 4 | dmareqenab | R/W | 0 | **DMA Request Enable**<br>0: Disable DMA for this IN endpoint<br>1: Enable DMA for this IN endpoint |
| 3 | frcdatatog | R/W | 0 | **Force IN Data-Toggle**<br>0: Toggle data-toggle only when an ACK is received<br>1: Toggle data-toggle regardless of whether an ACK is received<br>*Note: Useful for Interrupt IN endpoints that are communicating rate feedback to Isochronous endpoints.* |
| 2 | dmareqmode | R/W | 0 | **DMA Request Mode Enable**<br>0: Enable DMA Request Mode 0. A DMA request is generated for each packet transmission. This mode can only be selected after the *dmareqenab* bit is cleared first.<br>1: Enable DMA Request Mode 1. A DMA request is generated only when a packet of size *USBHS_INMAXP.maxpacketsize* is received. |
| 1 | dpktbufdis | R/W | 0 | **Double Packet Buffering Disable**<br>0: Enable double packet buffering. Firmware must also configure the FIFO and packet size.<br>1: Disable double packet buffering |
| 0 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

*Table 18-22: USBHS OUT Endpoint Maximum Packet Size Register*

| USBHS OUT Endpoint Maximum Packet Size | | USBHS_OUTMAXP | | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:11 | numpackminus1 | R/W | 0x00 | **Number of Split Packets**<br>Defines the maximum number of packets minus 1 that a USB payload is combined into. The value must be an exact multiple of *maxpacketsize*.<br>The total number of bytes transferred in the payload is *maxpacketsize* x (numpackminus1+1). This must match the *wMaxPacketSize* field of the Standard Endpoint Descriptor for the associated endpoint.<br>Only applicable for Hi-Speed (HS), High Bandwidth Isochronous endpoints and Bulk endpoints. Ignored in all other cases.<br>**HS High Bandwidth Isochronous endpoints**<br>The multiplier can only be 2 or 3, so this bit field value can only be 0x01 or 0x02.<br>**Bulk endpoints**<br>The max multiplier is 32, so the maximum value for this register is 31 (0x1F). |
| 10:0 | maxpacketsize | R/W | 0x000 | **Maximum Packet Size in a Single Transaction** [1]<br>This is the maximum packet size, in bytes, that is transmitted for each microframe. The maximum value is 1024, subject to the limitations for the endpoint type set in the USB 2.0 Specification, Chapter 9.<br>For all Bulk Transfers, the USB specification requires this to be 8, 16, 32, or 64. HS Bulk transfer also supports 512. |

*Table 18-23: USBHS OUT Endpoint Lower Control Status Register*

| USBHS OUT Endpoint Lower Control Status | USBHS_OUTCSRL | [0x0016] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 7 | clrdatatog | R/W1O | 0 | **Clear IN Endpoint Data Toggle**<br>1: Clear the OUT endpoint data-toggle to 0.<br>*Note: Automatically cleared.* |
| 6 | sentstall | R/W0C | 0 | **STALL Handshake Sent Status**<br>Automatically set when a STALL handshake is transmitted.<br>Write a 0 to clear. |
| 5 | sendstall | R/W | 0 | **Send STALL Handshake**<br>1: Send a STALL handshake to a DATA packet<br>0: Terminate STALL handshake<br>Ignored for Isochronous transfers. Write a 0 to clear. |
| 4 | flushfifo | R/W1O | 0 | **Flush OUT FIFO Packet**<br>1: Flush the next packet to be read from the OUT FIFO. This also clears the *outpktrdy* bit. This must only be set when *outpktrdy* = 1, or data corruption in the FIFO might occur.<br>If the out FIFO contains two packets, *flushfifo* might need to be set twice to completely clear the FIFO.<br>*Note: Automatically cleared when the packet is flushed.* |
| 3 | dataerror | RO | 0 | **OUT Packet CRC Error Status**<br>**Isochronous Mode:** Automatically set if a data packet is received (*outpktrdy* = 1), and the data packet has a CRC error. Automatically cleared when *outpktrdy* = 0.<br>**Bulk or Interrupt Modes:** Always returns 0. |
| 2 | overrun | R/W0C | 0 | **OUT FIFO Overrun Error Status**<br>**Isochronous Mode:**<br>Automatically set if the OUT FIFO is full (*fifofull* = 1), and an OUT packet arrives. In this case, the OUT packet is lost.<br>**Bulk or Interrupt Modes:**<br>Always reads 0.<br>*Note: Write a 0 to clear.* |
| 1 | fifofull | RO | 0 | **FIFO Full Status**<br>Set when the OUT FIFO is full.<br>*Note: Automatically cleared when the FIFO is no longer full.* |
| 0 | outpktrdy | R/W0C | 0 | **OUT Packet Ready Status**<br>Automatically set when a data packet is received in the OUT FIFO.<br>Write a 0 to clear after the packet is unloaded from the OUT FIFO.<br>*Note: Write 0 to clear.* |

*Table 18-24: USBHS OUT Endpoint Upper Control Status Register*

| USBHS OUT Endpoint Upper Control Status Register | USBHS_OUTCSRU | [0x0017] |
|---|---|---|

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 7 | autoclear | R/W | 0 | **Auto Clear outpktrdy**<br>0: *USBHS_OUTCSRL.outpktrdy* must be cleared by firmware<br>1: *USBHS_OUTCSRL.outpktrdy* is automatically cleared when data that is of the maximum packet size specified in the *USBHS_OUTMAXP* register is unloaded from the OUT FIFO. If packets less than the maximum packet size are unloaded, *outpktrdy* must be cleared by firmware.<br>*Note: Do not set for High Bandwidth Isochronous endpoints.* |
| 6 | iso | R/W | 0 | **Isochronous Transfer Enable**<br>0: Enable OUT Bulk and OUT Interrupt transfers<br>1: Enable OUT Isochronous transfers |
| 5 | dmareqen | R/W | 0 | **DMA Request Enable**<br>0: Disable DMA for this OUT endpoint<br>1: Enable DMA for this OUT endpoint |
| 4 | disnyet/piderror | R/W<br>R/W0C | 0<br>0 | **Disable NYET Packets (HS Bulk and HS Interrupt Modes)**<br>0: If the OUT FIFO is full, respond to newly-received OUT packets with a NYET (Not Yet) packet to indicate the FIFO is full.<br>1: Disable NYET packets. Respond to all received OUT packets with an ACK even when the FIFO is full.<br>**PID Error Status (Isochronous Mode only)**<br>Automatically set if there is a PID (Packet ID) error in the received OUT packet.<br>*Note: Write 0 to clear.*<br>*Note: Ignored in all other modes.*<br>*Note: Bit 4 is dual-use and can be addressed by two different names depending on the endpoint mode.* |
| 3 | dmareqmode | R/W | 0 | **DMA Request Mode Enable**<br>0: Enable DMA Request Mode 0. A DMA request is generated after each OUT packet is received.<br>1: Enable DMA Request Mode 1. A DMA request is generated only when a packet of *USBHS_OUTMAXP.maxpacketsize* is received. |
| 2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | dpktbufdis | R/W | 0 | **Double Packet Buffering Disable**<br>0: Enable double packet buffering. Firmware must configure the FIFO and packet size.<br>1: Disable double packet buffering. |
| 0 | incomprx | R | 0 | **Incomplete Isochronous Packet Received Error Status**<br>**High Bandwidth Isochronous Mode:**<br>Automatically set if an incomplete packet is received in the OUT FIFO. Automatically cleared when *USBHS_OUTCSRL.outpktrdy* is cleared.<br>**Bulk or Interrupt Modes:**<br>Always reads 0. |

*Note: Endpoint 1 to 11 have a memory mapped version of this register selected using the USBHS_INDEX register.*

*Table 18-25: USBHS Endpoint OUT FIFO Byte Count Register*

| USBHS Endpoint OUT FIFO Byte Count | | | USBHS_OUTCOUNT | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:13 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 12:0 | outcount | RO | 0 | **Read Number of Data Bytes in OUT FIFO**<br>Returns the number of data bytes in the packet that are read next in the OUT FIFO.<br>*Note: This value changes as the contents of the FIFO change.*<br>*Note: This value is only valid when a packet is in the OUT FIFO (USBHS_OUTCSRL.outpktrdy = 1).* |

*Table 18-26: USBHS Endpoint 0 IN FIFO Byte Count Register*

| USBHS Endpoint 0 IN FIFO Byte Count | | | USBHS_COUNT0 | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:7 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 6:0 | count0 | RO | 0 | **Read Number of Data Bytes in the Endpoint 0 FIFO**<br>Returns the number of data bytes in the Endpoint 0 FIFO.<br>*Note: This field changes as the contents of the FIFO change.*<br>*Note: This field is only valid when USBHS_OUTCSRL.outpktrdy = 1.* |

*Table 18-27: USBHS FIFO for Endpoint n Register*

| | | |
|---|---|---|
| **USBHS FIFO for Endpoint 0** | **USBHS_FIFO0** | **[0x0020]** |
| **USBHS FIFO for Endpoint 1** | **USBHS_FIFO1** | **[0x0024]** |
| **USBHS FIFO for Endpoint 2** | **USBHS_FIFO2** | **[0x0028]** |
| **USBHS FIFO for Endpoint 3** | **USBHS_FIFO3** | **[0x002C]** |
| **USBHS FIFO for Endpoint 4** | **USBHS_FIFO4** | **[0x0030]** |
| **USBHS FIFO for Endpoint 5** | **USBHS_FIFO5** | **[0x0034]** |
| **USBHS FIFO for Endpoint 6** | **USBHS_FIFO6** | **[0x0038]** |
| **USBHS FIFO for Endpoint 7** | **USBHS_FIFO7** | **[0x003C]** |
| **USBHS FIFO for Endpoint 8** | **USBHS_FIFO8** | **[0x0040]** |
| **USBHS FIFO for Endpoint 9** | **USBHS_FIFO9** | **[0x0044]** |
| **USBHS FIFO for Endpoint 10** | **USBHS_FIFO10** | **[0x0048]** |
| **USBHS FIFO for Endpoint 11** | **USBHS_FIFO11** | **[0x004C]** |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 31:0 | usbhs_fifo | R/W | - | **USBHS Endpoint FIFO Read/Write Register**<br>Reads from this register unloads data from the OUT FIFO for the corresponding endpoint.<br>Writes to this register loads data into the IN FIFO for the corresponding endpoint. FIFO reads and writes may be 8-bit, 16-bit, 24-bit or 32-bit. Any combination is allowed provided the data accessed is contiguous.<br>However, all reads and writes for a packet must be of the same width so that the data is consistently byte-, word- or double-word-aligned. The last transfer can contain fewer bytes than the previous transfers when completing an odd-byte or odd-word transfer.<br>*Note: The value of these registers at reset is undetermined.* |

*Table 18-28: USBHS Endpoint Count Info Register*

| | | |
|---|---|---|
| **USBHS Endpoint Count Info** | **USBHS_EPINFO** | **[0x0078]** |

| Bits | Name | Access | Reset | Description |
|---|---|---|---|---|
| 7:4 | outendpoints | RO | 0xB | **Number of OUT Endpoints**<br>There are 11 OUT endpoints in this USB HS peripheral.<br>0xB: 11 OUT Endpoints. |
| 3:0 | inendpoints | RO | 0xB | **Number of IN Endpoints**<br>Returns the number of IN endpoints in this USB HS peripheral.<br>0xB: 11 IN Endpoints |

*Table 18-29: USBHS RAM Info Register*

| USBHS RAM Info | | | USBHS_RAMINFO | [0x0079] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:4 | - | RO | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3:0 | rambits | RO | 0xC | **Number of RAM Address Bits**<br>The width of the RAM address bus in this USBHS module. The width is 12 bits.<br>0xC: 12 bit wide RAM address supported in the USB HS peripheral. |

*Table 18-30: USBHS Soft Reset Control Register*

| USBHS Soft Reset Control | | | USBHS_SOFTRESET | [0x007A] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | rstxs | R/W1O | 0 | **Reset the USB PHY.**<br>Write a 1 to reset the USB PHY.<br>This field is cleared by hardware automatically after a 1 is written and the USB PHY is reset.<br>0: USB PHY reset complete or not initiated.<br>1: Write 1 to reset the USB PHY. |
| 0 | rsts | R/W1O | 0 | **Reset the USB Controller.**<br>Write 1 to reset the USBHS controller.<br>This field is cleared by hardware automatically after a 1 is written and the USBHS controller is reset.<br>0: USBHS controller reset complete or not initiated.<br>1: Write 1 to reset the USBHS controller. |

*Table 18-31: USBHS Early DMA Register*

| USBHS Early DMA | | | USBHS_EARLYDMA | [0x007B] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | edmain | R/W | 1 | **Early DMA IN Endpoints Enable**<br>0: DMA Request signal for all IN endpoints is deasserted when MAXP bytes have been written to an endpoint.<br>1: DMA Request signal for all IN endpoints is deasserted when $(\mathrm{MAXP} - 8)$ bytes have been written to an endpoint. |
| 0 | edmaout | R/W | 0 | **Early DMA OUT Endpoints Enable**<br>0: DMA Request signal for all OUT endpoints is deasserted when MAXP bytes have been read from an endpoint.<br>1: DMA Request signal for all OUT endpoints is deasserted when $(\mathrm{MAXP} - 8)$ bytes have been read from an endpoint. |

*Table 18-32: USBHS Hi-Speed Chirp Timeout Register*

| USBHS Hi-Speed Chirp Timeout | | | USBHS_CTUCH | [0x0080] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | c_t_uch | R/W | 0x203A | **HS Chirp Timeout Clock Cycles**<br>This configures the chirp timeout used by this Device to negotiate a HS connection with a FS Host.<br>$t_{CHIRP\_TIMEOUT}(PHY\ clock\ cycles) = c\_t\_uch \times 4$<br>The timeout value represents the number of 30MHz PHY clock cycles (66.7ns) before the chirp timeout occurs. |

*Table 18-33: USBHS Hi-Speed RESUME Delay Register*

| USBHS Hi-Speed RESUME Delay | | | USBHS_CTHSRTN | [0x0082] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15:0 | c_t_hsrtn | R/W | 0x0019 | **Hi-Speed RESUME Delay Clock Cycles**<br>This configures the delay from when the RESUME state on the bus ends, to when the USBHS resumes normal operation.<br>$t_{HI\_SPEED\_DELAY}(PHY\ clock\ cycles) = c\_t\_hsrtn \times 4$<br>The delay value represents the number of 30MHz PHY clock cycles (66.7ns) from the end of the RESUME state to when normal USBHS operation begins. |

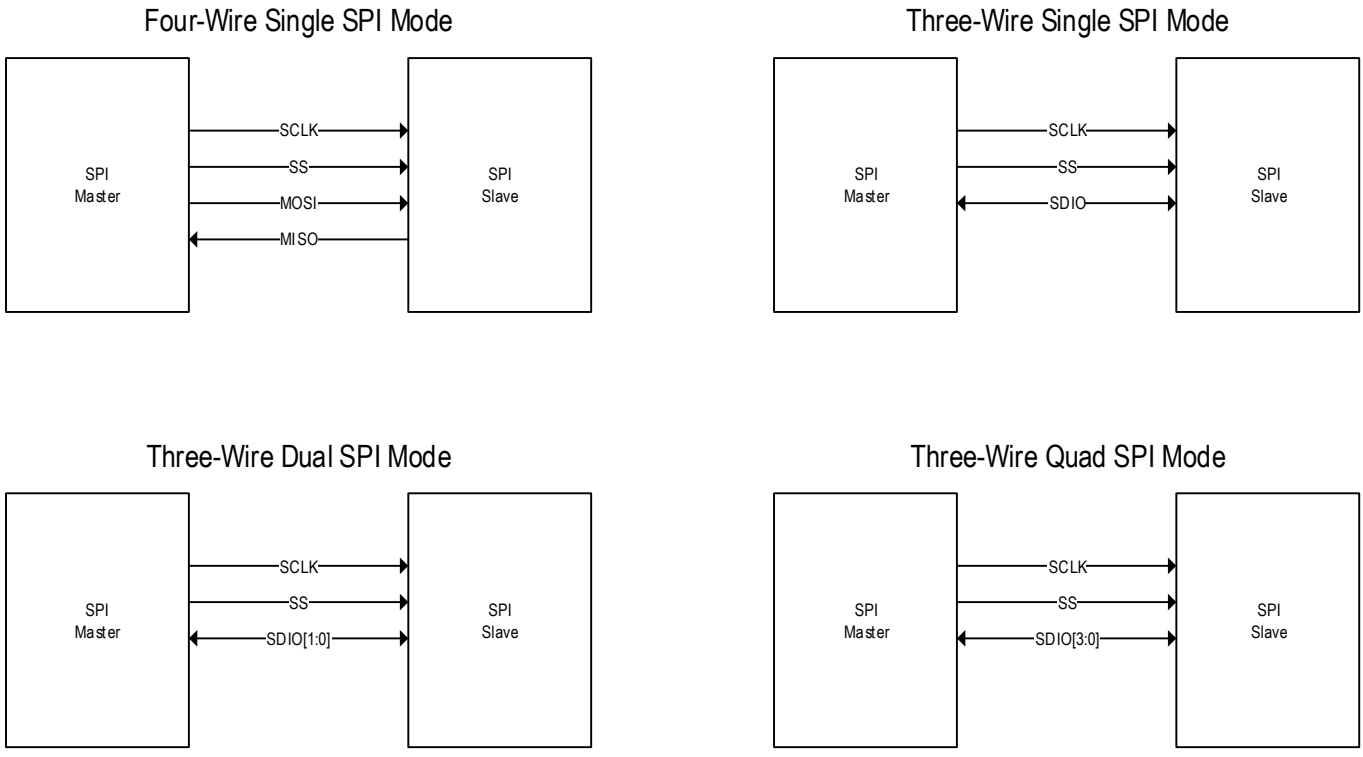
# 19 Quad Serial Peripheral Interface (SPI3)

The Quad Serial Peripheral Interface (SPI3) is a highly configurable, synchronous communications peripheral that interfaces to SPI devices as either a Master or Slave. The Quad SPI port is designated as SPI3.

## 19.1 Features

- Single, Dual and Quad SPI communication modes
- Four SPI modes (mode 0, 1, 2, and 3)
- Master, Multi-Master, and Slave modes
- Wakeup from low power state based on configurable Transmit and Receive FIFO Levels
- Up to four Slave Select (SS) control lines with programmable polarity
- Programmable Serial Clock (SCLK) frequency and duty cycle
- Programmable Slave Select Timing to enable flexible hold and recovery time for external slave devices
- 32-byte Transmit FIFO, 32-byte Receive FIFO

*Figure 19-1: SPI Modes of Operation*



### 19.1.1 *SPI Signals*

- SS = Slave Select with programmable polarity
  - SPI3 supports up to 4 Slave Select lines (SPI3_SS[0:3])
- SCLK = Serial Clock configurable as an output for master mode and an input for slave mode
- Four-Wire Single SPI Mode
  - Master Out Slave In (MOSI)
  - Serial data output pin in master mode.
  - Serial data input pin in slave mode.
  - Master In Slave Out (MISO)
  - Serial data input pin in master mode.
  - Serial data output pin in slave mode.
- Three-Wire Single SPI Mode
  - Serial Data Input/Output (SDIO)
  - Single Bidirectional serial data pin
  - Dual SPI Mode
  - Serial Data Input/Output (SDIO)
  - Two Bidirectional serial data pins (SDIO[0:1])
- Quad SPI Mode
  - Serial Data Input/Output (SDIO)
  - Four Bidirectional serial data pins (SDIO[0:3])

## 19.2 SPI Configuration

Before configuring the SPI peripheral, first disable the SPI port by clearing the register bit *SPI3_CTRL0.enable*.

With the SPI peripheral disabled, configure the SPI port for master mode (*SPI3_CTRL0.master* = 1) or for slave mode (*SPI3_CTRL0.master*=0.

Next, configure communication specific parameters such as clock phase, width, number of bits per character, and signal polarity using the *SPI3_CTRL2* register.

For Master Mode, Slave Select timing controls are available in *SPI3_SS_TIME* register.

Clock scaling and duty cycle control for SCLK are configured using the *SPI3_CLK_CFG* register.

Interrupt events are configured using the *SPI3_INT_EN* register.

Wakeup events are configured using the *SPI3_WAKE_EN* register.

The DMA is configured using *SPI3_DMA*.

If the SPI is configured in Master Mode, configure *SPI3_CTRL1* to set Master Mode parameters including the selection of the SS signals and the SS signal polarity.

Enable the Transmit FIFO if transmitting data and the Receive FIFO

If transmitting data, load data to the transmit FIFO.

Set *SPI3_CTRL0.start*=1 to begin a Master Mode transmission.

Do not modify the SPI timing registers while a SPI transaction is in progress. Modifying any SPI timing register while a SPI transfer is in progress will result in an invalid SPI communication transaction.

To prevent a stall condition when in Master Mode, ensure that the transmit FIFO does not empty until the entire transmission is complete.

### 19.2.1 SPI3 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.

### 19.2.2 SPI3 Interrupts and Wakeups

The SPI supports multiple interrupt sources. Interrupt source events can come from the FIFOs, the SS and SR signals, and SPI status. 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. Level is set by firmware.
- Receive FIFO Full
- Receive FIFO threshold. Level is set by firmware.
- Transmit FIFO Underrun (Slave mode only, Master mode will stall the clock)
- Transmit FIFO Overrun
- Receive FIFO Underrun
- Receive FIFO Overrun (Slave Mode only, Master Mode will stall the clock)

SPI3 supports interrupts for the internal state of the SPI as well as external signals. The following transmission interrupts are supported:

- SS Asserted or Deasserted
- Transaction Complete
- Slave Mode Transaction Aborted
- Multi-Master Fault

SPI3 includes four sources that wakeup the Arm processor from low-power modes when the WAKE event is enabled and subsequently occurs. The following wakeup events are supported:

- RX FIFO Full
- TX FIFO Empty

- RX FIFO Threshold
- TX FIFO Threshold

## 19.3 Timing Diagrams

The following waveform diagrams show SPI communications in each of the four SPI modes.

### 19.3.1 *SPI Mode 0*

*Figure 19-2: SPI Mode 0, Four-Wire Communication*

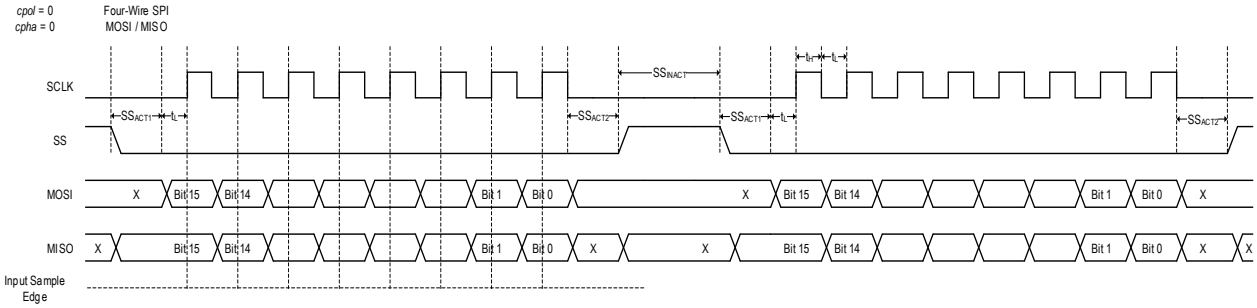
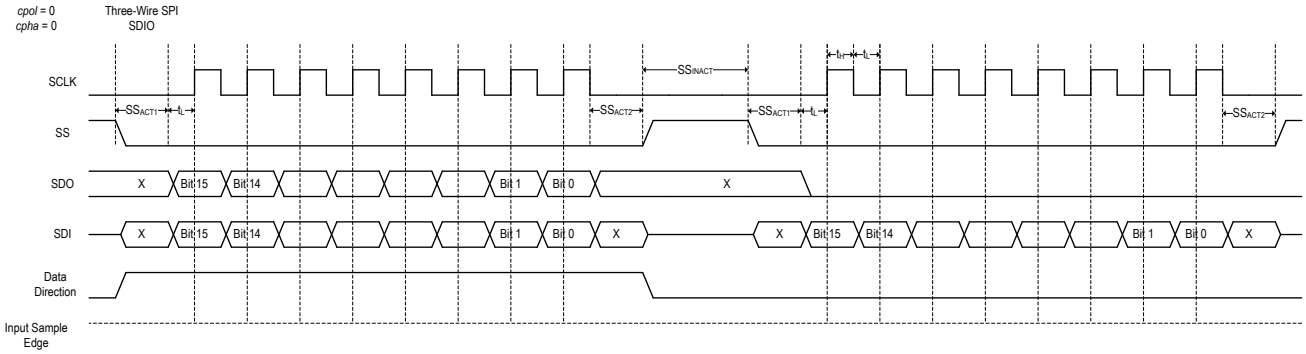



*Figure 19-3: SPI Mode 0, Three-Wire Communication*




### 19.3.2 *SPI Mode 1*

*Figure 19-4: SPI Mode 1, Four-Wire Communication*

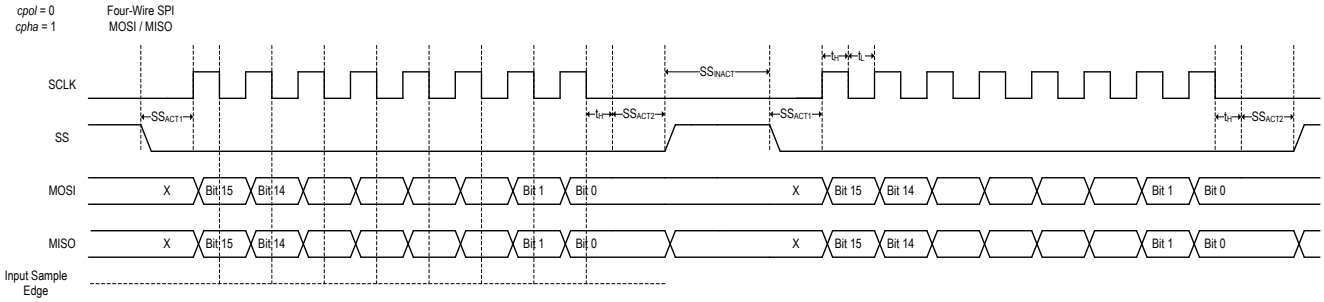
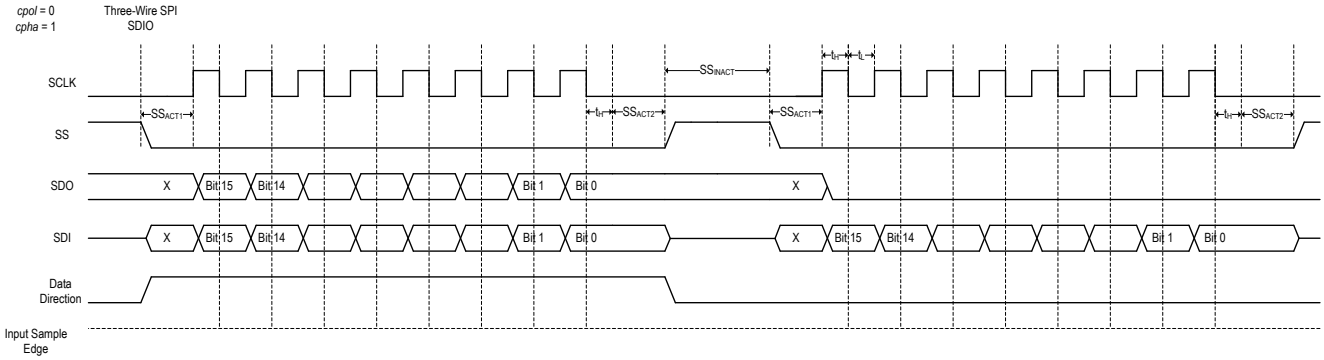


*Figure 19-5: SPI Mode 1, Three-Wire Communication*




## 19.3.3 *SPI Mode 2*

*Figure 19-6: SPI Mode 2, Four-Wire Communication*

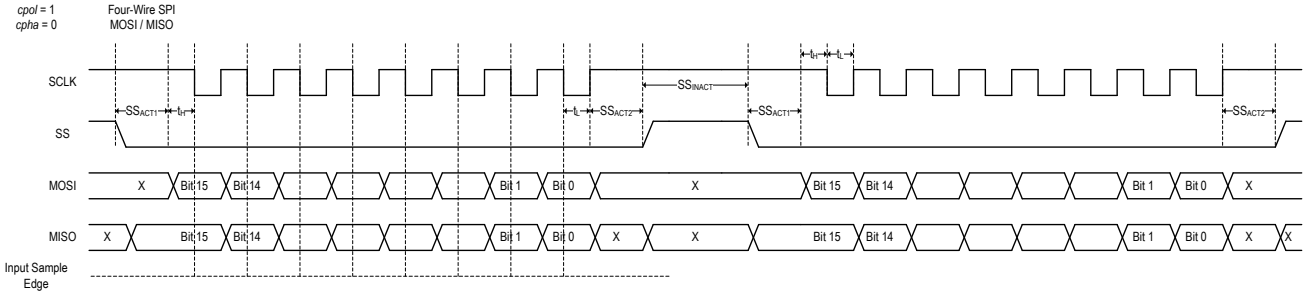
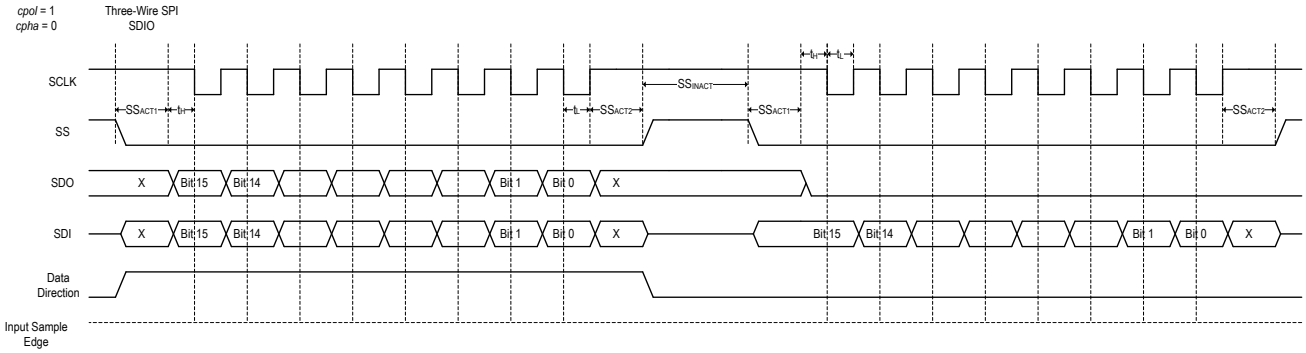



*Figure 19-7: SPI Mode 2, Three-Wire Communication*



### 19.3.4 *SPI Mode 3*

*Figure 19-8: SPI Mode 3, Four-Wire Communication*

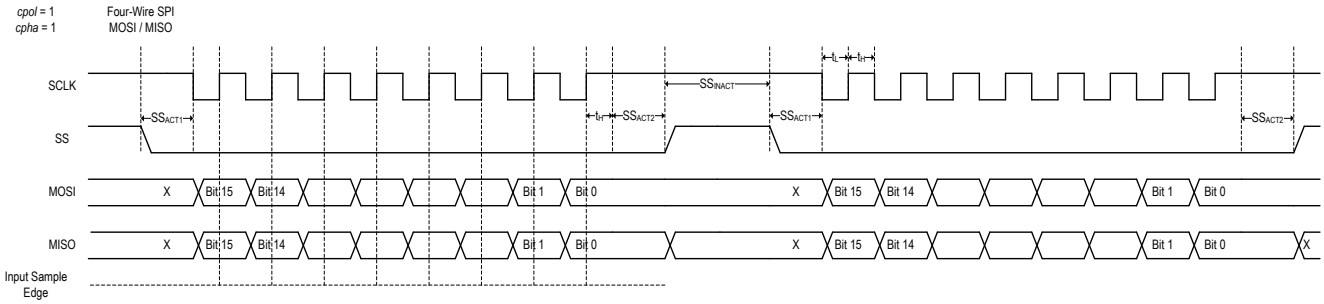
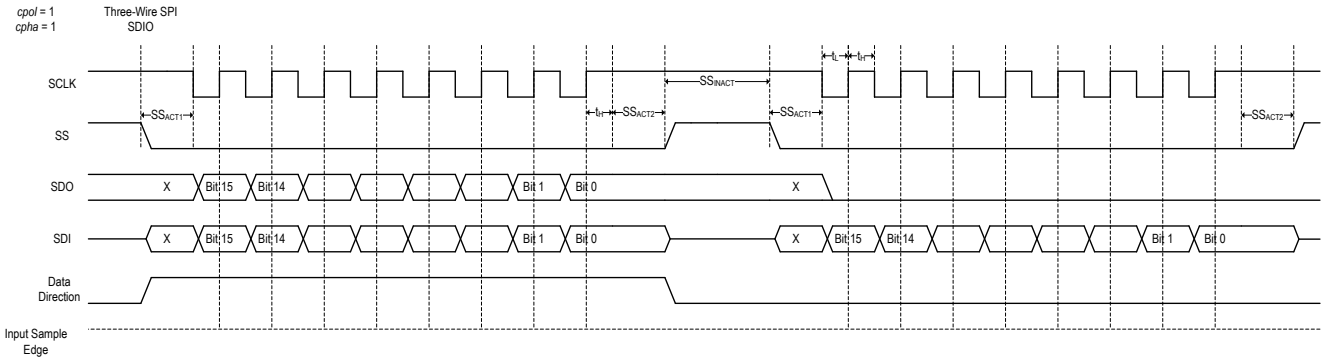



*Figure 19-9: SPI Mode 3, Three-Wire Communication*




## 19.4 Quad SPI Master(SPI3) Registers

Refer to *Table 2-1: APB Peripheral Base Address Map* for the Quad SPI Master (SPI3) Base Peripheral Address.

*Table 19-1: Quad SPI (SPI3) Offsets, Register Names, Access and Descriptions*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *SPI3_DATA* | R/W | SPI3 FIFO Data Register |
| [0x0004] | *SPI3_CTRL0* | R/W | SPI3 Master Signals Control Register |
| [0x0008] | *SPI3_CTRL1* | R/W | SPI3 Transmit Packet Size Register |
| [0x000C] | *SPI3_CTRL2* | R/W | SPI3 Static Configuration Register |
| [0x0010] | *SPI3_SS_TIME* | R/W | SPI3 Slave Select Timing Register |
| [0x0014] | *SPI3_CLK_CFG* | R/W | SPI3 Master Clock Configuration Register |
| [0x001C] | *SPI3_DMA* | R/W | SPI3 DMA Control Register |
| [0x0020] | *SPI3_INT_FL* | R/W1C | SPI3 Interrupt Flag Register |
| [0x0024] | *SPI3_INT_EN* | R/W | SPI3 Interrupt Enable Register |
| [0x0028] | *SPI3_WAKE_FL* | R/W1C | SPI3 Wakeup Flags Register |
| [0x002C] | *SPI3_WAKE_EN* | R/W | SPI3 Wakeup Enable Register |
| [0x0030] | *SPI3_STAT* | RO | SPI3 Status Register |

## 19.5 Quad SPI Master Register Details

*Table 19-2: SPI3 FIFO Data Registers*

| SPI3 FIFO Data Register | | | SPI3_DATA | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:0 | data | R/W | 0 | **SPI FIFO Data Register**<br>Reads dequeue data off the receive FIFO.<br>Writes queue data onto the transmit FIFO.<br>Reads and writes with this register are in 1-byte, 2-byte, or 4-byte formats only. |

*Table 19-3: SPI3 Control 0 Registers*

| SPI3 Control 0 Register | | | SPI3_CTRL0 | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:20 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 19:16 | ss | R/W | 0 | **Master Slave Select**<br>For SPI3, there is only one Slave Select line. Selects which SS signal is active when the next transaction is started (*SPI3_CTRL0.start* = 1). More than one SS output can be asserted by setting the appropriate bits in this field, for example, to use SPI3_SS0 and SPI3_SS3 for a transaction, write 0b1001 to this field.<br>0b0001: SPI3_SS0<br>0b0010: SPI3_SS1<br>0b0100: SPI3_SS2<br>0b1000: SPI3_SS3<br>*Note: This field is only used when the SPI3 is configured for Master Mode (SPI3_CTRL0.master = 1).* |
| 15:9 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 8 | ss_ctrl | R/W | 0 | **Master Slave Select Control**<br>0: Slave Select is deasserted at the end of a transmission<br>1: Slave Select stays asserted at the end of a transmission |
| 7:6 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 5 | start | R/WAC | 0 | **Master Start Data Transmission**<br>1: Master initiates a data transmission. Ensure that all pending transactions are complete before writing a 1.<br>This bit is cleared by hardware. Writing a 0 is ignored.<br>*Note: This field is only used when the SPI is configured for Master Mode (SPI3_CTRL0.master = 1).* |
| 4 | ss_io | R/W | 0 | **Master Slave Select Signal Direction**<br>0: Slave Select is an output<br>1: Slave Select is an input<br>*Note: This field is only used when the SPI is configured for Master Mode (SPI3_CTRL0.master = 1).* |

| SPI3 Control 0 Register | | | SPI3_CTRL0 | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 3:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | master | R/W | 0 | **SPI Master Mode Enable**<br>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.<br>0: SPI port is in Slave Mode.<br>1: SPI is in Master Mode |
| 0 | enable | R/W | 0 | **SPI Enable/Disable**<br>This field enables the SPI port instance. Setting this field disables the SPI port, but does not change the contents of the receive or transmit FIFOs or other SPI registers.<br>0: SPI port is disabled<br>1: SPI port is enabled |

*Table 19-4: SPI3 Transmit Packet Size Register*

| SPI3 Transmit Packet Size Register | | | SPI3_CTRL1 | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | rx_num_char | R/W | 0 | **Number of Receive Characters**<br>Number of characters to receive in RX FIFO.<br>*Note: If the SPI port is set to operate in 4-wire mode, this field is ignored and the tx_num_chars field is used for both the number of characters to receive or transmit.* |
| 15:0 | tx_num_char | R/W | 0 | **Number of Transmit Characters**<br>Number of characters to transmit from TX FIFO.<br>*Note: In 4-wire mode, this also applies to the RX FIFO.* |

*Table 19-5: SPI3 Control 2 Register*

| SPI3 Control 2 Register | | | SPI3_CTRL2 | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:20 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 19:16 | ss_pol | R/W | 0 | **Slave Select Polarity**<br>Controls the polarity of each individual SS signal where each bit position corresponds to a SS signal. SPI3_SS0 is controlled with bit position 0 and SPI3_SS3 is controlled with bit position 3.<br>For each bit position,<br>0: SS is active low<br>1: SS is active high |
| 15 | three_wire | R/W | 0 | **Three-Wire Mode Enable**<br>0: Four-wire mode enabled (Single Mode only)<br>1: Three-wire mode enabled |

| SPI3 Control 2 Register | | | SPI3_CTRL2 | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 14 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 13:12 | data_width | R/W | 0b00 | **SPI Data Width**<br>0: 1-data pin (Single Mode)<br>1: 2-data pins (Dual Mode)<br>2: 4-data pins (Quad Mode)<br>3: Reserved |
| 11:8 | numbits | R/W | 0x0 | **Number of Bits per Character**<br>1-bit and 9-bit character lengths are not supported in Slave Mode |
| 7:2 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 1 | cpol | R/W | 0 | **Clock Polarity**<br>0: Normal clock. Use when in SPI Mode 0 and Mode 1<br>1: Inverted clock. Use when in SPI Mode 2 and Mode 3 |
| 0 | cpha | R/W | 0 | **Clock Phase**<br>0: Data sampled on clock rising edge. Use when in SPI Mode 0 and Mode 2<br>1: Data sampled on clock falling edge. Use when in SPI Mode 1 and Mode 3 |

*Table 19-6: SPI3 Slave Select Timing Register*

| SPI3 Slave Select Timing Register | | | SPI3_SS_TIME | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:24 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 23:16 | inact | R/W | 0 | **Inactive Stretch**<br>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).<br>0: 256<br>1: 1<br>2: 2<br>3:3<br>…<br>…<br>254: 254<br>255: 255 |
| 15:8 | post | R/W | 0 | **Slave Select Hold Post Last SCLK**<br>Number of system clock cycles that SS remains active after the last SCLK edge.<br>0: 256<br>1: 1<br>2: 2<br>3:3<br>…<br>…<br>254: 254<br>255: 255 |

| SPI3 Slave Select Timing Register | | | SPI3_SS_TIME | [0x0010] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 7:0 | pre | R/W | 0 | **Slave Select Delay to First SCLK**<br>Set the number of system clock cycles the Slave Select is held active prior to the first SCLK edge.<br>0: 256<br>1: 1<br>2: 2<br>3:3<br>…<br>…<br>254: 254<br>255: 255 |

*Table 19-7: SPI3 Master Clock Configuration Registers*

| SPI3 Master Clock Configuration Register | | | SPI3_CLK_CFG | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:20 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 19:16 | scale | R/W | 0 | **SPI Peripheral Clock Scale**<br>Scales the Peripheral Clock (PCLK) by $2^{scale}$ to generate the SPI3 peripheral clock.<br>$f_{SPI3CLK} = \frac{f_{PCLK}}{2^{scale}}$<br>Valid values for scale are 0 to 8 inclusive. Values greater than 8 are reserved for future use.<br>*Note: If SPI3_CLK_CFG.scale = 0, SPI3_CLK_CFG.hi = 0, and SPI3_CLK_CFG.lo = 0, character sizes of 2 and 10 bits are not supported.* |
| 15:8 | hi | R/W | 0 | **SCLK Hi Clock Cycles Control**<br>0: Hi duty cycle control disabled. Only valid if scale = 0.<br>1 to 15: The number of SPI peripheral clocks, $f_{SPI3CLK}$, that SCLK is high.<br>*Note: If SPI3_CLK_CFG.scale=0, SPI3_CLK_CFG.hi=0, and SPI3_CLK_CFG.lo=0, character sizes of 2 and 10 bits are not supported.* |
| 7:0 | lo | R/W | 0 | **SCLK Low Clock Cycles Control**<br>0: Low duty cycle control disabled. Setting this field to 0 is only valid if Only valid if SPI3_CLK_CFG.scale = 0.<br>1 to 15: The number of SPI3 peripheral clocks, $f_{SPI3CLK}$, that the SCLK signal is low.<br>*Note: If SPI3_CLK_CFG.scale = 0, SPI3_CLK_CFG.hi = 0, and SPI3_CLK_CFG.lo = 0, character sizes of 2 and 10 bits are not supported.* |

*Table 19-8: SPI3 DMA Control Registers*

| SPI3 DMA Control Register | | | SPI3_DMA | [0x001C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31 | rx_dma_en | R/W | 0 | **RX DMA Enable**<br>0: RX DMA is disabled. Any pending DMA requests are cleared<br>1: RX DMA is enabled |

| SPI3 DMA Control Register | | SPI3_DMA | | [0x001C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 30 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 29:24 | rx_fifo_cnt | R | 0 | **Number of Bytes in the RX FIFO**<br>Read returns the number of bytes currently in the RX FIFO |
| 23 | rx_fifo_clear | W | - | **Clear the RX FIFO**<br>1: Clear the RX FIFO and any pending RX FIFO flags in SPI3_INTFL. This should be done when the RX FIFO is inactive.<br>Writing a 0 has no effect. |
| 22 | rx_fifo_en | R/W | 0 | **RX FIFO Enabled**<br>0: RX FIFO disabled<br>1: RX FIFO enabled |
| 21 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 20:16 | rx_fifo_level | R/W | 0x00 | **RX FIFO Threshold Level**<br>When the RX FIFO has more than this many bytes, a DMA request is triggered, and *SPI3_INT_FL.rx_thresh* is set. Valid values are 0 to 30.<br>*Note: 31 is an invalid setting and reserved for future use.* |
| 15 | tx_dma_en | R/W | 0 | **TX DMA Enable**<br>0: TX DMA is disabled. Any pending DMA requests are cleared<br>1: TX DMA is enabled |
| 14 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 13:8 | tx_fifo_cnt | RO | 0 | **Number of Bytes in the TX FIFO**<br>Read this field to determine the number of bytes currently in the TX FIFO. |
| 7 | tx_fifo_clear | R/W | 0 | **TX FIFO Clear**<br>Set this bit to clear the TX FIFO and all TX FIFO flags in the *SPI3_INT_FL* register.<br>*Note: The TX FIFO should be disabled (SPI3_DMA.tx_fifo_en = 0) prior to setting this field.*<br>*Note: Setting this field to 0 has no effect.* |
| 6 | tx_fifo_en | R/W | 0 | **TX FIFO Enabled**<br>Set this field to enable the TX FIFO.<br>0: TX FIFO disabled<br>1: TX FIFO enabled |
| 5 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 4:0 | tx_fifo_level | R/W | 0x10 | **TX FIFO Threshold Level**<br>When the TX FIFO fills past the threshold a DMA request is triggered and *SPI3_INT_FL.tx_thresh* is set. |

*Table 19-9: SPI3 Interrupt Status Flags Registers*

| SPI3 Interrupt Status Flags Register | | SPI3_INT_FL | | [0x0020] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15 | rx_und | R/1 | 0 | **RX FIFO Underrun Flag**<br>Set when a read is attempted from an empty RX FIFO. |
| 14 | rx_ovr | R/W1C | 0 | **RX FIFO Overrun Flag**<br>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. |
| 13 | tx_und | R/W1C | 0 | **TX FIFO Underrun Flag**<br>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**<br>Set when a write is attempted to a full TX FIFO. |
| 11 | m_done | R/W1C | 0 | **Master Data Transmission Done Flag**<br>Set if SPI is in Master Mode, and all transactions have completed. |
| 10 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 9 | abort | R/W1C | 0 | **Slave Mode Transaction Abort Detected Flag**<br>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**<br>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**<br>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**<br>Set when the RX FIFO exceeds the value in *SPI3_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**<br>Set when the TX FIFO is less than the value in *SPI3_DMA.tx_fifo_*level. |

*Table 19-10: SPI3 Interrupt Enable Registers*

| SPI3 Interrupt Enable Register | | SPI3_INT_EN | | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| SPI3 Interrupt Enable Register | | | SPI3_INT_EN | [0x0024] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 15 | rx_und | R/W | 0 | **RX FIFO Underrun Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 14 | rx_ovr | R/W | 0 | **RX FIFO Overrun Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 13 | tx_und | R/W | 0 | **TX FIFO Underrun Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 12 | tx_ovr | R/W | 0 | **TX FIFO Overrun Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 11 | m_done | R/W | 0 | **Master Data Transmission Done Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 10 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 9 | abort | R/W | 0 | **Slave Mode Abort Detected Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 8 | fault | R/W | 0 | **Multi-Master Fault Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 7:6 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 5 | ssd | R/W | 0 | **Slave Select Deasserted Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 4 | ssa | R/W | 0 | **Slave Select Asserted Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 3 | rx_full | R/W | 0 | **RX FIFO Full Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 2 | rx_thresh | R/W | | **RX FIFO Threshold Level Crossed Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 1 | tx_empty | R/W | 0 | **TX FIFO Empty Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |
| 0 | tx_thresh | R/W | 0 | **TX FIFO Threshold Level Crossed Interrupt Enable**<br>0: Interrupt is disabled<br>1: Interrupt is enabled |

*Table 19-11: SPI3 Wakeup Status Flags Registers*

| SPI3 Wakeup Flags Register | | | SPI3_WAKE_FL | [0x0028] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:4 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3 | rx_full | R/W1C | 0 | **Wake on RX FIFO Full Flag**<br>0: Wake condition has not occurred.<br>1: Wake condition occurred. |
| 2 | rx_thresh | R/W1C | 0 | **Wake on RX FIFO Threshold Level Crossed Flag**<br>0: Wake condition has not occurred.<br>1: Wake condition occurred. |
| 1 | tx_empty | R/W1C | 0 | **Wake on TX FIFO Empty Flag**<br>0: Wake condition has not occurred.<br>1: Wake condition occurred. |
| 0 | tx_thresh | R/W1C | 0 | **Wake on TX FIFO Threshold Level Crossed Flag**<br>0: Wake condition has not occurred.<br>1: Wake condition occurred. |

*Table 19-12: SPI3 Wakeup Enable Registers*

| SPI3 Wakeup Enable Register | | | SPI3_WAKE_EN | [0x002C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:4 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 3 | rx_full | R/W | 0 | **Wake on RX FIFO Full Enable**<br>0: Wake event is disabled<br>1: Wake event is enabled. |
| 2 | rx_thresh | R/W | 0 | **Wake on RX FIFO Threshold Level Crossed Enable**<br>0: Wake event is disabled<br>1: Wake event is enabled. |
| 1 | tx_empty | R/W | 0 | **Wake on TX FIFO Empty Enable**<br>0: Wake event is disabled<br>1: Wake event is enabled. |
| 0 | tx_thresh | R/W | 0 | **Wake on TX FIFO Threshold Level Crossed Enable**<br>0: Wake event is disabled<br>1: Wake event is enabled. |

*Table 19-13: SPI3 Slave Select Timing Registers*

| SPI3 Status Register | | | SPI3_STAT | [0x0030] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:1 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| SPI3 Status Register | | | SPI3_STAT | [0x0030] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | busy | R | 0 | **SPI Active Status**<br>0: SPI is not active. In Master Mode, cleared when the last character is set. In Slave Mode, cleared when SS is deasserted.<br>1: SPI is active. In Master Mode, set when transmit starts. In Slave Mode, set when SS is asserted. |

# 20 SPIMSS for I²S

## 20.1 Overview

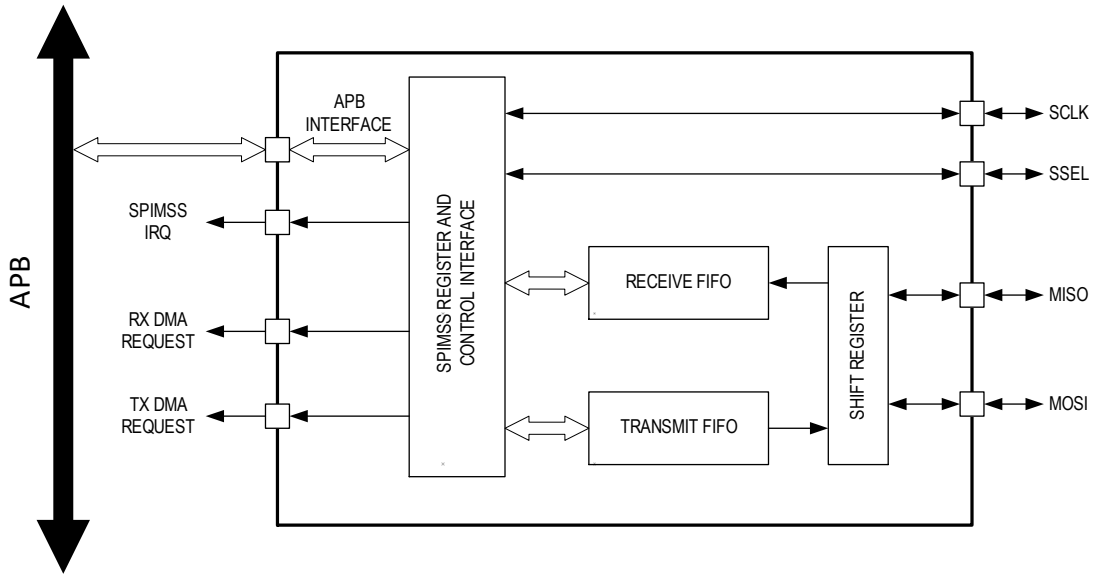
The SPIMSS peripheral provides independent serial communication support for I²S (Inter-IC Sound) for 16-bit mono or stereo audio transfer to or from an external I²S audio codec.

### 20.1.1 *Features*

- I²S mode
  - 16-bit audio transfer
- I²S Master mode
- I²S Slave mode

The block diagram shows the SPIMSS external interface signals, control unit, receive and transmit FIFOs, and single shift register common to the transmit and receive data path. Each time that an SPIMSS transfer completes, the received character is transferred to the receive FIFO.

*Figure 20-1. SPIMSS Block Diagram*




The SPIMSS may be configured as either a SPI master (in single or multi-master systems) or a SPI slave. An SPI system has a single master and one or more slaves for any given transaction.

*Figure 20-2. SPI Single-Master, Single-Slave*

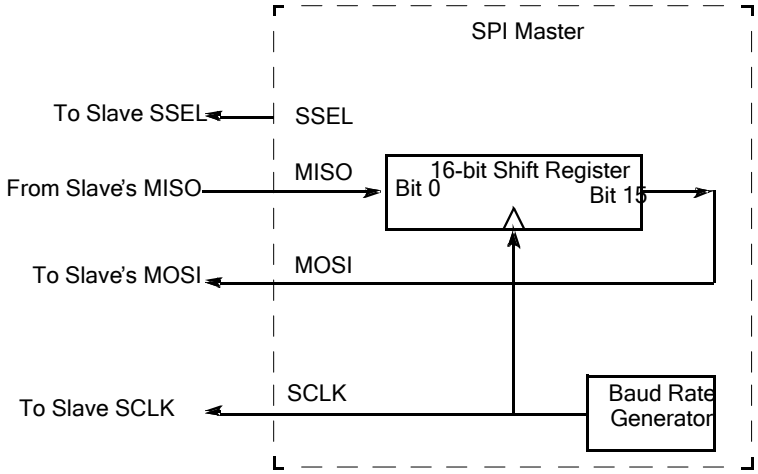



Figure 20-3. SPI Multi-Master, Multi-Slave

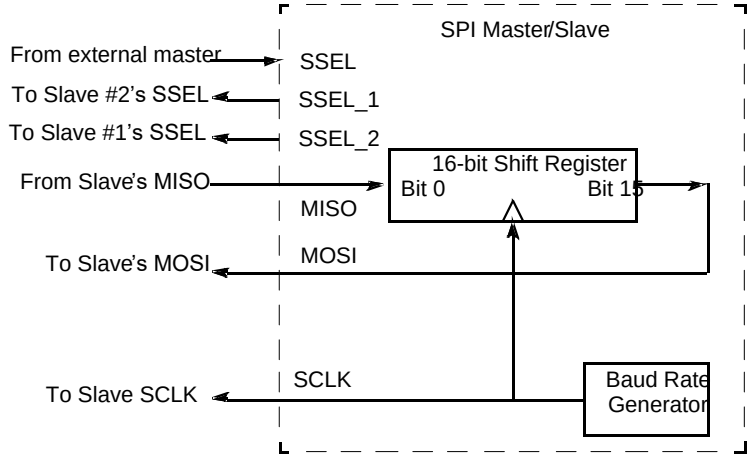
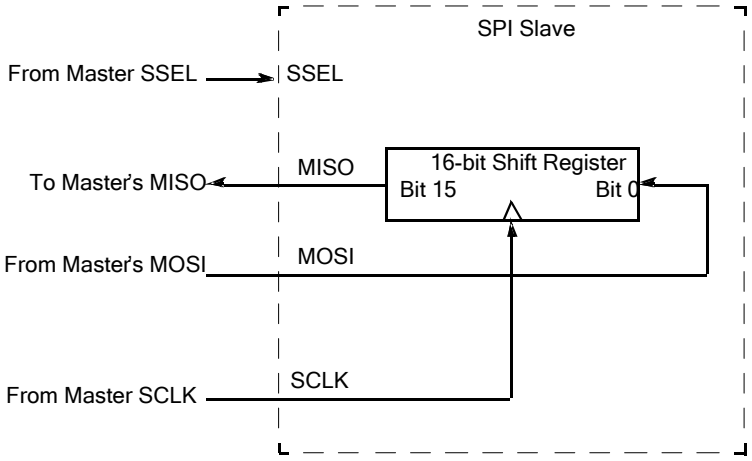



Figure 20-4. SPI Slave




### *20.1.1.1 I²S System*

In I²S mode the SSEL output is controlled by hardware and distinguishes left and right channel audio data. When operating as the I²S master, the SCLK and SSEL signals are outputs. When operating as the I²S slave, the SCLK and SSEL signals are inputs. This SSEL signal is referred to as word select signal (WS) in the I²S protocol. Normally the WS signal transitions one

SCLK period before the MSB of the audio data word, however if the *SPIMSSn_I2S_CTRL.i2s_lj* bit is set, the audio data word is “left justified” to be in phase with the WS signal.

## 20.2 SPIMSS Clock Phase and Polarity Control

The SPI supports four combinations of SCLK phase and polarity. Clock Polarity (*SPIMSSn_CTRL.clkpol*) selects an active low/high clock and has no effect on the transfer format. Clock Phase (*SPIMSSn_CTRL.phase*) selects one of two fundamentally different transfer formats.

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

*Table 20-1. Clock Phase and Polarity Operation*

| *SPIMSSn_CTRL phase* | *SPIMSSn_CTRL clkpol* | SCLK Transmit Edge | SCLK Receive Edge | SCLK Idle State |
|---|---|---|---|---|
| 0 | 0 | Falling | Rising | Low |
| 0 | 1 | Rising | Falling | High |
| 1 | 0 | Rising | Falling | Low |
| 1 | 1 | Falling | Rising | High |

## 20.3 Data Movement

Data movement can be controlled in one of the following ways:

Software polling the SPIMSSn_INT_FL.*txst* bit (transfer one word at a time) or polling the *SPIMSSn_DMA.tx_fifo_level* or *SPIMSSn_DMA.rx_fifo_level* fields (can transfer up to 8 characters at a time).

The SPIMSSn_CTRL.*irqe* bit can be set to enable data and error interrupts. The SPIMSSn_INT_FL.*str* bit may be used if desired to force a “startup” data interrupt. A data interrupt will be generated on completion of each character transfer.

DMA control of data transferred is enabled via the *SPIMSSn_DMA.rx_dma_en* and/or *SPIMSSn_DMA.tx_dma_en* bits. The *SPIMSSn_DMA.tx_fifo_level* and *SPIMSSn_DMA.rx_fifo_level* control when DMA requests are asserted. When DMA is enabled, data interrupts are disabled (error interrupts will still occur). DMA operation is beneficial for block transfers as the CPU only needs to service one DMA interrupt per block of data versus one interrupt for each character transferred.

The SPIMSS Data register is used for transferring data in both incoming and outgoing directions.

For incoming data, the receive data is shifted into an internal shift register. Once a full character has been shifted in, the character is automatically moved into the Receive FIFO. The Receive FIFO data is read through the SPIMSSn Data Register.

For outgoing data, the transmit data written to the SPIMSSn Data Register is written into the Transmit FIFO. When the shift register is empty, data is automatically moved into the shift register from the Transmit FIFO.

*Note: When the SPIMSS is not actively transmitting or receiving data (SPIMSSn_CTRL.start = 0), data written to the SPIMSSn Data Register is stored in the FIFO as long as it is not full. Any data in the FIFO when the SPIMSS start is set to 1 is transmitted immediately. Flush the FIFO at any time by setting the SPIMSSn_INT_FL.tx_fifo_clr bit to 1.*

With the SPI configured as a master, writing data to this register initiates the data transmission. With the SPIMSS is configured as a SPI slave, writing data to this register loads the shift register in preparation for the next data transfer with the external master. In either the master or slave mode, when the transmit FIFO is full, writes to this register are ignored and the Transmit Overrun error flag, *SPIMSSn_INT_FL.tovr*, is set in the SPIMSS Interrupt register.

Data is shifted out starting with bit 15. The last bit received will reside in bit position 0. When the character length is less than 16 bits (as set by the *SPIMSSn_MOD.numbits* field), the transmit character must be left justified in the SPIMSSn Data Register. A received character of less than 16 bits will be right justified (last bit received will be in bit position 0). For

example, if the SPIMSS is configured for 4-bit characters, the transmit characters must be written to *SPIMSSn_DATA[15:12]* and the received characters are read from *SPIMSSn_DATA[3:0]*.

The software overhead to left justify the transmit data can be eliminated by setting the *SPIMSSn_MOD.tx_lj* bit to. When *SPIMSSn_MOD.tx_lj* = 1, transmit data is always written by software or DMA to *SPIMSSn_DATA* in right justified form and hardware performs the left justify according to *SPIMSSn_MOD.numbits* when the shift register is loaded. For the 4-bit character example, when *SPIMSSn_MOD.tx_lj* = 1, transmit data is written to *SPIMSSn_DATA[3:0]* and hardware shifts these to bits to *SPIMSSn_DATA[15:12]* when the shift register is loaded. The *SPIMSSn_MOD.tx_lj* bit has no effect on receive data which is always right justified.

## 20.4 I²S (Inter-IC Sound) Mode

The SPIMSS block is configured for I²S mode operation by setting:

- *SPIMSSn_I2S_CTRL.i2s_en* = 1
- *SPIMSSn_CTRL***Error! Reference source not found.**.*phase* = 0
- *SPIMSSn_CTRL.clkpol* = 0
- *SPIMSSn_MOD numbits* = 0 (to select 16-bit characters)
- *SPIMSSn_CTRL.start* = 1

The *mmen* and *ssio* bits are set in accordance with either master or slave mode of operation. The SSV bit is ignored by hardware in I²S mode. In I²S, the master hardware sources SSEL (known as WS in I²S protocol) and SCLK. In this mode SSEL toggles between consecutive audio words. SSEL=0 indicates left channel data and SSEL=1 indicates right channel audio data.

The receive and/or transmit DMA channels must be enabled when operating in I²S mode. Typically, audio data will only flow in one direction as defined by the *SPIMSSn_DMA.rx_dma_en* or *SPIMSSn_DMA.tx_dma_en* bits, however audio data may be transferred in both directions simultaneously if desired. Data in the transmit buffer should be initialized with the first 16-bit character containing a left channel audio sample, then alternating right and left channel 16-bit audio samples. When audio data is being received, the first sample written into the receive buffer is the left channel audio sample.

### 20.4.1 *Mute*

The *SPIMSSn_I2S_CTRL.i2s_mute* bit can be set by software asynchronously with respect to a DMA transfer to mute the transmit output. At the beginning of the next left channel audio sample after *SPIMSSn_I2S_CTRL.i2s_mute* is set to 1, the DMA and FIFO accesses continue, however, the data read from the transmit FIFO is discarded and replaced with zeroes. When the *SPIMSSn_I2S_CTRL.i2s_mute* is set to 0, the transmit output resumes at the beginning of the next left channel audio sample.

### 20.4.2 *Pause*

The *SPIMSSn_I2S_CTRL.i2s_pause* bit can be set by software asynchronously with respect to DMA transfers to halt the DMA and any FIFO accesses. At the beginning of the next left channel audio sample after *SPIMSSn_I2S_CTRL.i2s_pause* is set, both transmit and receive DMA and FIFO accesses are halted and the transmit data is forced to zero. At the beginning of the next left channel audio sample after *SPIMSSn_I2S_CTRL.i2s_pause* is set to 0, the DMA access resumes from the point it was previously halted. Pause takes precedence over mute.

### 20.4.3 *Mono*

The *SPIMSSn_I2S_CTRL.i2s_mono* bit selects single channel audio data or stereo format. Set *SPIMSSn_I2S_CTRL.i2s_mono* to 1 to enable single channel mono audio. In mono mode each transmit data word read from the transmit FIFO is duplicated for both left and right channel output words.. The receive channel will read the data from the left channel (SSEL = 0) and ignore data in the right channel. This allows DMA buffers for mono mode to be one-half the size of DMA buffers for stereo mode.

### 20.4.4 *Left Justify*

The *SPIMSSn_I2S_CTRL.i2s_lj* bit selects the phase of the SSEL signal versus the data. When *SPIMSSn_I2S_CTRL.i2s_lj* = 0 (normal I²S mode), the audio data lags the SSEL signal by one SCLK period. When *SPIMSSn_I2S_CTRL.i2s_lj* = 1, the audio data is "left justified" so that it is in sync with the SSEL signal.

*Figure 20-5: I²S Mode Right Justify Mode*

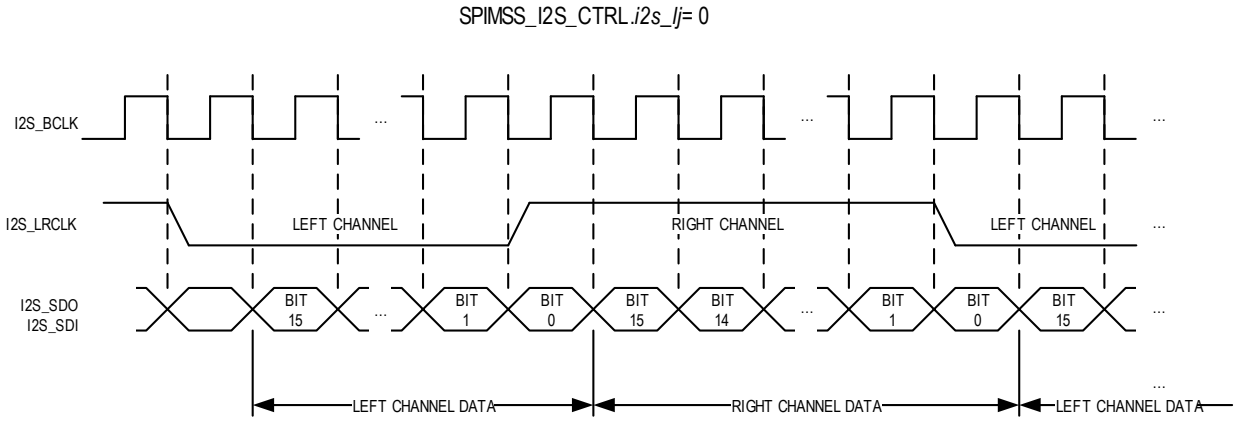
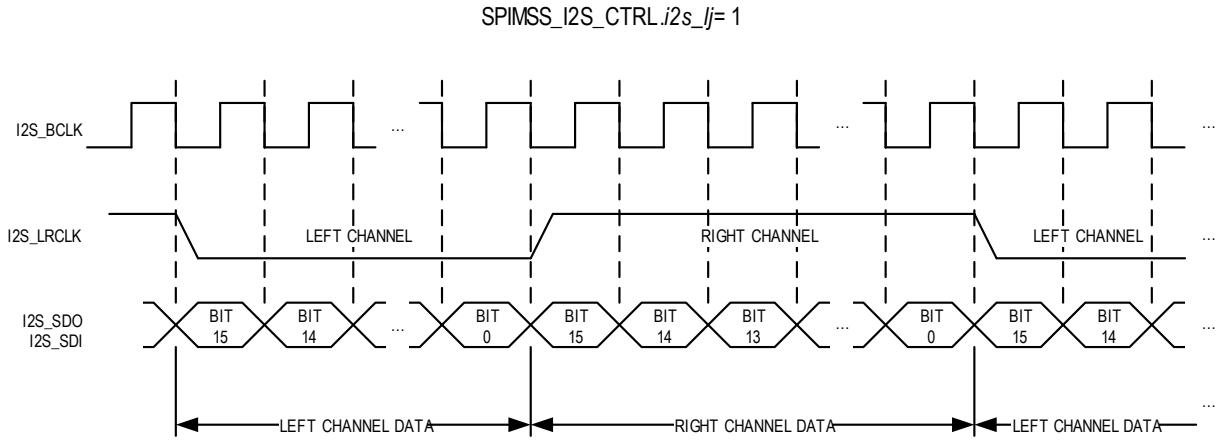



*Figure 20-6: I²S Mode Left Justify Mode*




## 20.5 Error Detection

The SPIMSS contains error detection logic to support the I²S communication protocol and recognize when communication errors have occurred. If the IRQE bit is set, error conditions will generate an interrupt. The SPIMSS Interrupt Flag Register indicates which error has been detected.

### 20.5.1 *Transmit Overrun*

A transmit overrun error indicates a write to the FIFO was attempted when the internal transmit FIFO was full in either master or slave mode. An overrun condition sets the *SPIMSSn_INT_FL.tovr* bit to 1. Writing a 1 to *SPIMSSn_INT_FL.tovr* clears this error flag.

A transmit FIFO overrun in I²S mode may result in mixing left and right channel data. Software should reinitialize the DMA channel and data buffer and restart the I²S transfer.

### 20.5.2 *Mode Fault (Multi-Master Collision)*

A mode fault indicates more than one master is trying to communicate at the same time (a multi-master collision). The mode fault is detected when an enabled master's SSEL input pin is asserted low. A mode fault sets the *SPIMSSn_INT_FL.col* bit to 1. Writing a 1 to *SPIMSSn_INT_FL.col* clears this error flag.

This error interrupt will not occur in I²S mode.

### 20.5.3 *Slave Mode Abort*

A slave mode abort indicates that the SSEL pin deasserted before all bits in a character were transferred (while operating in slave mode). The next time SSEL asserts, the MISO pin will output *SPIMSSn_DATA[15]*, regardless of where the previous transaction left off. A slave mode abort sets the *SPIMSSn_INT_FL.abt* bit to 1. Writing a 1 to *SPIMSSn_INT_FL.abt* clears this error flag.

This error interrupt will not occur in I²S mode.

### 20.5.4 *Receive Overrun*

A receive overrun error indicates a write to the receive FIFO occurred when the internal receive FIFO was full (in either master or slave mode). An overrun sets *SPIMSSn_INT_FL.rovr* = 1. Writing a 1 to *SPIMSSn_INT_FL.rovr* clears this error flag.

A receive FIFO overrun in I²S mode may result in mixing left and right channel data. Software should reinitialize the DMA channel and data buffer and restart the I²S transfer.

## 20.6 SPIMSS Interrupts

When the SPI interrupt is enabled, *SPIMSSn_CTRL.irqe* = 1, the SPIMSS generates an interrupt an enabled interrupt condition occurs. The interrupt condition is indicated by the *SPIMSSn_INT_FL.irq* bit. Writing a 1 to the *SPIMSSn_INT_FL.irq* bit clears the pending SPIMSS interrupt request.

### 20.6.1 *Data Interrupt*

A data interrupt occurs when the transmit character has been fully moved out of the shift register AND the Transmit FIFO is empty (in either master or slave I²S mode). Since transmit and receive are always interlocked, there is no need for a separate receive interrupt. If either transmit or receive DMA is enabled via the *SPIMSSn_DMA.rx_dma_en* and *SPIMSSn_DMA.tx_dma_en* bits, the data interrupt will not occur, however error interrupts are still enabled when using DMA. A data interrupt is indicated by *SPIMSSn_INT_FL.irq* = 1 and no error interrupt flags set.

### 20.6.2 *Forced Interrupt*

Force an SPIMSS interrupt to start a transaction by writing 1 to the *SPIMSSn_CTRL.str* bit.

### 20.6.3 *Error Condition Interrupt*

If any of the SPIMSS error conditions occurs as described in the *Error Detection* section, *above*, the corresponding error bit in the *SPIMSSn_INT_FL* register and the *SPIMSSn_INT_FL.irq* bit are set and a SPIMSS interrupt (IRQ) is generated. The error status bits and the *SPIMSSn_INT_FL.irq* bit should be cleared at the same time by writing a 1 to the corresponding bits.

### 20.6.4 *Bit Rate Generator Timeout Interrupt*

When the SPIMSS is disabled an SPIMSS interrupt can be generated using the SPIMSS Bit Rate Generator timeout. Enable the SPIMSS Bit Rate Generator by setting *SPIMSSn_CTRL.birq* to 1 to use the SPIMSS BRG as a timer.

## 20.7 SPIMSS Bit Rate Generator

### 20.7.1 *Slave Mode*

The Bit Rate Generator is not used in I²S slave mode. When the SPIMSS is configured as an I²S slave, the I2S_BCLK input frequency must be less than or equal to $f_{PCLK}/8$.

### 20.7.2 *Master Mode*

In I²S master mode, the Bit Rate Generator creates a lower frequency clock (I2S_BCLK) for data transmission synchronization between the master and the external slave. The input to the Bit Rate Generator is the peripheral clock, PCLK. The SPIMSS Bit Rate Register is a 16-bit reload value for the SPIMSS Bit Rate Generator. The reload value must be 2 or greater for I²S operation with a maxim I2S_BCLK frequency of $f_{PCLK}/4$. The I²S bit rate is calculated using *Equation 20-1*, *below* (for the special case *SPIMSSn_BRG.div* = 0, substitute $2^{16}$ for *SPIMSSn_BRG.div* in the equation).

*Equation 20-1: SPIMSS Bit Rate Equation*

$$\text{SPI Bit Rate}\left(\text{bits}/\text{sec}\right) = \left(\frac{\text{f}_{\text{PCLK}}}{2 \times \text{SPIMSS\_BRG.div}}\right)$$

$$\text{For SPIMSSn\_BRG.div} = 0, \text{substitute } 2^{16}$$

### 20.7.3 *Timer Mode*

When the SPIMSS is disabled the Bit Rate Generator can function as a continuous mode 16-bit timer with interrupt on timeout. Configure the Bit Rate Generator as a timer with interrupt on timeout using the following sequence:

1. Set *SPIMSSn_CTRL.start* = 0 to disable the SPIMSS SPI or I²S activity.
2. Load the desired 16-bit count value into the SPIMSS Bit Rate Register field, *SPIMSSn_BRG.div*.
3. Set *SPIMSSn_CTRL.birq* = 1 to enable the bit rate generator.
4. When the Bit Rate Generator timer expires the *SPIMSSn_INT_FL.irq* flag is set by hardware.

## 20.8 SPIMSS Registers

The SPIMSSn instance is controlled by a block of registers assigned to this peripheral. Refer to *Table 2-1: APB Peripheral Base Address Map* SPIMSS Base Peripheral Address.

*Table 20-2: SPIMSSn Register Offsets, Access and Descriptions*

| Offset | Register Name | Access | Description |
|---|---|---|---|
| [0x0000] | *SPIMSSn_DATA* | R/W | SPIMSS Data Register |
| [0x0004] | *SPIMSSn_CTRL* | R/W | SPIMSS Control Register |
| [0x0008] | *SPIMSSn_INT_FL* | R/W | SPIMSS Interrupt Flag Register |
| [0x000C] | *SPIMSSn_MOD* | R/W | SPIMSS Mode Register |
| [0x0014] | *SPIMSSn_BRG* | R/W | SPIMSS Bit Rate Register |
| [0x0018] | *SPIMSSn_DMA* | R/W | SPIMSS DMA Register |
| [0x001C] | *SPIMSSn_I2S_CTRL* | R/W | SPIMSS I²S Control Register |

## 20.9 SPIMSS Register Details

*Table 20-3. SPIMSS Data Register*

| SPIMSS Data Register | | | SPIMSSn_DATA | [0x0000] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15:0 | data | R/W | 0 | **SPIMSS Data**<br>Refer to the *Data Movement* section for details. |

Table 20-4: SPIMSS Control Register

| SPIMSSn Control Register | | | SPIMSSn_CTRL | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:8 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7 | irqe | R/W | 0 | **Interrupt Request Enable**<br>Set to enable interrupts for the SPIMSS peripheral.<br>0: SPI interrupts are disabled.<br>1: SPI interrupts are enabled. Interrupt requests are sent to the Interrupt Controller<br>*Note: If transmit or receive DMA is enabled, the transmit data complete interrupt is disabled, but other interrupt sources are enabled.* |
| 6 | str | R/W | 0 | **Start SPI Interrupt**<br>Setting this bit starts a SPIMSS interrupt request. Setting this bit also sets *SPIMSSn_INT_FL.irq* to 1. Setting this bit forces the SPIMSS to send an interrupt request to the Interrupt Controller if *SPIMSSn_CTRL.irqe* = 1. Write 0 to clear this bit or by clearing all pending interrupts in the *SPIMSSn_INT_FL* register. |
| 5 | birq | R/W | 0 | **Bit Rate Generator Timer Interrupt Request**<br>Enable the Bit Rate Generator and the Bit Rate Generator Interrupt if the SPIMSS is stopped, *SPIMSSn_CTRL.enable* = 0.<br>0: Bit Rate Generation timer function disabled.<br>1: Enable the Bit Rate Generator and the associated Bit Rate Generator Interrupt.<br>*Note: If SPIMSSn_CTRL.enable = 1, this bit has no effect.* |
| 4 | phase | R/W | 0 | **Phase Select**<br>Refer to the *SPIMSS Clock Phase and Polarity Control* section for details. |
| 3 | clkpol | R/W | 0 | **Clock Polarity**<br>Sets the idle state for the SCK clock pin after a character transaction.<br>0: SCK idles Low (0) after character transmission/reception.<br>1: SCK idles High (1) after character transmission/reception. |
| 2 | od_out_en | R/W | 0 | **Wired OR (Open Drain) Enable**<br>Set to enable wired OR for the SPI signal pins (SPI1_SCK, SPI1_SS0, SPI1_MOSI, SPI1_MISO).<br>0: Wired OR configuration disabled.<br>1: Wired OR configuration enabled. |
| 1 | mmen | R/W | 0 | **SPI Master Mode Enable**<br>Set this field to enable Master Mode for SPI.<br>0: SPI set to slave mode operation<br>1: SPI set to master mode operation |

| SPIMSSn Control Register | | | SPIMSSn_CTRL | [0x0004] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | start | R/W | 0 | **SPI Start**<br>Set this field to start operation of the SPIMSSn port as configured. If the FIFOs contain data, the data is considered valid by the SPIMSSn peripheral and is used.<br>0: Stop SPIMSS operation.<br>1: Start SPIMSS transaction as configured.<br>*Note: This bit should be set to 1 only after the SPIMSSn is configured for operation. Setting this bit to 0 does not reset or change any configuration of the SPIMSSn peripheral and does not affect any data in the FIFOs.* |

*Table 20-5: SPIMSS Interrupt Flag Register*

| SPIMSSn Interrupt Flag Register | | | SPIMSSn_INT_FL | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:8 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7 | irq | R/W1C | 0 | **SPI Interrupt Request Flag**<br>This bit is set by hardware when an SPI interrupt request is pending. Write 1 to clear.<br>0: No SPI interrupt request is pending<br>1: An SPI interrupt request is pending<br>*Note: This field cannot be cleared unless all interrupt flags in this register are cleared.* |
| 6 | tovr | R/W1C | 0 | **Transmit Overrun Flag**<br>This bit is set by hardware when a transmit FIFO overrun has occurred. Write 1 to clear.<br>0: No SPI interrupt request is pending<br>1: An SPI interrupt request is pending |
| 5 | col | R/W1C | 0 | **Collision Flag**<br>This bit is set by hardware when a multi-master collision (mode fault) occurs. Write 1 to clear.<br>0: No multi-master collision has occurred<br>1: A multi-master collision has occurred |
| 4 | abt | R/W1C | 0 | **Slave Mode Transaction Abort Flag**<br>This bit is set by hardware when a slave mode transaction abort occurs. Write 1 to clear.<br>0: No slave mode transaction abort has occurred<br>1: A slave mode transaction abort has occurred |
| 3 | rovr | R/W1C | 0 | **Receive Overrun Flag**<br>This bit is set by hardware when a receive FIFO overrun occurs. Write 1 to clear.<br>0: No FIFO overrun has occurred<br>1: A FIFO overrun has occurred. |
| 2 | tund | R/W1C | 0 | **Transmit Underrun Flag**<br>This bit is set by hardware to indicate a transmit FIFO underrun has occurred. Write 1 to clear.<br>0: No FIFO underrun has occurred<br>1: A FIFO underrun has occurred |
| 1 | txst | RO | 0 | **Transmit Status**<br>This field reads 1 if a SPIMSS data transmission is currently in progress.<br>0: No data transmission currently in progress.<br>1: Data transmission currently in progress |

| SPIMSSn Interrupt Flag Register | | | SPIMSSn_INT_FL | [0x0008] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 0 | slas | R/W | 0 | **Slave Select**<br>If the SPI is in slave mode, this bit indicates if the SPI is selected. If the SPI is in master mode this bit has no meaning.<br>0: Slave SPI is selected<br>1: Slave SPI is not selected |

*Table 20-6: SPIMSS Mode Register*

| SPIMSSn Mode Register | | | SPIMSSn_MOD | [0x000C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:8 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 7 | tx_align | R/W | 0 | **Transmit Data Alignment**<br>Selects left or right alignment when data is loaded into the *SPIMSSn_DATA.data* field for transmission if the character size is less than 16-bits.<br>0: Data is LSB aligned with the unused bits set to 0 up to the MSB (right aligned)<br>1: Data is MSB aligned with the unused bits set to 0 down to the LSB (left aligned) |
| 6 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 5:2 | numbits | R/W | 0 | **Number of Data Bits per Character**<br>This field contains the number of bits to shift for each character transfer. Refer to *Data Movement* section for information on valid bit positions when the character length is less than 16-bits.<br>0b0000: 16-bits<br>0b0001: 1-bits<br>0b0010: 2-bits<br>…<br>0b1110: 14-bits<br>0b1111: 15-bits<br>*Note: Setting this field to 0 (default) sets the number of bits per character to 16.* |
| 1 | ssel_mode | R/W | 0 | **Slave Select Input/Output Mode**<br>Setting this field to 1 sets the slave select pin, SPI1_SS0, as an output. Clearing this field sets the slave select pin, SPI1_SS0, to an input.<br>0 = The SPI1_SS0 pin is configured as an input.<br>1 = The SPI1_SS0 pin is configured as an output<br>*Note: This field is only used if the SPIMSSn is in SPI Master mode (SPIMSSn_CTRL.mmen = 1).* |
| 0 | ssv | R/W | 0 | **Slave Select Value**<br>This indicates the value of the I2S_LRCLK pin if the SPIMSSn slave select pin is configured as an output, *SPIMSSn_MOD.ssel_mode* = 1, writing this field drives the pin to the value written. If the slave select pin is set to an input *SPIMSSn_MOD.ssel_mode* = 0, reading this field returns the level of the slave select pin. |

*Table 20-7: SPIMSS Bit Rate Generator Register*

| SPIMSSn Bit Rate Generator Register | | | SPIMSSn_BRG | [0x0014] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:16 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 15:0 | div | R/W | 0 | **Bit Rate Reload Value**<br>The SPI Bit Rate register is a 16-bit reload value for the SPI Bit Rate Generator. The reload value must be greater than or equal to 2 for proper SPI or $I^2S$ operation (maximum bit rate is $f_{PCLK}$ divided by 4). Refer to the section *SPIMSS Bit Rate Generator* for calculation. |

*Table 20-8: SPIMSS DMA Register*

| SPIMSSn DMA Register | | | SPIMSSn_DMA | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31 | rx_dma_en | R/W | 0 | **Receive DMA Enable**<br>Disabling clears any active request to the DMA controller.<br>0: Disable RX DMA requests<br>1: Enable RX DMA requests |
| 30:28 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 27:24 | rx_fifo_cnt | R/W | 0 | **Receive FIFO Count**<br>0: RX FIFO empty (0 entries)<br>1: RX FIFO contains 1 entry<br>2: RX FIFO contains 2 entries<br>3: RX FIFO contains 3 entries<br>…<br>15: RX FIFO contains 15 entries |
| 23:21 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 20 | rx_fifo_clr | R/W | 0 | **Receive FIFO Clear**<br>Write 1 to reset the Receive FIFO. Writing 0 has no effect.<br>0: Ignored<br>1: Reset Receive FIFO |
| 19 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 18:16 | rx_fifo_lvl | R/W | 0 | **Receive FIFO Level**<br>Sets the RX FIFO DMA request threshold. This configures the number of filled RX FIFO entries before activating an RX DMA request.<br>0: Request Receive DMA when RX FIFO contains 1 entry<br>1: Request Receive DMA when RX FIFO contains 2 entries<br>2: Request Receive DMA when RX FIFO contains 3 entries<br>…<br>7: Request Receive DMA when RX FIFO contains 8 entries |
| 15 | tx_dma_en | R/W | 0 | **Transmit DMA Enable**<br>Disabling clears any active request to the DMA controller.<br>0: Disable TX DMA requests<br>1: Enable TX DMA requests |
| 14:12 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |

| SPIMSSn DMA Register | | | SPIMSSn_DMA | [0x0018] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 11:8 | tx_fifo_cnt | R/W | 0 | **Transmit FIFO Count**<br>0: TX FIFO empty (0 entries)<br>1: TX FIFO contains 1 entry<br>2: TX FIFO contains 2 entries<br>3: TX FIFO contains 3 entries<br>…<br>15: TX FIFO contains 15 entries |
| 7:5 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 4 | tx_fifo_clr | R/W | 0 | **Transmit FIFO Clear**<br>Write 1 to reset the Receive FIFO. Writing 0 has no effect.<br>0: Ignored<br>1: Reset Receive FIFO |
| 3 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 2:0 | tx_fifo_lvl | R/W | 0 | **Transmit FIFO Level**<br>Sets theTX FIFO DMA request threshold. This configures the number of empty TX FIFO entries before activating a Transmit DMA request.<br>0: Request Transmit DMA when TX FIFO has 1 free entry.<br>1: Request Transmit DMA when TX FIFO has 2 free entries<br>2: Request Transmit DMA when TX FIFO has 3 free entries<br>…<br>7: Request Transmit DMA when TX FIFO has 8 free entries |

*Table 20-9: SPIMSS I²S Control Register*

| SPIMSSn I²S Control Register | | | SPIMSSn_I2S_CTRL | [0x001C] |
|---|---|---|---|---|
| **Bits** | **Name** | **Access** | **Reset** | **Description** |
| 31:5 | - | R/W | 0 | **Reserved for Future Use**<br>Do not modify this field. |
| 4 | i2s_lj | R/W | 0 | **I²S Left Justify**<br>0: Normal I²S audio protocol - audio data lags left/right channel signal by one SCLK period.<br>1: Audio data is synchronized with SSEL (left/right channel signal). |
| 3 | i2s_mono | R/W | 0 | **I²S Monophonic Audio Mode**<br>Set this field to enable monophonic audio mode. In this mode, each transmit data word is replicated on both left and right channels. Receive data is taken from left channel, right channel receive data is ignored.<br>0 - Stereophonic audio.<br>1 - Monophonic audio format |
| 2 | i2s_pause | R/W | 0 | **I²S Pause Transmit/Receive**<br>0: Normal transmission/reception.<br>1: Halt transmit and receive FIFO and DMA accesses, transmit 0. |
| 1 | i2s_mute | R/W | 0 | **I²S Mute Transmit**<br>0: Normal transmit.<br>1: Transmit data is replaced with 0 |
| 0 | i2s_en | R/W | 0 | **I²S Mode Enable**<br>Set to enable I²S mode.<br>0: I²S mode is disabled.<br>1: I²S mode enabled. |

# 21 Trademarks

1-Wire is a registered trademark of Maxim Integrated Products, Inc.

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

Cortex is a registered trademark of Arm Limited.

Cypress is a registered trademark of Cypress Semiconductor Corporation.

HyperBus is a registered trademark of Spansion LLC.

HyperFlash is a registered trademark of Spansion LLC.

HyperRAM is a registered trademark of Spansion LLC.

iButton is a registered trademark of Maxim Integrated Products, Inc.

Spansion is a registered trademark of Spansion LLC.

Xccela is a registered trademark of Micron Technology, Inc.

# 22 Revision History

| REVISION NUMBER | REVISION DATE | DESCRIPTION |
|---|---|---|
| 0 | 9/18 | Initial release |