
# Stellaris® LM3S828 Microcontroller

## DATA SHEET

DS-LM3S828-9102

# Copyright



PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

Texas Instruments Incorporated
108 Wild Basin, Suite 350
Austin, TX 78746
http://www.ti.com/stellaris
http://www-k.ext.ti.com/sc/technical-support/product-information-centers.htm

# Table of Contents

# List of Figures

# List of Tables

# List of Registers

# Revision History

The revision history table notes changes made between the indicated revisions of the LM3S828 data sheet.

**Table 1. Revision History**

| Date | Revision | Description |
|---|---|---|
| January 2011 | 9102 | ■ In **Application Interrupt and Reset Control (APINT)** register, changed bit name from SYSRESETREQ to SYSRESREQ.<br>■ Added DEBUG (Debug Priority) bit field to **System Handler Priority 3 (SYSPRI3)** register.<br>■ Added "Reset Sources" table to System Control chapter.<br>■ Removed mention of false-start bit detection in the UART chapter. This feature is not supported.<br>■ Added note that specific module clocks must be enabled before that module's registers can be programmed. There must be a delay of 3 system clocks after the module clock is enabled before any of that module's registers are accessed.<br>■ Changed $I^2C$ slave register base addresses and offsets to be relative to the $I^2C$ module base address of 0x4002.0000 , so register bases and offsets were changed for all $I^2C$ slave registers. Note that the hw_i2c.h file in the StellarisWare Driver Library uses a base address of 0x4002.0800 for the $I^2C$ slave registers. Be aware when using registers with offsets between 0x800 and 0x818 that StellarisWare uses the old slave base address for these offsets.<br>■ Corrected nonlinearity and offset error parameters ($E_L$, $E_D$ and $E_O$) in ADC Characteristics table.<br>■ Added specification for maximum input voltage on a non-power pin when the microcontroller is unpowered ($V_{NON}$ parameter in Maximum Ratings table).<br>■ Additional minor data sheet clarifications and corrections. |
| September 2010 | 7783 | ■ Reorganized ARM Cortex-M3 Processor Core, Memory Map and Interrupts chapters, creating two new chapters, The Cortex-M3 Processor and Cortex-M3 Peripherals. Much additional content was added, including all the Cortex-M3 registers.<br>■ Changed register names to be consistent with StellarisWare® names: the Cortex-M3 **Interrupt Control and Status (ICSR)** register to the **Interrupt Control and State (INTCTRL)** register, and the Cortex-M3 **Interrupt Set Enable (SETNA)** register to the **Interrupt 0-31 Set Enable (EN0)** register.<br>■ Added clarification of instruction execution during Flash operations.<br>■ Modified Figure 7-2 on page 222 to clarify operation of the GPIO inputs when used as an alternate function.<br>■ Added caution not to apply a Low value to PB7 when debugging; a Low value on the pin causes the JTAG controller to be reset, resulting in a loss of JTAG communication.<br>■ In General-Purpose Timers chapter, clarified operation of the 32-bit RTC mode.<br>■ Added missing table "Connections for Unused Signals" (Table 15-5 on page 474).<br>■ In Electrical Characteristics chapter:<br>– Added $I_{LKG}$ parameter (GPIO input leakage current) to Table 17-4 on page 477.<br>– Corrected values for $t_{CLKRF}$ parameter (SSIClk rise/fall time) in Table 17-16 on page 485.<br>■ Added dimensions for Tray and Tape and Reel shipping mediums. |

**Table 1. Revision History *(continued)***

| Date | Revision | Description |
|---|---|---|
| June 2010 | 7393 | ■ Corrected base address for SRAM in architectural overview chapter.<br>■ Clarified system clock operation, adding content to “Clock Control” on page 154.<br>■ In Signal Tables chapter, added table "Connections for Unused Signals."<br>■ In "Reset Characteristics" table, corrected value for supply voltage (VDD) rise time.<br>■ Additional minor data sheet clarifications and corrections. |
| April 2010 | 7004 | ■ Added caution note to the **I²C Master Timer Period (I2CMTPR)** register description and changed field width to 7 bits.<br>■ Added note about $\overline{\text{RST}}$ signal routing.<br>■ Clarified the function of the TnSTALL bit in the **GPTMCTL** register.<br>■ Additional minor data sheet clarifications and corrections. |
| January 2010 | 6712 | ■ In "System Control" section, clarified Debug Access Port operation after Sleep modes.<br>■ Clarified wording on Flash memory access errors.<br>■ Added section on Flash interrupts.<br>■ Changed the reset value of the **ADC Sample Sequence Result FIFO n (ADCSSFIFOn)** registers to be indeterminate.<br>■ Clarified operation of SSI transmit FIFO.<br>■ Made these changes to the Operating Characteristics chapter:<br>– Added storage temperature ratings to "Temperature Characteristics" table<br>– Added "ESD Absolute Maximum Ratings" table<br>■ Made these changes to the Electrical Characteristics chapter:<br>– In "Flash Memory Characteristics" table, corrected Mass erase time<br>– Added sleep and deep-sleep wake-up times ("Sleep Modes AC Characteristics" table)<br>– In "Reset Characteristics" table, corrected supply voltage (VDD) rise time |
| October 2009 | 6438 | ■ The reset value for the **DID1** register may change, depending on the package.<br>■ Deleted MAXADCSPD bit field from **DCGC0** register as it is not applicable in Deep-Sleep mode.<br>■ Deleted reset value for 16-bit mode from **GPTMTAILR**, **GPTMTAMATCHR**, and **GPTMTAR** registers because the module resets in 32-bit mode.<br>■ Made these changes to the Electrical Characteristics chapter:<br>– Removed VSIH and VSIL parameters from Operating Conditions table.<br>– Changed SSI set up and hold times to be expressed in system clocks, not ns.<br>– Revised ADC electrical specifications to clarify, including reorganizing and adding new data.<br>■ Added 48QFN package.<br>■ Additional minor data sheet clarifications and corrections. |

**Table 1. Revision History *(continued)***

| Date | Revision | Description |
|---|---|---|
| July 2009 | 5953 | ■ Clarified Power-on reset and RST pin operation; added new diagrams.<br>■ Added DBG bits missing from **FMPRE** register. This changes register reset value.<br>■ In ADC characteristics table, changed Max value for GAIN parameter from ±1 to ±3 and added $E_{IR}$ (Internal voltage reference error) parameter.<br>■ Corrected ordering numbers.<br>■ Additional minor data sheet clarifications and corrections. |
| April 2009 | 5369 | ■ Added JTAG/SWD clarification (see “Communication with JTAG/SWD” on page 144).<br>■ Added "GPIO Module DC Characteristics" table (see Table 17-4 on page 477).<br>■ Additional minor data sheet clarifications and corrections. |
| January 2009 | 4644 | ■ Incorrect bit type for RELOAD bit field in SysTick Reload Value register; changed to R/W.<br>■ Clarification added as to what happens when the SSI in slave mode is required to transmit but there is no data in the TX FIFO.<br>■ Minor corrections to comparator operating mode tables.<br>■ Additional minor data sheet clarifications and corrections. |
| November 2008 | 4283 | ■ Revised High-Level Block Diagram.<br>■ Corrected descriptions for UART1 signals.<br>■ Additional minor data sheet clarifications and corrections were made. |
| October 2008 | 4149 | ■ Added note on clearing interrupts to the Interrupts chapter:<br>**Note:** It may take several processor cycles after a write to clear an interrupt source in order for NVIC to see the interrupt source de-assert. This means if the interrupt clear is done as the last action in an interrupt handler, it is possible for the interrupt handler to complete while NVIC sees the interrupt as still asserted, causing the interrupt handler to be re-entered errantly. This can be avoided by either clearing the interrupt source at the beginning of the interrupt handler or by performing a read or write after the write to clear the interrupt source (and flush the write buffer)<br>■ Step 1 of the Initialization and Configuration procedure in the ADC chapter states the wrong register to use to enable the ADC clock. Sentence changed to:<br>1. Enable the ADC clock by writing a value of 0x0001.0000 to the **RCGC0** register.<br>■ Additional minor data sheet clarifications and corrections were made. |
| June 2008 | 2972 | Started tracking revision history. |

# About This Document

This data sheet provides reference information for the LM3S828 microcontroller, describing the functional blocks of the system-on-chip (SoC) device designed around the ARM® Cortex™-M3 core.

## Audience

This manual is intended for system software developers, hardware designers, and application developers.

## About This Manual

This document is organized into sections that correspond to each major feature.

## Related Documents

The following related documents are available on the Stellaris® web site at www.ti.com/stellaris:

- *Stellaris® Errata*
- *ARM® Cortex™-M3 Errata*
- *Cortex™-M3 Instruction Set Technical User's Manual*
- *Stellaris® Graphics Library User's Guide*
- *Stellaris® Peripheral Driver Library User's Guide*

The following related documents are also referenced:

- *ARM® Debug Interface V5 Architecture Specification*
- *IEEE Standard 1149.1-Test Access Port and Boundary-Scan Architecture*

This documentation list was current as of publication date. Please check the web site for additional documentation, including application notes and white papers.

## Documentation Conventions

This document uses the conventions shown in Table 2 on page 23.

**Table 2. Documentation Conventions**

| Notation | Meaning |
|---|---|
| **General Register Notation** | |
| **REGISTER** | APB registers are indicated in uppercase bold. For example, **PBORCTL** is the Power-On and Brown-Out Reset Control register. If a register name contains a lowercase n, it represents more than one register. For example, **SRCRn** represents any (or all) of the three Software Reset Control registers: **SRCR0, SRCR1** , and **SRCR2**. |
| bit | A single bit in a register. |
| bit field | Two or more consecutive and related bits. |
| offset 0x*nnn* | A hexadecimal increment to a register's address, relative to that module's base address as specified in Table 2-4 on page 58. |
| Register *N* | Registers are numbered consecutively throughout the document to aid in referencing them. The register number has no meaning to software. |
| reserved | Register bits marked *reserved* are reserved for future use. In most cases, reserved bits are set to 0; however, user software should not rely on the value of a reserved bit. To provide software compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| *yy:xx* | The range of register bits inclusive from xx to yy. For example, 31:15 means bits 15 through 31 in that register. |
| **Register Bit/Field Types** | This value in the register bit diagram indicates whether software running on the controller can change the value of the bit field. |
| RC | Software can read this field. The bit or field is cleared by hardware after reading the bit/field. |
| RO | Software can read this field. Always write the chip reset value. |
| R/W | Software can read or write this field. |
| R/WC | Software can read or write this field. Writing to it with any value clears the register. |
| R/W1C | Software can read or write this field. A write of a 0 to a W1C bit does not affect the bit value in the register. A write of a 1 clears the value of the bit in the register; the remaining bits remain unchanged. This register type is primarily used for clearing interrupt status bits where the read operation provides the interrupt status and the write of the read value clears only the interrupts being reported at the time the register was read. |
| R/W1S | Software can read or write a 1 to this field. A write of a 0 to a R/W1S bit does not affect the bit value in the register. |
| W1C | Software can write this field. A write of a 0 to a W1C bit does not affect the bit value in the register. A write of a 1 clears the value of the bit in the register; the remaining bits remain unchanged. A read of the register returns no meaningful data. This register is typically used to clear the corresponding bit in an interrupt register. |
| WO | Only a write by software is valid; a read of the register returns no meaningful data. |
| **Register Bit/Field Reset Value** | This value in the register bit diagram shows the bit/field value after any reset, unless noted. |
| 0 | Bit cleared to 0 on chip reset. |
| 1 | Bit set to 1 on chip reset. |
| - | Nondeterministic. |
| **Pin/Signal Notation** | |
| [ ] | Pin alternate function; a pin defaults to the signal without the brackets. |
| pin | Refers to the physical connection on the package. |
| signal | Refers to the electrical signal encoding of a pin. |

**Table 2. Documentation Conventions *(continued)***

| Notation | Meaning |
|---|---|
| assert a signal | Change the value of the signal from the logically False state to the logically True state. For active High signals, the asserted signal value is 1 (High); for active Low signals, the asserted signal value is 0 (Low). The active polarity (High or Low) is defined by the signal name (see SIGNAL and $\overline{\text{SIGNAL}}$ below). |
| deassert a signal | Change the value of the signal from the logically True state to the logically False state. |
| $\overline{\text{SIGNAL}}$ | Signal names are in uppercase and in the Courier font. An overbar on a signal name indicates that it is active Low. To assert $\overline{\text{SIGNAL}}$ is to drive it Low; to deassert $\overline{\text{SIGNAL}}$ is to drive it High. |
| SIGNAL | Signal names are in uppercase and in the Courier font. An active High signal has no overbar. To assert SIGNAL is to drive it High; to deassert SIGNAL is to drive it Low. |
| **Numbers** | |
| X | An uppercase X indicates any of several values is allowed, where X can be any legal pattern. For example, a binary value of 0X00 can be either 0100 or 0000, a hex value of 0xX is 0x0 or 0x1, and so on. |
| 0x | Hexadecimal numbers have a prefix of 0x. For example, 0x00FF is the hexadecimal number FF. All other numbers within register tables are assumed to be binary. Within conceptual information, binary numbers are indicated with a b suffix, for example, 1011b, and decimal numbers are written without a prefix or suffix. |

# 1 Architectural Overview

The Stellaris® family of microcontrollers—the first ARM® Cortex™-M3 based controllers—brings high-performance 32-bit computing to cost-sensitive embedded microcontroller applications. These pioneering parts deliver customers 32-bit performance at a cost equivalent to legacy 8- and 16-bit devices, all in a package with a small footprint.

The LM3S828 microcontroller is targeted for industrial applications, including test and measurement equipment, factory automation, HVAC and building control, motion control, medical instrumentation, fire and security, and power/energy.

In addition, the LM3S828 microcontroller offers the advantages of ARM's widely available development tools, System-on-Chip (SoC) infrastructure IP applications, and a large user community. Additionally, the microcontroller uses ARM's Thumb®-compatible Thumb-2 instruction set to reduce memory requirements and, thereby, cost. Finally, the LM3S828 microcontroller is code-compatible to all members of the extensive Stellaris family; providing flexibility to fit our customers' precise needs.

Texas Instruments offers a complete solution to get to market quickly, with evaluation and development boards, white papers and application notes, an easy-to-use peripheral driver library, and a strong support, sales, and distributor network. See “Ordering and Contact Information” on page 511 for ordering information for Stellaris family devices.

## 1.1 Product Features

The LM3S828 microcontroller includes the following product features:

- 32-Bit RISC Performance
  - 32-bit ARM® Cortex™-M3 v7M architecture optimized for small-footprint embedded applications
  - System timer (SysTick), providing a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism
  - Thumb®-compatible Thumb-2-only instruction set processor core for high code density
  - 50-MHz operation
  - Hardware-division and single-cycle-multiplication
  - Integrated Nested Vectored Interrupt Controller (NVIC) providing deterministic interrupt handling
  - 22 interrupts with eight priority levels
  - Memory protection unit (MPU), providing a privileged mode for protected operating system functionality
  - Unaligned data access, enabling data to be efficiently packed into memory
  - Atomic bit manipulation (bit-banding), delivering maximum memory utilization and streamlined peripheral control
- ARM® Cortex™-M3 Processor Core

- Compact core.
- Thumb-2 instruction set, delivering the high-performance expected of an ARM core in the memory size usually associated with 8- and 16-bit devices; typically in the range of a few kilobytes of memory for microcontroller class applications.
- Rapid application execution through Harvard architecture characterized by separate buses for instruction and data.
- Exceptional interrupt handling, by implementing the register manipulations required for handling an interrupt in hardware.
- Deterministic, fast interrupt processing: always 12 cycles, or just 6 cycles with tail-chaining
- Memory protection unit (MPU) to provide a privileged mode of operation for complex applications.
- Migration from the ARM7™ processor family for better performance and power efficiency.
- Full-featured debug solution
  - Serial Wire JTAG Debug Port (SWJ-DP)
  - Flash Patch and Breakpoint (FPB) unit for implementing breakpoints
  - Data Watchpoint and Trigger (DWT) unit for implementing watchpoints, trigger resources, and system profiling
  - Instrumentation Trace Macrocell (ITM) for support of printf style debugging
  - Trace Port Interface Unit (TPIU) for bridging to a Trace Port Analyzer
- Optimized for single-cycle flash usage
- Three sleep modes with clock gating for low power
- Single-cycle multiply instruction and hardware divide
- Atomic operations
- ARM Thumb2 mixed 16-/32-bit instruction set
- 1.25 DMIPS/MHz

■ JTAG

- IEEE 1149.1-1990 compatible Test Access Port (TAP) controller
- Four-bit Instruction Register (IR) chain for storing JTAG instructions
- IEEE standard instructions: BYPASS, IDCODE, SAMPLE/PRELOAD, EXTEST and INTEST
- ARM additional instructions: APACC, DPACC and ABORT
- Integrated ARM Serial Wire Debug (SWD)

- Internal Memory
  - 64 KB single-cycle flash
    - User-managed flash block protection on a 2-KB block basis
    - User-managed flash data programming
    - User-defined and managed flash-protection block
  - 8 KB single-cycle SRAM
- GPIOs
  - 7-28 GPIOs, depending on configuration
  - 5-V-tolerant in input configuration
  - Programmable control for GPIO interrupts
    - Interrupt generation masking
    - Edge-triggered on rising, falling, or both
    - Level-sensitive on High or Low values
  - Bit masking in both read and write operations through address lines
  - Can initiate an ADC sample sequence
  - Pins configured as digital inputs are Schmitt-triggered.
  - Programmable control for GPIO pad configuration
    - Weak pull-up or pull-down resistors
    - 2-mA, 4-mA, and 8-mA pad drive for digital communication
    - Slew rate control for the 8-mA drive
    - Open drain enables
    - Digital input enables
- General-Purpose Timers
  - Three General-Purpose Timer Modules (GPTM), each of which provides two 16-bit timers/counters. Each GPTM can be configured to operate independently:
    - As a single 32-bit timer
    - As one 32-bit Real-Time Clock (RTC) to event capture
    - For Pulse Width Modulation (PWM)
    - To trigger analog-to-digital conversions

  - 32-bit Timer modes
    - Programmable one-shot timer
    - Programmable periodic timer
    - Real-Time Clock when using an external 32.768-KHz clock as the input
    - User-enabled stalling when the controller asserts CPU Halt flag during debug
    - ADC event trigger
  - 16-bit Timer modes
    - General-purpose timer function with an 8-bit prescaler (for one-shot and periodic modes only)
    - Programmable one-shot timer
    - Programmable periodic timer
    - User-enabled stalling when the controller asserts CPU Halt flag during debug
    - ADC event trigger
  - 16-bit Input Capture modes
    - Input edge count capture
    - Input edge time capture
  - 16-bit PWM mode
    - Simple PWM mode with software-programmable output inversion of the PWM signal
- ARM FiRM-compliant Watchdog Timer
  - 32-bit down counter with a programmable load register
  - Separate watchdog clock with an enable
  - Programmable interrupt generation logic with interrupt masking
  - Lock register protection from runaway software
  - Reset generation logic with an enable/disable
  - User-enabled stalling when the controller asserts the CPU Halt flag during debug
- ADC
  - Eight analog input channels
  - Single-ended and differential-input configurations
  - On-chip internal temperature sensor

- Sample rate of one million samples/second
- Flexible, configurable analog-to-digital conversion
- Four programmable sample conversion sequences from one to eight entries long, with corresponding conversion result FIFOs
- Flexible trigger control
  - Controller (software)
  - Timers
  - GPIO
- Hardware averaging of up to 64 samples for improved accuracy
- Converter uses an internal 3-V reference

■ UART

- Two fully programmable 16C550-type UARTs
- Separate 16x8 transmit (TX) and receive (RX) FIFOs to reduce CPU interrupt service loading
- Programmable baud-rate generator allowing speeds up to 3.125 Mbps
- Programmable FIFO length, including 1-byte deep operation providing conventional double-buffered interface
- FIFO trigger levels of 1/8, 1/4, 1/2, 3/4, and 7/8
- Standard asynchronous communication bits for start, stop, and parity
- Line-break generation and detection
- Fully programmable serial interface characteristics
  - 5, 6, 7, or 8 data bits
  - Even, odd, stick, or no-parity bit generation/detection
  - 1 or 2 stop bit generation

■ Synchronous Serial Interface (SSI)

- Master or slave operation
- Programmable clock bit rate and prescale
- Separate transmit and receive FIFOs, 16 bits wide, 8 locations deep
- Programmable interface operation for Freescale SPI, MICROWIRE, or Texas Instruments synchronous serial interfaces
- Programmable data frame size from 4 to 16 bits

  - Internal loopback test mode for diagnostic/debug testing

- $I^2C$
  - Devices on the $I^2C$ bus can be designated as either a master or a slave
    - Supports both sending and receiving data as either a master or a slave
    - Supports simultaneous master and slave operation
  - Four $I^2C$ modes
    - Master transmit
    - Master receive
    - Slave transmit
    - Slave receive
  - Two transmission speeds: Standard (100 Kbps) and Fast (400 Kbps)
  - Master and slave interrupt generation
    - Master generates interrupts when a transmit or receive operation completes (or aborts due to an error)
    - Slave generates interrupts when data has been sent or requested by a master
  - Master with arbitration and clock synchronization, multimaster support, and 7-bit addressing mode
- Power
  - On-chip Low Drop-Out (LDO) voltage regulator, with programmable output user-adjustable from 2.25 V to 2.75 V
  - Low-power options on controller: Sleep and Deep-sleep modes
  - Low-power options for peripherals: software controls shutdown of individual peripherals
  - User-enabled LDO unregulated voltage detection and automatic reset
  - 3.3-V supply brown-out detection and reporting via interrupt or reset
- Flexible Reset Sources
  - Power-on reset (POR)
  - Reset pin assertion
  - Brown-out (BOR) detector alerts to system power drops
  - Software reset
  - Watchdog timer reset

  - Internal low drop-out (LDO) regulator output goes unregulated
- Industrial and extended temperature 48-pin RoHS-compliant LQFP package
- Industrial and extended temperature 48-pin RoHS-compliant QFN package

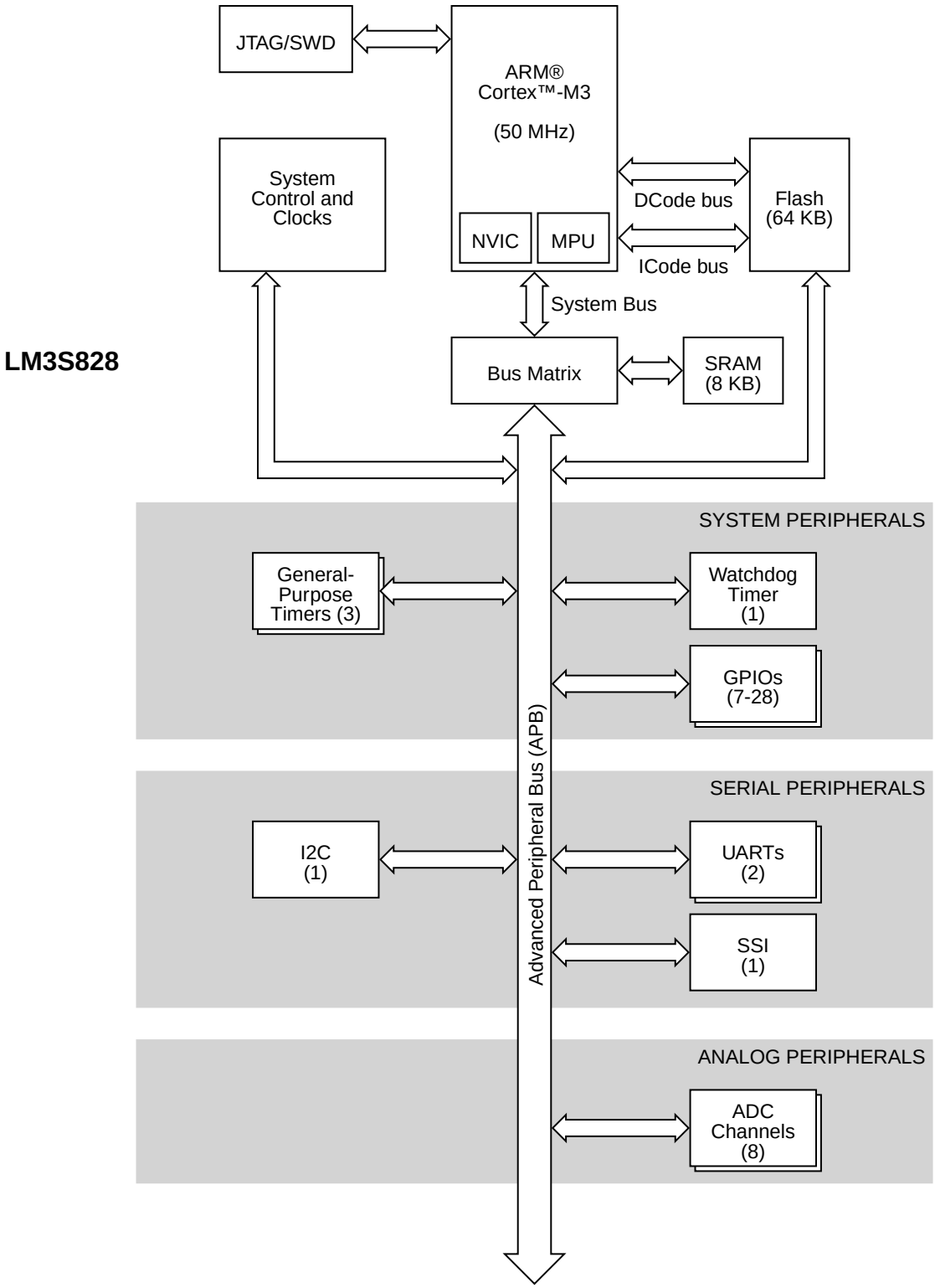
## 1.2 Target Applications

- Factory automation and control
- Industrial control power devices
- Building and home automation
- Stepper motors
- Brushless DC motors
- AC induction motors

## 1.3 High-Level Block Diagram

Figure 1-1 on page 32 depicts the features on the Stellaris LM3S828 microcontroller.

**Figure 1-1. Stellaris LM3S828 Microcontroller High-Level Block Diagram**



## 1.4 Functional Overview

The following sections provide an overview of the features of the LM3S828 microcontroller. The page number in parenthesis indicates where that feature is discussed in detail. Ordering and support information can be found in “Ordering and Contact Information” on page 511.

### 1.4.1 ARM Cortex™-M3

#### 1.4.1.1 Processor Core (see page 39)

All members of the Stellaris product family, including the LM3S828 microcontroller, are designed around an ARM Cortex™-M3 processor core. The ARM Cortex-M3 processor provides the core for a high-performance, low-cost platform that meets the needs of minimal memory implementation, reduced pin count, and low-power consumption, while delivering outstanding computational performance and exceptional system response to interrupts.

#### 1.4.1.2 Memory Map (see page 58)

A memory map lists the location of instructions and data in memory. The memory map for the LM3S828 controller can be found in Table 2-4 on page 58. Register addresses are given as a hexadecimal increment, relative to the module's base address as shown in the memory map.

#### 1.4.1.3 System Timer (SysTick) (see page 81)

Cortex-M3 includes an integrated system timer, SysTick. SysTick provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism. The counter can be used in several different ways, for example:

- An RTOS tick timer which fires at a programmable rate (for example, 100 Hz) and invokes a SysTick routine.
- A high-speed alarm timer using the system clock.
- A variable rate alarm or signal timer—the duration is range-dependent on the reference clock used and the dynamic range of the counter.
- A simple counter. Software can use this to measure time to completion and time used.
- An internal clock source control based on missing/meeting durations. The COUNTFLAG bit-field in the control and status register can be used to determine if an action completed within a set duration, as part of a dynamic clock management control loop.

#### 1.4.1.4 Nested Vectored Interrupt Controller (NVIC) (see page 82)

The LM3S828 controller includes the ARM Nested Vectored Interrupt Controller (NVIC) on the ARM® Cortex™-M3 core. The NVIC and Cortex-M3 prioritize and handle all exceptions. All exceptions are handled in Handler Mode. The processor state is automatically stored to the stack on an exception, and automatically restored from the stack at the end of the Interrupt Service Routine (ISR). The vector is fetched in parallel to the state saving, which enables efficient interrupt entry. The processor supports tail-chaining, which enables back-to-back interrupts to be performed without the overhead of state saving and restoration. Software can set eight priority levels on 7 exceptions (system handlers) and 22 interrupts.

#### 1.4.1.5 System Control Block (SCB) (see page 84)

The SCB provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

#### 1.4.1.6 Memory Protection Unit (MPU) (see page 84)

The MPU supports the standard ARMv7 Protected Memory System Architecture (PMSA) model. The MPU provides full support for protection regions, overlapping protection regions, access permissions, and exporting memory attributes to the system.

### 1.4.2 Motor Control Peripherals

To enhance motor control, the LM3S828 controller features Pulse Width Modulation (PWM) outputs.

#### 1.4.2.1 PWM

Pulse width modulation (PWM) is a powerful technique for digitally encoding analog signal levels. High-resolution counters are used to generate a square wave, and the duty cycle of the square wave is modulated to encode an analog signal. Typical applications include switching power supplies and motor control.

On the LM3S828, PWM motion control functionality can be achieved through:

- The motion control features of the general-purpose timers using the CCP pins

***CCP Pins (see page 265)***

The General-Purpose Timer Module's CCP (Capture Compare PWM) pins are software programmable to support a simple PWM mode with a software-programmable output inversion of the PWM signal.

### 1.4.3 Analog Peripherals

To handle analog signals, the LM3S828 microcontroller offers an Analog-to-Digital Converter (ADC).

#### 1.4.3.1 ADC (see page 319)

An analog-to-digital converter (ADC) is a peripheral that converts a continuous analog voltage to a discrete digital number.

The LM3S828 ADC module features 10-bit conversion resolution and supports eight input channels, plus an internal temperature sensor. Four buffered sample sequences allow rapid sampling of up to eight analog input sources without controller intervention. Each sample sequence provides flexible programming with fully configurable input source, trigger events, interrupt generation, and sequence priority.

### 1.4.4 Serial Communications Peripherals

The LM3S828 controller supports both asynchronous and synchronous serial communications with:

- Two fully programmable 16C550-type UARTs
- One SSI module
- One $I^2C$ module

#### 1.4.4.1 UART (see page 354)

A Universal Asynchronous Receiver/Transmitter (UART) is an integrated circuit used for RS-232C serial communications, containing a transmitter (parallel-to-serial converter) and a receiver (serial-to-parallel converter), each clocked separately.

The LM3S828 controller includes two fully programmable 16C550-type UARTs that support data transfer speeds up to 3.125 Mbps. (Although similar in functionality to a 16C550 UART, it is not register-compatible.)

Separate 16x8 transmit (TX) and receive (RX) FIFOs reduce CPU interrupt service loading. The UART can generate individually masked interrupts from the RX, TX, modem status, and error conditions. The module provides a single combined interrupt when any of the interrupts are asserted and are unmasked.

#### 1.4.4.2 SSI (see page 393)

Synchronous Serial Interface (SSI) is a four-wire bi-directional full and low-speed communications interface.

The LM3S828 controller includes one SSI module that provides the functionality for synchronous serial communications with peripheral devices, and can be configured to use the Freescale SPI, MICROWIRE, or TI synchronous serial interface frame formats. The size of the data frame is also configurable, and can be set between 4 and 16 bits, inclusive.

The SSI module performs serial-to-parallel conversion on data received from a peripheral device, and parallel-to-serial conversion on data transmitted to a peripheral device. The TX and RX paths are buffered with internal FIFOs, allowing up to eight 16-bit values to be stored independently.

The SSI module can be configured as either a master or slave device. As a slave device, the SSI module can also be configured to disable its output, which allows a master device to be coupled with multiple slave devices.

The SSI module also includes a programmable bit rate clock divider and prescaler to generate the output serial clock derived from the SSI module's input clock. Bit rates are generated based on the input clock and the maximum bit rate is determined by the connected peripheral.

#### 1.4.4.3 $I^2C$ (see page 430)

The Inter-Integrated Circuit ($I^2C$) bus provides bi-directional data transfer through a two-wire design (a serial data line SDA and a serial clock line SCL).

The $I^2C$ bus interfaces to external $I^2C$ devices such as serial memory (RAMs and ROMs), networking devices, LCDs, tone generators, and so on. The $I^2C$ bus may also be used for system testing and diagnostic purposes in product development and manufacture.

The LM3S828 controller includes one $I^2C$ module that provides the ability to communicate to other IC devices over an $I^2C$ bus. The $I^2C$ bus supports devices that can both transmit and receive (write and read) data.

Devices on the $I^2C$ bus can be designated as either a master or a slave. The $I^2C$ module supports both sending and receiving data as either a master or a slave, and also supports the simultaneous operation as both a master and a slave. The four $I^2C$ modes are: Master Transmit, Master Receive, Slave Transmit, and Slave Receive.

A Stellaris $I^2C$ module can operate at two speeds: Standard (100 Kbps) and Fast (400 Kbps).

Both the I²C master and slave can generate interrupts. The I²C master generates interrupts when a transmit or receive operation completes (or aborts due to an error). The I²C slave generates interrupts when data has been sent or requested by a master.

### 1.4.5 System Peripherals

#### 1.4.5.1 Programmable GPIOs (see page 220)

General-purpose input/output (GPIO) pins offer flexibility for a variety of connections.

The Stellaris GPIO module is comprised of five physical GPIO blocks, each corresponding to an individual GPIO port. The GPIO module is FiRM-compliant (compliant to the ARM Foundation IP for Real-Time Microcontrollers specification) and supports 7-28 programmable input/output pins. The number of GPIOs available depends on the peripherals being used (see “Signal Tables” on page 468 for the signals available to each GPIO pin).

The GPIO module features programmable interrupt generation as either edge-triggered or level-sensitive on all pins, programmable control for GPIO pad configuration, and bit masking in both read and write operations through address lines. Pins configured as digital inputs are Schmitt-triggered.

#### 1.4.5.2 Three Programmable Timers (see page 259)

Programmable timers can be used to count or time external events that drive the Timer input pins.

The Stellaris General-Purpose Timer Module (GPTM) contains three GPTM blocks. Each GPTM block provides two 16-bit timers/counters that can be configured to operate independently as timers or event counters, or configured to operate as one 32-bit timer or one 32-bit Real-Time Clock (RTC). Timers can also be used to trigger analog-to-digital (ADC) conversions.

When configured in 32-bit mode, a timer can run as a Real-Time Clock (RTC), one-shot timer or periodic timer. When in 16-bit mode, a timer can run as a one-shot timer or periodic timer, and can extend its precision by using an 8-bit prescaler. A 16-bit timer can also be configured for event capture or Pulse Width Modulation (PWM) generation.

#### 1.4.5.3 Watchdog Timer (see page 295)

A watchdog timer can generate an interrupt or a reset when a time-out value is reached. The watchdog timer is used to regain control when a system has failed due to a software error or to the failure of an external device to respond in the expected way.

The Stellaris Watchdog Timer module consists of a 32-bit down counter, a programmable load register, interrupt generation logic, and a locking register.

The Watchdog Timer can be configured to generate an interrupt to the controller on its first time-out, and to generate a reset signal on its second time-out. Once the Watchdog Timer has been configured, the lock register can be written to prevent the timer configuration from being inadvertently altered.

### 1.4.6 Memory Peripherals

The LM3S828 controller offers both single-cycle SRAM and single-cycle Flash memory.

#### 1.4.6.1 SRAM (see page 203)

The LM3S828 static random access memory (SRAM) controller supports 8 KB SRAM. The internal SRAM of the Stellaris devices starts at base address 0x2000.0000 of the device memory map. To reduce the number of time-consuming read-modify-write (RMW) operations, ARM has introduced *bit-banding* technology in the new Cortex-M3 processor. With a bit-band-enabled processor, certain

regions in the memory map (SRAM and peripheral space) can use address aliases to access individual bits in a single, atomic operation.

#### 1.4.6.2 Flash (see page 204)

The LM3S828 Flash controller supports 64 KB of flash memory. The flash is organized as a set of 1-KB blocks that can be individually erased. Erasing a block causes the entire contents of the block to be reset to all 1s. These blocks are paired into a set of 2-KB blocks that can be individually protected. The blocks can be marked as read-only or execute-only, providing different levels of code protection. Read-only blocks cannot be erased or programmed, protecting the contents of those blocks from being modified. Execute-only blocks cannot be erased or programmed, and can only be read by the controller instruction fetch mechanism, protecting the contents of those blocks from being read by either the controller or by a debugger.

### 1.4.7 Additional Features

#### 1.4.7.1 JTAG TAP Controller (see page 139)

The Joint Test Action Group (JTAG) port is an IEEE standard that defines a Test Access Port and Boundary Scan Architecture for digital integrated circuits and provides a standardized serial interface for controlling the associated test logic. The TAP, Instruction Register (IR), and Data Registers (DR) can be used to test the interconnections of assembled printed circuit boards and obtain manufacturing information on the components. The JTAG Port also provides a means of accessing and controlling design-for-test features such as I/O pin observation and control, scan testing, and debugging.

The JTAG port is composed of the standard five pins: $\overline{\text{TRST}}$, TCK, TMS, TDI, and TDO. Data is transmitted serially into the controller on TDI and out of the controller on TDO. The interpretation of this data is dependent on the current state of the TAP controller. For detailed information on the operation of the JTAG port and TAP controller, please refer to the *IEEE Standard 1149.1-Test Access Port and Boundary-Scan Architecture*.

The Stellaris JTAG controller works with the ARM JTAG controller built into the Cortex-M3 core. This is implemented by multiplexing the TDO outputs from both JTAG controllers. ARM JTAG instructions select the ARM TDO output while Stellaris JTAG instructions select the Stellaris TDO outputs. The multiplexer is controlled by the Stellaris JTAG controller, which has comprehensive programming for the ARM, Stellaris, and unimplemented JTAG instructions.

#### 1.4.7.2 System Control and Clocks (see page 149)

System control determines the overall operation of the device. It provides information about the device, controls the clocking of the device and individual peripherals, and handles reset detection and reporting.

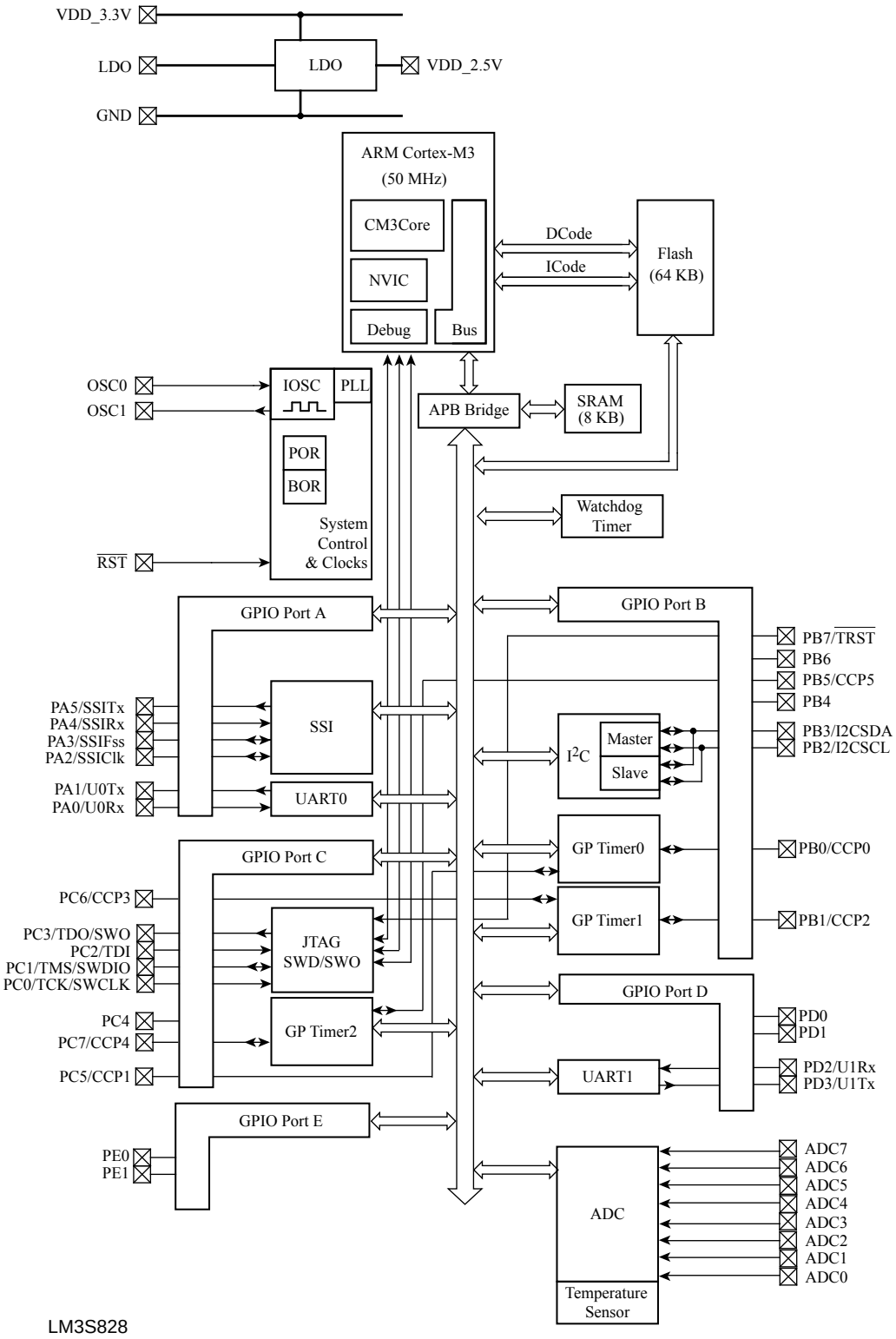
### 1.4.8 Hardware Details

Details on the pins and package can be found in the following sections:

- “Pin Diagram” on page 466
- “Signal Tables” on page 468
- “Operating Characteristics” on page 475
- “Electrical Characteristics” on page 476
- “Package Information” on page 513

### 1.4.9 System Block Diagram

**Figure 1-2. LM3S828 Controller System-Level Block Diagram**



# 2 The Cortex-M3 Processor

The ARM® Cortex™-M3 processor provides a high-performance, low-cost platform that meets the system requirements of minimal memory implementation, reduced pin count, and low power consumption, while delivering outstanding computational performance and exceptional system response to interrupts. Features include:

- Compact core.
- Thumb-2 instruction set, delivering the high-performance expected of an ARM core in the memory size usually associated with 8- and 16-bit devices; typically in the range of a few kilobytes of memory for microcontroller class applications.
- Rapid application execution through Harvard architecture characterized by separate buses for instruction and data.
- Exceptional interrupt handling, by implementing the register manipulations required for handling an interrupt in hardware.
- Deterministic, fast interrupt processing: always 12 cycles, or just 6 cycles with tail-chaining
- Memory protection unit (MPU) to provide a privileged mode of operation for complex applications.
- Migration from the ARM7™ processor family for better performance and power efficiency.
- Full-featured debug solution
  - Serial Wire JTAG Debug Port (SWJ-DP)
  - Flash Patch and Breakpoint (FPB) unit for implementing breakpoints
  - Data Watchpoint and Trigger (DWT) unit for implementing watchpoints, trigger resources, and system profiling
  - Instrumentation Trace Macrocell (ITM) for support of printf style debugging
  - Trace Port Interface Unit (TPIU) for bridging to a Trace Port Analyzer
- Optimized for single-cycle flash usage
- Three sleep modes with clock gating for low power
- Single-cycle multiply instruction and hardware divide
- Atomic operations
- ARM Thumb2 mixed 16-/32-bit instruction set
- 1.25 DMIPS/MHz

The Stellaris® family of microcontrollers builds on this core to bring high-performance 32-bit computing to cost-sensitive embedded microcontroller applications, such as factory automation and control, industrial control power devices, building and home automation, and stepper motor control.

This chapter provides information on the Stellaris implementation of the Cortex-M3 processor, including the programming model, the memory model, the exception model, fault handling, and power management.

For technical details on the instruction set, see the *Cortex™-M3 Instruction Set Technical User's Manual*.

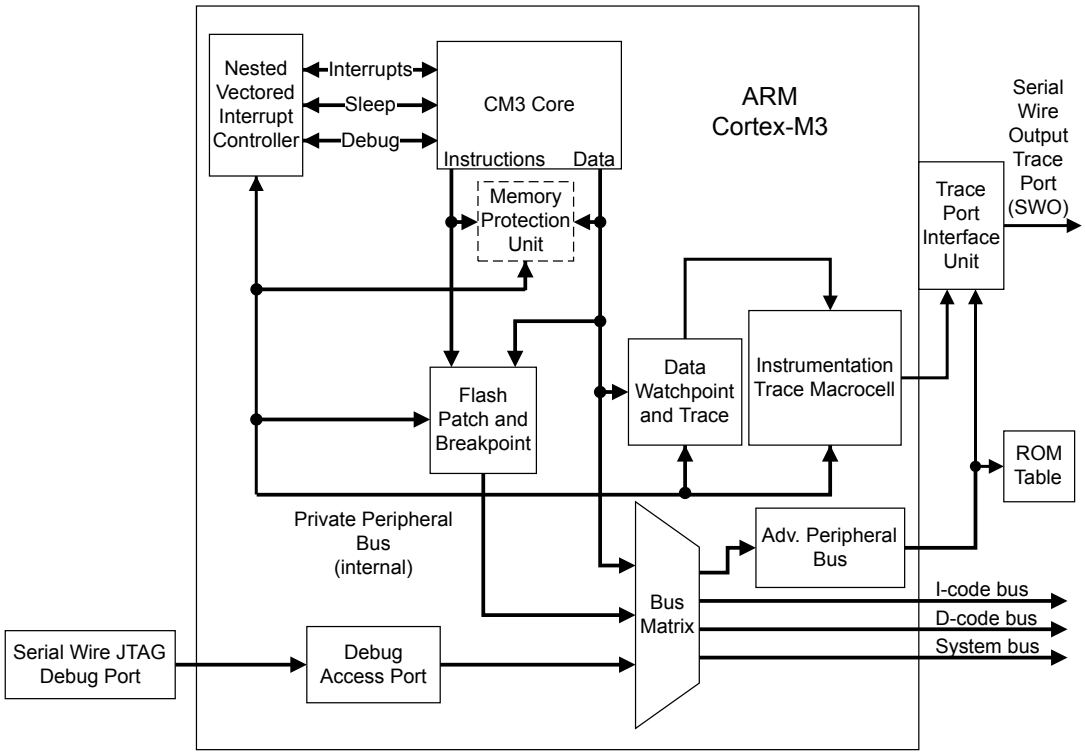
## 2.1 Block Diagram

The Cortex-M3 processor is built on a high-performance processor core, with a 3-stage pipeline Harvard architecture, making it ideal for demanding embedded applications. The processor delivers exceptional power efficiency through an efficient instruction set and extensively optimized design, providing high-end processing hardware including single-cycle 32x32 multiplication and dedicated hardware division.

To facilitate the design of cost-sensitive devices, the Cortex-M3 processor implements tightly coupled system components that reduce processor area while significantly improving interrupt handling and system debug capabilities. The Cortex-M3 processor implements a version of the Thumb® instruction set, ensuring high code density and reduced program memory requirements. The Cortex-M3 instruction set provides the exceptional performance expected of a modern 32-bit architecture, with the high code density of 8-bit and 16-bit microcontrollers.

The Cortex-M3 processor closely integrates a nested interrupt controller (NVIC), to deliver industry-leading interrupt performance. The Stellaris NVIC includes a non-maskable interrupt (NMI) and provides eight interrupt priority levels. The tight integration of the processor core and NVIC provides fast execution of interrupt service routines (ISRs), dramatically reducing interrupt latency. The hardware stacking of registers and the ability to suspend load-multiple and store-multiple operations further reduce interrupt latency. Interrupt handlers do not require any assembler stubs which removes code overhead from the ISRs. Tail-chaining optimization also significantly reduces the overhead when switching from one ISR to another. To optimize low-power designs, the NVIC integrates with the sleep modes, including Deep-sleep mode, which enables the entire device to be rapidly powered down.

**Figure 2-1. CPU Block Diagram**




## 2.2 Overview

### 2.2.1 System-Level Interface

The Cortex-M3 processor provides multiple interfaces using AMBA® technology to provide high-speed, low-latency memory accesses. The core supports unaligned data accesses and implements atomic bit manipulation that enables faster peripheral controls, system spinlocks, and thread-safe Boolean data handling.

The Cortex-M3 processor has a memory protection unit (MPU) that provides fine-grain memory control, enabling applications to implement security privilege levels and separate code, data and stack on a task-by-task basis.

### 2.2.2 Integrated Configurable Debug

The Cortex-M3 processor implements a complete hardware debug solution, providing high system visibility of the processor and memory through either a traditional JTAG port or a 2-pin Serial Wire Debug (SWD) port that is ideal for microcontrollers and other small package devices. The Stellaris implementation replaces the ARM SW-DP and JTAG-DP with the ARM CoreSight™-compliant Serial Wire JTAG Debug Port (SWJ-DP) interface. The SWJ-DP interface combines the SWD and JTAG debug ports into one module. See the *ARM® Debug Interface V5 Architecture Specification* for details on SWJ-DP.

For system trace, the processor integrates an Instrumentation Trace Macrocell (ITM) alongside data watchpoints and a profiling unit. To enable simple and cost-effective profiling of the system trace events, a Serial Wire Viewer (SWV) can export a stream of software-generated messages, data trace, and profiling information through a single pin.

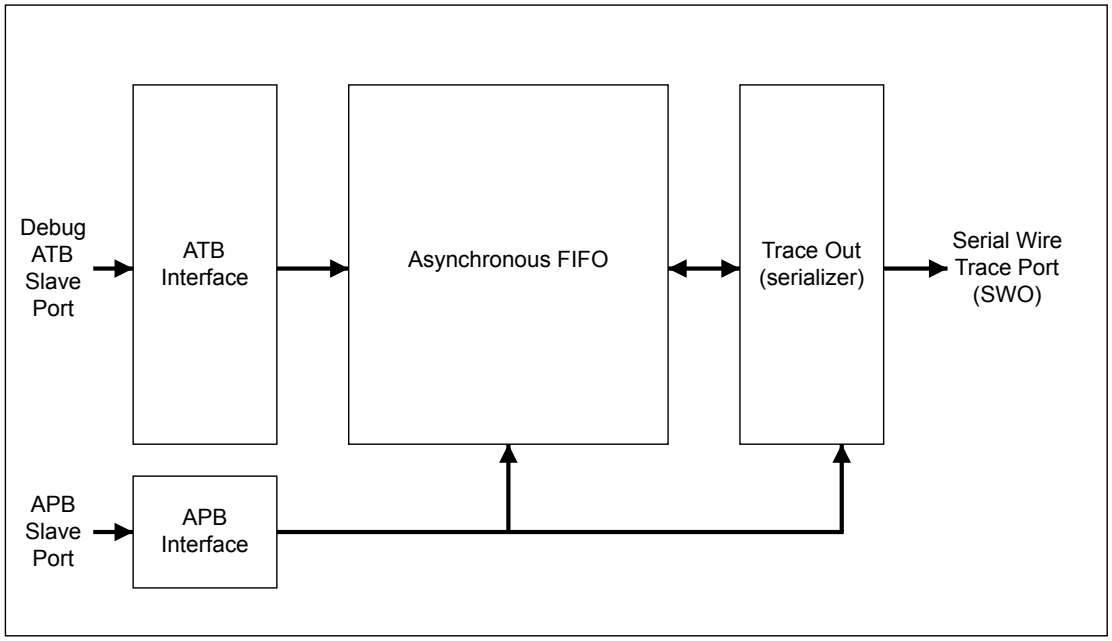
The Flash Patch and Breakpoint Unit (FPB) provides up to eight hardware breakpoint comparators that debuggers can use. The comparators in the FPB also provide remap functions of up to eight words in the program code in the CODE memory region. This enables applications stored in a read-only area of Flash memory to be patched in another area of on-chip SRAM or Flash memory. If a patch is required, the application programs the FPB to remap a number of addresses. When those addresses are accessed, the accesses are redirected to a remap table specified in the FPB configuration.

For more information on the Cortex-M3 debug capabilities, see the*ARM® Debug Interface V5 Architecture Specification*.

### 2.2.3 Trace Port Interface Unit (TPIU)

The TPIU acts as a bridge between the Cortex-M3 trace data from the ITM, and an off-chip Trace Port Analyzer, as shown in Figure 2-2 on page 42.

**Figure 2-2. TPIU Block Diagram**




### 2.2.4 Cortex-M3 System Component Details

The Cortex-M3 includes the following system components:

- SysTick

  A 24-bit count-down timer that can be used as a Real-Time Operating System (RTOS) tick timer or as a simple counter (see “System Timer (SysTick)” on page 81).

- Nested Vectored Interrupt Controller (NVIC)

  An embedded interrupt controller that supports low latency interrupt processing (see “Nested Vectored Interrupt Controller (NVIC)” on page 82).

- System Control Block (SCB)

The programming model interface to the processor. The SCB provides system implementation information and system control, including configuration, control, and reporting of system exceptions( see “System Control Block (SCB)” on page 84).

- Memory Protection Unit (MPU)

  Improves system reliability by defining the memory attributes for different memory regions. The MPU provides up to eight different regions and an optional predefined background region (see “Memory Protection Unit (MPU)” on page 84).

## 2.3 Programming Model

This section describes the Cortex-M3 programming model. In addition to the individual core register descriptions, information about the processor modes and privilege levels for software execution and stacks is included.

### 2.3.1 Processor Mode and Privilege Levels for Software Execution

The Cortex-M3 has two modes of operation:

- Thread mode

  Used to execute application software. The processor enters Thread mode when it comes out of reset.

- Handler mode

  Used to handle exceptions. When the processor has finished exception processing, it returns to Thread mode.

In addition, the Cortex-M3 has two privilege levels:

- Unprivileged

  In this mode, software has the following restrictions:

  – Limited access to the MSR and MRS instructions and no use of the CPS instruction

  – No access to the system timer, NVIC, or system control block

  – Possibly restricted access to memory or peripherals

- Privileged

  In this mode, software can use all the instructions and has access to all resources.

In Thread mode, the **CONTROL** register (see page 57) controls whether software execution is privileged or unprivileged. In Handler mode, software execution is always privileged.

Only privileged software can write to the **CONTROL** register to change the privilege level for software execution in Thread mode. Unprivileged software can use the SVC instruction to make a supervisor call to transfer control to privileged software.

### 2.3.2 Stacks

The processor uses a full descending stack, meaning that the stack pointer indicates the last stacked item on the stack memory. When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location. The processor implements

two stacks: the main stack and the process stack, with independent copies of the stack pointer (see the **SP** register on page 47).

In Thread mode, the **CONTROL** register (see page 57) controls whether the processor uses the main stack or the process stack. In Handler mode, the processor always uses the main stack. The options for processor operations are shown in Table 2-1 on page 44.

**Table 2-1. Summary of Processor Mode, Privilege Level, and Stack Use**

| Processor Mode | Use | Privilege Level | Stack Used |
|---|---|---|---|
| Thread | Applications | Privileged or unprivileged [a] | Main stack or process stack [a] |
| Handler | Exception handlers | Always privileged | Main stack |

a. See **CONTROL** (page 57).

## 2.3.3 Register Map

Figure 2-3 on page 44 shows the Cortex-M3 register set. Table 2-2 on page 45 lists the Core registers. The core registers are not memory mapped and are accessed by register name, so the base address is n/a (not applicable) and there is no offset.

**Figure 2-3. Cortex-M3 Register Set**

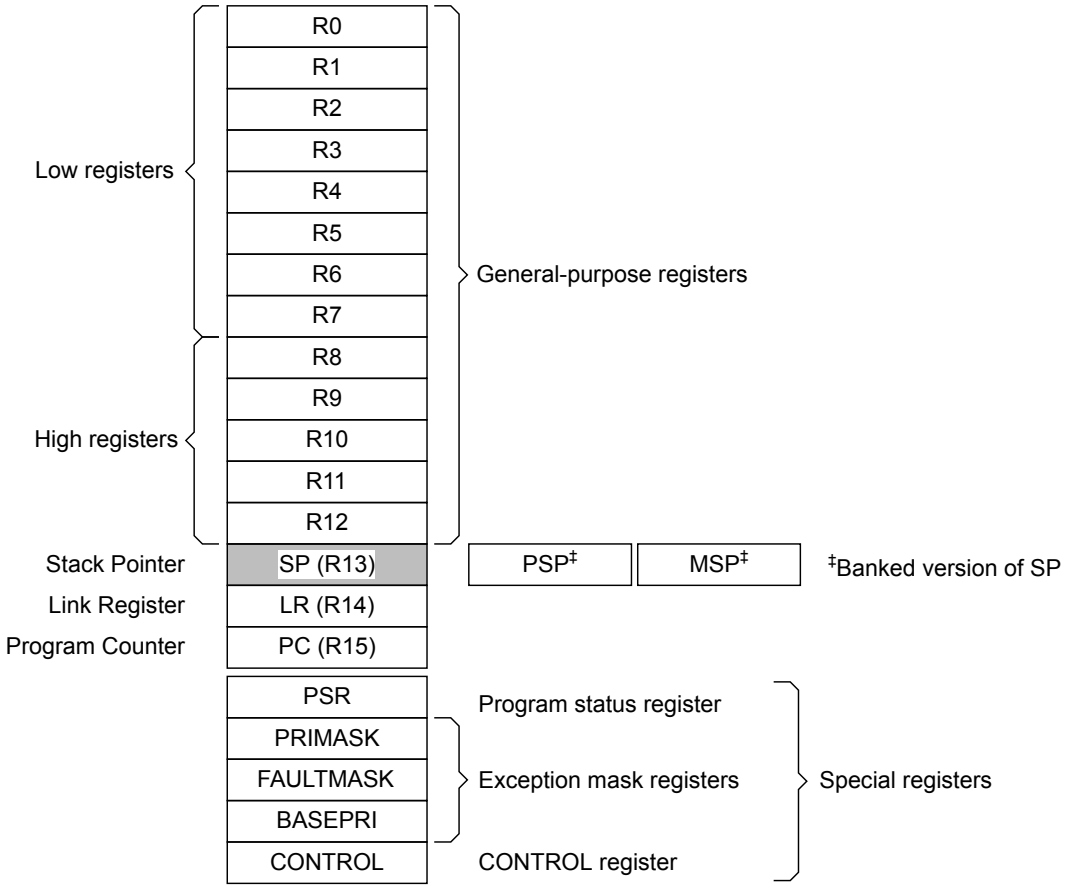


**Table 2-2. Processor Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| - | R0 | R/W | - | Cortex General-Purpose Register 0 | 46 |
| - | R1 | R/W | - | Cortex General-Purpose Register 1 | 46 |
| - | R2 | R/W | - | Cortex General-Purpose Register 2 | 46 |
| - | R3 | R/W | - | Cortex General-Purpose Register 3 | 46 |
| - | R4 | R/W | - | Cortex General-Purpose Register 4 | 46 |
| - | R5 | R/W | - | Cortex General-Purpose Register 5 | 46 |
| - | R6 | R/W | - | Cortex General-Purpose Register 6 | 46 |
| - | R7 | R/W | - | Cortex General-Purpose Register 7 | 46 |
| - | R8 | R/W | - | Cortex General-Purpose Register 8 | 46 |
| - | R9 | R/W | - | Cortex General-Purpose Register 9 | 46 |
| - | R10 | R/W | - | Cortex General-Purpose Register 10 | 46 |
| - | R11 | R/W | - | Cortex General-Purpose Register 11 | 46 |
| - | R12 | R/W | - | Cortex General-Purpose Register 12 | 46 |
| - | SP | R/W | - | Stack Pointer | 47 |
| - | LR | R/W | 0xFFFF.FFFF | Link Register | 48 |
| - | PC | R/W | - | Program Counter | 49 |
| - | PSR | R/W | 0x0100.0000 | Program Status Register | 50 |
| - | PRIMASK | R/W | 0x0000.0000 | Priority Mask Register | 54 |
| - | FAULTMASK | R/W | 0x0000.0000 | Fault Mask Register | 55 |
| - | BASEPRI | R/W | 0x0000.0000 | Base Priority Mask Register | 56 |
| - | CONTROL | R/W | 0x0000.0000 | Control Register | 57 |

### 2.3.4 Register Descriptions

This section lists and describes the Cortex-M3 registers, in the order shown in Figure 2-3 on page 44. The core registers are not memory mapped and are accessed by register name rather than offset.

**Note:** The register type shown in the register descriptions refers to type during program execution in Thread mode and Handler mode. Debug access can differ.

### Register 1: Cortex General-Purpose Register 0 (R0)

### Register 2: Cortex General-Purpose Register 1 (R1)

### Register 3: Cortex General-Purpose Register 2 (R2)

### Register 4: Cortex General-Purpose Register 3 (R3)

### Register 5: Cortex General-Purpose Register 4 (R4)

### Register 6: Cortex General-Purpose Register 5 (R5)

### Register 7: Cortex General-Purpose Register 6 (R6)

### Register 8: Cortex General-Purpose Register 7 (R7)

### Register 9: Cortex General-Purpose Register 8 (R8)

### Register 10: Cortex General-Purpose Register 9 (R9)

### Register 11: Cortex General-Purpose Register 10 (R10)

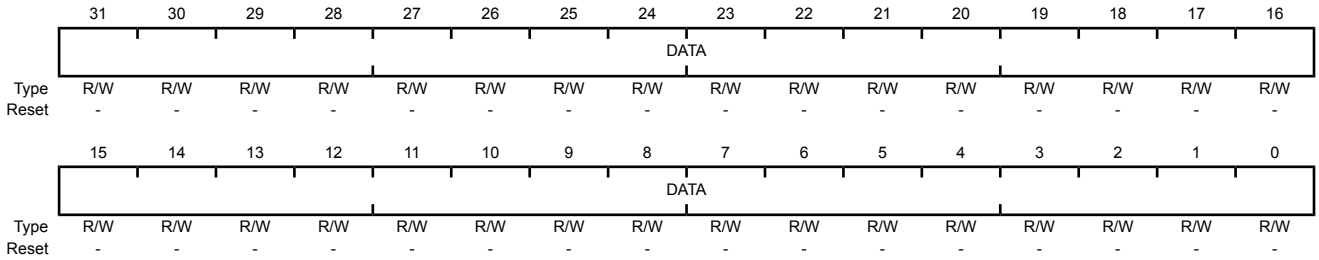
### Register 12: Cortex General-Purpose Register 11 (R11)

### Register 13: Cortex General-Purpose Register 12 (R12)

The **Rn** registers are 32-bit general-purpose registers for data operations and can be accessed from either privileged or unprivileged mode.

Cortex General-Purpose Register 0 (R0)

Type R/W, reset -



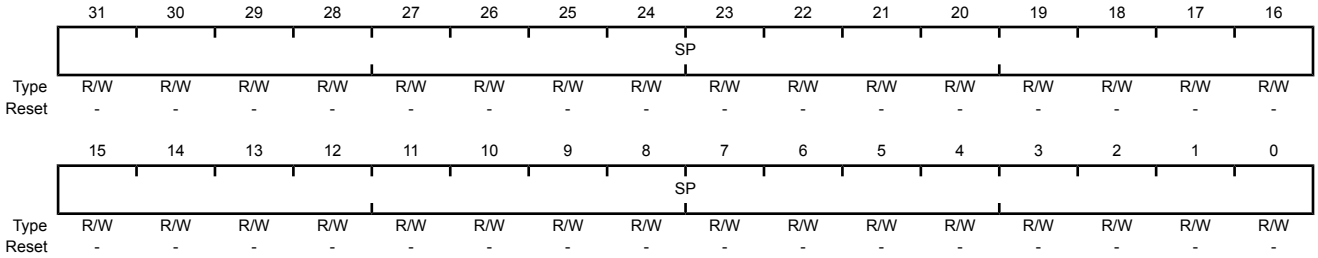

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R/W | - | Register data. |

### Register 14: Stack Pointer (SP)

The **Stack Pointer (SP)** is register R13. In Thread mode, the function of this register changes depending on the ASP bit in the **Control Register (CONTROL)** register. When the ASP bit is clear, this register is the **Main Stack Pointer (MSP)**. When the ASP bit is set, this register is the **Process Stack Pointer (PSP)**. On reset, the ASP bit is clear, and the processor loads the **MSP** with the value from address 0x0000.0000. The **MSP** can only be accessed in privileged mode; the **PSP** can be accessed in either privileged or unprivileged mode.

Stack Pointer (SP)

Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | SP | R/W | - | This field is the address of the stack pointer. |

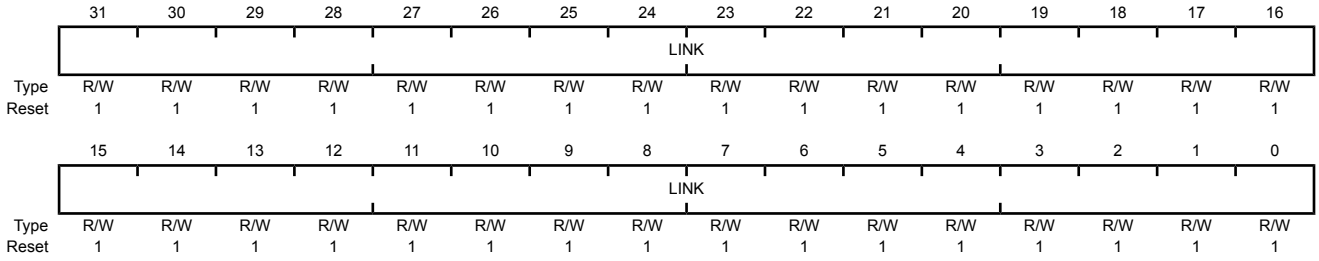
### Register 15: Link Register (LR)

The **Link Register (LR)** is register R14, and it stores the return information for subroutines, function calls, and exceptions. **LR** can be accessed from either privileged or unprivileged mode.

EXC_RETURN is loaded into **LR** on exception entry. See Table 2-10 on page 73 for the values and description.

Link Register (LR)

Type R/W, reset 0xFFFF.FFFF



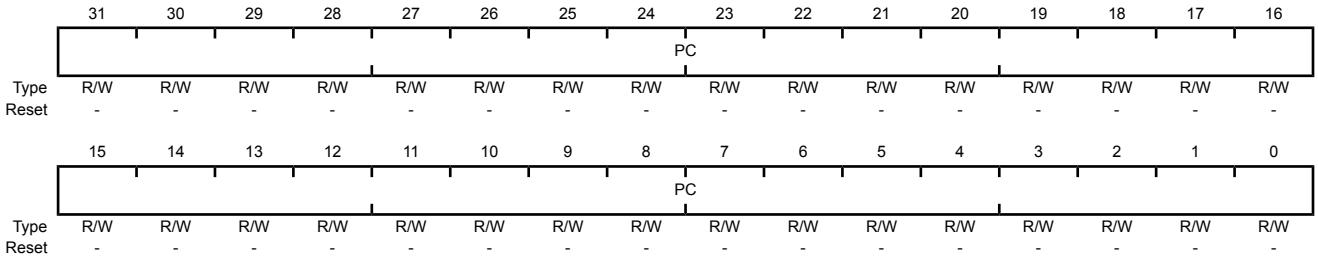

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | LINK | R/W | 0xFFFF.FFFF | This field is the return address. |

### Register 16: Program Counter (PC)

The **Program Counter (PC)** is register R15, and it contains the current program address. On reset, the processor loads the **PC** with the value of the reset vector, which is at address 0x0000.0004. Bit 0 of the reset vector is loaded into the THUMB bit of the **EPSR** at reset and must be 1. The **PC** register can be accessed in either privileged or unprivileged mode.

Program Counter (PC)

Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | PC | R/W | - | This field is the current program address. |

### Register 17: Program Status Register (PSR)

**Note:** This register is also referred to as **xPSR**.

The **Program Status Register (PSR)** has three functions, and the register bits are assigned to the different functions:

- **Application Program Status Register (APSR)**, bits 31:27,
- **Execution Program Status Register (EPSR)**, bits 26:24, 15:10
- **Interrupt Program Status Register (IPSR)**, bits 5:0

The **PSR**, **IPSR**, and **EPSR** registers can only be accessed in privileged mode; the **APSR** register can be accessed in either privileged or unprivileged mode.

**APSR** contains the current state of the condition flags from previous instruction executions.

**EPSR** contains the Thumb state bit and the execution state bits for the If-Then (`IT`) instruction or the Interruptible-Continuable Instruction (`ICI`) field for an interrupted load multiple or store multiple instruction. Attempts to read the **EPSR** directly through application software using the `MSR` instruction always return zero. Attempts to write the **EPSR** using the `MSR` instruction in application software are always ignored. Fault handlers can examine the **EPSR** value in the stacked **PSR** to determine the operation that faulted (see “Exception Entry and Return” on page 71).

**IPSR** contains the exception type number of the current Interrupt Service Routine (ISR).

These registers can be accessed individually or as a combination of any two or all three registers, using the register name as an argument to the `MSR` or `MRS` instructions. For example, all of the registers can be read using **PSR** with the `MRS` instruction, or **APSR** only can be written to using **APSR** with the `MSR` instruction. page 50 shows the possible register combinations for the **PSR**. See the `MRS` and `MSR` instruction descriptions in the *Cortex™-M3 Instruction Set Technical User's Manual* for more information about how to access the program status registers.

**Table 2-3. PSR Register Combinations**

| Register | Type | Combination |
|---|---|---|
| **PSR** | R/W[a, b] | **APSR**, **EPSR**, and **IPSR** |
| **IEPSR** | RO | **EPSR** and **IPSR** |
| **IAPSR** | R/W[a] | **APSR** and **IPSR** |
| **EAPSR** | R/W[b] | **APSR** and **EPSR** |

a. The processor ignores writes to the **IPSR** bits.

b. Reads of the **EPSR** bits return zero, and the processor ignores writes to these bits.

Program Status Register (PSR)

Type R/W, reset 0x0100.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | N | Z | C | V | Q | ICI / IT | | THUMB | reserved | | | | | | | |
| Type | R/W | R/W | R/W | R/W | R/W | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ICI / IT | | | | | | reserved | | | | ISRNUM | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | N | R/W | 0 | **APSR** Negative or Less Flag<br><br>Value Description<br>1 The previous operation result was negative or less than.<br>0 The previous operation result was positive, zero, greater than, or equal.<br><br>The value of this bit is only meaningful when accessing **PSR** or **APSR**. |
| 30 | Z | R/W | 0 | **APSR** Zero Flag<br><br>Value Description<br>1 The previous operation result was zero.<br>0 The previous operation result was non-zero.<br><br>The value of this bit is only meaningful when accessing **PSR** or **APSR**. |
| 29 | C | R/W | 0 | **APSR** Carry or Borrow Flag<br><br>Value Description<br>1 The previous add operation resulted in a carry bit or the previous subtract operation did not result in a borrow bit.<br>0 The previous add operation did not result in a carry bit or the previous subtract operation resulted in a borrow bit.<br><br>The value of this bit is only meaningful when accessing **PSR** or **APSR**. |
| 28 | V | R/W | 0 | **APSR** Overflow Flag<br><br>Value Description<br>1 The previous operation resulted in an overflow.<br>0 The previous operation did not result in an overflow.<br><br>The value of this bit is only meaningful when accessing **PSR** or **APSR**. |
| 27 | Q | R/W | 0 | **APSR** DSP Overflow and Saturation Flag<br><br>Value Description<br>1 DSP Overflow or saturation has occurred.<br>0 DSP overflow or saturation has not occurred since reset or since the bit was last cleared.<br><br>The value of this bit is only meaningful when accessing **PSR** or **APSR**. This bit is cleared by software using an MRS instruction. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 26:25 | ICI / IT | RO | 0x0 | **EPSR** ICI / IT status<br>These bits, along with bits 15:10, contain the Interruptible-Continuable Instruction (`ICI`) field for an interrupted load multiple or store multiple instruction or the execution state bits of the `IT` instruction.<br>When **EPSR** holds the `ICI` execution state, bits 26:25 are zero.<br>The If-Then block contains up to four instructions following a 16-bit `IT` instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See the *Cortex™-M3 Instruction Set Technical User's Manual* for more information.<br>The value of this field is only meaningful when accessing **PSR** or **EPSR**. |
| 24 | THUMB | RO | 1 | **EPSR** Thumb State<br>This bit indicates the Thumb state and should always be set.<br>The following can clear the `THUMB` bit:<br>■ The `BLX`, `BX` and `POP{PC}` instructions<br>■ Restoration from the stacked **xPSR** value on an exception return<br>■ Bit 0 of the vector value on an exception entry<br>Attempting to execute instructions when this bit is clear results in a fault or lockup. See "Lockup" on page 75 for more information.<br>The value of this bit is only meaningful when accessing **PSR** or **EPSR**. |
| 23:16 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:10 | ICI / IT | RO | 0x0 | **EPSR** ICI / IT status<br>These bits, along with bits 26:25, contain the Interruptible-Continuable Instruction (`ICI`) field for an interrupted load multiple or store multiple instruction or the execution state bits of the `IT` instruction.<br>When an interrupt occurs during the execution of an `LDM`, `STM`, `PUSH` or `POP` instruction, the processor stops the load multiple or store multiple instruction operation temporarily and stores the next register operand in the multiple operation to bits 15:12. After servicing the interrupt, the processor returns to the register pointed to by bits 15:12 and resumes execution of the multiple load or store instruction. When **EPSR** holds the `ICI` execution state, bits 11:10 are zero.<br>The If-Then block contains up to four instructions following a 16-bit `IT` instruction. Each instruction in the block is conditional. The conditions for the instructions are either all the same, or some can be the inverse of others. See the *Cortex™-M3 Instruction Set Technical User's Manual* for more information.<br>The value of this field is only meaningful when accessing **PSR** or **EPSR**. |
| 9:6 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

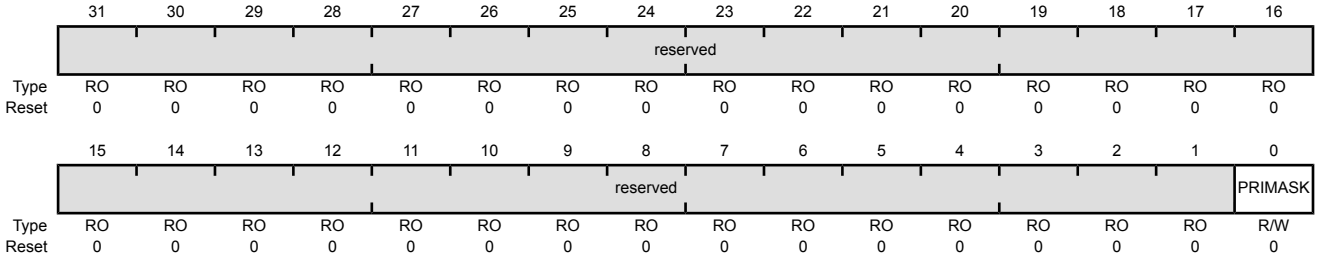
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 5:0 | ISRNUM | RO | 0x00 | **IPSR** ISR Number<br>This field contains the exception type number of the current Interrupt Service Routine (ISR).<br><br>Value / Description<br>0x00 Thread mode<br>0x01 Reserved<br>0x02 NMI<br>0x03 Hard fault<br>0x04 Memory management fault<br>0x05 Bus fault<br>0x06 Usage fault<br>0x07-0x0A Reserved<br>0x0B SVCall<br>0x0C Reserved for Debug<br>0x0D Reserved<br>0x0E PendSV<br>0x0F SysTick<br>0x10 Interrupt Vector 0<br>0x11 Interrupt Vector 1<br>... ...<br>0x2D Interrupt Vector 29<br>0x2E-0x3F Reserved<br><br>See “Exception Types” on page 67 for more information.<br>The value of this field is only meaningful when accessing **PSR** or **IPSR**. |

### Register 18: Priority Mask Register (PRIMASK)

The **PRIMASK** register prevents activation of all exceptions with programmable priority. Reset, non-maskable interrupt (NMI), and hard fault are the only exceptions with fixed priority. Exceptions should be disabled when they might impact the timing of critical tasks. This register is only accessible in privileged mode. The MSR and MRS instructions are used to access the **PRIMASK** register, and the CPS instruction may be used to change the value of the **PRIMASK** register. See the *Cortex™-M3 Instruction Set Technical User's Manual* for more information on these instructions. For more information on exception priority levels, see “Exception Types” on page 67.

Priority Mask Register (PRIMASK)

Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | PRIMASK | R/W | 0 | Priority Mask |

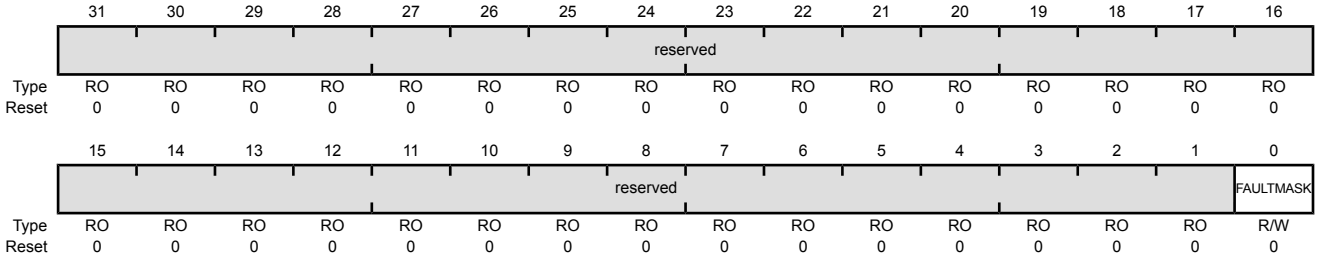
| Value | Description |
|---|---|
| 1 | Prevents the activation of all exceptions with configurable priority. |
| 0 | No effect. |

### Register 19: Fault Mask Register (FAULTMASK)

The **FAULTMASK** register prevents activation of all exceptions except for the Non-Maskable Interrupt (NMI). Exceptions should be disabled when they might impact the timing of critical tasks. This register is only accessible in privileged mode. The MSR and MRS instructions are used to access the **FAULTMASK** register, and the CPS instruction may be used to change the value of the **FAULTMASK** register. See the *Cortex™-M3 Instruction Set Technical User's Manual* for more information on these instructions. For more information on exception priority levels, see “Exception Types” on page 67.

Fault Mask Register (FAULTMASK)

Type R/W, reset 0x0000.0000



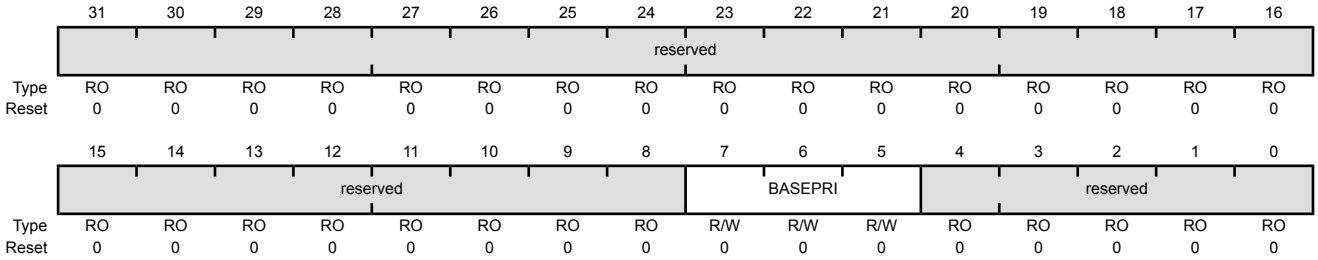

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | FAULTMASK | R/W | 0 | Fault Mask<br><br>Value Description<br>1 Prevents the activation of all exceptions except for NMI.<br>0 No effect.<br><br>The processor clears the FAULTMASK bit on exit from any exception handler except the NMI handler. |

### Register 20: Base Priority Mask Register (BASEPRI)

The **BASEPRI** register defines the minimum priority for exception processing. When **BASEPRI** is set to a nonzero value, it prevents the activation of all exceptions with the same or lower priority level as the **BASEPRI** value. Exceptions should be disabled when they might impact the timing of critical tasks. This register is only accessible in privileged mode. For more information on exception priority levels, see “Exception Types” on page 67.

Base Priority Mask Register (BASEPRI)

Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:5 | BASEPRI | R/W | 0x0 | Base Priority<br>Any exception that has a programmable priority level with the same or lower priority as the value of this field is masked. The **PRIMASK** register can be used to mask all exceptions with programmable priority levels. Higher priority exceptions have lower priority levels.<br><br>Value Description<br>0x0 All exceptions are unmasked.<br>0x1 All exceptions with priority level 1-7 are masked.<br>0x2 All exceptions with priority level 2-7 are masked.<br>0x3 All exceptions with priority level 3-7 are masked.<br>0x4 All exceptions with priority level 4-7 are masked.<br>0x5 All exceptions with priority level 5-7 are masked.<br>0x6 All exceptions with priority level 6-7 are masked.<br>0x7 All exceptions with priority level 7 are masked. |
| 4:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

### Register 21: Control Register (CONTROL)

The **CONTROL** register controls the stack used and the privilege level for software execution when the processor is in Thread mode. This register is only accessible in privileged mode.

Handler mode always uses **MSP**, so the processor ignores explicit writes to the ASP bit of the **CONTROL** register when in Handler mode. The exception entry and return mechanisms automatically update the **CONTROL** register based on the EXC_RETURN value (see Table 2-10 on page 73). In an OS environment, threads running in Thread mode should use the process stack and the kernel and exception handlers should use the main stack. By default, Thread mode uses **MSP**. To switch the stack pointer used in Thread mode to **PSP**, either use the MSR instruction to set the ASP bit, as detailed in the *Cortex™-M3 Instruction Set Technical User's Manual*, or perform an exception return to Thread mode with the appropriate EXC_RETURN value, as shown in Table 2-10 on page 73.

**Note:** When changing the stack pointer, software must use an ISB instruction immediately after the MSR instruction, ensuring that instructions after the ISB execute use the new stack pointer. See the *Cortex™-M3 Instruction Set Technical User's Manual*.

Control Register (CONTROL)

Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | ASP | TMPL |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | ASP | R/W | 0 | Active Stack Pointer<br><br>Value Description<br>1 **PSP** is the current stack pointer.<br>0 **MSP** is the current stack pointer<br><br>In Handler mode, this bit reads as zero and ignores writes. The Cortex-M3 updates this bit automatically on exception return. |
| 0 | TMPL | R/W | 0 | Thread Mode Privilege Level<br><br>Value Description<br>1 Unprivileged software can be executed in Thread mode.<br>0 Only privileged software can be executed in Thread mode. |

### 2.3.5 Exceptions and Interrupts

The Cortex-M3 processor supports interrupts and system exceptions. The processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions. An exception changes the normal flow of software control. The processor uses Handler mode to handle all exceptions except for reset. See “Exception Entry and Return” on page 71 for more information.

The NVIC registers control interrupt handling. See “Nested Vectored Interrupt Controller (NVIC)” on page 82 for more information.

### 2.3.6 Data Types

The Cortex-M3 supports 32-bit words, 16-bit halfwords, and 8-bit bytes. The processor also supports 64-bit data transfer instructions. All instruction and data memory accesses are little endian. See “Memory Regions, Types and Attributes” on page 59 for more information.

## 2.4 Memory Model

This section describes the processor memory map, the behavior of memory accesses, and the bit-banding features. The processor has a fixed memory map that provides up to 4 GB of addressable memory.

The memory map for the LM3S828 controller is provided in Table 2-4 on page 58. In this manual, register addresses are given as a hexadecimal increment, relative to the module’s base address as shown in the memory map.

The regions for SRAM and peripherals include bit-band regions. Bit-banding provides atomic operations to bit data (see “Bit-Banding” on page 62).

The processor reserves regions of the Private peripheral bus (PPB) address range for core peripheral registers (see “Cortex-M3 Peripherals” on page 81).

**Note:** Within the memory map, all reserved space returns a bus fault when read or written.

**Table 2-4. Memory Map**

| Start | End | Description | For details, see page ... |
|---|---|---|---|
| **Memory** | | | |
| 0x0000.0000 | 0x0000.FFFF | On-chip Flash | 204 |
| 0x0001.0000 | 0x1FFF.FFFF | Reserved | - |
| 0x2000.0000 | 0x2000.1FFF | Bit-banded on-chip SRAM | 203 |
| 0x2000.2000 | 0x21FF.FFFF | Reserved | - |
| 0x2200.0000 | 0x2203.FFFF | Bit-band alias of 0x2000.0000 through 0x200F.FFFF | 203 |
| 0x2204.0000 | 0x3FFF.FFFF | Reserved | - |
| **FiRM Peripherals** | | | |
| 0x4000.0000 | 0x4000.0FFF | Watchdog timer 0 | 298 |
| 0x4000.1000 | 0x4000.3FFF | Reserved | - |
| 0x4000.4000 | 0x4000.4FFF | GPIO Port A | 227 |
| 0x4000.5000 | 0x4000.5FFF | GPIO Port B | 227 |
| 0x4000.6000 | 0x4000.6FFF | GPIO Port C | 227 |
| 0x4000.7000 | 0x4000.7FFF | GPIO Port D | 227 |
| 0x4000.8000 | 0x4000.8FFF | SSI0 | 404 |

**Table 2-4. Memory Map *(continued)***

| Start | End | Description | For details, see page ... |
|---|---|---|---|
| 0x4000.9000 | 0x4000.BFFF | Reserved | - |
| 0x4000.C000 | 0x4000.CFFF | UART0 | 360 |
| 0x4000.D000 | 0x4000.DFFF | UART1 | 360 |
| 0x4000.E000 | 0x4001.FFFF | Reserved | - |
| **Peripherals** | | | |
| 0x4002.0000 | 0x4002.0FFF | $I^2C$ 0 | 444 |
| 0x4002.1000 | 0x4002.3FFF | Reserved | - |
| 0x4002.4000 | 0x4002.4FFF | GPIO Port E | 227 |
| 0x4002.5000 | 0x4002.FFFF | Reserved | - |
| 0x4003.0000 | 0x4003.0FFF | Timer 0 | 270 |
| 0x4003.1000 | 0x4003.1FFF | Timer 1 | 270 |
| 0x4003.2000 | 0x4003.2FFF | Timer 2 | 270 |
| 0x4003.3000 | 0x4003.7FFF | Reserved | - |
| 0x4003.8000 | 0x4003.8FFF | ADC0 | 327 |
| 0x4003.9000 | 0x400F.CFFF | Reserved | - |
| 0x400F.D000 | 0x400F.DFFF | Flash memory control | 208 |
| 0x400F.E000 | 0x400F.EFFF | System control | 160 |
| 0x400F.F000 | 0x41FF.FFFF | Reserved | - |
| 0x4200.0000 | 0x43FF.FFFF | Bit-banded alias of 0x4000.0000 through 0x400F.FFFF | - |
| 0x4400.0000 | 0xDFFF.FFFF | Reserved | - |
| **Private Peripheral Bus** | | | |
| 0xE000.0000 | 0xE000.0FFF | Instrumentation Trace Macrocell (ITM) | 41 |
| 0xE000.1000 | 0xE000.1FFF | Data Watchpoint and Trace (DWT) | 41 |
| 0xE000.2000 | 0xE000.2FFF | Flash Patch and Breakpoint (FPB) | 41 |
| 0xE000.3000 | 0xE000.DFFF | Reserved | - |
| 0xE000.E000 | 0xE000.EFFF | Cortex-M3 Peripherals (SysTick, NVIC, SCB, and MPU) | 66 |
| 0xE000.F000 | 0xE003.FFFF | Reserved | - |
| 0xE004.0000 | 0xE004.0FFF | Trace Port Interface Unit (TPIU) | 42 |
| 0xE004.1000 | 0xFFFF.FFFF | Reserved | - |

### 2.4.1 Memory Regions, Types and Attributes

The memory map and the programming of the MPU split the memory map into regions. Each region has a defined memory type, and some regions have additional memory attributes. The memory type and attributes determine the behavior of accesses to the region.

The memory types are:

- Normal: The processor can re-order transactions for efficiency and perform speculative reads.
- Device: The processor preserves transaction order relative to other transactions to Device or Strongly Ordered memory.
- Strongly Ordered: The processor preserves transaction order relative to all other transactions.

The different ordering requirements for Device and Strongly Ordered memory mean that the memory system can buffer a write to Device memory but must not buffer a write to Strongly Ordered memory.

An additional memory attribute is Execute Never (XN), which means the processor prevents instruction accesses. A fault exception is generated only on execution of an instruction executed from an XN region.

### 2.4.2 Memory System Ordering of Memory Accesses

For most memory accesses caused by explicit memory access instructions, the memory system does not guarantee that the order in which the accesses complete matches the program order of the instructions, providing the order does not affect the behavior of the instruction sequence. Normally, if correct program execution depends on two memory accesses completing in program order, software must insert a memory barrier instruction between the memory access instructions (see “Software Ordering of Memory Accesses” on page 61).

However, the memory system does guarantee ordering of accesses to Device and Strongly Ordered memory. For two memory access instructions A1 and A2, if both A1 and A2 are accesses to either Device or Strongly Ordered memory, and if A1 occurs before A2 in program order, A1 is always observed before A2.

### 2.4.3 Behavior of Memory Accesses

Table 2-5 on page 60 shows the behavior of accesses to each region in the memory map. See “Memory Regions, Types and Attributes” on page 59 for more information on memory types and the XN attribute. Stellaris devices may have reserved memory areas within the address ranges shown below (refer to Table 2-4 on page 58 for more information).

**Table 2-5. Memory Access Behavior**

| Address Range | Memory Region | Memory Type | Execute Never (XN) | Description |
|---|---|---|---|---|
| 0x0000.0000 - 0x1FFF.FFFF | Code | Normal | - | This executable region is for program code. Data can also be stored here. |
| 0x2000.0000 - 0x3FFF.FFFF | SRAM | Normal | - | This executable region is for data. Code can also be stored here. This region includes bit band and bit band alias areas (see Table 2-6 on page 62). |
| 0x4000.0000 - 0x5FFF.FFFF | Peripheral | Device | XN | This region includes bit band and bit band alias areas (see Table 2-7 on page 62). |
| 0x6000.0000 - 0x9FFF.FFFF | External RAM | Normal | - | This executable region is for data. |
| 0xA000.0000 - 0xDFFF.FFFF | External device | Device | XN | This region is for external device memory. |
| 0xE000.0000- 0xE00F.FFFF | Private peripheral bus | Strongly Ordered | XN | This region includes the NVIC, system timer, and system control block. |
| 0xE010.0000- 0xFFFF.FFFF | Reserved | - | - | - |

The Code, SRAM, and external RAM regions can hold programs. However, it is recommended that programs always use the Code region because the Cortex-M3 has separate buses that can perform instruction fetches and data accesses simultaneously.

The MPU can override the default memory access behavior described in this section. For more information, see “Memory Protection Unit (MPU)” on page 84.

The Cortex-M3 prefetches instructions ahead of execution and speculatively prefetches from branch target addresses.

### 2.4.4 Software Ordering of Memory Accesses

The order of instructions in the program flow does not always guarantee the order of the corresponding memory transactions for the following reasons:

- The processor can reorder some memory accesses to improve efficiency, providing this does not affect the behavior of the instruction sequence.
- The processor has multiple bus interfaces.
- Memory or devices in the memory map have different wait states.
- Some memory accesses are buffered or speculative.

"Memory System Ordering of Memory Accesses" on page 60 describes the cases where the memory system guarantees the order of memory accesses. Otherwise, if the order of memory accesses is critical, software must include memory barrier instructions to force that ordering. The Cortex-M3 has the following memory barrier instructions:

- The Data Memory Barrier (DMB) instruction ensures that outstanding memory transactions complete before subsequent memory transactions.
- The Data Synchronization Barrier (DSB) instruction ensures that outstanding memory transactions complete before subsequent instructions execute.
- The Instruction Synchronization Barrier (ISB) instruction ensures that the effect of all completed memory transactions is recognizable by subsequent instructions.

Memory barrier instructions can be used in the following situations:

- MPU programming
  - If the MPU settings are changed and the change must be effective on the very next instruction, use a DSB instruction to ensure the effect of the MPU takes place immediately at the end of context switching.
  - Use an ISB instruction to ensure the new MPU setting takes effect immediately after programming the MPU region or regions, if the MPU configuration code was accessed using a branch or call. If the MPU configuration code is entered using exception mechanisms, then an ISB instruction is not required.
- Vector table

  If the program changes an entry in the vector table and then enables the corresponding exception, use a DMB instruction between the operations. The DMB instruction ensures that if the exception is taken immediately after being enabled, the processor uses the new exception vector.
- Self-modifying code

  If a program contains self-modifying code, use an ISB instruction immediately after the code modification in the program. The ISB instruction ensures subsequent instruction execution uses the updated program.
- Memory map switching

If the system contains a memory map switching mechanism, use a DSB instruction after switching the memory map in the program. The DSB instruction ensures subsequent instruction execution uses the updated memory map.

- Dynamic exception priority change

  When an exception priority has to change when the exception is pending or active, use DSB instructions after the change. The change then takes effect on completion of the DSB instruction.

Memory accesses to Strongly Ordered memory, such as the System Control Block, do not require the use of DMB instructions.

For more information on the memory barrier instructions, see the *Cortex™-M3 Instruction Set Technical User's Manual*.

### 2.4.5 Bit-Banding

A bit-band region maps each word in a bit-band alias region to a single bit in the bit-band region. The bit-band regions occupy the lowest 1 MB of the SRAM and peripheral memory regions. Accesses to the 32-MB SRAM alias region map to the 1-MB SRAM bit-band region, as shown in Table 2-6 on page 62. Accesses to the 32-MB peripheral alias region map to the 1-MB peripheral bit-band region, as shown in Table 2-7 on page 62. For the specific address range of the bit-band regions, see Table 2-4 on page 58.

**Note:** A word access to the SRAM or the peripheral bit-band alias region maps to a single bit in the SRAM or peripheral bit-band region.

A word access to a bit band address results in a word access to the underlying memory, and similarly for halfword and byte accesses. This allows bit band accesses to match the access requirements of the underlying peripheral.

**Table 2-6. SRAM Memory Bit-Banding Regions**

| Address Range | Memory Region | Instruction and Data Accesses |
|---|---|---|
| 0x2000.0000 - 0x200F.FFFF | SRAM bit-band region | Direct accesses to this memory range behave as SRAM memory accesses, but this region is also bit addressable through bit-band alias. |
| 0x2200.0000 - 0x23FF.FFFF | SRAM bit-band alias | Data accesses to this region are remapped to bit band region. A write operation is performed as read-modify-write. Instruction accesses are not remapped. |

**Table 2-7. Peripheral Memory Bit-Banding Regions**

| Address Range | Memory Region | Instruction and Data Accesses |
|---|---|---|
| 0x4000.0000 - 0x400F.FFFF | Peripheral bit-band region | Direct accesses to this memory range behave as peripheral memory accesses, but this region is also bit addressable through bit-band alias. |
| 0x4200.0000 - 0x43FF.FFFF | Peripheral bit-band alias | Data accesses to this region are remapped to bit band region. A write operation is performed as read-modify-write. Instruction accesses are not permitted. |

The following formula shows how the alias region maps onto the bit-band region:

```
bit_word_offset = (byte_offset x 32) + (bit_number x 4)

bit_word_addr = bit_band_base + bit_word_offset
```

where:

bit_word_offset
The position of the target bit in the bit-band memory region.

bit_word_addr
The address of the word in the alias memory region that maps to the targeted bit.

bit_band_base
The starting address of the alias region.

byte_offset
The number of the byte in the bit-band region that contains the targeted bit.

bit_number
The bit position, 0-7, of the targeted bit.

Figure 2-4 on page 64 shows examples of bit-band mapping between the SRAM bit-band alias region and the SRAM bit-band region:

- The alias word at 0x23FF.FFE0 maps to bit 0 of the bit-band byte at 0x200F.FFFF:

  ```
  0x23FF.FFE0 = 0x2200.0000 + (0x000F.FFFF*32) + (0*4)
  ```

- The alias word at 0x23FF.FFFC maps to bit 7 of the bit-band byte at 0x200F.FFFF:

  ```
  0x23FF.FFFC = 0x2200.0000 + (0x000F.FFFF*32) + (7*4)
  ```

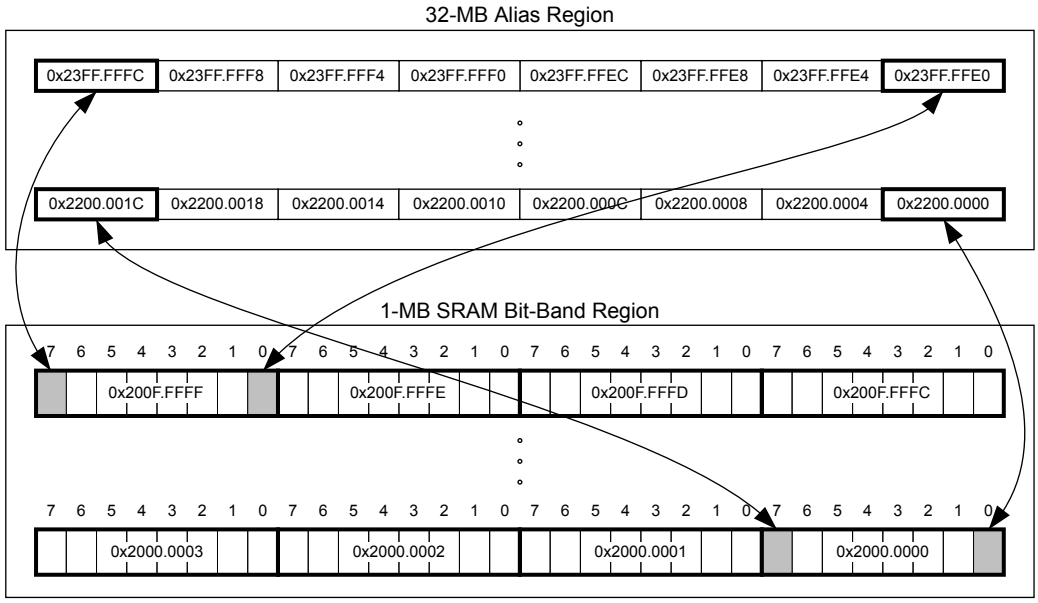
- The alias word at 0x2200.0000 maps to bit 0 of the bit-band byte at 0x2000.0000:

  ```
  0x2200.0000 = 0x2200.0000 + (0*32) + (0*4)
  ```

- The alias word at 0x2200.001C maps to bit 7 of the bit-band byte at 0x2000.0000:

  ```
  0x2200.001C = 0x2200.0000+ (0*32) + (7*4)
  ```

**Figure 2-4. Bit-Band Mapping**




### 2.4.5.1 Directly Accessing an Alias Region

Writing to a word in the alias region updates a single bit in the bit-band region.

Bit 0 of the value written to a word in the alias region determines the value written to the targeted bit in the bit-band region. Writing a value with bit 0 set writes a 1 to the bit-band bit, and writing a value with bit 0 clear writes a 0 to the bit-band bit.

Bits 31:1 of the alias word have no effect on the bit-band bit. Writing 0x01 has the same effect as writing 0xFF. Writing 0x00 has the same effect as writing 0x0E.

When reading a word in the alias region, 0x0000.0000 indicates that the targeted bit in the bit-band region is clear and 0x0000.0001 indicates that the targeted bit in the bit-band region is set.

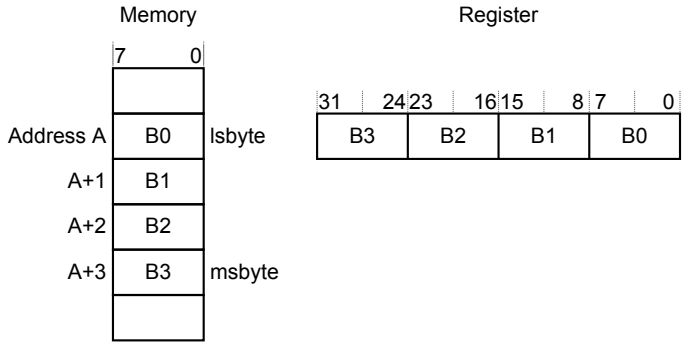
### 2.4.5.2 Directly Accessing a Bit-Band Region

"Behavior of Memory Accesses" on page 60 describes the behavior of direct byte, halfword, or word accesses to the bit-band regions.

## 2.4.6 Data Storage

The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 hold the first stored word, and bytes 4-7 hold the second stored word. Data is stored in little-endian format, with the least-significant byte (lsbyte) of a word stored at the lowest-numbered byte, and the most-significant byte (msbyte) stored at the highest-numbered byte. Figure 2-5 on page 65 illustrates how data is stored.

**Figure 2-5. Data Storage**




## 2.4.7 Synchronization Primitives

The Cortex-M3 instruction set includes pairs of synchronization primitives which provide a non-blocking mechanism that a thread or process can use to obtain exclusive access to a memory location. Software can use these primitives to perform a guaranteed read-modify-write memory update sequence or for a semaphore mechanism.

A pair of synchronization primitives consists of:

- A Load-Exclusive instruction, which is used to read the value of a memory location and requests exclusive access to that location.
- A Store-Exclusive instruction, which is used to attempt to write to the same memory location and returns a status bit to a register. If this status bit is clear, it indicates that the thread or process gained exclusive access to the memory and the write succeeds; if this status bit is set, it indicates that the thread or process did not gain exclusive access to the memory and no write is performed.

The pairs of Load-Exclusive and Store-Exclusive instructions are:

- The word instructions LDREX and STREX
- The halfword instructions LDREXH and STREXH
- The byte instructions LDREXB and STREXB

Software must use a Load-Exclusive instruction with the corresponding Store-Exclusive instruction.

To perform a guaranteed read-modify-write of a memory location, software must:

1. Use a Load-Exclusive instruction to read the value of the location.
2. Update the value, as required.
3. Use a Store-Exclusive instruction to attempt to write the new value back to the memory location, and test the returned status bit. If the status bit is clear, the read-modify-write completed successfully; if the status bit is set, no write was performed, which indicates that the value returned at step 1 might be out of date. The software must retry the read-modify-write sequence.

Software can use the synchronization primitives to implement a semaphore as follows:

1. Use a Load-Exclusive instruction to read from the semaphore address to check whether the semaphore is free.

2. If the semaphore is free, use a Store-Exclusive to write the claim value to the semaphore address.

3. If the returned status bit from step 2 indicates that the Store-Exclusive succeeded, then the software has claimed the semaphore. However, if the Store-Exclusive failed, another process might have claimed the semaphore after the software performed step 1.

The Cortex-M3 includes an exclusive access monitor that tags the fact that the processor has executed a Load-Exclusive instruction. The processor removes its exclusive access tag if:

- It executes a CLREX instruction.
- It executes a Store-Exclusive instruction, regardless of whether the write succeeds.
- An exception occurs, which means the processor can resolve semaphore conflicts between different threads.

For more information about the synchronization primitive instructions, see the *Cortex™-M3 Instruction Set Technical User's Manual*.

## 2.5 Exception Model

The ARM Cortex-M3 processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions in Handler Mode. The processor state is automatically stored to the stack on an exception and automatically restored from the stack at the end of the Interrupt Service Routine (ISR). The vector is fetched in parallel to the state saving, enabling efficient interrupt entry. The processor supports tail-chaining, which enables back-to-back interrupts to be performed without the overhead of state saving and restoration.

Table 2-8 on page 68 lists all exception types. Software can set eight priority levels on seven of these exceptions (system handlers) as well as on 22 interrupts (listed in Table 2-9 on page 69).

Priorities on the system handlers are set with the NVIC **System Handler Priority n (SYSPRIn)** registers. Interrupts are enabled through the NVIC **Interrupt Set Enable n (ENn)** register and prioritized with the NVIC **Interrupt Priority n (PRIn)** registers. Priorities can be grouped by splitting priority levels into preemption priorities and subpriorities. All the interrupt registers are described in “Nested Vectored Interrupt Controller (NVIC)” on page 82.

Internally, the highest user-programmable priority (0) is treated as fourth priority, after a Reset, Non-Maskable Interrupt (NMI), and a Hard Fault, in that order. Note that 0 is the default priority for all the programmable priorities.

**Important:** After a write to clear an interrupt source, it may take several processor cycles for the NVIC to see the interrupt source de-assert. Thus if the interrupt clear is done as the last action in an interrupt handler, it is possible for the interrupt handler to complete while the NVIC sees the interrupt as still asserted, causing the interrupt handler to be re-entered errantly. This situation can be avoided by either clearing the interrupt source at the beginning of the interrupt handler or by performing a read or write after the write to clear the interrupt source (and flush the write buffer).

See “Nested Vectored Interrupt Controller (NVIC)” on page 82 for more information on exceptions and interrupts.

### 2.5.1 Exception States

Each exception is in one of the following states:

- **Inactive.** The exception is not active and not pending.
- **Pending.** The exception is waiting to be serviced by the processor. An interrupt request from a peripheral or from software can change the state of the corresponding interrupt to pending.
- **Active.** An exception that is being serviced by the processor but has not completed.

  **Note:** An exception handler can interrupt the execution of another exception handler. In this case, both exceptions are in the active state.

- **Active and Pending.** The exception is being serviced by the processor, and there is a pending exception from the same source.

### 2.5.2 Exception Types

The exception types are:

- **Reset.** Reset is invoked on power up or a warm reset. The exception model treats reset as a special form of exception. When reset is asserted, the operation of the processor stops, potentially at any point in an instruction. When reset is deasserted, execution restarts from the address provided by the reset entry in the vector table. Execution restarts as privileged execution in Thread mode.
- **NMI.** A non-maskable Interrupt (NMI) can be signaled using the NMI signal or triggered by software using the **Interrupt Control and State (INTCTRL)** register. This exception has the highest priority other than reset. NMI is permanently enabled and has a fixed priority of -2. NMIs cannot be masked or prevented from activation by any other exception or preempted by any exception other than reset.
- **Hard Fault.** A hard fault is an exception that occurs because of an error during exception processing, or because an exception cannot be managed by any other exception mechanism. Hard faults have a fixed priority of -1, meaning they have higher priority than any exception with configurable priority.
- **Memory Management Fault.** A memory management fault is an exception that occurs because of a memory protection related fault, including access violation and no match. The MPU or the fixed memory protection constraints determine this fault, for both instruction and data memory transactions. This fault is used to abort instruction accesses to Execute Never (XN) memory regions, even if the MPU is disabled.
- **Bus Fault.** A bus fault is an exception that occurs because of a memory-related fault for an instruction or data memory transaction such as a prefetch fault or a memory access fault. This fault can be enabled or disabled.
- **Usage Fault.** A usage fault is an exception that occurs because of a fault related to instruction execution, such as:
  - An undefined instruction
  - An illegal unaligned access
  - Invalid state on instruction execution
  - An error on exception return

An unaligned address on a word or halfword memory access or division by zero can cause a usage fault when the core is properly configured.

- **SVCall.** A supervisor call (SVC) is an exception that is triggered by the SVC instruction. In an OS environment, applications can use SVC instructions to access OS kernel functions and device drivers.
- **Debug Monitor.** This exception is caused by the debug monitor (when not halting). This exception is only active when enabled. This exception does not activate if it is a lower priority than the current activation.
- **PendSV.** PendSV is a pendable, interrupt-driven request for system-level service. In an OS environment, use PendSV for context switching when no other exception is active. PendSV is triggered using the **Interrupt Control and State (INTCTRL)** register.
- **SysTick.** A SysTick exception is an exception that the system timer generates when it reaches zero when it is enabled to generate an interrupt. Software can also generate a SysTick exception using the **Interrupt Control and State (INTCTRL)** register. In an OS environment, the processor can use this exception as system tick.
- **Interrupt (IRQ).** An interrupt, or IRQ, is an exception signaled by a peripheral or generated by a software request and fed through the NVIC (prioritized). All interrupts are asynchronous to instruction execution. In the system, peripherals use interrupts to communicate with the processor. Table 2-9 on page 69 lists the interrupts on the LM3S828 controller.

For an asynchronous exception, other than reset, the processor can execute another instruction between when the exception is triggered and when the processor enters the exception handler.

Privileged software can disable the exceptions that Table 2-8 on page 68 shows as having configurable priority (see the **SYSHNDCTRL** register on page 117 and the **DIS0** register on page 96).

For more information about hard faults, memory management faults, bus faults, and usage faults, see “Fault Handling” on page 73.

**Table 2-8. Exception Types**

| Exception Type | Vector Number | Priority[a] | Vector Address or Offset[b] | Activation |
|---|---|---|---|---|
| - | 0 | - | 0x0000.0000 | Stack top is loaded from the first entry of the vector table on reset. |
| Reset | 1 | -3 (highest) | 0x0000.0004 | Asynchronous |
| Non-Maskable Interrupt (NMI) | 2 | -2 | 0x0000.0008 | Asynchronous |
| Hard Fault | 3 | -1 | 0x0000.000C | - |
| Memory Management | 4 | programmable[c] | 0x0000.0010 | Synchronous |
| Bus Fault | 5 | programmable[c] | 0x0000.0014 | Synchronous when precise and asynchronous when imprecise |
| Usage Fault | 6 | programmable[c] | 0x0000.0018 | Synchronous |
| - | 7-10 | - | - | Reserved |
| SVCall | 11 | programmable[c] | 0x0000.002C | Synchronous |
| Debug Monitor | 12 | programmable[c] | 0x0000.0030 | Synchronous |
| - | 13 | - | - | Reserved |
| PendSV | 14 | programmable[c] | 0x0000.0038 | Asynchronous |

**Table 2-8. Exception Types *(continued)***

| Exception Type | Vector Number | Priority[a] | Vector Address or Offset[b] | Activation |
|---|---|---|---|---|
| SysTick | 15 | programmable[c] | 0x0000.003C | Asynchronous |
| Interrupts | 16 and above | programmable[d] | 0x0000.0040 and above | Asynchronous |

a. 0 is the default priority for all the programmable priorities.

b. See “Vector Table” on page 70.

c. See **SYSPRI1** on page 114.

d. See **PRIn** registers on page 100.

**Table 2-9. Interrupts**

| Vector Number | Interrupt Number (Bit in Interrupt Registers) | Vector Address or Offset | Description |
|---|---|---|---|
| 0-15 | - | 0x0000.0000 - 0x0000.003C | Processor exceptions |
| 16 | 0 | 0x0000.0040 | GPIO Port A |
| 17 | 1 | 0x0000.0044 | GPIO Port B |
| 18 | 2 | 0x0000.0048 | GPIO Port C |
| 19 | 3 | 0x0000.004C | GPIO Port D |
| 20 | 4 | 0x0000.0050 | GPIO Port E |
| 21 | 5 | 0x0000.0054 | UART0 |
| 22 | 6 | 0x0000.0058 | UART1 |
| 23 | 7 | 0x0000.005C | SSI0 |
| 24 | 8 | 0x0000.0060 | $I^2C0$ |
| 25-29 | 9-13 | - | Reserved |
| 30 | 14 | 0x0000.0078 | ADC0 Sequence 0 |
| 31 | 15 | 0x0000.007C | ADC0 Sequence 1 |
| 32 | 16 | 0x0000.0080 | ADC0 Sequence 2 |
| 33 | 17 | 0x0000.0084 | ADC0 Sequence 3 |
| 34 | 18 | 0x0000.0088 | Watchdog Timer 0 |
| 35 | 19 | 0x0000.008C | Timer 0A |
| 36 | 20 | 0x0000.0090 | Timer 0B |
| 37 | 21 | 0x0000.0094 | Timer 1A |
| 38 | 22 | 0x0000.0098 | Timer 1B |
| 39 | 23 | 0x0000.009C | Timer 2A |
| 40 | 24 | 0x0000.00A0 | Timer 2B |
| 41-43 | 25-27 | - | Reserved |
| 44 | 28 | 0x0000.00B0 | System Control |
| 45 | 29 | 0x0000.00B4 | Flash Memory Control |

### 2.5.3 Exception Handlers

The processor handles exceptions using:

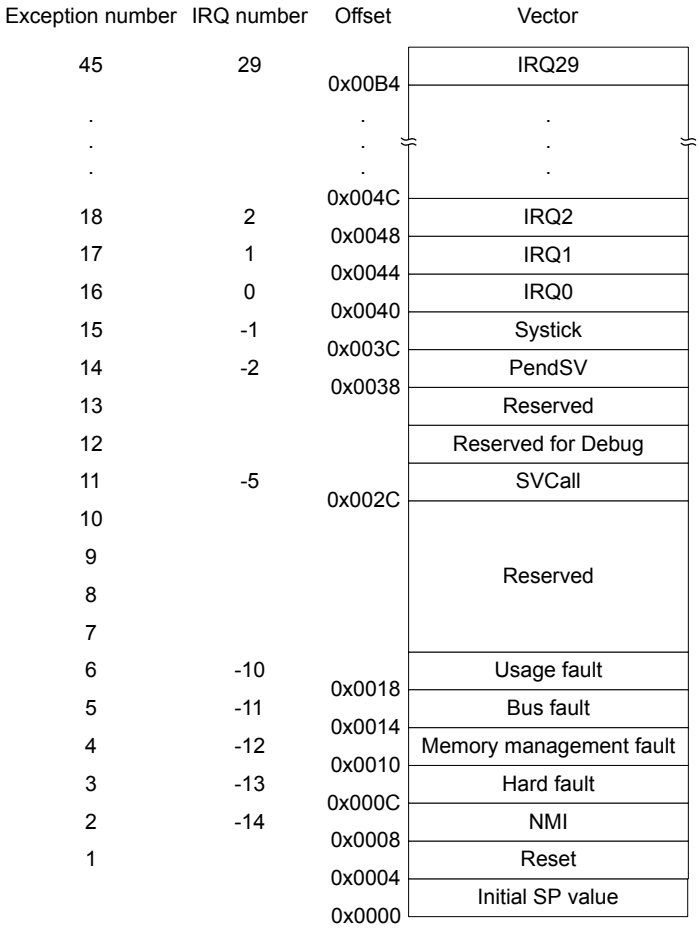
- **Interrupt Service Routines (ISRs).** Interrupts (IRQx) are the exceptions handled by ISRs.

- **Fault Handlers.** Hard fault, memory management fault, usage fault, and bus fault are fault exceptions handled by the fault handlers.
- **System Handlers.** NMI, PendSV, SVCall, SysTick, and the fault exceptions are all system exceptions that are handled by system handlers.

### 2.5.4 Vector Table

The vector table contains the reset value of the stack pointer and the start addresses, also called exception vectors, for all exception handlers. The vector table is constructed using the vector address or offset shown in Table 2-8 on page 68. Figure 2-6 on page 70 shows the order of the exception vectors in the vector table. The least-significant bit of each vector must be 1, indicating that the exception handler is Thumb code

**Figure 2-6. Vector table**




On system reset, the vector table is fixed at address 0x0000.0000. Privileged software can write to the **Vector Table Offset (VTABLE)** register to relocate the vector table start address to a different memory location, in the range 0x0000.0100 to 0x3FFF.FF00 (see “Vector Table” on page 70). Note that when configuring the **VTABLE** register, the offset must be aligned on a 256-byte boundary.

### 2.5.5 Exception Priorities

As Table 2-8 on page 68 shows, all exceptions have an associated priority, with a lower priority value indicating a higher priority and configurable priorities for all exceptions except Reset, Hard fault, and NMI. If software does not configure any priorities, then all exceptions with a configurable priority have a priority of 0. For information about configuring exception priorities, see page 114 and page 100.

**Note:** Configurable priority values for the Stellaris implementation are in the range 0-7. This means that the Reset, Hard fault, and NMI exceptions, with fixed negative priority values, always have higher priority than any other exception.

For example, assigning a higher priority value to IRQ[0] and a lower priority value to IRQ[1] means that IRQ[1] has higher priority than IRQ[0]. If both IRQ[1] and IRQ[0] are asserted, IRQ[1] is processed before IRQ[0].

If multiple pending exceptions have the same priority, the pending exception with the lowest exception number takes precedence. For example, if both IRQ[0] and IRQ[1] are pending and have the same priority, then IRQ[0] is processed before IRQ[1].

When the processor is executing an exception handler, the exception handler is preempted if a higher priority exception occurs. If an exception occurs with the same priority as the exception being handled, the handler is not preempted, irrespective of the exception number. However, the status of the new interrupt changes to pending.

### 2.5.6 Interrupt Priority Grouping

To increase priority control in systems with interrupts, the NVIC supports priority grouping. This grouping divides each interrupt priority register entry into two fields:

- An upper field that defines the group priority
- A lower field that defines a subpriority within the group

Only the group priority determines preemption of interrupt exceptions. When the processor is executing an interrupt exception handler, another interrupt with the same group priority as the interrupt being handled does not preempt the handler.

If multiple pending interrupts have the same group priority, the subpriority field determines the order in which they are processed. If multiple pending interrupts have the same group priority and subpriority, the interrupt with the lowest IRQ number is processed first.

For information about splitting the interrupt priority fields into group priority and subpriority, see page 108.

### 2.5.7 Exception Entry and Return

Descriptions of exception handling use the following terms:

- **Preemption.** When the processor is executing an exception handler, an exception can preempt the exception handler if its priority is higher than the priority of the exception being handled. See “Interrupt Priority Grouping” on page 71 for more information about preemption by an interrupt. When one exception preempts another, the exceptions are called nested exceptions. See “Exception Entry” on page 72 more information.
- **Return.** Return occurs when the exception handler is completed, and there is no pending exception with sufficient priority to be serviced and the completed exception handler was not

handling a late-arriving exception. The processor pops the stack and restores the processor state to the state it had before the interrupt occurred. See “Exception Return” on page 73 for more information.

- **Tail-Chaining.** This mechanism speeds up exception servicing. On completion of an exception handler, if there is a pending exception that meets the requirements for exception entry, the stack pop is skipped and control transfers to the new exception handler.
- **Late-Arriving.** This mechanism speeds up preemption. If a higher priority exception occurs during state saving for a previous exception, the processor switches to handle the higher priority exception and initiates the vector fetch for that exception. State saving is not affected by late arrival because the state saved is the same for both exceptions. Therefore, the state saving continues uninterrupted. The processor can accept a late arriving exception until the first instruction of the exception handler of the original exception enters the execute stage of the processor. On return from the exception handler of the late-arriving exception, the normal tail-chaining rules apply.

#### 2.5.7.1 Exception Entry

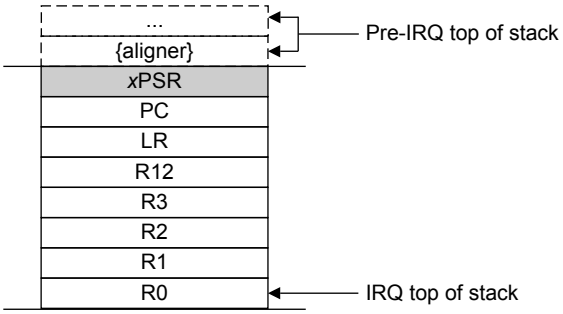
Exception entry occurs when there is a pending exception with sufficient priority and either the processor is in Thread mode or the new exception is of higher priority than the exception being handled, in which case the new exception preempts the original exception.

When one exception preempts another, the exceptions are nested.

Sufficient priority means the exception has more priority than any limits set by the mask registers (see **PRIMASK** on page 54, **FAULTMASK** on page 55, and **BASEPRI** on page 56). An exception with less priority than this is pending but is not handled by the processor.

When the processor takes an exception, unless the exception is a tail-chained or a late-arriving exception, the processor pushes information onto the current stack. This operation is referred to as *stacking* and the structure of eight data words is referred to as *stack frame*.

**Figure 2-7. Exception Stack Frame**




Immediately after stacking, the stack pointer indicates the lowest address in the stack frame. Unless stack alignment is disabled, the stack frame is aligned to a double-word address. If the STKALIGN bit of the **Configuration Control (CCR)** register is set, stack align adjustment is performed during stacking.

The stack frame includes the return address, which is the address of the next instruction in the interrupted program. This value is restored to the **PC** at exception return so that the interrupted program resumes.

In parallel to the stacking operation, the processor performs a vector fetch that reads the exception handler start address from the vector table. When stacking is complete, the processor starts executing the exception handler. At the same time, the processor writes an EXC_RETURN value to the **LR**, indicating which stack pointer corresponds to the stack frame and what operation mode the processor was in before the entry occurred.

If no higher-priority exception occurs during exception entry, the processor starts executing the exception handler and automatically changes the status of the corresponding pending interrupt to active.

If another higher-priority exception occurs during exception entry, known as late arrival, the processor starts executing the exception handler for this exception and does not change the pending status of the earlier exception.

#### 2.5.7.2 Exception Return

Exception return occurs when the processor is in Handler mode and executes one of the following instructions to load the EXC_RETURN value into the **PC**:

- An LDM or POP instruction that loads the **PC**
- A BX instruction using any register
- An LDR instruction with the **PC** as the destination

EXC_RETURN is the value loaded into the **LR** on exception entry. The exception mechanism relies on this value to detect when the processor has completed an exception handler. The lowest four bits of this value provide information on the return stack and processor mode. Table 2-10 on page 73 shows the EXC_RETURN values with a description of the exception return behavior.

EXC_RETURN bits 31:4 are all set. When this value is loaded into the **PC**, it indicates to the processor that the exception is complete, and the processor initiates the appropriate exception return sequence.

**Table 2-10. Exception Return Behavior**

| EXC_RETURN[31:0] | Description |
|---|---|
| 0xFFFF.FFF0 | Reserved |
| 0xFFFF.FFF1 | Return to Handler mode.<br>Exception return uses state from **MSP**.<br>Execution uses **MSP** after return. |
| 0xFFFF.FFF2 - 0xFFFF.FFF8 | Reserved |
| 0xFFFF.FFF9 | Return to Thread mode.<br>Exception return uses state from **MSP**.<br>Execution uses **MSP** after return. |
| 0xFFFF.FFFA - 0xFFFF.FFFC | Reserved |
| 0xFFFF.FFFD | Return to Thread mode.<br>Exception return uses state from **PSP**.<br>Execution uses **PSP** after return. |
| 0xFFFF.FFFE - 0xFFFF.FFFF | Reserved |

## 2.6 Fault Handling

Faults are a subset of the exceptions (see “Exception Model” on page 66). The following conditions generate a fault:

- A bus error on an instruction fetch or vector table load or a data access.
- An internally detected error such as an undefined instruction or an attempt to change state with a BX instruction.
- Attempting to execute an instruction from a memory region marked as Non-Executable (XN).
- An MPU fault because of a privilege violation or an attempt to access an unmanaged region.

### 2.6.1 Fault Types

Table 2-11 on page 74 shows the types of fault, the handler used for the fault, the corresponding fault status register, and the register bit that indicates the fault has occurred. See page 121 for more information about the fault status registers.

**Table 2-11. Faults**

| Fault | Handler | Fault Status Register | Bit Name |
|---|---|---|---|
| Bus error on a vector read | Hard fault | **Hard Fault Status (HFAULTSTAT)** | VECT |
| Fault escalated to a hard fault | Hard fault | **Hard Fault Status (HFAULTSTAT)** | FORCED |
| MPU or default memory mismatch on instruction access | Memory management fault | **Memory Management Fault Status (MFAULTSTAT)** | IERR [a] |
| MPU or default memory mismatch on data access | Memory management fault | **Memory Management Fault Status (MFAULTSTAT)** | DERR |
| MPU or default memory mismatch on exception stacking | Memory management fault | **Memory Management Fault Status (MFAULTSTAT)** | MSTKE |
| MPU or default memory mismatch on exception unstacking | Memory management fault | **Memory Management Fault Status (MFAULTSTAT)** | MUSTKE |
| Bus error during exception stacking | Bus fault | **Bus Fault Status (BFAULTSTAT)** | BSTKE |
| Bus error during exception unstacking | Bus fault | **Bus Fault Status (BFAULTSTAT)** | BUSTKE |
| Bus error during instruction prefetch | Bus fault | **Bus Fault Status (BFAULTSTAT)** | IBUS |
| Precise data bus error | Bus fault | **Bus Fault Status (BFAULTSTAT)** | PRECISE |
| Imprecise data bus error | Bus fault | **Bus Fault Status (BFAULTSTAT)** | IMPRE |
| Attempt to access a coprocessor | Usage fault | **Usage Fault Status (UFAULTSTAT)** | NOCP |
| Undefined instruction | Usage fault | **Usage Fault Status (UFAULTSTAT)** | UNDEF |
| Attempt to enter an invalid instruction set state [b] | Usage fault | **Usage Fault Status (UFAULTSTAT)** | INVSTAT |
| Invalid EXC_RETURN value | Usage fault | **Usage Fault Status (UFAULTSTAT)** | INVPC |
| Illegal unaligned load or store | Usage fault | **Usage Fault Status (UFAULTSTAT)** | UNALIGN |
| Divide by 0 | Usage fault | **Usage Fault Status (UFAULTSTAT)** | DIV0 |

a. Occurs on an access to an XN region even if the MPU is disabled.

b. Attempting to use an instruction set other than the Thumb instruction set, or returning to a non load-store-multiple instruction with ICI continuation.

### 2.6.2 Fault Escalation and Hard Faults

All fault exceptions except for hard fault have configurable exception priority (see **SYSPRI1** on page 114). Software can disable execution of the handlers for these faults (see **SYSHNDCTRL** on page 117).

Usually, the exception priority, together with the values of the exception mask registers, determines whether the processor enters the fault handler, and whether a fault handler can preempt another fault handler as described in “Exception Model” on page 66.

In some situations, a fault with configurable priority is treated as a hard fault. This process is called priority escalation, and the fault is described as *escalated to hard fault*. Escalation to hard fault occurs when:

- A fault handler causes the same kind of fault as the one it is servicing. This escalation to hard fault occurs because a fault handler cannot preempt itself because it must have the same priority as the current priority level.
- A fault handler causes a fault with the same or lower priority as the fault it is servicing. This situation happens because the handler for the new fault cannot preempt the currently executing fault handler.
- An exception handler causes a fault for which the priority is the same as or lower than the currently executing exception.
- A fault occurs and the handler for that fault is not enabled.

If a bus fault occurs during a stack push when entering a bus fault handler, the bus fault does not escalate to a hard fault. Thus if a corrupted stack causes a fault, the fault handler executes even though the stack push for the handler failed. The fault handler operates but the stack contents are corrupted.

**Note:** Only Reset and NMI can preempt the fixed priority hard fault. A hard fault can preempt any exception other than Reset, NMI, or another hard fault.

### 2.6.3 Fault Status Registers and Fault Address Registers

The fault status registers indicate the cause of a fault. For bus faults and memory management faults, the fault address register indicates the address accessed by the operation that caused the fault, as shown in Table 2-12 on page 75.

**Table 2-12. Fault Status and Fault Address Registers**

| Handler | Status Register Name | Address Register Name | Register Description |
|---|---|---|---|
| Hard fault | **Hard Fault Status (HFAULTSTAT)** | - | page 127 |
| Memory management fault | **Memory Management Fault Status (MFAULTSTAT)** | **Memory Management Fault Address (MMADDR)** | page 121<br>page 128 |
| Bus fault | **Bus Fault Status (BFAULTSTAT)** | **Bus Fault Address (FAULTADDR)** | page 121<br>page 129 |
| Usage fault | **Usage Fault Status (UFAULTSTAT)** | - | page 121 |

### 2.6.4 Lockup

The processor enters a lockup state if a hard fault occurs when executing the NMI or hard fault handlers. When the processor is in the lockup state, it does not execute any instructions. The processor remains in lockup state until it is reset or an NMI occurs.

**Note:** If the lockup state occurs from the NMI handler, a subsequent NMI does not cause the processor to leave the lockup state.

## 2.7 Power Management

The Cortex-M3 processor sleep modes reduce power consumption:

- Sleep mode stops the processor clock.
- Deep-sleep mode stops the system clock and switches off the PLL and Flash memory.

The SLEEPDEEP bit of the **System Control (SYSCTRL)** register selects which sleep mode is used (see page 110). For more information about the behavior of the sleep modes, see “System Control” on page 157.

This section describes the mechanisms for entering sleep mode and the conditions for waking up from sleep mode, both of which apply to Sleep mode and Deep-sleep mode.

### 2.7.1 Entering Sleep Modes

This section describes the mechanisms software can use to put the processor into one of the sleep modes.

The system can generate spurious wake-up events, for example a debug operation wakes up the processor. Therefore, software must be able to put the processor back into sleep mode after such an event. A program might have an idle loop to put the processor back to sleep mode.

#### 2.7.1.1 Wait for Interrupt

The wait for interrupt instruction, WFI, causes immediate entry to sleep mode unless the wake-up condition is true (see “Wake Up from WFI or Sleep-on-Exit” on page 77). When the processor executes a WFI instruction, it stops executing instructions and enters sleep mode. See the *Cortex™-M3 Instruction Set Technical User's Manual* for more information.

#### 2.7.1.2 Wait for Event

The wait for event instruction, WFE, causes entry to sleep mode conditional on the value of a one-bit event register. When the processor executes a WFE instruction, it checks the event register. If the register is 0, the processor stops executing instructions and enters sleep mode. If the register is 1, the processor clears the register and continues executing instructions without entering sleep mode.

If the event register is 1, the processor must not enter sleep mode on execution of a WFE instruction. Typically, this situation occurs if an SEV instruction has been executed. Software cannot access this register directly.

See the *Cortex™-M3 Instruction Set Technical User's Manual* for more information.

#### 2.7.1.3 Sleep-on-Exit

If the SLEEPEXIT bit of the **SYSCTRL** register is set, when the processor completes the execution of an exception handler, it returns to Thread mode and immediately enters sleep mode. This mechanism can be used in applications that only require the processor to run when an exception occurs.

### 2.7.2 Wake Up from Sleep Mode

The conditions for the processor to wake up depend on the mechanism that cause it to enter sleep mode.

#### 2.7.2.1 Wake Up from WFI or Sleep-on-Exit

Normally, the processor wakes up only when it detects an exception with sufficient priority to cause exception entry. Some embedded systems might have to execute system restore tasks after the processor wakes up and before executing an interrupt handler. Entry to the interrupt handler can be delayed by setting the PRIMASK bit and clearing the FAULTMASK bit. If an interrupt arrives that is enabled and has a higher priority than current exception priority, the processor wakes up but does not execute the interrupt handler until the processor clears PRIMASK. For more information about **PRIMASK** and **FAULTMASK**, see page 54 and page 55.

#### 2.7.2.2 Wake Up from WFE

The processor wakes up if it detects an exception with sufficient priority to cause exception entry.

In addition, if the SEVONPEND bit in the **SYSCTRL** register is set, any new pending interrupt triggers an event and wakes up the processor, even if the interrupt is disabled or has insufficient priority to cause exception entry. For more information about **SYSCTRL**, see page 110.

## 2.8 Instruction Set Summary

The processor implements a version of the Thumb instruction set. Table 2-13 on page 77 lists the supported instructions.

**Note:** In Table 2-13 on page 77:

- Angle brackets, <>, enclose alternative forms of the operand
- Braces, {}, enclose optional operands
- The Operands column is not exhaustive
- Op2 is a flexible second operand that can be either a register or a constant
- Most instructions can use an optional condition code suffix

For more information on the instructions and operands, see the instruction descriptions in the *Cortex™-M3 Instruction Set Technical User's Manual*.

**Table 2-13. Cortex-M3 Instruction Summary**

| Mnemonic | Operands | Brief Description | Flags |
|---|---|---|---|
| ADC, ADCS | {Rd,} Rn , Op2 | Add with carry | N,Z,C,V |
| ADD, ADDS | {Rd,} Rn , Op2 | Add | N,Z,C,V |
| ADD, ADDW | {Rd,} Rn , #imm12 | Add | N,Z,C,V |
| ADR | Rd , label | Load PC-relative address | - |
| AND, ANDS | {Rd ,} Rn , Op2 | Logical AND | N,Z,C |
| ASR, ASRS | Rd , Rm , <Rs\|#n> | Arithmetic shift right | N,Z,C |
| B | label | Branch | - |
| BFC | Rd , #lsb , #width | Bit field clear | - |
| BFI | Rd , Rn , #lsb , #width | Bit field insert | - |
| BIC, BICS | {Rd ,} Rn , Op2 | Bit clear | N,Z,C |
| BKPT | #imm | Breakpoint | - |
| BL | label | Branch with link | - |
| BLX | Rm | Branch indirect with link | - |
| BX | Rm | Branch indirect | - |
| CBNZ | Rn , label | Compare and branch if non-zero | - |

**Table 2-13. Cortex-M3 Instruction Summary *(continued)***

| Mnemonic | Operands | Brief Description | Flags |
|---|---|---|---|
| CBZ | Rn , label | Compare and branch if zero | - |
| CLREX | - | Clear exclusive | - |
| CLZ | Rd , Rm | Count leading zeros | - |
| CMN | Rn , Op2 | Compare negative | N,Z,C,V |
| CMP | Rn , Op2 | Compare | N,Z,C,V |
| CPSID | iflags | Change processor state, disable interrupts | - |
| CPSIE | iflags | Change processor state, enable interrupts | - |
| DMB | - | Data memory barrier | - |
| DSB | - | Data synchronization barrier | - |
| EOR, EORS | {Rd ,} Rn , Op2 | Exclusive OR | N,Z,C |
| ISB | - | Instruction synchronization barrier | - |
| IT | - | If-Then condition block | - |
| LDM | Rn{!} , reglist | Load multiple registers, increment after | - |
| LDMDB, LDMEA | Rn{!} , reglist | Load multiple registers, decrement before | - |
| LDMFD, LDMIA | Rn{!} , reglist | Load multiple registers, increment after | - |
| LDR | Rt , [ Rn {, #offset}] | Load register with word | - |
| LDRB, LDRBT | Rt , [ Rn {, #offset}] | Load register with byte | - |
| LDRD | Rt , Rt2 , [ Rn {, #offset}] | Load register with two words | - |
| LDREX | Rt , [ Rn , #offset ] | Load register exclusive | - |
| LDREXB | Rt, [Rn] | Load register exclusive with byte | - |
| LDREXH | Rt , [Rn] | Load register exclusive with halfword | - |
| LDRH, LDRHT | Rt , [ Rn{ , #offset}] | Load register with halfword | - |
| LDRSB, LDRSBT | Rt , [ Rn{ , #offset}] | Load register with signed byte | - |
| LDRSH, LDRSHT | Rt , [ Rn {, #offset}] | Load register with signed halfword | - |
| LDRT | Rt , [ Rn {, #offset}] | Load register with word | - |
| LSL, LSLS | Rd , Rm , <Rs\|#n> | Logical shift left | N,Z,C |
| LSR, LSRS | Rd , Rm , <Rs\|#n> | Logical shift right | N,Z,C |
| MLA | Rd , Rn , Rm, Ra | Multiply with accumulate, 32-bit result | - |
| MLS | Rd , Rn , Rm, Ra | Multiply and subtract, 32-bit result | - |
| MOV, MOVS | Rd , Op2 | Move | N,Z,C |
| MOV, MOVW | Rd , #imm16 | Move 16-bit constant | N,Z,C |
| MOVT | Rd , #imm16 | Move top | - |
| MRS | Rd , spec_reg | Move from special register to general register | - |
| MSR | spec_reg , Rn | Move from general register to special register | N,Z,C,V |
| MUL, MULS | {Rd,}Rn , Rm | Multiply, 32-bit result | N,Z |
| MVN, MVNS | Rd , Op2 | Move NOT | N,Z,C |
| NOP | - | No operation | - |
| ORN, ORNS | {Rd,} Rn , Op2 | Logical OR NOT | N,Z,C |

**Table 2-13. Cortex-M3 Instruction Summary *(continued)***

| Mnemonic | Operands | Brief Description | Flags |
|---|---|---|---|
| ORR, ORRS | {Rd,} Rn , Op2 | Logical OR | N,Z,C |
| POP | reglist | Pop registers from stack | - |
| PUSH | reglist | Push registers onto stack | - |
| RBIT | Rd , Rn | Reverse bits | - |
| REV | Rd , Rn | Reverse byte order in a word | - |
| REV16 | Rd , Rn | Reverse byte order in each halfword | - |
| REVSH | Rd , Rn | Reverse byte order in bottom halfword and sign extend | - |
| ROR, RORS | Rd , Rm , <Rs\|#n> | Rotate right | N,Z,C |
| RRX, RRXS | Rd , Rm | Rotate right with extend | N,Z,C |
| RSB, RSBS | {Rd,} Rn , Op2 | Reverse subtract | N,Z,C,V |
| SBC, SBCS | {Rd,} Rn , Op2 | Subtract with carry | N,Z,C,V |
| SBFX | Rd , Rn , #lsb , #width | Signed bit field extract | - |
| SDIV | {Rd ,} Rn , Rm | Signed divide | - |
| SEV | - | Send event | - |
| SMLAL | RdLo, RdHi, Rn, Rm | Signed multiply with accumulate (32x32+64), 64-bit result | - |
| SMULL | RdLo, RdHi, Rn, Rm | Signed multiply (32x32), 64-bit result | - |
| SSAT | Rd, #n, Rm {,shift #s} | Signed saturate | Q |
| STM | Rn{!} , reglist | Store multiple registers, increment after | - |
| STMDB, STMEA | Rn{!} , reglist | Store multiple registers, decrement before | - |
| STMFD, STMIA | Rn{!} , reglist | Store multiple registers, increment after | - |
| STR | Rt , [ Rn {, #offset}] | Store register word | - |
| STRB, STRBT | Rt , [ Rn {, #offset}] | Store register byte | - |
| STRD | Rt , Rt2 , [ Rn {, #offset}] | Store register two words | - |
| STREX | Rd , Rt , [ Rn , #offset ] | Store register exclusive | - |
| STREXB | Rd , Rt , [Rn] | Store register exclusive byte | - |
| STREXH | Rd , Rt , [Rn] | Store register exclusive halfword | - |
| STRH, STRHT | Rt , [ Rn {, #offset}] | Store register halfword | - |
| STRSB, STRSBT | Rt , [ Rn {, #offset}] | Store register signed byte | - |
| STRSH, STRSHT | Rt , [ Rn {, #offset}] | Store register signed halfword | - |
| STRT | Rt , [ Rn {, #offset}] | Store register word | - |
| SUB, SUBS | {Rd,} Rn , Op2 | Subtract | N,Z,C,V |
| SUB, SUBW | {Rd,} Rn , #imm12 | Subtract 12-bit constant | N,Z,C,V |
| SVC | #imm | Supervisor call | - |
| SXTB | {Rd,} Rm {,ROR #n} | Sign extend a byte | - |
| SXTH | {Rd,} Rm {,ROR #n} | Sign extend a halfword | - |
| TBB | [Rn, Rm] | Table branch byte | - |
| TBH | [Rn, Rm, LSL #1] | Table branch halfword | - |
| TEQ | Rn, Op2 | Test equivalence | N,Z,C |
| TST | Rn, Op2 | Test | N,Z,C |

**Table 2-13. Cortex-M3 Instruction Summary *(continued)***

| Mnemonic | Operands | Brief Description | Flags |
|---|---|---|---|
| UBFX | Rd , Rn , #lsb , #width | Unsigned bit field extract | - |
| UDIV | {Rd,} Rn , Rm | Unsigned divide | - |
| UMLAL | RdLo, RdHi, Rn, Rm | Unsigned multiply with accumulate (32x32+32+32), 64-bit result | - |
| UMULL | RdLo, RdHi, Rn, Rm | Unsigned multiply (32x 2), 64-bit result | - |
| USAT | Rd, #n, Rm {,shift #s} | Unsigned saturate | Q |
| UXTB | {Rd,} Rm {,ROR #n} | Zero extend a byte | - |
| UXTH | {Rd,} Rm {,ROR #n} | Zero extend a halfword | - |
| WFE | - | Wait for event | - |
| WFI | - | Wait for interrupt | - |

# 3 Cortex-M3 Peripherals

This chapter provides information on the Stellaris® implementation of the Cortex-M3 processor peripherals, including:

- SysTick (see page 81)

  Provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism.

- Nested Vectored Interrupt Controller (NVIC) (see page 82)
  - Facilitates low-latency exception and interrupt handling
  - Controls power management
  - Implements system control registers

- System Control Block (SCB) (see page 84)

  Provides system implementation information and system control, including configuration, control, and reporting of system exceptions.

- Memory Protection Unit (MPU) (see page 84)

  Supports the standard ARMv7 Protected Memory System Architecture (PMSA) model. The MPU provides full support for protection regions, overlapping protection regions, access permissions, and exporting memory attributes to the system.

Table 3-1 on page 81 shows the address map of the Private Peripheral Bus (PPB). Some peripheral register regions are split into two address regions, as indicated by two addresses listed.

**Table 3-1. Core Peripheral Register Regions**

| Address | Core Peripheral | Description (see page ...) |
|---|---|---|
| 0xE000.E010-0xE000.E01F | System Timer | 81 |
| 0xE000.E100-0xE000.E4EF<br>0xE000.EF00-0xE000.EF03 | Nested Vectored Interrupt Controller | 82 |
| 0xE000.ED00-0xE000.ED3F | System Control Block | 84 |
| 0xE000.ED90-0xE000.EDB8 | Memory Protection Unit | 84 |

## 3.1 Functional Description

This chapter provides information on the Stellaris implementation of the Cortex-M3 processor peripherals: SysTick, NVIC, SCB and MPU.

### 3.1.1 System Timer (SysTick)

Cortex-M3 includes an integrated system timer, SysTick, which provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism. The counter can be used in several different ways, for example as:

- An RTOS tick timer that fires at a programmable rate (for example, 100 Hz) and invokes a SysTick routine.

- A high-speed alarm timer using the system clock.

- A variable rate alarm or signal timer—the duration is range-dependent on the reference clock used and the dynamic range of the counter.
- A simple counter used to measure time to completion and time used.
- An internal clock source control based on missing/meeting durations. The COUNT bit in the **STCTRL** control and status register can be used to determine if an action completed within a set duration, as part of a dynamic clock management control loop.

The timer consists of three registers:

- **SysTick Control and Status (STCTRL)**: A control and status counter to configure its clock, enable the counter, enable the SysTick interrupt, and determine counter status.
- **SysTick Reload Value (STRELOAD)**: The reload value for the counter, used to provide the counter's wrap value.
- **SysTick Current Value (STCURRENT)**: The current value of the counter.

When enabled, the timer counts down on each clock from the reload value to zero, reloads (wraps) to the value in the **STRELOAD** register on the next clock edge, then decrements on subsequent clocks. Clearing the **STRELOAD** register disables the counter on the next wrap. When the counter reaches zero, the COUNT status bit is set. The COUNT bit clears on reads.

Writing to the **STCURRENT** register clears the register and the COUNT status bit. The write does not trigger the SysTick exception logic. On a read, the current value is the value of the register at the time the register is accessed.

The SysTick counter runs on the processor clock. If this clock signal is stopped for low power mode, the SysTick counter stops. Ensure software uses aligned word accesses to access the SysTick registers.

**Note:** When the processor is halted for debugging, the counter does not decrement.

### 3.1.2 Nested Vectored Interrupt Controller (NVIC)

This section describes the Nested Vectored Interrupt Controller (NVIC) and the registers it uses. The NVIC supports:

- 22 interrupts.
- A programmable priority level of 0-7 for each interrupt. A higher level corresponds to a lower priority, so level 0 is the highest interrupt priority.
- Low-latency exception and interrupt handling.
- Level and pulse detection of interrupt signals.
- Dynamic reprioritization of interrupts.
- Grouping of priority values into group priority and subpriority fields.
- Interrupt tail-chaining.
- An external Non-maskable interrupt (NMI).

The processor automatically stacks its state on exception entry and unstacks this state on exception exit, with no instruction overhead, providing low latency exception handling.

#### 3.1.2.1 Level-Sensitive and Pulse Interrupts

The processor supports both level-sensitive and pulse interrupts. Pulse interrupts are also described as edge-triggered interrupts.

A level-sensitive interrupt is held asserted until the peripheral deasserts the interrupt signal. Typically this happens because the ISR accesses the peripheral, causing it to clear the interrupt request. A pulse interrupt is an interrupt signal sampled synchronously on the rising edge of the processor clock. To ensure the NVIC detects the interrupt, the peripheral must assert the interrupt signal for at least one clock cycle, during which the NVIC detects the pulse and latches the interrupt.

When the processor enters the ISR, it automatically removes the pending state from the interrupt (see “Hardware and Software Control of Interrupts” on page 83 for more information). For a level-sensitive interrupt, if the signal is not deasserted before the processor returns from the ISR, the interrupt becomes pending again, and the processor must execute its ISR again. As a result, the peripheral can hold the interrupt signal asserted until it no longer needs servicing.

#### 3.1.2.2 Hardware and Software Control of Interrupts

The Cortex-M3 latches all interrupts. A peripheral interrupt becomes pending for one of the following reasons:

- The NVIC detects that the interrupt signal is High and the interrupt is not active.
- The NVIC detects a rising edge on the interrupt signal.
- Software writes to the corresponding interrupt set-pending register bit, or to the **Software Trigger Interrupt (SWTRIG)** register to make a Software-Generated Interrupt pending. See the `INT` bit in the **PEND0** register on page 97 or **SWTRIG** on page 102.

A pending interrupt remains pending until one of the following:

- The processor enters the ISR for the interrupt, changing the state of the interrupt from pending to active. Then:
  - For a level-sensitive interrupt, when the processor returns from the ISR, the NVIC samples the interrupt signal. If the signal is asserted, the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR. Otherwise, the state of the interrupt changes to inactive.
  - For a pulse interrupt, the NVIC continues to monitor the interrupt signal, and if this is pulsed the state of the interrupt changes to pending and active. In this case, when the processor returns from the ISR the state of the interrupt changes to pending, which might cause the processor to immediately re-enter the ISR.

    If the interrupt signal is not pulsed while the processor is in the ISR, when the processor returns from the ISR the state of the interrupt changes to inactive.
- Software writes to the corresponding interrupt clear-pending register bit
  - For a level-sensitive interrupt, if the interrupt signal is still asserted, the state of the interrupt does not change. Otherwise, the state of the interrupt changes to inactive.

– For a pulse interrupt, the state of the interrupt changes to inactive, if the state was pending or to active, if the state was active and pending.

### 3.1.3 System Control Block (SCB)

The System Control Block (SCB) provides system implementation information and system control, including configuration, control, and reporting of the system exceptions.

### 3.1.4 Memory Protection Unit (MPU)

This section describes the Memory protection unit (MPU). The MPU divides the memory map into a number of regions and defines the location, size, access permissions, and memory attributes of each region. The MPU supports independent attribute settings for each region, overlapping regions, and export of memory attributes to the system.

The memory attributes affect the behavior of memory accesses to the region. The Cortex-M3 MPU defines eight separate memory regions, 0-7, and a background region.

When memory regions overlap, a memory access is affected by the attributes of the region with the highest number. For example, the attributes for region 7 take precedence over the attributes of any region that overlaps region 7.

The background region has the same memory access attributes as the default memory map, but is accessible from privileged software only.

The Cortex-M3 MPU memory map is unified, meaning that instruction accesses and data accesses have the same region settings.

If a program accesses a memory location that is prohibited by the MPU, the processor generates a memory management fault, causing a fault exception and possibly causing termination of the process in an OS environment. In an OS environment, the kernel can update the MPU region setting dynamically based on the process to be executed. Typically, an embedded OS uses the MPU for memory protection.

Configuration of MPU regions is based on memory types (see “Memory Regions, Types and Attributes” on page 59 for more information).

Table 3-2 on page 84 shows the possible MPU region attributes. See the section called “MPU Configuration for a Stellaris Microcontroller” on page 88 for guidelines for programming a microcontroller implementation.

**Table 3-2. Memory Attributes Summary**

| Memory Type | Description |
|---|---|
| Strongly Ordered | All accesses to Strongly Ordered memory occur in program order. |
| Device | Memory-mapped peripherals |
| Normal | Normal memory |

To avoid unexpected behavior, disable the interrupts before updating the attributes of a region that the interrupt handlers might access.

Ensure software uses aligned accesses of the correct size to access MPU registers:

- Except for the **MPU Region Attribute and Size (MPUATTR)** register, all MPU registers must be accessed with aligned word accesses.
- The **MPUATTR** register can be accessed with byte or aligned halfword or word accesses.

The processor does not support unaligned accesses to MPU registers.

When setting up the MPU, and if the MPU has previously been programmed, disable unused regions to prevent any previous region settings from affecting the new MPU setup.

#### 3.1.4.1 Updating an MPU Region

To update the attributes for an MPU region, the **MPU Region Number (MPUNUMBER)**, **MPU Region Base Address (MPUBASE)** and **MPUATTR** registers must be updated. Each register can be programmed separately or with a multiple-word write to program all of these registers. You can use the **MPUBASEx** and **MPUATTRx** aliases to program up to four regions simultaneously using an STM instruction.

***Updating an MPU Region Using Separate Words***

This example simple code configures one region:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPUNUMBER        ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]       ; Region Number
STR R4, [R0, #0x4]       ; Region Base Address
STRH R2, [R0, #0x8]      ; Region Size and Enable
STRH R3, [R0, #0xA]      ; Region Attribute
```

Disable a region before writing new region settings to the MPU if you have previously enabled the region being changed. For example:

```
; R1 = region number
; R2 = size/enable
; R3 = attributes
; R4 = address
LDR R0,=MPUNUMBER         ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]        ; Region Number
BIC R2, R2, #1            ; Disable
STRH R2, [R0, #0x8]       ; Region Size and Enable
STR R4, [R0, #0x4]        ; Region Base Address
STRH R3, [R0, #0xA]       ; Region Attribute
ORR R2, #1                ; Enable
STRH R2, [R0, #0x8]       ; Region Size and Enable
```

Software must use memory barrier instructions:

- Before MPU setup, if there might be outstanding memory transfers, such as buffered writes, that might be affected by the change in MPU settings.
- After MPU setup, if it includes memory transfers that must use the new MPU settings.

However, memory barrier instructions are not required if the MPU setup process starts by entering an exception handler, or is followed by an exception return, because the exception entry and exception return mechanism cause memory barrier behavior.

Software does not need any memory barrier instructions during MPU setup, because it accesses the MPU through the Private Peripheral Bus (PPB), which is a Strongly Ordered memory region.

For example, if all of the memory access behavior is intended to take effect immediately after the programming sequence, then a DSB instruction and an ISB instruction should be used. A DSB is required after changing MPU settings, such as at the end of context switch. An ISB is required if the code that programs the MPU region or regions is entered using a branch or call. If the programming sequence is entered using a return from exception, or by taking an exception, then an ISB is not required.

### *Updating an MPU Region Using Multi-Word Writes*

The MPU can be programmed directly using multi-word writes, depending how the information is divided. Consider the following reprogramming:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPUNUMBER  ; 0xE000ED98, MPU region number register
STR R1, [R0, #0x0]  ; Region Number
STR R2, [R0, #0x4]  ; Region Base Address
STR R3, [R0, #0x8]  ; Region Attribute, Size and Enable
```

An STM instruction can be used to optimize this:

```
; R1 = region number
; R2 = address
; R3 = size, attributes in one
LDR R0, =MPUNUMBER  ; 0xE000ED98, MPU region number register
STM R0, {R1-R3}     ; Region number, address, attribute, size and enable
```

This operation can be done in two words for pre-packed information, meaning that the **MPU Region Base Address (MPUBASE)** register (see page 134) contains the required region number and has the VALID bit set. This method can be used when the data is statically packed, for example in a boot loader:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0, =MPUBASE    ; 0xE000ED9C, MPU Region Base register
STR R1, [R0, #0x0]  ; Region base address and region number combined
                    ; with VALID (bit 4) set
STR R2, [R0, #0x4]  ; Region Attribute, Size and Enable
```

An STM instruction can be used to optimize this:

```
; R1 = address and region number in one
; R2 = size and attributes in one
LDR R0,=MPUBASE     ; 0xE000ED9C, MPU Region Base register
STM R0, {R1-R2}     ; Region base address, region number and VALID bit,
                    ; and Region Attribute, Size and Enable
```

### *Subregions*

Regions of 256 bytes or more are divided into eight equal-sized subregions. Set the corresponding bit in the SRD field of the **MPU Region Attribute and Size (MPUATTR)** register (see page 136) to disable a subregion. The least-significant bit of the SRD field controls the first subregion, and the most-significant bit controls the last subregion. Disabling a subregion means another region

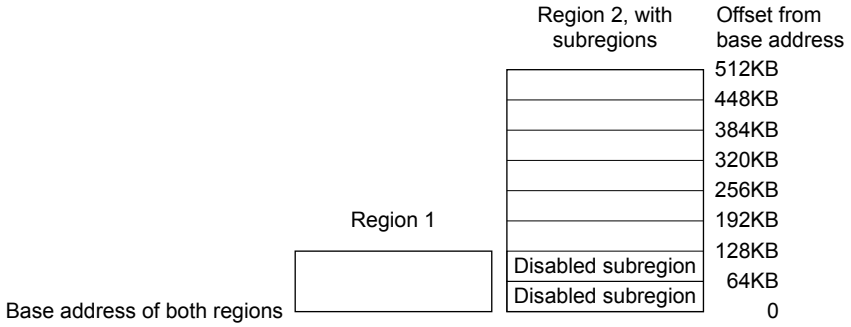
overlapping the disabled range matches instead. If no other enabled region overlaps the disabled subregion, the MPU issues a fault.

Regions of 32, 64, and 128 bytes do not support subregions. With regions of these sizes, the SRD field must be configured to 0x00, otherwise the MPU behavior is unpredictable.

***Example of SRD Use***

Two regions with the same base address overlap. Region one is 128 KB, and region two is 512 KB. To ensure the attributes from region one apply to the first 128 KB region, configure the SRD field for region two to 0x03 to disable the first two subregions, as Figure 3-1 on page 87 shows.

**Figure 3-1. SRD Use Example**




### 3.1.4.2 MPU Access Permission Attributes

The access permission bits, TEX, S, C, B, AP, and XN of the **MPUATTR** register, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

Table 3-3 on page 87 shows the encodings for the TEX, C, B, and S access permission bits. All encodings are shown for completeness, however the current implementation of the Cortex-M3 does not support the concept of cacheability or shareability. Refer to the section called “MPU Configuration for a Stellaris Microcontroller” on page 88 for information on programming the MPU for Stellaris implementations.

**Table 3-3. TEX, S, C, and B Bit Field Encoding**

| TEX | S | C | B | Memory Type | Shareability | Other Attributes |
|---|---|---|---|---|---|---|
| 000b | x[a] | 0 | 0 | Strongly Ordered | Shareable | - |
| 000 | x[a] | 0 | 1 | Device | Shareable | - |
| 000 | 0 | 1 | 0 | Normal | Not shareable | Outer and inner write-through. No write allocate. |
| 000 | 1 | 1 | 0 | Normal | Shareable | |
| 000 | 0 | 1 | 1 | Normal | Not shareable | |
| 000 | 1 | 1 | 1 | Normal | Shareable | |
| 001 | 0 | 0 | 0 | Normal | Not shareable | Outer and inner noncacheable. |
| 001 | 1 | 0 | 0 | Normal | Shareable | |
| 001 | x[a] | 0 | 1 | Reserved encoding | - | - |
| 001 | x[a] | 1 | 0 | Reserved encoding | - | - |
| 001 | 0 | 1 | 1 | Normal | Not shareable | Outer and inner write-back. Write and read allocate. |
| 001 | 1 | 1 | 1 | Normal | Shareable | |

**Table 3-3. TEX, S, C, and B Bit Field Encoding *(continued)***

| TEX | S | C | B | Memory Type | Shareability | Other Attributes |
|---|---|---|---|---|---|---|
| 010 | x[a] | 0 | 0 | Device | Not shareable | Nonshared Device. |
| 010 | x[a] | 0 | 1 | Reserved encoding | - | - |
| 010 | x[a] | 1 | x[a] | Reserved encoding | - | - |
| 1BB | 0 | A | A | Normal | Not shareable | Cached memory (BB = outer policy, AA = inner policy). See Table 3-4 for the encoding of the AA and BB bits. |
| 1BB | 1 | A | A | Normal | Shareable | |

a. The MPU ignores the value of this bit.

Table 3-4 on page 88 shows the cache policy for memory attribute encodings with a TEX value in the range of 0x4-0x7.

**Table 3-4. Cache Policy for Memory Attribute Encoding**

| Encoding, AA or BB | Corresponding Cache Policy |
|---|---|
| 00 | Non-cacheable |
| 01 | Write back, write and read allocate |
| 10 | Write through, no write allocate |
| 11 | Write back, no write allocate |

Table 3-5 on page 88 shows the AP encodings in the **MPUATTR** register that define the access permissions for privileged and unprivileged software.

**Table 3-5. AP Bit Field Encoding**

| AP Bit Field | Privileged Permissions | Unprivileged Permissions | Description |
|---|---|---|---|
| 000 | No access | No access | All accesses generate a permission fault. |
| 001 | R/W | No access | Access from privileged software only. |
| 010 | R/W | RO | Writes by unprivileged software generate a permission fault. |
| 011 | R/W | R/W | Full access. |
| 100 | Unpredictable | Unpredictable | Reserved. |
| 101 | RO | No access | Reads by privileged software only. |
| 110 | RO | RO | Read-only, by privileged or unprivileged software. |
| 111 | RO | RO | Read-only, by privileged or unprivileged software. |

### *MPU Configuration for a Stellaris Microcontroller*

Stellaris microcontrollers have only a single processor and no caches. As a result, the MPU should be programmed as shown in Table 3-6 on page 88.

**Table 3-6. Memory Region Attributes for Stellaris Microcontrollers**

| Memory Region | TEX | S | C | B | Memory Type and Attributes |
|---|---|---|---|---|---|
| Flash memory | 000b | 0 | 1 | 0 | Normal memory, non-shareable, write-through |
| Internal SRAM | 000b | 1 | 1 | 0 | Normal memory, shareable, write-through |

**Table 3-6. Memory Region Attributes for Stellaris Microcontrollers *(continued)***

| Memory Region | TEX | S | C | B | Memory Type and Attributes |
|---|---|---|---|---|---|
| External SRAM | 000b | 1 | 1 | 1 | Normal memory, shareable, write-back, write-allocate |
| Peripherals | 000b | 1 | 0 | 1 | Device memory, shareable |

In current Stellaris microcontroller implementations, the shareability and cache policy attributes do not affect the system behavior. However, using these settings for the MPU regions can make the application code more portable. The values given are for typical situations.

#### 3.1.4.3 MPU Mismatch

When an access violates the MPU permissions, the processor generates a memory management fault (see “Exceptions and Interrupts” on page 58 for more information). The **MFAULTSTAT** register indicates the cause of the fault. See page 121 for more information.

## 3.2 Register Map

Table 3-7 on page 89 lists the Cortex-M3 Peripheral SysTick, NVIC, SCB, and MPU registers. The offset listed is a hexadecimal increment to the register's address, relative to the Core Peripherals base address of 0xE000.E000.

**Note:** Register spaces that are not used are reserved for future or internal use. Software should not modify any reserved memory address.

**Table 3-7. Peripherals Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| **System Timer (SysTick) Registers** | | | | | |
| 0x010 | STCTRL | R/W | 0x0000.0000 | SysTick Control and Status Register | 91 |
| 0x014 | STRELOAD | R/W | 0x0000.0000 | SysTick Reload Value Register | 93 |
| 0x018 | STCURRENT | R/WC | 0x0000.0000 | SysTick Current Value Register | 94 |
| **Nested Vectored Interrupt Controller (NVIC) Registers** | | | | | |
| 0x100 | EN0 | R/W | 0x0000.0000 | Interrupt 0-29 Set Enable | 95 |
| 0x180 | DIS0 | R/W | 0x0000.0000 | Interrupt 0-29 Clear Enable | 96 |
| 0x200 | PEND0 | R/W | 0x0000.0000 | Interrupt 0-29 Set Pending | 97 |
| 0x280 | UNPEND0 | R/W | 0x0000.0000 | Interrupt 0-29 Clear Pending | 98 |
| 0x300 | ACTIVE0 | RO | 0x0000.0000 | Interrupt 0-29 Active Bit | 99 |
| 0x400 | PRI0 | R/W | 0x0000.0000 | Interrupt 0-3 Priority | 100 |
| 0x404 | PRI1 | R/W | 0x0000.0000 | Interrupt 4-7 Priority | 100 |
| 0x408 | PRI2 | R/W | 0x0000.0000 | Interrupt 8-11 Priority | 100 |
| 0x40C | PRI3 | R/W | 0x0000.0000 | Interrupt 12-15 Priority | 100 |
| 0x410 | PRI4 | R/W | 0x0000.0000 | Interrupt 16-19 Priority | 100 |
| 0x414 | PRI5 | R/W | 0x0000.0000 | Interrupt 20-23 Priority | 100 |

**Table 3-7. Peripherals Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x418 | PRI6 | R/W | 0x0000.0000 | Interrupt 24-27 Priority | 100 |
| 0x41C | PRI7 | R/W | 0x0000.0000 | Interrupt 28-29 Priority | 100 |
| 0xF00 | SWTRIG | WO | 0x0000.0000 | Software Trigger Interrupt | 102 |
| **System Control Block (SCB) Registers** | | | | | |
| 0xD00 | CPUID | RO | 0x410F.C231 | CPU ID Base | 103 |
| 0xD04 | INTCTRL | R/W | 0x0000.0000 | Interrupt Control and State | 104 |
| 0xD08 | VTABLE | R/W | 0x0000.0000 | Vector Table Offset | 107 |
| 0xD0C | APINT | R/W | 0xFA05.0000 | Application Interrupt and Reset Control | 108 |
| 0xD10 | SYSCTRL | R/W | 0x0000.0000 | System Control | 110 |
| 0xD14 | CFGCTRL | R/W | 0x0000.0000 | Configuration and Control | 112 |
| 0xD18 | SYSPRI1 | R/W | 0x0000.0000 | System Handler Priority 1 | 114 |
| 0xD1C | SYSPRI2 | R/W | 0x0000.0000 | System Handler Priority 2 | 115 |
| 0xD20 | SYSPRI3 | R/W | 0x0000.0000 | System Handler Priority 3 | 116 |
| 0xD24 | SYSHNDCTRL | R/W | 0x0000.0000 | System Handler Control and State | 117 |
| 0xD28 | FAULTSTAT | R/W1C | 0x0000.0000 | Configurable Fault Status | 121 |
| 0xD2C | HFAULTSTAT | R/W1C | 0x0000.0000 | Hard Fault Status | 127 |
| 0xD34 | MMADDR | R/W | - | Memory Management Fault Address | 128 |
| 0xD38 | FAULTADDR | R/W | - | Bus Fault Address | 129 |
| **Memory Protection Unit (MPU) Registers** | | | | | |
| 0xD90 | MPUTYPE | RO | 0x0000.0800 | MPU Type | 130 |
| 0xD94 | MPUCTRL | R/W | 0x0000.0000 | MPU Control | 131 |
| 0xD98 | MPUNUMBER | R/W | 0x0000.0000 | MPU Region Number | 133 |
| 0xD9C | MPUBASE | R/W | 0x0000.0000 | MPU Region Base Address | 134 |
| 0xDA0 | MPUATTR | R/W | 0x0000.0000 | MPU Region Attribute and Size | 136 |
| 0xDA4 | MPUBASE1 | R/W | 0x0000.0000 | MPU Region Base Address Alias 1 | 134 |
| 0xDA8 | MPUATTR1 | R/W | 0x0000.0000 | MPU Region Attribute and Size Alias 1 | 136 |
| 0xDAC | MPUBASE2 | R/W | 0x0000.0000 | MPU Region Base Address Alias 2 | 134 |
| 0xDB0 | MPUATTR2 | R/W | 0x0000.0000 | MPU Region Attribute and Size Alias 2 | 136 |
| 0xDB4 | MPUBASE3 | R/W | 0x0000.0000 | MPU Region Base Address Alias 3 | 134 |
| 0xDB8 | MPUATTR3 | R/W | 0x0000.0000 | MPU Region Attribute and Size Alias 3 | 136 |

## 3.3 System Timer (SysTick) Register Descriptions

This section lists and describes the System Timer registers, in numerical order by address offset.

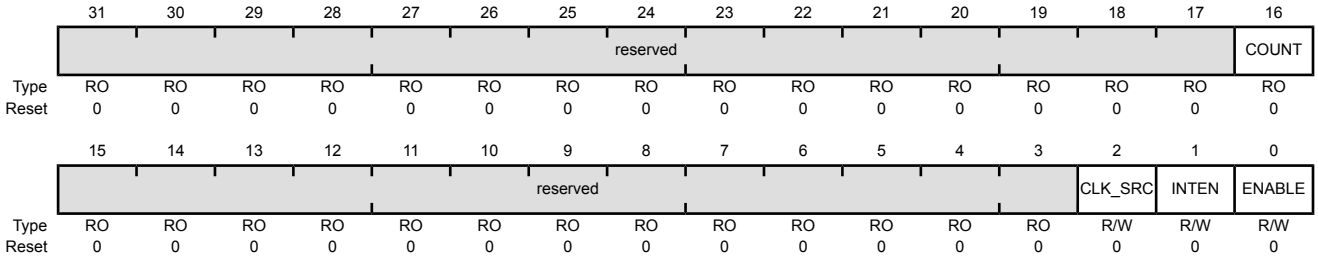
### Register 1: SysTick Control and Status Register (STCTRL), offset 0x010

**Note:** This register can only be accessed from privileged mode.

The SysTick **STCTRL** register enables the SysTick features.

SysTick Control and Status Register (STCTRL)

Base 0xE000.E000
Offset 0x010
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | COUNT | RO | 0 | Count Flag<br><br>Value Description<br>0 The SysTick timer has not counted to 0 since the last time this bit was read.<br>1 The SysTick timer has counted to 0 since the last time this bit was read.<br><br>This bit is cleared by a read of the register or if the **STCURRENT** register is written with any value.<br>If read by the debugger using the DAP, this bit is cleared only if the `MasterType` bit in the **AHB-AP Control Register** is clear. Otherwise, the `COUNT` bit is not changed by the debugger read. See the *ARM® Debug Interface V5 Architecture Specification* for more information on `MasterType`. |
| 15:3 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | CLK_SRC | R/W | 0 | Clock Source<br><br>Value Description<br>0 External reference clock. (Not implemented for Stellaris microcontrollers.)<br>1 System clock<br><br>Because an external reference clock is not implemented, this bit must be set in order for SysTick to operate. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | INTEN | R/W | 0 | Interrupt Enable |

| Value | Description |
|---|---|
| 0 | Interrupt generation is disabled. Software can use the COUNT bit to determine if the counter has ever reached 0. |
| 1 | An interrupt is generated to the NVIC when SysTick counts to 0. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | ENABLE | R/W | 0 | Enable |

| Value | Description |
|---|---|
| 0 | The counter is disabled. |
| 1 | Enables SysTick to operate in a multi-shot way. That is, the counter loads the RELOAD value and begins counting down. On reaching 0, the COUNT bit is set and an interrupt is generated if enabled by INTEN. The counter then loads the RELOAD value again and begins counting. |

## Register 2: SysTick Reload Value Register (STRELOAD), offset 0x014

**Note:** This register can only be accessed from privileged mode.

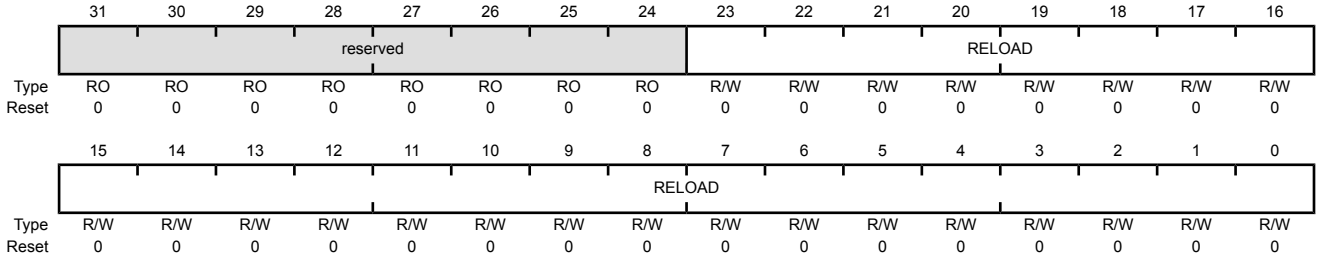
**Note:** This register can only be accessed from privileged mode.

The **STRELOAD** register specifies the start value to load into the **SysTick Current Value (STCURRENT)** register when the counter reaches 0. The start value can be between 0x1 and 0x00FF.FFFF. A start value of 0 is possible but has no effect because the SysTick interrupt and the COUNT bit are activated when counting from 1 to 0.

SysTick can be configured as a multi-shot timer, repeated over and over, firing every N+1 clock pulses, where N is any value from 1 to 0x00FF.FFFF. For example, if a tick interrupt is required every 100 clock pulses, 99 must be written into the RELOAD field.

SysTick Reload Value Register (STRELOAD)

Base 0xE000.E000
Offset 0x014
Type R/W, reset 0x0000.0000



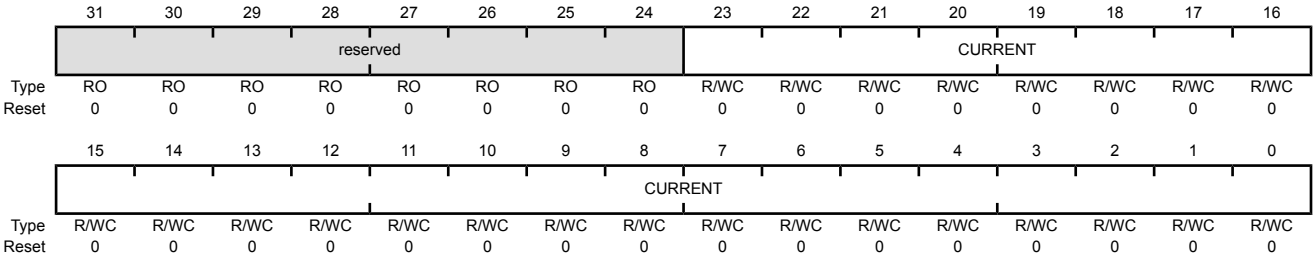

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:24 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:0 | RELOAD | R/W | 0x00.0000 | Reload Value<br>Value to load into the **SysTick Current Value (STCURRENT)** register when the counter reaches 0. |

### Register 3: SysTick Current Value Register (STCURRENT), offset 0x018

**Note:** This register can only be accessed from privileged mode.

The **STCURRENT** register contains the current value of the SysTick counter.

SysTick Current Value Register (STCURRENT)
Base 0xE000.E000
Offset 0x018
Type R/WC, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:24 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:0 | CURRENT | R/WC | 0x00.0000 | Current Value<br>This field contains the current value at the time the register is accessed. No read-modify-write protection is provided, so change with care.<br>This register is write-clear. Writing to it with any value clears the register. Clearing this register also clears the COUNT bit of the **STCTRL** register. |

## 3.4 NVIC Register Descriptions

This section lists and describes the NVIC registers, in numerical order by address offset.

The NVIC registers can only be fully accessed from privileged mode, but interrupts can be pended while in unprivileged mode by enabling the **Configuration and Control (CFGCTRL)** register. Any other unprivileged mode access causes a bus fault.

Ensure software uses correctly aligned register accesses. The processor does not support unaligned accesses to NVIC registers.

An interrupt can enter the pending state even if it is disabled.

Before programming the **VTABLE** register to relocate the vector table, ensure the vector table entries of the new vector table are set up for fault handlers, NMI, and all enabled exceptions such as interrupts. For more information, see page 107.

## Register 4: Interrupt 0-29 Set Enable (EN0), offset 0x100

**Note:** This register can only be accessed from privileged mode.

The **EN0** register enables interrupts and shows which interrupts are enabled. Bit 0 corresponds to Interrupt 0; bit 29 corresponds to Interrupt 29.

See Table 2-9 on page 69 for interrupt assignments.

If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

Interrupt 0-29 Set Enable (EN0)

Base 0xE000.E000
Offset 0x100
Type R/W, reset 0x0000.0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | INT | | | | | | | | | | | | | |
| Type | RO | RO | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| INT | | | | | | | | | | | | | | | |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 29:0 | INT | R/W | 0x000.0000 | Interrupt Enable |

| Value | Description |
|---|---|
| 0 | On a read, indicates the interrupt is disabled.<br>On a write, no effect. |
| 1 | On a read, indicates the interrupt is enabled.<br>On a write, enables the interrupt. |

A bit can only be cleared by setting the corresponding `INT[n]` bit in the **DISn** register.

## Register 5: Interrupt 0-29 Clear Enable (DIS0), offset 0x180

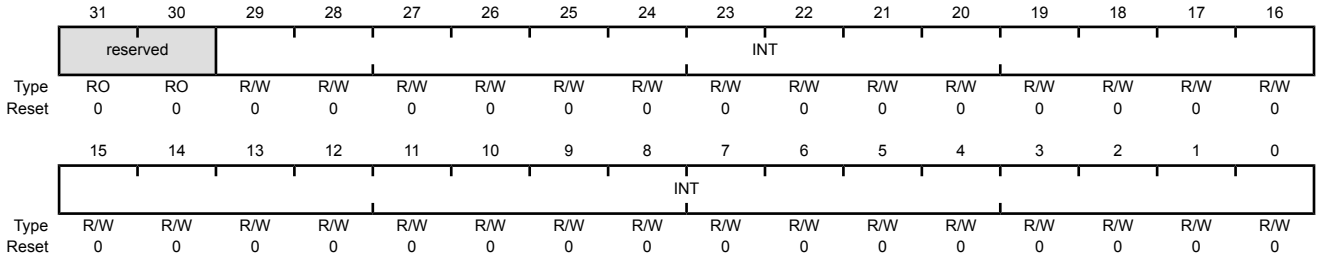
**Note:** This register can only be accessed from privileged mode.

The **DIS0** register disables interrupts. Bit 0 corresponds to Interrupt 0; bit 29 corresponds to Interrupt 29.

See Table 2-9 on page 69 for interrupt assignments.

Interrupt 0-29 Clear Enable (DIS0)

Base 0xE000.E000
Offset 0x180
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 29:0 | INT | R/W | 0x000.0000 | Interrupt Disable |

| Value | Description |
|---|---|
| 0 | On a read, indicates the interrupt is disabled.<br>On a write, no effect. |
| 1 | On a read, indicates the interrupt is enabled.<br>On a write, clears the corresponding `INT[n]` bit in the **EN0** register, disabling interrupt [n]. |

## Register 6: Interrupt 0-29 Set Pending (PEND0), offset 0x200

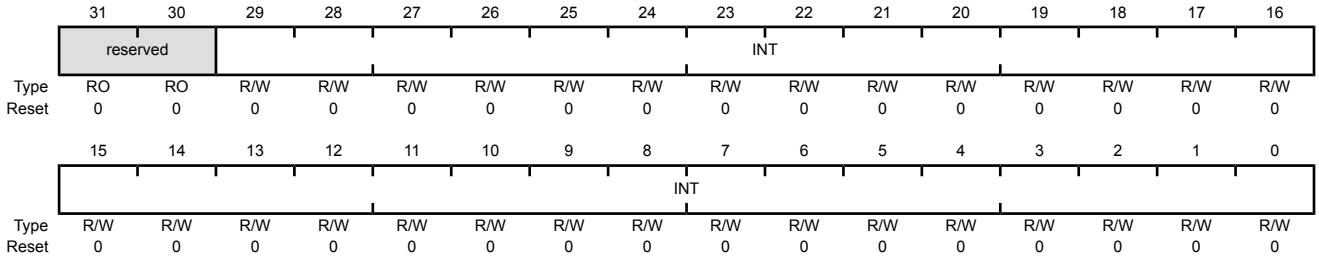
**Note:** This register can only be accessed from privileged mode.

The **PEND0** register forces interrupts into the pending state and shows which interrupts are pending. Bit 0 corresponds to Interrupt 0; bit 29 corresponds to Interrupt 29.

See Table 2-9 on page 69 for interrupt assignments.

Interrupt 0-29 Set Pending (PEND0)

Base 0xE000.E000
Offset 0x200
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 29:0 | INT | R/W | 0x000.0000 | Interrupt Set Pending |

| Value | Description |
|---|---|
| 0 | On a read, indicates that the interrupt is not pending.<br>On a write, no effect. |
| 1 | On a read, indicates that the interrupt is pending.<br>On a write, the corresponding interrupt is set to pending even if it is disabled. |

If the corresponding interrupt is already pending, setting a bit has no effect.
A bit can only be cleared by setting the corresponding `INT[n]` bit in the **UNPEND0** register.

## Register 7: Interrupt 0-29 Clear Pending (UNPEND0), offset 0x280

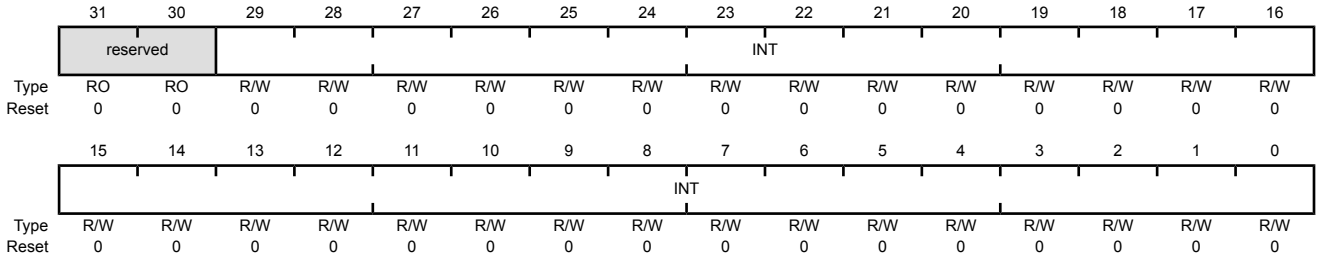
**Note:** This register can only be accessed from privileged mode.

The **UNPEND0** register shows which interrupts are pending and removes the pending state from interrupts. Bit 0 corresponds to Interrupt 0; bit 29 corresponds to Interrupt 29.

See Table 2-9 on page 69 for interrupt assignments.

Interrupt 0-29 Clear Pending (UNPEND0)

Base 0xE000.E000
Offset 0x280
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 29:0 | INT | R/W | 0x000.0000 | Interrupt Clear Pending |

| Value | Description |
|---|---|
| 0 | On a read, indicates that the interrupt is not pending.<br>On a write, no effect. |
| 1 | On a read, indicates that the interrupt is pending.<br>On a write, clears the corresponding `INT[n]` bit in the **PEND0** register, so that interrupt [n] is no longer pending.<br>Setting a bit does not affect the active state of the corresponding interrupt. |

## Register 8: Interrupt 0-29 Active Bit (ACTIVE0), offset 0x300

**Note:** This register can only be accessed from privileged mode.

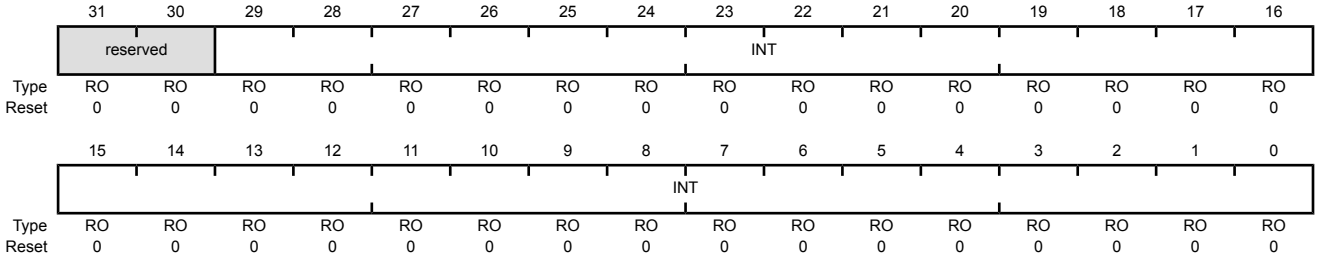
The **ACTIVE0** register indicates which interrupts are active. Bit 0 corresponds to Interrupt 0; bit 29 corresponds to Interrupt 29.

See Table 2-9 on page 69 for interrupt assignments.

**Caution – Do not manually set or clear the bits in this register.**

Interrupt 0-29 Active Bit (ACTIVE0)

Base 0xE000.E000
Offset 0x300
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 29:0 | INT | RO | 0x000.0000 | Interrupt Active |

| Value | Description |
|---|---|
| 0 | The corresponding interrupt is not active. |
| 1 | The corresponding interrupt is active, or active and pending. |

### Register 9: Interrupt 0-3 Priority (PRI0), offset 0x400

### Register 10: Interrupt 4-7 Priority (PRI1), offset 0x404

### Register 11: Interrupt 8-11 Priority (PRI2), offset 0x408

### Register 12: Interrupt 12-15 Priority (PRI3), offset 0x40C

### Register 13: Interrupt 16-19 Priority (PRI4), offset 0x410

### Register 14: Interrupt 20-23 Priority (PRI5), offset 0x414

### Register 15: Interrupt 24-27 Priority (PRI6), offset 0x418

### Register 16: Interrupt 28-29 Priority (PRI7), offset 0x41C

**Note:** This register can only be accessed from privileged mode.

The **PRIn** registers provide 3-bit priority fields for each interrupt. These registers are byte accessible. Each register holds four priority fields that are assigned to interrupts as follows:

| PRIn Register Bit Field | Interrupt |
|---|---|
| Bits 31:29 | Interrupt [4n+3] |
| Bits 23:21 | Interrupt [4n+2] |
| Bits 15:13 | Interrupt [4n+1] |
| Bits 7:5 | Interrupt [4n] |

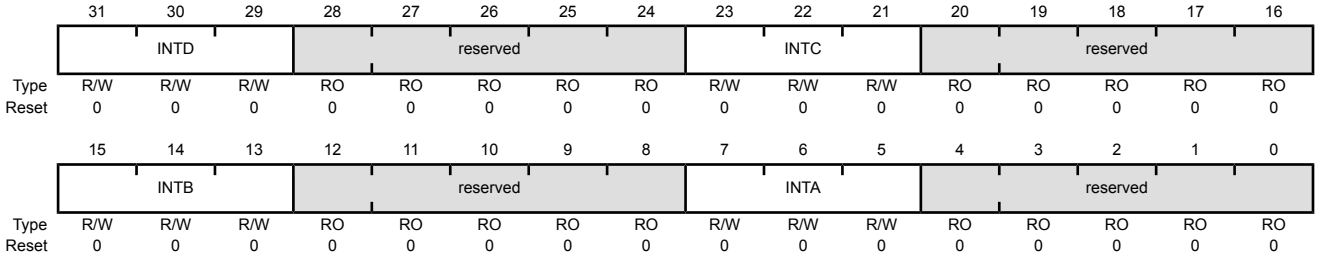
See Table 2-9 on page 69 for interrupt assignments.

Each priority level can be split into separate group priority and subpriority fields. The PRIGROUP field in the **Application Interrupt and Reset Control (APINT)** register (see page 108) indicates the position of the binary point that splits the priority and subpriority fields .

These registers can only be accessed from privileged mode.

Interrupt 0-3 Priority (PRI0)

Base 0xE000.E000
Offset 0x400
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:29 | INTD | R/W | 0x0 | Interrupt Priority for Interrupt [4n+3]<br>This field holds a priority value, 0-7, for the interrupt with the number [4n+3], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 28:24 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:21 | INTC | R/W | 0x0 | Interrupt Priority for Interrupt [4n+2]<br>This field holds a priority value, 0-7, for the interrupt with the number [4n+2], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt. |
| 20:16 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:13 | INTB | R/W | 0x0 | Interrupt Priority for Interrupt [4n+1]<br>This field holds a priority value, 0-7, for the interrupt with the number [4n+1], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt. |
| 12:8 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:5 | INTA | R/W | 0x0 | Interrupt Priority for Interrupt [4n]<br>This field holds a priority value, 0-7, for the interrupt with the number [4n], where n is the number of the **Interrupt Priority** register (n=0 for **PRI0**, and so on). The lower the value, the greater the priority of the corresponding interrupt. |
| 4:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

### Register 17: Software Trigger Interrupt (SWTRIG), offset 0xF00

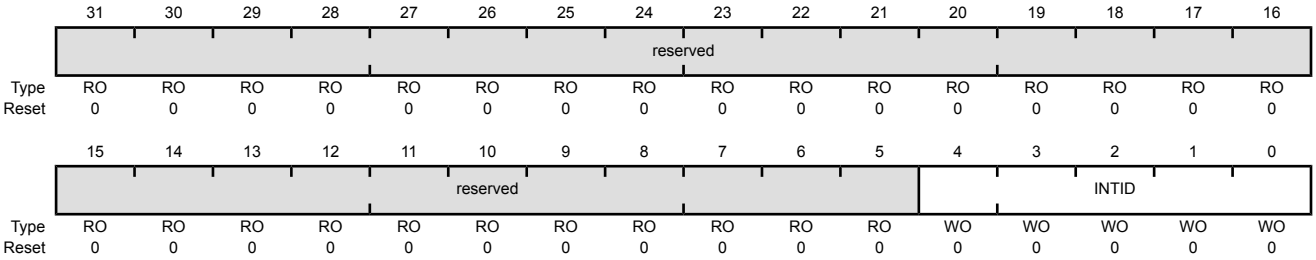
**Note:** Only privileged software can enable unprivileged access to the **SWTRIG** register.

Writing an interrupt number to the **SWTRIG** register generates a Software Generated Interrupt (SGI). See Table 2-9 on page 69 for interrupt assignments.

When the MAINPEND bit in the **Configuration and Control (CFGCTRL)** register (see page 112) is set, unprivileged software can access the **SWTRIG** register.

Software Trigger Interrupt (SWTRIG)

Base 0xE000.E000
Offset 0xF00
Type WO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4:0 | INTID | WO | 0x00 | Interrupt ID<br>This field holds the interrupt ID of the required SGI. For example, a value of 0x3 generates an interrupt on IRQ3. |

## 3.5 System Control Block (SCB) Register Descriptions

This section lists and describes the System Control Block (SCB) registers, in numerical order by address offset. The SCB registers can only be accessed from privileged mode.

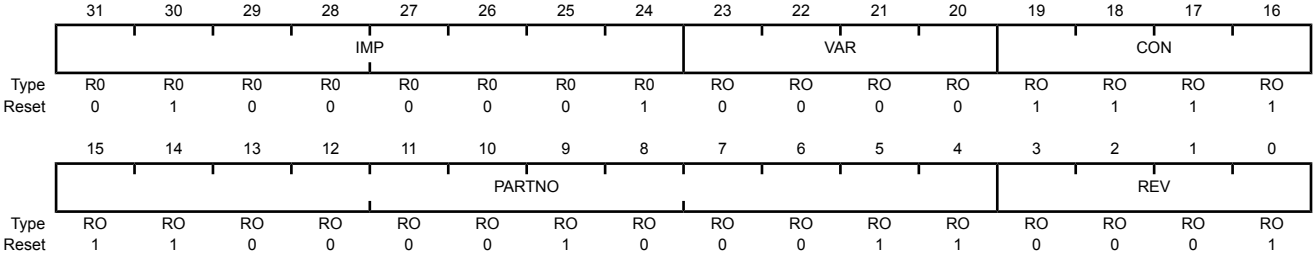
All registers must be accessed with aligned word accesses except for the **FAULTSTAT** and **SYSPRI1**-**SYSPRI3** registers, which can be accessed with byte or aligned halfword or word accesses. The processor does not support unaligned accesses to system control block registers.

## Register 18: CPU ID Base (CPUID), offset 0xD00

**Note:** This register can only be accessed from privileged mode.

The **CPUID** register contains the ARM® Cortex™-M3 processor part number, version, and implementation information.

CPU ID Base (CPUID)
Base 0xE000.E000
Offset 0xD00
Type RO, reset 0x410F.C231




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:24 | IMP | R0 | 0x41 | Implementer Code<br><br>Value Description<br>0x41 ARM |
| 23:20 | VAR | RO | 0x0 | Variant Number<br><br>Value Description<br>0x0 The rn value in the rnpn product revision identifier, for example, the 0 in r0p1. |
| 19:16 | CON | RO | 0xF | Constant<br><br>Value Description<br>0xF Always reads as 0xF. |
| 15:4 | PARTNO | RO | 0xC23 | Part Number<br><br>Value Description<br>0xC23 Cortex-M3 processor. |
| 3:0 | REV | RO | 0x1 | Revision Number<br><br>Value Description<br>0x1 The pn value in the rnpn product revision identifier, for example, the 1 in r0p1. |

## Register 19: Interrupt Control and State (INTCTRL), offset 0xD04

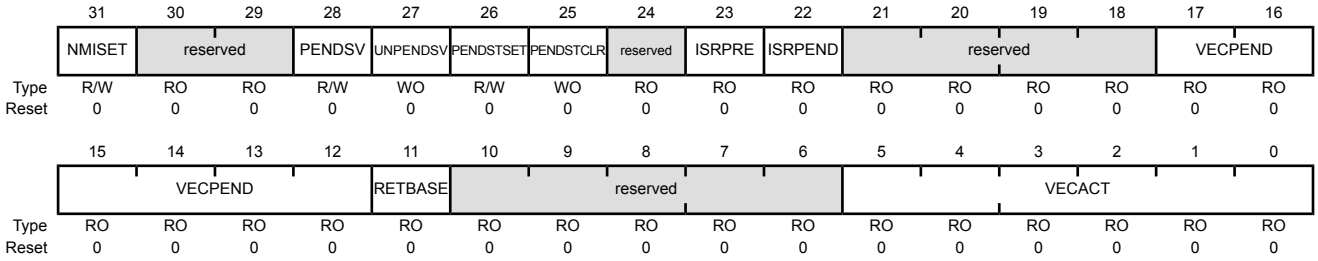
**Note:** This register can only be accessed from privileged mode.

The **INCTRL** register provides a set-pending bit for the NMI exception, and set-pending and clear-pending bits for the PendSV and SysTick exceptions. In addition, bits in this register indicate the exception number of the exception being processed, whether there are preempted active exceptions, the exception number of the highest priority pending exception, and whether any interrupts are pending.

When writing to **INCTRL**, the effect is unpredictable when writing a 1 to both the PENDSV and UNPENDSV bits, or writing a 1 to both the PENDSTSET and PENDSTCLR bits.

Interrupt Control and State (INTCTRL)

Base 0xE000.E000
Offset 0xD04
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | NMISET | R/W | 0 | NMI Set Pending<br><br>Value Description<br>0 On a read, indicates an NMI exception is not pending. On a write, no effect.<br>1 On a read, indicates an NMI exception is pending. On a write, changes the NMI exception state to pending.<br><br>Because NMI is the highest-priority exception, normally the processor enters the NMI exception handler as soon as it registers the setting of this bit, and clears this bit on entering the interrupt handler. A read of this bit by the NMI exception handler returns 1 only if the NMI signal is reasserted while the processor is executing that handler. |
| 30:29 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 28 | PENDSV | R/W | 0 | PendSV Set Pending<br><br>Value Description<br>0 On a read, indicates a PendSV exception is not pending. On a write, no effect.<br>1 On a read, indicates a PendSV exception is pending. On a write, changes the PendSV exception state to pending.<br><br>Setting this bit is the only way to set the PendSV exception state to pending. This bit is cleared by writing a 1 to the UNPENDSV bit. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 27 | UNPENDSV | WO | 0 | PendSV Clear Pending<br><br>Value Description<br>0 On a write, no effect.<br>1 On a write, removes the pending state from the PendSV exception.<br><br>This bit is write only; on a register read, its value is unknown. |
| 26 | PENDSTSET | R/W | 0 | SysTick Set Pending<br><br>Value Description<br>0 On a read, indicates a SysTick exception is not pending.<br>On a write, no effect.<br>1 On a read, indicates a SysTick exception is pending.<br>On a write, changes the SysTick exception state to pending.<br><br>This bit is cleared by writing a 1 to the PENDSTCLR bit. |
| 25 | PENDSTCLR | WO | 0 | SysTick Clear Pending<br><br>Value Description<br>0 On a write, no effect.<br>1 On a write, removes the pending state from the SysTick exception.<br><br>This bit is write only; on a register read, its value is unknown. |
| 24 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23 | ISRPRE | RO | 0 | Debug Interrupt Handling<br><br>Value Description<br>0 The release from halt does not take an interrupt.<br>1 The release from halt takes an interrupt.<br><br>This bit is only meaningful in Debug mode and reads as zero when the processor is not in Debug mode. |
| 22 | ISRPEND | RO | 0 | Interrupt Pending<br><br>Value Description<br>0 No interrupt is pending.<br>1 An interrupt is pending.<br><br>This bit provides status for all interrupts excluding NMI and Faults. |
| 21:18 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 17:12 | VECPEND | RO | 0x00 | Interrupt Pending Vector Number<br>This field contains the exception number of the highest priority pending enabled exception. The value indicated by this field includes the effect of the **BASEPRI** and **FAULTMASK** registers, but not any effect of the **PRIMASK** register.<br><br>Value / Description<br>0x00 No exceptions are pending<br>0x01 Reserved<br>0x02 NMI<br>0x03 Hard fault<br>0x04 Memory management fault<br>0x05 Bus fault<br>0x06 Usage fault<br>0x07-0x0A Reserved<br>0x0B SVCall<br>0x0C Reserved for Debug<br>0x0D Reserved<br>0x0E PendSV<br>0x0F SysTick<br>0x10 Interrupt Vector 0<br>0x11 Interrupt Vector 1<br>... ...<br>0x2D Interrupt Vector 29<br>0x2E-0x3F Reserved |
| 11 | RETBASE | RO | 0 | Return to Base<br><br>Value / Description<br>0 There are preempted active exceptions to execute.<br>1 There are no active exceptions, or the currently executing exception is the only active exception.<br><br>This bit provides status for all interrupts excluding NMI and Faults. This bit only has meaning if the processor is currently executing an ISR (the **Interrupt Program Status (IPSR)** register is non-zero). |
| 10:6 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:0 | VECACT | RO | 0x00 | Interrupt Pending Vector Number<br>This field contains the active exception number. The exception numbers can be found in the description for the `VECPEND` field. If this field is clear, the processor is in Thread mode. This field contains the same value as the `ISRNUM` field in the **IPSR** register.<br>Subtract 16 from this value to obtain the IRQ number required to index into the **Interrupt Set Enable (ENn)**, **Interrupt Clear Enable (DISn)**, **Interrupt Set Pending (PENDn)**, **Interrupt Clear Pending (UNPENDn)**, and **Interrupt Priority (PRIn)** registers (see page 50). |

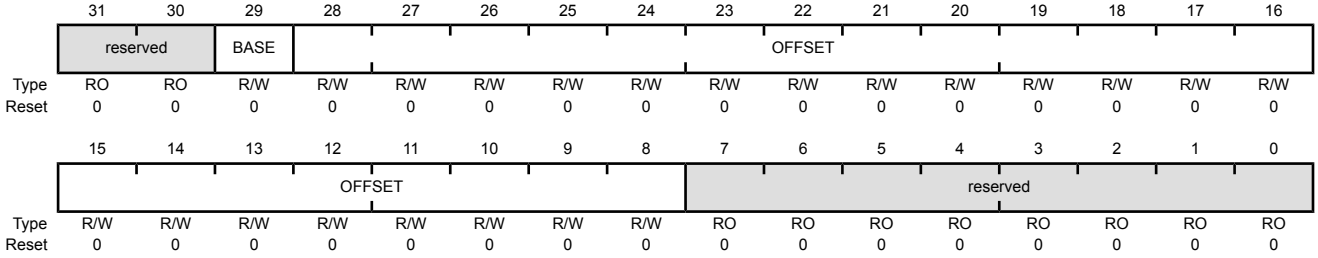
## Register 20: Vector Table Offset (VTABLE), offset 0xD08

**Note:** This register can only be accessed from privileged mode.

The **VTABLE** register indicates the offset of the vector table base address from memory address 0x0000.0000.

Vector Table Offset (VTABLE)

Base 0xE000.E000
Offset 0xD08
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 29 | BASE | R/W | 0 | Vector Table Base<br>Value Description<br>0 The vector table is in the code memory region.<br>1 The vector table is in the SRAM memory region. |
| 28:8 | OFFSET | R/W | 0x000.00 | Vector Table Offset<br>When configuring the OFFSET field, the offset must be aligned to the number of exception entries in the vector table. Because there are 29 interrupts, the minimum alignment is 64 words. |
| 7:0 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 21: Application Interrupt and Reset Control (APINT), offset 0xD0C

**Note:** This register can only be accessed from privileged mode.

The **APINT** register provides priority grouping control for the exception model, endian status for data accesses, and reset control of the system. To write to this register, 0x05FA must be written to the VECTKEY field, otherwise the write is ignored.

The PRIGROUP field indicates the position of the binary point that splits the INTx fields in the **Interrupt Priority (PRIx)** registers into separate group priority and subpriority fields. Table 3-8 on page 108 shows how the PRIGROUP value controls this split. The bit numbers in the Group Priority Field and Subpriority Field columns in the table refer to the bits in the INTA field. For the INTB field, the corresponding bits are 15:13; for INTC, 23:21; and for INTD, 31:29.

**Note:** Determining preemption of an exception uses only the group priority field.

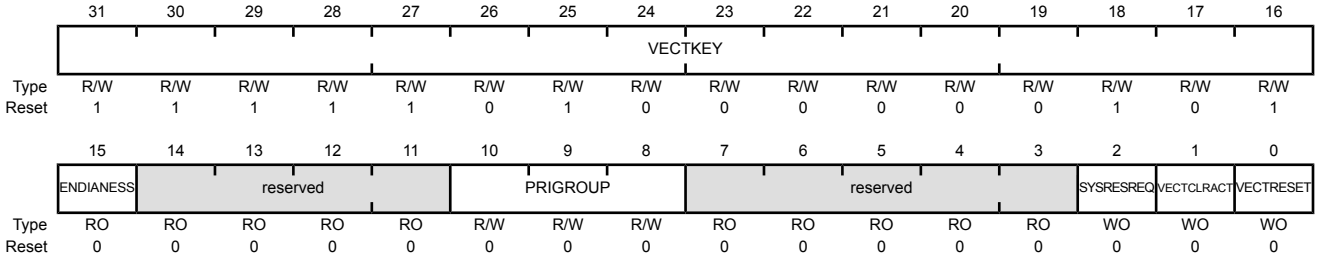
**Table 3-8. Interrupt Priority Levels**

| PRIGROUP Bit Field | Binary Point[a] | Group Priority Field | Subpriority Field | Group Priorities | Subpriorities |
|---|---|---|---|---|---|
| 0x0 - 0x4 | bxxx. | [7:5] | None | 8 | 1 |
| 0x5 | bxx.y | [7:6] | [5] | 4 | 2 |
| 0x6 | bx.yy | [7] | [6:5] | 2 | 4 |
| 0x7 | b.yyy | None | [7:5] | 1 | 8 |

a. INTx field showing the binary point. An x denotes a group priority field bit, and a y denotes a subpriority field bit.

Application Interrupt and Reset Control (APINT)

Base 0xE000.E000
Offset 0xD0C
Type R/W, reset 0xFA05.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | VECTKEY | R/W | 0xFA05 | Register Key<br>This field is used to guard against accidental writes to this register. 0x05FA must be written to this field in order to change the bits in this register. On a read, 0xFA05 is returned. |
| 15 | ENDIANESS | RO | 0 | Data Endianess<br>The Stellaris implementation uses only little-endian mode so this is cleared to 0. |
| 14:11 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 10:8 | PRIGROUP | R/W | 0x0 | Interrupt Priority Grouping<br>This field determines the split of group priority from subpriority (see Table 3-8 on page 108 for more information). |
| 7:3 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | SYSRESREQ | WO | 0 | System Reset Request<br>Value Description<br>0 No effect.<br>1 Resets the core and all on-chip peripherals except the Debug interface.<br>This bit is automatically cleared during the reset of the core and reads as 0. |
| 1 | VECTCLRACT | WO | 0 | Clear Active NMI / Fault<br>This bit is reserved for Debug use and reads as 0. This bit must be written as a 0, otherwise behavior is unpredictable. |
| 0 | VECTRESET | WO | 0 | System Reset<br>This bit is reserved for Debug use and reads as 0. This bit must be written as a 0, otherwise behavior is unpredictable. |

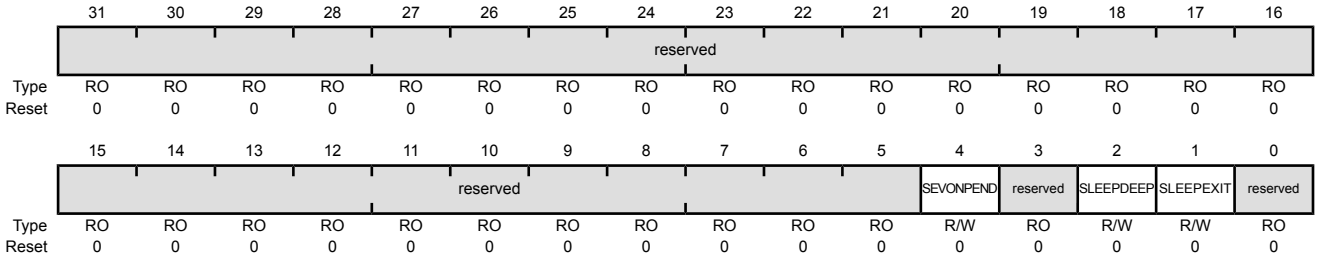
## Register 22: System Control (SYSCTRL), offset 0xD10

**Note:** This register can only be accessed from privileged mode.

The **SYSCTRL** register controls features of entry to and exit from low-power state.

System Control (SYSCTRL)

Base 0xE000.E000
Offset 0xD10
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SEVONPEND | R/W | 0 | Wake Up on Pending<br><br>Value Description<br>0 Only enabled interrupts or events can wake up the processor; disabled interrupts are excluded.<br>1 Enabled events and all interrupts, including disabled interrupts, can wake up the processor.<br><br>When an event or interrupt enters the pending state, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE.<br>The processor also wakes up on execution of a SEV instruction or an external event. |
| 3 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | SLEEPDEEP | R/W | 0 | Deep Sleep Enable<br><br>Value Description<br>0 Use Sleep mode as the low power mode.<br>1 Use Deep-sleep mode as the low power mode. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | SLEEPEXIT | R/W | 0 | Sleep on ISR Exit<br><br>Value Description<br>0 When returning from Handler mode to Thread mode, do not sleep when returning to Thread mode.<br>1 When returning from Handler mode to Thread mode, enter sleep or deep sleep on return from an ISR.<br><br>Setting this bit enables an interrupt-driven application to avoid returning to an empty main application. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

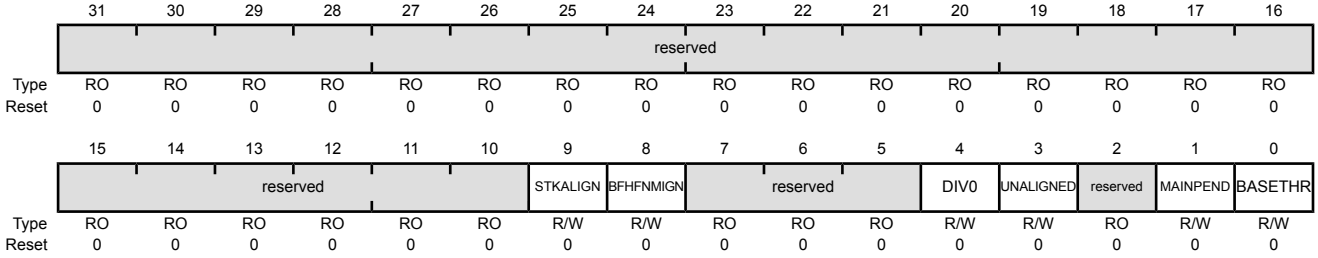
## Register 23: Configuration and Control (CFGCTRL), offset 0xD14

**Note:** This register can only be accessed from privileged mode.

The **CFGCTRL** register controls entry to Thread mode and enables: the handlers for NMI, hard fault and faults escalated by the **FAULTMASK** register to ignore bus faults; trapping of divide by zero and unaligned accesses; and access to the **SWTRIG** register by unprivileged software (see page 102).

Configuration and Control (CFGCTRL)

Base 0xE000.E000
Offset 0xD14
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | reserved | RO | 0x0000.00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9 | STKALIGN | R/W | 0 | Stack Alignment on Exception Entry<br><br>Value Description<br>0 The stack is 4-byte aligned.<br>1 The stack is 8-byte aligned.<br><br>On exception entry, the processor uses bit 9 of the stacked **PSR** to indicate the stack alignment. On return from the exception, it uses this stacked bit to restore the correct stack alignment. |
| 8 | BFHFNMIGN | R/W | 0 | Ignore Bus Fault in NMI and Fault<br>This bit enables handlers with priority -1 or -2 to ignore data bus faults caused by load and store instructions. The setting of this bit applies to the hard fault, NMI, and **FAULTMASK** escalated handlers.<br><br>Value Description<br>0 Data bus faults caused by load and store instructions cause a lock-up.<br>1 Handlers running at priority -1 and -2 ignore data bus faults caused by load and store instructions.<br><br>Set this bit only when the handler and its data are in absolutely safe memory. The normal use of this bit is to probe system devices and bridges to detect control path problems and fix them. |
| 7:5 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

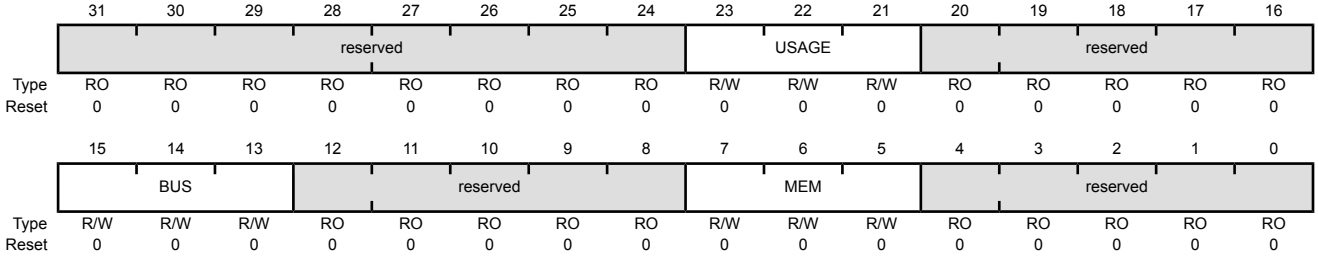
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | DIV0 | R/W | 0 | Trap on Divide by 0<br>This bit enables faulting or halting when the processor executes an SDIV or UDIV instruction with a divisor of 0.<br><br>Value Description<br>0 Do not trap on divide by 0. A divide by zero returns a quotient of 0.<br>1 Trap on divide by 0. |
| 3 | UNALIGNED | R/W | 0 | Trap on Unaligned Access<br><br>Value Description<br>0 Do not trap on unaligned halfword and word accesses.<br>1 Trap on unaligned halfword and word accesses. An unaligned access generates a usage fault.<br><br>Unaligned LDM, STM, LDRD, and STRD instructions always fault regardless of whether UNALIGNED is set. |
| 2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | MAINPEND | R/W | 0 | Allow Main Interrupt Trigger<br><br>Value Description<br>0 Disables unprivileged software access to the **SWTRIG** register.<br>1 Enables unprivileged software access to the **SWTRIG** register (see page 102). |
| 0 | BASETHR | R/W | 0 | Thread State Control<br><br>Value Description<br>0 The processor can enter Thread mode only when no exception is active.<br>1 The processor can enter Thread mode from any level under the control of an EXC_RETURN value (see “Exception Return” on page 73 for more information). |

## Register 24: System Handler Priority 1 (SYSPRI1), offset 0xD18

**Note:** This register can only be accessed from privileged mode.

The **SYSPRI1** register configures the priority level, 0 to 7 of the usage fault, bus fault, and memory management fault exception handlers. This register is byte-accessible.

System Handler Priority 1 (SYSPRI1)
Base 0xE000.E000
Offset 0xD18
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:24 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:21 | USAGE | R/W | 0x0 | Usage Fault Priority<br>This field configures the priority level of the usage fault. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 20:16 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:13 | BUS | R/W | 0x0 | Bus Fault Priority<br>This field configures the priority level of the bus fault. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 12:8 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:5 | MEM | R/W | 0x0 | Memory Management Fault Priority<br>This field configures the priority level of the memory management fault. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 4:0 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

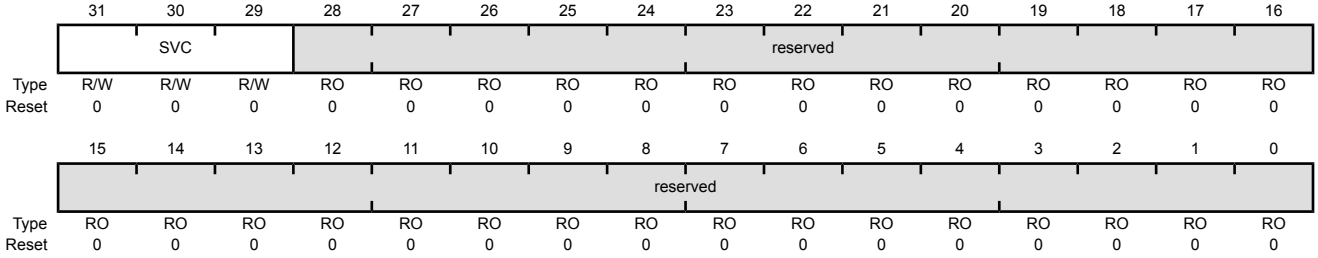
## Register 25: System Handler Priority 2 (SYSPRI2), offset 0xD1C

**Note:** This register can only be accessed from privileged mode.

The **SYSPRI2** register configures the priority level, 0 to 7 of the SVCall handler. This register is byte-accessible.

System Handler Priority 2 (SYSPRI2)

Base 0xE000.E000
Offset 0xD1C
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:29 | SVC | R/W | 0x0 | SVCall Priority<br>This field configures the priority level of SVCall. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 28:0 | reserved | RO | 0x000.0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

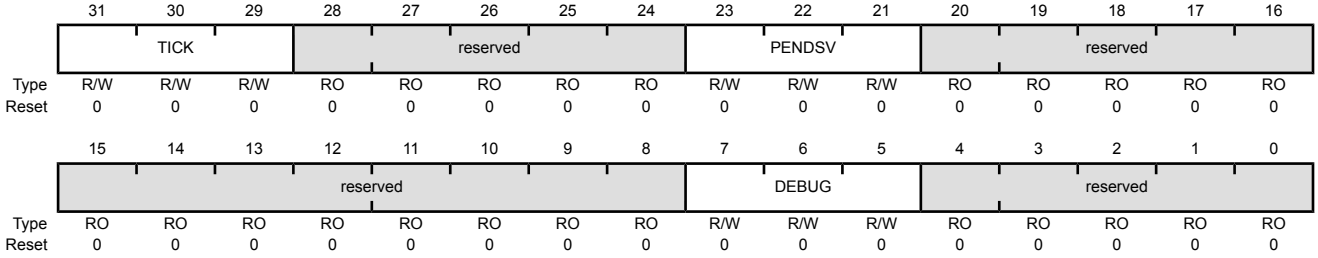
## Register 26: System Handler Priority 3 (SYSPRI3), offset 0xD20

**Note:** This register can only be accessed from privileged mode.

The **SYSPRI3** register configures the priority level, 0 to 7 of the SysTick exception and PendSV handlers. This register is byte-accessible.

System Handler Priority 3 (SYSPRI3)

Base 0xE000.E000
Offset 0xD20
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:29 | TICK | R/W | 0x0 | SysTick Exception Priority<br>This field configures the priority level of the SysTick exception. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 28:24 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:21 | PENDSV | R/W | 0x0 | PendSV Priority<br>This field configures the priority level of PendSV. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 20:8 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:5 | DEBUG | R/W | 0x0 | Debug Priority<br>This field configures the priority level of Debug. Configurable priority values are in the range 0-7, with lower values having higher priority. |
| 4:0 | reserved | RO | 0x0.0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 27: System Handler Control and State (SYSHNDCTRL), offset 0xD24

**Note:** This register can only be accessed from privileged mode.

The **SYSHNDCTRL** register enables the system handlers, and indicates the pending status of the usage fault, bus fault, memory management fault, and SVC exceptions as well as the active status of the system handlers.

If a system handler is disabled and the corresponding fault occurs, the processor treats the fault as a hard fault.

This register can be modified to change the pending or active status of system exceptions. An OS kernel can write to the active bits to perform a context switch that changes the current exception type.

**Caution – Software that changes the value of an active bit in this register without correct adjustment to the stacked content can cause the processor to generate a fault exception. Ensure software that writes to this register retains and subsequently restores the current active status.**

**If the value of a bit in this register must be modified after enabling the system handlers, a read-modify-write procedure must be used to ensure that only the required bit is modified.**

System Handler Control and State (SYSHNDCTRL)

Base 0xE000.E000
Offset 0xD24
Type R/W, reset 0x0000.0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | | | | | | | | | | | USAGE | BUS | MEM |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SVC | BUSP | MEMP | USAGEP | TICK | PNDSV | reserved | MON | SVCA | reserved | | | USGA | reserved | BUSA | MEMA |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | RO | R/W | R/W | RO | RO | RO | R/W | RO | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | USAGE | R/W | 0 | Usage Fault Enable<br>Value Description<br>0 Disables the usage fault exception.<br>1 Enables the usage fault exception. |
| 17 | BUS | R/W | 0 | Bus Fault Enable<br>Value Description<br>0 Disables the bus fault exception.<br>1 Enables the bus fault exception. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 16 | MEM | R/W | 0 | Memory Management Fault Enable<br><br>Value Description<br>0 Disables the memory management fault exception.<br>1 Enables the memory management fault exception. |
| 15 | SVC | R/W | 0 | SVC Call Pending<br><br>Value Description<br>0 An SVC call exception is not pending.<br>1 An SVC call exception is pending.<br><br>This bit can be modified to change the pending status of the SVC call exception. |
| 14 | BUSP | R/W | 0 | Bus Fault Pending<br><br>Value Description<br>0 A bus fault exception is not pending.<br>1 A bus fault exception is pending.<br><br>This bit can be modified to change the pending status of the bus fault exception. |
| 13 | MEMP | R/W | 0 | Memory Management Fault Pending<br><br>Value Description<br>0 A memory management fault exception is not pending.<br>1 A memory management fault exception is pending.<br><br>This bit can be modified to change the pending status of the memory management fault exception. |
| 12 | USAGEP | R/W | 0 | Usage Fault Pending<br><br>Value Description<br>0 A usage fault exception is not pending.<br>1 A usage fault exception is pending.<br><br>This bit can be modified to change the pending status of the usage fault exception. |
| 11 | TICK | R/W | 0 | SysTick Exception Active<br><br>Value Description<br>0 A SysTick exception is not active.<br>1 A SysTick exception is active.<br><br>This bit can be modified to change the active status of the SysTick exception, however, see the Caution above before setting this bit. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 10 | PNDSV | R/W | 0 | PendSV Exception Active<br><br>Value Description<br>0 A PendSV exception is not active.<br>1 A PendSV exception is active.<br><br>This bit can be modified to change the active status of the PendSV exception, however, see the Caution above before setting this bit. |
| 9 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 8 | MON | R/W | 0 | Debug Monitor Active<br><br>Value Description<br>0 The Debug monitor is not active.<br>1 The Debug monitor is active. |
| 7 | SVCA | R/W | 0 | SVC Call Active<br><br>Value Description<br>0 SVC call is not active.<br>1 SVC call is active.<br><br>This bit can be modified to change the active status of the SVC call exception, however, see the Caution above before setting this bit. |
| 6:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | USGA | R/W | 0 | Usage Fault Active<br><br>Value Description<br>0 Usage fault is not active.<br>1 Usage fault is active.<br><br>This bit can be modified to change the active status of the usage fault exception, however, see the Caution above before setting this bit. |
| 2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | BUSA | R/W | 0 | Bus Fault Active<br><br>Value Description<br>0 Bus fault is not active.<br>1 Bus fault is active.<br><br>This bit can be modified to change the active status of the bus fault exception, however, see the Caution above before setting this bit. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | MEMA | R/W | 0 | Memory Management Fault Active |

| Value | Description |
|---|---|
| 0 | Memory management fault is not active. |
| 1 | Memory management fault is active. |

This bit can be modified to change the active status of the memory management fault exception, however, see the Caution above before setting this bit.

## Register 28: Configurable Fault Status (FAULTSTAT), offset 0xD28

**Note:** This register can only be accessed from privileged mode.

The **FAULTSTAT** register indicates the cause of a memory management fault, bus fault, or usage fault. Each of these functions is assigned to a subregister as follows:

- **Usage Fault Status (UFAULTSTAT)**, bits 31:16
- **Bus Fault Status (BFAULTSTAT)**, bits 15:8
- **Memory Management Fault Status (MFAULTSTAT)**, bits 7:0

**FAULTSTAT** is byte accessible. **FAULTSTAT** or its subregisters can be accessed as follows:

- The complete **FAULTSTAT** register, with a word access to offset 0xD28
- The **MFAULTSTAT**, with a byte access to offset 0xD28
- The **MFAULTSTAT** and **BFAULTSTAT**, with a halfword access to offset 0xD28
- The **BFAULTSTAT**, with a byte access to offset 0xD29
- The **UFAULTSTAT**, with a halfword access to offset 0xD2A

Bits are cleared by writing a 1 to them.

In a fault handler, the true faulting address can be determined by:

1. Read and save the **Memory Management Fault Address (MMADDR)** or **Bus Fault Address (FAULTADDR)** value.
2. Read the MMARV bit in **MFAULTSTAT**, or the BFARV bit in **BFAULTSTAT** to determine if the **MMADDR** or **FAULTADDR** contents are valid.

Software must follow this sequence because another higher priority exception might change the **MMADDR** or **FAULTADDR** value. For example, if a higher priority handler preempts the current fault handler, the other fault might change the **MMADDR** or **FAULTADDR** value.

Configurable Fault Status (FAULTSTAT)
Base 0xE000.E000
Offset 0xD28
Type R/W1C, reset 0x0000.0000

| Bit | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | reserved | | | | | | DIV0 | UNALIGN | reserved | | | | NOCP | INVPC | INVSTAT | UNDEF |
| Type | RO | RO | RO | RO | RO | RO | R/W1C | R/W1C | RO | RO | RO | RO | R/W1C | R/W1C | R/W1C | R/W1C |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | BFARV | reserved | | BSTKE | BUSTKE | IMPRE | PRECISE | IBUS | MMARV | reserved | | MSTKE | MUSTKE | reserved | DERR | IERR |
| Type | R/W1C | RO | RO | R/W1C | R/W1C | R/W1C | R/W1C | R/W1C | R/W1C | RO | RO | R/W1C | R/W1C | RO | R/W1C | R/W1C |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:26 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 25 | DIV0 | R/W1C | 0 | Divide-by-Zero Usage Fault<br><br>Value Description<br>0 No divide-by-zero fault has occurred, or divide-by-zero trapping is not enabled.<br>1 The processor has executed an SDIV or UDIV instruction with a divisor of 0.<br><br>When this bit is set, the **PC** value stacked for the exception return points to the instruction that performed the divide by zero.<br>Trapping on divide-by-zero is enabled by setting the DIV0 bit in the **Configuration and Control (CFGCTRL)** register (see page 112).<br>This bit is cleared by writing a 1 to it. |
| 24 | UNALIGN | R/W1C | 0 | Unaligned Access Usage Fault<br><br>Value Description<br>0 No unaligned access fault has occurred, or unaligned access trapping is not enabled.<br>1 The processor has made an unaligned memory access.<br><br>Unaligned LDM, STM, LDRD, and STRD instructions always fault regardless of the configuration of this bit.<br>Trapping on unaligned access is enabled by setting the UNALIGNED bit in the **CFGCTRL** register (see page 112).<br>This bit is cleared by writing a 1 to it. |
| 23:20 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 19 | NOCP | R/W1C | 0 | No Coprocessor Usage Fault<br><br>Value Description<br>0 A usage fault has not been caused by attempting to access a coprocessor.<br>1 The processor has attempted to access a coprocessor.<br><br>This bit is cleared by writing a 1 to it. |
| 18 | INVPC | R/W1C | 0 | Invalid PC Load Usage Fault<br><br>Value Description<br>0 A usage fault has not been caused by attempting to load an invalid **PC** value.<br>1 The processor has attempted an illegal load of EXC_RETURN to the **PC** as a result of an invalid context or an invalid EXC_RETURN value.<br><br>When this bit is set, the **PC** value stacked for the exception return points to the instruction that tried to perform the illegal load of the **PC**.<br>This bit is cleared by writing a 1 to it. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 17 | INVSTAT | R/W1C | 0 | Invalid State Usage Fault<br><br>Value Description<br>0 A usage fault has not been caused by an invalid state.<br>1 The processor has attempted to execute an instruction that makes illegal use of the **EPSR** register.<br><br>When this bit is set, the **PC** value stacked for the exception return points to the instruction that attempted the illegal use of the **Execution Program Status Register (EPSR)** register.<br>This bit is not set if an undefined instruction uses the **EPSR** register.<br>This bit is cleared by writing a 1 to it. |
| 16 | UNDEF | R/W1C | 0 | Undefined Instruction Usage Fault<br><br>Value Description<br>0 A usage fault has not been caused by an undefined instruction.<br>1 The processor has attempted to execute an undefined instruction.<br><br>When this bit is set, the **PC** value stacked for the exception return points to the undefined instruction.<br>An undefined instruction is an instruction that the processor cannot decode.<br>This bit is cleared by writing a 1 to it. |
| 15 | BFARV | R/W1C | 0 | Bus Fault Address Register Valid<br><br>Value Description<br>0 The value in the **Bus Fault Address (FAULTADDR)** register is not a valid fault address.<br>1 The **FAULTADDR** register is holding a valid fault address.<br><br>This bit is set after a bus fault, where the address is known. Other faults can clear this bit, such as a memory management fault occurring later.<br>If a bus fault occurs and is escalated to a hard fault because of priority, the hard fault handler must clear this bit. This action prevents problems if returning to a stacked active bus fault handler whose **FAULTADDR** register value has been overwritten.<br>This bit is cleared by writing a 1 to it. |
| 14:13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12 | BSTKE | R/W1C | 0 | Stack Bus Fault<br><br>Value Description<br>0 No bus fault has occurred on stacking for exception entry.<br>1 Stacking for an exception entry has caused one or more bus faults.<br><br>When this bit is set, the **SP** is still adjusted but the values in the context area on the stack might be incorrect. A fault address is not written to the **FAULTADDR** register.<br>This bit is cleared by writing a 1 to it. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 11 | BUSTKE | R/W1C | 0 | Unstack Bus Fault<br><br>Value Description<br>0 No bus fault has occurred on unstacking for a return from exception.<br>1 Unstacking for a return from exception has caused one or more bus faults.<br><br>This fault is chained to the handler. Thus, when this bit is set, the original return stack is still present. The **SP** is not adjusted from the failing return, a new save is not performed, and a fault address is not written to the **FAULTADDR** register.<br>This bit is cleared by writing a 1 to it. |
| 10 | IMPRE | R/W1C | 0 | Imprecise Data Bus Error<br><br>Value Description<br>0 An imprecise data bus error has not occurred.<br>1 A data bus error has occurred, but the return address in the stack frame is not related to the instruction that caused the error.<br><br>When this bit is set, a fault address is not written to the **FAULTADDR** register.<br>This fault is asynchronous. Therefore, if the fault is detected when the priority of the current process is higher than the bus fault priority, the bus fault becomes pending and becomes active only when the processor returns from all higher-priority processes. If a precise fault occurs before the processor enters the handler for the imprecise bus fault, the handler detects that both the `IMPRE` bit is set and one of the precise fault status bits is set.<br>This bit is cleared by writing a 1 to it. |
| 9 | PRECISE | R/W1C | 0 | Precise Data Bus Error<br><br>Value Description<br>0 A precise data bus error has not occurred.<br>1 A data bus error has occurred, and the **PC** value stacked for the exception return points to the instruction that caused the fault.<br><br>When this bit is set, the fault address is written to the **FAULTADDR** register.<br>This bit is cleared by writing a 1 to it. |
| 8 | IBUS | R/W1C | 0 | Instruction Bus Error<br><br>Value Description<br>0 An instruction bus error has not occurred.<br>1 An instruction bus error has occurred.<br><br>The processor detects the instruction bus error on prefetching an instruction, but sets this bit only if it attempts to issue the faulting instruction.<br>When this bit is set, a fault address is not written to the **FAULTADDR** register.<br>This bit is cleared by writing a 1 to it. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | MMARV | R/W1C | 0 | Memory Management Fault Address Register Valid<br><br>Value Description<br>0 The value in the **Memory Management Fault Address (MMADDR)** register is not a valid fault address.<br>1 The **MMADDR** register is holding a valid fault address.<br><br>If a memory management fault occurs and is escalated to a hard fault because of priority, the hard fault handler must clear this bit. This action prevents problems if returning to a stacked active memory management fault handler whose **MMADDR** register value has been overwritten.<br>This bit is cleared by writing a 1 to it. |
| 6:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | MSTKE | R/W1C | 0 | Stack Access Violation<br><br>Value Description<br>0 No memory management fault has occurred on stacking for exception entry.<br>1 Stacking for an exception entry has caused one or more access violations.<br><br>When this bit is set, the **SP** is still adjusted but the values in the context area on the stack might be incorrect. A fault address is not written to the **MMADDR** register.<br>This bit is cleared by writing a 1 to it. |
| 3 | MUSTKE | R/W1C | 0 | Unstack Access Violation<br><br>Value Description<br>0 No memory management fault has occurred on unstacking for a return from exception.<br>1 Unstacking for a return from exception has caused one or more access violations.<br><br>This fault is chained to the handler. Thus, when this bit is set, the original return stack is still present. The **SP** is not adjusted from the failing return, a new save is not performed, and a fault address is not written to the **MMADDR** register.<br>This bit is cleared by writing a 1 to it. |
| 2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | DERR | R/W1C | 0 | Data Access Violation<br><br>Value Description<br>0 A data access violation has not occurred.<br>1 The processor attempted a load or store at a location that does not permit the operation.<br><br>When this bit is set, the **PC** value stacked for the exception return points to the faulting instruction and the address of the attempted access is written to the **MMADDR** register.<br>This bit is cleared by writing a 1 to it. |
| 0 | IERR | R/W1C | 0 | Instruction Access Violation<br><br>Value Description<br>0 An instruction access violation has not occurred.<br>1 The processor attempted an instruction fetch from a location that does not permit execution.<br><br>This fault occurs on any access to an XN region, even when the MPU is disabled or not present.<br>When this bit is set, the **PC** value stacked for the exception return points to the faulting instruction and the address of the attempted access is not written to the **MMADDR** register.<br>This bit is cleared by writing a 1 to it. |

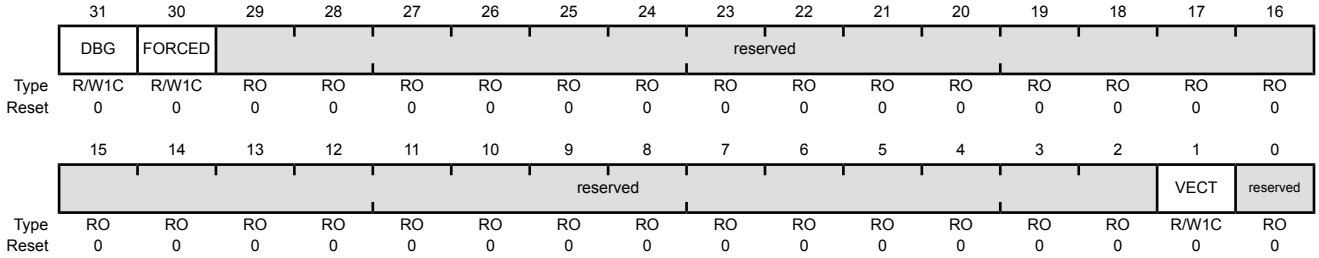
## Register 29: Hard Fault Status (HFAULTSTAT), offset 0xD2C

**Note:** This register can only be accessed from privileged mode.

The **HFAULTSTAT** register gives information about events that activate the hard fault handler. Bits are cleared by writing a 1 to them.

Hard Fault Status (HFAULTSTAT)

Base 0xE000.E000
Offset 0xD2C
Type R/W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | DBG | R/W1C | 0 | Debug Event<br>This bit is reserved for Debug use. This bit must be written as a 0, otherwise behavior is unpredictable. |
| 30 | FORCED | R/W1C | 0 | Forced Hard Fault<br><br>Value Description<br>0 No forced hard fault has occurred.<br>1 A forced hard fault has been generated by escalation of a fault with configurable priority that cannot be handled, either because of priority or because it is disabled.<br><br>When this bit is set, the hard fault handler must read the other fault status registers to find the cause of the fault.<br>This bit is cleared by writing a 1 to it. |
| 29:2 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | VECT | R/W1C | 0 | Vector Table Read Fault<br><br>Value Description<br>0 No bus fault has occurred on a vector table read.<br>1 A bus fault occurred on a vector table read.<br><br>This error is always handled by the hard fault handler.<br>When this bit is set, the **PC** value stacked for the exception return points to the instruction that was preempted by the exception.<br>This bit is cleared by writing a 1 to it. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

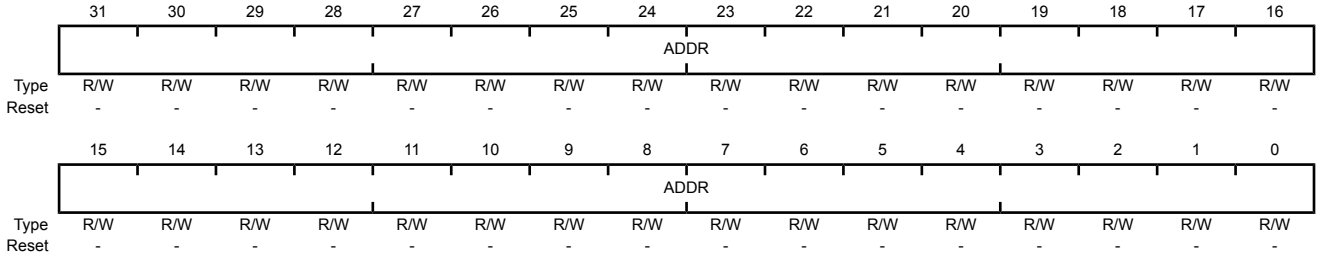
## Register 30: Memory Management Fault Address (MMADDR), offset 0xD34

**Note:** This register can only be accessed from privileged mode.

The **MMADDR** register contains the address of the location that generated a memory management fault. When an unaligned access faults, the address in the **MMADDR** register is the actual address that faulted. Because a single read or write instruction can be split into multiple aligned accesses, the fault address can be any address in the range of the requested access size. Bits in the **Memory Management Fault Status (MFAULTSTAT)** register indicate the cause of the fault and whether the value in the **MMADDR** register is valid (see page 121).

Memory Management Fault Address (MMADDR)

Base 0xE000.E000
Offset 0xD34
Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | ADDR | R/W | - | Fault Address<br>When the MMARV bit of **MFAULTSTAT** is set, this field holds the address of the location that generated the memory management fault. |

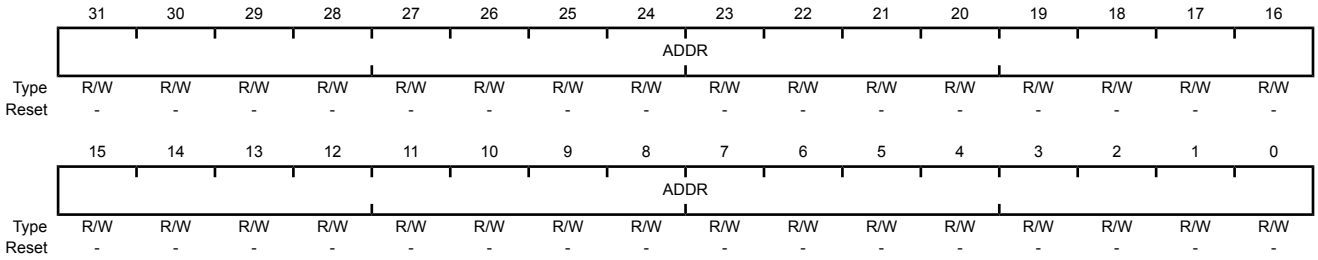
### Register 31: Bus Fault Address (FAULTADDR), offset 0xD38

**Note:** This register can only be accessed from privileged mode.

The **FAULTADDR** register contains the address of the location that generated a bus fault. When an unaligned access faults, the address in the **FAULTADDR** register is the one requested by the instruction, even if it is not the address of the fault. Bits in the **Bus Fault Status (BFAULTSTAT)** register indicate the cause of the fault and whether the value in the **FAULTADDR** register is valid (see page 121).

Bus Fault Address (FAULTADDR)

Base 0xE000.E000
Offset 0xD38
Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | ADDR | R/W | - | Fault Address<br>When the FAULTADDRV bit of **BFAULTSTAT** is set, this field holds the address of the location that generated the bus fault. |

## 3.6 Memory Protection Unit (MPU) Register Descriptions

This section lists and describes the Memory Protection Unit (MPU) registers, in numerical order by address offset.

The MPU registers can only be accessed from privileged mode.

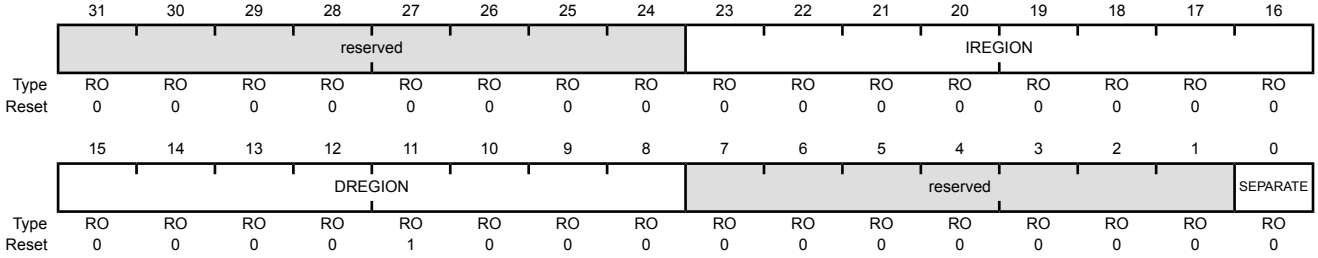
## Register 32: MPU Type (MPUTYPE), offset 0xD90

**Note:** This register can only be accessed from privileged mode.

The **MPUTYPE** register indicates whether the MPU is present, and if so, how many regions it supports.

MPU Type (MPUTYPE)

Base 0xE000.E000
Offset 0xD90
Type RO, reset 0x0000.0800




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:24 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 23:16 | IREGION | RO | 0x00 | Number of I Regions<br>This field indicates the number of supported MPU instruction regions. This field always contains 0x00. The MPU memory map is unified and is described by the `DREGION` field. |
| 15:8 | DREGION | RO | 0x08 | Number of D Regions<br>Value Description<br>0x08 Indicates there are eight supported MPU data regions. |
| 7:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | SEPARATE | RO | 0 | Separate or Unified MPU<br>Value Description<br>0 Indicates the MPU is unified. |

## Register 33: MPU Control (MPUCTRL), offset 0xD94

**Note:** This register can only be accessed from privileged mode.

The **MPUCTRL** register enables the MPU, enables the default memory map background region, and enables use of the MPU when in the hard fault, Non-maskable Interrupt (NMI), and **Fault Mask Register (FAULTMASK)** escalated handlers.

When the ENABLE and PRIVDEFEN bits are both set:

- For privileged accesses, the default memory map is as described in "Memory Model" on page 58. Any access by privileged software that does not address an enabled memory region behaves as defined by the default memory map.
- Any access by unprivileged software that does not address an enabled memory region causes a memory management fault.

Execute Never (XN) and Strongly Ordered rules always apply to the System Control Space regardless of the value of the ENABLE bit.

When the ENABLE bit is set, at least one region of the memory map must be enabled for the system to function unless the PRIVDEFEN bit is set. If the PRIVDEFEN bit is set and no regions are enabled, then only privileged software can operate.

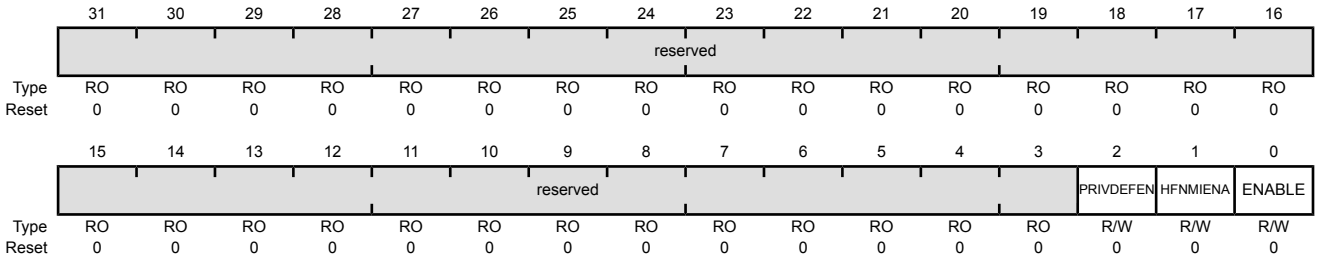
When the ENABLE bit is clear, the system uses the default memory map, which has the same memory attributes as if the MPU is not implemented (see Table 2-5 on page 60 for more information). The default memory map applies to accesses from both privileged and unprivileged software.

When the MPU is enabled, accesses to the System Control Space and vector table are always permitted. Other areas are accessible based on regions and whether PRIVDEFEN is set.

Unless HFNMIENA is set, the MPU is not enabled when the processor is executing the handler for an exception with priority –1 or –2. These priorities are only possible when handling a hard fault or NMI exception or when **FAULTMASK** is enabled. Setting the HFNMIENA bit enables the MPU when operating with these two priorities.

MPU Control (MPUCTRL)

Base 0xE000.E000
Offset 0xD94
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | PRIVDEFEN | R/W | 0 | MPU Default Region<br>This bit enables privileged software access to the default memory map.<br><br>Value Description<br>0 If the MPU is enabled, this bit disables use of the default memory map. Any memory access to a location not covered by any enabled region causes a fault.<br>1 If the MPU is enabled, this bit enables use of the default memory map as a background region for privileged software accesses.<br><br>When this bit is set, the background region acts as if it is region number -1. Any region that is defined and enabled has priority over this default map.<br>If the MPU is disabled, the processor ignores this bit. |
| 1 | HFNMIENA | R/W | 0 | MPU Enabled During Faults<br>This bit controls the operation of the MPU during hard fault, NMI, and **FAULTMASK** handlers.<br><br>Value Description<br>0 The MPU is disabled during hard fault, NMI, and **FAULTMASK** handlers, regardless of the value of the ENABLE bit.<br>1 The MPU is enabled during hard fault, NMI, and **FAULTMASK** handlers.<br><br>When the MPU is disabled and this bit is set, the resulting behavior is unpredictable. |
| 0 | ENABLE | R/W | 0 | MPU Enable<br><br>Value Description<br>0 The MPU is disabled.<br>1 The MPU is enabled.<br><br>When the MPU is disabled and the HFNMIENA bit is set, the resulting behavior is unpredictable. |

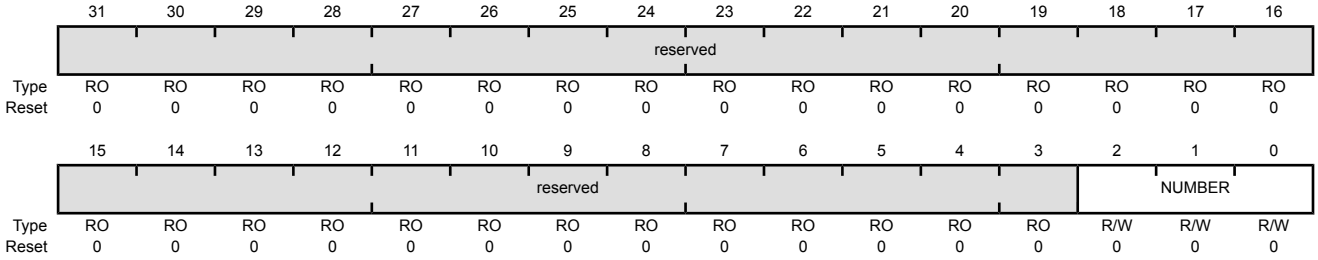
## Register 34: MPU Region Number (MPUNUMBER), offset 0xD98

**Note:** This register can only be accessed from privileged mode.

The **MPUNUMBER** register selects which memory region is referenced by the **MPU Region Base Address (MPUBASE)** and **MPU Region Attribute and Size (MPUATTR)** registers. Normally, the required region number should be written to this register before accessing the **MPUBASE** or the **MPUATTR** register. However, the region number can be changed by writing to the **MPUBASE** register with the VALID bit set (see page 134). This write updates the value of the REGION field.

MPU Region Number (MPUNUMBER)

Base 0xE000.E000
Offset 0xD98
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | NUMBER | R/W | 0x0 | MPU Region to Access<br>This field indicates the MPU region referenced by the **MPUBASE** and **MPUATTR** registers. The MPU supports eight memory regions. |

### Register 35: MPU Region Base Address (MPUBASE), offset 0xD9C

### Register 36: MPU Region Base Address Alias 1 (MPUBASE1), offset 0xDA4

### Register 37: MPU Region Base Address Alias 2 (MPUBASE2), offset 0xDAC

### Register 38: MPU Region Base Address Alias 3 (MPUBASE3), offset 0xDB4

**Note:** This register can only be accessed from privileged mode.

The **MPUBASE** register defines the base address of the MPU region selected by the **MPU Region Number (MPUNUMBER)** register and can update the value of the **MPUNUMBER** register. To change the current region number and update the **MPUNUMBER** register, write the **MPUBASE** register with the VALID bit set.

The ADDR field is bits 31:*N* of the **MPUBASE** register. Bits (*N*-1):5 are reserved. The region size, as specified by the SIZE field in the **MPU Region Attribute and Size (MPUATTR)** register, defines the value of *N* where:

$N = \text{Log}_2(\text{Region size in bytes})$

If the region size is configured to 4 GB in the **MPUATTR** register, there is no valid ADDR field. In this case, the region occupies the complete memory map, and the base address is 0x0000.0000.

The base address is aligned to the size of the region. For example, a 64-KB region must be aligned on a multiple of 64 KB, for example, at 0x0001.0000 or 0x0002.0000.

MPU Region Base Address (MPUBASE)

Base 0xE000.E000
Offset 0xD9C
Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ADDR | | | | | | | | | | | | | | | |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ADDR | | | | | | | | | | | VALID | reserved | REGION | | |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | WO | RO | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | ADDR | R/W | 0x0000.000 | Base Address Mask<br>Bits 31:*N* in this field contain the region base address. The value of *N* depends on the region size, as shown above. The remaining bits (*N*-1):5 are reserved.<br>Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | VALID | WO | 0 | Region Number Valid<br><br>Value Description<br>0 The **MPUNUMBER** register is not changed and the processor updates the base address for the region specified in the **MPUNUMBER** register and ignores the value of the `REGION` field.<br>1 The **MPUNUMBER** register is updated with the value of the `REGION` field and the base address is updated for the region specified in the `REGION` field.<br><br>This bit is always read as 0. |
| 3 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | REGION | R/W | 0x0 | Region Number<br>On a write, contains the value to be written to the **MPUNUMBER** register. On a read, returns the current region number in the **MPUNUMBER** register. |

## Register 39: MPU Region Attribute and Size (MPUATTR), offset 0xDA0

## Register 40: MPU Region Attribute and Size Alias 1 (MPUATTR1), offset 0xDA8

## Register 41: MPU Region Attribute and Size Alias 2 (MPUATTR2), offset 0xDB0

## Register 42: MPU Region Attribute and Size Alias 3 (MPUATTR3), offset 0xDB8

**Note:** This register can only be accessed from privileged mode.

The **MPUATTR** register defines the region size and memory attributes of the MPU region specified by the **MPU Region Number (MPUNUMBER)** register and enables that region and any subregions.

The **MPUATTR** register is accessible using word or halfword accesses with the most-significant halfword holding the region attributes and the least-significant halfword holds the region size and the region and subregion enable bits.

The MPU access permission attribute bits, XN, AP, TEX, S, C, and B, control access to the corresponding memory region. If an access is made to an area of memory without the required permissions, then the MPU generates a permission fault.

The SIZE field defines the size of the MPU memory region specified by the **MPUNUMBER** register as follows:

(Region size in bytes) = $2^{(SIZE+1)}$

The smallest permitted region size is 32 bytes, corresponding to a SIZE value of 4. Table 3-9 on page 136 gives example SIZE values with the corresponding region size and value of N in the **MPU Region Base Address (MPUBASE)** register.

**Table 3-9. Example SIZE Field Values**

| SIZE Encoding | Region Size | Value of N[a] | Note |
|---|---|---|---|
| 00100b (0x4) | 32 B | 5 | Minimum permitted size |
| 01001b (0x9) | 1 KB | 10 | - |
| 10011b (0x13) | 1 MB | 20 | - |
| 11101b (0x1D) | 1 GB | 30 | - |
| 11111b (0x1F) | 4 GB | No valid ADDR field in **MPUBASE**; the region occupies the complete memory map. | Maximum possible size |

a. Refers to the N parameter in the **MPUBASE** register (see page 134).

MPU Region Attribute and Size (MPUATTR)

Base 0xE000.E000
Offset 0xDA0
Type R/W, reset 0x0000.0000

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| reserved | | | XN | reserved | AP | | | reserved | | TEX | | | S | C | B |
| Type | RO | RO | RO | R/W | RO | R/W | R/W | R/W | RO | RO | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SRD | | | | | | | | reserved | | SIZE | | | | | ENABLE |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | RO | RO | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:29 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 28 | XN | R/W | 0 | Instruction Access Disable<br>Value Description<br>0 Instruction fetches are enabled.<br>1 Instruction fetches are disabled. |
| 27 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 26:24 | AP | R/W | 0 | Access Privilege<br>For information on using this bit field, see Table 3-5 on page 88. |
| 23:22 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 21:19 | TEX | R/W | 0x0 | Type Extension Mask<br>For information on using this bit field, see Table 3-3 on page 87. |
| 18 | S | R/W | 0 | Shareable<br>For information on using this bit, see Table 3-3 on page 87. |
| 17 | C | R/W | 0 | Cacheable<br>For information on using this bit, see Table 3-3 on page 87. |
| 16 | B | R/W | 0 | Bufferable<br>For information on using this bit, see Table 3-3 on page 87. |
| 15:8 | SRD | R/W | 0x00 | Subregion Disable Bits<br>Value Description<br>0 The corresponding subregion is enabled.<br>1 The corresponding subregion is disabled.<br>Region sizes of 128 bytes and less do not support subregions. When writing the attributes for such a region, configure the SRD field as 0x00. See the section called “Subregions” on page 86 for more information. |
| 7:6 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:1 | SIZE | R/W | 0x0 | Region Size Mask<br>The SIZE field defines the size of the MPU memory region specified by the **MPUNUMBER** register. Refer to Table 3-9 on page 136 for more information. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | ENABLE | R/W | 0 | Region Enable |

| Value | Description |
|---|---|
| 0 | The region is disabled. |
| 1 | The region is enabled. |

# 4 JTAG Interface

The Joint Test Action Group (JTAG) port is an IEEE standard that defines a Test Access Port and Boundary Scan Architecture for digital integrated circuits and provides a standardized serial interface for controlling the associated test logic. The TAP, Instruction Register (IR), and Data Registers (DR) can be used to test the interconnections of assembled printed circuit boards and obtain manufacturing information on the components. The JTAG Port also provides a means of accessing and controlling design-for-test features such as I/O pin observation and control, scan testing, and debugging.

The JTAG port is comprised of five pins: $\overline{\text{TRST}}$, TCK, TMS, TDI, and TDO. Data is transmitted serially into the controller on TDI and out of the controller on TDO. The interpretation of this data is dependent on the current state of the TAP controller. For detailed information on the operation of the JTAG port and TAP controller, please refer to the *IEEE Standard 1149.1-Test Access Port and Boundary-Scan Architecture*.

The Stellaris® JTAG controller works with the ARM JTAG controller built into the Cortex-M3 core. This is implemented by multiplexing the TDO outputs from both JTAG controllers. ARM JTAG instructions select the ARM TDO output while Stellaris JTAG instructions select the Stellaris TDO outputs. The multiplexer is controlled by the Stellaris JTAG controller, which has comprehensive programming for the ARM, Stellaris, and unimplemented JTAG instructions.

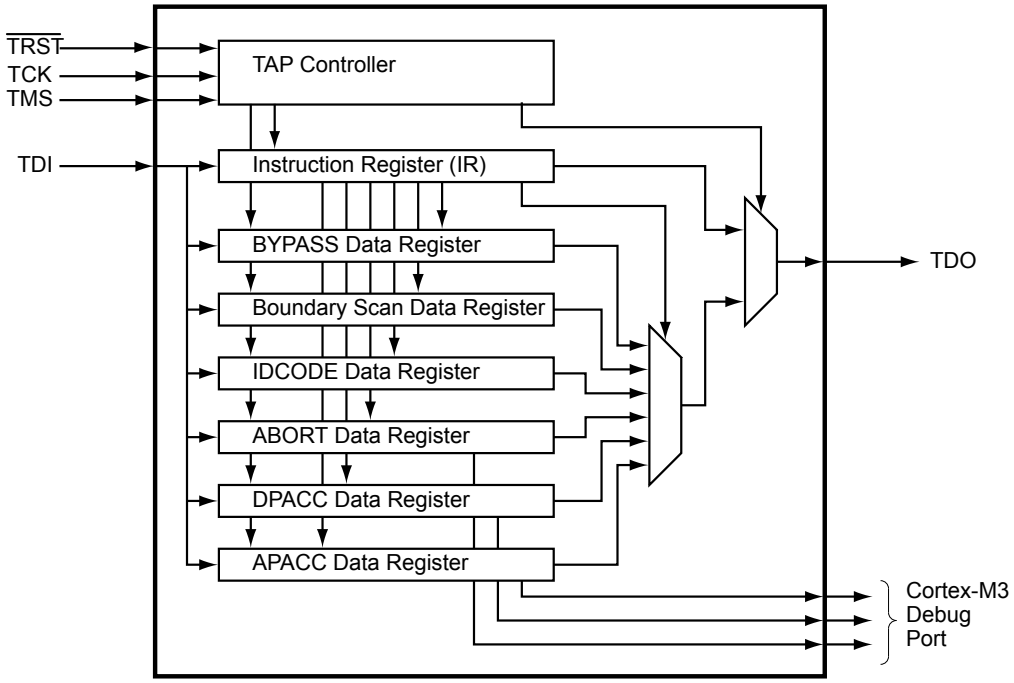
The Stellaris JTAG module has the following features:

- IEEE 1149.1-1990 compatible Test Access Port (TAP) controller
- Four-bit Instruction Register (IR) chain for storing JTAG instructions
- IEEE standard instructions: BYPASS, IDCODE, SAMPLE/PRELOAD, EXTEST and INTEST
- ARM additional instructions: APACC, DPACC and ABORT
- Integrated ARM Serial Wire Debug (SWD)

See the *ARM® Debug Interface V5 Architecture Specification* for more information on the ARM JTAG controller.

## 4.1 Block Diagram

**Figure 4-1. JTAG Module Block Diagram**




## 4.2 Functional Description

A high-level conceptual drawing of the JTAG module is shown in Figure 4-1 on page 140. The JTAG module is composed of the Test Access Port (TAP) controller and serial shift chains with parallel update registers. The TAP controller is a simple state machine controlled by the TRST, TCK and TMS inputs. The current state of the TAP controller depends on the current value of TRST and the sequence of values captured on TMS at the rising edge of TCK. The TAP controller determines when the serial shift chains capture new data, shift data from TDI towards TDO, and update the parallel load registers. The current state of the TAP controller also determines whether the Instruction Register (IR) chain or one of the Data Register (DR) chains is being accessed.

The serial shift chains with parallel load registers are comprised of a single Instruction Register (IR) chain and multiple Data Register (DR) chains. The current instruction loaded in the parallel load register determines which DR chain is captured, shifted, or updated during the sequencing of the TAP controller.

Some instructions, like EXTEST and INTEST, operate on data currently in a DR chain and do not capture, shift, or update any of the chains. Instructions that are not implemented decode to the BYPASS instruction to ensure that the serial path between TDI and TDO is always connected (see Table 4-2 on page 145 for a list of implemented instructions).

See “JTAG and Boundary Scan” on page 479 for JTAG timing diagrams.

### 4.2.1 JTAG Interface Pins

The JTAG interface consists of five standard pins: TRST,TCK, TMS, TDI, and TDO. These pins and their associated reset state are given in Table 4-1 on page 141. Detailed information on each pin follows.

**Table 4-1. JTAG Port Pins Reset State**

| Pin Name | Data Direction | Internal Pull-Up | Internal Pull-Down | Drive Strength | Drive Value |
|---|---|---|---|---|---|
| TRST | Input | Enabled | Disabled | N/A | N/A |
| TCK | Input | Enabled | Disabled | N/A | N/A |
| TMS | Input | Enabled | Disabled | N/A | N/A |
| TDI | Input | Enabled | Disabled | N/A | N/A |
| TDO | Output | Enabled | Disabled | 2-mA driver | High-Z |

#### 4.2.1.1 Test Reset Input ($\overline{\text{TRST}}$)

The $\overline{\text{TRST}}$ pin is an asynchronous active Low input signal for initializing and resetting the JTAG TAP controller and associated JTAG circuitry. When $\overline{\text{TRST}}$ is asserted, the TAP controller resets to the Test-Logic-Reset state and remains there while $\overline{\text{TRST}}$ is asserted. When the TAP controller enters the Test-Logic-Reset state, the JTAG Instruction Register (IR) resets to the default instruction, IDCODE.

By default, the internal pull-up resistor on the $\overline{\text{TRST}}$ pin is enabled after reset. Changes to the pull-up resistor settings on GPIO Port B should ensure that the internal pull-up resistor remains enabled on PB7/$\overline{\text{TRST}}$; otherwise JTAG communication could be lost.

#### 4.2.1.2 Test Clock Input (TCK)

The TCK pin is the clock for the JTAG module. This clock is provided so the test logic can operate independently of any other system clocks. In addition, it ensures that multiple JTAG TAP controllers that are daisy-chained together can synchronously communicate serial test data between components. During normal operation, TCK is driven by a free-running clock with a nominal 50% duty cycle. When necessary, TCK can be stopped at 0 or 1 for extended periods of time. While TCK is stopped at 0 or 1, the state of the TAP controller does not change and data in the JTAG Instruction and Data Registers is not lost.

By default, the internal pull-up resistor on the TCK pin is enabled after reset. This assures that no clocking occurs if the pin is not driven from an external source. The internal pull-up and pull-down resistors can be turned off to save internal power as long as the TCK pin is constantly being driven by an external source.

#### 4.2.1.3 Test Mode Select (TMS)

The TMS pin selects the next state of the JTAG TAP controller. TMS is sampled on the rising edge of TCK. Depending on the current TAP state and the sampled value of TMS, the next state is entered. Because the TMS pin is sampled on the rising edge of TCK, the *IEEE Standard 1149.1* expects the value on TMS to change on the falling edge of TCK.

Holding TMS high for five consecutive TCK cycles drives the TAP controller state machine to the Test-Logic-Reset state. When the TAP controller enters the Test-Logic-Reset state, the JTAG Instruction Register (IR) resets to the default instruction, IDCODE. Therefore, this sequence can be used as a reset mechanism, similar to asserting $\overline{\text{TRST}}$. The JTAG Test Access Port state machine can be seen in its entirety in Figure 4-2 on page 143.

By default, the internal pull-up resistor on the TMS pin is enabled after reset. Changes to the pull-up resistor settings on GPIO Port C should ensure that the internal pull-up resistor remains enabled on PC1/TMS; otherwise JTAG communication could be lost.

#### 4.2.1.4 Test Data Input (TDI)

The TDI pin provides a stream of serial information to the IR chain and the DR chains. TDI is sampled on the rising edge of TCK and, depending on the current TAP state and the current instruction, presents this data to the proper shift register chain. Because the TDI pin is sampled on the rising edge of TCK, the *IEEE Standard 1149.1* expects the value on TDI to change on the falling edge of TCK.

By default, the internal pull-up resistor on the TDI pin is enabled after reset. Changes to the pull-up resistor settings on GPIO Port C should ensure that the internal pull-up resistor remains enabled on PC2/TDI; otherwise JTAG communication could be lost.

#### 4.2.1.5 Test Data Output (TDO)

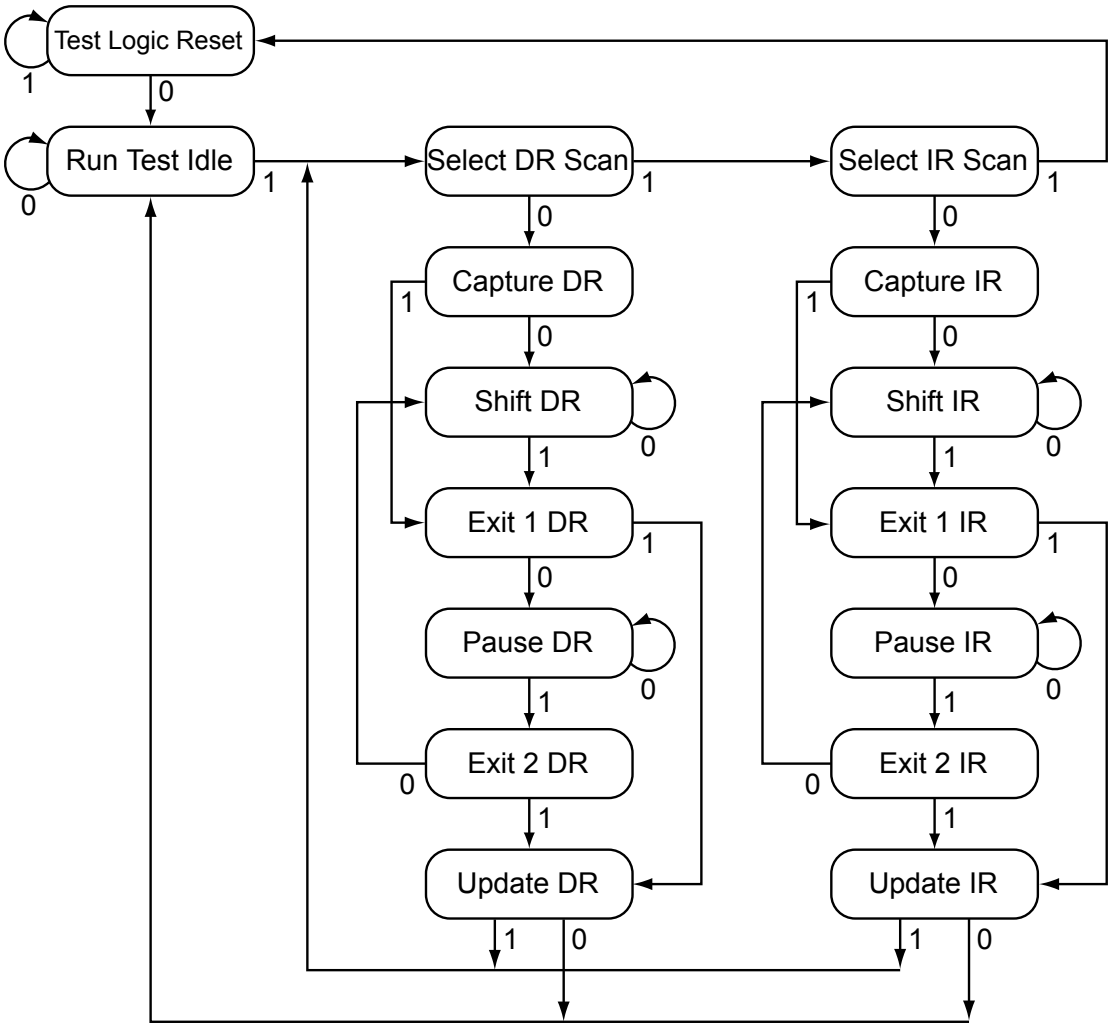
The TDO pin provides an output stream of serial information from the IR chain or the DR chains. The value of TDO depends on the current TAP state, the current instruction, and the data in the chain being accessed. In order to save power when the JTAG port is not being used, the TDO pin is placed in an inactive drive state when not actively shifting out data. Because TDO can be connected to the TDI of another controller in a daisy-chain configuration, the *IEEE Standard 1149.1* expects the value on TDO to change on the falling edge of TCK.

By default, the internal pull-up resistor on the TDO pin is enabled after reset. This assures that the pin remains at a constant logic level when the JTAG port is not being used. The internal pull-up and pull-down resistors can be turned off to save internal power if a High-Z output value is acceptable during certain TAP controller states.

### 4.2.2 JTAG TAP Controller

The JTAG TAP controller state machine is shown in Figure 4-2 on page 143. The TAP controller state machine is reset to the Test-Logic-Reset state on the assertion of a Power-On-Reset (POR) or the assertion of TRST. Asserting the correct sequence on the TMS pin allows the JTAG module to shift in new instructions, shift in data, or idle during extended testing sequences. For detailed information on the function of the TAP controller and the operations that occur in each state, please refer to *IEEE Standard 1149.1*.

**Figure 4-2. Test Access Port State Machine**




### 4.2.3 Shift Registers

The Shift Registers consist of a serial shift register chain and a parallel load register. The serial shift register chain samples specific information during the TAP controller's CAPTURE states and allows this information to be shifted out of TDO during the TAP controller's SHIFT states. While the sampled data is being shifted out of the chain on TDO, new data is being shifted into the serial shift register on TDI. This new data is stored in the parallel load register during the TAP controller's UPDATE states. Each of the shift registers is discussed in detail in "Register Descriptions" on page 145.

### 4.2.4 Operational Considerations

There are certain operational considerations when using the JTAG module. Because the JTAG pins can be programmed to be GPIOs, board configuration and reset conditions on these pins must be considered. In addition, because the JTAG module has integrated ARM Serial Wire Debug, the method for switching between these two operational modes is described below.

#### 4.2.4.1 GPIO Functionality

When the microcontroller is reset with either a POR or RST, the JTAG port pins default to their JTAG configurations. The default configuration includes enabling the pull-up resistors (setting **GPIOPUR** to 1 for PB7 and PC[3:0]) and enabling the alternate hardware function (setting **GPIOAFSEL** to 1 for PB7 and PC[3:0]) on the JTAG pins.

It is possible for software to configure these pins as GPIOs after reset by writing 0s to PB7 and PC[3:0] in the **GPIOAFSEL** register. If the user does not require the JTAG port for debugging or board-level testing, this provides five more GPIOs for use in the design.

---

**Caution – If the JTAG pins are used as GPIOs in a design, PB7 and PC2 cannot have external pull-down resistors connected to both of them at the same time. If both pins are pulled Low during reset, the controller has unpredictable behavior. If this happens, remove one or both of the pull-down resistors, and apply RST or power-cycle the part.**

**It is possible to create a software sequence that prevents the debugger from connecting to the Stellaris microcontroller. If the program code loaded into flash immediately changes the JTAG pins to their GPIO functionality, the debugger may not have enough time to connect and halt the controller before the JTAG pin functionality switches. This may lock the debugger out of the part. This can be avoided with a software routine that restores JTAG functionality based on an external or software trigger.**

---

#### 4.2.4.2 Communication with JTAG/SWD

Because the debug clock and the system clock can be running at different frequencies, care must be taken to maintain reliable communication with the JTAG/SWD interface. In the Capture-DR state, the result of the previous transaction, if any, is returned, together with a 3-bit ACK response. Software should check the ACK response to see if the previous operation has completed before initiating a new transaction. Alternatively, if the system clock is at least 8 times faster than the debug clock (TCK or SWCLK), the previous operation has enough time to complete and the ACK bits do not have to be checked.

#### 4.2.4.3 ARM Serial Wire Debug (SWD)

In order to seamlessly integrate the ARM Serial Wire Debug (SWD) functionality, a serial-wire debugger must be able to connect to the Cortex-M3 core without having to perform, or have any knowledge of, JTAG cycles. This is accomplished with a SWD preamble that is issued before the SWD session begins.

The switching preamble used to enable the SWD interface of the SWJ-DP module starts with the TAP controller in the Test-Logic-Reset state. From here, the preamble sequences the TAP controller through the following states: Run Test Idle, Select DR, Select IR, Capture IR, Exit1 IR, Update IR, Run Test Idle, Select DR, Select IR, Capture IR, Exit1 IR, Update IR, Run Test Idle, Select DR, Select IR, and Test-Logic-Reset states.

Stepping through the JTAG TAP Instruction Register (IR) load sequences of the TAP state machine twice without shifting in a new instruction enables the SWD interface and disables the JTAG interface. For more information on this operation and the SWD interface, see the *ARM® Debug Interface V5 Architecture Specification*.

Because this sequence is a valid series of JTAG operations that could be issued, the ARM JTAG TAP controller is not fully compliant to the *IEEE Standard 1149.1*. This is the only instance where the ARM JTAG TAP controller does not meet full compliance with the specification. Due to the low probability of this sequence occurring during normal operation of the TAP controller, it should not affect normal performance of the JTAG interface.

## 4.3 Initialization and Configuration

After a Power-On-Reset or an external reset ($\overline{\text{RST}}$), the JTAG pins are automatically configured for JTAG communication. No user-defined initialization or configuration is needed. However, if the user application changes these pins to their GPIO function, they must be configured back to their JTAG functionality before JTAG communication can be restored. This is done by enabling the five JTAG pins (PB7 and PC[3:0]) for their alternate function using the **GPIOAFSEL** register. In addition to enabling the alternate functions, any other changes to the GPIO pad configurations on the five JTAG pins (PB7 andPC[3:0]) should be reverted to their default settings.

## 4.4 Register Descriptions

There are no APB-accessible registers in the JTAG TAP Controller or Shift Register chains. The registers within the JTAG controller are all accessed serially through the TAP Controller. The registers can be broken down into two main categories: Instruction Registers and Data Registers.

### 4.4.1 Instruction Register (IR)

The JTAG TAP Instruction Register (IR) is a four-bit serial scan chain connected between the JTAG TDI and TDO pins with a parallel load register. When the TAP Controller is placed in the correct states, bits can be shifted into the Instruction Register. Once these bits have been shifted into the chain and updated, they are interpreted as the current instruction. The decode of the Instruction Register bits is shown in Table 4-2 on page 145. A detailed explanation of each instruction, along with its associated Data Register, follows.

**Table 4-2. JTAG Instruction Register Commands**

| IR[3:0] | Instruction | Description |
|---|---|---|
| 0000 | EXTEST | Drives the values preloaded into the Boundary Scan Chain by the SAMPLE/PRELOAD instruction onto the pads. |
| 0001 | INTEST | Drives the values preloaded into the Boundary Scan Chain by the SAMPLE/PRELOAD instruction into the controller. |
| 0010 | SAMPLE / PRELOAD | Captures the current I/O values and shifts the sampled values out of the Boundary Scan Chain while new preload data is shifted in. |
| 1000 | ABORT | Shifts data into the ARM Debug Port Abort Register. |
| 1010 | DPACC | Shifts data into and out of the ARM DP Access Register. |
| 1011 | APACC | Shifts data into and out of the ARM AC Access Register. |
| 1110 | IDCODE | Loads manufacturing information defined by the *IEEE Standard 1149.1* into the IDCODE chain and shifts it out. |
| 1111 | BYPASS | Connects TDI to TDO through a single Shift Register chain. |
| All Others | Reserved | Defaults to the BYPASS instruction to ensure that TDI is always connected to TDO. |

#### 4.4.1.1 EXTEST Instruction

The EXTEST instruction is not associated with its own Data Register chain. The EXTEST instruction uses the data that has been preloaded into the Boundary Scan Data Register using the SAMPLE/PRELOAD instruction. When the EXTEST instruction is present in the Instruction Register, the preloaded data in the Boundary Scan Data Register associated with the outputs and output enables are used to drive the GPIO pads rather than the signals coming from the core. This allows tests to be developed that drive known values out of the controller, which can be used to verify connectivity. While the EXTEST instruction is present in the Instruction Register, the Boundary Scan

Data Register can be accessed to sample and shift out the current data and load new data into the Boundary Scan Data Register.

#### 4.4.1.2 INTEST Instruction

The INTEST instruction is not associated with its own Data Register chain. The INTEST instruction uses the data that has been preloaded into the Boundary Scan Data Register using the SAMPLE/PRELOAD instruction. When the INTEST instruction is present in the Instruction Register, the preloaded data in the Boundary Scan Data Register associated with the inputs are used to drive the signals going into the core rather than the signals coming from the GPIO pads. This allows tests to be developed that drive known values into the controller, which can be used for testing. It is important to note that although the $\overline{\text{RST}}$ input pin is on the Boundary Scan Data Register chain, it is only observable. While the INTEXT instruction is present in the Instruction Register, the Boundary Scan Data Register can be accessed to sample and shift out the current data and load new data into the Boundary Scan Data Register.

#### 4.4.1.3 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction connects the Boundary Scan Data Register chain between TDI and TDO. This instruction samples the current state of the pad pins for observation and preloads new test data. Each GPIO pad has an associated input, output, and output enable signal. When the TAP controller enters the Capture DR state during this instruction, the input, output, and output-enable signals to each of the GPIO pads are captured. These samples are serially shifted out of TDO while the TAP controller is in the Shift DR state and can be used for observation or comparison in various tests.

While these samples of the inputs, outputs, and output enables are being shifted out of the Boundary Scan Data Register, new data is being shifted into the Boundary Scan Data Register from TDI. Once the new data has been shifted into the Boundary Scan Data Register, the data is saved in the parallel load registers when the TAP controller enters the Update DR state. This update of the parallel load register preloads data into the Boundary Scan Data Register that is associated with each input, output, and output enable. This preloaded data can be used with the EXTEST and INTEST instructions to drive data into or out of the controller. Please see “Boundary Scan Data Register” on page 148 for more information.

#### 4.4.1.4 ABORT Instruction

The ABORT instruction connects the associated ABORT Data Register chain between TDI and TDO. This instruction provides read and write access to the ABORT Register of the ARM Debug Access Port (DAP). Shifting the proper data into this Data Register clears various error bits or initiates a DAP abort of a previous request. Please see the “ABORT Data Register” on page 148 for more information.

#### 4.4.1.5 DPACC Instruction

The DPACC instruction connects the associated DPACC Data Register chain between TDI and TDO. This instruction provides read and write access to the DPACC Register of the ARM Debug Access Port (DAP). Shifting the proper data into this register and reading the data output from this register allows read and write access to the ARM debug and status registers. Please see “DPACC Data Register” on page 148 for more information.

#### 4.4.1.6 APACC Instruction

The APACC instruction connects the associated APACC Data Register chain between TDI and TDO. This instruction provides read and write access to the APACC Register of the ARM Debug Access Port (DAP). Shifting the proper data into this register and reading the data output from this

register allows read and write access to internal components and buses through the Debug Port. Please see “APACC Data Register” on page 148 for more information.

#### 4.4.1.7 IDCODE Instruction

The IDCODE instruction connects the associated IDCODE Data Register chain between TDI and TDO. This instruction provides information on the manufacturer, part number, and version of the ARM core. This information can be used by testing equipment and debuggers to automatically configure their input and output data streams. IDCODE is the default instruction that is loaded into the JTAG Instruction Register when a Power-On-Reset (POR) is asserted, TRST is asserted, or the Test-Logic-Reset state is entered. Please see “IDCODE Data Register” on page 147 for more information.

#### 4.4.1.8 BYPASS Instruction

The BYPASS instruction connects the associated BYPASS Data Register chain between TDI and TDO. This instruction is used to create a minimum length serial path between the TDI and TDO ports. The BYPASS Data Register is a single-bit shift register. This instruction improves test efficiency by allowing components that are not needed for a specific test to be bypassed in the JTAG scan chain by loading them with the BYPASS instruction. Please see “BYPASS Data Register” on page 147 for more information.

### 4.4.2 Data Registers

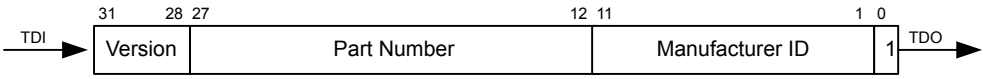
The JTAG module contains six Data Registers. These include: IDCODE, BYPASS, Boundary Scan, APACC, DPACC, and ABORT serial Data Register chains. Each of these Data Registers is discussed in the following sections.

#### 4.4.2.1 IDCODE Data Register

The format for the 32-bit IDCODE Data Register defined by the *IEEE Standard 1149.1* is shown in Figure 4-3 on page 147. The standard requires that every JTAG-compliant device implement either the IDCODE instruction or the BYPASS instruction as the default instruction. The LSB of the IDCODE Data Register is defined to be a 1 to distinguish it from the BYPASS instruction, which has an LSB of 0. This allows auto configuration test tools to determine which instruction is the default instruction.

The major uses of the JTAG port are for manufacturer testing of component assembly, and program development and debug. To facilitate the use of auto-configuration debug tools, the IDCODE instruction outputs a value of 0x1BA0.0477. This allows the debuggers to automatically configure themselves to work correctly with the Cortex-M3 during debug.

**Figure 4-3. IDCODE Register Format**

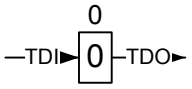



#### 4.4.2.2 BYPASS Data Register

The format for the 1-bit BYPASS Data Register defined by the *IEEE Standard 1149.1* is shown in Figure 4-4 on page 148. The standard requires that every JTAG-compliant device implement either the BYPASS instruction or the IDCODE instruction as the default instruction. The LSB of the BYPASS

Data Register is defined to be a 0 to distinguish it from the IDCODE instruction, which has an LSB of 1. This allows auto configuration test tools to determine which instruction is the default instruction.

**Figure 4-4. BYPASS Register Format**

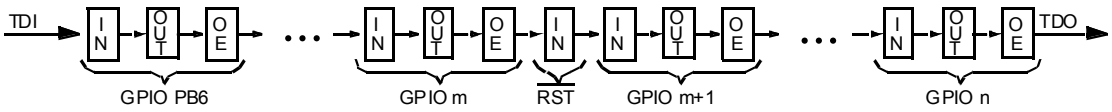



### 4.4.2.3 Boundary Scan Data Register

The format of the Boundary Scan Data Register is shown in Figure 4-5 on page 148. Each GPIO pin, starting with a GPIO pin next to the JTAG port pins, is included in the Boundary Scan Data Register. Each GPIO pin has three associated digital signals that are included in the chain. These signals are input, output, and output enable, and are arranged in that order as can be seen in the figure.

When the Boundary Scan Data Register is accessed with the SAMPLE/PRELOAD instruction, the input, output, and output enable from each digital pad are sampled and then shifted out of the chain to be verified. The sampling of these values occurs on the rising edge of TCK in the Capture DR state of the TAP controller. While the sampled data is being shifted out of the Boundary Scan chain in the Shift DR state of the TAP controller, new data can be preloaded into the chain for use with the EXTEST and INTEST instructions. These instructions either force data out of the controller, with the EXTEST instruction, or into the controller, with the INTEST instruction.

**Figure 4-5. Boundary Scan Register Format**




### 4.4.2.4 APACC Data Register

The format for the 35-bit APACC Data Register defined by ARM is described in the *ARM® Debug Interface V5 Architecture Specification*.

### 4.4.2.5 DPACC Data Register

The format for the 35-bit DPACC Data Register defined by ARM is described in the *ARM® Debug Interface V5 Architecture Specification*.

### 4.4.2.6 ABORT Data Register

The format for the 35-bit ABORT Data Register defined by ARM is described in the *ARM® Debug Interface V5 Architecture Specification*.

# 5 System Control

System control determines the overall operation of the device. It provides information about the device, controls the clocking to the core and individual peripherals, and handles reset detection and reporting.

## 5.1 Functional Description

The System Control module provides the following capabilities:

- Device identification (see “Device Identification” on page 149)
- Local control, such as reset (see “Reset Control” on page 149), power (see “Power Control” on page 154) and clock control (see “Clock Control” on page 154)
- System control (Run, Sleep, and Deep-Sleep modes); see “System Control” on page 157

### 5.1.1 Device Identification

Several read-only registers provide software with information on the microcontroller, such as version, part number, SRAM size, flash size, and other features. See the **DID0**, **DID1**, and **DC0**-**DC4** registers.

### 5.1.2 Reset Control

This section discusses aspects of hardware functions during reset as well as system software requirements following the reset sequence.

#### 5.1.2.1 Reset Sources

The controller has six sources of reset:

1. External reset input pin ($\overline{\text{RST}}$) assertion; see “External $\overline{\text{RST}}$ Pin” on page 150.
2. Power-on reset (POR); see “Power-On Reset (POR)” on page 150.
3. Internal brown-out (BOR) detector; see “Brown-Out Reset (BOR)” on page 152.
4. Software-initiated reset (with the software reset registers); see “Software Reset” on page 153.
5. A watchdog timer reset condition violation; see “Watchdog Timer Reset” on page 153.
6. Internal low drop-out (LDO) regulator output.

Table 5-1 provides a summary of results of the various reset operations.

**Table 5-1. Reset Sources**

| Reset Source | Core Reset? | JTAG Reset? | On-Chip Peripherals Reset? |
|---|---|---|---|
| Power-On Reset | Yes | Yes | Yes |
| $\overline{\text{RST}}$ | Yes | Pin Config Only | Yes |
| Brown-Out Reset | Yes | No | Yes |
| Software System Request Reset[a] | Yes | No | Yes |
| Software Peripheral Reset | No | No | Yes[b] |

**Table 5-1. Reset Sources *(continued)***

| Reset Source | Core Reset? | JTAG Reset? | On-Chip Peripherals Reset? |
|---|---|---|---|
| Watchdog Reset | Yes | No | Yes |
| LDO Reset | Yes | No | Yes |

a. By using the SYSRESREQ bit in the ARM Cortex-M3 **Application Interrupt and Reset Control (APINT)** register

b. Programmable on a module-by-module basis using the Software Reset Control Registers.

After a reset, the **Reset Cause (RESC)** register is set with the reset cause. The bits in this register are sticky and maintain their state across multiple reset sequences,except when an external reset is the cause, and then all the other bits in the **RESC** register are cleared.

**Note:** The main oscillator is used for external resets and power-on resets; the internal oscillator is used during the internal process by internal reset and clock verification circuitry.

### 5.1.2.2 Power-On Reset (POR)

**Note:** The power-on reset also resets the JTAG controller. An external reset does not.

The internal Power-On Reset (POR) circuit monitors the power supply voltage ($V_{DD}$) and generates a reset signal to all of the internal logic including JTAG when the power supply ramp reaches a threshold value ($V_{TH}$). The microcontroller must be operating within the specified operating parameters when the on-chip power-on reset pulse is complete. The 3.3-V power supply to the microcontroller must reach 3.0 V within 10 msec of $V_{DD}$ crossing 2.0 V to guarantee proper operation. For applications that require the use of an external reset signal to hold the microcontroller in reset longer than the internal POR, the $\overline{RST}$ input may be used as discussed in “External $\overline{RST}$ Pin” on page 150.

The Power-On Reset sequence is as follows:

1. The microcontroller waits for internal POR to go inactive.
2. The internal reset is released and the core loads from memory the initial stack pointer, the initial program counter, and the first instruction designated by the program counter, and then begins execution.

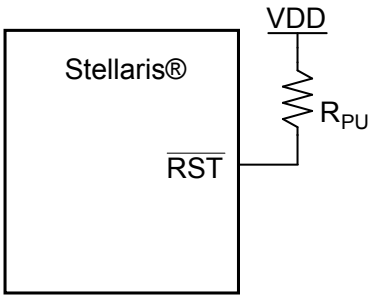
The internal POR is only active on the initial power-up of the microcontroller. The Power-On Reset timing is shown in Figure 17-6 on page 482.

### 5.1.2.3 External $\overline{RST}$ Pin

**Note:** It is recommended that the trace for the $\overline{RST}$ signal must be kept as short as possible. Be sure to place any components connected to the $\overline{RST}$ signal as close to the microcontroller as possible.

If the application only uses the internal POR circuit, the $\overline{RST}$ input must be connected to the power supply ($V_{DD}$) through an optional pull-up resistor (0 to 100K Ω) as shown in Figure 5-1 on page 151.

**Figure 5-1. Basic $\overline{\text{RST}}$ Configuration**



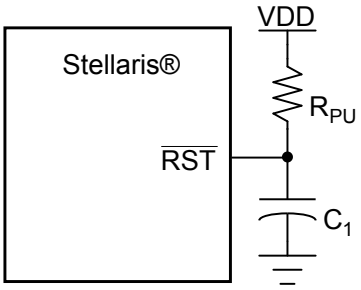

$R_{PU}$ = 0 to 100 kΩ

The external reset pin ($\overline{\text{RST}}$) resets the microcontroller including the core and all the on-chip peripherals except the JTAG TAP controller (see "JTAG Interface" on page 139). The external reset sequence is as follows:

1. The external reset pin ($\overline{\text{RST}}$) is asserted for the duration specified by $T_{MIN}$ and then de-asserted (see "Reset" on page 481).
2. The internal reset is released and the core loads from memory the initial stack pointer, the initial program counter, and the first instruction designated by the program counter, and then begins execution.

To improve noise immunity and/or to delay reset at power up, the $\overline{\text{RST}}$ input may be connected to an RC network as shown in Figure 5-2 on page 151.

**Figure 5-2. External Circuitry to Extend Power-On Reset**

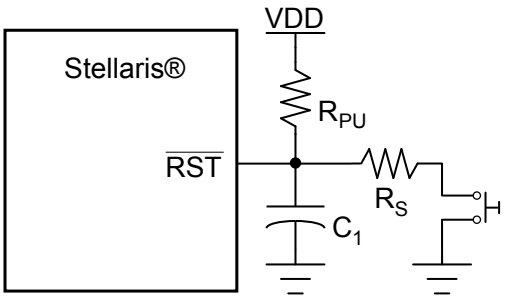



$R_{PU}$ = 1 kΩ to 100 kΩ

$C_1$ = 1 nF to 10 µF

If the application requires the use of an external reset switch, Figure 5-3 on page 152 shows the proper circuitry to use.

**Figure 5-3. Reset Circuit Controlled by Switch**




Typical $R_{PU}$ = 10 kΩ

Typical $R_S$ = 470 Ω

$C_1$ = 10 nF

The $R_{PU}$ and $C_1$ components define the power-on delay.

The external reset timing is shown in Figure 17-5 on page 482.

### 5.1.2.4 Brown-Out Reset (BOR)

A drop in the input voltage resulting in the assertion of the internal brown-out detector can be used to reset the controller. This is initially disabled and may be enabled by software.

The system provides a brown-out detection circuit that triggers if the power supply ($V_{DD}$) drops below a brown-out threshold voltage ($V_{BTH}$). The circuit is provided to guard against improper operation of logic and peripherals that operate off the power supply voltage ($V_{DD}$) and not the LDO voltage. If a brown-out condition is detected, the system may generate a controller interrupt or a system reset. The BOR circuit has a digital filter that protects against noise-related detection for the interrupt condition. This feature may be optionally enabled.

Brown-out resets are controlled with the **Power-On and Brown-Out Reset Control (PBORCTL)** register. The `BORIOR` bit in the **PBORCTL** register must be set for a brown-out condition to trigger a reset.

The brown-out reset sequence is as follows:

1. When $V_{DD}$ drops below $V_{BTH}$, an internal BOR condition is set.
2. If the `BORWT` bit in the **PBORCTL** register is set and `BORIOR` is not set, the BOR condition is resampled, after a delay specified by `BORTIM`, to determine if the original condition was caused by noise. If the BOR condition is not met the second time, then no further action is taken.
3. If the BOR condition exists, an internal reset is asserted.
4. The internal reset is released and the controller fetches and loads the initial stack pointer, the initial program counter, the first instruction designated by the program counter, and begins execution.
5. The internal BOR condition is reset after 500 µs to prevent another BOR condition from being set before software has a chance to investigate the original cause.

The internal Brown-Out Reset timing is shown in Figure 17-7 on page 482.

#### 5.1.2.5 Software Reset

Software can reset a specific peripheral or generate a reset to the entire system .

Peripherals can be individually reset by software via three registers that control reset signals to each peripheral (see the **SRCRn** registers). If the bit position corresponding to a peripheral is set and subsequently cleared, the peripheral is reset. The encoding of the reset registers is consistent with the encoding of the clock gating control for peripherals and on-chip functions (see “System Control” on page 157). Note that all reset signals for all clocks of the specified unit are asserted as a result of a software-initiated reset.

The entire system can be reset by software by setting the SYSRESETREQ bit in the Cortex-M3 Application Interrupt and Reset Control register resets the entire system including the core. The software-initiated system reset sequence is as follows:

1. A software system reset is initiated by writing the SYSRESETREQ bit in the ARM Cortex-M3 Application Interrupt and Reset Control register.
2. An internal reset is asserted.
3. The internal reset is deasserted and the controller loads from memory the initial stack pointer, the initial program counter, and the first instruction designated by the program counter, and then begins execution.

The software-initiated system reset timing is shown in Figure 17-8 on page 483.

#### 5.1.2.6 Watchdog Timer Reset

The watchdog timer module's function is to prevent system hangs. The watchdog timer can be configured to generate an interrupt to the controller on its first time-out, and to generate a reset signal on its second time-out.

After the first time-out event, the 32-bit counter is reloaded with the value of the **Watchdog Timer Load (WDTLOAD)** register, and the timer resumes counting down from that value. If the timer counts down to its zero state again before the first time-out interrupt is cleared, and the reset signal has been enabled, the watchdog timer asserts its reset signal to the system. The watchdog timer reset sequence is as follows:

1. The watchdog timer times out for the second time without being serviced.
2. An internal reset is asserted.
3. The internal reset is released and the controller loads from memory the initial stack pointer, the initial program counter, the first instruction designated by the program counter, and begins execution.

The watchdog reset timing is shown in Figure 17-9 on page 483.

#### 5.1.2.7 Low Drop-Out (LDO)

A reset can be initiated when the internal low drop-out (LDO) regulator output goes unregulated. This is initially disabled and may be enabled by software. LDO is controlled with the **LDO Power Control (LDOPCTL)** register. The LDO reset sequence is as follows:

1. LDO goes unregulated and the LDOARST bit in the **LDOARST** register is set.
2. An internal reset is asserted.

3. The internal reset is released and the controller fetches and loads the initial stack pointer, the initial program counter, the first instruction designated by the program counter, and begins execution.

The LDO reset timing is shown in Figure 17-10 on page 483.

### 5.1.3 Power Control

The Stellaris® microcontroller provides an integrated LDO regulator that is used to provide power to the majority of the controller's internal logic. For power reduction, the LDO regulator provides software a mechanism to adjust the regulated value, in small increments (VSTEP), over the range of 2.25 V to 2.75 V (inclusive)—or 2.5 V ± 10%. The adjustment is made by changing the value of the VADJ field in the **LDO Power Control (LDOPCTL)** register.

### 5.1.4 Clock Control

System control determines the control of clocks in this part.

#### 5.1.4.1 Fundamental Clock Sources

There are multiple clock sources for use in the device:

- **Internal Oscillator (IOSC).** The internal oscillator is an on-chip clock source. It does not require the use of any external components. The frequency of the internal oscillator is 12 MHz ± 30%.
- **Main Oscillator (MOSC).** The main oscillator provides a frequency-accurate clock source by one of two means: an external single-ended clock source is connected to the OSC0 input pin, or an external crystal is connected across the OSC0 input and OSC1 output pins. The crystal value allowed depends on whether the main oscillator is used as the clock reference source to the PLL. If so, the crystal must be one of the supported frequencies between 3.579545 MHz through 8.192 MHz (inclusive). If the PLL is not being used, the crystal may be any one of the supported frequencies between 1 MHz and 8.192 MHz. The single-ended clock source range is from DC through the specified speed of the device. The supported crystals are listed in the XTAL bit field in the **RCC** register (see page 169).

The internal system clock (SysClk), is derived from any of the above sources plus two others: the output of the main internal PLL, and the internal oscillator divided by four (3 MHz ± 30%). The frequency of the PLL clock reference must be in the range of 3.579545 MHz to 8.192 MHz (inclusive). Table 5-2 on page 154 shows how the various clock sources can be used in a system.

**Table 5-2. Clock Source Options**

| Clock Source | Drive PLL? | | Used as SysClk? | |
|---|---|---|---|---|
| Internal Oscillator (12 MHz) | Yes | BYPASS = 0, OSCSRC = 0x1 | Yes | BYPASS = 1, OSCSRC = 0x1 |
| Internal Oscillator divide by 4 (3 MHz) | Yes | BYPASS = 0, OSCSRC = 0x2 | Yes | BYPASS = 1, OSCSRC = 0x2 |
| Main Oscillator | Yes | BYPASS = 0, OSCSRC = 0x0 | Yes | BYPASS = 1, OSCSRC = 0x0 |

#### 5.1.4.2 Clock Configuration

Nearly all of the control for the clocks is provided by the **Run-Mode Clock Configuration (RCC)** register. This register controls the following clock functionality:

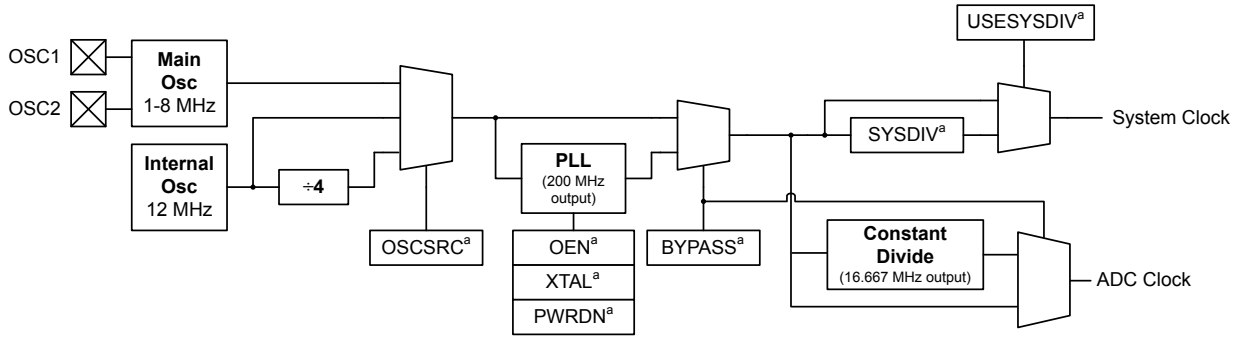
- Source of clocks in sleep and deep-sleep modes

- System clock derived from PLL or other clock source
- Enabling/disabling of oscillators and PLL
- Clock divisors
- Crystal input selection

Figure 5-4 on page 155 shows the logic for the main clock tree. The peripheral blocks are driven by the system clock signal and can be individually enabled/disabled. The ADC clock signal is automatically divided down to 16.67 MHz for proper ADC operation.

**Note:** When the ADC module is in operation, the system clock must be at least 16.667 MHz.

**Figure 5-4. Main Clock Tree**




a. These are bit fields within the **Run-Mode Clock Configuration (RCC)** register.

In the **RCC** register, the SYSDIV field specifies which divisor is used to generate the system clock from either the PLL output or the oscillator source (depending on how the BYPASS bit in this register is configured). Table 5-3 shows how the SYSDIV encoding affects the system clock frequency, depending on whether the PLL is used (BYPASS=0) or another clock source is used (BYPASS=1). The divisor is equivalent to the SYSDIV encoding plus 1. For a list of possible clock sources, see Table 5-2 on page 154.

**Table 5-3. Possible System Clock Frequencies Using the SYSDIV Field**

| SYSDIV | Divisor | Frequency (BYPASS=0) | Frequency (BYPASS=1) | StellarisWare Parameter[a] |
|---|---|---|---|---|
| 0x0 | /1 | reserved | Clock source frequency/2 | SYSCTL_SYSDIV_1[b] |
| 0x1 | /2 | reserved | Clock source frequency/2 | SYSCTL_SYSDIV_2 |
| 0x2 | /3 | reserved | Clock source frequency/3 | SYSCTL_SYSDIV_3 |
| 0x3 | /4 | 50 MHz | Clock source frequency/4 | SYSCTL_SYSDIV_4 |
| 0x4 | /5 | 40 MHz | Clock source frequency/5 | SYSCTL_SYSDIV_5 |
| 0x5 | /6 | 33.33 MHz | Clock source frequency/6 | SYSCTL_SYSDIV_6 |
| 0x6 | /7 | 28.57 MHz | Clock source frequency/7 | SYSCTL_SYSDIV_7 |
| 0x7 | /8 | 25 MHz | Clock source frequency/8 | SYSCTL_SYSDIV_8 |
| 0x8 | /9 | 22.22 MHz | Clock source frequency/9 | SYSCTL_SYSDIV_9 |
| 0x9 | /10 | 20 MHz | Clock source frequency/10 | SYSCTL_SYSDIV_10 |
| 0xA | /11 | 18.18 MHz | Clock source frequency/11 | SYSCTL_SYSDIV_11 |

**Table 5-3. Possible System Clock Frequencies Using the SYSDIV Field *(continued)***

| SYSDIV | Divisor | Frequency (BYPASS=0) | Frequency (BYPASS=1) | StellarisWare Parameter[a] |
|---|---|---|---|---|
| 0xB | /12 | 16.67 MHz | Clock source frequency/12 | SYSCTL_SYSDIV_12 |
| 0xC | /13 | 15.38 MHz | Clock source frequency/13 | SYSCTL_SYSDIV_13 |
| 0xD | /14 | 14.29 MHz | Clock source frequency/14 | SYSCTL_SYSDIV_14 |
| 0xE | /15 | 13.33 MHz | Clock source frequency/15 | SYSCTL_SYSDIV_15 |
| 0xF | /16 | 12.5 MHz (default) | Clock source frequency/16 | SYSCTL_SYSDIV_16 |

a. This parameter is used in functions such as SysCtlClockSet() in the Stellaris Peripheral Driver Library.

b. SYSCTL_SYSDIV_1 does not set the USESYSDIV bit. As a result, using this parameter without enabling the PLL results in the system clock having the same frequency as the clock source.

#### 5.1.4.3 Crystal Configuration for the Main Oscillator (MOSC)

The main oscillator supports the use of a select number of crystals. If the main oscillator is used by the PLL as a reference clock, the supported range of crystals is 3.579545 to 8.192 MHz, otherwise, the range of supported crystals is 1 to 8.192 MHz.

The XTAL bit in the **RCC** register (see page 169) describes the available crystal choices and default programming values.

Software configures the **RCC** register XTAL field with the crystal number. If the PLL is used in the design, the XTAL field value is internally translated to the PLL settings.

#### 5.1.4.4 Main PLL Frequency Configuration

The main PLL is disabled by default during power-on reset and is enabled later by software if required. Software configures the main PLL input reference clock source, specifies the output divisor to set the system clock frequency, and enables the main PLL to drive the output.

If the main oscillator provides the clock reference to the main PLL, the translation provided by hardware and used to program the PLL is available for software in the **XTAL to PLL Translation (PLLCFG)** register (see page 172). The internal translation provides a translation within ± 1% of the targeted PLL VCO frequency.

The Crystal Value field (XTAL) in the **Run-Mode Clock Configuration (RCC)** register (see page 169) describes the available crystal choices and default programming of the **PLLCFG** register. Any time the XTAL field changes, the new settings are translated and the internal PLL settings are updated.

#### 5.1.4.5 PLL Modes

The PLL has two modes of operation: Normal and Power-Down

- Normal: The PLL multiplies the input clock reference and drives the output.
- Power-Down: Most of the PLL internal circuitry is disabled and the PLL does not drive the output.

The modes are programmed using the **RCC** register fields (see page 169).

#### 5.1.4.6 PLL Operation

If a PLL configuration is changed, the PLL output frequency is unstable until it reconverges (relocks) to the new setting. The time between the configuration change and relock is $T_{READY}$ (see Table 17-7 on page 479). During the relock time, the affected PLL is not usable as a clock reference.

PLL is changed by one of the following:

- Change to the XTAL value in the **RCC** register—writes of the same value do not cause a relock.
- Change in the PLL from Power-Down to Normal mode.

A counter is defined to measure the $T_{READY}$ requirement. The counter is clocked by the main oscillator. The range of the main oscillator has been taken into account and the down counter is set to 0x1200 (that is, ~600 µs at an 8.192 MHz external oscillator clock). Hardware is provided to keep the PLL from being used as a system clock until the $T_{READY}$ condition is met after one of the two changes above. It is the user's responsibility to have a stable clock source (like the main oscillator) before the **RCC** register is switched to use the PLL.

If the main PLL is enabled and the system clock is switched to use the PLL in one step, the system control hardware continues to clock the controller from the oscillator selected by the **RCC** register until the main PLL is stable ($T_{READY}$ time met), after which it changes to the PLL. Software can use many methods to ensure that the system is clocked from the main PLL, including periodically polling the PLLLRIS bit in the **Raw Interrupt Status (RIS)** register, and enabling the PLL Lock interrupt.

#### 5.1.4.7 Clock Verification Timers

There are three identical clock verification circuits that can be enabled though software. The circuit checks the faster clock by a slower clock using timers:

- The main oscillator checks the PLL.
- The main oscillator checks the internal oscillator.
- The internal oscillator divided by 64 checks the main oscillator.

If the verification timer function is enabled and a failure is detected, the main clock tree is immediately switched to a working clock and an interrupt is generated to the controller. Software can then determine the course of action to take. The actual failure indication and clock switching does not clear without a write to the **CLKVCLR** register, an external reset, or a POR reset. The clock verification timers are controlled by the PLLVER , IOSCVER , and MOSCVER bits in the **RCC** register.

### 5.1.5 System Control

For power-savings purposes, the **RCGCn** , **SCGCn** , and **DCGCn** registers control the clock gating logic for each peripheral or block in the system while the controller is in Run, Sleep, and Deep-Sleep mode, respectively. The **DC1** , **DC2** and **DC4** registers act as a write mask for the **RCGCn** , **SCGCn**, and **DCGCn** registers.

There are three levels of operation for the device defined as:

- **Run Mode.** In Run mode, the controller actively executes code. Run mode provides normal operation of the processor and all of the peripherals that are currently enabled by the **RCGCn** registers. The system clock can be any of the available clock sources including the PLL.
- **Sleep Mode.** In Sleep mode, the clock frequency of the active peripherals is unchanged, but the processor and the memory subsystem are not clocked and therefore no longer execute code. Sleep mode is entered by the Cortex-M3 core executing a `WFI(Wait for Interrupt)` instruction. Any properly configured interrupt event in the system will bring the processor back into Run mode. See “Power Management” on page 76 for more details.

  Peripherals are clocked that are enabled in the **SCGCn** register when auto-clock gating is enabled (see the **RCC** register) or the **RCGCn** register when the auto-clock gating is disabled. The system clock has the same source and frequency as that during Run mode.

- **Deep-Sleep Mode.** In Deep-Sleep mode, the clock frequency of the active peripherals may change (depending on the Run mode clock configuration) in addition to the processor clock being stopped. An interrupt returns the device to Run mode from one of the sleep modes. Deep-Sleep mode is entered by first writing the Deep Sleep Enable bit in the ARM Cortex-M3 NVIC system control register and then executing a `WFI` instruction. Any properly configured interrupt event in the system will bring the processor back into Run mode. See “Power Management” on page 76 for more details.

  The Cortex-M3 processor core and the memory subsystem are not clocked. Peripherals are clocked that are enabled in the **DCGCn** register when auto-clock gating is enabled (see the **RCC** register) or the **RCGCn** register when auto-clock gating is disabled. The system clock source is the main oscillator by default or the internal oscillator specified in the **DSLPCLKCFG** register if one is enabled. When the **DSLPCLKCFG** register is used, the internal oscillator is powered up, if necessary, and the main oscillator is powered down. If the PLL is running at the time of the WFI instruction, hardware will power the PLL down and override the `SYSDIV` field of the active **RCC** register to be /16 or /64, respectively. When the Deep-Sleep exit event occurs, hardware brings the system clock back to the source and frequency it had at the onset of Deep-Sleep mode before enabling the clocks that had been stopped during the Deep-Sleep duration.

---

**Caution – If the Cortex-M3 Debug Access Port (DAP) has been enabled, and the device wakes from a low power sleep or deep-sleep mode, the core may start executing code before all clocks to peripherals have been restored to their run mode configuration. The DAP is usually enabled by software tools accessing the JTAG or SWD interface when debugging or flash programming. If this condition occurs, a Hard Fault is triggered when software accesses a peripheral with an invalid clock.**

**A software delay loop can be used at the beginning of the interrupt routine that is used to wake up a system from a WFI (Wait For Interrupt) instruction. This stalls the execution of any code that accesses a peripheral register that might cause a fault. This loop can be removed for production software as the DAP is most likely not enabled during normal execution.**

**Because the DAP is disabled by default (power on reset), the user can also power-cycle the device. The DAP is not enabled unless it is enabled through the JTAG or SWD interface.**

---

## 5.2 Initialization and Configuration

The PLL is configured using direct register writes to the **RCC** register. The steps required to successfully change the PLL-based system clock are:

1. Bypass the PLL and system clock divider by setting the `BYPASS` bit and clearing the `USESYS` bit in the **RCC** register. This configures the system to run off a “raw” clock source and allows for the new PLL configuration to be validated before switching the system clock to the PLL.
2. Select the crystal value (`XTAL`) and oscillator source (`OSCSRC`), and clear the `PWRDN` and `OEN` bits in **RCC**. Setting the `XTAL` field automatically pulls valid PLL configuration data for the appropriate crystal, and clearing the `PWRDN` and `OEN` bits powers and enables the PLL and its output.
3. Select the desired system divider (`SYSDIV`) in **RCC** and set the `USESYS` bit in **RCC**. The `SYSDIV` field determines the system frequency for the microcontroller.
4. Wait for the PLL to lock by polling the `PLLLRIS` bit in the **Raw Interrupt Status (RIS)** register.
5. Enable use of the PLL by clearing the `BYPASS` bit in **RCC**.

**Note:** If the BYPASS bit is cleared before the PLL locks, it is possible to render the device unusable.

## 5.3 Register Map

Table 5-4 on page 159 lists the System Control registers, grouped by function. The offset listed is a hexadecimal increment to the register's address, relative to the System Control base address of 0x400F.E000.

**Note:** Spaces in the System Control register space that are not used are reserved for future or internal use. Software should not modify any reserved memory address.

**Table 5-4. System Control Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | DID0 | RO | - | Device Identification 0 | 161 |
| 0x004 | DID1 | RO | - | Device Identification 1 | 176 |
| 0x008 | DC0 | RO | 0x001F.001F | Device Capabilities 0 | 178 |
| 0x010 | DC1 | RO | 0x0001.33BF | Device Capabilities 1 | 179 |
| 0x014 | DC2 | RO | 0x0007.1013 | Device Capabilities 2 | 181 |
| 0x018 | DC3 | RO | 0xBFFF.0000 | Device Capabilities 3 | 183 |
| 0x01C | DC4 | RO | 0x0000.001F | Device Capabilities 4 | 185 |
| 0x030 | PBORCTL | R/W | 0x0000.7FFD | Power-On and Brown-Out Reset Control | 163 |
| 0x034 | LDOPCTL | R/W | 0x0000.0000 | LDO Power Control | 164 |
| 0x040 | SRCR0 | R/W | 0x00000000 | Software Reset Control 0 | 200 |
| 0x044 | SRCR1 | R/W | 0x00000000 | Software Reset Control 1 | 201 |
| 0x048 | SRCR2 | R/W | 0x00000000 | Software Reset Control 2 | 202 |
| 0x050 | RIS | RO | 0x0000.0000 | Raw Interrupt Status | 165 |
| 0x054 | IMC | R/W | 0x0000.0000 | Interrupt Mask Control | 166 |
| 0x058 | MISC | R/W1C | 0x0000.0000 | Masked Interrupt Status and Clear | 167 |
| 0x05C | RESC | R/W | - | Reset Cause | 168 |
| 0x060 | RCC | R/W | 0x0780.3AC0 | Run-Mode Clock Configuration | 169 |
| 0x064 | PLLCFG | RO | - | XTAL to PLL Translation | 172 |
| 0x100 | RCGC0 | R/W | 0x00000040 | Run Mode Clock Gating Control Register 0 | 186 |
| 0x104 | RCGC1 | R/W | 0x00000000 | Run Mode Clock Gating Control Register 1 | 191 |
| 0x108 | RCGC2 | R/W | 0x00000000 | Run Mode Clock Gating Control Register 2 | 197 |
| 0x110 | SCGC0 | R/W | 0x00000040 | Sleep Mode Clock Gating Control Register 0 | 188 |
| 0x114 | SCGC1 | R/W | 0x00000000 | Sleep Mode Clock Gating Control Register 1 | 193 |
| 0x118 | SCGC2 | R/W | 0x00000000 | Sleep Mode Clock Gating Control Register 2 | 198 |
| 0x120 | DCGC0 | R/W | 0x00000040 | Deep Sleep Mode Clock Gating Control Register 0 | 190 |

**Table 5-4. System Control Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x124 | DCGC1 | R/W | 0x00000000 | Deep Sleep Mode Clock Gating Control Register 1 | 195 |
| 0x128 | DCGC2 | R/W | 0x00000000 | Deep Sleep Mode Clock Gating Control Register 2 | 199 |
| 0x144 | DSLPCLKCFG | R/W | 0x0780.0000 | Deep Sleep Clock Configuration | 173 |
| 0x150 | CLKVCLR | R/W | 0x0000.0000 | Clock Verification Clear | 174 |
| 0x160 | LDOARST | R/W | 0x0000.0000 | Allow Unregulated LDO to Reset the Part | 175 |

## 5.4 Register Descriptions

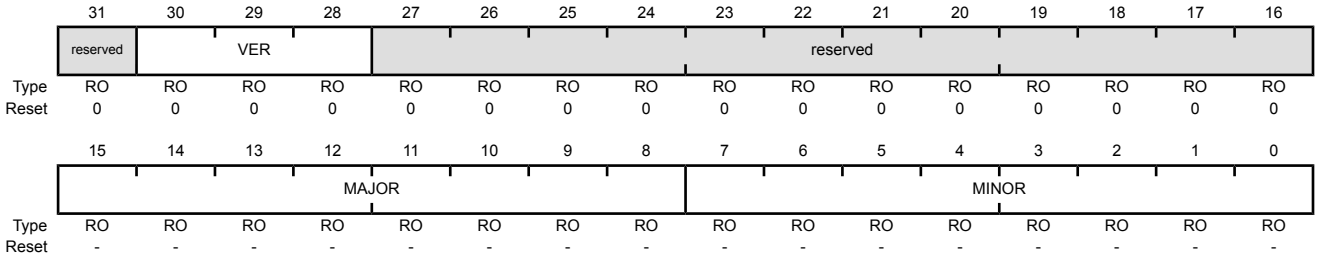
All addresses given are relative to the System Control base address of 0x400F.E000.

## Register 1: Device Identification 0 (DID0), offset 0x000

This register identifies the version of the device.

Device Identification 0 (DID0)

Base 0x400F.E000
Offset 0x000
Type RO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 30:28 | VER | RO | 0x0 | DID0 Version<br>This field defines the **DID0** register format version. The version number is numeric. The value of the VER field is encoded as follows:<br><br>Value Description<br>0x0 Initial **DID0** register format definition for Stellaris® Sandstorm-class devices. |
| 27:16 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:8 | MAJOR | RO | - | Major Revision<br>This field specifies the major revision number of the device. The major revision reflects changes to base layers of the design. The major revision number is indicated in the part number as a letter (A for first revision, B for second, and so on). This field is encoded as follows:<br><br>Value Description<br>0x0 Revision A (initial device)<br>0x1 Revision B (first base layer revision)<br>0x2 Revision C (second base layer revision)<br><br>and so on. |

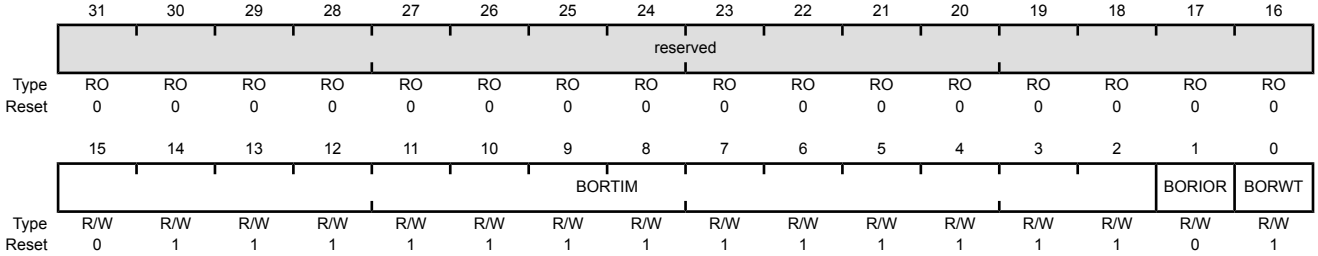
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:0 | MINOR | RO | - | Minor Revision<br>This field specifies the minor revision number of the device. The minor revision reflects changes to the metal layers of the design. The MINOR field value is reset when the MAJOR field is changed. This field is numeric and is encoded as follows:<br><br>Value Description<br>0x0 Initial device, or a major revision update.<br>0x1 First metal layer change.<br>0x2 Second metal layer change.<br><br>and so on. |

## Register 2: Power-On and Brown-Out Reset Control (PBORCTL), offset 0x030

This register is responsible for controlling reset conditions after initial power-on reset.

Power-On and Brown-Out Reset Control (PBORCTL)

Base 0x400F.E000
Offset 0x030
Type R/W, reset 0x0000.7FFD



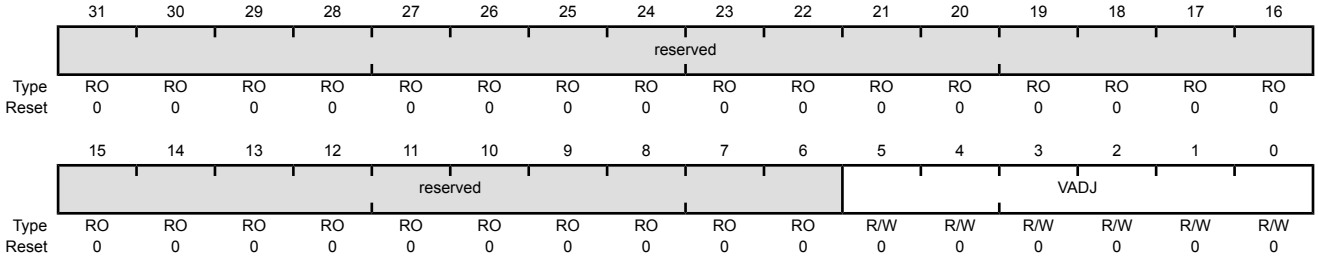

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:2 | BORTIM | R/W | 0x1FFF | BOR Time Delay<br>This field specifies the number of internal oscillator clocks delayed before the BOR output is resampled if the BORWT bit is set.<br>The width of this field is derived by the $t_{BOR}$ width of 500 µs and the internal oscillator (IOSC) frequency of 12 MHz ± 30%. At +30%, the counter value has to exceed 7,800. |
| 1 | BORIOR | R/W | 0 | BOR Interrupt or Reset<br>This bit controls how a BOR event is signaled to the controller. If set, a reset is signaled. Otherwise, an interrupt is signaled. |
| 0 | BORWT | R/W | 1 | BOR Wait and Check for Noise<br>This bit specifies the response to a brown-out signal assertion if BORIOR is not set.<br>If BORWT is set to 1 and BORIOR is cleared to 0, the controller waits BORTIM IOSC periods and resamples the BOR output. If still asserted, a BOR interrupt is signalled. If no longer asserted, the initial assertion is suppressed (attributable to noise).<br>If BORWT is 0, BOR assertions do not resample the output and any condition is reported immediately if enabled. |

## Register 3: LDO Power Control (LDOPCTL), offset 0x034

The VADJ field in this register adjusts the on-chip output voltage ($V_{OUT}$).

LDO Power Control (LDOPCTL)

Base 0x400F.E000
Offset 0x034
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:0 | VADJ | R/W | 0x0 | LDO Output Voltage<br>This field sets the on-chip output voltage. The programming values for the VADJ field are provided below. |

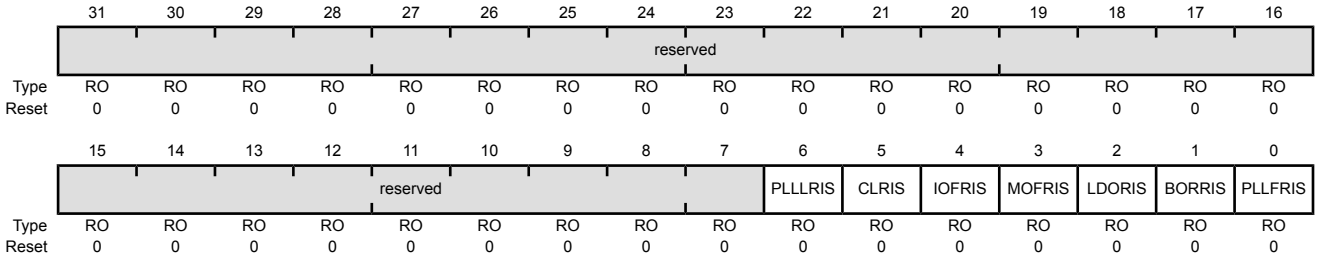
| Value | $V_{OUT}$ (V) |
|---|---|
| 0x00 | 2.50 |
| 0x01 | 2.45 |
| 0x02 | 2.40 |
| 0x03 | 2.35 |
| 0x04 | 2.30 |
| 0x05 | 2.25 |
| 0x06-0x3F | Reserved |
| 0x1B | 2.75 |
| 0x1C | 2.70 |
| 0x1D | 2.65 |
| 0x1E | 2.60 |
| 0x1F | 2.55 |

## Register 4: Raw Interrupt Status (RIS), offset 0x050

Central location for system control raw interrupts. These are set and cleared by hardware.

Raw Interrupt Status (RIS)

Base 0x400F.E000
Offset 0x050
Type RO, reset 0x0000.0000



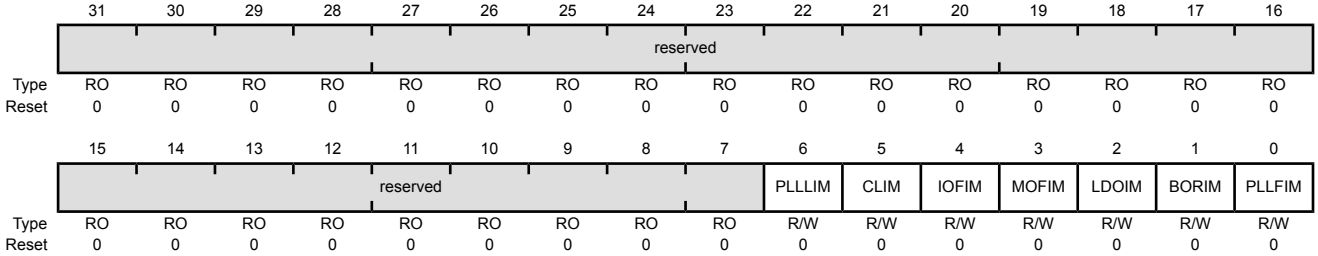

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | PLLLRIS | RO | 0 | PLL Lock Raw Interrupt Status<br>This bit is set when the PLL $T_{READY}$ Timer asserts. |
| 5 | CLRIS | RO | 0 | Current Limit Raw Interrupt Status<br>This bit is set if the LDO's CLE output asserts. |
| 4 | IOFRIS | RO | 0 | Internal Oscillator Fault Raw Interrupt Status<br>This bit is set if an internal oscillator fault is detected. |
| 3 | MOFRIS | RO | 0 | Main Oscillator Fault Raw Interrupt Status<br>This bit is set if a main oscillator fault is detected. |
| 2 | LDORIS | RO | 0 | LDO Power Unregulated Raw Interrupt Status<br>This bit is set if a LDO voltage is unregulated. |
| 1 | BORRIS | RO | 0 | Brown-Out Reset Raw Interrupt Status<br>This bit is the raw interrupt status for any brown-out conditions. If set, a brown-out condition is currently active. This is an unregistered signal from the brown-out detection circuit. An interrupt is reported if the `BORIM` bit in the **IMC** register is set and the `BORIOR` bit in the **PBORCTL** register is cleared. |
| 0 | PLLFRIS | RO | 0 | PLL Fault Raw Interrupt Status<br>This bit is set if a PLL fault is detected (stops oscillating). |

## Register 5: Interrupt Mask Control (IMC), offset 0x054

Central location for system control interrupt masks.

Interrupt Mask Control (IMC)

Base 0x400F.E000
Offset 0x054
Type R/W, reset 0x0000.0000



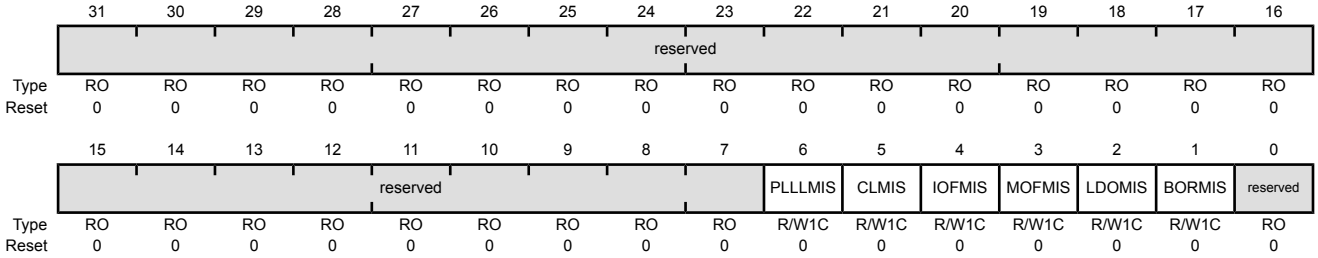

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | PLLLIM | R/W | 0 | PLL Lock Interrupt Mask<br>This bit specifies whether a PLL Lock interrupt is promoted to a controller interrupt. If set, an interrupt is generated if `PLLLRIS` in **RIS** is set; otherwise, an interrupt is not generated. |
| 5 | CLIM | R/W | 0 | Current Limit Interrupt Mask<br>This bit specifies whether a current limit detection is promoted to a controller interrupt. If set, an interrupt is generated if `CLRIS` is set; otherwise, an interrupt is not generated. |
| 4 | IOFIM | R/W | 0 | Internal Oscillator Fault Interrupt Mask<br>This bit specifies whether an internal oscillator fault detection is promoted to a controller interrupt. If set, an interrupt is generated if `IOFRIS` is set; otherwise, an interrupt is not generated. |
| 3 | MOFIM | R/W | 0 | Main Oscillator Fault Interrupt Mask<br>This bit specifies whether a main oscillator fault detection is promoted to a controller interrupt. If set, an interrupt is generated if `MOFRIS` is set; otherwise, an interrupt is not generated. |
| 2 | LDOIM | R/W | 0 | LDO Power Unregulated Interrupt Mask<br>This bit specifies whether an LDO unregulated power situation is promoted to a controller interrupt. If set, an interrupt is generated if `LDORIS` is set; otherwise, an interrupt is not generated. |
| 1 | BORIM | R/W | 0 | Brown-Out Reset Interrupt Mask<br>This bit specifies whether a brown-out condition is promoted to a controller interrupt. If set, an interrupt is generated if `BORRIS` is set; otherwise, an interrupt is not generated. |
| 0 | PLLFIM | R/W | 0 | PLL Fault Interrupt Mask<br>This bit specifies whether a PLL fault detection is promoted to a controller interrupt. If set, an interrupt is generated if `PLLFRIS` is set; otherwise, an interrupt is not generated. |

## Register 6: Masked Interrupt Status and Clear (MISC), offset 0x058

On a read, this register gives the current masked status value of the corresponding interrupt. All of the bits are R/W1C and this action also clears the corresponding raw interrupt bit in the **RIS** register (see page 165).

Masked Interrupt Status and Clear (MISC)

Base 0x400F.E000
Offset 0x058
Type R/W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | PLLLMIS | R/W1C | 0 | PLL Lock Masked Interrupt Status<br>This bit is set when the PLL $T_{READY}$ timer asserts. The interrupt is cleared by writing a 1 to this bit. |
| 5 | CLMIS | R/W1C | 0 | Current Limit Masked Interrupt Status<br>This bit is set if the LDO's CLE output asserts. The interrupt is cleared by writing a 1 to this bit. |
| 4 | IOFMIS | R/W1C | 0 | Internal Oscillator Fault Masked Interrupt Status<br>This bit is set if an internal oscillator fault is detected. The interrupt is cleared by writing a 1 to this bit. |
| 3 | MOFMIS | R/W1C | 0 | Main Oscillator Fault Masked Interrupt Status<br>This bit is set if a main oscillator fault is detected. The interrupt is cleared by writing a 1 to this bit. |
| 2 | LDOMIS | R/W1C | 0 | LDO Power Unregulated Masked Interrupt Status<br>This bit is set if LDO power is unregulated. The interrupt is cleared by writing a 1 to this bit. |
| 1 | BORMIS | R/W1C | 0 | BOR Masked Interrupt Status<br>This bit is the masked interrupt status for any brown-out conditions. If set, a brown-out condition was detected. An interrupt is reported if the `BORIM` bit in the IMC register is set and the `BORIOR` bit in the **PBORCTL** register is cleared. The interrupt is cleared by writing a 1 to this bit. |
| 0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 7: Reset Cause (RESC), offset 0x05C

This field specifies the cause of the reset event to software. The reset value is determined by the cause of the reset. When an external reset is the cause (EXT is set), all other reset bits are cleared. However, if the reset is due to any other cause, the remaining bits are sticky, allowing software to see all causes.

Reset Cause (RESC)

Base 0x400F.E000
Offset 0x05C
Type R/W, reset -

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | LDO | SW | WDT | BOR | POR | EXT |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | - | - | - | - | - | - |

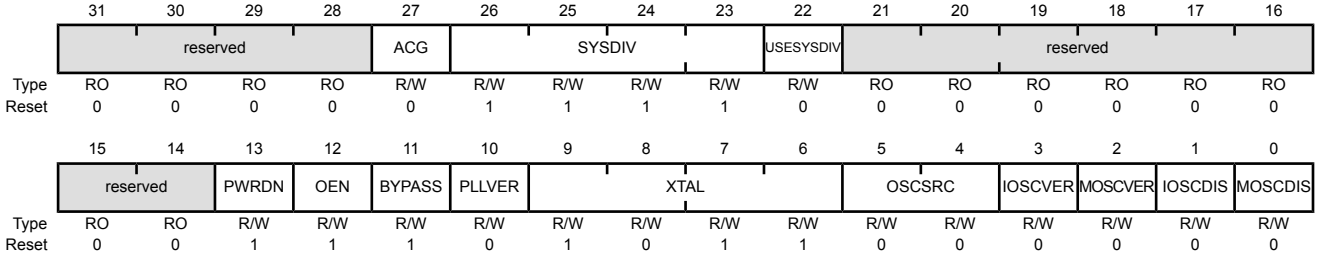
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5 | LDO | R/W | - | LDO Reset<br>When set, indicates the LDO circuit has lost regulation and has generated a reset event. |
| 4 | SW | R/W | - | Software Reset<br>When set, indicates a software reset is the cause of the reset event. |
| 3 | WDT | R/W | - | Watchdog Timer Reset<br>When set, indicates a watchdog reset is the cause of the reset event. |
| 2 | BOR | R/W | - | Brown-Out Reset<br>When set, indicates a brown-out reset is the cause of the reset event. |
| 1 | POR | R/W | - | Power-On Reset<br>When set, indicates a power-on reset is the cause of the reset event. |
| 0 | EXT | R/W | - | External Reset<br>When set, indicates an external reset ($\overline{\text{RST}}$ assertion) is the cause of the reset event. |

## Register 8: Run-Mode Clock Configuration (RCC), offset 0x060

This register is defined to provide source control and frequency speed.

Run-Mode Clock Configuration (RCC)

Base 0x400F.E000
Offset 0x060
Type R/W, reset 0x0780.3AC0




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:28 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 27 | ACG | R/W | 0 | Auto Clock Gating<br>This bit specifies whether the system uses the **Sleep-Mode Clock Gating Control (SCGCn)** registers and **Deep-Sleep-Mode Clock Gating Control (DCGCn)** registers if the controller enters a Sleep or Deep-Sleep mode (respectively). If set, the **SCGCn** or **DCGCn** registers are used to control the clocks distributed to the peripherals when the controller is in a sleep mode. Otherwise, the **Run-Mode Clock Gating Control (RCGCn)** registers are used when the controller enters a sleep mode.<br>The **RCGCn** registers are always used to control the clocks in Run mode.<br>This allows peripherals to consume less power when the controller is in a sleep mode and the peripheral is unused. |
| 26:23 | SYSDIV | R/W | 0xF | System Clock Divisor<br>Specifies which divisor is used to generate the system clock from either the PLL output or the oscillator source (depending on how the BYPASS bit in this register is configured). See Table 5-3 on page 155 for bit encodings.<br>The PLL VCO frequency is 200 MHz.<br>If the SYSDIV value is less than MINSYSDIV (see page 179), and the PLL is being used, then the MINSYSDIV value is used as the divisor.<br>If the PLL is not being used, the SYSDIV value can be less than MINSYSDIV. |
| 22 | USESYSDIV | R/W | 0 | Enable System Clock Divider<br>Use the system clock divider as the source for the system clock. The system clock divider is forced to be used when the PLL is selected as the source.<br>If the USERCC2 bit in the **RCC2** register is set, then the SYSDIV2 field in the **RCC2** register is used as the system clock divider rather than the SYSDIV field in this register. |
| 21:14 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 13 | PWRDN | R/W | 1 | PLL Power Down<br>This bit connects to the PLL PWRDN input. The reset value of 1 powers down the PLL. See Table 5-5 on page 171 for PLL mode control. |
| 12 | OEN | R/W | 1 | PLL Output Enable<br>This bit specifies whether the PLL output driver is enabled. If cleared, the driver transmits the PLL clock to the output. Otherwise, the PLL clock does not oscillate outside the PLL module.<br>**Note:** Both PWRDN and OEN must be cleared to run the PLL. |
| 11 | BYPASS | R/W | 1 | PLL Bypass<br>Chooses whether the system clock is derived from the PLL output or the OSC source. If set, the clock that drives the system is the OSC source. Otherwise, the clock that drives the system is the PLL output clock divided by the system divider.<br>See Table 5-3 on page 155 for programming guidelines.<br>**Note:** The ADC must be clocked from the PLL or directly from a 14-MHz to 18-MHz clock source to operate properly. |
| 10 | PLLVER | R/W | 0 | PLL Verification<br>This bit controls the PLL verification timer function. If set, the verification timer is enabled and an interrupt is generated if the PLL becomes inoperative. Otherwise, the verification timer is not enabled. |
| 9:6 | XTAL | R/W | 0xB | Crystal Value<br>This field specifies the crystal value attached to the main oscillator. The encoding for this field is provided below. |

| Value | Crystal Frequency (MHz) Not Using the PLL | Crystal Frequency (MHz) Using the PLL |
|---|---|---|
| 0x0 | 1.000 | reserved |
| 0x1 | 1.8432 | reserved |
| 0x2 | 2.000 | reserved |
| 0x3 | 2.4576 | reserved |
| 0x4 | 3.579545 MHz | |
| 0x5 | 3.6864 MHz | |
| 0x6 | 4 MHz | |
| 0x7 | 4.096 MHz | |
| 0x8 | 4.9152 MHz | |
| 0x9 | 5 MHz | |
| 0xA | 5.12 MHz | |
| 0xB | 6 MHz (reset value) | |
| 0xC | 6.144 MHz | |
| 0xD | 7.3728 MHz | |
| 0xE | 8 MHz | |
| 0xF | 8.192 MHz | |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 5:4 | OSCSRC | R/W | 0x0 | Oscillator Source<br>Selects the input source for the OSC. The values are:<br><br>Value Input Source<br>0x0 MOSC<br>Main oscillator (default)<br>0x1 IOSC<br>Internal oscillator<br>0x2 IOSC/4<br>Internal oscillator / 4 (this is necessary if used as input to PLL)<br>0x3 reserved |
| 3 | IOSCVER | R/W | 0 | Internal Oscillator Verification Timer<br>This bit controls the internal oscillator verification timer function. If set, the verification timer is enabled and an interrupt is generated if the timer becomes inoperative. Otherwise, the verification timer is not enabled. |
| 2 | MOSCVER | R/W | 0 | Main Oscillator Verification Timer<br>This bit controls the main oscillator verification timer function. If set, the verification timer is enabled and an interrupt is generated if the timer becomes inoperative. Otherwise, the verification timer is not enabled. |
| 1 | IOSCDIS | R/W | 0 | Internal Oscillator Disable<br>0: Internal oscillator (IOSC) is enabled.<br>1: Internal oscillator is disabled. |
| 0 | MOSCDIS | R/W | 0 | Main Oscillator Disable<br>0: Main oscillator is enabled (default).<br>1: Main oscillator is disabled . |

**Table 5-5. PLL Mode Control**

| PWRDN | OEN | Mode |
|---|---|---|
| 1 | X | Power down |
| 0 | 0 | Normal |

## Register 9: XTAL to PLL Translation (PLLCFG), offset 0x064

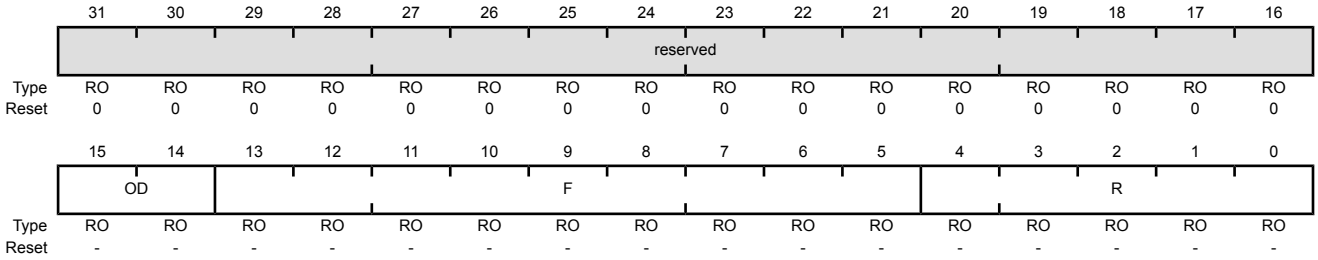
This register provides a means of translating external crystal frequencies into the appropriate PLL settings. This register is initialized during the reset sequence and updated anytime that the XTAL field changes in the **Run-Mode Clock Configuration (RCC)** register (see page 169).

The PLL frequency is calculated using the **PLLCFG** field values, as follows:

```
PLLFreq = OSCFreq * (F + 2) / (R + 2)
```

XTAL to PLL Translation (PLLCFG)

Base 0x400F.E000
Offset 0x064
Type RO, reset -



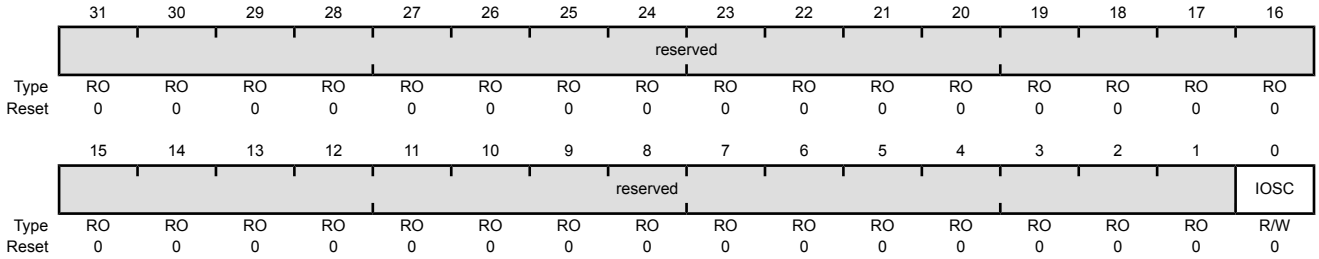

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:14 | OD | RO | - | PLL OD Value<br>This field specifies the value supplied to the PLL's OD input.<br><br>Value Description<br>0x0 Divide by 1<br>0x1 Divide by 2<br>0x2 Divide by 4<br>0x3 Reserved |
| 13:5 | F | RO | - | PLL F Value<br>This field specifies the value supplied to the PLL's F input. |
| 4:0 | R | RO | - | PLL R Value<br>This field specifies the value supplied to the PLL's R input. |

## Register 10: Deep Sleep Clock Configuration (DSLPCLKCFG), offset 0x144

This register is used to automatically switch from the main oscillator to the internal oscillator when entering Deep-Sleep mode. The system clock source is the main oscillator by default. When this register is set, the internal oscillator is powered up and the main oscillator is powered down. When the Deep-Sleep exit event occurs, hardware brings the system clock back to the source and frequency it had at the onset of Deep-Sleep mode.

Deep Sleep Clock Configuration (DSLPCLKCFG)

Base 0x400F.E000
Offset 0x144
Type R/W, reset 0x0780.0000



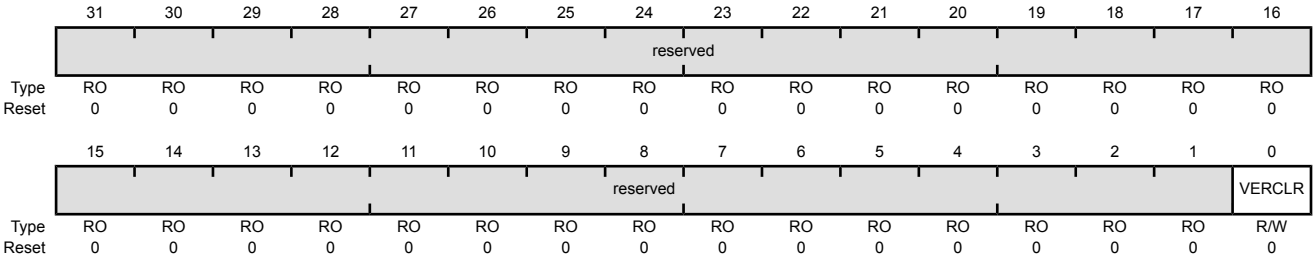

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | IOSC | R/W | 0 | IOSC Clock Source<br>When set, forces IOSC to be clock source during Deep-Sleep (overrides DSOSCSRC field if set) |

## Register 11: Clock Verification Clear (CLKVCLR), offset 0x150

This register is provided as a means of clearing the clock verification circuits by software. Since the clock verification circuits force a known good clock to control the process, the controller is allowed the opportunity to solve the problem and clear the verification fault. This register clears all clock verification faults. To clear a clock verification fault, the VERCLR bit must be set and then cleared by software. This bit is not self-clearing.

Clock Verification Clear (CLKVCLR)

Base 0x400F.E000
Offset 0x150
Type R/W, reset 0x0000.0000



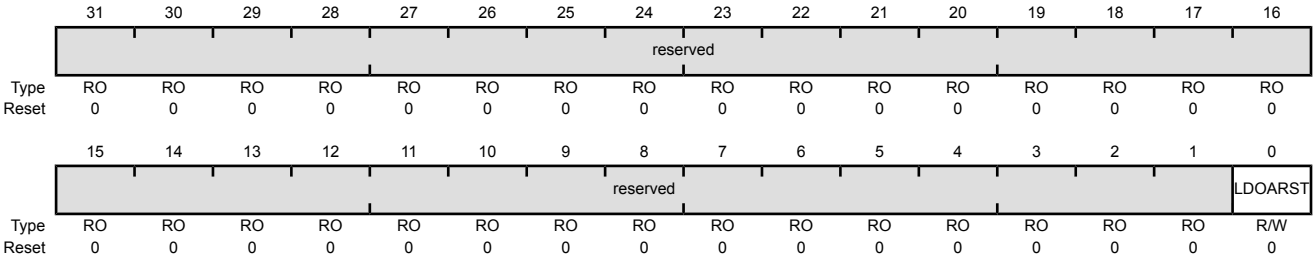

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | VERCLR | R/W | 0 | Clock Verification Clear<br>Clears clock verification faults. |

## Register 12: Allow Unregulated LDO to Reset the Part (LDOARST), offset 0x160

This register is provided as a means of allowing the LDO to reset the part if the voltage goes unregulated. Use this register to choose whether to automatically reset the part if the LDO goes unregulated, based on the design tolerance for LDO fluctuation.

Allow Unregulated LDO to Reset the Part (LDOARST)

Base 0x400F.E000
Offset 0x160
Type R/W, reset 0x0000.0000



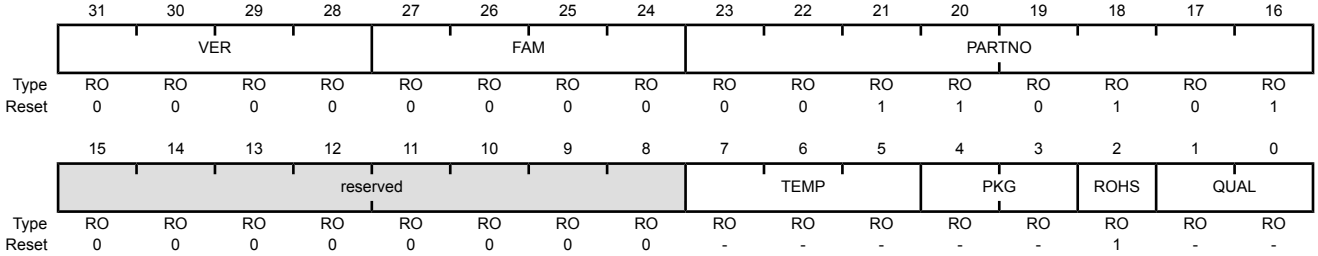

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | LDOARST | R/W | 0 | LDO Reset<br>When set, allows unregulated LDO output to reset the part. |

## Register 13: Device Identification 1 (DID1), offset 0x004

This register identifies the device family, part number, temperature range, and package type.

Device Identification 1 (DID1)

Base 0x400F.E000
Offset 0x004
Type RO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:28 | VER | RO | 0x0 | DID1 Version<br>This field defines the **DID1** register format version. The version number is numeric. The value of the VER field is encoded as follows (all other encodings are reserved):<br><br>Value Description<br>0x0 Initial **DID1** register format definition, indicating a Stellaris LM3Snnn device. |
| 27:24 | FAM | RO | 0x0 | Family<br>This field provides the family identification of the device within the Luminary Micro product portfolio. The value is encoded as follows (all other encodings are reserved):<br><br>Value Description<br>0x0 Stellaris family of microcontollers, that is, all devices with external part numbers starting with LM3S. |
| 23:16 | PARTNO | RO | 0x35 | Part Number<br>This field provides the part number of the device within the family. The value is encoded as follows (all other encodings are reserved):<br><br>Value Description<br>0x35 LM3S828 |
| 15:8 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

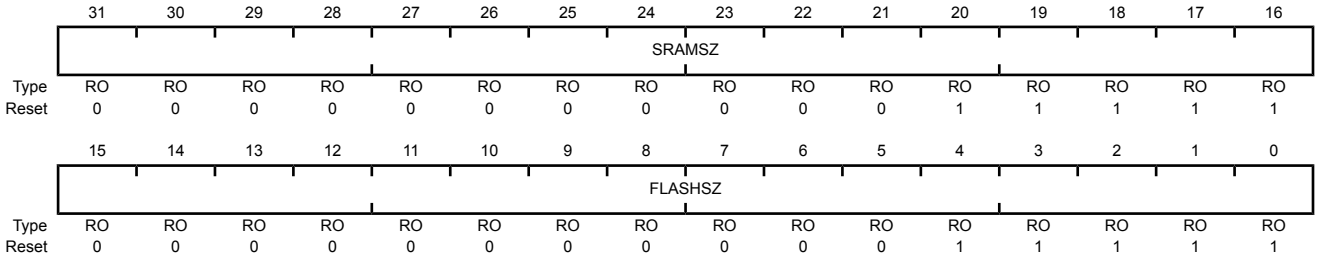
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:5 | TEMP | RO | - | Temperature Range<br>This field specifies the temperature rating of the device. The value is encoded as follows (all other encodings are reserved):<br><br>Value Description<br>0x0 Commercial temperature range (0°C to 70°C)<br>0x1 Industrial temperature range (-40°C to 85°C)<br>0x2 Extended temperature range (-40°C to 105°C) |
| 4:3 | PKG | RO | - | Package Type<br>This field specifies the package type. The value is encoded as follows (all other encodings are reserved):<br><br>Value Description<br>0x0 28-pin SOIC package<br>0x1 48-pin LQFP package<br>0x3 48-pin QFN package |
| 2 | ROHS | RO | 1 | RoHS-Compliance<br>This bit specifies whether the device is RoHS-compliant. A 1 indicates the part is RoHS-compliant. |
| 1:0 | QUAL | RO | - | Qualification Status<br>This field specifies the qualification status of the device. The value is encoded as follows (all other encodings are reserved):<br><br>Value Description<br>0x0 Engineering Sample (unqualified)<br>0x1 Pilot Production (unqualified)<br>0x2 Fully Qualified |

## Register 14: Device Capabilities 0 (DC0), offset 0x008

This register is predefined by the part and can be used to verify features.

Device Capabilities 0 (DC0)

Base 0x400F.E000
Offset 0x008
Type RO, reset 0x001F.001F



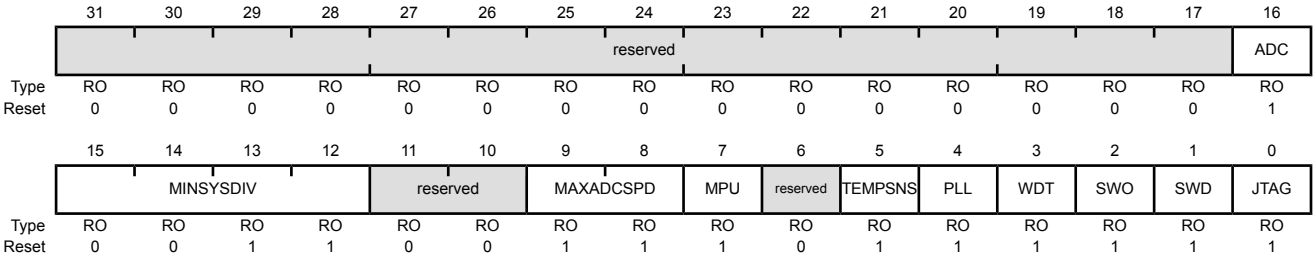

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | SRAMSZ | RO | 0x001F | SRAM Size<br>Indicates the size of the on-chip SRAM memory.<br><br>Value Description<br>0x001F 8 KB of SRAM |
| 15:0 | FLASHSZ | RO | 0x001F | Flash Size<br>Indicates the size of the on-chip flash memory.<br><br>Value Description<br>0x001F 64 KB of Flash |

## Register 15: Device Capabilities 1 (DC1), offset 0x010

This register provides a list of features available in the system. The Stellaris family uses this register format to indicate the availability of the following family features in the specific device: PWM, ADC, Watchdog timer, and debug capabilities. This register also indicates the maximum clock frequency and maximum ADC sample rate. The format of this register is consistent with the **RCGC0**, **SCGC0**, and **DCGC0** clock control registers and the **SRCR0** software reset control register.

Device Capabilities 1 (DC1)

Base 0x400F.E000
Offset 0x010
Type RO, reset 0x0001.33BF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | ADC | RO | 1 | ADC Module Present<br>When set, indicates that the ADC module is present. |
| 15:12 | MINSYSDIV | RO | 0x3 | System Clock Divider<br>Minimum 4-bit divider value for system clock. The reset value is hardware-dependent. See the **RCC** register for how to change the system clock divisor using the SYSDIV bit.<br><br>Value Description<br>0x3 Specifies a 50-MHz CPU clock with a PLL divider of 4. |
| 11:10 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9:8 | MAXADCSPD | RO | 0x3 | Max ADC Speed<br>Indicates the maximum rate at which the ADC samples data.<br><br>Value Description<br>0x3 1M samples/second |
| 7 | MPU | RO | 1 | MPU Present<br>When set, indicates that the Cortex-M3 Memory Protection Unit (MPU) module is present. See the "Cortex-M3 Peripherals" chapter in the Stellaris Data Sheet for details on the MPU. |
| 6 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

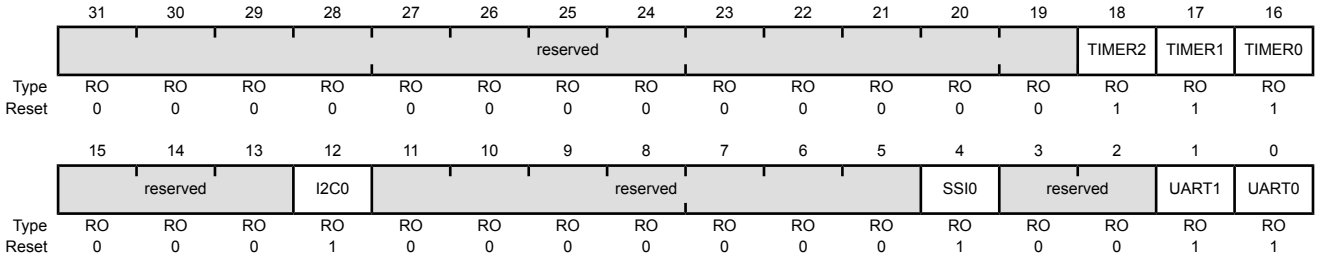
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 5 | TEMPSNS | RO | 1 | Temp Sensor Present<br>When set, indicates that the on-chip temperature sensor is present. |
| 4 | PLL | RO | 1 | PLL Present<br>When set, indicates that the on-chip Phase Locked Loop (PLL) is present. |
| 3 | WDT | RO | 1 | Watchdog Timer Present<br>When set, indicates that a watchdog timer is present. |
| 2 | SWO | RO | 1 | SWO Trace Port Present<br>When set, indicates that the Serial Wire Output (SWO) trace port is present. |
| 1 | SWD | RO | 1 | SWD Present<br>When set, indicates that the Serial Wire Debugger (SWD) is present. |
| 0 | JTAG | RO | 1 | JTAG Present<br>When set, indicates that the JTAG debugger interface is present. |

## Register 16: Device Capabilities 2 (DC2), offset 0x014

This register provides a list of features available in the system. The Stellaris family uses this register format to indicate the availability of the following family features in the specific device: Analog Comparators, General-Purpose Timers, I2Cs, QEIs, SSIs, and UARTs. The format of this register is consistent with the **RCGC1**, **SCGC1**, and **DCGC1** clock control registers and the **SRCR1** software reset control register.

Device Capabilities 2 (DC2)

Base 0x400F.E000
Offset 0x014
Type RO, reset 0x0007.1013




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | TIMER2 | RO | 1 | Timer 2 Present<br>When set, indicates that General-Purpose Timer module 2 is present. |
| 17 | TIMER1 | RO | 1 | Timer 1 Present<br>When set, indicates that General-Purpose Timer module 1 is present. |
| 16 | TIMER0 | RO | 1 | Timer 0 Present<br>When set, indicates that General-Purpose Timer module 0 is present. |
| 15:13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12 | I2C0 | RO | 1 | I2C Module 0 Present<br>When set, indicates that I2C module 0 is present. |
| 11:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SSI0 | RO | 1 | SSI0 Present<br>When set, indicates that SSI module 0 is present. |
| 3:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | UART1 | RO | 1 | UART1 Present<br>When set, indicates that UART module 1 is present. |

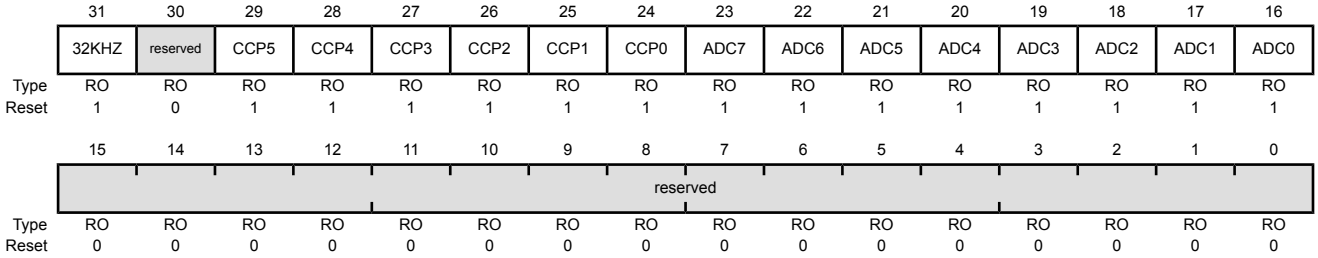
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | UART0 | RO | 1 | UART0 Present<br>When set, indicates that UART module 0 is present. |

## Register 17: Device Capabilities 3 (DC3), offset 0x018

This register provides a list of features available in the system. The Stellaris family uses this register format to indicate the availability of the following family features in the specific device: Analog Comparator I/Os, CCP I/Os, ADC I/Os, and PWM I/Os.

Device Capabilities 3 (DC3)

Base 0x400F.E000
Offset 0x018
Type RO, reset 0xBFFF.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | 32KHZ | RO | 1 | 32KHz Input Clock Available<br>When set, indicates the 32KHz pin or an even CCP pin is present and can be used as a 32-KHz input clock. |
| 30 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 29 | CCP5 | RO | 1 | CCP5 Pin Present<br>When set, indicates that Capture/Compare/PWM pin 5 is present. |
| 28 | CCP4 | RO | 1 | CCP4 Pin Present<br>When set, indicates that Capture/Compare/PWM pin 4 is present. |
| 27 | CCP3 | RO | 1 | CCP3 Pin Present<br>When set, indicates that Capture/Compare/PWM pin 3 is present. |
| 26 | CCP2 | RO | 1 | CCP2 Pin Present<br>When set, indicates that Capture/Compare/PWM pin 2 is present. |
| 25 | CCP1 | RO | 1 | CCP1 Pin Present<br>When set, indicates that Capture/Compare/PWM pin 1 is present. |
| 24 | CCP0 | RO | 1 | CCP0 Pin Present<br>When set, indicates that Capture/Compare/PWM pin 0 is present. |
| 23 | ADC7 | RO | 1 | ADC7 Pin Present<br>When set, indicates that ADC pin 7 is present. |
| 22 | ADC6 | RO | 1 | ADC6 Pin Present<br>When set, indicates that ADC pin 6 is present. |
| 21 | ADC5 | RO | 1 | ADC5 Pin Present<br>When set, indicates that ADC pin 5 is present. |
| 20 | ADC4 | RO | 1 | ADC4 Pin Present<br>When set, indicates that ADC pin 4 is present. |

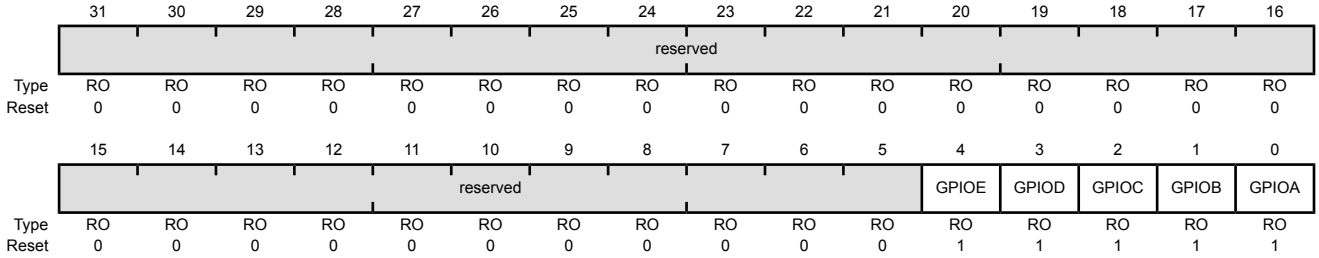
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 19 | ADC3 | RO | 1 | ADC3 Pin Present<br>When set, indicates that ADC pin 3 is present. |
| 18 | ADC2 | RO | 1 | ADC2 Pin Present<br>When set, indicates that ADC pin 2 is present. |
| 17 | ADC1 | RO | 1 | ADC1 Pin Present<br>When set, indicates that ADC pin 1 is present. |
| 16 | ADC0 | RO | 1 | ADC0 Pin Present<br>When set, indicates that ADC pin 0 is present. |
| 15:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 18: Device Capabilities 4 (DC4), offset 0x01C

This register provides a list of features available in the system. The Stellaris family uses this register format to indicate the availability of GPIOs in the specific device. The format of this register is consistent with the **RCGC2**, **SCGC2**, and **DCGC2** clock control registers and the **SRCR2** software reset control register.

Device Capabilities 4 (DC4)

Base 0x400F.E000
Offset 0x01C
Type RO, reset 0x0000.001F



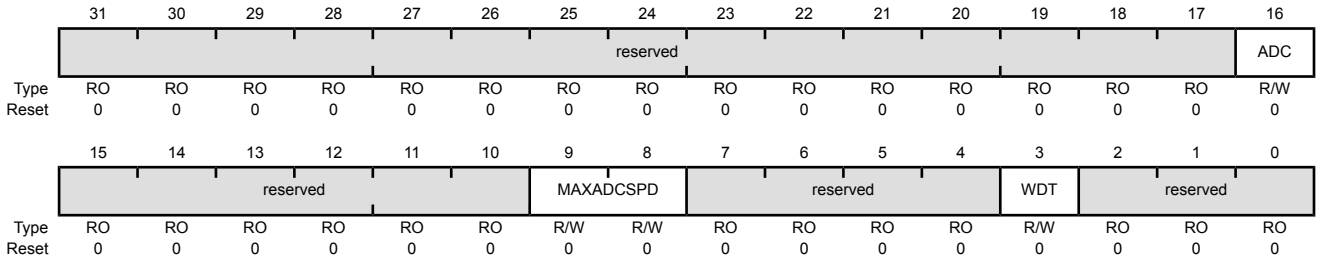

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | GPIOE | RO | 1 | GPIO Port E Present<br>When set, indicates that GPIO Port E is present. |
| 3 | GPIOD | RO | 1 | GPIO Port D Present<br>When set, indicates that GPIO Port D is present. |
| 2 | GPIOC | RO | 1 | GPIO Port C Present<br>When set, indicates that GPIO Port C is present. |
| 1 | GPIOB | RO | 1 | GPIO Port B Present<br>When set, indicates that GPIO Port B is present. |
| 0 | GPIOA | RO | 1 | GPIO Port A Present<br>When set, indicates that GPIO Port A is present. |

## Register 19: Run Mode Clock Gating Control Register 0 (RCGC0), offset 0x100

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC0** is the clock configuration register for running operation, **SCGC0** for Sleep operation, and **DCGC0** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Run Mode Clock Gating Control Register 0 (RCGC0)

Base 0x400F.E000
Offset 0x100
Type R/W, reset 0x00000040




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | ADC | R/W | 0 | ADC0 Clock Gating Control<br>This bit controls the clock gating for SAR ADC module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 15:10 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9:8 | MAXADCSPD | R/W | 0 | ADC Sample Speed<br>This field sets the rate at which the ADC samples data. You cannot set the rate higher than the maximum rate. You can set the sample rate by setting the MAXADCSPD bit as follows:<br><br>Value Description<br>0x3 1M samples/second<br>0x2 500K samples/second<br>0x1 250K samples/second<br>0x0 125K samples/second |

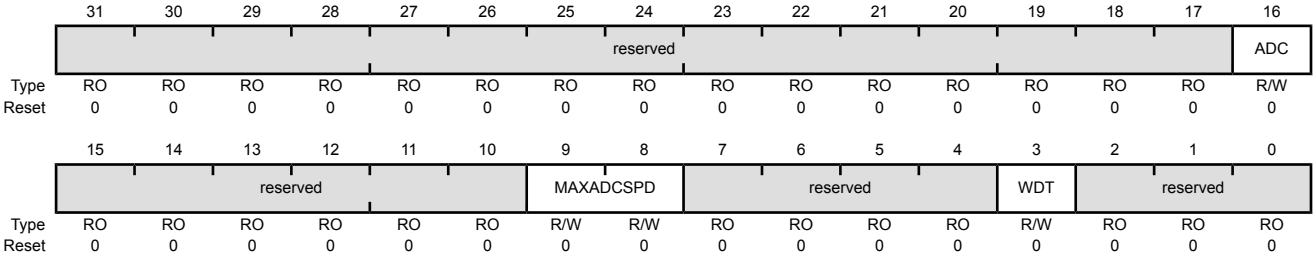
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | WDT | R/W | 0 | WDT Clock Gating Control<br>This bit controls the clock gating for the WDT module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 2:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 20: Sleep Mode Clock Gating Control Register 0 (SCGC0), offset 0x110

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC0** is the clock configuration register for running operation, **SCGC0** for Sleep operation, and **DCGC0** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Sleep Mode Clock Gating Control Register 0 (SCGC0)

Base 0x400F.E000
Offset 0x110
Type R/W, reset 0x00000040




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | ADC | R/W | 0 | ADC0 Clock Gating Control<br>This bit controls the clock gating for SAR ADC module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 15:10 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9:8 | MAXADCSPD | R/W | 0 | ADC Sample Speed<br>This field sets the rate at which the ADC samples data. You cannot set the rate higher than the maximum rate. You can set the sample rate by setting the MAXADCSPD bit as follows:<br><br>Value Description<br>0x3 1M samples/second<br>0x2 500K samples/second<br>0x1 250K samples/second<br>0x0 125K samples/second |

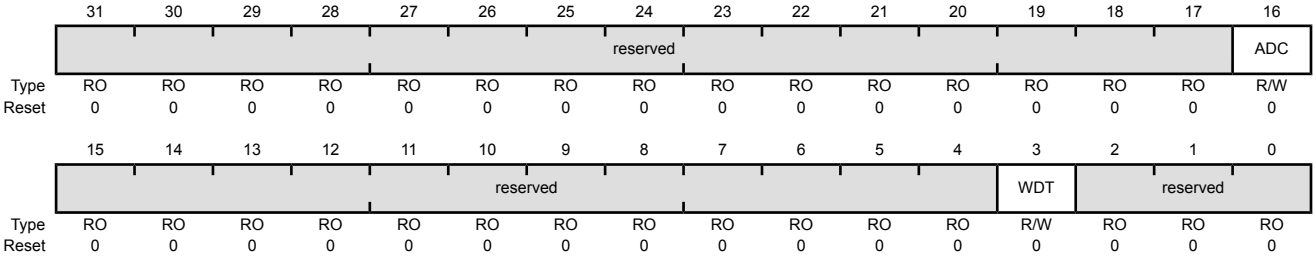
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | WDT | R/W | 0 | WDT Clock Gating Control<br>This bit controls the clock gating for the WDT module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 2:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 21: Deep Sleep Mode Clock Gating Control Register 0 (DCGC0), offset 0x120

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC0** is the clock configuration register for running operation, **SCGC0** for Sleep operation, and **DCGC0** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Deep Sleep Mode Clock Gating Control Register 0 (DCGC0)

Base 0x400F.E000
Offset 0x120
Type R/W, reset 0x00000040



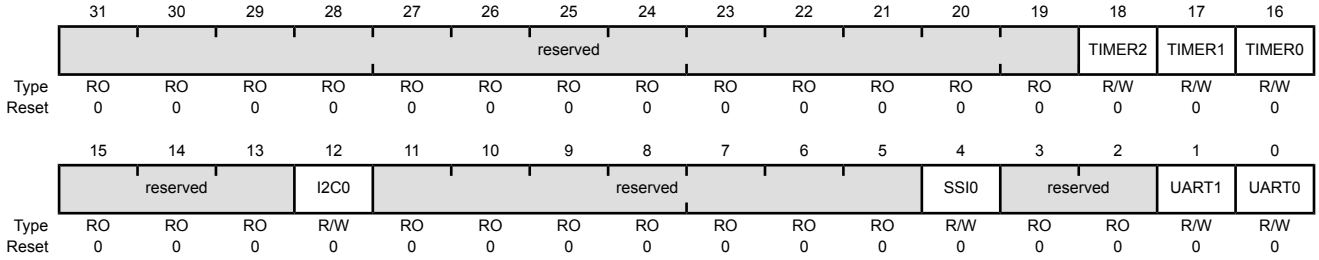

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | ADC | R/W | 0 | ADC0 Clock Gating Control<br>This bit controls the clock gating for SAR ADC module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 15:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | WDT | R/W | 0 | WDT Clock Gating Control<br>This bit controls the clock gating for the WDT module. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, a read or write to the unit generates a bus fault. |
| 2:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 22: Run Mode Clock Gating Control Register 1 (RCGC1), offset 0x104

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC1** is the clock configuration register for running operation, **SCGC1** for Sleep operation, and **DCGC1** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Run Mode Clock Gating Control Register 1 (RCGC1)

Base 0x400F.E000
Offset 0x104
Type R/W, reset 0x00000000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | TIMER2 | R/W | 0 | Timer 2 Clock Gating Control<br>This bit controls the clock gating for General-Purpose Timer module 2. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 17 | TIMER1 | R/W | 0 | Timer 1 Clock Gating Control<br>This bit controls the clock gating for General-Purpose Timer module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 16 | TIMER0 | R/W | 0 | Timer 0 Clock Gating Control<br>This bit controls the clock gating for General-Purpose Timer module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 15:13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

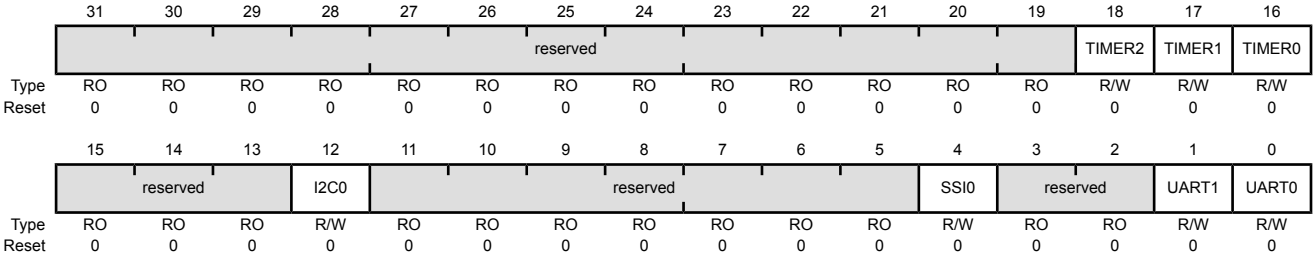
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 12 | I2C0 | R/W | 0 | I2C0 Clock Gating Control<br>This bit controls the clock gating for I2C module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 11:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SSI0 | R/W | 0 | SSI0 Clock Gating Control<br>This bit controls the clock gating for SSI module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 3:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | UART1 | R/W | 0 | UART1 Clock Gating Control<br>This bit controls the clock gating for UART module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | UART0 | R/W | 0 | UART0 Clock Gating Control<br>This bit controls the clock gating for UART module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 23: Sleep Mode Clock Gating Control Register 1 (SCGC1), offset 0x114

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC1** is the clock configuration register for running operation, **SCGC1** for Sleep operation, and **DCGC1** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Sleep Mode Clock Gating Control Register 1 (SCGC1)

Base 0x400F.E000
Offset 0x114
Type R/W, reset 0x00000000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | TIMER2 | R/W | 0 | Timer 2 Clock Gating Control<br>This bit controls the clock gating for General-Purpose Timer module 2. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 17 | TIMER1 | R/W | 0 | Timer 1 Clock Gating Control<br>This bit controls the clock gating for General-Purpose Timer module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 16 | TIMER0 | R/W | 0 | Timer 0 Clock Gating Control<br>This bit controls the clock gating for General-Purpose Timer module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 15:13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 12 | I2C0 | R/W | 0 | I2C0 Clock Gating Control<br>This bit controls the clock gating for I2C module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 11:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SSI0 | R/W | 0 | SSI0 Clock Gating Control<br>This bit controls the clock gating for SSI module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 3:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | UART1 | R/W | 0 | UART1 Clock Gating Control<br>This bit controls the clock gating for UART module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | UART0 | R/W | 0 | UART0 Clock Gating Control<br>This bit controls the clock gating for UART module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 24: Deep Sleep Mode Clock Gating Control Register 1 (DCGC1), offset 0x124

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC1** is the clock configuration register for running operation, **SCGC1** for Sleep operation, and **DCGC1** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Deep Sleep Mode Clock Gating Control Register 1 (DCGC1)

Base 0x400F.E000
Offset 0x124
Type R/W, reset 0x00000000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | I2C0 | reserved | | | | | | | SSI0 | reserved | | UART1 | UART0 |
| Type | RO | RO | RO | R/W | RO | RO | RO | RO | RO | RO | RO | R/W | RO | RO | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | TIMER2 | R/W | 0 | Timer 2 Clock Gating Control<br>This bit controls the clock gating for General-Purpose Timer module 2. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 17 | TIMER1 | R/W | 0 | Timer 1 Clock Gating Control<br>This bit controls the clock gating for General-Purpose Timer module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 16 | TIMER0 | R/W | 0 | Timer 0 Clock Gating Control<br>This bit controls the clock gating for General-Purpose Timer module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 15:13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

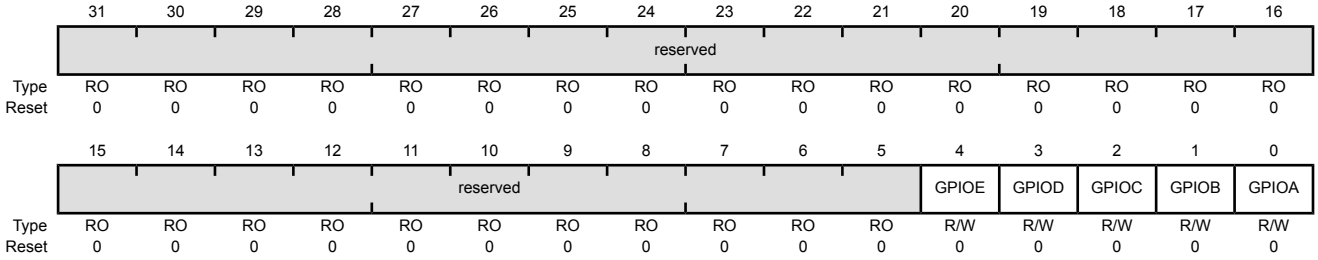
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 12 | I2C0 | R/W | 0 | I2C0 Clock Gating Control<br>This bit controls the clock gating for I2C module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 11:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SSI0 | R/W | 0 | SSI0 Clock Gating Control<br>This bit controls the clock gating for SSI module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 3:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | UART1 | R/W | 0 | UART1 Clock Gating Control<br>This bit controls the clock gating for UART module 1. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | UART0 | R/W | 0 | UART0 Clock Gating Control<br>This bit controls the clock gating for UART module 0. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 25: Run Mode Clock Gating Control Register 2 (RCGC2), offset 0x108

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC2** is the clock configuration register for running operation, **SCGC2** for Sleep operation, and **DCGC2** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Run Mode Clock Gating Control Register 2 (RCGC2)

Base 0x400F.E000
Offset 0x108
Type R/W, reset 0x00000000



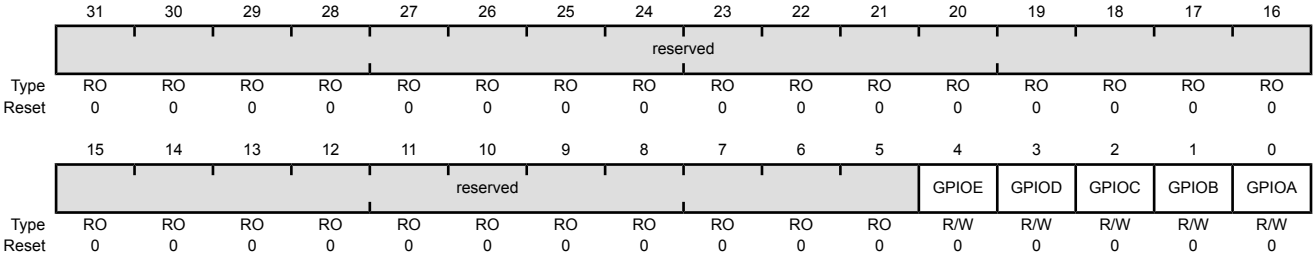

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | GPIOE | R/W | 0 | Port E Clock Gating Control<br>This bit controls the clock gating for Port E. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 3 | GPIOD | R/W | 0 | Port D Clock Gating Control<br>This bit controls the clock gating for Port D. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 2 | GPIOC | R/W | 0 | Port C Clock Gating Control<br>This bit controls the clock gating for Port C. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 1 | GPIOB | R/W | 0 | Port B Clock Gating Control<br>This bit controls the clock gating for Port B. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | GPIOA | R/W | 0 | Port A Clock Gating Control<br>This bit controls the clock gating for Port A. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 26: Sleep Mode Clock Gating Control Register 2 (SCGC2), offset 0x118

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC2** is the clock configuration register for running operation, **SCGC2** for Sleep operation, and **DCGC2** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Sleep Mode Clock Gating Control Register 2 (SCGC2)

Base 0x400F.E000
Offset 0x118
Type R/W, reset 0x00000000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | GPIOE | R/W | 0 | Port E Clock Gating Control<br>This bit controls the clock gating for Port E. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 3 | GPIOD | R/W | 0 | Port D Clock Gating Control<br>This bit controls the clock gating for Port D. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 2 | GPIOC | R/W | 0 | Port C Clock Gating Control<br>This bit controls the clock gating for Port C. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 1 | GPIOB | R/W | 0 | Port B Clock Gating Control<br>This bit controls the clock gating for Port B. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | GPIOA | R/W | 0 | Port A Clock Gating Control<br>This bit controls the clock gating for Port A. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 27: Deep Sleep Mode Clock Gating Control Register 2 (DCGC2), offset 0x128

This register controls the clock gating logic. Each bit controls a clock enable for a given interface, function, or unit. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled (saving power). If the unit is unclocked, reads or writes to the unit will generate a bus fault. The reset state of these bits is 0 (unclocked) unless otherwise noted, so that all functional units are disabled. It is the responsibility of software to enable the ports necessary for the application. Note that these registers may contain more bits than there are interfaces, functions, or units to control. This is to assure reasonable code compatibility with other family and future parts. **RCGC2** is the clock configuration register for running operation, **SCGC2** for Sleep operation, and **DCGC2** for Deep-Sleep operation. Setting the ACG bit in the **Run-Mode Clock Configuration (RCC)** register specifies that the system uses sleep modes.

Deep Sleep Mode Clock Gating Control Register 2 (DCGC2)

Base 0x400F.E000
Offset 0x128
Type R/W, reset 0x00000000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

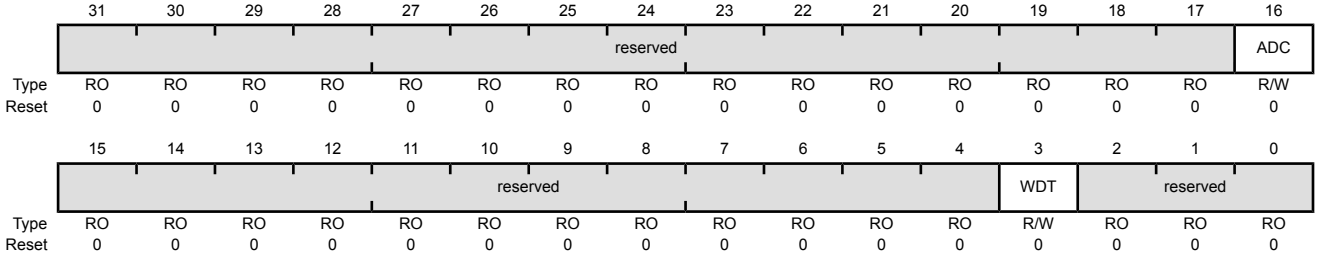
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | GPIOE | R/W | 0 | Port E Clock Gating Control<br>This bit controls the clock gating for Port E. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 3 | GPIOD | R/W | 0 | Port D Clock Gating Control<br>This bit controls the clock gating for Port D. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 2 | GPIOC | R/W | 0 | Port C Clock Gating Control<br>This bit controls the clock gating for Port C. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 1 | GPIOB | R/W | 0 | Port B Clock Gating Control<br>This bit controls the clock gating for Port B. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |
| 0 | GPIOA | R/W | 0 | Port A Clock Gating Control<br>This bit controls the clock gating for Port A. If set, the unit receives a clock and functions. Otherwise, the unit is unclocked and disabled. If the unit is unclocked, reads or writes to the unit will generate a bus fault. |

## Register 28: Software Reset Control 0 (SRCR0), offset 0x040

Writes to this register are masked by the bits in the **Device Capabilities 1 (DC1)** register.

Software Reset Control 0 (SRCR0)

Base 0x400F.E000
Offset 0x040
Type R/W, reset 0x00000000



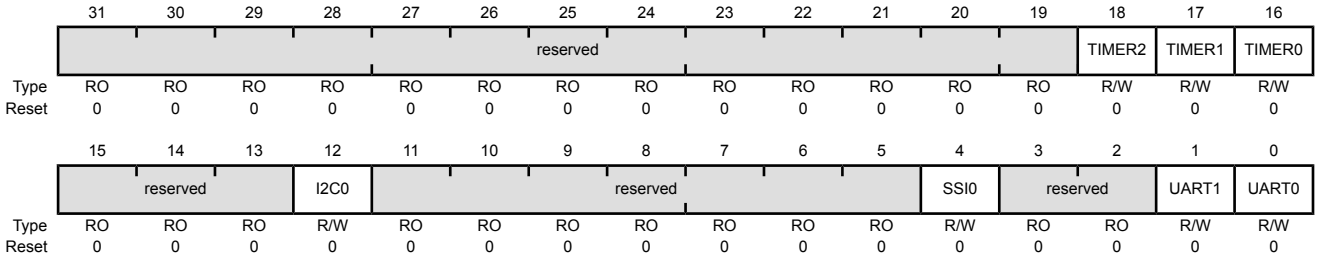

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:17 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 16 | ADC | R/W | 0 | ADC0 Reset Control<br>Reset control for SAR ADC module 0. |
| 15:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | WDT | R/W | 0 | WDT Reset Control<br>Reset control for Watchdog unit. |
| 2:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 29: Software Reset Control 1 (SRCR1), offset 0x044

Writes to this register are masked by the bits in the **Device Capabilities 2 (DC2)** register.

Software Reset Control 1 (SRCR1)

Base 0x400F.E000
Offset 0x044
Type R/W, reset 0x00000000



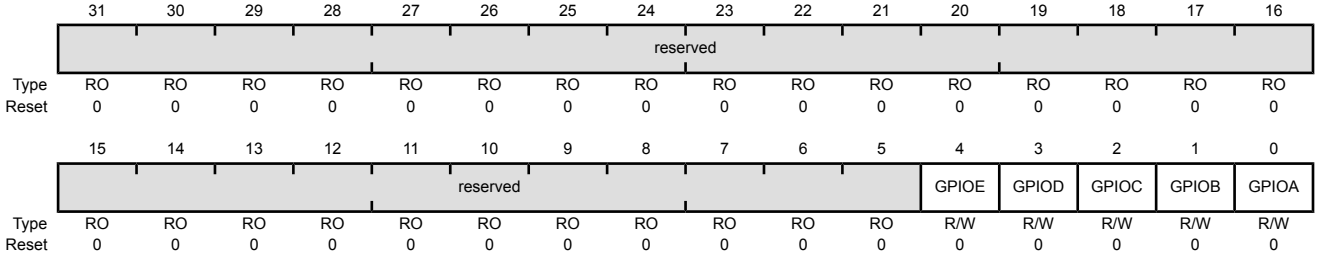

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:19 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 18 | TIMER2 | R/W | 0 | Timer 2 Reset Control<br>Reset control for General-Purpose Timer module 2. |
| 17 | TIMER1 | R/W | 0 | Timer 1 Reset Control<br>Reset control for General-Purpose Timer module 1. |
| 16 | TIMER0 | R/W | 0 | Timer 0 Reset Control<br>Reset control for General-Purpose Timer module 0. |
| 15:13 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12 | I2C0 | R/W | 0 | I2C0 Reset Control<br>Reset control for I2C unit 0. |
| 11:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | SSI0 | R/W | 0 | SSI0 Reset Control<br>Reset control for SSI unit 0. |
| 3:2 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | UART1 | R/W | 0 | UART1 Reset Control<br>Reset control for UART unit 1. |
| 0 | UART0 | R/W | 0 | UART0 Reset Control<br>Reset control for UART unit 0. |

## Register 30: Software Reset Control 2 (SRCR2), offset 0x048

Writes to this register are masked by the bits in the **Device Capabilities 4 (DC4)** register.

Software Reset Control 2 (SRCR2)

Base 0x400F.E000
Offset 0x048
Type R/W, reset 0x00000000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | GPIOE | R/W | 0 | Port E Reset Control<br>Reset control for GPIO Port E. |
| 3 | GPIOD | R/W | 0 | Port D Reset Control<br>Reset control for GPIO Port D. |
| 2 | GPIOC | R/W | 0 | Port C Reset Control<br>Reset control for GPIO Port C. |
| 1 | GPIOB | R/W | 0 | Port B Reset Control<br>Reset control for GPIO Port B. |
| 0 | GPIOA | R/W | 0 | Port A Reset Control<br>Reset control for GPIO Port A. |

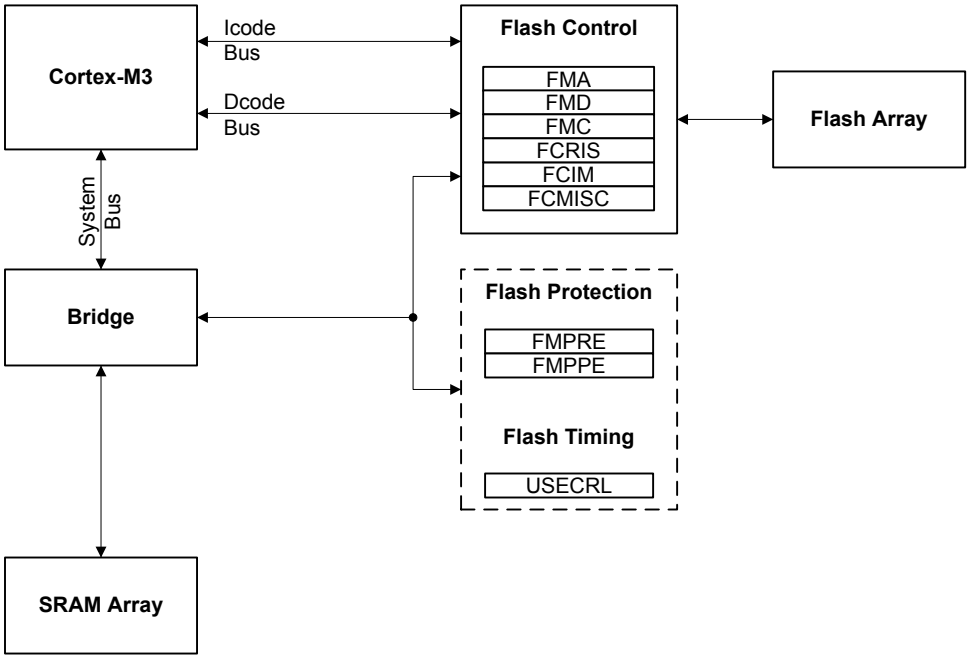
# 6 Internal Memory

The LM3S828 microcontroller comes with 8 KB of bit-banded SRAM and 64 KB of flash memory. The flash controller provides a user-friendly interface, making flash programming a simple task. Flash protection can be applied to the flash memory on a 2-KB block basis.

## 6.1 Block Diagram

Figure 6-1 on page 203 illustrates the Flash functions. The dashed boxes in the figure indicate registers residing in the System Control module rather than the Flash Control module.

**Figure 6-1. Flash Block Diagram**




## 6.2 Functional Description

This section describes the functionality of the SRAM and Flash memories.

### 6.2.1 SRAM Memory

The internal SRAM of the Stellaris® devices is located at address 0x2000.0000 of the device memory map. To reduce the number of time consuming read-modify-write (RMW) operations, ARM has introduced *bit-banding* technology in the Cortex-M3 processor. With a bit-band-enabled processor, certain regions in the memory map (SRAM and peripheral space) can use address aliases to access individual bits in a single, atomic operation.

The bit-band alias is calculated by using the formula:

```
bit-band alias = bit-band base + (byte offset * 32) + (bit number * 4)
```

For example, if bit 3 at address 0x2000.1000 is to be modified, the bit-band alias is calculated as:

```
0x2200.0000 + (0x1000 * 32) + (3 * 4) = 0x2202.000C
```

With the alias address calculated, an instruction performing a read/write to address 0x2202.000C allows direct access to only bit 3 of the byte at address 0x2000.1000.

For details about bit-banding, see “Bit-Banding” on page 62.

### 6.2.2 Flash Memory

The flash is organized as a set of 1-KB blocks that can be individually erased. Erasing a block causes the entire contents of the block to be reset to all 1s. An individual 32-bit word can be programmed to change bits that are currently 1 to a 0. These blocks are paired into a set of 2-KB blocks that can be individually protected. The protection allows blocks to be marked as read-only or execute-only, providing different levels of code protection. Read-only blocks cannot be erased or programmed, protecting the contents of those blocks from being modified. Execute-only blocks cannot be erased or programmed, and can only be read by the controller instruction fetch mechanism, protecting the contents of those blocks from being read by either the controller or by a debugger.

See also “Serial Flash Loader” on page 489 for a preprogrammed flash-resident utility used to download code to the flash memory of a device without the use of a debug interface.

#### 6.2.2.1 Flash Memory Timing

The timing for the flash is automatically handled by the flash controller. However, in order to do so, it must know the clock rate of the system in order to time its internal signals properly. The number of clock cycles per microsecond must be provided to the flash controller for it to accomplish this timing. It is software's responsibility to keep the flash controller updated with this information via the **USec Reload (USECRL)** register.

On reset, the **USECRL** register is loaded with a value that configures the flash timing so that it works with the maximum clock rate of the part. If software changes the system operating frequency, the new operating frequency minus 1 (in MHz) must be loaded into **USECRL** before any flash modifications are attempted. For example, if the device is operating at a speed of 20 MHz, a value of 0x13 (20-1) must be written to the **USECRL** register.

#### 6.2.2.2 Flash Memory Protection

The user is provided two forms of flash protection per 2-KB flash blocks in two 32-bit wide registers.The protection policy for each form is controlled by individual bits (per policy per block) in the **FMPPEn** and **FMPREn** registers.

- **Flash Memory Protection Program Enable (FMPPEn)**: If set, the block may be programmed (written) or erased. If cleared, the block may not be changed.
- **Flash Memory Protection Read Enable (FMPREn)**: If a bit is set, the corresponding block may be executed or read by software or debuggers. If a bit is cleared, the corresponding block may only be executed, and contents of the memory block are prohibited from being read as data.

The policies may be combined as shown in Table 6-1 on page 204.

**Table 6-1. Flash Protection Policy Combinations**

| FMPPEn | FMPREn | Protection |
|---|---|---|
| 0 | 0 | Execute-only protection. The block may only be executed and may not be written or erased. This mode is used to protect code. |
| 1 | 0 | The block may be written, erased or executed, but not read. This combination is unlikely to be used. |

**Table 6-1. Flash Protection Policy Combinations *(continued)***

| FMPPEn | FMPREn | Protection |
|---|---|---|
| 0 | 1 | Read-only protection. The block may be read or executed but may not be written or erased. This mode is used to lock the block from further modification while allowing any read or execute access. |
| 1 | 1 | No protection. The block may be written, erased, executed or read. |

A Flash memory access that attempts to read a read-protected block (**FMPREn** bit is set) is prohibited and generates a bus fault. A Flash memory access that attempts to program or erase a program-protected block (**FMPPEn** bit is set) is prohibited and can optionally generate an interrupt (by setting the AMASK bit in the **Flash Controller Interrupt Mask (FCIM)** register) to alert software developers of poorly behaving software during the development and debug phases.

The factory settings for the **FMPREn** and **FMPPEn** registers are a value of 1 for all implemented banks. These settings create a policy of open access and programmability. The register bits may be changed by clearing the specific register bit. The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. The changes are committed using the **Flash Memory Control (FMC)** register.

#### 6.2.2.3 Interrupts

The Flash memory controller can generate interrupts when the following conditions are observed:

- Programming Interrupt - signals when a program or erase action is complete.
- Access Interrupt - signals when a program or erase action has been attempted on a 2-kB block of memory that is protected by its corresponding **FMPPEn** bit.

The interrupt events that can trigger a controller-level interrupt are defined in the **Flash Controller Masked Interrupt Status (FCMIS)** register (see page 214) by setting the corresponding MASK bits. If interrupts are not used, the raw interrupt status is always visible via the **Flash Controller Raw Interrupt Status (FCRIS)** register (see page 213).

Interrupts are always cleared (for both the **FCMIS** and **FCRIS** registers) by writing a 1 to the corresponding bit in the **Flash Controller Masked Interrupt Status and Clear (FCMISC)** register (see page 215).

#### 6.2.2.4 Flash Memory Protection by Disabling Debug Access

Flash memory may also be protected by permanently disabling access to the Debug Access Port (DAP) through the JTAG and SWD interfaces. Access is disabled by clearing the DBG field of the **FMPRE** register.

If the DBG field in the **Flash Memory Protection Read Enable (FMPRE)** register is programmed to 0x2, access to the DAP is enabled through the JTAG and SWD interfaces. If clear, access to the DAP is disabled. The DBG field programming becomes permanent and irreversible after a commit sequence is performed.

In the initial state provided from the factory, access is enabled in order to facilitate code development and debug. Access to the DAP may be disabled at the end of the manufacturing flow, once all tests have passed and software has been loaded. This change does not take effect until the next power-up of the device. Note that it is recommended that disabling access to the DAP be combined with a mechanism for providing end-user installable updates (if necessary) such as the Stellaris boot loader.

**Important:** Once the DBG field is cleared and committed, this field can never be restored to the factory-programmed value—which means the JTAG/SWD interface to the debug module can never be re-enabled. This sequence does NOT disable the JTAG controller, it only disables the access of the DAP through the JTAG or SWD interfaces. The JTAG interface remains functional and access to the Test Access Port remains enabled, allowing the user to execute the IEEE JTAG-defined instructions (for example, to perform boundary scan operations).

When using the **FMPRE** bits to protect Flash memory from being read as data (to mark sets of 2-KB blocks of Flash memory as execute-only), these one-time-programmable bits should be written at the same time that the debug disable bits are programmed. Mechanisms to execute the one-time code sequence to disable all debug access include:

- Selecting the debug disable option in the Stellaris boot loader
- Loading the debug disable sequence into SRAM and running it once from SRAM after programming the final end application code into Flash memory

## 6.3 Flash Memory Initialization and Configuration

This section shows examples for using the flash controller to perform various operations on the contents of the flash memory.

### 6.3.1 Changing Flash Protection Bits

As discussed in “Flash Memory Protection” on page 204, changes to the protection bits must be committed before they take effect. The sequence below is used change and commit a block protection bit in the **FMPRE** or **FMPPE** registers. The sequence to change and commit a bit in software is as follows:

1. The **Flash Memory Protection Read Enable (FMPRE)** and **Flash Memory Protection Program Enable (FMPPE)** registers are written, changing the intended bit(s). The action of these changes can be tested by software while in this state.
2. The **Flash Memory Address (FMA)** register (see page 209) bit 0 is set to 1 if the **FMPPE** register is to be committed; otherwise, a 0 commits the **FMPRE** register.
3. The **Flash Memory Control (FMC)** register (see page 211) is written with the COMT bit set. This initiates a write sequence and commits the changes.

There is a special sequence to change and commit the DBG bits in the **Flash Memory Protection Read Enable (FMPRE)** register. This sequence also sets and commits any changes from 1 to 0 in the block protection bits (for execute-only) in the **FMPRE** register.

1. The F**lash Memory Protection Read Enable (FMPRE)** register is written, changing the intended bit(s). The action of these changes can be tested by software while in this state.
2. The **Flash Memory Address (FMA)** register (see page 209) is written with a value of 0x900.
3. The **Flash Memory Control (FMC)** register (see page 211) is written with the COMT bit set. This initiates a write sequence and commits the changes.

Below is an example code sequence to permanently disable the JTAG and SWD interface to the debug module using DriverLib:

```
#include "hw_types.h"
#include "hw_flash.h"
void
permanently_disable_jtag_swd(void)
{
     //
     // Clear the DBG field of the FMPRE register. Note that the value
     // used in this instance does not affect the state of the BlockN
     // bits, but were the value different, all bits in the FMPRE are
     // affected by this function!
     //
     HWREG(FLASH_FMPRE) &= 0x3fffffff;
     //
     // The following sequence activates the one-time
     // programming of the FMPRE register.
     //
     HWREG(FLASH_FMA) = 0x900;
     HWREG(FLASH_FMC) = (FLASH_FMC_WRKEY | FLASH_FMC_COMT);
     //
     // Wait until the operation is complete.
     //
     while (HWREG(FLASH_FMC) & FLASH_FMC_COMT)
     {
     }
}
```

### 6.3.2 Flash Programming

The Stellaris devices provide a user-friendly interface for flash programming. All erase/program operations are handled via three registers: **FMA**, **FMD**, and **FMC**.

During a Flash memory operation (write, page erase, or mass erase) access to the Flash memory is inhibited. As a result, instruction and literal fetches are held off until the Flash memory operation is complete. If instruction execution is required during a Flash memory operation, the code that is executing must be placed in SRAM and executed from there while the flash operation is in progress.

#### 6.3.2.1 To program a 32-bit word

1. Write source data to the **FMD** register.
2. Write the target address to the **FMA** register.
3. Write the flash write key and the WRITE bit (a value of 0xA442.0001) to the **FMC** register.
4. Poll the **FMC** register until the WRITE bit is cleared.

#### 6.3.2.2 To perform an erase of a 1-KB page

1. Write the page address to the **FMA** register.
2. Write the flash write key and the ERASE bit (a value of 0xA442.0002) to the **FMC** register.
3. Poll the **FMC** register until the ERASE bit is cleared.

#### 6.3.2.3 To perform a mass erase of the flash

1. Write the flash write key and the MERASE bit (a value of 0xA442.0004) to the **FMC** register.
2. Poll the **FMC** register until the MERASE bit is cleared.

## 6.4 Register Map

Table 6-2 on page 208 lists the Flash memory and control registers. The offset listed is a hexadecimal increment to the register's address. The **FMA**, **FMD**, **FMC**, **FCRIS**, **FCIM**, and **FCMISC** register offsets are relative to the Flash memory control base address of 0x400F.D000. The Flash memory protection register offsets are relative to the System Control base address of 0x400F.E000.

**Table 6-2. Flash Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| **Flash Memory Control Registers (Flash Control Offset)** | | | | | |
| 0x000 | FMA | R/W | 0x0000.0000 | Flash Memory Address | 209 |
| 0x004 | FMD | R/W | 0x0000.0000 | Flash Memory Data | 210 |
| 0x008 | FMC | R/W | 0x0000.0000 | Flash Memory Control | 211 |
| 0x00C | FCRIS | RO | 0x0000.0000 | Flash Controller Raw Interrupt Status | 213 |
| 0x010 | FCIM | R/W | 0x0000.0000 | Flash Controller Interrupt Mask | 214 |
| 0x014 | FCMISC | R/W1C | 0x0000.0000 | Flash Controller Masked Interrupt Status and Clear | 215 |
| **Flash Memory Protection Registers (System Control Offset)** | | | | | |
| 0x130 | FMPRE | R/W | 0xBFFF.FFFF | Flash Memory Protection Read Enable | 218 |
| 0x134 | FMPPE | R/W | 0xFFFF.FFFF | Flash Memory Protection Program Enable | 219 |
| 0x140 | USECRL | R/W | 0x31 | USec Reload | 217 |

## 6.5 Flash Register Descriptions (Flash Control Offset)

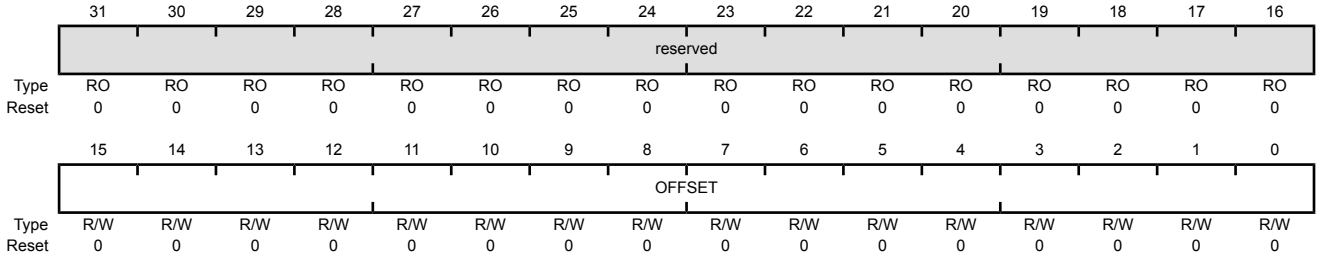
This section lists and describes the Flash Memory registers, in numerical order by address offset. Registers in this section are relative to the Flash control base address of 0x400F.D000.

### Register 1: Flash Memory Address (FMA), offset 0x000

During a write operation, this register contains a 4-byte-aligned address and specifies where the data is written. During erase operations, this register contains a 1 KB-aligned address and specifies which page is erased. Note that the alignment requirements must be met by software or the results of the operation are unpredictable.

Flash Memory Address (FMA)

Base 0x400F.D000
Offset 0x000
Type R/W, reset 0x0000.0000



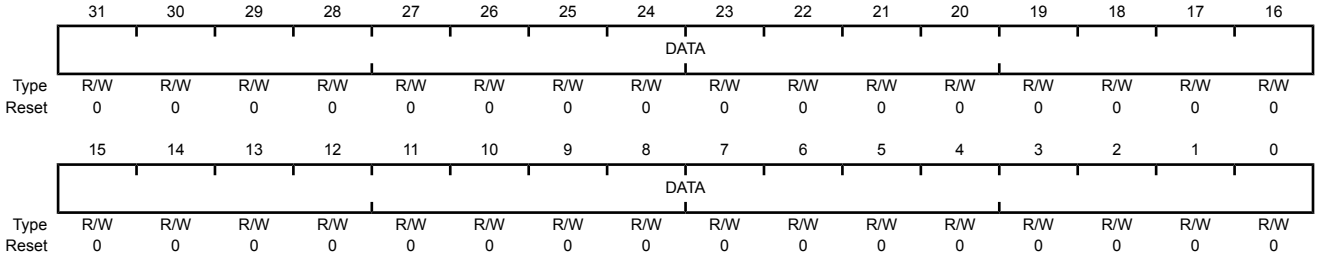

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | OFFSET | R/W | 0x0 | Address Offset<br>Address offset in flash where operation is performed. |

### Register 2: Flash Memory Data (FMD), offset 0x004

This register contains the data to be written during the programming cycle or read during the read cycle. Note that the contents of this register are undefined for a read access of an execute-only block. This register is not used during the erase cycles.

Flash Memory Data (FMD)

Base 0x400F.D000
Offset 0x004
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | DATA | R/W | 0x0 | Data Value<br>Data value for write operation. |

## Register 3: Flash Memory Control (FMC), offset 0x008

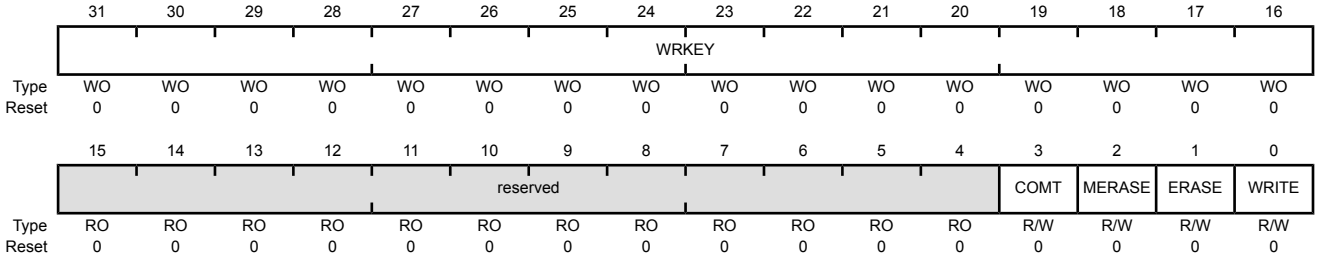
When this register is written, the flash controller initiates the appropriate access cycle for the location specified by the **Flash Memory Address (FMA)** register (see page 209). If the access is a write access, the data contained in the **Flash Memory Data (FMD)** register (see page 210) is written.

This is the final register written and initiates the memory operation. There are four control bits in the lower byte of this register that, when set, initiate the memory operation. The most used of these register bits are the ERASE and WRITE bits.

It is a programming error to write multiple control bits and the results of such an operation are unpredictable.

Flash Memory Control (FMC)

Base 0x400F.D000
Offset 0x008
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | WRKEY | WO | 0x0 | Flash Write Key<br>This field contains a write key, which is used to minimize the incidence of accidental flash writes. The value 0xA442 must be written into this field for a write to occur. Writes to the **FMC** register without this WRKEY value are ignored. A read of this field returns the value 0. |
| 15:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | COMT | R/W | 0 | Commit Register Value<br>Commit (write) of register value to nonvolatile storage. A write of 0 has no effect on the state of this bit.<br>If read, the state of the previous commit access is provided. If the previous commit access is complete, a 0 is returned; otherwise, if the commit access is not complete, a 1 is returned.<br>This can take up to 50 µs. |
| 2 | MERASE | R/W | 0 | Mass Erase Flash Memory<br>If this bit is set, the flash main memory of the device is all erased. A write of 0 has no effect on the state of this bit.<br>If read, the state of the previous mass erase access is provided. If the previous mass erase access is complete, a 0 is returned; otherwise, if the previous mass erase access is not complete, a 1 is returned.<br>This can take up to 250 ms. |

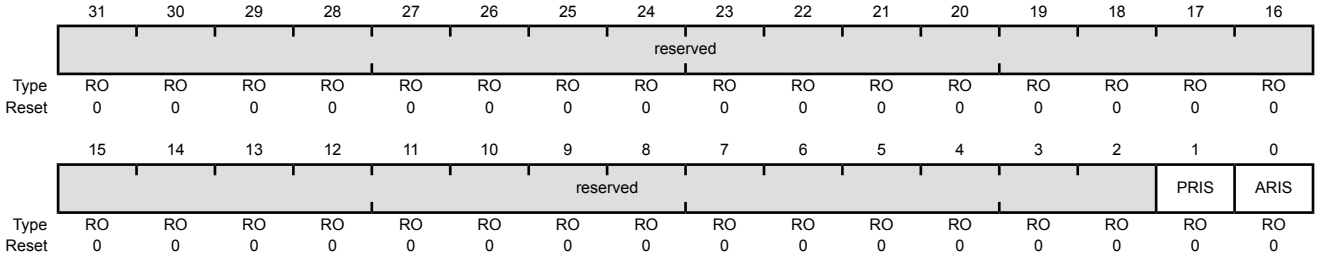
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | ERASE | R/W | 0 | Erase a Page of Flash Memory<br>If this bit is set, the page of flash main memory as specified by the contents of **FMA** is erased. A write of 0 has no effect on the state of this bit.<br>If read, the state of the previous erase access is provided. If the previous erase access is complete, a 0 is returned; otherwise, if the previous erase access is not complete, a 1 is returned.<br>This can take up to 25 ms. |
| 0 | WRITE | R/W | 0 | Write a Word into Flash Memory<br>If this bit is set, the data stored in **FMD** is written into the location as specified by the contents of **FMA**. A write of 0 has no effect on the state of this bit.<br>If read, the state of the previous write update is provided. If the previous write access is complete, a 0 is returned; otherwise, if the write access is not complete, a 1 is returned.<br>This can take up to 50 µs. |

## Register 4: Flash Controller Raw Interrupt Status (FCRIS), offset 0x00C

This register indicates that the flash controller has an interrupt condition. An interrupt is only signaled if the corresponding **FCIM** register bit is set.

Flash Controller Raw Interrupt Status (FCRIS)

Base 0x400F.D000
Offset 0x00C
Type RO, reset 0x0000.0000



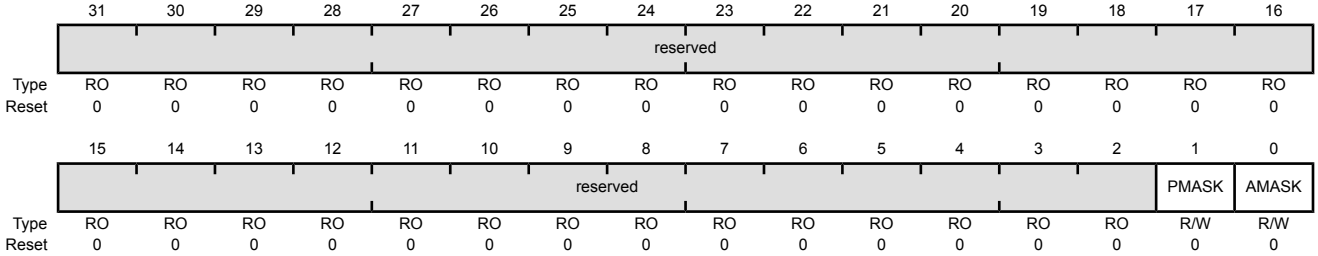

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | PRIS | RO | 0 | Programming Raw Interrupt Status<br>This bit provides status on programming cycles which are write or erase actions generated through the **FMC** register bits (see page 211).<br><br>Value Description<br>1 The programming cycle has completed.<br>0 The programming cycle has not completed.<br><br>This status is sent to the interrupt controller when the PMASK bit in the **FCIM** register is set.<br>This bit is cleared by writing a 1 to the PMISC bit in the **FCMISC** register. |
| 0 | ARIS | RO | 0 | Access Raw Interrupt Status<br><br>Value Description<br>1 A program or erase action was attempted on a block of Flash memory that contradicts the protection policy for that block as set in the **FMPPEn** registers.<br>0 No access has tried to improperly program or erase the Flash memory.<br><br>This status is sent to the interrupt controller when the AMASK bit in the **FCIM** register is set.<br>This bit is cleared by writing a 1 to the AMISC bit in the **FCMISC** register. |

## Register 5: Flash Controller Interrupt Mask (FCIM), offset 0x010

This register controls whether the flash controller generates interrupts to the controller.

Flash Controller Interrupt Mask (FCIM)

Base 0x400F.D000
Offset 0x010
Type R/W, reset 0x0000.0000



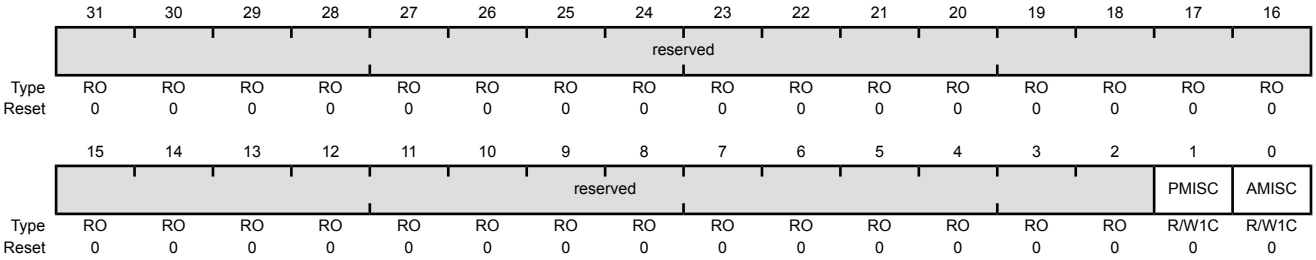

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | PMASK | R/W | 0 | Programming Interrupt Mask<br>This bit controls the reporting of the programming raw interrupt status to the interrupt controller.<br><br>Value Description<br>1 An interrupt is sent to the interrupt controller when the PRIS bit is set.<br>0 The PRIS interrupt is suppressed and not sent to the interrupt controller. |
| 0 | AMASK | R/W | 0 | Access Interrupt Mask<br>This bit controls the reporting of the access raw interrupt status to the interrupt controller.<br><br>Value Description<br>1 An interrupt is sent to the interrupt controller when the ARIS bit is set.<br>0 The ARIS interrupt is suppressed and not sent to the interrupt controller. |

## Register 6: Flash Controller Masked Interrupt Status and Clear (FCMISC), offset 0x014

This register provides two functions. First, it reports the cause of an interrupt by indicating which interrupt source or sources are signalling the interrupt. Second, it serves as the method to clear the interrupt reporting.

Flash Controller Masked Interrupt Status and Clear (FCMISC)

Base 0x400F.D000
Offset 0x014
Type R/W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | PMISC | R/W1C | 0 | Programming Masked Interrupt Status and Clear<br><br>Value Description<br>1 When read, a 1 indicates that an unmasked interrupt was signaled because a programming cycle completed.<br>Writing a 1 to this bit clears PMISC and also the PRIS bit in the **FCRIS** register (see page 213).<br>0 When read, a 0 indicates that a programming cycle complete interrupt has not occurred.<br>A write of 0 has no effect on the state of this bit. |
| 0 | AMISC | R/W1C | 0 | Access Masked Interrupt Status and Clear<br><br>Value Description<br>1 When read, a 1 indicates that an unmasked interrupt was signaled because a program or erase action was attempted on a block of Flash memory that contradicts the protection policy for that block as set in the **FMPPEn** registers.<br>Writing a 1 to this bit clears AMISC and also the ARIS bit in the **FCRIS** register (see page 213).<br>0 When read, a 0 indicates that no improper accesses have occurred.<br>A write of 0 has no effect on the state of this bit. |

## 6.6 Flash Register Descriptions (System Control Offset)

The remainder of this section lists and describes the Flash Memory registers, in numerical order by address offset. Registers in this section are relative to the System Control base address of 0x400F.E000.

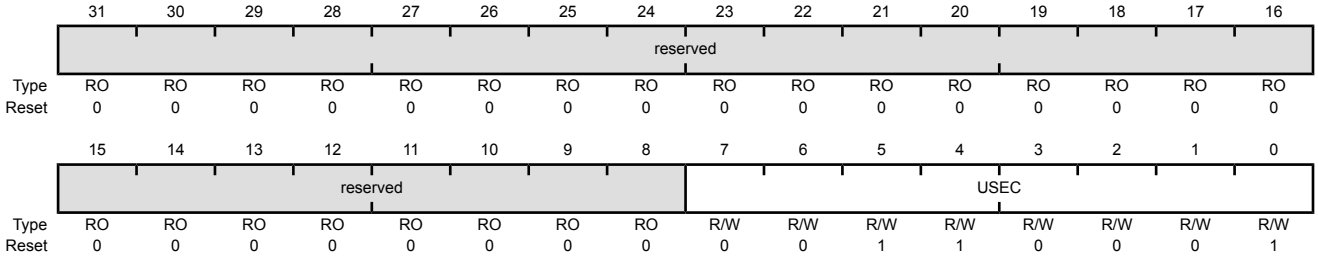
## Register 7: USec Reload (USECRL), offset 0x140

**Note:** Offset is relative to System Control base address of 0x400F.E000

This register is provided as a means of creating a 1-µs tick divider reload value for the flash controller. The internal flash has specific minimum and maximum requirements on the length of time the high voltage write pulse can be applied. It is required that this register contain the operating frequency (in MHz -1) whenever the flash is being erased or programmed. The user is required to change this value if the clocking conditions are changed for a flash erase/program operation.

USec Reload (USECRL)

Base 0x400F.E000
Offset 0x140
Type R/W, reset 0x31




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | USEC | R/W | 0x31 | Microsecond Reload Value<br>MHz -1 of the controller clock when the flash is being erased or programmed.<br>If the maximum system frequency is being used, USEC should be set to 0x31 (50 MHz) whenever the flash is being erased or programmed. |

## Register 8: Flash Memory Protection Read Enable (FMPRE), offset 0x130

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register stores the read-only protection bits for each 2-KB flash block (see the **FMPPE** registers for the execute-only protection bits). This register is loaded during the power-on reset sequence. The factory settingsare a value of 1 for all implemented banks. This implements a policy of open access and programmability. The register bits may be changed by writing the specific register bit. However, this register is R/W0; the user can only change the protection bit from a 1 to a 0 (and may NOT change a 0 to a 1). The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. For additional information, see the “Flash Memory Protection” section.

Flash Memory Protection Read Enable (FMPRE)

Base 0x400F.E000
Offset 0x130
Type R/W, reset 0xBFFF.FFFF

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DBG | | READ_ENABLE | | | | | | | | | | | | | |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | READ_ENABLE | | | | | | | | | | | | | | | |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:30 | DBG | R/W | 0x2 | User Controlled Debug Enable<br>Each bit position maps 2 Kbytes of Flash to be read-enabled.<br><br>Value Description<br>0x2 Debug access allowed |
| 29:0 | READ_ENABLE | R/W | 0x3FFFFFFF | Flash Read Enable<br>Each bit position maps 2 Kbytes of Flash to be read-enabled.<br><br>Value Description<br>0x3FFFFFFF Enables 64 KB of flash. |

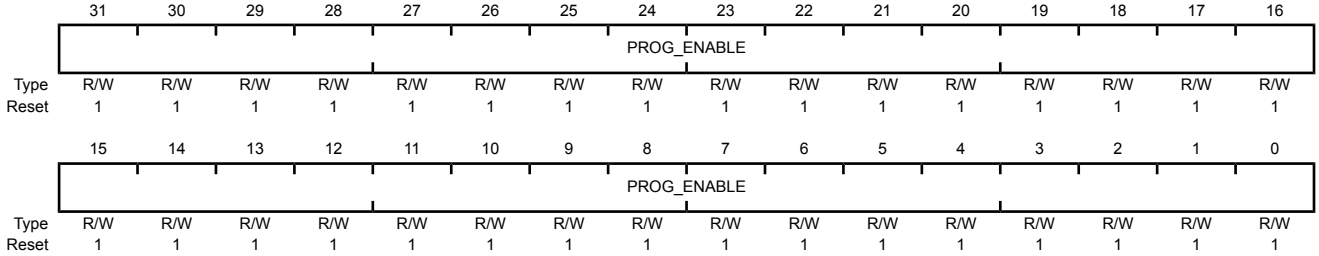
## Register 9: Flash Memory Protection Program Enable (FMPPE), offset 0x134

**Note:** Offset is relative to System Control base address of 0x400FE000.

This register stores the execute-only protection bits for each 2-KB flash block (see the **FMPRE** registers for the read-only protection bits). This register is loaded during the power-on reset sequence. The factory settings are a value of 1 for all implemented banks. This implements a policy of open access and programmability. The register bits may be changed by writing the specific register bit. However, this register is R/W0; the user can only change the protection bit from a 1 to a 0 (and may NOT change a 0 to a 1). The changes are not permanent until the register is committed (saved), at which point the bit change is permanent. If a bit is changed from a 1 to a 0 and not committed, it may be restored by executing a power-on reset sequence. For additional information, see the “Flash Memory Protection” section.

Flash Memory Protection Program Enable (FMPPE)

Base 0x400F.E000
Offset 0x134
Type R/W, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | PROG_ENABLE | R/W | 0xFFFFFFFF | Flash Programming Enable<br>Each bit position maps 2 Kbytes of Flash to be write-enabled. |

| Value | Description |
|---|---|
| 0xFFFFFFFF | Enables 64 KB of flash. |

# 7 General-Purpose Input/Outputs (GPIOs)

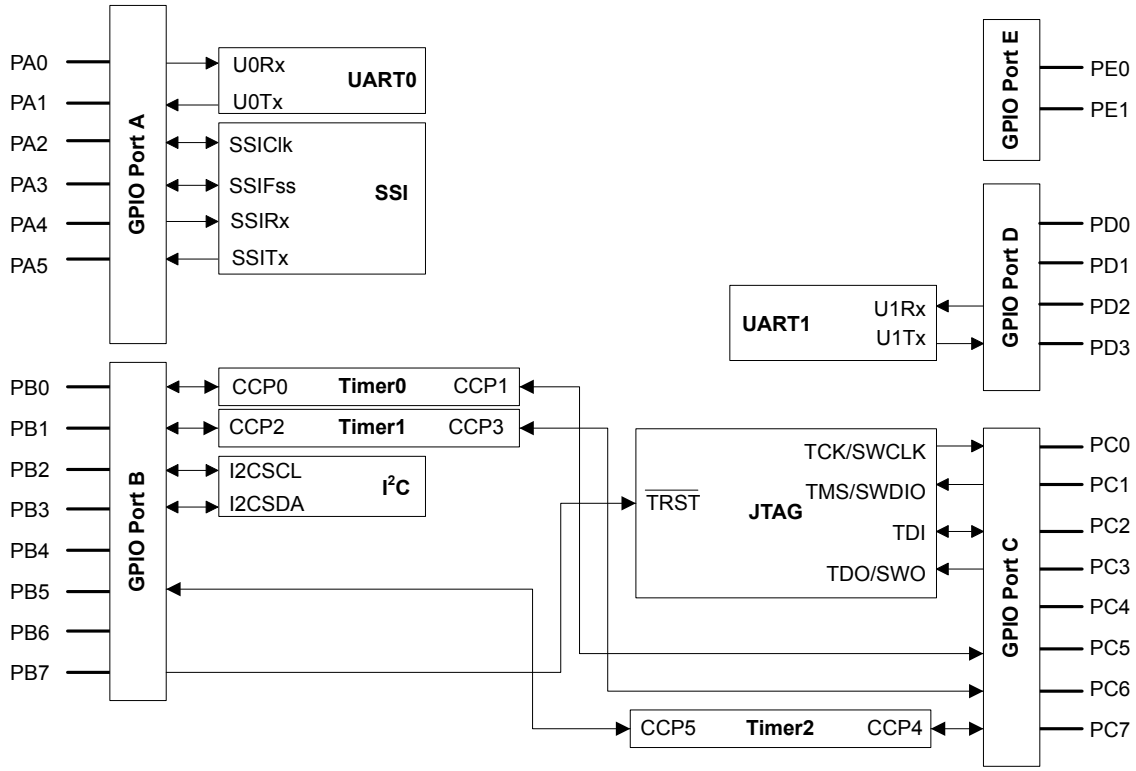
The GPIO module is composed of five physical GPIO blocks, each corresponding to an individual GPIO port (Port A, Port B, Port C, Port D, Port E). The GPIO module supports 7-28 programmable input/output pins, depending on the peripherals being used.

The GPIO module has the following features:

- 7-28 GPIOs, depending on configuration
- 5-V-tolerant in input configuration
- Programmable control for GPIO interrupts
  - Interrupt generation masking
  - Edge-triggered on rising, falling, or both
  - Level-sensitive on High or Low values
- Bit masking in both read and write operations through address lines
- Can initiate an ADC sample sequence
- Pins configured as digital inputs are Schmitt-triggered.
- Programmable control for GPIO pad configuration
  - Weak pull-up or pull-down resistors
  - 2-mA, 4-mA, and 8-mA pad drive for digital communication
  - Slew rate control for the 8-mA drive
  - Open drain enables
  - Digital input enables

## 7.1 Block Diagram

**Figure 7-1. GPIO Module Block Diagram**



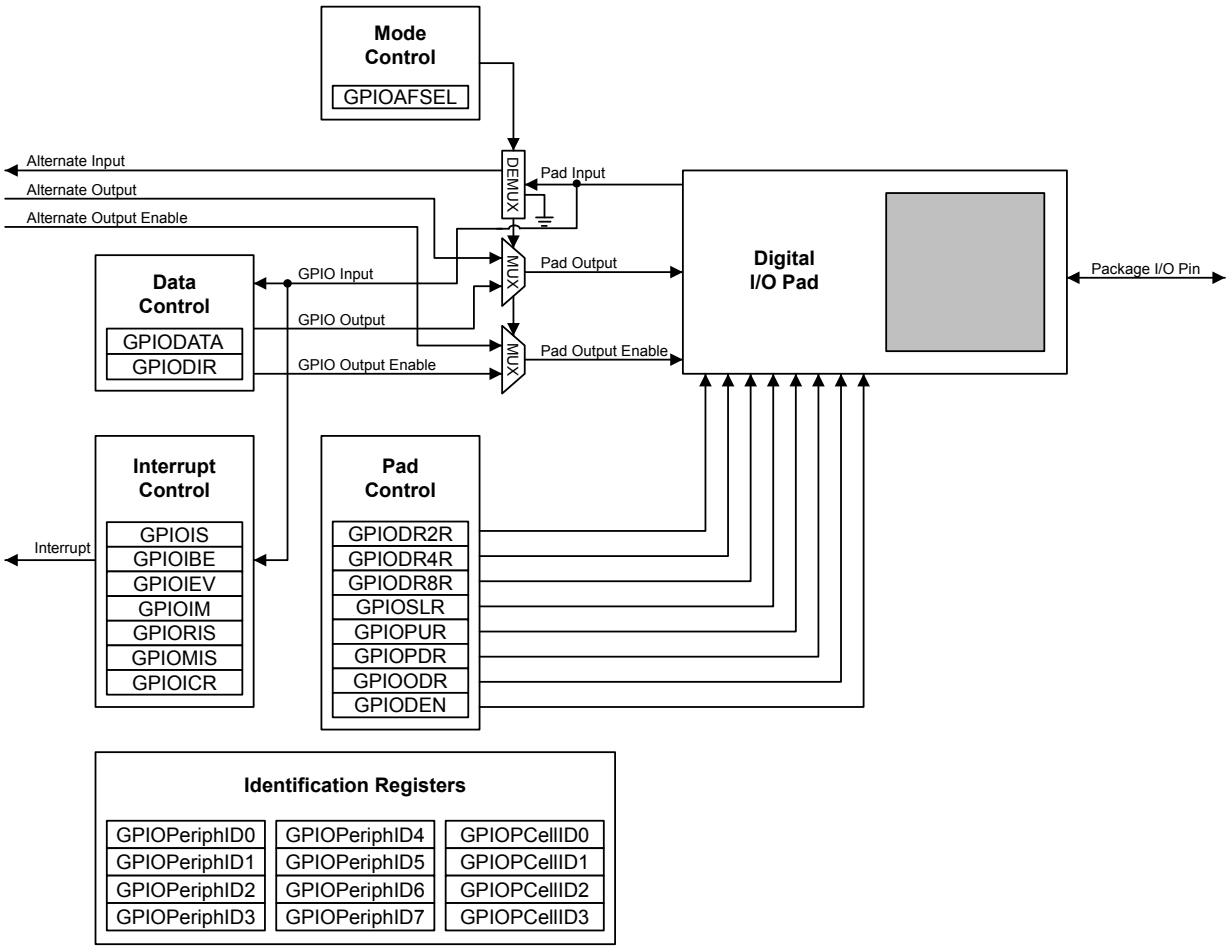

## 7.2 Functional Description

**Important:** All GPIO pins are inputs by default (**GPIODIR**=0 and **GPIOAFSEL**=0), with the exception of the five JTAG pins (PB7 and PC[3:0]). The JTAG pins default to their JTAG functionality (**GPIOAFSEL**=1). A Power-On-Reset ($\overline{\text{POR}}$) or asserting an external reset ($\overline{\text{RST}}$) puts both groups of pins back to their default state.

While debugging systems where PB7 is being used as a GPIO, care must be taken to ensure that a Low value is not applied to the pin when the part is reset. Because PB7 reverts to the $\overline{\text{TRST}}$ function after reset, a Low value on the pin causes the JTAG controller to be reset, resulting in a loss of JTAG communication.

Each GPIO port is a separate hardware instantiation of the same physical block (see Figure 7-2 on page 222). The LM3S828 microcontroller contains five ports and thus five of these physical GPIO blocks.

**Figure 7-2. GPIO Port Block Diagram**




## 7.2.1 Data Control

The data control registers allow software to configure the operational modes of the GPIOs. The data direction register configures the GPIO as an input or an output while the data register either captures incoming data or drives it out to the pads.

### 7.2.1.1 Data Direction Operation

The **GPIO Direction (GPIODIR)** register (see page 229) is used to configure each individual pin as an input or output. When the data direction bit is set to 0, the GPIO is configured as an input and the corresponding data register bit will capture and store the value on the GPIO port. When the data direction bit is set to 1, the GPIO is configured as an output and the corresponding data register bit will be driven out on the GPIO port.

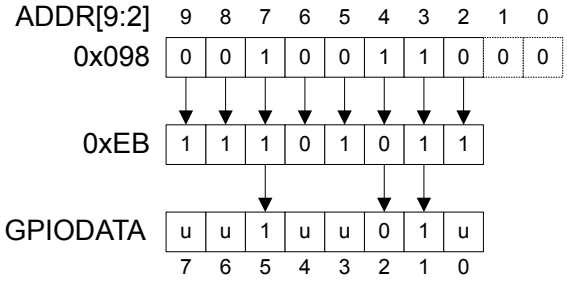
### 7.2.1.2 Data Register Operation

To aid in the efficiency of software, the GPIO ports allow for the modification of individual bits in the **GPIO Data (GPIODATA)** register (see page 228) by using bits [9:2] of the address bus as a mask. This allows software drivers to modify individual GPIO pins in a single instruction, without affecting the state of the other pins. This is in contrast to the "typical" method of doing a read-modify-write operation to set or clear an individual GPIO pin. To accommodate this feature, the **GPIODATA** register covers 256 locations in the memory map.

During a write, if the address bit associated with that data bit is set to 1, the value of the **GPIODATA** register is altered. If it is cleared to 0, it is left unchanged.

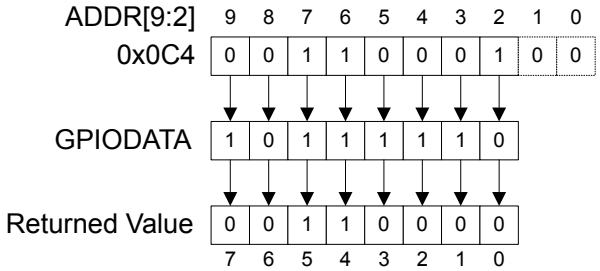
For example, writing a value of 0xEB to the address GPIODATA + 0x098 would yield as shown in Figure 7-3 on page 223, where u is data unchanged by the write.

**Figure 7-3. GPIODATA Write Example**




During a read, if the address bit associated with the data bit is set to 1, the value is read. If the address bit associated with the data bit is set to 0, it is read as a zero, regardless of its actual value. For example, reading address GPIODATA + 0x0C4 yields as shown in Figure 7-4 on page 223.

**Figure 7-4. GPIODATA Read Example**




## 7.2.2 Interrupt Control

The interrupt capabilities of each GPIO port are controlled by a set of seven registers. With these registers, it is possible to select the source of the interrupt, its polarity, and the edge properties. When one or more GPIO inputs cause an interrupt, a single interrupt output is sent to the interrupt controller for the entire GPIO port. For edge-triggered interrupts, software must clear the interrupt to enable any further interrupts. For a level-sensitive interrupt, it is assumed that the external source holds the level constant for the interrupt to be recognized by the controller.

Three registers are required to define the edge or sense that causes interrupts:

- **GPIO Interrupt Sense (GPIOIS)** register (see page 230)
- **GPIO Interrupt Both Edges (GPIOIBE)** register (see page 231)
- **GPIO Interrupt Event (GPIOIEV)** register (see page 232)

Interrupts are enabled/disabled via the **GPIO Interrupt Mask (GPIOIM)** register (see page 233).

When an interrupt condition occurs, the state of the interrupt signal can be viewed in two locations: the **GPIO Raw Interrupt Status (GPIORIS)** and **GPIO Masked Interrupt Status (GPIOMIS)** registers (see page 234 and page 235). As the name implies, the **GPIOMIS** register only shows interrupt

conditions that are allowed to be passed to the controller. The **GPIORIS** register indicates that a GPIO pin meets the conditions for an interrupt, but has not necessarily been sent to the controller.

In addition to providing GPIO functionality, PB4 can also be used as an external trigger for the ADC. If PB4 is configured as a non-masked interrupt pin (the appropriate bit of GPIOIM is set to 1), not only is an interrupt for PortB generated, but an external trigger signal is sent to the ADC. If the **ADC Event Multiplexer Select (ADCEMUX)** register is configured to use the external trigger, an ADC conversion is initiated.

If no other PortB pins are being used to generate interrupts, the **Interrupt 0-31 Set Enable (EN0)** register can disable the PortB interrupts, and the ADC interrupt can be used to read back the converted data. Otherwise, the PortB interrupt handler needs to ignore and clear interrupts on PB4, and wait for the ADC interrupt or the ADC interrupt must be disabled in the **EN0** register and the PortB interrupt handler must poll the ADC registers until the conversion is completed. See page 95 for more information.

Interrupts are cleared by writing a 1 to the appropriate bit of the **GPIO Interrupt Clear (GPIOICR)** register (see page 236).

When programming the following interrupt control registers, the interrupts should be masked (**GPIOIM** set to 0). Writing any value to an interrupt control register (**GPIOIS**, **GPIOIBE**, or **GPIOIEV**) can generate a spurious interrupt if the corresponding bits are enabled.

### 7.2.3 Mode Control

The GPIO pins can be controlled by either hardware or software. When hardware control is enabled via the **GPIO Alternate Function Select (GPIOAFSEL)** register (see page 237), the pin state is controlled by its alternate function (that is, the peripheral). Software control corresponds to GPIO mode, where the **GPIODATA** register is used to read/write the corresponding pins.

### 7.2.4 Pad Control

The pad control registers allow for GPIO pad configuration by software based on the application requirements. The pad control registers include the **GPIODR2R**, **GPIODR4R**, **GPIODR8R**, **GPIOODR**, **GPIOPUR**, **GPIOPDR**, **GPIOSLR**, and **GPIODEN** registers. These registers control drive strength, open-drain configuration, pull-up and pull-down resistors, slew-rate control and digital input enable.

### 7.2.5 Identification

The identification registers configured at reset allow software to detect and identify the module as a GPIO block. The identification registers include the **GPIOPeriphID0**-**GPIOPeriphID7** registers as well as the **GPIOPCellID0**-**GPIOPCellID3** registers.

## 7.3 Initialization and Configuration

To use the GPIO, the peripheral clock must be enabled by setting the appropriate GPIO Port bit field (GPIOn) in the **RCGC2** register.

On reset, all GPIO pins (except for the five JTAG pins) default to general-purpose input mode (**GPIODIR**=0 and **GPIOAFSEL**=0). Table 7-1 on page 225 shows all possible configurations of the GPIO pads and the control register settings required to achieve them. Table 7-2 on page 225 shows how a rising edge interrupt would be configured for pin 2 of a GPIO port.

**Table 7-1. GPIO Pad Configuration Examples**

| Configuration | GPIO Register Bit Value[a] | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | AFSEL | DIR | ODR | DEN | PUR | PDR | DR2R | DR4R | DR8R | SLR |
| Digital Input (GPIO) | 0 | 0 | 0 | 1 | ? | ? | X | X | X | X |
| Digital Output (GPIO) | 0 | 1 | 0 | 1 | ? | ? | ? | ? | ? | ? |
| Open Drain Output (GPIO) | 0 | 1 | 1 | 1 | X | X | ? | ? | ? | ? |
| Open Drain Input/Output ($I^2C$) | 1 | X | 1 | 1 | X | X | ? | ? | ? | ? |
| Digital Input (Timer CCP) | 1 | X | 0 | 1 | ? | ? | X | X | X | X |
| Digital Output (Timer PWM) | 1 | X | 0 | 1 | ? | ? | ? | ? | ? | ? |
| Digital Input/Output (SSI) | 1 | X | 0 | 1 | ? | ? | ? | ? | ? | ? |
| Digital Input/Output (UART) | 1 | X | 0 | 1 | ? | ? | ? | ? | ? | ? |

a. X=Ignored (don't care bit)

?=Can be either 0 or 1, depending on the configuration

**Table 7-2. GPIO Interrupt Configuration Example**

| Register | Desired Interrupt Event Trigger | Pin 2 Bit Value[a] | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| GPIOIS | 0=edge 1=level | X | X | X | X | X | 0 | X | X |
| GPIOIBE | 0=single edge 1=both edges | X | X | X | X | X | 0 | X | X |
| GPIOIEV | 0=Low level, or negative edge 1=High level, or positive edge | X | X | X | X | X | 1 | X | X |
| GPIOIM | 0=masked 1=not masked | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

a. X=Ignored (don't care bit)

## 7.4 Register Map

Table 7-3 on page 226 lists the GPIO registers. The offset listed is a hexadecimal increment to the register's address, relative to that GPIO port's base address:

- GPIO Port A: 0x4000.4000
- GPIO Port B: 0x4000.5000
- GPIO Port C: 0x4000.6000
- GPIO Port D: 0x4000.7000
- GPIO Port E: 0x4002.4000

Note that the GPIO module clock must be enabled before the registers can be programmed (see page 197). There must be a delay of 3 system clocks after the GPIO module clock is enabled before any GPIO module registers are accessed.

**Important:** The GPIO registers in this chapter are duplicated in each GPIO block; however, depending on the block, all eight bits may not be connected to a GPIO pad. In those cases, writing to those unconnected bits has no effect, and reading those unconnected bits returns no meaningful data.

**Note:** The default reset value for the **GPIOAFSEL** register is 0x0000.0000 for all GPIO pins, with the exception of the five JTAG pins (PB7 and PC[3:0]). These five pins default to JTAG functionality. Because of this, the default reset value of **GPIOAFSEL** for GPIO Port B is 0x0000.0080 while the default reset value for Port C is 0x0000.000F.

**Table 7-3. GPIO Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | GPIODATA | R/W | 0x0000.0000 | GPIO Data | 228 |
| 0x400 | GPIODIR | R/W | 0x0000.0000 | GPIO Direction | 229 |
| 0x404 | GPIOIS | R/W | 0x0000.0000 | GPIO Interrupt Sense | 230 |
| 0x408 | GPIOIBE | R/W | 0x0000.0000 | GPIO Interrupt Both Edges | 231 |
| 0x40C | GPIOIEV | R/W | 0x0000.0000 | GPIO Interrupt Event | 232 |
| 0x410 | GPIOIM | R/W | 0x0000.0000 | GPIO Interrupt Mask | 233 |
| 0x414 | GPIORIS | RO | 0x0000.0000 | GPIO Raw Interrupt Status | 234 |
| 0x418 | GPIOMIS | RO | 0x0000.0000 | GPIO Masked Interrupt Status | 235 |
| 0x41C | GPIOICR | W1C | 0x0000.0000 | GPIO Interrupt Clear | 236 |
| 0x420 | GPIOAFSEL | R/W | - | GPIO Alternate Function Select | 237 |
| 0x500 | GPIODR2R | R/W | 0x0000.00FF | GPIO 2-mA Drive Select | 239 |
| 0x504 | GPIODR4R | R/W | 0x0000.0000 | GPIO 4-mA Drive Select | 240 |
| 0x508 | GPIODR8R | R/W | 0x0000.0000 | GPIO 8-mA Drive Select | 241 |
| 0x50C | GPIOODR | R/W | 0x0000.0000 | GPIO Open Drain Select | 242 |
| 0x510 | GPIOPUR | R/W | 0x0000.00FF | GPIO Pull-Up Select | 243 |
| 0x514 | GPIOPDR | R/W | 0x0000.0000 | GPIO Pull-Down Select | 244 |
| 0x518 | GPIOSLR | R/W | 0x0000.0000 | GPIO Slew Rate Control Select | 245 |
| 0x51C | GPIODEN | R/W | 0x0000.00FF | GPIO Digital Enable | 246 |
| 0xFD0 | GPIOPeriphID4 | RO | 0x0000.0000 | GPIO Peripheral Identification 4 | 247 |
| 0xFD4 | GPIOPeriphID5 | RO | 0x0000.0000 | GPIO Peripheral Identification 5 | 248 |
| 0xFD8 | GPIOPeriphID6 | RO | 0x0000.0000 | GPIO Peripheral Identification 6 | 249 |
| 0xFDC | GPIOPeriphID7 | RO | 0x0000.0000 | GPIO Peripheral Identification 7 | 250 |
| 0xFE0 | GPIOPeriphID0 | RO | 0x0000.0061 | GPIO Peripheral Identification 0 | 251 |

**Table 7-3. GPIO Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0xFE4 | GPIOPeriphID1 | RO | 0x0000.0000 | GPIO Peripheral Identification 1 | 252 |
| 0xFE8 | GPIOPeriphID2 | RO | 0x0000.0018 | GPIO Peripheral Identification 2 | 253 |
| 0xFEC | GPIOPeriphID3 | RO | 0x0000.0001 | GPIO Peripheral Identification 3 | 254 |
| 0xFF0 | GPIOPCellID0 | RO | 0x0000.000D | GPIO PrimeCell Identification 0 | 255 |
| 0xFF4 | GPIOPCellID1 | RO | 0x0000.00F0 | GPIO PrimeCell Identification 1 | 256 |
| 0xFF8 | GPIOPCellID2 | RO | 0x0000.0005 | GPIO PrimeCell Identification 2 | 257 |
| 0xFFC | GPIOPCellID3 | RO | 0x0000.00B1 | GPIO PrimeCell Identification 3 | 258 |

## 7.5 Register Descriptions

The remainder of this section lists and describes the GPIO registers, in numerical order by address offset.

### Register 1: GPIO Data (GPIODATA), offset 0x000

The **GPIODATA** register is the data register. In software control mode, values written in the **GPIODATA** register are transferred onto the GPIO port pins if the respective pins have been configured as outputs through the **GPIO Direction (GPIODIR)** register (see page 229).

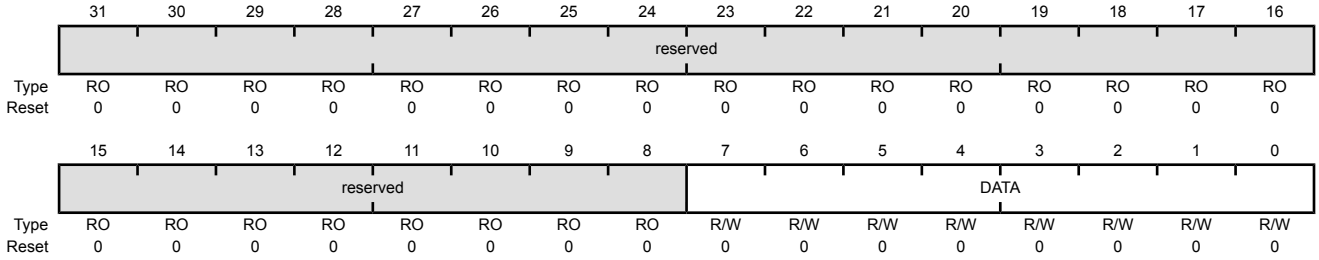
In order to write to **GPIODATA**, the corresponding bits in the mask, resulting from the address bus bits [9:2], must be High. Otherwise, the bit values remain unchanged by the write.

Similarly, the values read from this register are determined for each bit by the mask bit derived from the address used to access the data register, bits [9:2]. Bits that are 1 in the address mask cause the corresponding bits in **GPIODATA** to be read, and bits that are 0 in the address mask cause the corresponding bits in **GPIODATA** to be read as 0, regardless of their value.

A read from **GPIODATA** returns the last bit value written if the respective pins are configured as outputs, or it returns the value on the corresponding input pin when these are configured as inputs. All bits are cleared by a reset.

GPIO Data (GPIODATA)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x000
Type R/W, reset 0x0000.0000



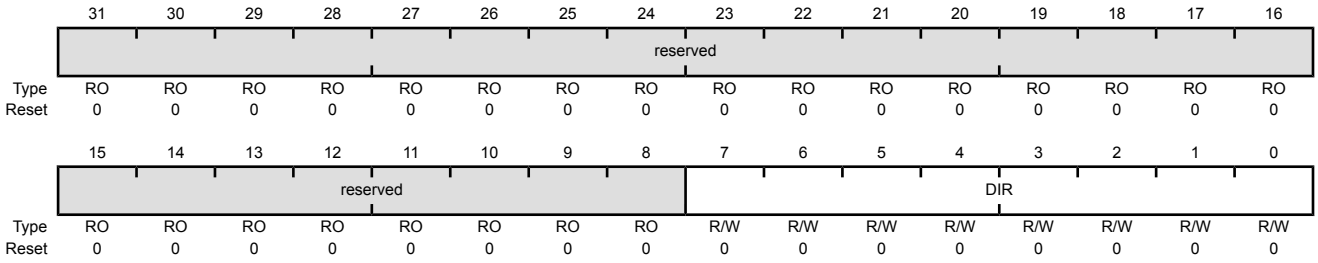

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DATA | R/W | 0x00 | GPIO Data<br>This register is virtually mapped to 256 locations in the address space. To facilitate the reading and writing of data to these registers by independent drivers, the data read from and the data written to the registers are masked by the eight address lines `ipaddr[9:2]`. Reads from this register return its current state. Writes to this register only affect bits that are not masked by `ipaddr[9:2]` and are configured as outputs. See “Data Register Operation” on page 222 for examples of reads and writes. |

## Register 2: GPIO Direction (GPIODIR), offset 0x400

The **GPIODIR** register is the data direction register. Bits set to 1 in the **GPIODIR** register configure the corresponding pin to be an output, while bits set to 0 configure the pins to be inputs. All bits are cleared by a reset, meaning all GPIO pins are inputs by default.

GPIO Direction (GPIODIR)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x400
Type R/W, reset 0x0000.0000



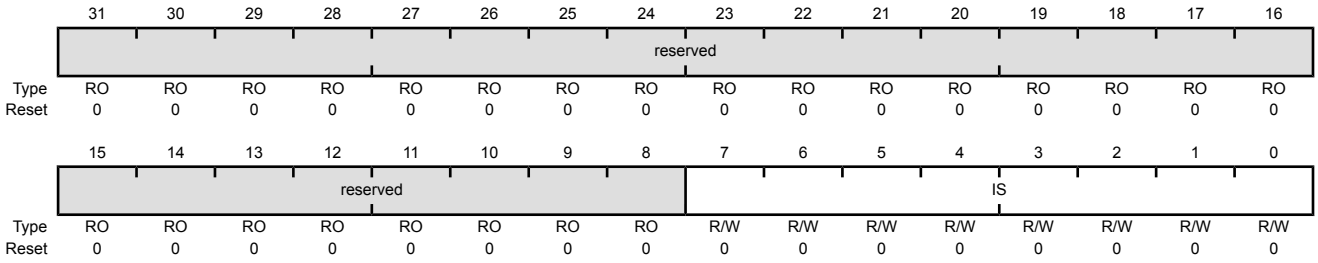

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DIR | R/W | 0x00 | GPIO Data Direction<br>The `DIR` values are defined as follows:<br><br>Value Description<br>0 Pins are inputs.<br>1 Pins are outputs. |

## Register 3: GPIO Interrupt Sense (GPIOIS), offset 0x404

The **GPIOIS** register is the interrupt sense register. Bits set to 1 in **GPIOIS** configure the corresponding pins to detect levels, while bits set to 0 configure the pins to detect edges. All bits are cleared by a reset.

GPIO Interrupt Sense (GPIOIS)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x404
Type R/W, reset 0x0000.0000



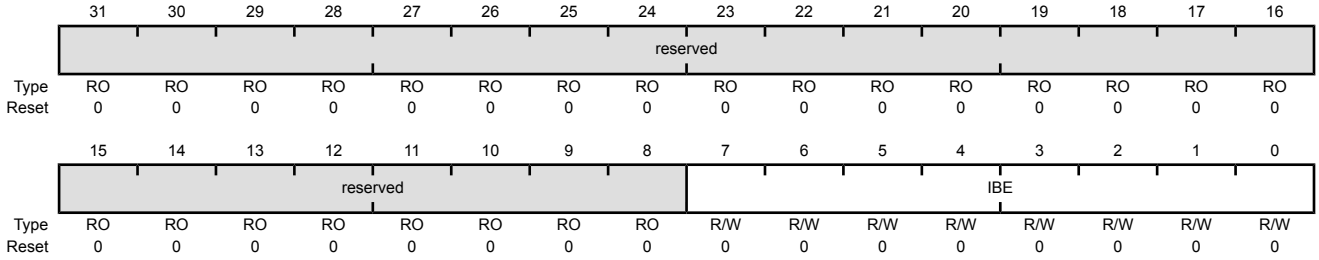

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | IS | R/W | 0x00 | GPIO Interrupt Sense<br>The `IS` values are defined as follows:<br><br>Value Description<br>0 Edge on corresponding pin is detected (edge-sensitive).<br>1 Level on corresponding pin is detected (level-sensitive). |

## Register 4: GPIO Interrupt Both Edges (GPIOIBE), offset 0x408

The **GPIOIBE** register is the interrupt both-edges register. When the corresponding bit in the **GPIO Interrupt Sense (GPIOIS)** register (see page 230) is set to detect edges, bits set to High in **GPIOIBE** configure the corresponding pin to detect both rising and falling edges, regardless of the corresponding bit in the **GPIO Interrupt Event (GPIOIEV)** register (see page 232). Clearing a bit configures the pin to be controlled by **GPIOIEV**. All bits are cleared by a reset.

GPIO Interrupt Both Edges (GPIOIBE)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x408
Type R/W, reset 0x0000.0000



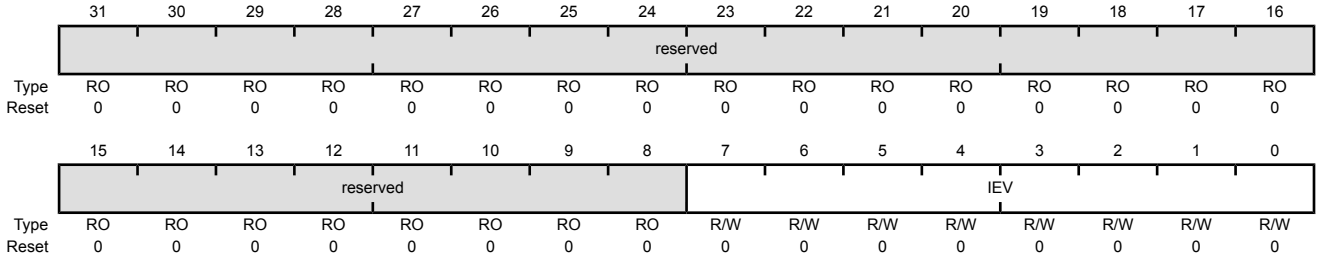

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | IBE | R/W | 0x00 | GPIO Interrupt Both Edges<br>The `IBE` values are defined as follows:<br><br>Value Description<br>0 Interrupt generation is controlled by the **GPIO Interrupt Event (GPIOIEV)** register (see page 232).<br>1 Both edges on the corresponding pin trigger an interrupt.<br><br>**Note:** Single edge is determined by the corresponding bit in **GPIOIEV**. |

## Register 5: GPIO Interrupt Event (GPIOIEV), offset 0x40C

The **GPIOIEV** register is the interrupt event register. Bits set to High in **GPIOIEV** configure the corresponding pin to detect rising edges or high levels, depending on the corresponding bit value in the **GPIO Interrupt Sense (GPIOIS)** register (see page 230). Clearing a bit configures the pin to detect falling edges or low levels, depending on the corresponding bit value in **GPIOIS**. All bits are cleared by a reset.

GPIO Interrupt Event (GPIOIEV)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x40C
Type R/W, reset 0x0000.0000



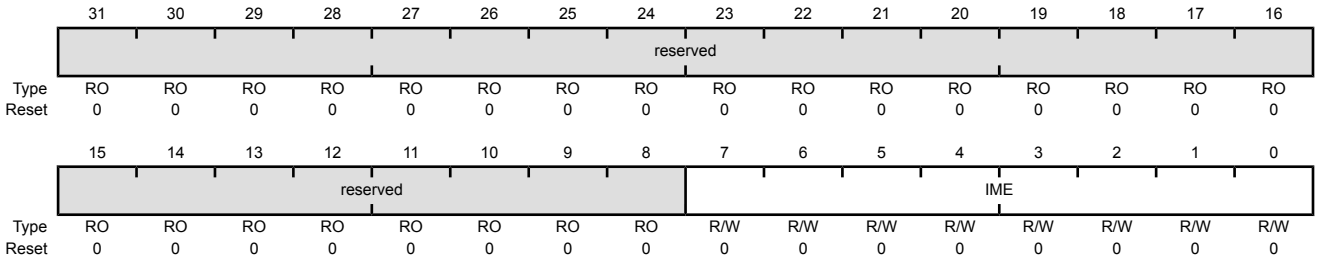

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | IEV | R/W | 0x00 | GPIO Interrupt Event<br>The `IEV` values are defined as follows:<br><br>Value Description<br>0 Falling edge or Low levels on corresponding pins trigger interrupts.<br>1 Rising edge or High levels on corresponding pins trigger interrupts. |

## Register 6: GPIO Interrupt Mask (GPIOIM), offset 0x410

The **GPIOIM** register is the interrupt mask register. Bits set to High in **GPIOIM** allow the corresponding pins to trigger their individual interrupts and the combined **GPIOINTR** line. Clearing a bit disables interrupt triggering on that pin. All bits are cleared by a reset.

GPIO Interrupt Mask (GPIOIM)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x410
Type R/W, reset 0x0000.0000



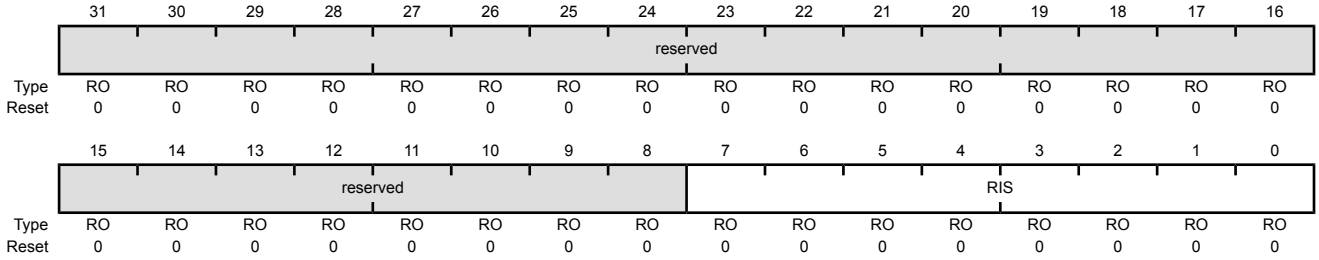

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | IME | R/W | 0x00 | GPIO Interrupt Mask Enable<br>The `IME` values are defined as follows:<br><br>Value Description<br>0 Corresponding pin interrupt is masked.<br>1 Corresponding pin interrupt is not masked. |

## Register 7: GPIO Raw Interrupt Status (GPIORIS), offset 0x414

The **GPIORIS** register is the raw interrupt status register. Bits read High in **GPIORIS** reflect the status of interrupt trigger conditions detected (raw, prior to masking), indicating that all the requirements have been met, before they are finally allowed to trigger by the **GPIO Interrupt Mask (GPIOIM)** register (see page 233). Bits read as zero indicate that corresponding input pins have not initiated an interrupt. All bits are cleared by a reset.

GPIO Raw Interrupt Status (GPIORIS)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x414
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | RIS | RO | 0x00 | GPIO Interrupt Raw Status<br>Reflects the status of interrupt trigger condition detection on pins (raw, prior to masking).<br>The `RIS` values are defined as follows:<br><br>Value Description<br>0 Corresponding pin interrupt requirements not met.<br>1 Corresponding pin interrupt has met requirements. |

## Register 8: GPIO Masked Interrupt Status (GPIOMIS), offset 0x418

The **GPIOMIS** register is the masked interrupt status register. Bits read High in **GPIOMIS** reflect the status of input lines triggering an interrupt. Bits read as Low indicate that either no interrupt has been generated, or the interrupt is masked.

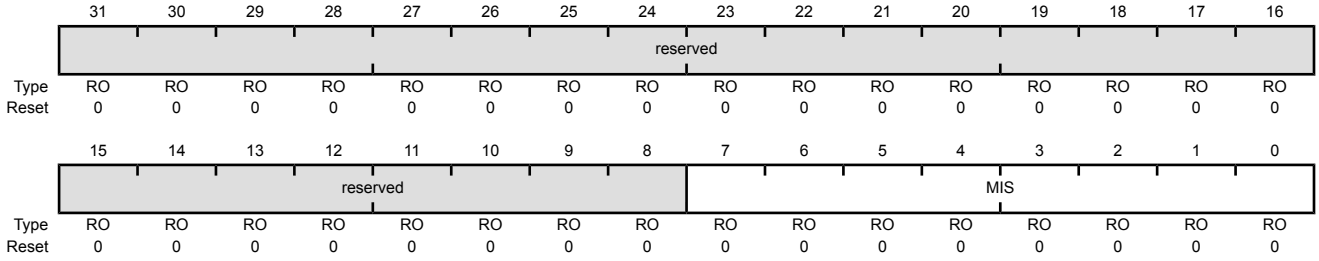
In addition to providing GPIO functionality, PB4 can also be used as an external trigger for the ADC. If PB4 is configured as a non-masked interrupt pin (the appropriate bit of GPIOIM is set to 1), not only is an interrupt for PortB generated, but an external trigger signal is sent to the ADC. If the **ADC Event Multiplexer Select (ADCEMUX)** register is configured to use the external trigger, an ADC conversion is initiated.

If no other PortB pins are being used to generate interrupts, the **Interrupt 0-31 Set Enable (EN0)** register can disable the PortB interrupts, and the ADC interrupt can be used to read back the converted data. Otherwise, the PortB interrupt handler needs to ignore and clear interrupts on PB4, and wait for the ADC interrupt or the ADC interrupt must be disabled in the **EN0** register and the PortB interrupt handler must poll the ADC registers until the conversion is completed. See page 95 for more information.

**GPIOMIS** is the state of the interrupt after masking.

GPIO Masked Interrupt Status (GPIOMIS)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x418
Type RO, reset 0x0000.0000



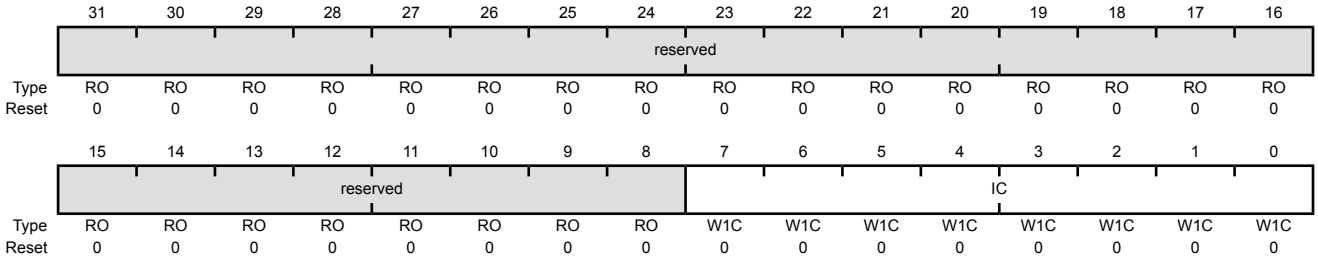

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | MIS | RO | 0x00 | GPIO Masked Interrupt Status<br>Masked value of interrupt due to corresponding pin.<br>The MIS values are defined as follows:<br><br>Value Description<br>0 Corresponding GPIO line interrupt not active.<br>1 Corresponding GPIO line asserting interrupt. |

## Register 9: GPIO Interrupt Clear (GPIOICR), offset 0x41C

The **GPIOICR** register is the interrupt clear register. Writing a 1 to a bit in this register clears the corresponding interrupt edge detection logic register. Writing a 0 has no effect.

GPIO Interrupt Clear (GPIOICR)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x41C
Type W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | IC | W1C | 0x00 | GPIO Interrupt Clear<br>The `IC` values are defined as follows:<br><br>Value Description<br>0 Corresponding interrupt is unaffected.<br>1 Corresponding interrupt is cleared. |

## Register 10: GPIO Alternate Function Select (GPIOAFSEL), offset 0x420

The **GPIOAFSEL** register is the mode control select register. Writing a 1 to any bit in this register selects the hardware control for the corresponding GPIO line. All bits are cleared by a reset, therefore no GPIO line is set to hardware control by default.

**Important:** All GPIO pins are inputs by default (**GPIODIR**=0 and **GPIOAFSEL**=0), with the exception of the five JTAG pins (PB7 and PC[3:0]). The JTAG pins default to their JTAG functionality (**GPIOAFSEL**=1). A Power-On-Reset ($\overline{\text{POR}}$) or asserting an external reset ($\overline{\text{RST}}$) puts both groups of pins back to their default state.

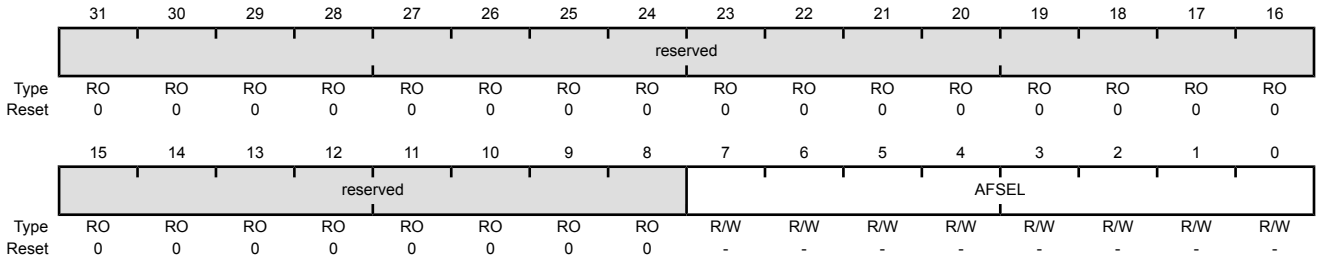
While debugging systems where PB7 is being used as a GPIO, care must be taken to ensure that a Low value is not applied to the pin when the part is reset. Because PB7 reverts to the $\overline{\text{TRST}}$ function after reset, a Low value on the pin causes the JTAG controller to be reset, resulting in a loss of JTAG communication.

**Caution – If the JTAG pins are used as GPIOs in a design, PB7 and PC2 cannot have external pull-down resistors connected to both of them at the same time. If both pins are pulled Low during reset, the controller has unpredictable behavior. If this happens, remove one or both of the pull-down resistors, and apply $\overline{\text{RST}}$ or power-cycle the part.**

**It is possible to create a software sequence that prevents the debugger from connecting to the Stellaris® microcontroller. If the program code loaded into flash immediately changes the JTAG pins to their GPIO functionality, the debugger may not have enough time to connect and halt the controller before the JTAG pin functionality switches. This may lock the debugger out of the part. This can be avoided with a software routine that restores JTAG functionality based on an external or software trigger.**

GPIO Alternate Function Select (GPIOAFSEL)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x420
Type R/W, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7:0 | AFSEL | R/W | - | GPIO Alternate Function Select<br>The AFSEL values are defined as follows: |

| Value | Description |
|---|---|
| 0 | Software control of corresponding GPIO line (GPIO mode). |
| 1 | Hardware control of corresponding GPIO line (alternate hardware function). |

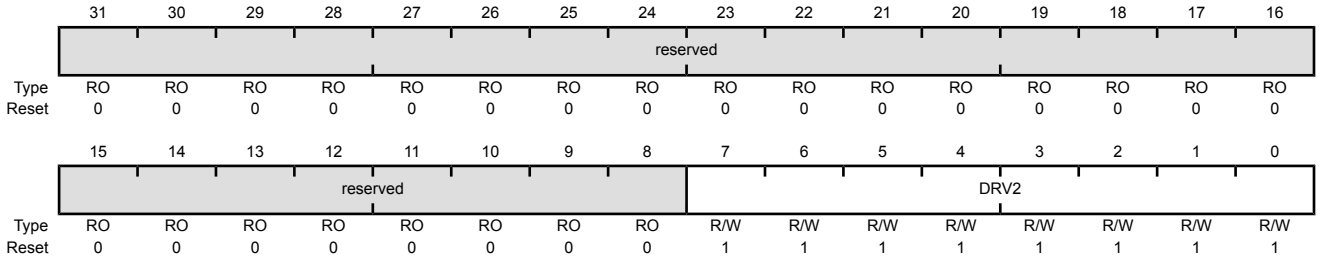
**Note:** The default reset value for the **GPIOAFSEL** register is 0x0000.0000 for all GPIO pins, with the exception of the five JTAG pins (PB7 and PC[3:0]). These five pins default to JTAG functionality. Because of this, the default reset value of **GPIOAFSEL** for GPIO Port B is 0x0000.0080 while the default reset value for Port C is 0x0000.000F.

## Register 11: GPIO 2-mA Drive Select (GPIODR2R), offset 0x500

The **GPIODR2R** register is the 2-mA drive control register. It allows for each GPIO signal in the port to be individually configured without affecting the other pads. When writing a DRV2 bit for a GPIO signal, the corresponding DRV4 bit in the **GPIODR4R** register and the DRV8 bit in the **GPIODR8R** register are automatically cleared by hardware.

GPIO 2-mA Drive Select (GPIODR2R)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x500
Type R/W, reset 0x0000.00FF



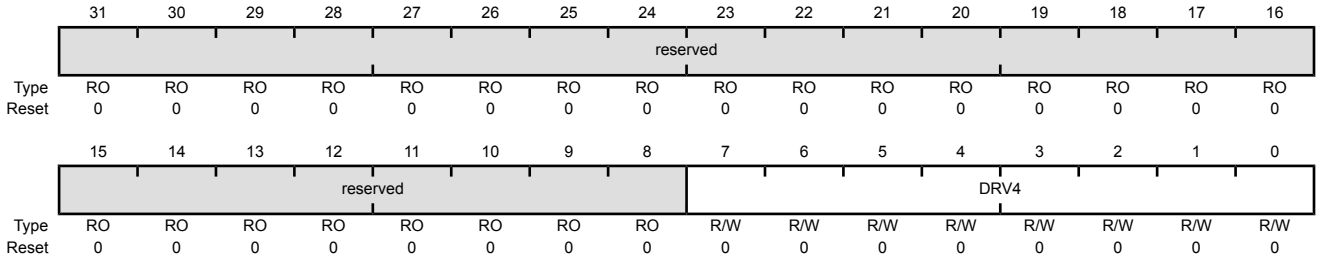

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DRV2 | R/W | 0xFF | Output Pad 2-mA Drive Enable<br>A write of 1 to either **GPIODR4[n]** or **GPIODR8[n]** clears the corresponding 2-mA enable bit. The change is effective on the second clock cycle after the write. |

## Register 12: GPIO 4-mA Drive Select (GPIODR4R), offset 0x504

The **GPIODR4R** register is the 4-mA drive control register. It allows for each GPIO signal in the port to be individually configured without affecting the other pads. When writing the DRV4 bit for a GPIO signal, the corresponding DRV2 bit in the **GPIODR2R** register and the DRV8 bit in the **GPIODR8R** register are automatically cleared by hardware.

GPIO 4-mA Drive Select (GPIODR4R)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x504
Type R/W, reset 0x0000.0000



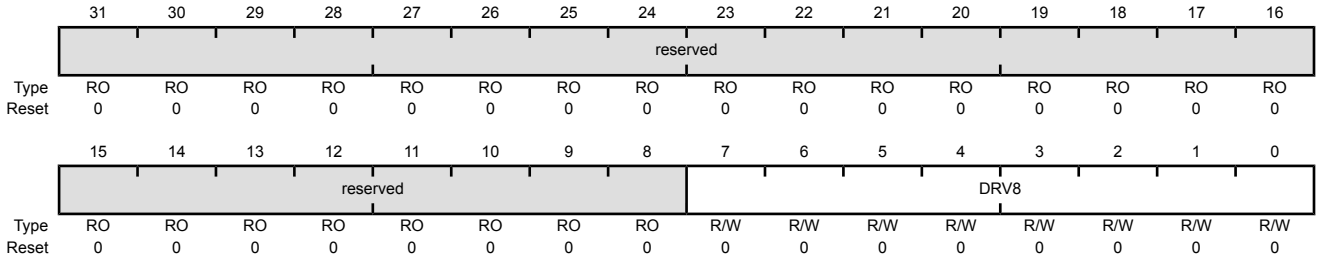

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DRV4 | R/W | 0x00 | Output Pad 4-mA Drive Enable<br>A write of 1 to either **GPIODR2[n]** or **GPIODR8[n]** clears the corresponding 4-mA enable bit. The change is effective on the second clock cycle after the write. |

## Register 13: GPIO 8-mA Drive Select (GPIODR8R), offset 0x508

The **GPIODR8R** register is the 8-mA drive control register. It allows for each GPIO signal in the port to be individually configured without affecting the other pads. When writing the DRV8 bit for a GPIO signal, the corresponding DRV2 bit in the **GPIODR2R** register and the DRV4 bit in the **GPIODR4R** register are automatically cleared by hardware.

GPIO 8-mA Drive Select (GPIODR8R)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x508
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DRV8 | R/W | 0x00 | Output Pad 8-mA Drive Enable<br>A write of 1 to either **GPIODR2[n]** or **GPIODR4[n]** clears the corresponding 8-mA enable bit. The change is effective on the second clock cycle after the write. |

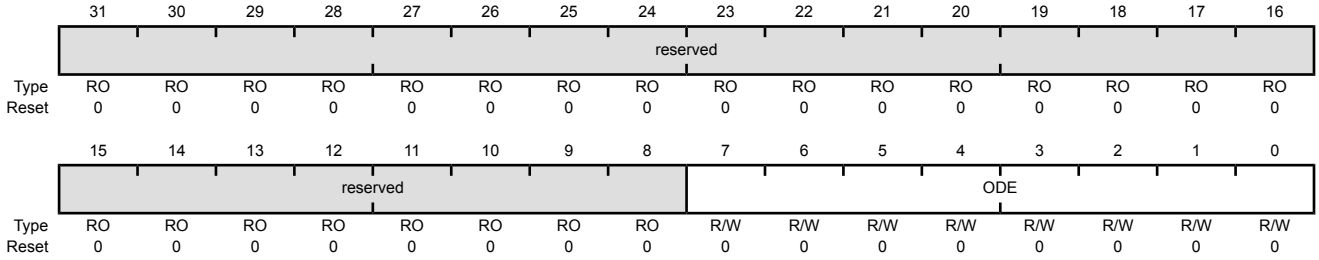
## Register 14: GPIO Open Drain Select (GPIOODR), offset 0x50C

The **GPIOODR** register is the open drain control register. Setting a bit in this register enables the open drain configuration of the corresponding GPIO pad. When open drain mode is enabled, the corresponding bit should also be set in the **GPIO Digital Input Enable (GPIODEN)** register (see page 246). Corresponding bits in the drive strength registers (**GPIODR2R**, **GPIODR4R**, **GPIODR8R**, and **GPIOSLR** ) can be set to achieve the desired rise and fall times. The GPIO acts as an open-drain input if the corresponding bit in the **GPIODIR** register is cleared. If open drain is selected while the GPIO is configured as an input, the GPIO will remain an input and the open-drain selection has no effect until the GPIO is changed to an output.

When using the $I^2C$ module, in addition to configuring the pin to open drain, the **GPIO Alternate Function Select (GPIOAFSEL)** register bits for the $I^2C$ clock and data pins should be set to 1 (see examples in “Initialization and Configuration” on page 224).

GPIO Open Drain Select (GPIOODR)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x50C
Type R/W, reset 0x0000.0000



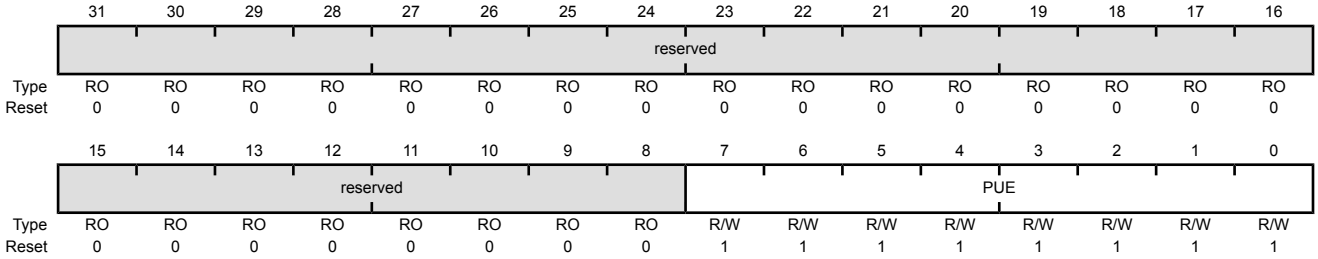

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | ODE | R/W | 0x00 | Output Pad Open Drain Enable<br>The ODE values are defined as follows:<br><br>Value Description<br>0 Open drain configuration is disabled.<br>1 Open drain configuration is enabled. |

## Register 15: GPIO Pull-Up Select (GPIOPUR), offset 0x510

The **GPIOPUR** register is the pull-up control register. When a bit is set to 1, it enables a weak pull-up resistor on the corresponding GPIO signal. Setting a bit in **GPIOPUR** automatically clears the corresponding bit in the **GPIO Pull-Down Select (GPIOPDR)** register (see page 244).

GPIO Pull-Up Select (GPIOPUR)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x510
Type R/W, reset 0x0000.00FF



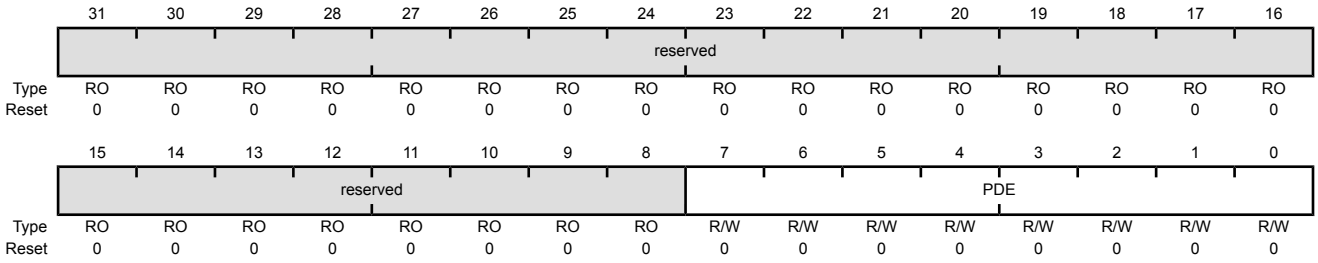

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PUE | R/W | 0xFF | Pad Weak Pull-Up Enable<br>A write of 1 to **GPIOPDR[n]** clears the corresponding **GPIOPUR[n]** enables. The change is effective on the second clock cycle after the write. |

## Register 16: GPIO Pull-Down Select (GPIOPDR), offset 0x514

The **GPIOPDR** register is the pull-down control register. When a bit is set to 1, it enables a weak pull-down resistor on the corresponding GPIO signal. Setting a bit in **GPIOPDR** automatically clears the corresponding bit in the **GPIO Pull-Up Select (GPIOPUR)** register (see page 243).

GPIO Pull-Down Select (GPIOPDR)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x514
Type R/W, reset 0x0000.0000



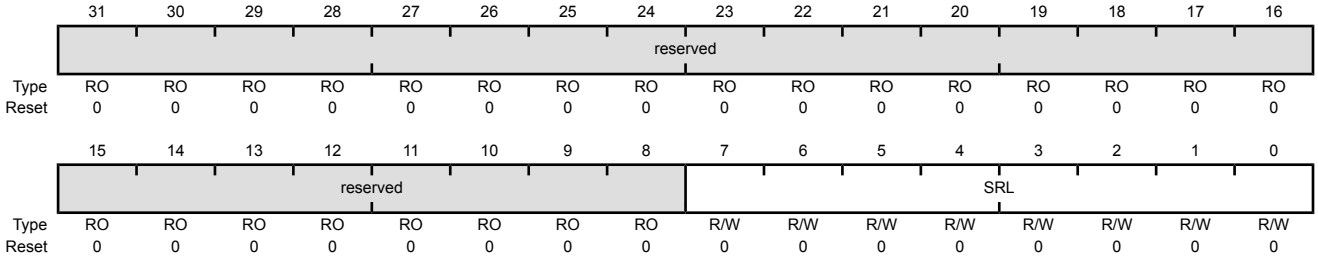

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PDE | R/W | 0x00 | Pad Weak Pull-Down Enable<br>A write of 1 to **GPIOPUR[n]** clears the corresponding **GPIOPDR[n]** enables. The change is effective on the second clock cycle after the write. |

## Register 17: GPIO Slew Rate Control Select (GPIOSLR), offset 0x518

The **GPIOSLR** register is the slew rate control register. Slew rate control is only available when using the 8-mA drive strength option via the **GPIO 8-mA Drive Select (GPIODR8R)** register (see page 241).

GPIO Slew Rate Control Select (GPIOSLR)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x518
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | SRL | R/W | 0x00 | Slew Rate Limit Enable (8-mA drive only)<br>The SRL values are defined as follows:<br><br>Value Description<br>0 Slew rate control disabled.<br>1 Slew rate control enabled. |

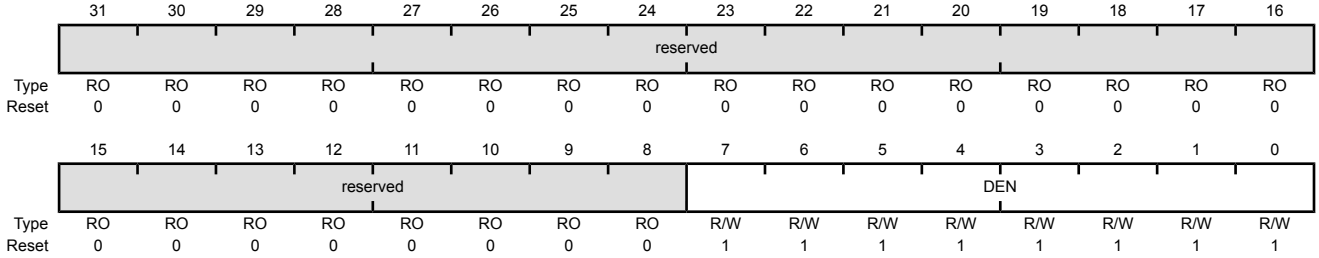
## Register 18: GPIO Digital Enable (GPIODEN), offset 0x51C

**Note:** Pins configured as digital inputs are Schmitt-triggered.

The **GPIODEN** register is the digital input enable register. By default, all GPIO signals are configured as digital inputs at reset. If a pin is being used as a GPIO or its Alternate Hardware Function, it should be configured as a digital input.

GPIO Digital Enable (GPIODEN)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0x51C
Type R/W, reset 0x0000.00FF



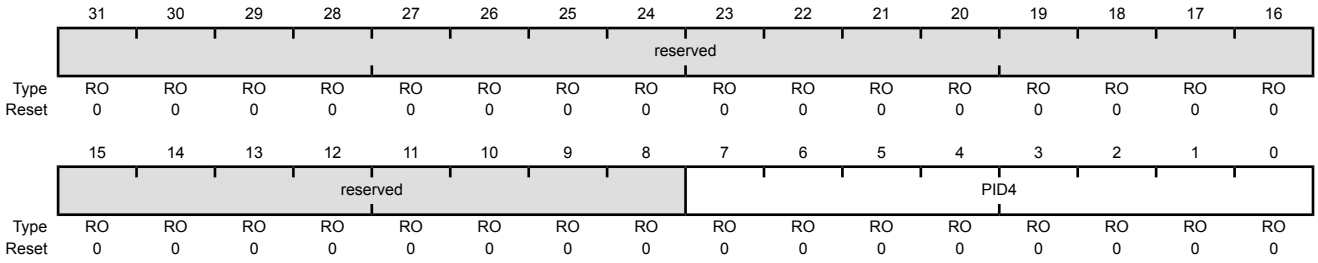

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DEN | R/W | 0xFF | Digital Enable<br>The DEN values are defined as follows:<br><br>Value Description<br>0 Digital functions disabled.<br>1 Digital functions enabled. |

## Register 19: GPIO Peripheral Identification 4 (GPIOPeriphID4), offset 0xFD0

The **GPIOPeriphID4**, **GPIOPeriphID5**, **GPIOPeriphID6**, and **GPIOPeriphID7** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 4 (GPIOPeriphID4)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFD0
Type RO, reset 0x0000.0000



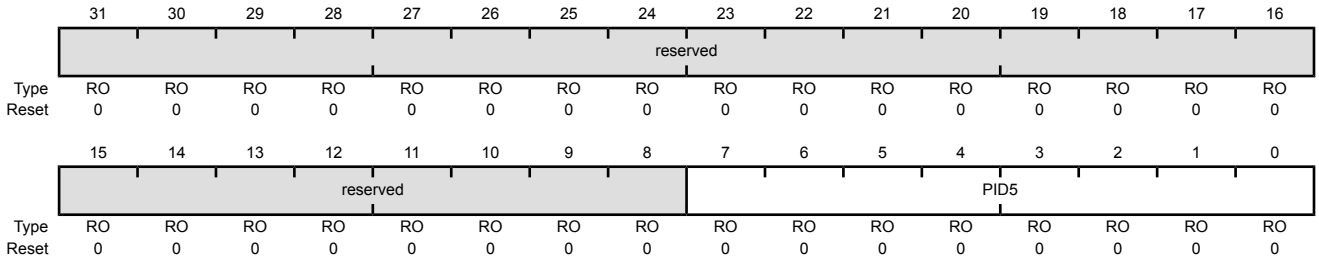

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID4 | RO | 0x00 | GPIO Peripheral ID Register[7:0] |

## Register 20: GPIO Peripheral Identification 5 (GPIOPeriphID5), offset 0xFD4

The **GPIOPeriphID4**, **GPIOPeriphID5**, **GPIOPeriphID6**, and **GPIOPeriphID7** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 5 (GPIOPeriphID5)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFD4
Type RO, reset 0x0000.0000



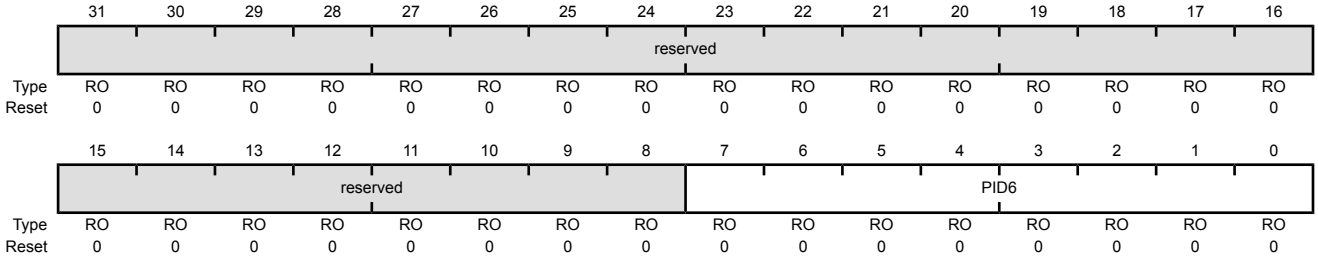

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID5 | RO | 0x00 | GPIO Peripheral ID Register[15:8] |

## Register 21: GPIO Peripheral Identification 6 (GPIOPeriphID6), offset 0xFD8

The **GPIOPeriphID4**, **GPIOPeriphID5**, **GPIOPeriphID6**, and **GPIOPeriphID7** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 6 (GPIOPeriphID6)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFD8
Type RO, reset 0x0000.0000



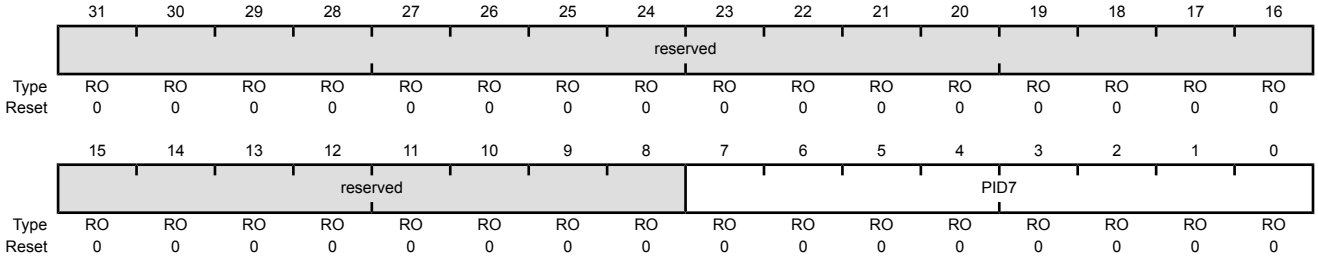

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID6 | RO | 0x00 | GPIO Peripheral ID Register[23:16] |

## Register 22: GPIO Peripheral Identification 7 (GPIOPeriphID7), offset 0xFDC

The **GPIOPeriphID4**, **GPIOPeriphID5**, **GPIOPeriphID6**, and **GPIOPeriphID7** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 7 (GPIOPeriphID7)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFDC
Type RO, reset 0x0000.0000



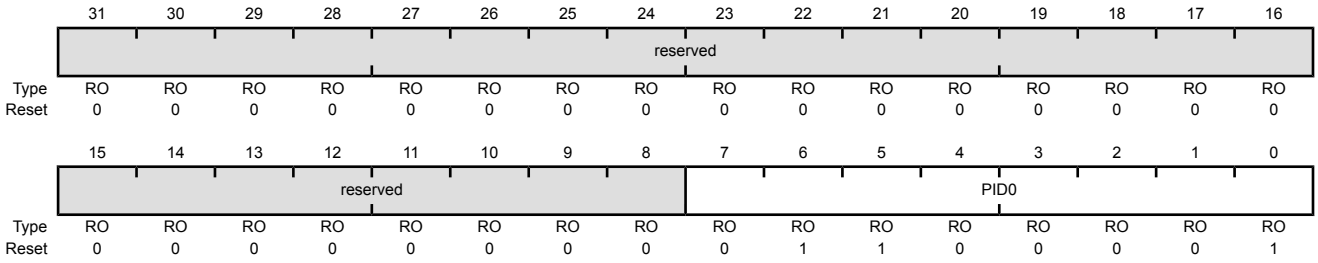

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID7 | RO | 0x00 | GPIO Peripheral ID Register[31:24] |

## Register 23: GPIO Peripheral Identification 0 (GPIOPeriphID0), offset 0xFE0

The **GPIOPeriphID0**, **GPIOPeriphID1**, **GPIOPeriphID2**, and **GPIOPeriphID3** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 0 (GPIOPeriphID0)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFE0
Type RO, reset 0x0000.0061



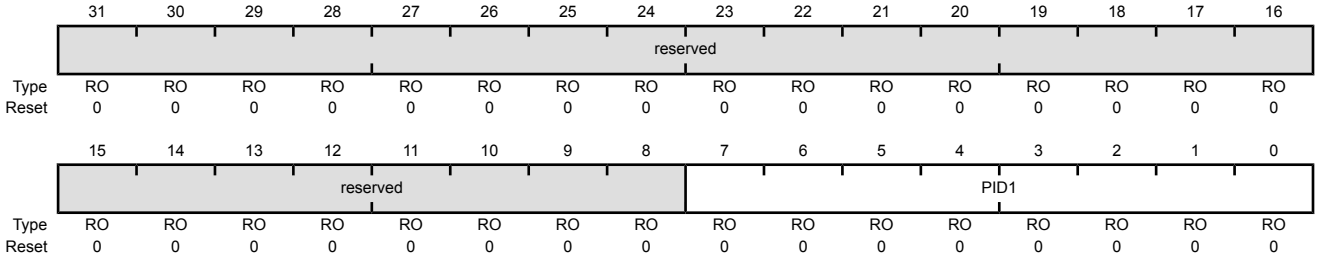

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID0 | RO | 0x61 | GPIO Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

## Register 24: GPIO Peripheral Identification 1 (GPIOPeriphID1), offset 0xFE4

The **GPIOPeriphID0**, **GPIOPeriphID1**, **GPIOPeriphID2**, and **GPIOPeriphID3** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 1 (GPIOPeriphID1)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFE4
Type RO, reset 0x0000.0000



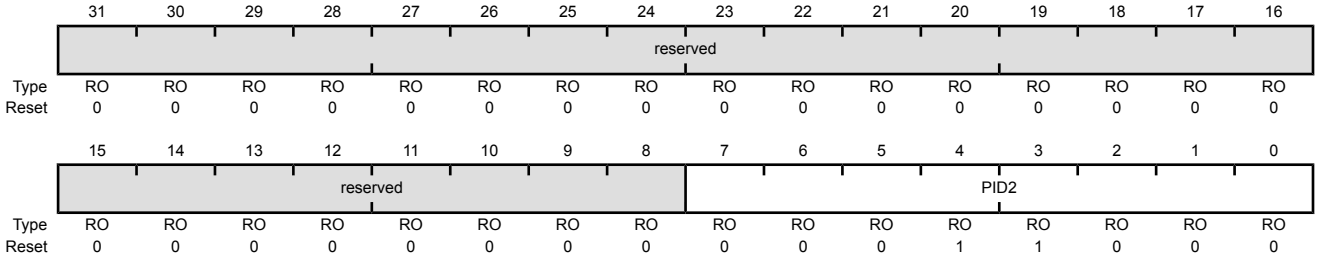

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID1 | RO | 0x00 | GPIO Peripheral ID Register[15:8]<br>Can be used by software to identify the presence of this peripheral. |

## Register 25: GPIO Peripheral Identification 2 (GPIOPeriphID2), offset 0xFE8

The **GPIOPeriphID0**, **GPIOPeriphID1**, **GPIOPeriphID2**, and **GPIOPeriphID3** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 2 (GPIOPeriphID2)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFE8
Type RO, reset 0x0000.0018



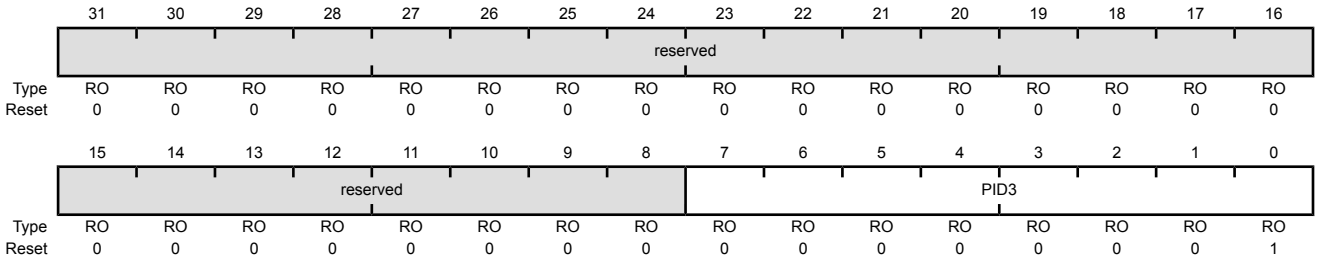

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID2 | RO | 0x18 | GPIO Peripheral ID Register[23:16]<br>Can be used by software to identify the presence of this peripheral. |

## Register 26: GPIO Peripheral Identification 3 (GPIOPeriphID3), offset 0xFEC

The **GPIOPeriphID0**, **GPIOPeriphID1**, **GPIOPeriphID2**, and **GPIOPeriphID3** registers can conceptually be treated as one 32-bit register; each register contains eight bits of the 32-bit register, used by software to identify the peripheral.

GPIO Peripheral Identification 3 (GPIOPeriphID3)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFEC
Type RO, reset 0x0000.0001



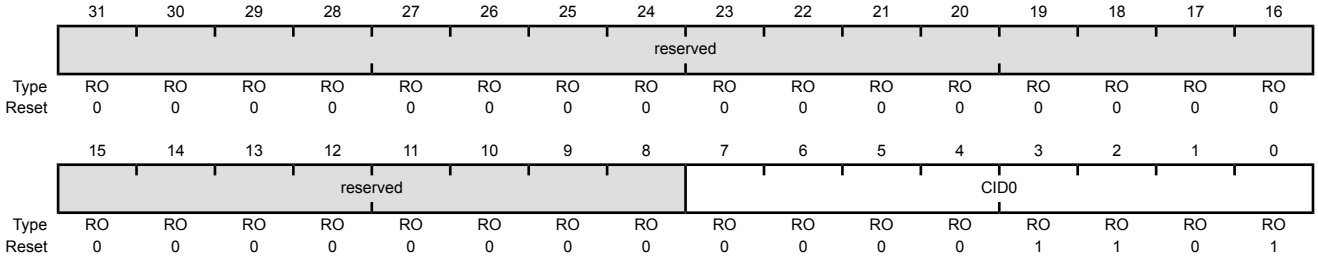

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID3 | RO | 0x01 | GPIO Peripheral ID Register[31:24]<br>Can be used by software to identify the presence of this peripheral. |

## Register 27: GPIO PrimeCell Identification 0 (GPIOPCellID0), offset 0xFF0

The **GPIOPCellID0**, **GPIOPCellID1**, **GPIOPCellID2**, and **GPIOPCellID3** registers are four 8-bit wide registers, that can conceptually be treated as one 32-bit register. The register is used as a standard cross-peripheral identification system.

GPIO PrimeCell Identification 0 (GPIOPCellID0)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFF0
Type RO, reset 0x0000.000D



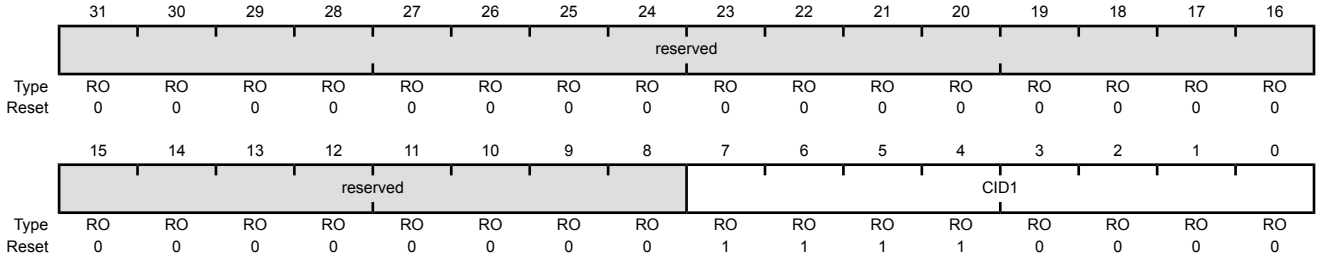

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID0 | RO | 0x0D | GPIO PrimeCell ID Register[7:0]<br>Provides software a standard cross-peripheral identification system. |

## Register 28: GPIO PrimeCell Identification 1 (GPIOPCellID1), offset 0xFF4

The **GPIOPCellID0**, **GPIOPCellID1**, **GPIOPCellID2**, and **GPIOPCellID3** registers are four 8-bit wide registers, that can conceptually be treated as one 32-bit register. The register is used as a standard cross-peripheral identification system.

GPIO PrimeCell Identification 1 (GPIOPCellID1)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFF4
Type RO, reset 0x0000.00F0



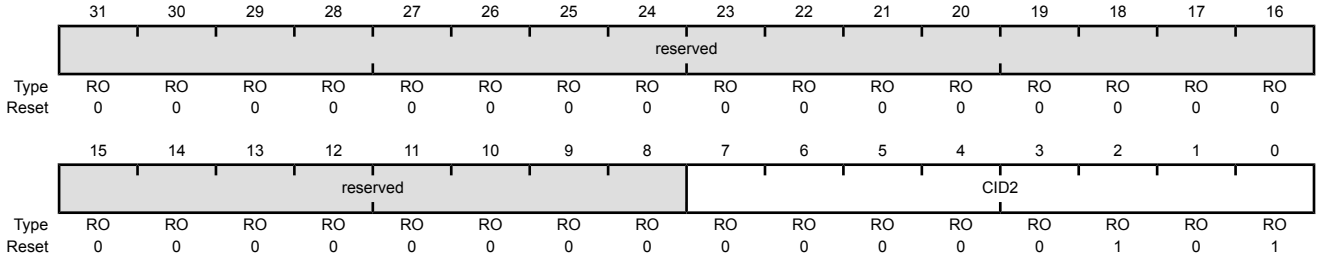

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID1 | RO | 0xF0 | GPIO PrimeCell ID Register[15:8]<br>Provides software a standard cross-peripheral identification system. |

## Register 29: GPIO PrimeCell Identification 2 (GPIOPCellID2), offset 0xFF8

The **GPIOPCellID0**, **GPIOPCellID1**, **GPIOPCellID2**, and **GPIOPCellID3** registers are four 8-bit wide registers, that can conceptually be treated as one 32-bit register. The register is used as a standard cross-peripheral identification system.

GPIO PrimeCell Identification 2 (GPIOPCellID2)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFF8
Type RO, reset 0x0000.0005



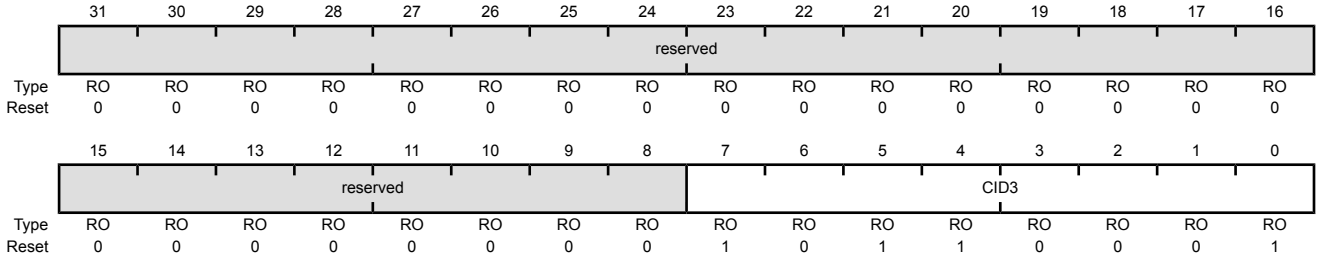

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID2 | RO | 0x05 | GPIO PrimeCell ID Register[23:16]<br>Provides software a standard cross-peripheral identification system. |

## Register 30: GPIO PrimeCell Identification 3 (GPIOPCellID3), offset 0xFFC

The **GPIOPCellID0**, **GPIOPCellID1**, **GPIOPCellID2**, and **GPIOPCellID3** registers are four 8-bit wide registers, that can conceptually be treated as one 32-bit register. The register is used as a standard cross-peripheral identification system.

GPIO PrimeCell Identification 3 (GPIOPCellID3)

GPIO Port A base: 0x4000.4000
GPIO Port B base: 0x4000.5000
GPIO Port C base: 0x4000.6000
GPIO Port D base: 0x4000.7000
GPIO Port E base: 0x4002.4000
Offset 0xFFC
Type RO, reset 0x0000.00B1




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID3 | RO | 0xB1 | GPIO PrimeCell ID Register[31:24]<br>Provides software a standard cross-peripheral identification system. |

# 8 General-Purpose Timers

Programmable timers can be used to count or time external events that drive the Timer input pins. The Stellaris® General-Purpose Timer Module (GPTM) contains three GPTM blocks (Timer0, Timer1, and Timer 2). Each GPTM block provides two 16-bit timers/counters (referred to as TimerA and TimerB) that can be configured to operate independently as timers or event counters, or configured to operate as one 32-bit timer or one 32-bit Real-Time Clock (RTC).

In addition, timers can be used to trigger analog-to-digital conversions (ADC). The ADC trigger signals from all of the general-purpose timers are ORed together before reaching the ADC module, so only one timer should be used to trigger ADC events.

The GPT Module is one timing resource available on the Stellaris microcontrollers. Other timer resources include the System Timer (SysTick) (see 81).

The General-Purpose Timers provide the following features:

- Three General-Purpose Timer Modules (GPTM), each of which provides two 16-bit timers/counters. Each GPTM can be configured to operate independently:
  - As a single 32-bit timer
  - As one 32-bit Real-Time Clock (RTC) to event capture
  - For Pulse Width Modulation (PWM)
  - To trigger analog-to-digital conversions
- 32-bit Timer modes
  - Programmable one-shot timer
  - Programmable periodic timer
  - Real-Time Clock when using an external 32.768-KHz clock as the input
  - User-enabled stalling when the controller asserts CPU Halt flag during debug
  - ADC event trigger
- 16-bit Timer modes
  - General-purpose timer function with an 8-bit prescaler (for one-shot and periodic modes only)
  - Programmable one-shot timer
  - Programmable periodic timer
  - User-enabled stalling when the controller asserts CPU Halt flag during debug
  - ADC event trigger
- 16-bit Input Capture modes
  - Input edge count capture

– Input edge time capture

■ 16-bit PWM mode

– Simple PWM mode with software-programmable output inversion of the PWM signal

## 8.1 Block Diagram

**Note:** In Figure 8-1 on page 260, the specific CCP pins available depend on the Stellaris device. See Table 8-1 on page 260 for the available CCPs.

**Figure 8-1. GPTM Module Block Diagram**

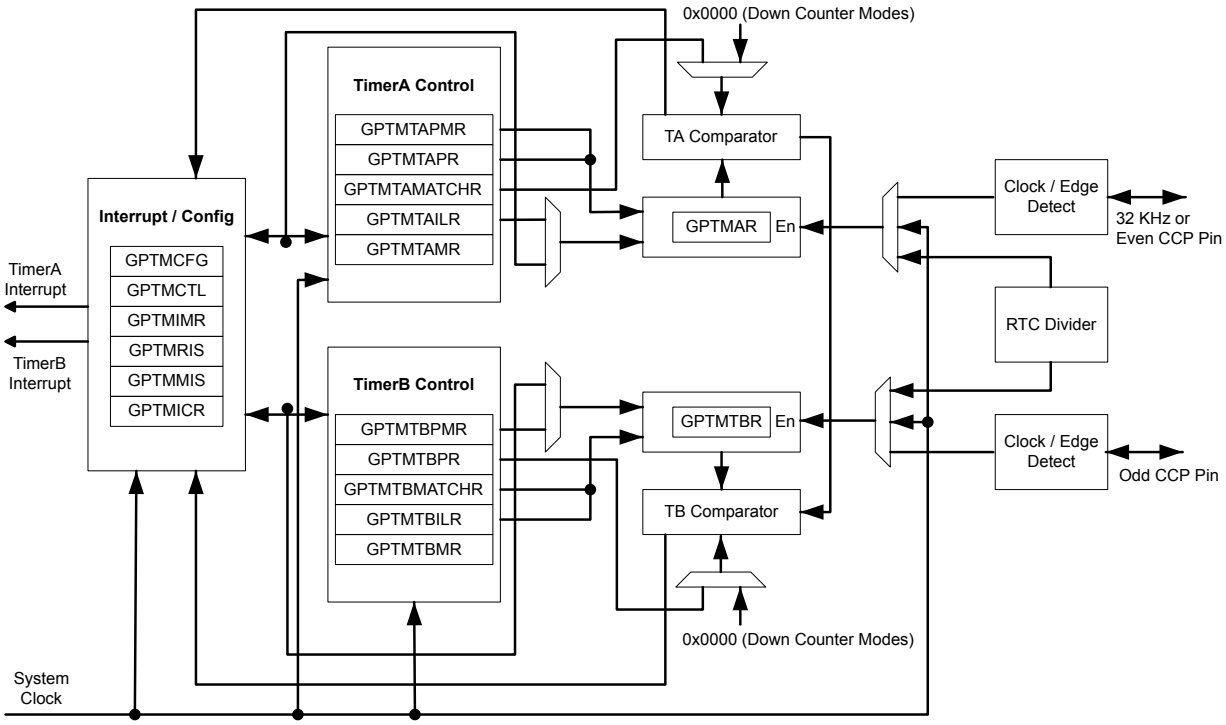



**Table 8-1. Available CCP Pins**

| Timer | 16-Bit Up/Down Counter | Even CCP Pin | Odd CCP Pin |
|---|---|---|---|
| Timer 0 | TimerA | CCP0 | - |
| | TimerB | - | CCP1 |
| Timer 1 | TimerA | CCP2 | - |
| | TimerB | - | CCP3 |
| Timer 2 | TimerA | CCP4 | - |
| | TimerB | - | CCP5 |

## 8.2 Functional Description

The main components of each GPTM block are two free-running 16-bit up/down counters (referred to as TimerA and TimerB), two 16-bit match registers, two prescaler match registers, and two 16-bit

load/initialization registers and their associated control functions. The exact functionality of each GPTM is controlled by software and configured through the register interface.

Software configures the GPTM using the **GPTM Configuration (GPTMCFG)** register (see page 271), the **GPTM TimerA Mode (GPTMTAMR)** register (see page 272), and the **GPTM TimerB Mode (GPTMTBMR)** register (see page 274). When in one of the 32-bit modes, the timer can only act as a 32-bit timer. However, when configured in 16-bit mode, the GPTM can have its two 16-bit timers configured in any combination of the 16-bit modes.

### 8.2.1 GPTM Reset Conditions

After reset has been applied to the GPTM module, the module is in an inactive state, and all control registers are cleared and in their default states. Counters TimerA and TimerB are initialized to 0xFFFF, along with their corresponding load registers: the **GPTM TimerA Interval Load (GPTMTAILR)** register (see page 285) and the **GPTM TimerB Interval Load (GPTMTBILR)** register (see page 286). The prescale counters are initialized to 0x00: the **GPTM TimerA Prescale (GPTMTAPR)** register (see page 289) and the **GPTM TimerB Prescale (GPTMTBPR)** register (see page 290).

### 8.2.2 32-Bit Timer Operating Modes

This section describes the three GPTM 32-bit timer modes (One-Shot, Periodic, and RTC) and their configuration.

The GPTM is placed into 32-bit mode by writing a 0 (One-Shot/Periodic 32-bit timer mode) or a 1 (RTC mode) to the **GPTM Configuration (GPTMCFG)** register. In both configurations, certain GPTM registers are concatenated to form pseudo 32-bit registers. These registers include:

- **GPTM TimerA Interval Load (GPTMTAILR)** register [15:0], see page 285
- **GPTM TimerB Interval Load (GPTMTBILR)** register [15:0], see page 286
- **GPTM TimerA (GPTMTAR)** register [15:0], see page 293
- **GPTM TimerB (GPTMTBR)** register [15:0], see page 294

In the 32-bit modes, the GPTM translates a 32-bit write access to **GPTMTAILR** into a write access to both **GPTMTAILR** and **GPTMTBILR**. The resulting word ordering for such a write operation is:

```
GPTMTBILR[15:0]:GPTMTAILR[15:0]
```

Likewise, a read access to **GPTMTAR** returns the value:

```
GPTMTBR[15:0]:GPTMTAR[15:0]
```

#### 8.2.2.1 32-Bit One-Shot/Periodic Timer Mode

In 32-bit one-shot and periodic timer modes, the concatenated versions of the TimerA and TimerB registers are configured as a 32-bit down-counter. The selection of one-shot or periodic mode is determined by the value written to the TAMR field of the **GPTM TimerA Mode (GPTMTAMR)** register (see page 272), and there is no need to write to the **GPTM TimerB Mode (GPTMTBMR)** register.

When software writes the TAEN bit in the **GPTM Control (GPTMCTL)** register (see page 276), the timer begins counting down from its preloaded value. Once the 0x0000.0000 state is reached, the timer reloads its start value from the concatenated **GPTMTAILR** on the next cycle. If configured to be a one-shot timer, the timer stops counting and clears the TAEN bit in the **GPTMCTL** register. If configured as a periodic timer, it continues counting.

In addition to reloading the count value, the GPTM generates interrupts and triggers when it reaches the 0x000.0000 state. The GPTM sets the TATORIS bit in the **GPTM Raw Interrupt Status (GPTMRIS)** register (see page 281), and holds it until it is cleared by writing the **GPTM Interrupt Clear (GPTMICR)** register (see page 283). If the time-out interrupt is enabled in the **GPTM Interrupt Mask (GPTMIMR)** register (see page 279), the GPTM also sets the TATOMIS bit in the **GPTM Masked Interrupt Status (GPTMMIS)** register (see page 282). The ADC trigger is enabled by setting the TAOTE bit in **GPTMCTL**.

If software reloads the **GPTMTAILR** register while the counter is running, the counter loads the new value on the next clock cycle and continues counting from the new value.

If the TASTALL bit in the **GPTMCTL** register is set, the timer freezes counting while the processor is halted by the debugger. The timer resumes counting when the processor resumes execution.

#### 8.2.2.2 32-Bit Real-Time Clock Timer Mode

In Real-Time Clock (RTC) mode, the concatenated versions of the TimerA and TimerB registers are configured as a 32-bit up-counter. When RTC mode is selected for the first time, the counter is loaded with a value of 0x0000.0001. All subsequent load values must be written to the **GPTM TimerA Match (GPTMTAMATCHR)** register (see page 287) by the controller.

The input clock on an even CCP input is required to be 32.768 KHz in RTC mode. The clock signal is then divided down to a 1 Hz rate and is passed along to the input of the 32-bit counter.

When software writes the TAEN bit inthe **GPTMCTL** register, the counter starts counting up from its preloaded value of 0x0000.0001. When the current count value matches the preloaded value in the **GPTMTAMATCHR** register, it rolls over to a value of 0x0000.0000 and continues counting until either a hardware reset, or it is disabled by software (clearing the TAEN bit). When a match occurs, the GPTM asserts the RTCRIS bit in **GPTMRIS**. If the RTC interrupt is enabled in **GPTMIMR**, the GPTM also sets the RTCMIS bit in **GPTMMIS** and generates a controller interrupt. The status flags are cleared by writing the RTCCINT bit in **GPTMICR**.

If the TASTALL and/or TBSTALL bits in the **GPTMCTL** register are set, the timer does not freeze if the RTCEN bit is set in **GPTMCTL**.

### 8.2.3 16-Bit Timer Operating Modes

The GPTM is placed into global 16-bit mode by writing a value of 0x4 to the **GPTM Configuration (GPTMCFG)** register (see page 271). This section describes each of the GPTM 16-bit modes of operation. TimerA and TimerB have identical modes, so a single description is given using an **n** to reference both.

#### 8.2.3.1 16-Bit One-Shot/Periodic Timer Mode

In 16-bit one-shot and periodic timer modes, the timer is configured as a 16-bit down-counter with an optional 8-bit prescaler that effectively extends the counting range of the timer to 24 bits. The selection of one-shot or periodic mode is determined by the value written to the TnMR field of the **GPTMTnMR** register. The optional prescaler is loaded into the **GPTM Timern Prescale (GPTMTnPR)** register.

When software writes the TnEN bit in the **GPTMCTL** register, the timer begins counting down from its preloaded value. Once the 0x0000 state is reached, the timer reloads its start value from **GPTMTnILR** and **GPTMTnPR** on the next cycle. If configured to be a one-shot timer, the timer stops counting and clears the TnEN bit in the **GPTMCTL** register. If configured as a periodic timer, it continues counting.

In addition to reloading the count value, the timer generates interrupts and triggers when it reaches the 0x0000 state. The GPTM sets the TnTORIS bit in the **GPTMRIS** register, and holds it until it is cleared by writing the **GPTMICR** register. If the time-out interrupt is enabled in **GPTMIMR**, the GPTM also sets the TnTOMIS bit in **GPTMISR** and generates a controller interrupt. The ADC trigger is enabled by setting the TnOTE bit in the **GPTMCTL** register.

If software reloads the **GPTMTAILR** register while the counter is running, the counter loads the new value on the next clock cycle and continues counting from the new value.

If the TnSTALL bit in the **GPTMCTL** register is set, the timer freezes counting while the processor is halted by the debugger. The timer resumes counting when the processor resumes execution.

The following example shows a variety of configurations for a 16-bit free running timer while using the prescaler. All values assume a 50-MHz clock with Tc=20 ns (clock period).

**Table 8-2. 16-Bit Timer With Prescaler Configurations**

| Prescale | #Clock (T c)[a] | Max Time | Units |
|---|---|---|---|
| 00000000 | 1 | 1.3107 | mS |
| 00000001 | 2 | 2.6214 | mS |
| 00000010 | 3 | 3.9322 | mS |
| ---------- | -- | -- | -- |
| 11111101 | 254 | 332.9229 | mS |
| 11111110 | 255 | 334.2336 | mS |
| 11111111 | 256 | 335.5443 | mS |

a. Tc is the clock period.

### 8.2.3.2 16-Bit Input Edge Count Mode

**Note:** For rising-edge detection, the input signal must be High for at least two system clock periods following the rising edge. Similarly, for falling-edge detection, the input signal must be Low for at least two system clock periods following the falling edge. Based on this criteria, the maximum input frequency for edge detection is 1/4 of the system frequency.

**Note:** The prescaler is not available in 16-Bit Input Edge Count mode.

In Edge Count mode, the timer is configured as a down-counter capable of capturing three types of events: rising edge, falling edge, or both. To place the timer in Edge Count mode, the TnCMR bit of the **GPTMTnMR** register must be set to 0. The type of edge that the timer counts is determined by the TnEVENT fields of the **GPTMCTL** register. During initialization, the **GPTM Timern Match (GPTMTnMATCHR)** register is configured so that the difference between the value in the **GPTMTnILR** register and the **GPTMTnMATCHR** register equals the number of edge events that must be counted.

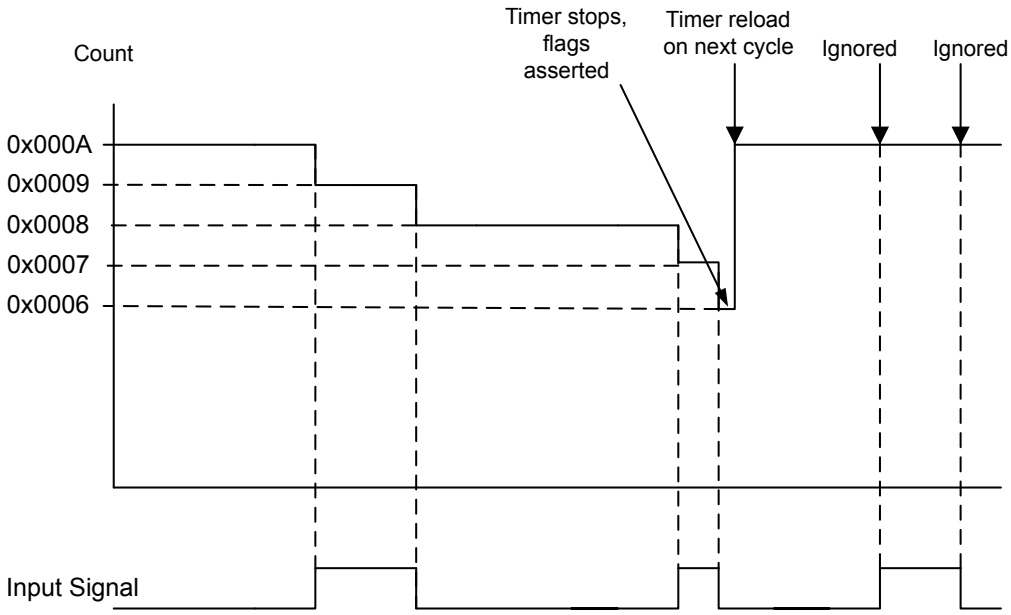
When software writes the TnEN bit in the **GPTM Control (GPTMCTL)** register, the timer is enabled for event capture. Each input event on the CCP pin decrements the counter by 1 until the event count matches **GPTMTnMATCHR**. When the counts match, the GPTM asserts the CnMRIS bit in the **GPTMRIS** register (and the CnMMIS bit, if the interrupt is not masked).

The counter is then reloaded using the value in **GPTMTnILR**, and stopped since the GPTM automatically clears the TnEN bit in the **GPTMCTL** register. Once the event count has been reached, all further events are ignored until TnEN is re-enabled by software.

Figure 8-2 on page 264 shows how input edge count mode works. In this case, the timer start value is set to **GPTMTnILR** =0x000A and the match value is set to **GPTMTnMATCHR** =0x0006 so that four edge events are counted. The counter is configured to detect both edges of the input signal.

Note that the last two edges are not counted since the timer automatically clears the TnEN bit after the current count matches the value in the **GPTMTnMATCHR** register.

**Figure 8-2. 16-Bit Input Edge Count Mode Example**




### 8.2.3.3 16-Bit Input Edge Time Mode

**Note:** For rising-edge detection, the input signal must be High for at least two system clock periods following the rising edge. Similarly, for falling edge detection, the input signal must be Low for at least two system clock periods following the falling edge. Based on this criteria, the maximum input frequency for edge detection is 1/4 of the system frequency.

**Note:** The prescaler is not available in 16-Bit Input Edge Time mode.

In Edge Time mode, the timer is configured as a free-running down-counter initialized to the value loaded in the **GPTMTnILR** register (or 0xFFFF at reset). This mode allows for event capture of either rising or falling edges, but not both. The timer is placed into Edge Time mode by setting the TnCMR bit in the **GPTMTnMR** register, and the type of event that the timer captures is determined by the TnEVENT fields of the **GPTMCTL** register.

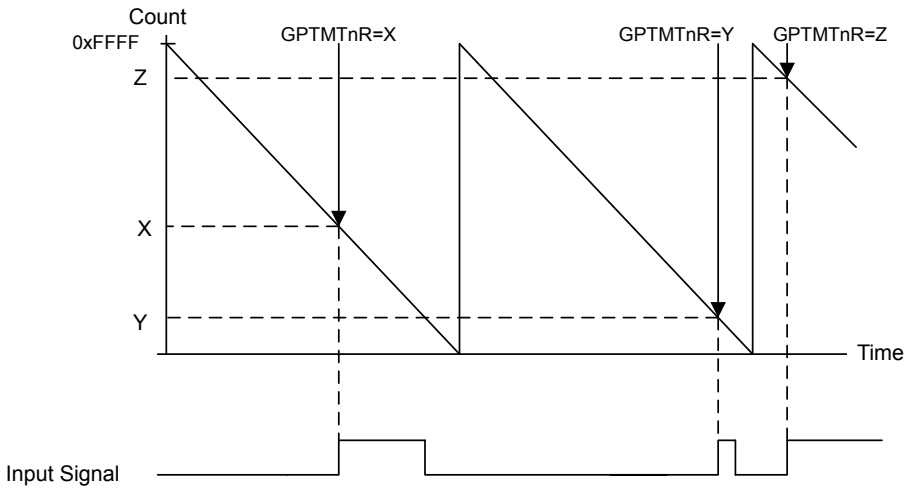
When software writes the TnEN bit in the **GPTMCTL** register, the timer is enabled for event capture. When the selected input event is detected, the current Tn counter value is captured in the **GPTMTnR** register and is available to be read by the controller. The GPTM then asserts the CnERIS bit (and the CnEMIS bit, if the interrupt is not masked).

After an event has been captured, the timer does not stop counting. It continues to count until the TnEN bit is cleared. When the timer reaches the 0x0000 state, it is reloaded with the value from the **GPTMTnILR** register.

Figure 8-3 on page 265 shows how input edge timing mode works. In the diagram, it is assumed that the start value of the timer is the default value of 0xFFFF, and the timer is configured to capture rising edge events.

Each time a rising edge event is detected, the current count value is loaded into the **GPTMTnR** register, and is held there until another rising edge is detected (at which point the new count value is loaded into **GPTMTnR**).

**Figure 8-3. 16-Bit Input Edge Time Mode Example**




### 8.2.3.4 16-Bit PWM Mode

**Note:** The prescaler is not available in 16-Bit PWM mode.

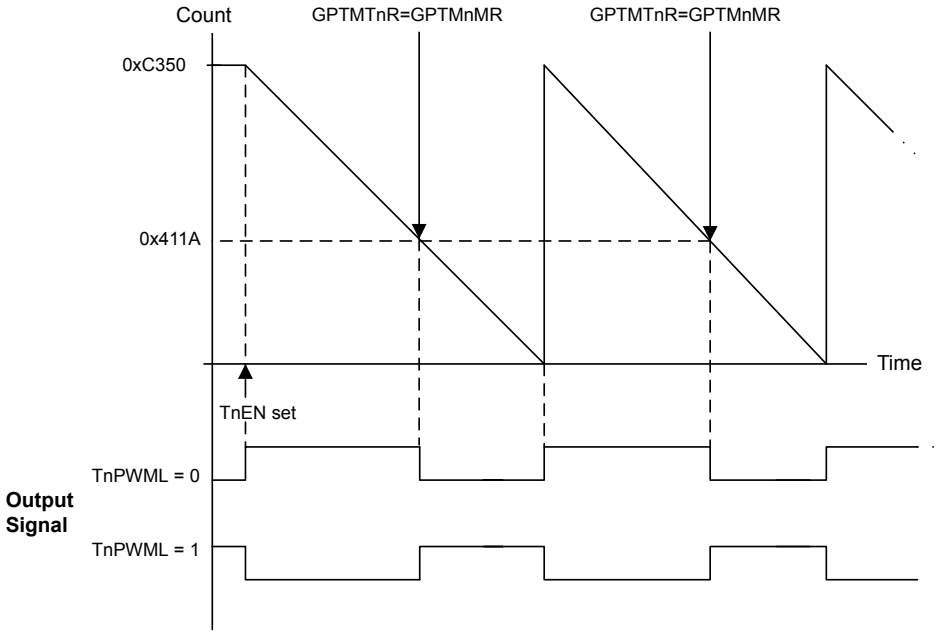
The GPTM supports a simple PWM generation mode. In PWM mode, the timer is configured as a down-counter with a start value (and thus period) defined by **GPTMTnILR**. In this mode, the PWM frequency and period are synchronous events and therefore guaranteed to be glitch free. PWM mode is enabled with the **GPTMTnMR** register by setting the TnAMS bit to 0x1, the TnCMR bit to 0x0, and the TnMR field to 0x2.

When software writes the TnEN bit in the **GPTMCTL** register, the counter begins counting down until it reaches the 0x0000 state. On the next counter cycle, the counter reloads its start value from **GPTMTnILR** and continues counting until disabled by software clearing the TnEN bit in the **GPTMCTL** register. No interrupts or status bits are asserted in PWM mode.

The output PWM signal asserts when the counter is at the value of the **GPTMTnILR** register (its start state), and is deasserted when the counter value equals the value in the **GPTM Timern Match Register (GPTMTnMATCHR)**. Software has the capability of inverting the output PWM signal by setting the TnPWML bit in the **GPTMCTL** register.

Figure 8-4 on page 266 shows how to generate an output PWM with a 1-ms period and a 66% duty cycle assuming a 50-MHz input clock and **TnPWML** =0 (duty cycle would be 33% for the **TnPWML** =1 configuration). For this example, the start value is **GPTMTnIRL**=0xC350 and the match value is **GPTMTnMATCHR**=0x411A.

**Figure 8-4. 16-Bit PWM Mode Example**




## 8.3 Initialization and Configuration

To use the general-purpose timers, the peripheral clock must be enabled by setting the TIMER0, TIMER1, and TIMER2 bits in the **RCGC1** register.

This section shows module initialization and configuration examples for each of the supported timer modes.

### 8.3.1 32-Bit One-Shot/Periodic Timer Mode

The GPTM is configured for 32-bit One-Shot and Periodic modes by the following sequence:

1. Ensure the timer is disabled (the TAEN bit in the **GPTMCTL** register is cleared) before making any changes.
2. Write the **GPTM Configuration Register (GPTMCFG)** with a value of 0x0.
3. Set the TAMR field in the **GPTM TimerA Mode Register (GPTMTAMR)**:
   a. Write a value of 0x1 for One-Shot mode.
   b. Write a value of 0x2 for Periodic mode.
4. Load the start value into the **GPTM TimerA Interval Load Register (GPTMTAILR)**.
5. If interrupts are required, set the TATOIM bit in the **GPTM Interrupt Mask Register (GPTMIMR)**.
6. Set the TAEN bit in the **GPTMCTL** register to enable the timer and start counting.

7. Poll the TATORIS bit in the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the TATOCINT bit of the **GPTM Interrupt Clear Register (GPTMICR)**.

In One-Shot mode, the timer stops counting after step 7 on page 267. To re-enable the timer, repeat the sequence. A timer configured in Periodic mode does not stop counting after it times out.

### 8.3.2 32-Bit Real-Time Clock (RTC) Mode

To use the RTC mode, the timer must have a 32.768-KHz input signal on an even CCP input. To enable the RTC feature, follow these steps:

1. Ensure the timer is disabled (the TAEN bit is cleared) before making any changes.
2. Write the **GPTM Configuration Register (GPTMCFG)** with a value of 0x1.
3. Write the desired match value to the **GPTM TimerA Match Register (GPTMTAMATCHR)**.
4. Set/clear the RTCEN bit in the **GPTM Control Register (GPTMCTL)** as desired.
5. If interrupts are required, set the RTCIM bit in the **GPTM Interrupt Mask Register (GPTMIMR)**.
6. Set the TAEN bit in the **GPTMCTL** register to enable the timer and start counting.

When the timer count equals the value in the **GPTMTAMATCHR** register, the GPTM asserts the RTCRIS bit in the **GPTMRIS** register and continues counting until Timer A is disabled or a hardware reset. The interrupt is cleared by writing the RTCCINT bit in the **GPTMICR** register.

### 8.3.3 16-Bit One-Shot/Periodic Timer Mode

A timer is configured for 16-bit One-Shot and Periodic modes by the following sequence:

1. Ensure the timer is disabled (the TnEN bit is cleared) before making any changes.
2. Write the **GPTM Configuration Register (GPTMCFG)** with a value of 0x4.
3. Set the TnMR field in the **GPTM Timer Mode (GPTMTnMR)** register:
   a. Write a value of 0x1 for One-Shot mode.
   b. Write a value of 0x2 for Periodic mode.
4. If a prescaler is to be used, write the prescale value to the **GPTM Timern Prescale Register (GPTMTnPR)**.
5. Load the start value into the **GPTM Timer Interval Load Register (GPTMTnILR)**.
6. If interrupts are required, set the TnTOIM bit in the **GPTM Interrupt Mask Register (GPTMIMR)**.
7. Set the TnEN bit in the **GPTM Control Register (GPTMCTL)** to enable the timer and start counting.
8. Poll the TnTORIS bit in the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the TnTOCINT bit of the **GPTM Interrupt Clear Register (GPTMICR)**.

In One-Shot mode, the timer stops counting after step 8 on page 267. To re-enable the timer, repeat the sequence. A timer configured in Periodic mode does not stop counting after it times out.

### 8.3.4 16-Bit Input Edge Count Mode

A timer is configured to Input Edge Count mode by the following sequence:

1. Ensure the timer is disabled (the TnEN bit is cleared) before making any changes.
2. Write the **GPTM Configuration (GPTMCFG)** register with a value of 0x4.
3. In the **GPTM Timer Mode (GPTMTnMR)** register, write the TnCMR field to 0x0 and the TnMR field to 0x3.
4. Configure the type of event(s) that the timer captures by writing the TnEVENT field of the **GPTM Control (GPTMCTL)** register.
5. Load the timer start value into the **GPTM Timern Interval Load (GPTMTnILR)** register.
6. Load the desired event count into the **GPTM Timern Match (GPTMTnMATCHR)** register.
7. If interrupts are required, set the CnMIM bit in the **GPTM Interrupt Mask (GPTMIMR)** register.
8. Set the TnEN bit in the **GPTMCTL** register to enable the timer and begin waiting for edge events.
9. Poll the CnMRIS bit in the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the CnMCINT bit of the **GPTM Interrupt Clear (GPTMICR)** register.

In Input Edge Count Mode, the timer stops after the desired number of edge events has been detected. To re-enable the timer, ensure that the TnEN bit is cleared and repeat step 4 on page 268 through step 9 on page 268.

### 8.3.5 16-Bit Input Edge Timing Mode

A timer is configured to Input Edge Timing mode by the following sequence:

1. Ensure the timer is disabled (the TnEN bit is cleared) before making any changes.
2. Write the **GPTM Configuration (GPTMCFG)** register with a value of 0x4.
3. In the **GPTM Timer Mode (GPTMTnMR)** register, write the TnCMR field to 0x1 and the TnMR field to 0x3.
4. Configure the type of event that the timer captures by writing the TnEVENT field of the **GPTM Control (GPTMCTL)** register.
5. Load the timer start value into the **GPTM Timern Interval Load (GPTMTnILR)** register.
6. If interrupts are required, set the CnEIM bit in the **GPTM Interrupt Mask (GPTMIMR)** register.
7. Set the TnEN bit in the **GPTM Control (GPTMCTL)** register to enable the timer and start counting.
8. Poll the CnERIS bit in the **GPTMRIS** register or wait for the interrupt to be generated (if enabled). In both cases, the status flags are cleared by writing a 1 to the CnECINT bit of the **GPTM**

**Interrupt Clear (GPTMICR)** register. The time at which the event happened can be obtained by reading the **GPTM Timern (GPTMTnR)** register.

In Input Edge Timing mode, the timer continues running after an edge event has been detected, but the timer interval can be changed at any time by writing the **GPTMTnILR** register. The change takes effect at the next cycle after the write.

### 8.3.6 16-Bit PWM Mode

A timer is configured to PWM mode using the following sequence:

1. Ensure the timer is disabled (the TnEN bit is cleared) before making any changes.
2. Write the **GPTM Configuration (GPTMCFG)** register with a value of 0x4.
3. In the **GPTM Timer Mode (GPTMTnMR)** register, set the TnAMS bit to 0x1, the TnCMR bit to 0x0, and the TnMR field to 0x2.
4. Configure the output state of the PWM signal (whether or not it is inverted) in the TnPWML field of the **GPTM Control (GPTMCTL)** register.
5. Load the timer start value into the **GPTM Timern Interval Load (GPTMTnILR)** register.
6. Load the **GPTM Timern Match (GPTMTnMATCHR)** register with the desired value.
7. Set the TnEN bit in the **GPTM Control (GPTMCTL)** register to enable the timer and begin generation of the output PWM signal.

In PWM Timing mode, the timer continues running after the PWM signal has been generated. The PWM period can be adjusted at any time by writing the **GPTMTnILR** register, and the change takes effect at the next cycle after the write.

## 8.4 Register Map

Table 8-3 on page 269 lists the GPTM registers. The offset listed is a hexadecimal increment to the register's address, relative to that timer's base address:

- Timer0: 0x4003.0000
- Timer1: 0x4003.1000
- Timer2: 0x4003.2000

Note that the Timer module clock must be enabled before the registers can be programmed (see page 191). There must be a delay of 3 system clocks after the Timer module clock is enabled before any Timer module registers are accessed.

**Table 8-3. Timers Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | GPTMCFG | R/W | 0x0000.0000 | GPTM Configuration | 271 |
| 0x004 | GPTMTAMR | R/W | 0x0000.0000 | GPTM TimerA Mode | 272 |
| 0x008 | GPTMTBMR | R/W | 0x0000.0000 | GPTM TimerB Mode | 274 |
| 0x00C | GPTMCTL | R/W | 0x0000.0000 | GPTM Control | 276 |

**Table 8-3. Timers Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x018 | GPTMIMR | R/W | 0x0000.0000 | GPTM Interrupt Mask | 279 |
| 0x01C | GPTMRIS | RO | 0x0000.0000 | GPTM Raw Interrupt Status | 281 |
| 0x020 | GPTMMIS | RO | 0x0000.0000 | GPTM Masked Interrupt Status | 282 |
| 0x024 | GPTMICR | W1C | 0x0000.0000 | GPTM Interrupt Clear | 283 |
| 0x028 | GPTMTAILR | R/W | 0xFFFF.FFFF | GPTM TimerA Interval Load | 285 |
| 0x02C | GPTMTBILR | R/W | 0x0000.FFFF | GPTM TimerB Interval Load | 286 |
| 0x030 | GPTMTAMATCHR | R/W | 0xFFFF.FFFF | GPTM TimerA Match | 287 |
| 0x034 | GPTMTBMATCHR | R/W | 0x0000.FFFF | GPTM TimerB Match | 288 |
| 0x038 | GPTMTAPR | R/W | 0x0000.0000 | GPTM TimerA Prescale | 289 |
| 0x03C | GPTMTBPR | R/W | 0x0000.0000 | GPTM TimerB Prescale | 290 |
| 0x040 | GPTMTAPMR | R/W | 0x0000.0000 | GPTM TimerA Prescale Match | 291 |
| 0x044 | GPTMTBPMR | R/W | 0x0000.0000 | GPTM TimerB Prescale Match | 292 |
| 0x048 | GPTMTAR | RO | 0xFFFF.FFFF | GPTM TimerA | 293 |
| 0x04C | GPTMTBR | RO | 0x0000.FFFF | GPTM TimerB | 294 |

## 8.5 Register Descriptions

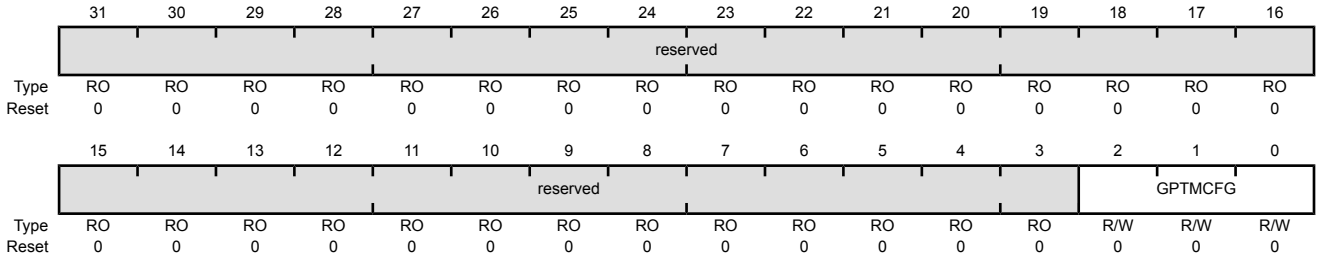
The remainder of this section lists and describes the GPTM registers, in numerical order by address offset.

## Register 1: GPTM Configuration (GPTMCFG), offset 0x000

This register configures the global operation of the GPTM module. The value written to this register determines whether the GPTM is in 32- or 16-bit mode.

GPTM Configuration (GPTMCFG)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x000
Type R/W, reset 0x0000.0000



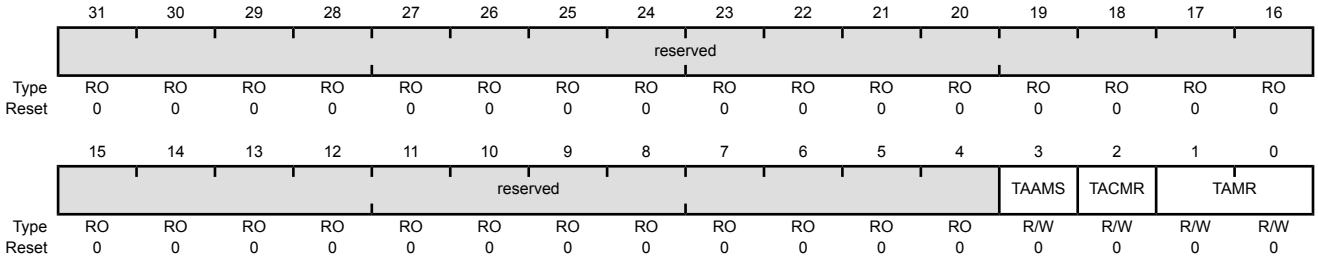

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | GPTMCFG | R/W | 0x0 | GPTM Configuration<br>The GPTMCFG values are defined as follows:<br><br>Value Description<br>0x0 32-bit timer configuration.<br>0x1 32-bit real-time clock (RTC) counter configuration.<br>0x2 Reserved<br>0x3 Reserved<br>0x4-0x7 16-bit timer configuration, function is controlled by bits 1:0 of **GPTMTAMR** and **GPTMTBMR**. |

## Register 2: GPTM TimerA Mode (GPTMTAMR), offset 0x004

This register configures the GPTM based on the configuration selected in the **GPTMCFG** register. When in 16-bit PWM mode, set the TAAMS bit to 0x1, the TACMR bit to 0x0, and the TAMR field to 0x2.

GPTM TimerA Mode (GPTMTAMR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x004
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TAAMS | R/W | 0 | GPTM TimerA Alternate Mode Select<br>The TAAMS values are defined as follows:<br><br>Value Description<br>0 Capture mode is enabled.<br>1 PWM mode is enabled.<br><br>**Note:** To enable PWM mode, you must also clear the TACMR bit and set the TAMR field to 0x2. |
| 2 | TACMR | R/W | 0 | GPTM TimerA Capture Mode<br>The TACMR values are defined as follows:<br><br>Value Description<br>0 Edge-Count mode<br>1 Edge-Time mode |

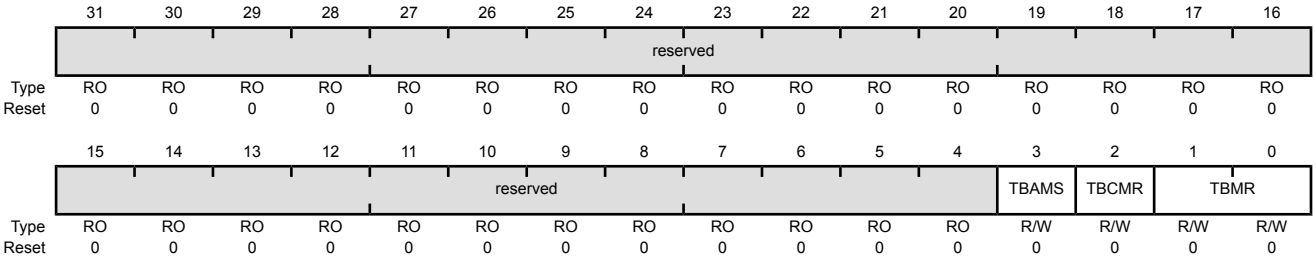
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1:0 | TAMR | R/W | 0x0 | GPTM TimerA Mode<br>The TAMR values are defined as follows:<br><br>Value Description<br>0x0 Reserved<br>0x1 One-Shot Timer mode<br>0x2 Periodic Timer mode<br>0x3 Capture mode<br><br>The Timer mode is based on the timer configuration defined by bits 2:0 in the **GPTMCFG** register (16-or 32-bit).<br>In 16-bit timer configuration, TAMR controls the 16-bit timer modes for TimerA.<br>In 32-bit timer configuration, this register controls the mode and the contents of **GPTMTBMR** are ignored. |

## Register 3: GPTM TimerB Mode (GPTMTBMR), offset 0x008

This register configures the GPTM based on the configuration selected in the **GPTMCFG** register. When in 16-bit PWM mode, set the TBAMS bit to 0x1, the TBCMR bit to 0x0, and the TBMR field to 0x2.

GPTM TimerB Mode (GPTMTBMR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x008
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TBAMS | R/W | 0 | GPTM TimerB Alternate Mode Select<br>The TBAMS values are defined as follows:<br><br>Value Description<br>0 Capture mode is enabled.<br>1 PWM mode is enabled.<br><br>**Note:** To enable PWM mode, you must also clear the TBCMR bit and set the TBMR field to 0x2. |
| 2 | TBCMR | R/W | 0 | GPTM TimerB Capture Mode<br>The TBCMR values are defined as follows:<br><br>Value Description<br>0 Edge-Count mode<br>1 Edge-Time mode |

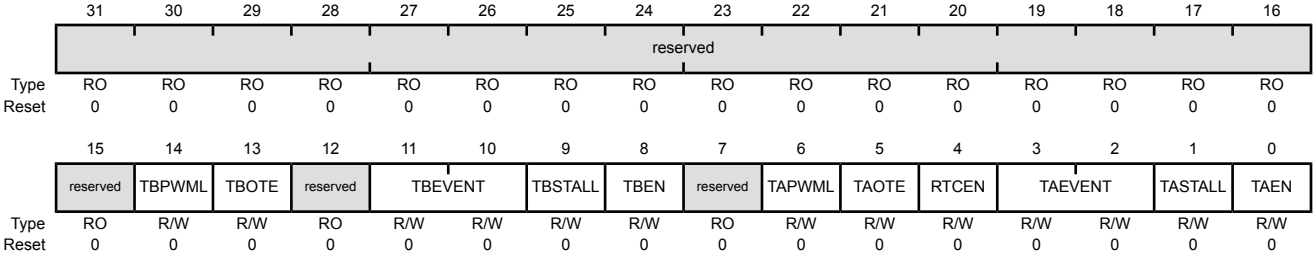
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1:0 | TBMR | R/W | 0x0 | GPTM TimerB Mode<br>The TBMR values are defined as follows:<br><br>Value Description<br>0x0 Reserved<br>0x1 One-Shot Timer mode<br>0x2 Periodic Timer mode<br>0x3 Capture mode<br><br>The timer mode is based on the timer configuration defined by bits 2:0 in the **GPTMCFG** register.<br>In 16-bit timer configuration, these bits control the 16-bit timer modes for TimerB.<br>In 32-bit timer configuration, this register's contents are ignored and **GPTMTAMR** is used. |

## Register 4: GPTM Control (GPTMCTL), offset 0x00C

This register is used alongside the **GPTMCFG** and **GMTMTnMR** registers to fine-tune the timer configuration, and to enable other features such as timer stall and the output trigger. The output trigger can be used to initiate transfers on the ADC module.

GPTM Control (GPTMCTL)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x00C
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:15 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14 | TBPWML | R/W | 0 | GPTM TimerB PWM Output Level<br>The TBPWML values are defined as follows:<br><br>Value Description<br>0 Output is unaffected.<br>1 Output is inverted. |
| 13 | TBOTE | R/W | 0 | GPTM TimerB Output Trigger Enable<br>The TBOTE values are defined as follows:<br><br>Value Description<br>0 The output TimerB ADC trigger is disabled.<br>1 The output TimerB ADC trigger is enabled.<br><br>In addition, the ADC must be enabled and the timer selected as a trigger source with the EMn bit in the **ADCEMUX** register (see page 333). |
| 12 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 11:10 | TBEVENT | R/W | 0x0 | GPTM TimerB Event Mode<br>The TBEVENT values are defined as follows:<br><br>Value Description<br>0x0 Positive edge<br>0x1 Negative edge<br>0x2 Reserved<br>0x3 Both edges |
| 9 | TBSTALL | R/W | 0 | GPTM Timer B Stall Enable<br>The TBSTALL values are defined as follows:<br><br>Value Description<br>0 Timer B continues counting while the processor is halted by the debugger.<br>1 Timer B freezes counting while the processor is halted by the debugger.<br><br>If the processor is executing normally, the TBSTALL bit is ignored. |
| 8 | TBEN | R/W | 0 | GPTM TimerB Enable<br>The TBEN values are defined as follows:<br><br>Value Description<br>0 TimerB is disabled.<br>1 TimerB is enabled and begins counting or the capture logic is enabled based on the **GPTMCFG** register. |
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | TAPWML | R/W | 0 | GPTM TimerA PWM Output Level<br>The TAPWML values are defined as follows:<br><br>Value Description<br>0 Output is unaffected.<br>1 Output is inverted. |
| 5 | TAOTE | R/W | 0 | GPTM TimerA Output Trigger Enable<br>The TAOTE values are defined as follows:<br><br>Value Description<br>0 The output TimerA ADC trigger is disabled.<br>1 The output TimerA ADC trigger is enabled.<br><br>In addition, the ADC must be enabled and the timer selected as a trigger source with the EMn bit in the **ADCEMUX** register (see page 333). |

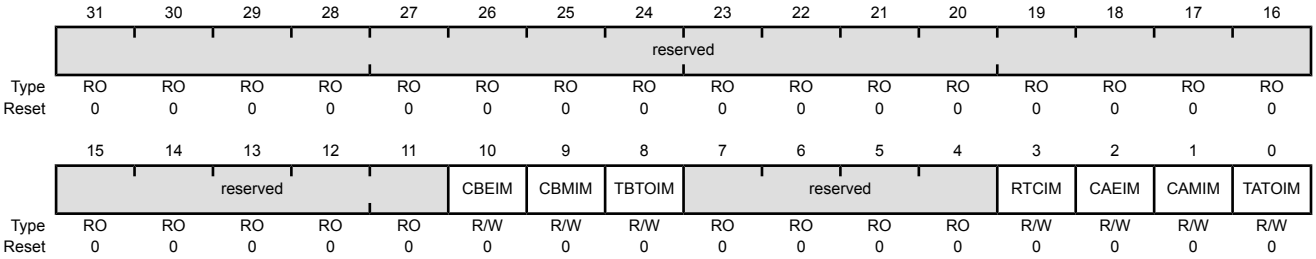
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 4 | RTCEN | R/W | 0 | GPTM RTC Enable<br>The RTCEN values are defined as follows:<br><br>Value Description<br>0 RTC counting is disabled.<br>1 RTC counting is enabled. |
| 3:2 | TAEVENT | R/W | 0x0 | GPTM TimerA Event Mode<br>The TAEVENT values are defined as follows:<br><br>Value Description<br>0x0 Positive edge<br>0x1 Negative edge<br>0x2 Reserved<br>0x3 Both edges |
| 1 | TASTALL | R/W | 0 | GPTM Timer A Stall Enable<br>The TASTALL values are defined as follows:<br><br>Value Description<br>0 Timer A continues counting while the processor is halted by the debugger.<br>1 Timer A freezes counting while the processor is halted by the debugger.<br><br>If the processor is executing normally, the TASTALL bit is ignored. |
| 0 | TAEN | R/W | 0 | GPTM TimerA Enable<br>The TAEN values are defined as follows:<br><br>Value Description<br>0 TimerA is disabled.<br>1 TimerA is enabled and begins counting or the capture logic is enabled based on the **GPTMCFG** register. |

## Register 5: GPTM Interrupt Mask (GPTMIMR), offset 0x018

This register allows software to enable/disable GPTM controller-level interrupts. Writing a 1 enables the interrupt, while writing a 0 disables it.

GPTM Interrupt Mask (GPTMIMR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x018
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | CBEIM | R/W | 0 | GPTM CaptureB Event Interrupt Mask<br>The CBEIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 9 | CBMIM | R/W | 0 | GPTM CaptureB Match Interrupt Mask<br>The CBMIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 8 | TBTOIM | R/W | 0 | GPTM TimerB Time-Out Interrupt Mask<br>The TBTOIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 7:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

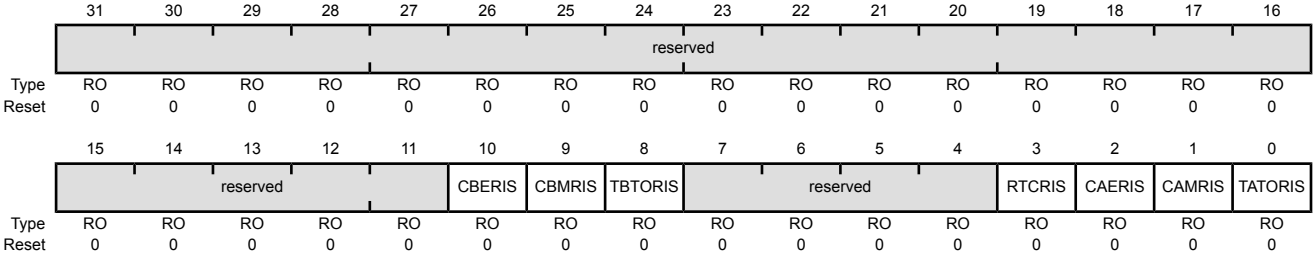
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | RTCIM | R/W | 0 | GPTM RTC Interrupt Mask<br>The RTCIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 2 | CAEIM | R/W | 0 | GPTM CaptureA Event Interrupt Mask<br>The CAEIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 1 | CAMIM | R/W | 0 | GPTM CaptureA Match Interrupt Mask<br>The CAMIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |
| 0 | TATOIM | R/W | 0 | GPTM TimerA Time-Out Interrupt Mask<br>The TATOIM values are defined as follows:<br><br>Value Description<br>0 Interrupt is disabled.<br>1 Interrupt is enabled. |

## Register 6: GPTM Raw Interrupt Status (GPTMRIS), offset 0x01C

This register shows the state of the GPTM's internal interrupt signal. These bits are set whether or not the interrupt is masked in the **GPTMIMR** register. Each bit can be cleared by writing a 1 to its corresponding bit in **GPTMICR**.

GPTM Raw Interrupt Status (GPTMRIS)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x01C
Type RO, reset 0x0000.0000



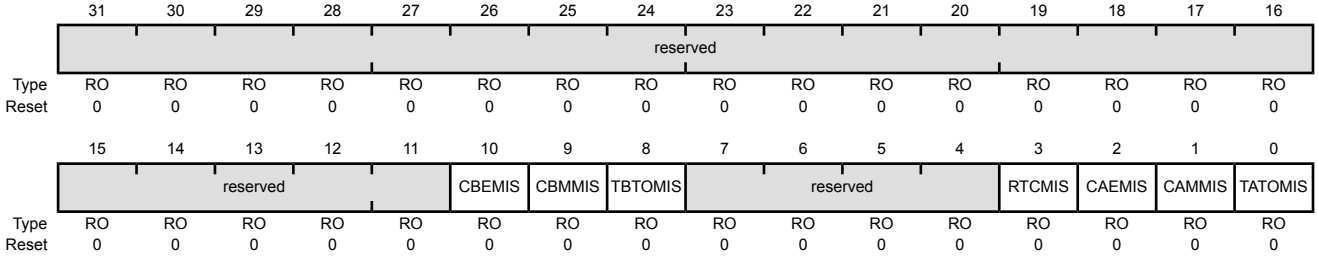

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | CBERIS | RO | 0 | GPTM CaptureB Event Raw Interrupt<br>This is the CaptureB Event interrupt status prior to masking. |
| 9 | CBMRIS | RO | 0 | GPTM CaptureB Match Raw Interrupt<br>This is the CaptureB Match interrupt status prior to masking. |
| 8 | TBTORIS | RO | 0 | GPTM TimerB Time-Out Raw Interrupt<br>This is the TimerB time-out interrupt status prior to masking. |
| 7:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | RTCRIS | RO | 0 | GPTM RTC Raw Interrupt<br>This is the RTC Event interrupt status prior to masking. |
| 2 | CAERIS | RO | 0 | GPTM CaptureA Event Raw Interrupt<br>This is the CaptureA Event interrupt status prior to masking. |
| 1 | CAMRIS | RO | 0 | GPTM CaptureA Match Raw Interrupt<br>This is the CaptureA Match interrupt status prior to masking. |
| 0 | TATORIS | RO | 0 | GPTM TimerA Time-Out Raw Interrupt<br>This the TimerA time-out interrupt status prior to masking. |

## Register 7: GPTM Masked Interrupt Status (GPTMMIS), offset 0x020

This register show the state of the GPTM's controller-level interrupt. If an interrupt is unmasked in **GPTMIMR**, and there is an event that causes the interrupt to be asserted, the corresponding bit is set in this register. All bits are cleared by writing a 1 to the corresponding bit in **GPTMICR**.

GPTM Masked Interrupt Status (GPTMMIS)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x020
Type RO, reset 0x0000.0000



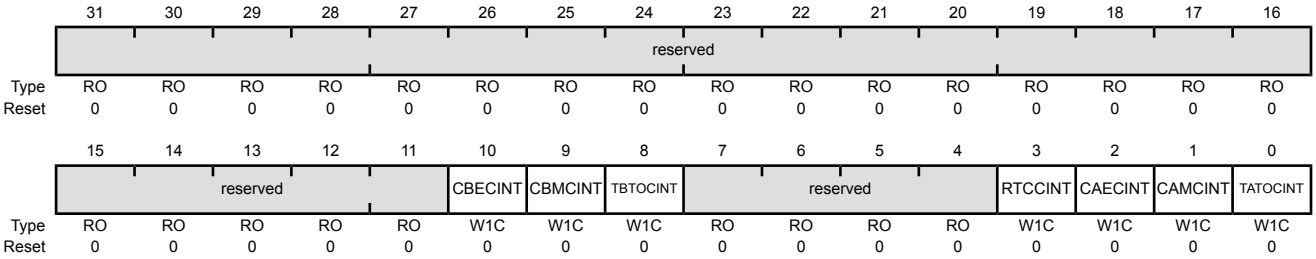

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | CBEMIS | RO | 0 | GPTM CaptureB Event Masked Interrupt<br>This is the CaptureB event interrupt status after masking. |
| 9 | CBMMIS | RO | 0 | GPTM CaptureB Match Masked Interrupt<br>This is the CaptureB match interrupt status after masking. |
| 8 | TBTOMIS | RO | 0 | GPTM TimerB Time-Out Masked Interrupt<br>This is the TimerB time-out interrupt status after masking. |
| 7:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | RTCMIS | RO | 0 | GPTM RTC Masked Interrupt<br>This is the RTC event interrupt status after masking. |
| 2 | CAEMIS | RO | 0 | GPTM CaptureA Event Masked Interrupt<br>This is the CaptureA event interrupt status after masking. |
| 1 | CAMMIS | RO | 0 | GPTM CaptureA Match Masked Interrupt<br>This is the CaptureA match interrupt status after masking. |
| 0 | TATOMIS | RO | 0 | GPTM TimerA Time-Out Masked Interrupt<br>This is the TimerA time-out interrupt status after masking. |

## Register 8: GPTM Interrupt Clear (GPTMICR), offset 0x024

This register is used to clear the status bits in the **GPTMRIS** and **GPTMMIS** registers. Writing a 1 to a bit clears the corresponding bit in the **GPTMRIS** and **GPTMMIS** registers.

GPTM Interrupt Clear (GPTMICR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x024
Type W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | CBECINT | W1C | 0 | GPTM CaptureB Event Interrupt Clear<br>The CBECINT values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 9 | CBMCINT | W1C | 0 | GPTM CaptureB Match Interrupt Clear<br>The CBMCINT values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 8 | TBTOCINT | W1C | 0 | GPTM TimerB Time-Out Interrupt Clear<br>The TBTOCINT values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 7:4 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

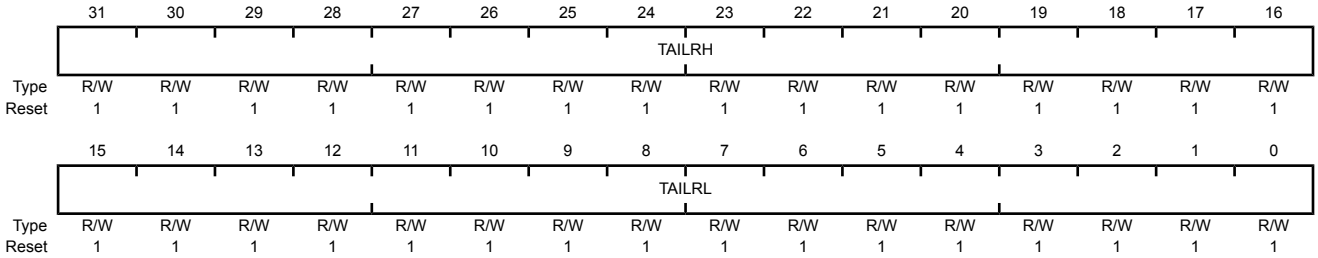
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | RTCCINT | W1C | 0 | GPTM RTC Interrupt Clear<br>The RTCCINT values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 2 | CAECINT | W1C | 0 | GPTM CaptureA Event Interrupt Clear<br>The CAECINT values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 1 | CAMCINT | W1C | 0 | GPTM CaptureA Match Interrupt Clear<br>The CAMCINT values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |
| 0 | TATOCINT | W1C | 0 | GPTM TimerA Time-Out Interrupt Clear<br>The TATOCINT values are defined as follows:<br><br>Value Description<br>0 The interrupt is unaffected.<br>1 The interrupt is cleared. |

## Register 9: GPTM TimerA Interval Load (GPTMTAILR), offset 0x028

This register is used to load the starting count value into the timer. When GPTM is configured to one of the 32-bit modes, **GPTMTAILR** appears as a 32-bit register (the upper 16-bits correspond to the contents of the **GPTM TimerB Interval Load (GPTMTBILR)** register). In 16-bit mode, the upper 16 bits of this register read as 0s and have no effect on the state of **GPTMTBILR**.

GPTM TimerA Interval Load (GPTMTAILR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x028
Type R/W, reset 0xFFFF.FFFF



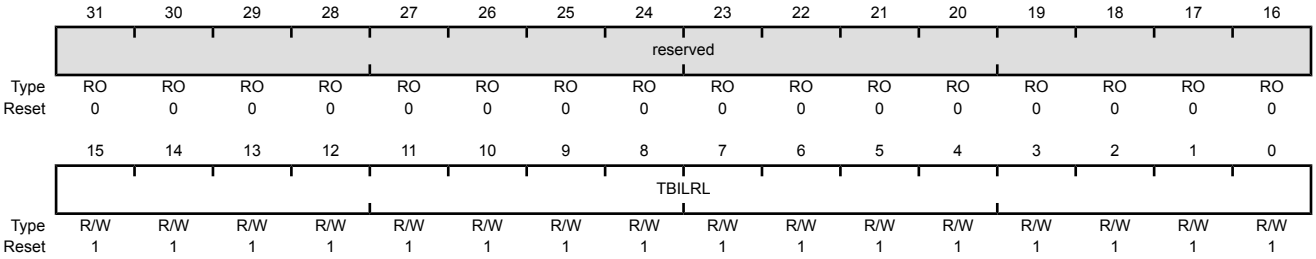

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | TAILRH | R/W | 0xFFFF | GPTM TimerA Interval Load Register High<br>When configured for 32-bit mode via the **GPTMCFG** register, the **GPTM TimerB Interval Load (GPTMTBILR)** register loads this value on a write. A read returns the current value of **GPTMTBILR**.<br>In 16-bit mode, this field reads as 0 and does not have an effect on the state of **GPTMTBILR**. |
| 15:0 | TAILRL | R/W | 0xFFFF | GPTM TimerA Interval Load Register Low<br>For both 16- and 32-bit modes, writing this field loads the counter for TimerA. A read returns the current value of **GPTMTAILR**. |

## Register 10: GPTM TimerB Interval Load (GPTMTBILR), offset 0x02C

This register is used to load the starting count value into TimerB. When the GPTM is configured to a 32-bit mode, **GPTMTBILR** returns the current value of TimerB and ignores writes.

GPTM TimerB Interval Load (GPTMTBILR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x02C
Type R/W, reset 0x0000.FFFF



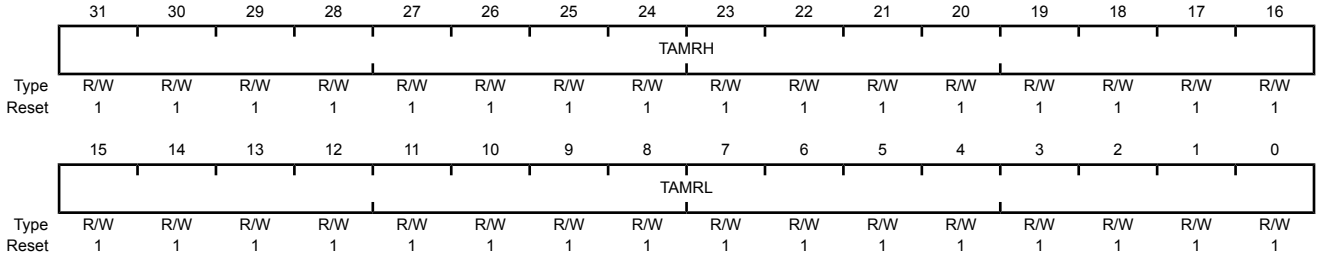

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | TBILRL | R/W | 0xFFFF | GPTM TimerB Interval Load Register<br>When the GPTM is not configured as a 32-bit timer, a write to this field updates **GPTMTBILR**. In 32-bit mode, writes are ignored, and reads return the current value of **GPTMTBILR**. |

## Register 11: GPTM TimerA Match (GPTMTAMATCHR), offset 0x030

This register is used in 32-bit Real-Time Clock mode and 16-bit PWM and Input Edge Count modes.

GPTM TimerA Match (GPTMTAMATCHR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x030
Type R/W, reset 0xFFFF.FFFF



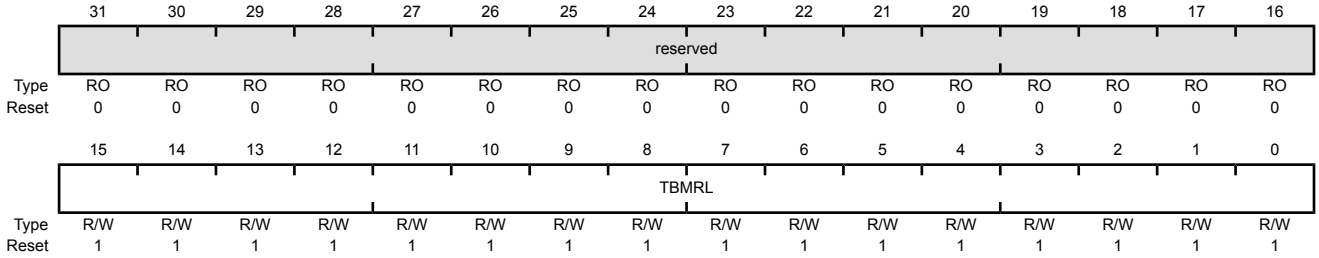

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | TAMRH | R/W | 0xFFFF | GPTM TimerA Match Register High<br>When configured for 32-bit Real-Time Clock (RTC) mode via the **GPTMCFG** register, this value is compared to the upper half of **GPTMTAR**, to determine match events.<br>In 16-bit mode, this field reads as 0 and does not have an effect on the state of **GPTMTBMATCHR**. |
| 15:0 | TAMRL | R/W | 0xFFFF | GPTM TimerA Match Register Low<br>When configured for 32-bit Real-Time Clock (RTC) mode via the **GPTMCFG** register, this value is compared to the lower half of **GPTMTAR**, to determine match events.<br>When configured for PWM mode, this value along with **GPTMTAILR**, determines the duty cycle of the output PWM signal.<br>When configured for Edge Count mode, this value along with **GPTMTAILR**, determines how many edge events are counted. The total number of edge events counted is equal to the value in **GPTMTAILR** minus this value. |

## Register 12: GPTM TimerB Match (GPTMTBMATCHR), offset 0x034

This register is used in 16-bit PWM and Input Edge Count modes.

GPTM TimerB Match (GPTMTBMATCHR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x034
Type R/W, reset 0x0000.FFFF



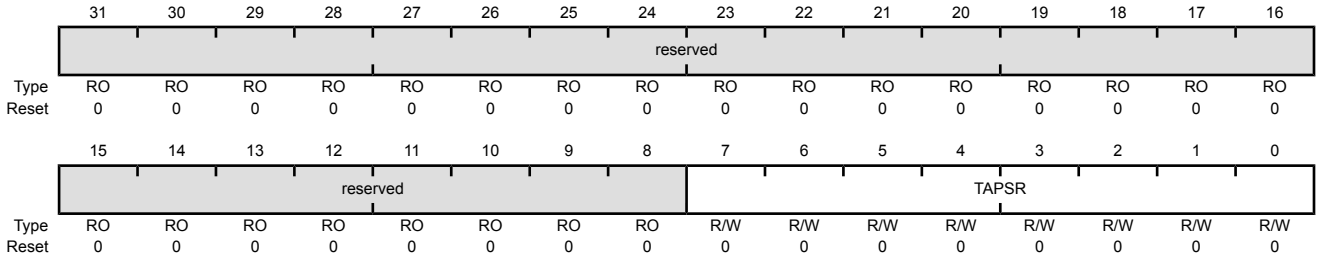

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | TBMRL | R/W | 0xFFFF | GPTM TimerB Match Register Low<br>When configured for PWM mode, this value along with **GPTMTBILR**, determines the duty cycle of the output PWM signal.<br>When configured for Edge Count mode, this value along with **GPTMTBILR**, determines how many edge events are counted. The total number of edge events counted is equal to the value in **GPTMTBILR** minus this value. |

## Register 13: GPTM TimerA Prescale (GPTMTAPR), offset 0x038

This register allows software to extend the range of the 16-bit timers when operating in one-shot or periodic mode.

GPTM TimerA Prescale (GPTMTAPR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x038
Type R/W, reset 0x0000.0000



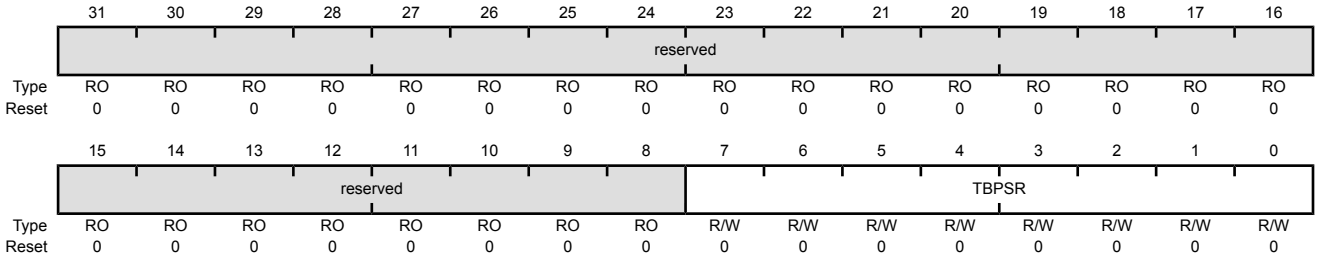

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | TAPSR | R/W | 0x00 | GPTM TimerA Prescale<br>The register loads this value on a write. A read returns the current value of the register.<br>Refer to Table 8-2 on page 263 for more details and an example. |

## Register 14: GPTM TimerB Prescale (GPTMTBPR), offset 0x03C

This register allows software to extend the range of the 16-bit timers when operating in one-shot or periodic mode.

GPTM TimerB Prescale (GPTMTBPR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x03C
Type R/W, reset 0x0000.0000



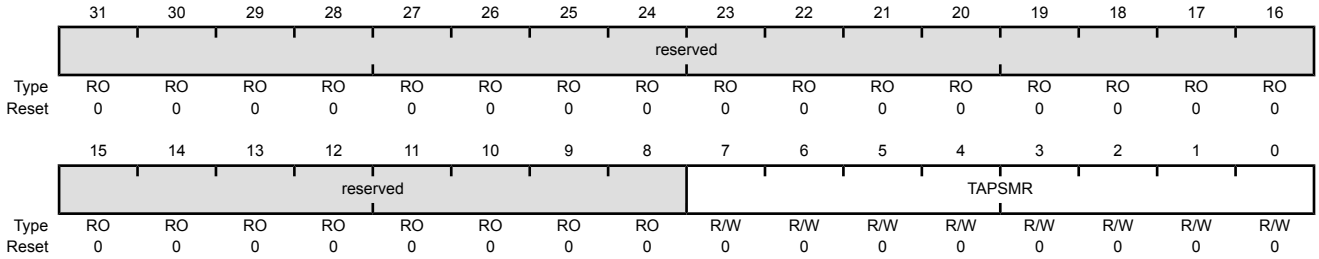

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | TBPSR | R/W | 0x00 | GPTM TimerB Prescale<br>The register loads this value on a write. A read returns the current value of this register.<br>Refer to Table 8-2 on page 263 for more details and an example. |

## Register 15: GPTM TimerA Prescale Match (GPTMTAPMR), offset 0x040

This register effectively extends the range of **GPTMTAMATCHR** to 24 bits when operating in 16-bit one-shot or periodic mode.

GPTM TimerA Prescale Match (GPTMTAPMR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x040
Type R/W, reset 0x0000.0000



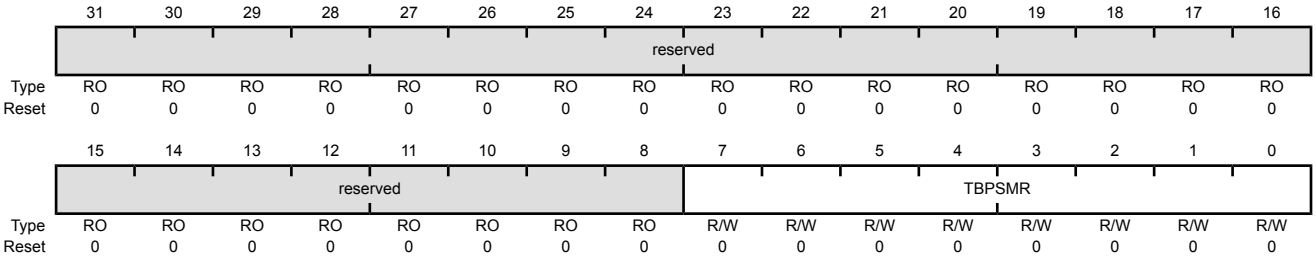

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | TAPSMR | R/W | 0x00 | GPTM TimerA Prescale Match<br>This value is used alongside **GPTMTAMATCHR** to detect timer match events while using a prescaler. |

## Register 16: GPTM TimerB Prescale Match (GPTMTBPMR), offset 0x044

This register effectively extends the range of **GPTMTBMATCHR** to 24 bits when operating in 16-bit one-shot or periodic mode.

GPTM TimerB Prescale Match (GPTMTBPMR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x044
Type R/W, reset 0x0000.0000



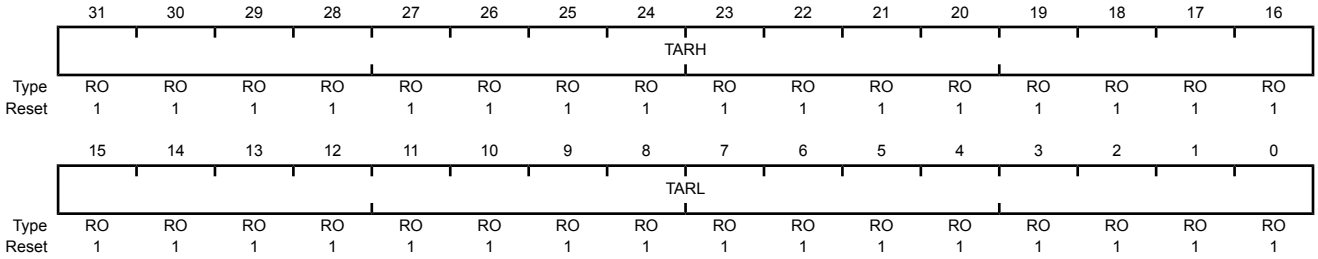

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | TBPSMR | R/W | 0x00 | GPTM TimerB Prescale Match<br>This value is used alongside **GPTMTBMATCHR** to detect timer match events while using a prescaler. |

## Register 17: GPTM TimerA (GPTMTAR), offset 0x048

This register shows the current value of the TimerA counter in all cases except for Input Edge Count mode. When in this mode, this register contains the number of edges that have occurred.

GPTM TimerA (GPTMTAR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x048
Type RO, reset 0xFFFF.FFFF



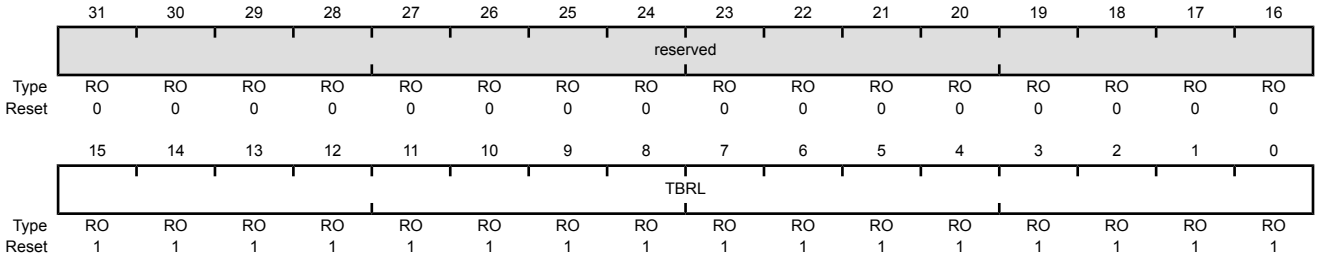

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | TARH | RO | 0xFFFF | GPTM TimerA Register High<br>If the **GPTMCFG** is in a 32-bit mode, TimerB value is read. If the **GPTMCFG** is in a 16-bit mode, this is read as zero. |
| 15:0 | TARL | RO | 0xFFFF | GPTM TimerA Register Low<br>A read returns the current value of the **GPTM TimerA Count Register**, except in Input Edge-Count mode, when it returns the number of edges that have occurred. |

## Register 18: GPTM TimerB (GPTMTBR), offset 0x04C

This register shows the current value of the TimerB counter in all cases except for Input Edge Count mode. When in this mode, this register contains the number of edges that have occurred.

GPTM TimerB (GPTMTBR)

Timer0 base: 0x4003.0000
Timer1 base: 0x4003.1000
Timer2 base: 0x4003.2000
Offset 0x04C
Type RO, reset 0x0000.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | TBRL | RO | 0xFFFF | GPTM TimerB<br>A read returns the current value of the **GPTM TimerB Count Register**, except in Input Edge-Count mode, when it returns the number of edges that have occurred. |

# 9 Watchdog Timer

A watchdog timer can generate nonmaskable interrupts (NMIs) or a reset when a time-out value is reached. The watchdog timer is used to regain control when a system has failed due to a software error or due to the failure of an external device to respond in the expected way.

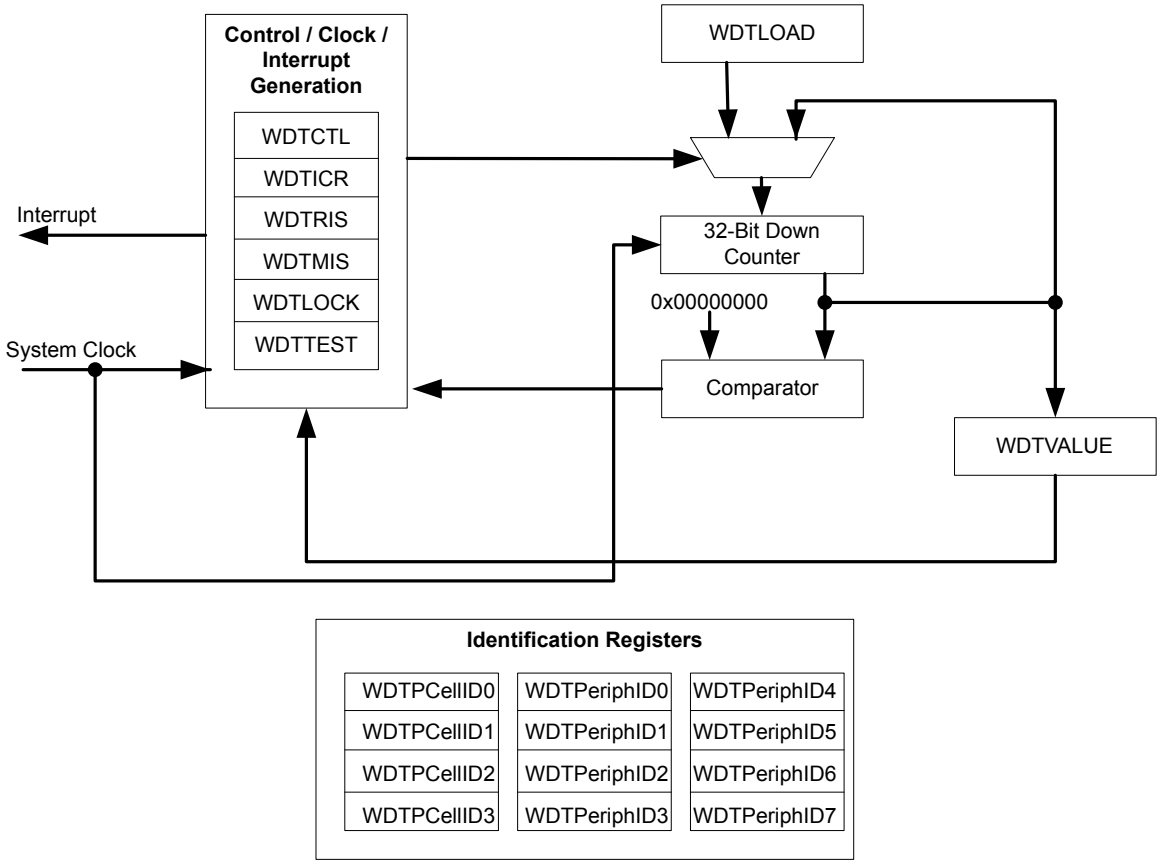
The Stellaris® Watchdog Timer module has the following features:

- 32-bit down counter with a programmable load register
- Separate watchdog clock with an enable
- Programmable interrupt generation logic with interrupt masking
- Lock register protection from runaway software
- Reset generation logic with an enable/disable
- User-enabled stalling when the controller asserts the CPU Halt flag during debug

The Watchdog Timer can be configured to generate an interrupt to the controller on its first time-out, and to generate a reset signal on its second time-out. Once the Watchdog Timer has been configured, the lock register can be written to prevent the timer configuration from being inadvertently altered.

## 9.1 Block Diagram

**Figure 9-1. WDT Module Block Diagram**




## 9.2 Functional Description

The Watchdog Timer module generates the first time-out signal when the 32-bit counter reaches the zero state after being enabled; enabling the counter also enables the watchdog timer interrupt. After the first time-out event, the 32-bit counter is re-loaded with the value of the **Watchdog Timer Load (WDTLOAD)** register, and the timer resumes counting down from that value. Once the Watchdog Timer has been configured, the **Watchdog Timer Lock (WDTLOCK)** register is written, which prevents the timer configuration from being inadvertently altered by software.

If the timer counts down to its zero state again before the first time-out interrupt is cleared, and the reset signal has been enabled (via the `WatchdogResetEnable` function), the Watchdog timer asserts its reset signal to the system. If the interrupt is cleared before the 32-bit counter reaches its second time-out, the 32-bit counter is loaded with the value in the **WDTLOAD** register, and counting resumes from that value.

If **WDTLOAD** is written with a new value while the Watchdog Timer counter is counting, then the counter is loaded with the new value and continues counting.

Writing to **WDTLOAD** does not clear an active interrupt. An interrupt must be specifically cleared by writing to the **Watchdog Interrupt Clear (WDTICR)** register.

The Watchdog module interrupt and reset generation can be enabled or disabled as required. When the interrupt is re-enabled, the 32-bit counter is preloaded with the load register value and not its last state.

## 9.3 Initialization and Configuration

To use the WDT, its peripheral clock must be enabled by setting the WDT bit in the **RCGC0** register. The Watchdog Timer is configured using the following sequence:

1. Load the **WDTLOAD** register with the desired timer load value.
2. If the Watchdog is configured to trigger system resets, set the RESEN bit in the **WDTCTL** register.
3. Set the INTEN bit in the **WDTCTL** register to enable the Watchdog and lock the control register.

If software requires that all of the watchdog registers are locked, the Watchdog Timer module can be fully locked by writing any value to the **WDTLOCK** register. To unlock the Watchdog Timer, write a value of 0x1ACC.E551.

## 9.4 Register Map

Table 9-1 on page 297 lists the Watchdog registers. The offset listed is a hexadecimal increment to the register's address, relative to the Watchdog Timer base address of 0x4000.0000.

**Table 9-1. Watchdog Timer Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | WDTLOAD | R/W | 0xFFFF.FFFF | Watchdog Load | 299 |
| 0x004 | WDTVALUE | RO | 0xFFFF.FFFF | Watchdog Value | 300 |
| 0x008 | WDTCTL | R/W | 0x0000.0000 | Watchdog Control | 301 |
| 0x00C | WDTICR | WO | - | Watchdog Interrupt Clear | 302 |
| 0x010 | WDTRIS | RO | 0x0000.0000 | Watchdog Raw Interrupt Status | 303 |
| 0x014 | WDTMIS | RO | 0x0000.0000 | Watchdog Masked Interrupt Status | 304 |
| 0x418 | WDTTEST | R/W | 0x0000.0000 | Watchdog Test | 305 |
| 0xC00 | WDTLOCK | R/W | 0x0000.0000 | Watchdog Lock | 306 |
| 0xFD0 | WDTPeriphID4 | RO | 0x0000.0000 | Watchdog Peripheral Identification 4 | 307 |
| 0xFD4 | WDTPeriphID5 | RO | 0x0000.0000 | Watchdog Peripheral Identification 5 | 308 |
| 0xFD8 | WDTPeriphID6 | RO | 0x0000.0000 | Watchdog Peripheral Identification 6 | 309 |
| 0xFDC | WDTPeriphID7 | RO | 0x0000.0000 | Watchdog Peripheral Identification 7 | 310 |
| 0xFE0 | WDTPeriphID0 | RO | 0x0000.0005 | Watchdog Peripheral Identification 0 | 311 |
| 0xFE4 | WDTPeriphID1 | RO | 0x0000.0018 | Watchdog Peripheral Identification 1 | 312 |
| 0xFE8 | WDTPeriphID2 | RO | 0x0000.0018 | Watchdog Peripheral Identification 2 | 313 |

**Table 9-1. Watchdog Timer Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0xFEC | WDTPeriphID3 | RO | 0x0000.0001 | Watchdog Peripheral Identification 3 | 314 |
| 0xFF0 | WDTPCellID0 | RO | 0x0000.000D | Watchdog PrimeCell Identification 0 | 315 |
| 0xFF4 | WDTPCellID1 | RO | 0x0000.00F0 | Watchdog PrimeCell Identification 1 | 316 |
| 0xFF8 | WDTPCellID2 | RO | 0x0000.0005 | Watchdog PrimeCell Identification 2 | 317 |
| 0xFFC | WDTPCellID3 | RO | 0x0000.00B1 | Watchdog PrimeCell Identification 3 | 318 |

## 9.5 Register Descriptions

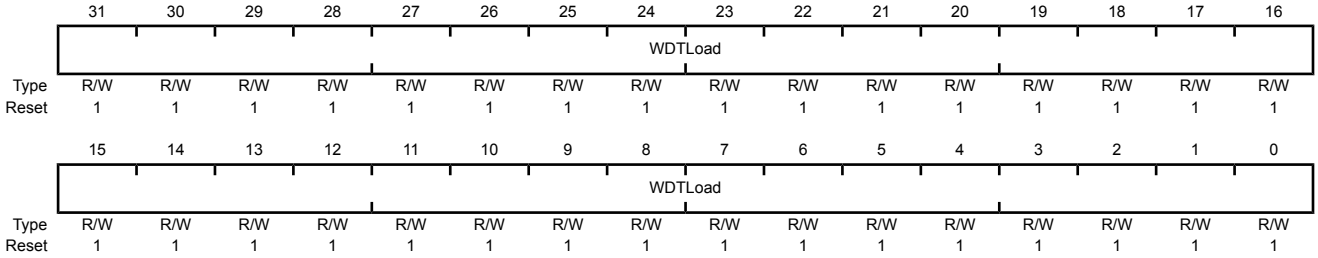
The remainder of this section lists and describes the WDT registers, in numerical order by address offset.

## Register 1: Watchdog Load (WDTLOAD), offset 0x000

This register is the 32-bit interval value used by the 32-bit counter. When this register is written, the value is immediately loaded and the counter restarts counting down from the new value. If the **WDTLOAD** register is loaded with 0x0000.0000, an interrupt is immediately generated.

Watchdog Load (WDTLOAD)

Base 0x4000.0000
Offset 0x000
Type R/W, reset 0xFFFF.FFFF



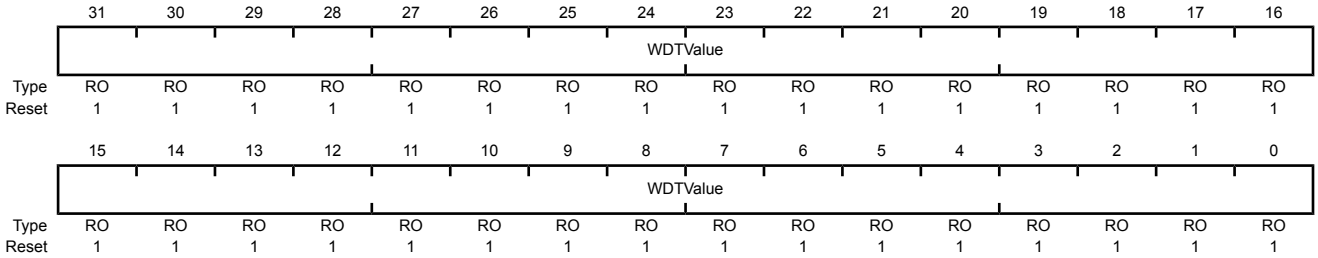

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTLoad | R/W | 0xFFFF.FFFF | Watchdog Load Value |

## Register 2: Watchdog Value (WDTVALUE), offset 0x004

This register contains the current count value of the timer.

Watchdog Value (WDTVALUE)

Base 0x4000.0000
Offset 0x004
Type RO, reset 0xFFFF.FFFF




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTValue | RO | 0xFFFF.FFFF | Watchdog Value<br>Current value of the 32-bit down counter. |

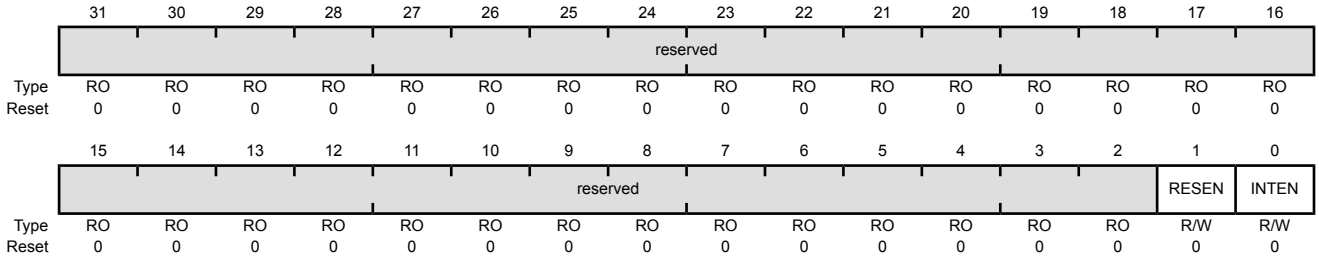
## Register 3: Watchdog Control (WDTCTL), offset 0x008

This register is the watchdog control register. The watchdog timer can be configured to generate a reset signal (on second time-out) or an interrupt on time-out.

When the watchdog interrupt has been enabled, all subsequent writes to the control register are ignored. The only mechanism that can re-enable writes is a hardware reset.

Watchdog Control (WDTCTL)

Base 0x4000.0000
Offset 0x008
Type R/W, reset 0x0000.0000



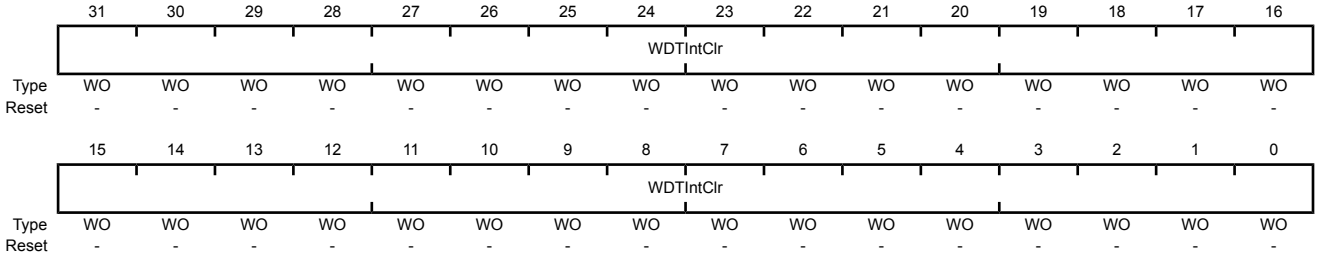

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | RESEN | R/W | 0 | Watchdog Reset Enable<br>The RESEN values are defined as follows:<br><br>Value Description<br>0 Disabled.<br>1 Enable the Watchdog module reset output. |
| 0 | INTEN | R/W | 0 | Watchdog Interrupt Enable<br>The INTEN values are defined as follows:<br><br>Value Description<br>0 Interrupt event disabled (once this bit is set, it can only be cleared by a hardware reset).<br>1 Interrupt event enabled. Once enabled, all writes are ignored. |

## Register 4: Watchdog Interrupt Clear (WDTICR), offset 0x00C

This register is the interrupt clear register. A write of any value to this register clears the Watchdog interrupt and reloads the 32-bit counter from the **WDTLOAD** register. Value for a read or reset is indeterminate.

Watchdog Interrupt Clear (WDTICR)

Base 0x4000.0000
Offset 0x00C
Type WO, reset -



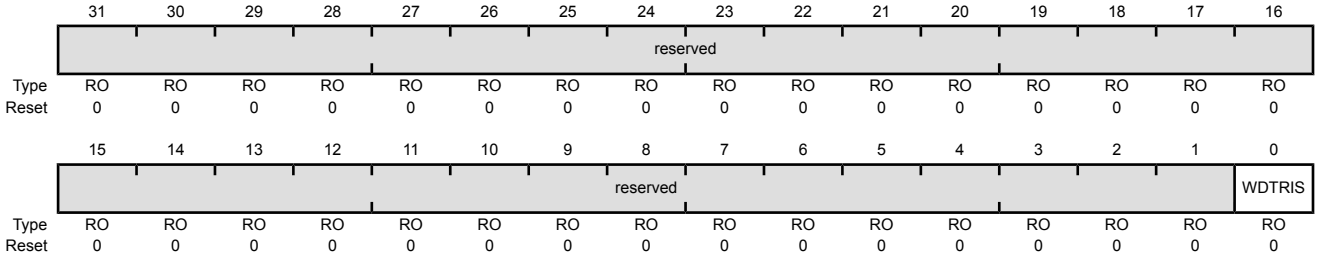

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTIntClr | WO | - | Watchdog Interrupt Clear |

## Register 5: Watchdog Raw Interrupt Status (WDTRIS), offset 0x010

This register is the raw interrupt status register. Watchdog interrupt events can be monitored via this register if the controller interrupt is masked.

Watchdog Raw Interrupt Status (WDTRIS)

Base 0x4000.0000
Offset 0x010
Type RO, reset 0x0000.0000



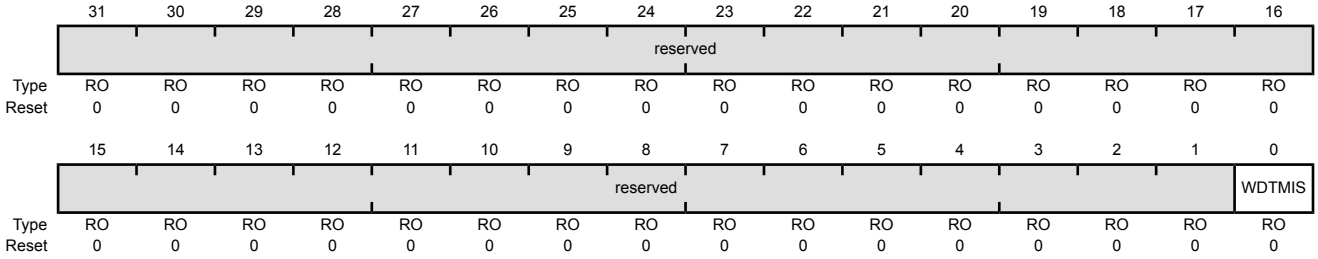

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | WDTRIS | RO | 0 | Watchdog Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of **WDTINTR**. |

## Register 6: Watchdog Masked Interrupt Status (WDTMIS), offset 0x014

This register is the masked interrupt status register. The value of this register is the logical AND of the raw interrupt bit and the Watchdog interrupt enable bit.

Watchdog Masked Interrupt Status (WDTMIS)

Base 0x4000.0000
Offset 0x014
Type RO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | WDTMIS | RO | 0 | Watchdog Masked Interrupt Status<br>Gives the masked interrupt state (after masking) of the **WDTINTR** interrupt. |

## Register 7: Watchdog Test (WDTTEST), offset 0x418

This register provides user-enabled stalling when the microcontroller asserts the CPU halt flag during debug.

Watchdog Test (WDTTEST)

Base 0x4000.0000
Offset 0x418
Type R/W, reset 0x0000.0000

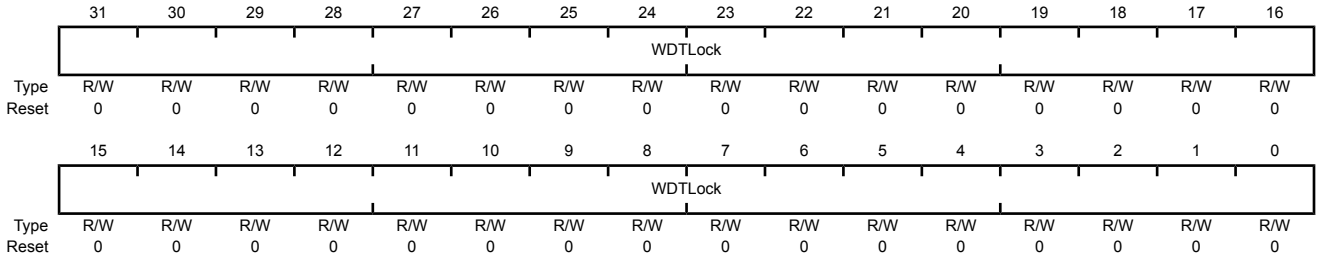

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:9 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 8 | STALL | R/W | 0 | Watchdog Stall Enable<br>When set to 1, if the Stellaris microcontroller is stopped with a debugger, the watchdog timer stops counting. Once the microcontroller is restarted, the watchdog timer resumes counting. |
| 7:0 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 8: Watchdog Lock (WDTLOCK), offset 0xC00

Writing 0x1ACC.E551 to the **WDTLOCK** register enables write access to all other registers. Writing any other value to the **WDTLOCK** register re-enables the locked state for register writes to all the other registers. Reading the **WDTLOCK** register returns the lock status rather than the 32-bit value written. Therefore, when write accesses are disabled, reading the **WDTLOCK** register returns 0x0000.0001 (when locked; otherwise, the returned value is 0x0000.0000 (unlocked)).

Watchdog Lock (WDTLOCK)

Base 0x4000.0000
Offset 0xC00
Type R/W, reset 0x0000.0000



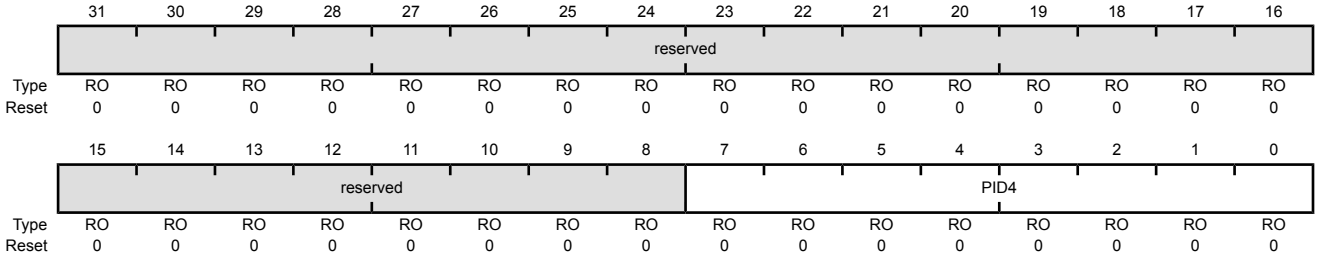

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:0 | WDTLock | R/W | 0x0000 | Watchdog Lock<br>A write of the value 0x1ACC.E551 unlocks the watchdog registers for write access. A write of any other value reapplies the lock, preventing any register updates.<br>A read of this register returns the following values:<br>Value Description<br>0x0000.0001 Locked<br>0x0000.0000 Unlocked |

## Register 9: Watchdog Peripheral Identification 4 (WDTPeriphID4), offset 0xFD0

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 4 (WDTPeriphID4)

Base 0x4000.0000
Offset 0xFD0
Type RO, reset 0x0000.0000



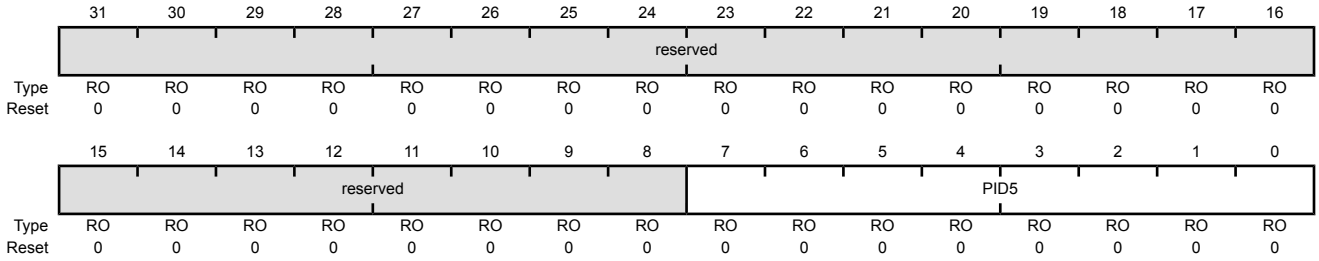

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID4 | RO | 0x00 | WDT Peripheral ID Register[7:0] |

## Register 10: Watchdog Peripheral Identification 5 (WDTPeriphID5), offset 0xFD4

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 5 (WDTPeriphID5)

Base 0x4000.0000
Offset 0xFD4
Type RO, reset 0x0000.0000



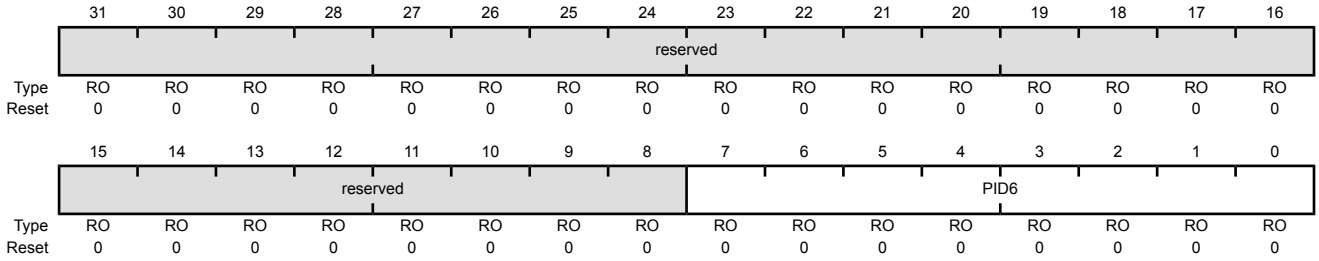

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID5 | RO | 0x00 | WDT Peripheral ID Register[15:8] |

## Register 11: Watchdog Peripheral Identification 6 (WDTPeriphID6), offset 0xFD8

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 6 (WDTPeriphID6)

Base 0x4000.0000
Offset 0xFD8
Type RO, reset 0x0000.0000



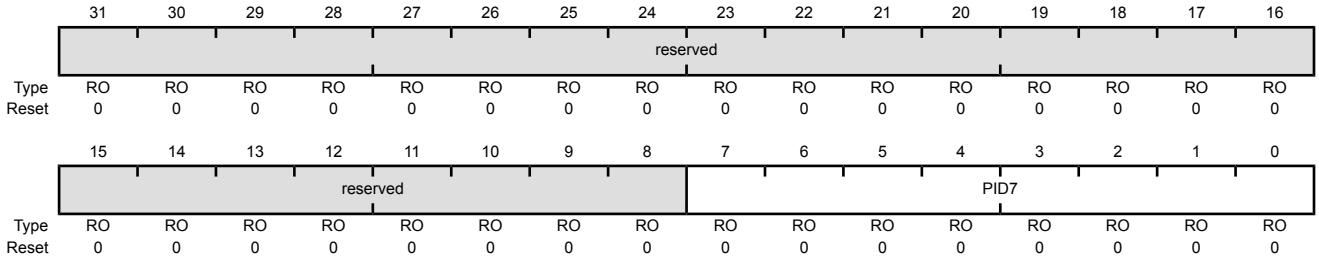

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID6 | RO | 0x00 | WDT Peripheral ID Register[23:16] |

## Register 12: Watchdog Peripheral Identification 7 (WDTPeriphID7), offset 0xFDC

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 7 (WDTPeriphID7)

Base 0x4000.0000
Offset 0xFDC
Type RO, reset 0x0000.0000



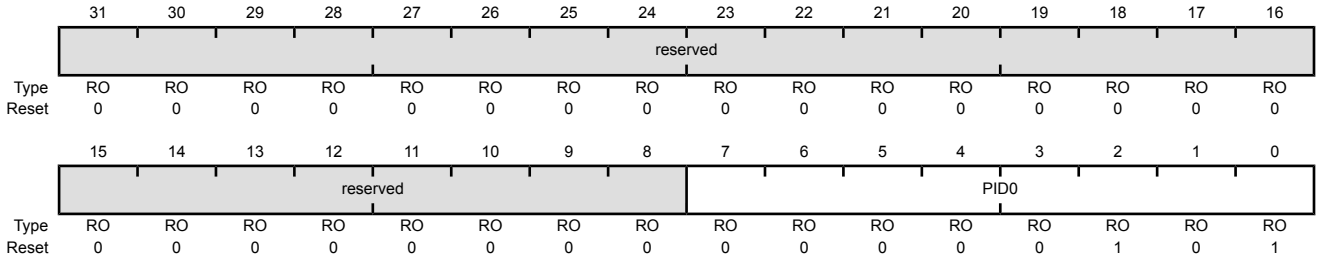

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID7 | RO | 0x00 | WDT Peripheral ID Register[31:24] |

## Register 13: Watchdog Peripheral Identification 0 (WDTPeriphID0), offset 0xFE0

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 0 (WDTPeriphID0)

Base 0x4000.0000
Offset 0xFE0
Type RO, reset 0x0000.0005



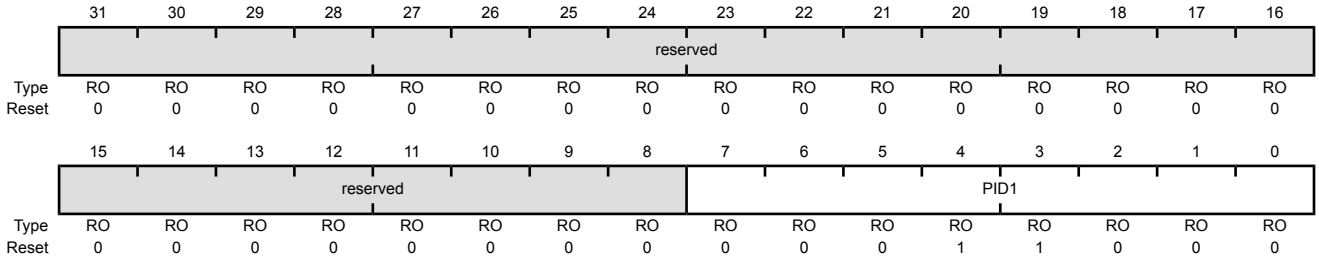

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID0 | RO | 0x05 | Watchdog Peripheral ID Register[7:0] |

## Register 14: Watchdog Peripheral Identification 1 (WDTPeriphID1), offset 0xFE4

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 1 (WDTPeriphID1)

Base 0x4000.0000
Offset 0xFE4
Type RO, reset 0x0000.0018



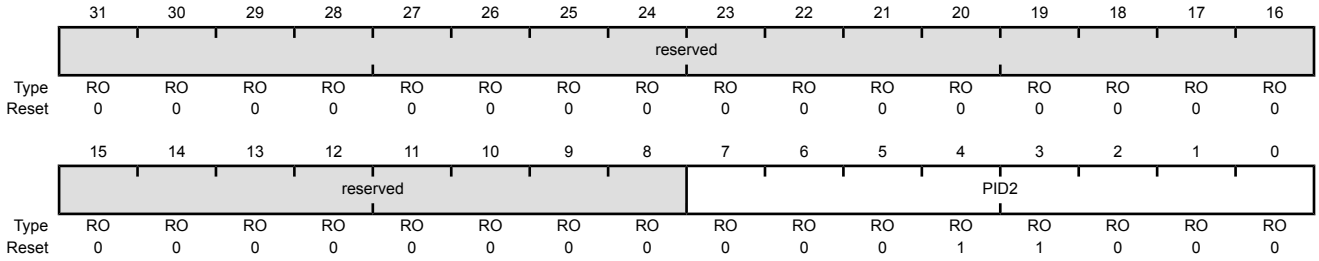

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID1 | RO | 0x18 | Watchdog Peripheral ID Register[15:8] |

## Register 15: Watchdog Peripheral Identification 2 (WDTPeriphID2), offset 0xFE8

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 2 (WDTPeriphID2)

Base 0x4000.0000
Offset 0xFE8
Type RO, reset 0x0000.0018



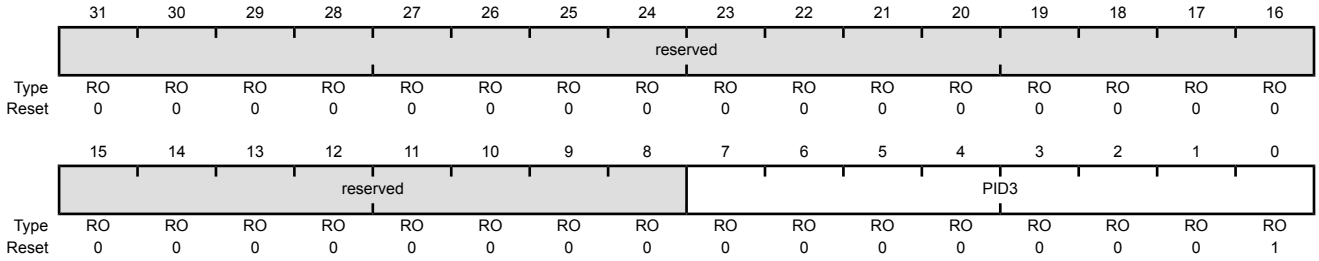

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID2 | RO | 0x18 | Watchdog Peripheral ID Register[23:16] |

## Register 16: Watchdog Peripheral Identification 3 (WDTPeriphID3), offset 0xFEC

The **WDTPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog Peripheral Identification 3 (WDTPeriphID3)

Base 0x4000.0000
Offset 0xFEC
Type RO, reset 0x0000.0001



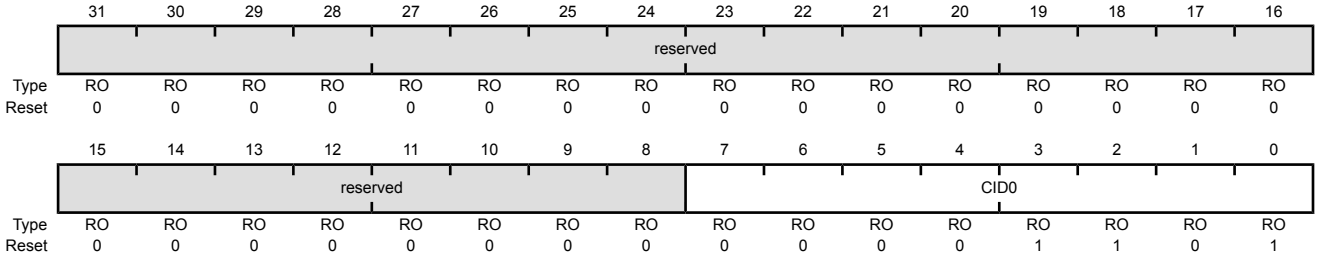

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID3 | RO | 0x01 | Watchdog Peripheral ID Register[31:24] |

### Register 17: Watchdog PrimeCell Identification 0 (WDTPCellID0), offset 0xFF0

The **WDTPCellIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog PrimeCell Identification 0 (WDTPCellID0)

Base 0x4000.0000
Offset 0xFF0
Type RO, reset 0x0000.000D



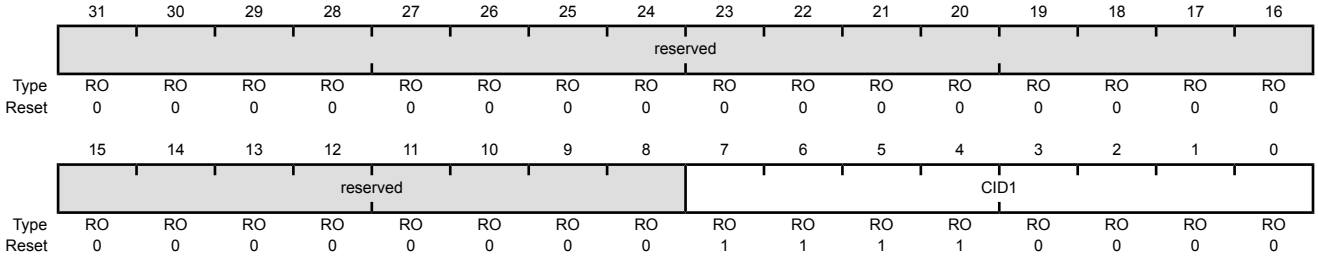

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID0 | RO | 0x0D | Watchdog PrimeCell ID Register[7:0] |

## Register 18: Watchdog PrimeCell Identification 1 (WDTPCellID1), offset 0xFF4

The **WDTPCellIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog PrimeCell Identification 1 (WDTPCellID1)

Base 0x4000.0000
Offset 0xFF4
Type RO, reset 0x0000.00F0



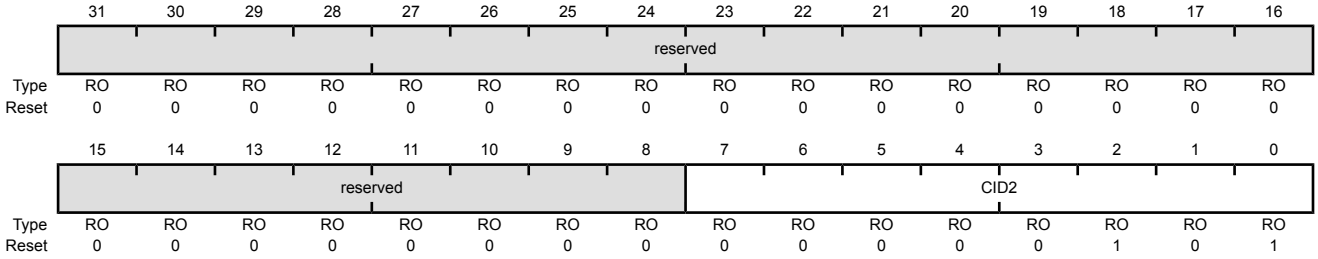

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID1 | RO | 0xF0 | Watchdog PrimeCell ID Register[15:8] |

### Register 19: Watchdog PrimeCell Identification 2 (WDTPCellID2), offset 0xFF8

The **WDTPCellIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog PrimeCell Identification 2 (WDTPCellID2)

Base 0x4000.0000
Offset 0xFF8
Type RO, reset 0x0000.0005



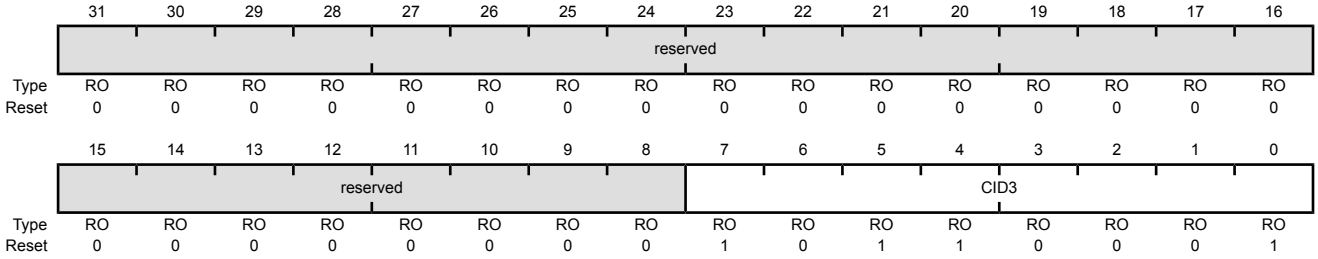

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID2 | RO | 0x05 | Watchdog PrimeCell ID Register[23:16] |

## Register 20: Watchdog PrimeCell Identification 3 (WDTPCellID3 ), offset 0xFFC

The **WDTPCellIDn** registers are hard-coded and the fields within the register determine the reset value.

Watchdog PrimeCell Identification 3 (WDTPCellID3)

Base 0x4000.0000
Offset 0xFFC
Type RO, reset 0x0000.00B1




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID3 | RO | 0xB1 | Watchdog PrimeCell ID Register[31:24] |

# 10 Analog-to-Digital Converter (ADC)

An analog-to-digital converter (ADC) is a peripheral that converts a continuous analog voltage to a discrete digital number.

The Stellaris® ADC module features 10-bit conversion resolution and supports eight input channels, plus an internal temperature sensor. The ADC module contains four programmable sequencer which allows for the sampling of multiple analog input sources without controller intervention. Each sample sequence provides flexible programming with fully configurable input source, trigger events, interrupt generation, and sequence priority.

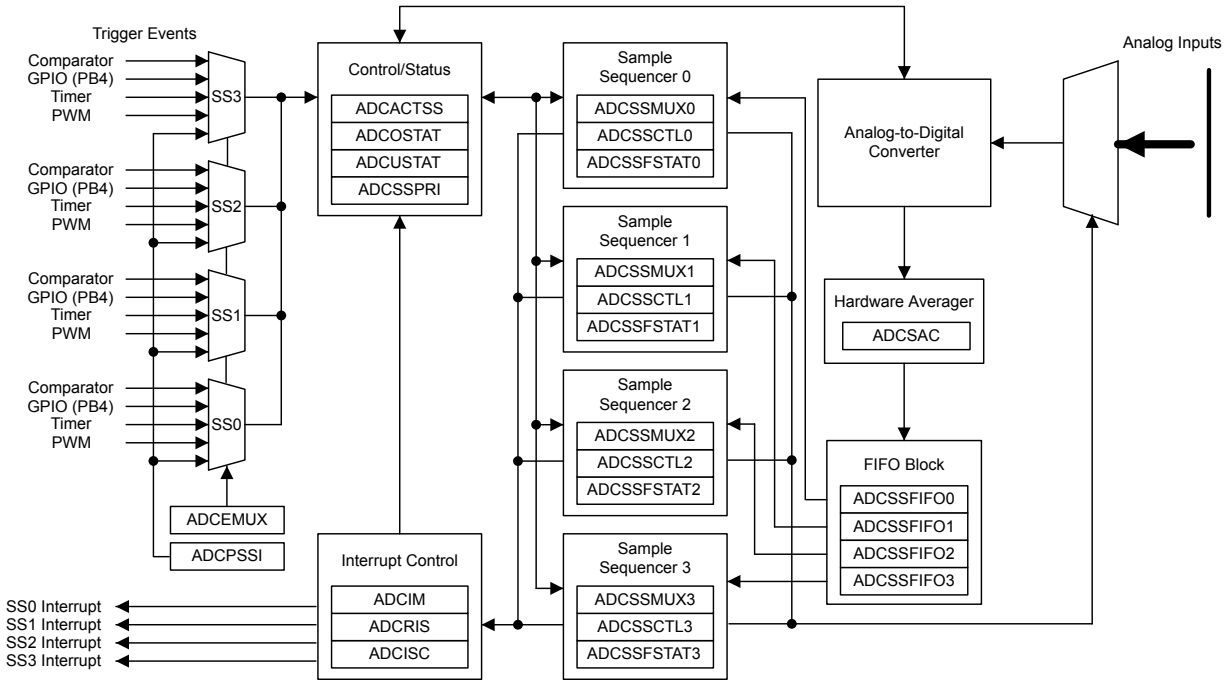
The Stellaris ADC module provides the following features:

- Eight analog input channels
- Single-ended and differential-input configurations
- On-chip internal temperature sensor
- Sample rate of one million samples/second
- Flexible, configurable analog-to-digital conversion
- Four programmable sample conversion sequences from one to eight entries long, with corresponding conversion result FIFOs
- Flexible trigger control
  - Controller (software)
  - Timers
  - GPIO
- Hardware averaging of up to 64 samples for improved accuracy
- Converter uses an internal 3-V reference

## 10.1 Block Diagram

Figure 10-1 on page 320 provides details on the internal configuration of the ADC controls and data registers.

**Figure 10-1. ADC Module Block Diagram**




## 10.2 Functional Description

The Stellaris ADC collects sample data by using a programmable sequence-based approach instead of the traditional single or double-sampling approaches found on many ADC modules. Each *sample sequence* is a fully programmed series of consecutive (back-to-back) samples, allowing the ADC to collect data from multiple input sources without having to be re-configured or serviced by the controller. The programming of each sample in the sample sequence includes parameters such as the input source and mode (differential versus single-ended input), interrupt generation on sample completion, and the indicator for the last sample in the sequence.

### 10.2.1 Sample Sequencers

The sampling control and data capture is handled by the sample sequencers. All of the sequencers are identical in implementation except for the number of samples that can be captured and the depth of the FIFO. Table 10-1 on page 320 shows the maximum number of samples that each sequencer can capture and its corresponding FIFO depth. In this implementation, each FIFO entry is a 32-bit word, with the lower 10 bits containing the conversion result.

**Table 10-1. Samples and FIFO Depth of Sequencers**

| Sequencer | Number of Samples | Depth of FIFO |
|---|---|---|
| SS3 | 1 | 1 |
| SS2 | 4 | 4 |
| SS1 | 4 | 4 |
| SS0 | 8 | 8 |

For a given sample sequence, each sample is defined by two 4-bit nibbles in the **ADC Sample Sequence Input Multiplexer Select (ADCSSMUXn)** and **ADC Sample Sequence Control (ADCSSCTLn)** registers, where "n" corresponds to the sequence number. The **ADCSSMUXn**

nibbles select the input pin, while the **ADCSSCTLn** nibbles contain the sample control bits corresponding to parameters such as temperature sensor selection, interrupt enable, end of sequence, and differential input mode. Sample sequencers are enabled by setting the respective ASENn bit in the **ADC Active Sample Sequencer (ADCACTSS)** register, and should be configured before being enabled.

When configuring a sample sequence, multiple uses of the same input pin within the same sequence is allowed. In the **ADCSSCTLn** register, the IEn bits can be set for any combination of samples, allowing interrupts to be generated after every sample in the sequence if necessary. Also, the END bit can be set at any point within a sample sequence. For example, if Sequencer 0 is used, the END bit can be set in the nibble associated with the fifth sample, allowing Sequencer 0 to complete execution of the sample sequence after the fifth sample.

After a sample sequence completes execution, the result data can be retrieved from the **ADC Sample Sequence Result FIFO (ADCSSFIFOn)** registers. The FIFOs are simple circular buffers that read a single address to "pop" result data. For software debug purposes, the positions of the FIFO head and tail pointers are visible in the **ADC Sample Sequence FIFO Status (ADCSSFSTATn)** registers along with FULL and EMPTY status flags. Overflow and underflow conditions are monitored using the **ADCOSTAT** and **ADCUSTAT** registers.

### 10.2.2 Module Control

Outside of the sample sequencers, the remainder of the control logic is responsible for tasks such as:

- Interrupt generation
- Sequence prioritization
- Trigger configuration

Most of the ADC control logic runs at the ADC clock rate of 14-18 MHz. The internal ADC divider is configured automatically by hardware when the system XTAL is selected. The automatic clock divider configuration targets 16.667 MHz operation for all Stellaris devices.

#### 10.2.2.1 Interrupts

The register configurations of the sample sequencers dictate which events generate raw interrupts, but do not have control over whether the interrupt is actually sent to the interrupt controller. The ADC module's interrupt signals are controlled by the state of the MASK bits in the **ADC Interrupt Mask (ADCIM)** register. Interrupt status can be viewed at two locations: the **ADC Raw Interrupt Status (ADCRIS)** register, which shows the raw status of the various interrupt signals, and the **ADC Interrupt Status and Clear (ADCISC)** register, which shows active interrupts that are enabled by the **ADCIM** register. Sequencer interrupts are cleared by writing a 1 to the corresponding IN bit in **ADCISC**.

#### 10.2.2.2 Prioritization

When sampling events (triggers) happen concurrently, they are prioritized for processing by the values in the **ADC Sample Sequencer Priority (ADCSSPRI)** register. Valid priority values are in the range of 0-3, with 0 being the highest priority and 3 being the lowest. Multiple active sample sequencer units with the same priority do not provide consistent results, so software must ensure that all active sample sequencer units have a unique priority value.

#### 10.2.2.3 Sampling Events

Sample triggering for each sample sequencer is defined in the **ADC Event Multiplexer Select (ADCEMUX)** register. The external peripheral triggering sources vary by Stellaris family member, but all devices share the "Controller" and "Always" triggers. Software can initiate sampling by setting the SSx bits in the **ADC Processor Sample Sequence Initiate (ADCPSSI)** register.

Care must be taken when using the "Always" trigger. If a sequence's priority is too high, it is possible to starve other lower priority sequences.

### 10.2.3 Hardware Sample Averaging Circuit

Higher precision results can be generated using the hardware averaging circuit, however, the improved results are at the cost of throughput. Up to 64 samples can be accumulated and averaged to form a single data entry in the sequencer FIFO. Throughput is decreased proportionally to the number of samples in the averaging calculation. For example, if the averaging circuit is configured to average 16 samples, the throughput is decreased by a factor of 16.

By default the averaging circuit is off and all data from the converter passes through to the sequencer FIFO. The averaging hardware is controlled by the **ADC Sample Averaging Control (ADCSAC)** register (see page 340). There is a single averaging circuit and all input channels receive the same amount of averaging whether they are single-ended or differential.

### 10.2.4 Analog-to-Digital Converter

The converter itself generates a 10-bit output value for selected analog input. Special analog pads are used to minimize the distortion on the input. An internal 3 V reference is used by the converter resulting in sample values ranging from 0x000 at 0 V input to 0x3FF at 3 V input when in single-ended input mode.

### 10.2.5 Differential Sampling

In addition to traditional single-ended sampling, the ADC module supports differential sampling of two analog input channels. To enable differential sampling, software must set the Dn bit in the **ADCSSCTL0n** register in a step's configuration nibble.

When a sequence step is configured for differential sampling, its corresponding value in the **ADCSSMUXn** register must be set to one of the four differential pairs, numbered 0-3. Differential pair 0 samples analog inputs 0 and 1; differential pair 1 samples analog inputs 2 and 3; and so on (see Table 10-2 on page 322). The ADC does not support other differential pairings such as analog input 0 with analog input 3. The number of differential pairs supported is dependent on the number of analog inputs (see Table 10-2 on page 322).

**Table 10-2. Differential Sampling Pairs**

| Differential Pair | Analog Inputs |
|---|---|
| 0 | 0 and 1 |
| 1 | 2 and 3 |
| 2 | 4 and 5 |
| 3 | 6 and 7 |

The voltage sampled in differential mode is the difference between the odd and even channels:

$\Delta V$ (differential voltage) = $V_{IN\_EVEN}$ (even channels) – $V_{IN\_ODD}$ (odd channels), therefore:

- If $\Delta V = 0$, then the conversion result = 0x1FF

- If ΔV > 0, then the conversion result > 0x1FF (range is 0x1FF–0x3FF)
- If ΔV < 0, then the conversion result < 0x1FF (range is 0–0x1FF)

The differential pairs assign polarities to the analog inputs: the even-numbered input is always positive, and the odd-numbered input is always negative. In order for a valid conversion result to appear, the negative input must be in the range of ± 1.5 V of the positive input. If an analog input is greater than 3 V or less than 0 V (the valid range for analog inputs), the input voltage is clipped, meaning it appears as either 3 V or 0 V, respectively, to the ADC.

Figure 10-2 on page 323 shows an example of the negative input centered at 1.5 V. In this configuration, the differential range spans from -1.5 V to 1.5 V. Figure 10-3 on page 324 shows an example where the negative input is centered at -0.75 V, meaning inputs on the positive input saturate past a differential voltage of -0.75 V since the input voltage is less than 0 V. Figure 10-4 on page 324 shows an example of the negative input centered at 2.25 V, where inputs on the positive channel saturate past a differential voltage of 0.75 V since the input voltage would be greater than 3 V.

**Figure 10-2. Differential Sampling Range, $V_{IN\_ODD}$ = 1.5 V**

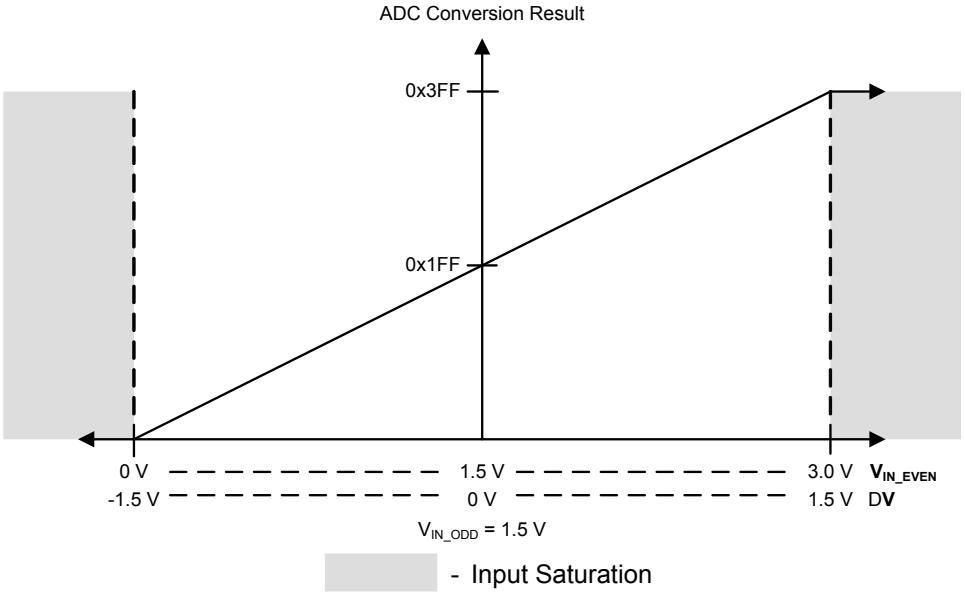


**Figure 10-3. Differential Sampling Range, $V_{IN\_ODD}$ = 0.75 V**

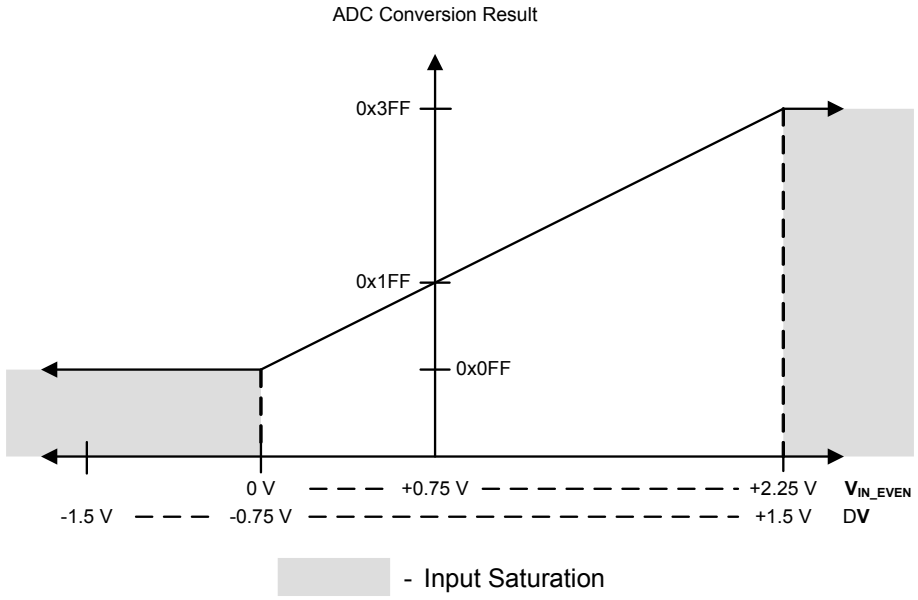
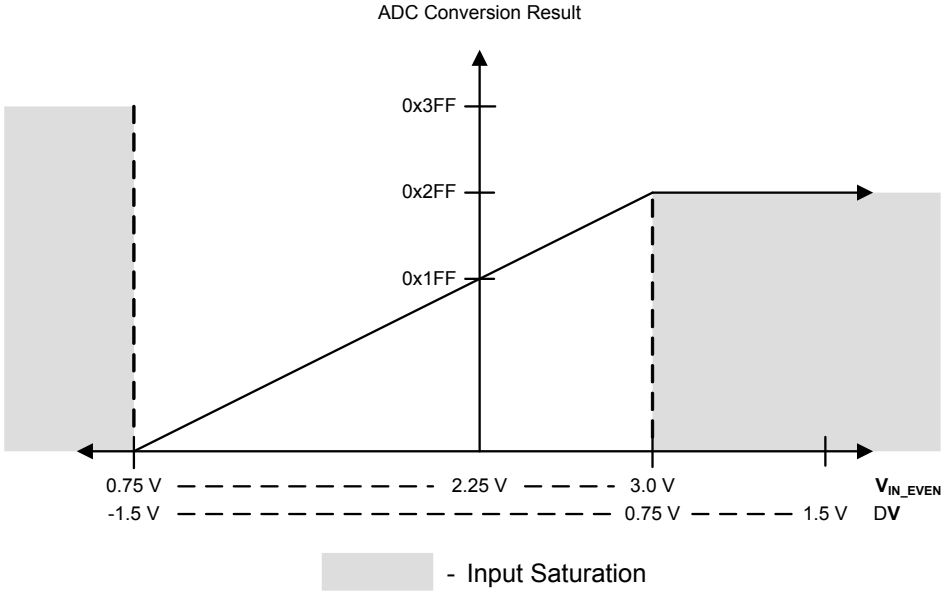



**Figure 10-4. Differential Sampling Range, $V_{IN\_ODD}$ = 2.25 V**




## 10.2.6 Test Modes

There is a user-available test mode that allows for loopback operation within the digital portion of the ADC module. This can be useful for debugging software without having to provide actual analog stimulus. This mode is available through the **ADC Test Mode Loopback (ADCTMLB)** register (see page 353).

### 10.2.7 Internal Temperature Sensor

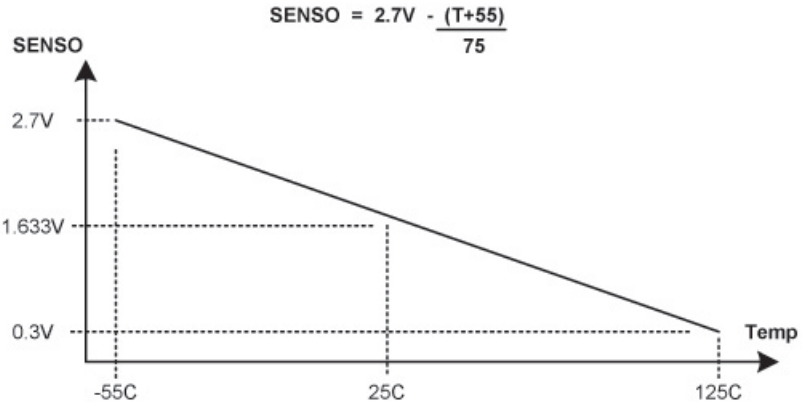
The temperature sensor does not have a separate enable, since it also contains the bandgap reference and must always be enabled. The reference is supplied to other analog modules; not just the ADC.

The internal temperature sensor provides an analog temperature reading as well as a reference voltage. The voltage at the output terminal SENSO is given by the following equation:

*SENSO* = 2.7 - ((T + 55) / 75)

This relation is shown in Figure 10-5 on page 325.

**Figure 10-5. Internal Temperature Sensor Characteristic**




## 10.3 Initialization and Configuration

In order for the ADC module to be used, the PLL must be enabled and using a supported crystal frequency (see the **RCC** register). Using unsupported frequencies can cause faulty operation in the ADC module.

### 10.3.1 Module Initialization

Initialization of the ADC module is a simple process with very few steps. The main steps include enabling the clock to the ADC and reconfiguring the sample sequencer priorities (if needed).

The initialization sequence for the ADC is as follows:

1. Enable the ADC clock by writing a value of 0x0001.0000 to the **RCGC0** register (see page 186).
2. If required by the application, reconfigure the sample sequencer priorities in the **ADCSSPRI** register. The default configuration has Sample Sequencer 0 with the highest priority, and Sample Sequencer 3 as the lowest priority.

### 10.3.2 Sample Sequencer Configuration

Configuration of the sample sequencers is slightly more complex than the module initialization since each sample sequence is completely programmable.

The configuration for each sample sequencer should be as follows:

1. Ensure that the sample sequencer is disabled by writing a 0 to the corresponding ASENn bit in the **ADCACTSS** register. Programming of the sample sequencers is allowed without having them enabled. Disabling the sequencer during programming prevents erroneous execution if a trigger event were to occur during the configuration process.
2. Configure the trigger event for the sample sequencer in the **ADCEMUX** register.
3. For each sample in the sample sequence, configure the corresponding input source in the **ADCSSMUXn** register.
4. For each sample in the sample sequence, configure the sample control bits in the corresponding nibble in the **ADCSSCTLn** register. When programming the last nibble, ensure that the END bit is set. Failure to set the END bit causes unpredictable behavior.
5. If interrupts are to be used, write a 1 to the corresponding MASK bit in the **ADCIM** register.
6. Enable the sample sequencer logic by writing a 1 to the corresponding ASENn bit in the **ADCACTSS** register.

## 10.4 Register Map

Table 10-3 on page 326 lists the ADC registers. The offset listed is a hexadecimal increment to the register's address, relative to the ADC base address of 0x4003.8000.

Note that the ADC module clock must be enabled before the registers can be programmed (see page 186). There must be a delay of 3 system clocks after the ADC module clock is enabled before any ADC module registers are accessed.

**Table 10-3. ADC Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | ADCACTSS | R/W | 0x0000.0000 | ADC Active Sample Sequencer | 328 |
| 0x004 | ADCRIS | RO | 0x0000.0000 | ADC Raw Interrupt Status | 329 |
| 0x008 | ADCIM | R/W | 0x0000.0000 | ADC Interrupt Mask | 330 |
| 0x00C | ADCISC | R/W1C | 0x0000.0000 | ADC Interrupt Status and Clear | 331 |
| 0x010 | ADCOSTAT | R/W1C | 0x0000.0000 | ADC Overflow Status | 332 |
| 0x014 | ADCEMUX | R/W | 0x0000.0000 | ADC Event Multiplexer Select | 333 |
| 0x018 | ADCUSTAT | R/W1C | 0x0000.0000 | ADC Underflow Status | 336 |
| 0x020 | ADCSSPRI | R/W | 0x0000.3210 | ADC Sample Sequencer Priority | 337 |
| 0x028 | ADCPSSI | WO | - | ADC Processor Sample Sequence Initiate | 339 |
| 0x030 | ADCSAC | R/W | 0x0000.0000 | ADC Sample Averaging Control | 340 |
| 0x040 | ADCSSMUX0 | R/W | 0x0000.0000 | ADC Sample Sequence Input Multiplexer Select 0 | 341 |
| 0x044 | ADCSSCTL0 | R/W | 0x0000.0000 | ADC Sample Sequence Control 0 | 343 |
| 0x048 | ADCSSFIFO0 | RO | - | ADC Sample Sequence Result FIFO 0 | 346 |
| 0x04C | ADCSSFSTAT0 | RO | 0x0000.0100 | ADC Sample Sequence FIFO 0 Status | 347 |

**Table 10-3. ADC Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x060 | ADCSSMUX1 | R/W | 0x0000.0000 | ADC Sample Sequence Input Multiplexer Select 1 | 348 |
| 0x064 | ADCSSCTL1 | R/W | 0x0000.0000 | ADC Sample Sequence Control 1 | 349 |
| 0x068 | ADCSSFIFO1 | RO | - | ADC Sample Sequence Result FIFO 1 | 346 |
| 0x06C | ADCSSFSTAT1 | RO | 0x0000.0100 | ADC Sample Sequence FIFO 1 Status | 347 |
| 0x080 | ADCSSMUX2 | R/W | 0x0000.0000 | ADC Sample Sequence Input Multiplexer Select 2 | 348 |
| 0x084 | ADCSSCTL2 | R/W | 0x0000.0000 | ADC Sample Sequence Control 2 | 349 |
| 0x088 | ADCSSFIFO2 | RO | - | ADC Sample Sequence Result FIFO 2 | 346 |
| 0x08C | ADCSSFSTAT2 | RO | 0x0000.0100 | ADC Sample Sequence FIFO 2 Status | 347 |
| 0x0A0 | ADCSSMUX3 | R/W | 0x0000.0000 | ADC Sample Sequence Input Multiplexer Select 3 | 351 |
| 0x0A4 | ADCSSCTL3 | R/W | 0x0000.0002 | ADC Sample Sequence Control 3 | 352 |
| 0x0A8 | ADCSSFIFO3 | RO | - | ADC Sample Sequence Result FIFO 3 | 346 |
| 0x0AC | ADCSSFSTAT3 | RO | 0x0000.0100 | ADC Sample Sequence FIFO 3 Status | 347 |
| 0x100 | ADCTMLB | R/W | 0x0000.0000 | ADC Test Mode Loopback | 353 |

## 10.5 Register Descriptions

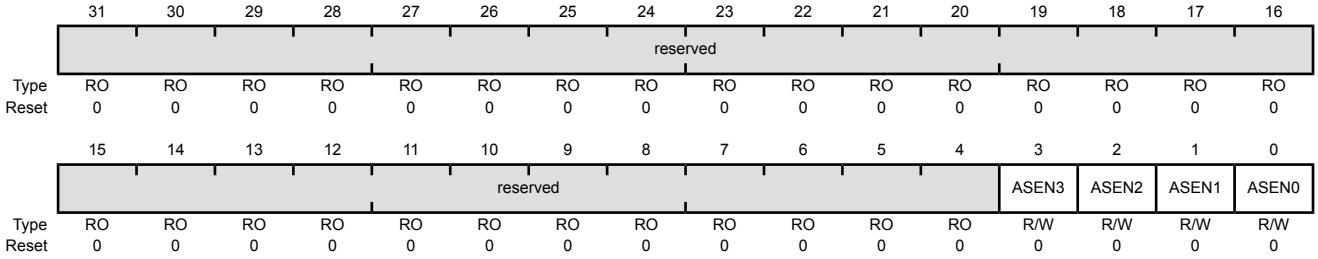
The remainder of this section lists and describes the ADC registers, in numerical order by address offset.

## Register 1: ADC Active Sample Sequencer (ADCACTSS), offset 0x000

This register controls the activation of the sample sequencers. Each sample sequencer can be enabled or disabled independently.

ADC Active Sample Sequencer (ADCACTSS)

Base 0x4003.8000
Offset 0x000
Type R/W, reset 0x0000.0000



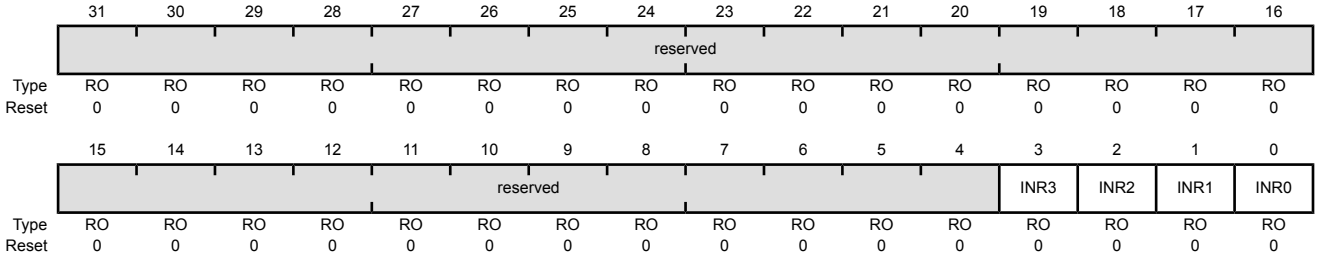

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | ASEN3 | R/W | 0 | ADC SS3 Enable<br>Specifies whether Sample Sequencer 3 is enabled. If set, the sample sequence logic for Sequencer 3 is active. Otherwise, the sequencer is inactive. |
| 2 | ASEN2 | R/W | 0 | ADC SS2 Enable<br>Specifies whether Sample Sequencer 2 is enabled. If set, the sample sequence logic for Sequencer 2 is active. Otherwise, the sequencer is inactive. |
| 1 | ASEN1 | R/W | 0 | ADC SS1 Enable<br>Specifies whether Sample Sequencer 1 is enabled. If set, the sample sequence logic for Sequencer 1 is active. Otherwise, the sequencer is inactive. |
| 0 | ASEN0 | R/W | 0 | ADC SS0 Enable<br>Specifies whether Sample Sequencer 0 is enabled. If set, the sample sequence logic for Sequencer 0 is active. Otherwise, the sequencer is inactive. |

## Register 2: ADC Raw Interrupt Status (ADCRIS), offset 0x004

This register shows the status of the raw interrupt signal of each sample sequencer. These bits may be polled by software to look for interrupt conditions without having to generate controller interrupts.

ADC Raw Interrupt Status (ADCRIS)

Base 0x4003.8000
Offset 0x004
Type RO, reset 0x0000.0000



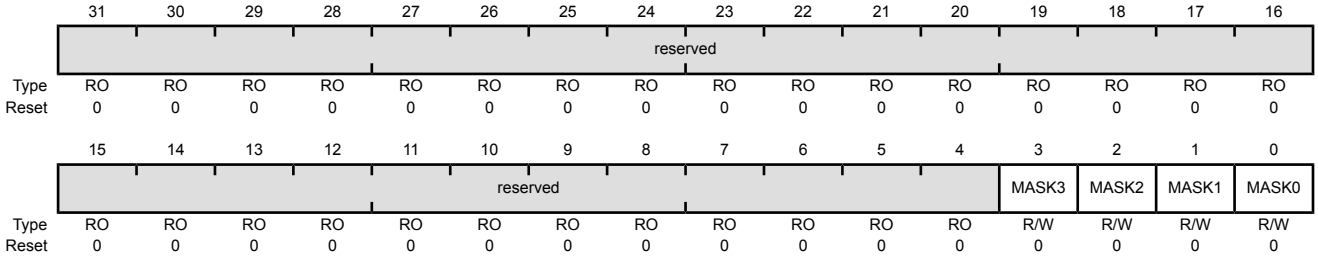

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | INR3 | RO | 0 | SS3 Raw Interrupt Status<br>This bit is set by hardware when a sample with its respective **ADCSSCTL3** `IE` bit has completed conversion. This bit is cleared by setting the `IN3` bit in the **ADCISC** register. |
| 2 | INR2 | RO | 0 | SS2 Raw Interrupt Status<br>This bit is set by hardware when a sample with its respective **ADCSSCTL2** `IE` bit has completed conversion. This bit is cleared by setting the `IN2` bit in the **ADCISC** register. |
| 1 | INR1 | RO | 0 | SS1 Raw Interrupt Status<br>This bit is set by hardware when a sample with its respective **ADCSSCTL1** `IE` bit has completed conversion. This bit is cleared by setting the `IN1` bit in the **ADCISC** register. |
| 0 | INR0 | RO | 0 | SS0 Raw Interrupt Status<br>This bit is set by hardware when a sample with its respective **ADCSSCTL0** `IE` bit has completed conversion. This bit is cleared by setting the `IN30` bit in the **ADCISC** register. |

## Register 3: ADC Interrupt Mask (ADCIM), offset 0x008

This register controls whether the sample sequencer raw interrupt signals are promoted to controller interrupts. Each raw interrupt signal can be masked independently.

ADC Interrupt Mask (ADCIM)

Base 0x4003.8000
Offset 0x008
Type R/W, reset 0x0000.0000



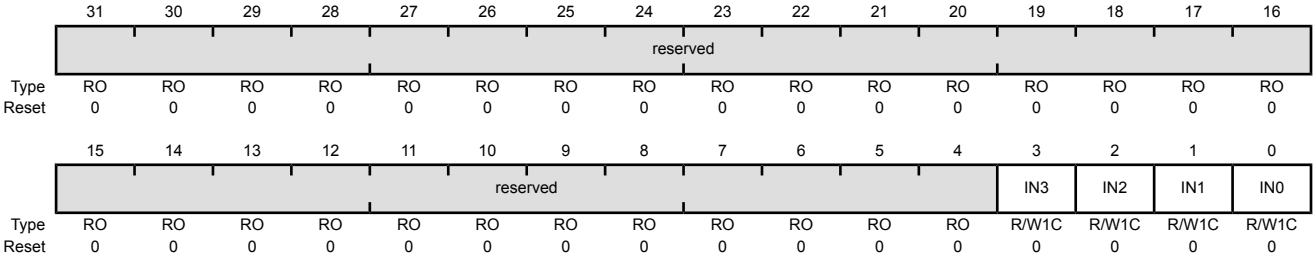

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | MASK3 | R/W | 0 | SS3 Interrupt Mask<br>When set, this bit allows the raw interrupt signal from Sample Sequencer 3 (**ADCRIS** register `INR3` bit) to be promoted to a controller interrupt.<br>When clear, the status of Sample Sequencer 3 does not affect the SS3 interrupt status. |
| 2 | MASK2 | R/W | 0 | SS2 Interrupt Mask<br>When set, this bit allows the raw interrupt signal from Sample Sequencer 2 (**ADCRIS** register `INR2` bit) to be promoted to a controller interrupt.<br>When clear, the status of Sample Sequencer 2 does not affect the SS2 interrupt status. |
| 1 | MASK1 | R/W | 0 | SS1 Interrupt Mask<br>When set, this bit allows the raw interrupt signal from Sample Sequencer 1 (**ADCRIS** register `INR1` bit) to be promoted to a controller interrupt.<br>When clear, the status of Sample Sequencer 1 does not affect the SS1 interrupt status. |
| 0 | MASK0 | R/W | 0 | SS0 Interrupt Mask<br>When set, this bit allows the raw interrupt signal from Sample Sequencer 0 (**ADCRIS** register `INR0` bit) to be promoted to a controller interrupt.<br>When clear, the status of Sample Sequencer 0 does not affect the SS0 interrupt status. |

## Register 4: ADC Interrupt Status and Clear (ADCISC), offset 0x00C

This register provides the mechanism for clearing sample sequence interrupt conditions and shows the status of controller interrupts generated by the sample sequencers. When read, each bit field is the logical AND of the respective INR and MASK bits. Sample sequence nterrupts are cleared by setting the corresponding bit position. If software is polling the **ADCRIS** instead of generating interrupts, the sample sequence INR bits are still cleared via the **ADCISC** register, even if the IN bit is not set.

ADC Interrupt Status and Clear (ADCISC)

Base 0x4003.8000
Offset 0x00C
Type R/W1C, reset 0x0000.0000



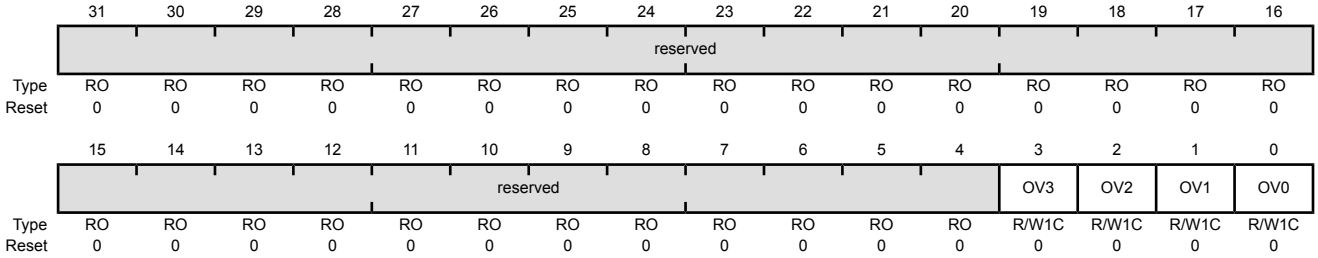

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | IN3 | R/W1C | 0 | SS3 Interrupt Status and Clear<br>This bit is set when both the INR3 bit in the **ADCRIS** register and the MASK3 bit in the **ADCIM** register are set, providing a level-based interrupt to the controller.<br>This bit is cleared by writing a 1. Clearing this bit also clears the INR3 bit. |
| 2 | IN2 | R/W1C | 0 | SS2 Interrupt Status and Clear<br>This bit is set when both the INR2 bit in the **ADCRIS** register and the MASK2 bit in the **ADCIM** register are set, providing a level-based interrupt to the controller.<br>This bit is cleared by writing a 1. Clearing this bit also clears the INR2 bit. |
| 1 | IN1 | R/W1C | 0 | SS1 Interrupt Status and Clear<br>This bit is set when both the INR1 bit in the **ADCRIS** register and the MASK1 bit in the **ADCIM** register are set, providing a level-based interrupt to the controller.<br>This bit is cleared by writing a 1. Clearing this bit also clears the INR1 bit. |
| 0 | IN0 | R/W1C | 0 | SS0 Interrupt Status and Clear<br>This bit is set when both the INR0 bit in the **ADCRIS** register and the MASK0 bit in the **ADCIM** register are set, providing a level-based interrupt to the controller.<br>This bit is cleared by writing a 1. Clearing this bit also clears the INR0 bit. |

## Register 5: ADC Overflow Status (ADCOSTAT), offset 0x010

This register indicates overflow conditions in the sample sequencer FIFOs. Once the overflow condition has been handled by software, the condition can be cleared by writing a 1 to the corresponding bit position.

ADC Overflow Status (ADCOSTAT)

Base 0x4003.8000
Offset 0x010
Type R/W1C, reset 0x0000.0000



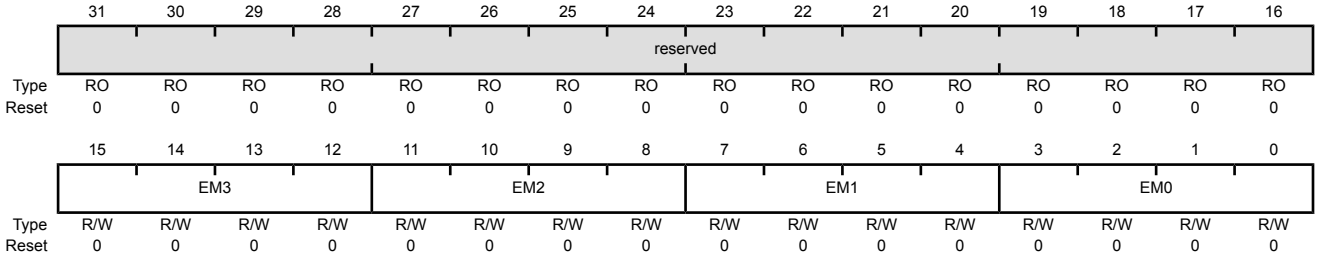

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | OV3 | R/W1C | 0 | SS3 FIFO Overflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 3 has hit an overflow condition where the FIFO is full and a write was requested. When an overflow is detected, the most recent write is dropped.<br>This bit is cleared by writing a 1. |
| 2 | OV2 | R/W1C | 0 | SS2 FIFO Overflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 2 has hit an overflow condition where the FIFO is full and a write was requested. When an overflow is detected, the most recent write is dropped.<br>This bit is cleared by writing a 1. |
| 1 | OV1 | R/W1C | 0 | SS1 FIFO Overflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 1 has hit an overflow condition where the FIFO is full and a write was requested. When an overflow is detected, the most recent write is dropped.<br>This bit is cleared by writing a 1. |
| 0 | OV0 | R/W1C | 0 | SS0 FIFO Overflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 0 has hit an overflow condition where the FIFO is full and a write was requested. When an overflow is detected, the most recent write is dropped.<br>This bit is cleared by writing a 1. |

## Register 6: ADC Event Multiplexer Select (ADCEMUX), offset 0x014

The **ADCEMUX** selects the event (trigger) that initiates sampling for each sample sequencer. Each sample sequencer can be configured with a unique trigger source.

ADC Event Multiplexer Select (ADCEMUX)

Base 0x4003.8000
Offset 0x014
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:12 | EM3 | R/W | 0x0 | SS3 Trigger Select<br>This field selects the trigger source for Sample Sequencer 3.<br>The valid configurations for this field are: |

| Value | Event |
|---|---|
| 0x0 | Controller (default) |
| 0x1 | Reserved |
| 0x2 | Reserved |
| 0x3 | Reserved |
| 0x4 | External (GPIO PB4) |
| 0x5 | Timer<br>In addition, the trigger must be enabled with the TnOTE bit in the **GPTMCTL** register (see page 276). |
| 0x6 | reserved |
| 0x7 | reserved |
| 0x8 | reserved |
| 0x9-0xE | reserved |
| 0xF | Always (continuously sample) |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 11:8 | EM2 | R/W | 0x0 | SS2 Trigger Select<br>This field selects the trigger source for Sample Sequencer 2.<br>The valid configurations for this field are:<br><br>Value Event<br>0x0 Controller (default)<br>0x1 Reserved<br>0x2 Reserved<br>0x3 Reserved<br>0x4 External (GPIO PB4)<br>0x5 Timer<br>In addition, the trigger must be enabled with the TnOTE bit in the **GPTMCTL** register (see page 276).<br>0x6 reserved<br>0x7 reserved<br>0x8 reserved<br>0x9-0xE reserved<br>0xF Always (continuously sample) |
| 7:4 | EM1 | R/W | 0x0 | SS1 Trigger Select<br>This field selects the trigger source for Sample Sequencer 1.<br>The valid configurations for this field are:<br><br>Value Event<br>0x0 Controller (default)<br>0x1 Reserved<br>0x2 Reserved<br>0x3 Reserved<br>0x4 External (GPIO PB4)<br>0x5 Timer<br>In addition, the trigger must be enabled with the TnOTE bit in the **GPTMCTL** register (see page 276).<br>0x6 reserved<br>0x7 reserved<br>0x8 reserved<br>0x9-0xE reserved<br>0xF Always (continuously sample) |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3:0 | EM0 | R/W | 0x0 | SS0 Trigger Select<br>This field selects the trigger source for Sample Sequencer 0.<br>The valid configurations for this field are: |

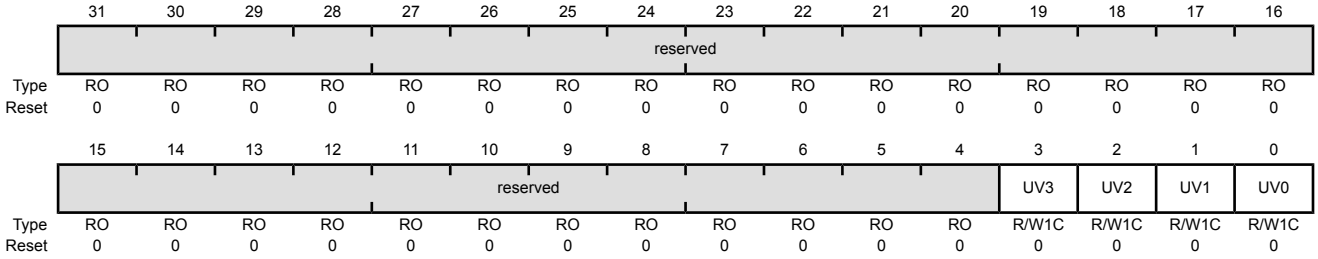
| Value | Event |
|---|---|
| 0x0 | Controller (default) |
| 0x1 | Reserved |
| 0x2 | Reserved |
| 0x3 | Reserved |
| 0x4 | External (GPIO PB4) |
| 0x5 | Timer<br>In addition, the trigger must be enabled with the TnOTE bit in the **GPTMCTL** register (see page 276). |
| 0x6 | reserved |
| 0x7 | reserved |
| 0x8 | reserved |
| 0x9-0xE | reserved |
| 0xF | Always (continuously sample) |

## Register 7: ADC Underflow Status (ADCUSTAT), offset 0x018

This register indicates underflow conditions in the sample sequencer FIFOs. The corresponding underflow condition is cleared by writing a 1 to the relevant bit position.

ADC Underflow Status (ADCUSTAT)

Base 0x4003.8000
Offset 0x018
Type R/W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | UV3 | R/W1C | 0 | SS3 FIFO Underflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 3 has hit an underflow condition where the FIFO is empty and a read was requested. The problematic read does not move the FIFO pointers, and 0s are returned.<br>This bit is cleared by writing a 1. |
| 2 | UV2 | R/W1C | 0 | SS2 FIFO Underflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 2 has hit an underflow condition where the FIFO is empty and a read was requested. The problematic read does not move the FIFO pointers, and 0s are returned.<br>This bit is cleared by writing a 1. |
| 1 | UV1 | R/W1C | 0 | SS1 FIFO Underflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 1 has hit an underflow condition where the FIFO is empty and a read was requested. The problematic read does not move the FIFO pointers, and 0s are returned.<br>This bit is cleared by writing a 1. |
| 0 | UV0 | R/W1C | 0 | SS0 FIFO Underflow<br>When set, this bit specifies that the FIFO for Sample Sequencer 0 has hit an underflow condition where the FIFO is empty and a read was requested. The problematic read does not move the FIFO pointers, and 0s are returned.<br>This bit is cleared by writing a 1. |

## Register 8: ADC Sample Sequencer Priority (ADCSSPRI), offset 0x020

This register sets the priority for each of the sample sequencers. Out of reset, Sequencer 0 has the highest priority, and Sequencer 3 has the lowest priority. When reconfiguring sequence priorities, each sequence must have a unique priority for the ADC to operate properly.

ADC Sample Sequencer Priority (ADCSSPRI)

Base 0x4003.8000
Offset 0x020
Type R/W, reset 0x0000.3210


| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:14 | reserved | RO | 0x0000.0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 13:12 | SS3 | R/W | 0x3 | SS3 Priority<br>This field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 3. A priority encoding of 0 is highest and 3 is lowest. The priorities assigned to the sequencers must be uniquely mapped. The ADC may not operate properly if two or more fields are equal. |
| 11:10 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9:8 | SS2 | R/W | 0x2 | SS2 Priority<br>This field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 2. A priority encoding of 0 is highest and 3 is lowest. The priorities assigned to the sequencers must be uniquely mapped. The ADC may not operate properly if two or more fields are equal. |
| 7:6 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:4 | SS1 | R/W | 0x1 | SS1 Priority<br>This field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 1. A priority encoding of 0 is highest and 3 is lowest. The priorities assigned to the sequencers must be uniquely mapped. The ADC may not operate properly if two or more fields are equal. |
| 3:2 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

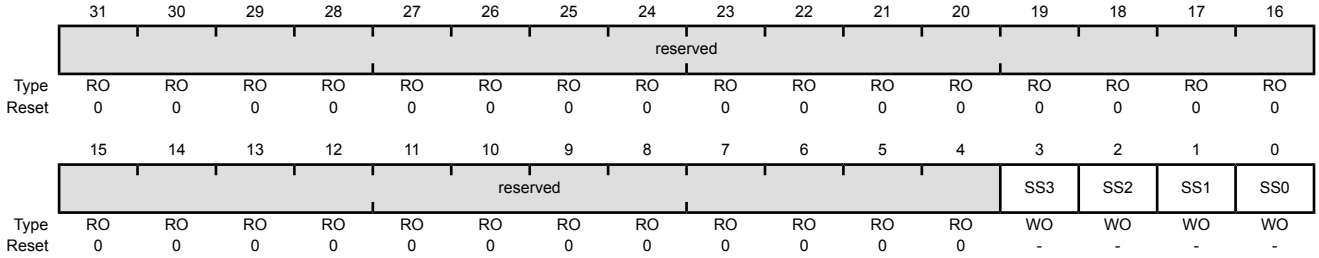
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1:0 | SS0 | R/W | 0x0 | SS0 Priority<br>This field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 0. A priority encoding of 0 is highest and 3 is lowest. The priorities assigned to the sequencers must be uniquely mapped. The ADC may not operate properly if two or more fields are equal. |

## Register 9: ADC Processor Sample Sequence Initiate (ADCPSSI), offset 0x028

This register provides a mechanism for application software to initiate sampling in the sample sequencers. Sample sequences can be initiated individually or in any combination. When multiple sequences are triggered simultaneously, the priority encodings in **ADCSSPRI** dictate execution order.

ADC Processor Sample Sequence Initiate (ADCPSSI)

Base 0x4003.8000
Offset 0x028
Type WO, reset -



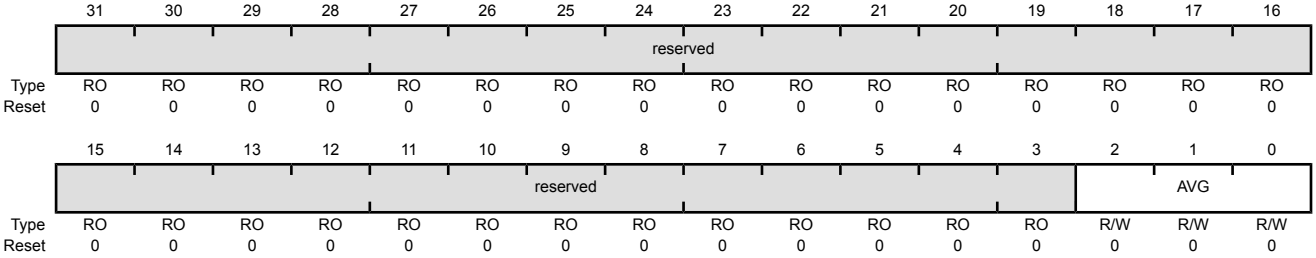

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | SS3 | WO | - | SS3 Initiate<br>When set, this bit triggers sampling on Sample Sequencer 3 if the sequencer is enabled in the **ADCACTSS** register.<br>Only a write by software is valid; a read of this register returns no meaningful data. |
| 2 | SS2 | WO | - | SS2 Initiate<br>When set, this bit triggers sampling on Sample Sequencer 2 if the sequencer is enabled in the **ADCACTSS** register.<br>Only a write by software is valid; a read of this register returns no meaningful data. |
| 1 | SS1 | WO | - | SS1 Initiate<br>When set, this bit triggers sampling on Sample Sequencer 1 if the sequencer is enabled in the **ADCACTSS** register.<br>Only a write by software is valid; a read of this register returns no meaningful data. |
| 0 | SS0 | WO | - | SS0 Initiate<br>When set, this bit triggers sampling on Sample Sequencer 0 if the sequencer is enabled in the **ADCACTSS** register.<br>Only a write by software is valid; a read of this register returns no meaningful data. |

## Register 10: ADC Sample Averaging Control (ADCSAC), offset 0x030

This register controls the amount of hardware averaging applied to conversion results. The final conversion result stored in the FIFO is averaged from $2^{AVG}$ consecutive ADC samples at the specified ADC speed. If AVG is 0, the sample is passed directly through without any averaging. If AVG=6, then 64 consecutive ADC samples are averaged to generate one result in the sequencer FIFO. An AVG = 7 provides unpredictable results.

ADC Sample Averaging Control (ADCSAC)

Base 0x4003.8000
Offset 0x030
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | AVG | R/W | 0x0 | Hardware Averaging Control<br>Specifies the amount of hardware averaging that will be applied to ADC samples. The AVG field can be any value between 0 and 6. Entering a value of 7 creates unpredictable results. |

| Value | Description |
|---|---|
| 0x0 | No hardware oversampling |
| 0x1 | 2x hardware oversampling |
| 0x2 | 4x hardware oversampling |
| 0x3 | 8x hardware oversampling |
| 0x4 | 16x hardware oversampling |
| 0x5 | 32x hardware oversampling |
| 0x6 | 64x hardware oversampling |
| 0x7 | Reserved |

## Register 11: ADC Sample Sequence Input Multiplexer Select 0 (ADCSSMUX0), offset 0x040

This register defines the analog input configuration for each sample in a sequence executed with Sample Sequencer 0. This register is 32 bits wide and contains information for eight possible samples.

ADC Sample Sequence Input Multiplexer Select 0 (ADCSSMUX0)

Base 0x4003.8000
Offset 0x040
Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | MUX7 | | | reserved | MUX6 | | | reserved | MUX5 | | | reserved | MUX4 | | |
| Type | RO | R/W | R/W | R/W | RO | R/W | R/W | R/W | RO | R/W | R/W | R/W | RO | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | MUX3 | | | reserved | MUX2 | | | reserved | MUX1 | | | reserved | MUX0 | | |
| Type | RO | R/W | R/W | R/W | RO | R/W | R/W | R/W | RO | R/W | R/W | R/W | RO | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 30:28 | MUX7 | R/W | 0x0 | 8th Sample Input Select<br>The MUX7 field is used during the eighth sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. The value set here indicates the corresponding pin, for example, a value of 1 indicates the input is ADC1. |
| 27 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 26:24 | MUX6 | R/W | 0x0 | 7th Sample Input Select<br>The MUX6 field is used during the seventh sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 23 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 22:20 | MUX5 | R/W | 0x0 | 6th Sample Input Select<br>The MUX5 field is used during the sixth sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 19 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 18:16 | MUX4 | R/W | 0x0 | 5th Sample Input Select<br>The MUX4 field is used during the fifth sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 15 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14:12 | MUX3 | R/W | 0x0 | 4th Sample Input Select<br>The MUX3 field is used during the fourth sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 11 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10:8 | MUX2 | R/W | 0x0 | 3rd Sample Input Select<br>The MUX72 field is used during the third sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:4 | MUX1 | R/W | 0x0 | 2nd Sample Input Select<br>The MUX1 field is used during the second sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |
| 3 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | MUX0 | R/W | 0x0 | 1st Sample Input Select<br>The MUX0 field is used during the first sample of a sequence executed with the sample sequencer. It specifies which of the analog inputs is sampled for the analog-to-digital conversion. |

## Register 12: ADC Sample Sequence Control 0 (ADCSSCTL0), offset 0x044

This register contains the configuration information for each sample for a sequence executed with a sample sequencer. When configuring a sample sequence, the END bit must be set at some point, whether it be after the first sample, last sample, or any sample in between. This register is 32-bits wide and contains information for eight possible samples.

ADC Sample Sequence Control 0 (ADCSSCTL0)

Base 0x4003.8000
Offset 0x044
Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TS7 | IE7 | END7 | D7 | TS6 | IE6 | END6 | D6 | TS5 | IE5 | END5 | D5 | TS4 | IE4 | END4 | D4 |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | TS3 | IE3 | END3 | D3 | TS2 | IE2 | END2 | D2 | TS1 | IE1 | END1 | D1 | TS0 | IE0 | END0 | D0 |
| Type | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31 | TS7 | R/W | 0 | 8th Sample Temp Sensor Select<br>This bit is used during the eighth sample of the sample sequence and and specifies the input source of the sample.<br>When set, the temperature sensor is read.<br>When clear, the input pin specified by the **ADCSSMUX** register is read. |
| 30 | IE7 | R/W | 0 | 8th Sample Interrupt Enable<br>This bit is used during the eighth sample of the sample sequence and specifies whether the raw interrupt signal (INR0 bit) is asserted at the end of the sample's conversion. If the MASK0 bit in the **ADCIM** register is set, the interrupt is promoted to a controller-level interrupt.<br>When this bit is set, the raw interrupt is asserted.<br>When this bit is clear, the raw interrupt is not asserted.<br>It is legal to have multiple samples within a sequence generate interrupts. |
| 29 | END7 | R/W | 0 | 8th Sample is End of Sequence<br>The END7 bit indicates that this is the last sample of the sequence. It is possible to end the sequence on any sample position. Samples defined after the sample containing a set END are not requested for conversion even though the fields may be non-zero. It is required that software write the END bit somewhere within the sequence. (Sample Sequencer 3, which only has a single sample in the sequence, is hardwired to have the END0 bit set.)<br>Setting this bit indicates that this sample is the last in the sequence. |
| 28 | D7 | R/W | 0 | 8th Sample Diff Input Select<br>The D7 bit indicates that the analog input is to be differentially sampled. The corresponding **ADCSSMUXx** nibble must be set to the pair number "i", where the paired inputs are "2i and 2i+1". The temperature sensor does not have a differential option. When set, the analog inputs are differentially sampled. |
| 27 | TS6 | R/W | 0 | 7th Sample Temp Sensor Select<br>Same definition as TS7 but used during the seventh sample. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 26 | IE6 | R/W | 0 | 7th Sample Interrupt Enable<br>Same definition as IE7 but used during the seventh sample. |
| 25 | END6 | R/W | 0 | 7th Sample is End of Sequence<br>Same definition as END7 but used during the seventh sample. |
| 24 | D6 | R/W | 0 | 7th Sample Diff Input Select<br>Same definition as D7 but used during the seventh sample. |
| 23 | TS5 | R/W | 0 | 6th Sample Temp Sensor Select<br>Same definition as TS7 but used during the sixth sample. |
| 22 | IE5 | R/W | 0 | 6th Sample Interrupt Enable<br>Same definition as IE7 but used during the sixth sample. |
| 21 | END5 | R/W | 0 | 6th Sample is End of Sequence<br>Same definition as END7 but used during the sixth sample. |
| 20 | D5 | R/W | 0 | 6th Sample Diff Input Select<br>Same definition as D7 but used during the sixth sample. |
| 19 | TS4 | R/W | 0 | 5th Sample Temp Sensor Select<br>Same definition as TS7 but used during the fifth sample. |
| 18 | IE4 | R/W | 0 | 5th Sample Interrupt Enable<br>Same definition as IE7 but used during the fifth sample. |
| 17 | END4 | R/W | 0 | 5th Sample is End of Sequence<br>Same definition as END7 but used during the fifth sample. |
| 16 | D4 | R/W | 0 | 5th Sample Diff Input Select<br>Same definition as D7 but used during the fifth sample. |
| 15 | TS3 | R/W | 0 | 4th Sample Temp Sensor Select<br>Same definition as TS7 but used during the fourth sample. |
| 14 | IE3 | R/W | 0 | 4th Sample Interrupt Enable<br>Same definition as IE7 but used during the fourth sample. |
| 13 | END3 | R/W | 0 | 4th Sample is End of Sequence<br>Same definition as END7 but used during the fourth sample. |
| 12 | D3 | R/W | 0 | 4th Sample Diff Input Select<br>Same definition as D7 but used during the fourth sample. |
| 11 | TS2 | R/W | 0 | 3rd Sample Temp Sensor Select<br>Same definition as TS7 but used during the third sample. |
| 10 | IE2 | R/W | 0 | 3rd Sample Interrupt Enable<br>Same definition as IE7 but used during the third sample. |
| 9 | END2 | R/W | 0 | 3rd Sample is End of Sequence<br>Same definition as END7 but used during the third sample. |
| 8 | D2 | R/W | 0 | 3rd Sample Diff Input Select<br>Same definition as D7 but used during the third sample. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | TS1 | R/W | 0 | 2nd Sample Temp Sensor Select<br>Same definition as TS7 but used during the second sample. |
| 6 | IE1 | R/W | 0 | 2nd Sample Interrupt Enable<br>Same definition as IE7 but used during the second sample. |
| 5 | END1 | R/W | 0 | 2nd Sample is End of Sequence<br>Same definition as END7 but used during the second sample. |
| 4 | D1 | R/W | 0 | 2nd Sample Diff Input Select<br>Same definition as D7 but used during the second sample. |
| 3 | TS0 | R/W | 0 | 1st Sample Temp Sensor Select<br>Same definition as TS7 but used during the first sample. |
| 2 | IE0 | R/W | 0 | 1st Sample Interrupt Enable<br>Same definition as IE7 but used during the first sample. |
| 1 | END0 | R/W | 0 | 1st Sample is End of Sequence<br>Same definition as END7 but used during the first sample. |
| 0 | D0 | R/W | 0 | 1st Sample Diff Input Select<br>Same definition as D7 but used during the first sample. |

### Register 13: ADC Sample Sequence Result FIFO 0 (ADCSSFIFO0), offset 0x048

### Register 14: ADC Sample Sequence Result FIFO 1 (ADCSSFIFO1), offset 0x068

### Register 15: ADC Sample Sequence Result FIFO 2 (ADCSSFIFO2), offset 0x088

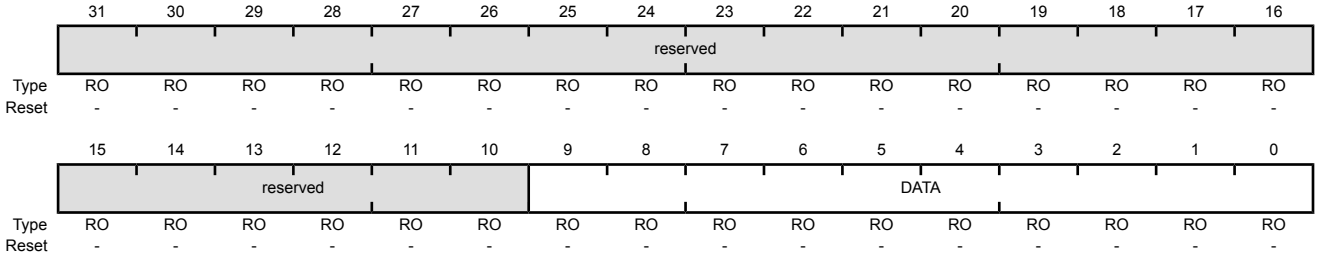
### Register 16: ADC Sample Sequence Result FIFO 3 (ADCSSFIFO3), offset 0x0A8

**Important:** Use caution when reading this register. Performing a read may change bit status.

This register contains the conversion results for samples collected with the sample sequencer (the **ADCSSFIFO0** register is used for Sample Sequencer 0, **ADCSSFIFO1** for Sequencer 1, **ADCSSFIFO2** for Sequencer 2, and **ADCSSFIFO3** for Sequencer 3). Reads of this register return conversion result data in the order sample 0, sample 1, and so on, until the FIFO is empty. If the FIFO is not properly handled by software, overflow and underflow conditions are registered in the **ADCOSTAT** and **ADCUSTAT** registers.

ADC Sample Sequence Result FIFO 0 (ADCSSFIFO0)

Base 0x4003.8000
Offset 0x048
Type RO, reset -




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | reserved | RO | - | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9:0 | DATA | RO | - | Conversion Result Data |

### Register 17: ADC Sample Sequence FIFO 0 Status (ADCSSFSTAT0), offset 0x04C

### Register 18: ADC Sample Sequence FIFO 1 Status (ADCSSFSTAT1), offset 0x06C

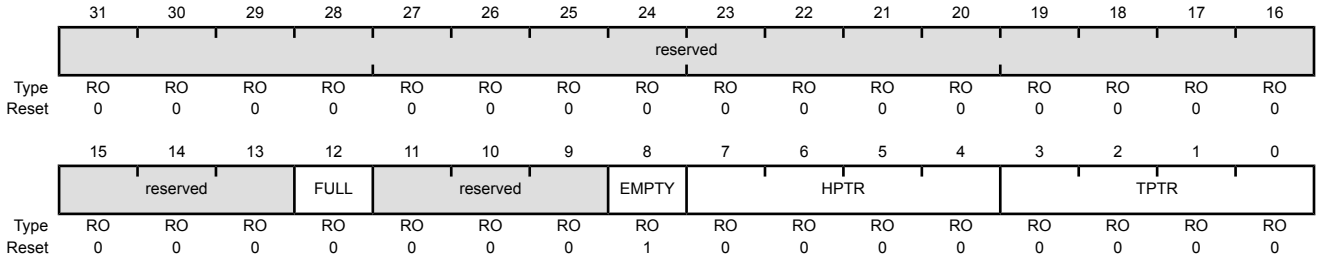
### Register 19: ADC Sample Sequence FIFO 2 Status (ADCSSFSTAT2), offset 0x08C

### Register 20: ADC Sample Sequence FIFO 3 Status (ADCSSFSTAT3), offset 0x0AC

This register provides a window into the sample sequencer, providing full/empty status information as well as the positions of the head and tail pointers. The reset value of 0x100 indicates an empty FIFO. The **ADCSSFSTAT0** register provides status on FIFO0, **ADCSSFSTAT1** on FIFO1, **ADCSSFSTAT2** on FIFO2, and **ADCSSFSTAT3** on FIFO3.

ADC Sample Sequence FIFO 0 Status (ADCSSFSTAT0)

Base 0x4003.8000
Offset 0x04C
Type RO, reset 0x0000.0100




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:13 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 12 | FULL | RO | 0 | FIFO Full<br>When set, this bit indicates that the FIFO is currently full. |
| 11:9 | reserved | RO | 0x0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 8 | EMPTY | RO | 1 | FIFO Empty<br>When set, this bit indicates that the FIFO is currently empty. |
| 7:4 | HPTR | RO | 0x0 | FIFO Head Pointer<br>This field contains the current "head" pointer index for the FIFO, that is, the next entry to be written. |
| 3:0 | TPTR | RO | 0x0 | FIFO Tail Pointer<br>This field contains the current "tail" pointer index for the FIFO, that is, the next entry to be read. |

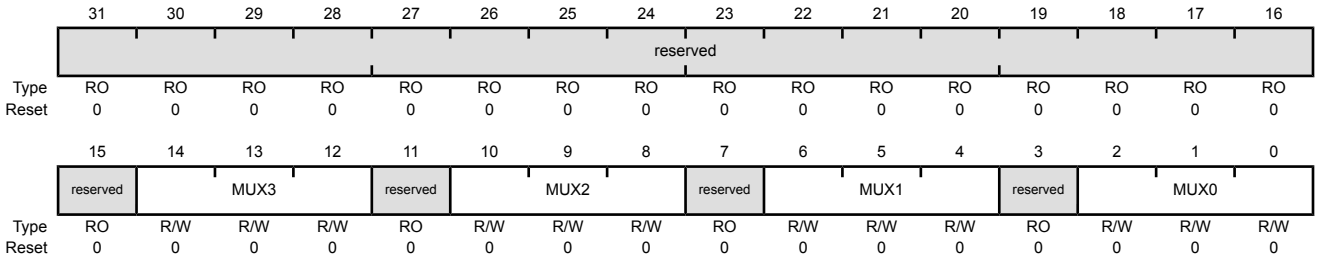
## Register 21: ADC Sample Sequence Input Multiplexer Select 1 (ADCSSMUX1), offset 0x060

## Register 22: ADC Sample Sequence Input Multiplexer Select 2 (ADCSSMUX2), offset 0x080

This register defines the analog input configuration for each sample in a sequence executed with Sample Sequencer 1 or 2. These registers are 16-bits wide and contain information for four possible samples. See the **ADCSSMUX0** register on page 341 for detailed bit descriptions. The **ADCSSMUX1** register affects Sample Sequencer 1 and the **ADCSSMUX2** register affects Sample Sequencer 2.

ADC Sample Sequence Input Multiplexer Select 1 (ADCSSMUX1)

Base 0x4003.8000
Offset 0x060
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:15 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 14:12 | MUX3 | R/W | 0x0 | 4th Sample Input Select |
| 11 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10:8 | MUX2 | R/W | 0x0 | 3rd Sample Input Select |
| 7 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:4 | MUX1 | R/W | 0x0 | 2nd Sample Input Select |
| 3 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | MUX0 | R/W | 0x0 | 1st Sample Input Select |

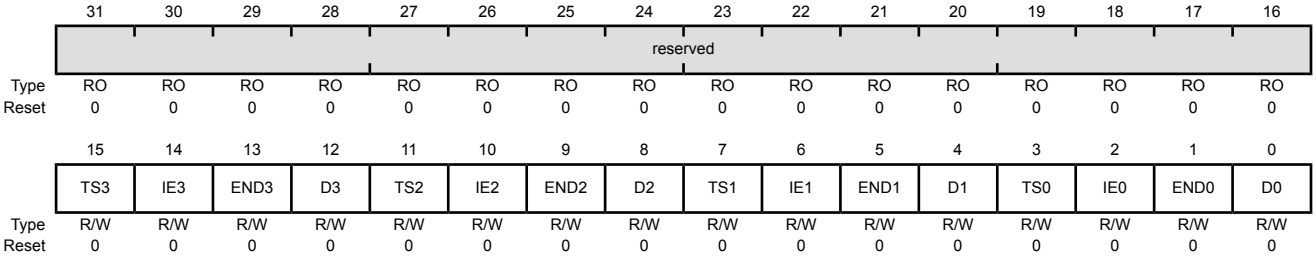
## Register 23: ADC Sample Sequence Control 1 (ADCSSCTL1), offset 0x064

## Register 24: ADC Sample Sequence Control 2 (ADCSSCTL2), offset 0x084

These registers contain the configuration information for each sample for a sequence executed with Sample Sequencer 1 or 2. When configuring a sample sequence, the END bit must be set at some point, whether it be after the first sample, last sample, or any sample in between. These registers are 16-bits wide and contain information for four possible samples. See the **ADCSSCTL0** register on page 343 for detailed bit descriptions. The **ADCSSCTL1** register configures Sample Sequencer 1 and the **ADCSSCTL2** register configures Sample Sequencer 2.

ADC Sample Sequence Control 1 (ADCSSCTL1)

Base 0x4003.8000
Offset 0x064
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15 | TS3 | R/W | 0 | 4th Sample Temp Sensor Select<br>Same definition as TS7 but used during the fourth sample. |
| 14 | IE3 | R/W | 0 | 4th Sample Interrupt Enable<br>Same definition as IE7 but used during the fourth sample. |
| 13 | END3 | R/W | 0 | 4th Sample is End of Sequence<br>Same definition as END7 but used during the fourth sample. |
| 12 | D3 | R/W | 0 | 4th Sample Diff Input Select<br>Same definition as D7 but used during the fourth sample. |
| 11 | TS2 | R/W | 0 | 3rd Sample Temp Sensor Select<br>Same definition as TS7 but used during the third sample. |
| 10 | IE2 | R/W | 0 | 3rd Sample Interrupt Enable<br>Same definition as IE7 but used during the third sample. |
| 9 | END2 | R/W | 0 | 3rd Sample is End of Sequence<br>Same definition as END7 but used during the third sample. |
| 8 | D2 | R/W | 0 | 3rd Sample Diff Input Select<br>Same definition as D7 but used during the third sample. |
| 7 | TS1 | R/W | 0 | 2nd Sample Temp Sensor Select<br>Same definition as TS7 but used during the second sample. |

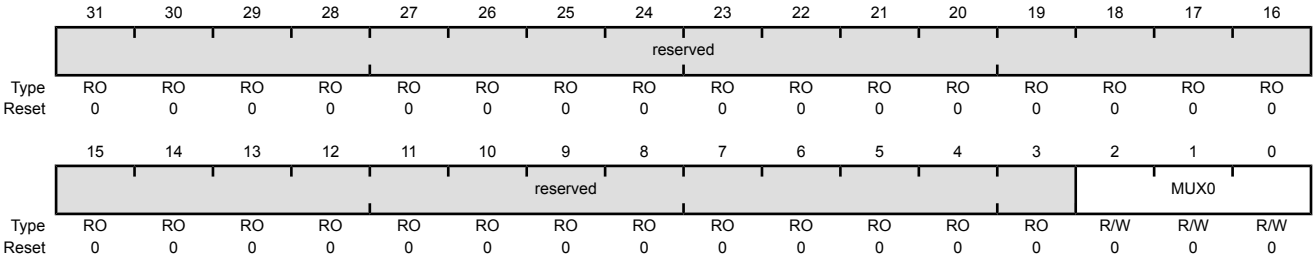
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | IE1 | R/W | 0 | 2nd Sample Interrupt Enable<br>Same definition as IE7 but used during the second sample. |
| 5 | END1 | R/W | 0 | 2nd Sample is End of Sequence<br>Same definition as END7 but used during the second sample. |
| 4 | D1 | R/W | 0 | 2nd Sample Diff Input Select<br>Same definition as D7 but used during the second sample. |
| 3 | TS0 | R/W | 0 | 1st Sample Temp Sensor Select<br>Same definition as TS7 but used during the first sample. |
| 2 | IE0 | R/W | 0 | 1st Sample Interrupt Enable<br>Same definition as IE7 but used during the first sample. |
| 1 | END0 | R/W | 0 | 1st Sample is End of Sequence<br>Same definition as END7 but used during the first sample. |
| 0 | D0 | R/W | 0 | 1st Sample Diff Input Select<br>Same definition as D7 but used during the first sample. |

## Register 25: ADC Sample Sequence Input Multiplexer Select 3 (ADCSSMUX3), offset 0x0A0

This register defines the analog input configuration for a sample executed with Sample Sequencer 3. This register is 4-bits wide and contains information for one possible sample. See the **ADCSSMUX0** register on page 341 for detailed bit descriptions.

ADC Sample Sequence Input Multiplexer Select 3 (ADCSSMUX3)

Base 0x4003.8000
Offset 0x0A0
Type R/W, reset 0x0000.0000



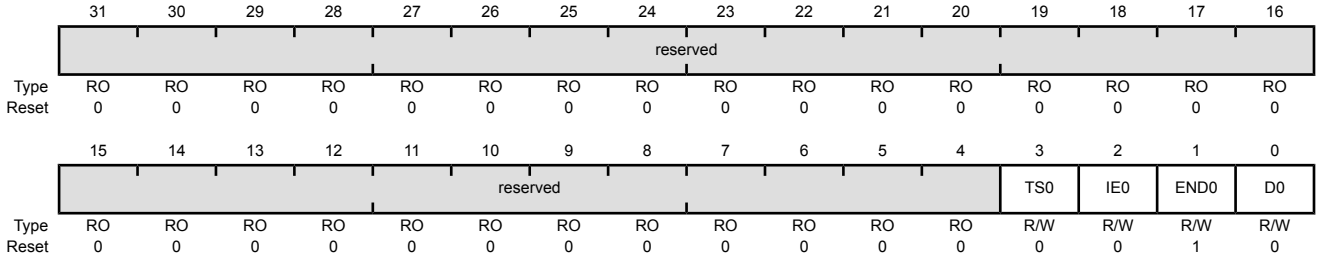

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2:0 | MUX0 | R/W | 0 | 1st Sample Input Select |

## Register 26: ADC Sample Sequence Control 3 (ADCSSCTL3), offset 0x0A4

This register contains the configuration information for a sample executed with Sample Sequencer 3. The END bit is always set since there is only one sample in this sequencer. This register is 4-bits wide and contains information for one possible sample. See the **ADCSSCTL0** register on page 343 for detailed bit descriptions.

ADC Sample Sequence Control 3 (ADCSSCTL3)

Base 0x4003.8000
Offset 0x0A4
Type R/W, reset 0x0000.0002



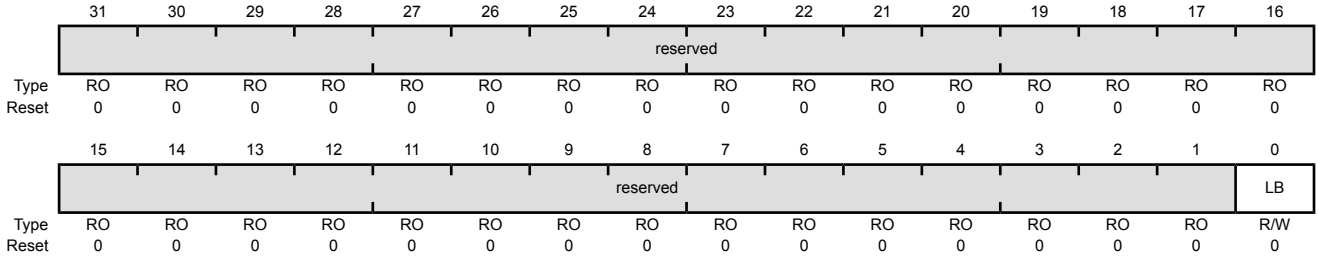

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TS0 | R/W | 0 | 1st Sample Temp Sensor Select<br>Same definition as TS7 but used during the first sample. |
| 2 | IE0 | R/W | 0 | 1st Sample Interrupt Enable<br>Same definition as IE7 but used during the first sample. |
| 1 | END0 | R/W | 1 | 1st Sample is End of Sequence<br>Same definition as END7 but used during the first sample.<br>Since this sequencer has only one entry, this bit must be set. |
| 0 | D0 | R/W | 0 | 1st Sample Diff Input Select<br>Same definition as D7 but used during the first sample. |

## Register 27: ADC Test Mode Loopback (ADCTMLB), offset 0x100

This register provides loopback operation within the digital logic of the ADC, which can be useful in debugging software without having to provide actual analog stimulus. This test mode is entered by writing a value of 0x0000.0001 to this register. When data is read from the FIFO in loopback mode, the read-only portion of this register is returned.

ADC Test Mode Loopback (ADCTMLB)

Base 0x4003.8000
Offset 0x100
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x0000.000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | LB | R/W | 0 | Loopback Mode Enable<br>When set, forces a loopback within the digital block to provide information on input and unique numbering. The **ADCSSFIFOn** registers do not provide sample data, but instead provide the 10-bit loopback data as shown below. |

| Bit/Field | Name | Description |
|---|---|---|
| 9:6 | CNT | Continuous Sample Counter<br>Continuous sample counter that is initialized to 0 and counts each sample as it processed. This helps provide a unique value for the data received. |
| 5 | CONT | Continuation Sample Indicator<br>When set, indicates that this is a continuation sample. For example, if two sequencers were to run back-to-back, this indicates that the controller kept continuously sampling at full rate. |
| 4 | DIFF | Differential Sample Indicator<br>When set, indicates that this is a differential sample. |
| 3 | TS | Temp Sensor Sample Indicator<br>When set, indicates that this is a temperature sensor sample. |
| 2:0 | MUX | Analog Input Indicator<br>Indicates which analog input is to be sampled. |

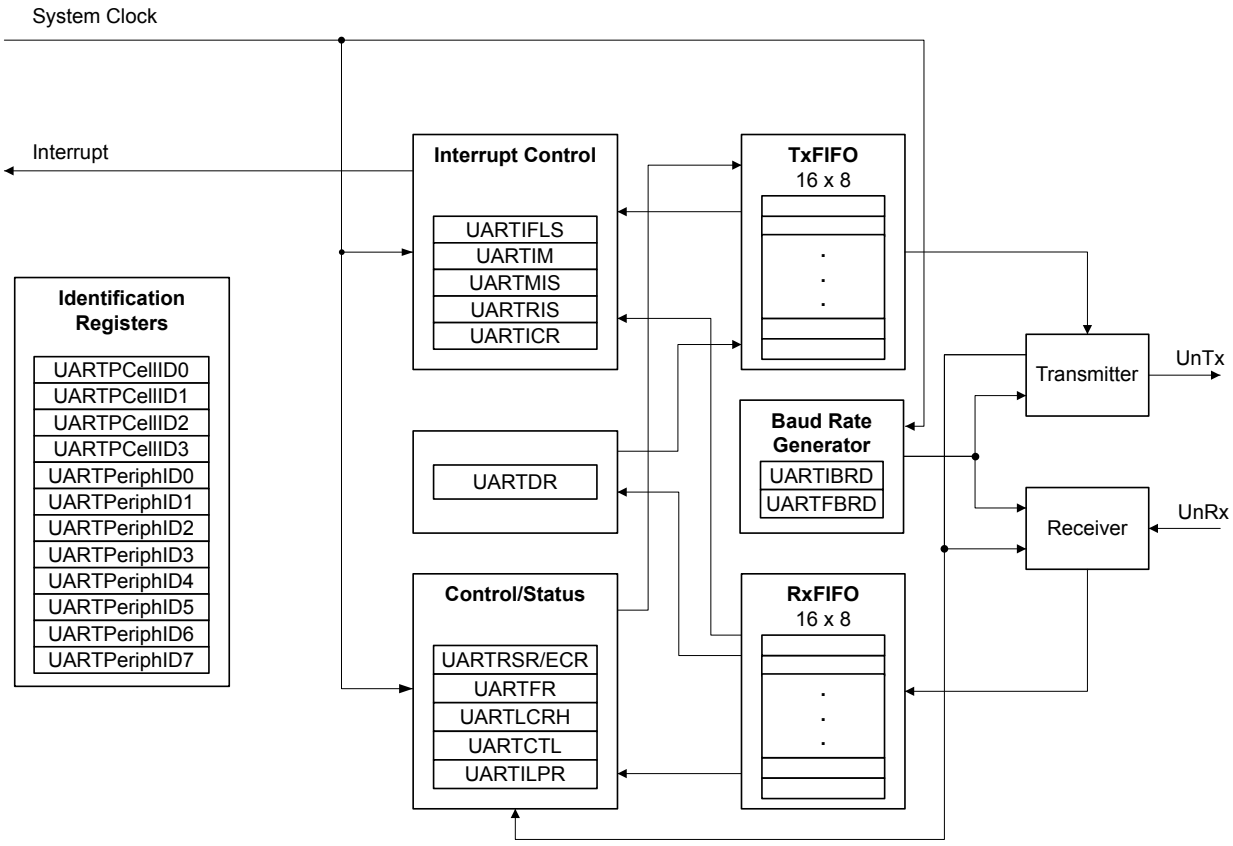
# 11 Universal Asynchronous Receivers/Transmitters (UARTs)

Each Stellaris® Universal Asynchronous Receiver/Transmitter (UART) has the following features:

- Two fully programmable 16C550-type UARTs
- Separate 16x8 transmit (TX) and receive (RX) FIFOs to reduce CPU interrupt service loading
- Programmable baud-rate generator allowing speeds up to 3.125 Mbps
- Programmable FIFO length, including 1-byte deep operation providing conventional double-buffered interface
- FIFO trigger levels of 1/8, 1/4, 1/2, 3/4, and 7/8
- Standard asynchronous communication bits for start, stop, and parity
- Line-break generation and detection
- Fully programmable serial interface characteristics
  - 5, 6, 7, or 8 data bits
  - Even, odd, stick, or no-parity bit generation/detection
  - 1 or 2 stop bit generation

## 11.1 Block Diagram

**Figure 11-1. UART Module Block Diagram**




## 11.2 Functional Description

Each Stellaris UART performs the functions of parallel-to-serial and serial-to-parallel conversions. It is similar in functionality to a 16C550 UART, but is not register compatible.

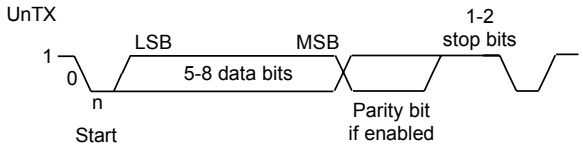
The UART is configured for transmit and/or receive via the TXE and RXE bits of the **UART Control (UARTCTL)** register (see page 371). Transmit and receive are both enabled out of reset. Before any control registers are programmed, the UART must be disabled by clearing the UARTEN bit in **UARTCTL**. If the UART is disabled during a TX or RX operation, the current transaction is completed prior to the UART stopping.

### 11.2.1 Transmit/Receive Logic

The transmit logic performs parallel-to-serial conversion on the data read from the transmit FIFO. The control logic outputs the serial bit stream beginning with a start bit, and followed by the data bits (LSB first), parity bit, and the stop bits according to the programmed configuration in the control registers. See Figure 11-2 on page 356 for details.

The receive logic performs serial-to-parallel conversion on the received bit stream after a valid start pulse has been detected. Overrun, parity, frame error checking, and line-break detection are also performed, and their status accompanies the data that is written to the receive FIFO.

**Figure 11-2. UART Character Frame**




## 11.2.2 Baud-Rate Generation

The baud-rate divisor is a 22-bit number consisting of a 16-bit integer and a 6-bit fractional part. The number formed by these two values is used by the baud-rate generator to determine the bit period. Having a fractional baud-rate divider allows the UART to generate all the standard baud rates.

The 16-bit integer is loaded through the **UART Integer Baud-Rate Divisor (UARTIBRD)** register (see page 367) and the 6-bit fractional part is loaded with the **UART Fractional Baud-Rate Divisor (UARTFBRD)** register (see page 368). The baud-rate divisor (BRD) has the following relationship to the system clock (where *BRDI* is the integer part of the BRD and *BRDF* is the fractional part, separated by a decimal place.)

```
BRD = BRDI + BRDF = UARTSysClk / (16 * Baud Rate)
```

where UARTSysClk is the system clock connected to the UART.

The 6-bit fractional number (that is to be loaded into the DIVFRAC bit field in the **UARTFBRD** register) can be calculated by taking the fractional part of the baud-rate divisor, multiplying it by 64, and adding 0.5 to account for rounding errors:

```
UARTFBRD[DIVFRAC] = integer(BRDF * 64 + 0.5)
```

The UART generates an internal baud-rate reference clock at 16x the baud-rate (referred to as Baud16). This reference clock is divided by 16 to generate the transmit clock, and is used for error detection during receive operations.

Along with the **UART Line Control, High Byte (UARTLCRH)** register (see page 369), the **UARTIBRD** and **UARTFBRD** registers form an internal 30-bit register. This internal register is only updated when a write operation to **UARTLCRH** is performed, so any changes to the baud-rate divisor must be followed by a write to the **UARTLCRH** register for the changes to take effect.

To update the baud-rate registers, there are four possible sequences:

- **UARTIBRD** write, **UARTFBRD** write, and **UARTLCRH** write
- **UARTFBRD** write, **UARTIBRD** write, and **UARTLCRH** write
- **UARTIBRD** write and **UARTLCRH** write
- **UARTFBRD** write and **UARTLCRH** write

## 11.2.3 Data Transmission

Data received or transmitted is stored in two 16-byte FIFOs, though the receive FIFO has an extra four bits per character for status information. For transmission, data is written into the transmit FIFO. If the UART is enabled, it causes a data frame to start transmitting with the parameters indicated in the **UARTLCRH** register. Data continues to be transmitted until there is no data left in the transmit FIFO. The BUSY bit in the **UART Flag (UARTFR)** register (see page 365) is asserted as soon as

data is written to the transmit FIFO (that is, if the FIFO is non-empty) and remains asserted while data is being transmitted. The BUSY bit is negated only when the transmit FIFO is empty, and the last character has been transmitted from the shift register, including the stop bits. The UART can indicate that it is busy even though the UART may no longer be enabled.

When the receiver is idle (the UnRx is continuously 1) and the data input goes Low (a start bit has been received), the receive counter begins running and data is sampled on the eighth cycle of Baud16 (described in "Transmit/Receive Logic" on page 355).

The start bit is valid and recognized if UnRx is still low on the eighth cycle of Baud16, otherwise it is ignored. After a valid start bit is detected, successive data bits are sampled on every 16th cycle of Baud16 (that is, one bit period later) according to the programmed length of the data characters. The parity bit is then checked if parity mode was enabled. Data length and parity are defined in the **UARTLCRH** register.

Lastly, a valid stop bit is confirmed if UnRx is High, otherwise a framing error has occurred. When a full word is received, the data is stored in the receive FIFO, with any error bits associated with that word.

### 11.2.4 FIFO Operation

The UART has two 16-entry FIFOs; one for transmit and one for receive. Both FIFOs are accessed via the **UART Data (UARTDR)** register (see page 361). Read operations of the **UARTDR** register return a 12-bit value consisting of 8 data bits and 4 error flags while write operations place 8-bit data in the transmit FIFO.

Out of reset, both FIFOs are disabled and act as 1-byte-deep holding registers. The FIFOs are enabled by setting the FEN bit in **UARTLCRH** (page 369).

FIFO status can be monitored via the **UART Flag (UARTFR)** register (see page 365) and the **UART Receive Status (UARTRSR)** register. Hardware monitors empty, full and overrun conditions. The **UARTFR** register contains empty and full flags (TXFE, TXFF, RXFE, and RXFF bits) and the **UARTRSR** register shows overrun status via the OE bit.

The trigger points at which the FIFOs generate interrupts is controlled via the **UART Interrupt FIFO Level Select (UARTIFLS)** register (see page 373). Both FIFOs can be individually configured to trigger interrupts at different levels. Available configurations include 1/8, ¼, ½, ¾, and 7/8. For example, if the ¼ option is selected for the receive FIFO, the UART generates a receive interrupt after 4 data bytes are received. Out of reset, both FIFOs are configured to trigger an interrupt at the ½ mark.

### 11.2.5 Interrupts

The UART can generate interrupts when the following conditions are observed:

- Overrun Error
- Break Error
- Parity Error
- Framing Error
- Receive Timeout
- Transmit (when condition defined in the TXIFLSEL bit in the **UARTIFLS** register is met)

- Receive (when condition defined in the RXIFLSEL bit in the **UARTIFLS** register is met)

All of the interrupt events are ORed together before being sent to the interrupt controller, so the UART can only generate a single interrupt request to the controller at any given time. Software can service multiple interrupt events in a single interrupt service routine by reading the **UART Masked Interrupt Status (UARTMIS)** register (see page 378).

The interrupt events that can trigger a controller-level interrupt are defined in the **UART Interrupt Mask (UARTIM** ) register (see page 375) by setting the corresponding IM bit to 1. If interrupts are not used, the raw interrupt status is always visible via the **UART Raw Interrupt Status (UARTRIS)** register (see page 377).

Interrupts are always cleared (for both the **UARTMIS** and **UARTRIS** registers) by setting the corresponding bit in the **UART Interrupt Clear (UARTICR)** register (see page 379).

The receive timeout interrupt is asserted when the receive FIFO is not empty, and no further data is received over a 32-bit period. The receive timeout interrupt is cleared either when the FIFO becomes empty through reading all the data (or by reading the holding register), or when a 1 is written to the corresponding bit in the **UARTICR** register.

### 11.2.6 Loopback Operation

The UART can be placed into an internal loopback mode for diagnostic or debug work. This is accomplished by setting the LBE bit in the **UARTCTL** register (see page 371). In loopback mode, data transmitted on UnTx is received on the UnRx input.

## 11.3 Initialization and Configuration

To use the UARTs, the peripheral clock must be enabled by setting the UART0 or UART1 bits in the **RCGC1** register.

This section discusses the steps that are required to use a UART module. For this example, the UART clock is assumed to be 20 MHz and the desired UART configuration is:

- 115200 baud rate
- Data length of 8 bits
- One stop bit
- No parity
- FIFOs disabled
- No interrupts

The first thing to consider when programming the UART is the baud-rate divisor (BRD), since the **UARTIBRD** and **UARTFBRD** registers must be written before the **UARTLCRH** register. Using the equation described in “Baud-Rate Generation” on page 356, the BRD can be calculated:

```
BRD = 20,000,000 / (16 * 115,200) = 10.8507
```

which means that the DIVINT field of the **UARTIBRD** register (see page 367) should be set to 10. The value to be loaded into the **UARTFBRD** register (see page 368) is calculated by the equation:

```
UARTFBRD[DIVFRAC] = integer(0.8507 * 64 + 0.5) = 54
```

With the BRD values in hand, the UART configuration is written to the module in the following order:

1. Disable the UART by clearing the UARTEN bit in the **UARTCTL** register.
2. Write the integer portion of the BRD to the **UARTIBRD** register.
3. Write the fractional portion of the BRD to the **UARTFBRD** register.
4. Write the desired serial parameters to the **UARTLCRH** register (in this case, a value of 0x0000.0060).
5. Enable the UART by setting the UARTEN bit in the **UARTCTL** register.

## 11.4 Register Map

Table 11-1 on page 359 lists the UART registers. The offset listed is a hexadecimal increment to the register's address, relative to that UART's base address:

- UART0: 0x4000.C000
- UART1: 0x4000.D000

Note that the UART module clock must be enabled before the registers can be programmed (see page 191). There must be a delay of 3 system clocks after the UART module clock is enabled before any UART module registers are accessed.

**Note:** The UART must be disabled (see the UARTEN bit in the **UARTCTL** register on page 371) before any of the control registers are reprogrammed. When the UART is disabled during a TX or RX operation, the current transaction is completed prior to the UART stopping.

**Table 11-1. UART Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | UARTDR | R/W | 0x0000.0000 | UART Data | 361 |
| 0x004 | UARTRSR/UARTECR | R/W | 0x0000.0000 | UART Receive Status/Error Clear | 363 |
| 0x018 | UARTFR | RO | 0x0000.0090 | UART Flag | 365 |
| 0x024 | UARTIBRD | R/W | 0x0000.0000 | UART Integer Baud-Rate Divisor | 367 |
| 0x028 | UARTFBRD | R/W | 0x0000.0000 | UART Fractional Baud-Rate Divisor | 368 |
| 0x02C | UARTLCRH | R/W | 0x0000.0000 | UART Line Control | 369 |
| 0x030 | UARTCTL | R/W | 0x0000.0300 | UART Control | 371 |
| 0x034 | UARTIFLS | R/W | 0x0000.0012 | UART Interrupt FIFO Level Select | 373 |
| 0x038 | UARTIM | R/W | 0x0000.0000 | UART Interrupt Mask | 375 |
| 0x03C | UARTRIS | RO | 0x0000.000F | UART Raw Interrupt Status | 377 |
| 0x040 | UARTMIS | RO | 0x0000.0000 | UART Masked Interrupt Status | 378 |
| 0x044 | UARTICR | W1C | 0x0000.0000 | UART Interrupt Clear | 379 |
| 0xFD0 | UARTPeriphID4 | RO | 0x0000.0000 | UART Peripheral Identification 4 | 381 |
| 0xFD4 | UARTPeriphID5 | RO | 0x0000.0000 | UART Peripheral Identification 5 | 382 |

**Table 11-1. UART Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0xFD8 | UARTPeriphID6 | RO | 0x0000.0000 | UART Peripheral Identification 6 | 383 |
| 0xFDC | UARTPeriphID7 | RO | 0x0000.0000 | UART Peripheral Identification 7 | 384 |
| 0xFE0 | UARTPeriphID0 | RO | 0x0000.0011 | UART Peripheral Identification 0 | 385 |
| 0xFE4 | UARTPeriphID1 | RO | 0x0000.0000 | UART Peripheral Identification 1 | 386 |
| 0xFE8 | UARTPeriphID2 | RO | 0x0000.0018 | UART Peripheral Identification 2 | 387 |
| 0xFEC | UARTPeriphID3 | RO | 0x0000.0001 | UART Peripheral Identification 3 | 388 |
| 0xFF0 | UARTPCellID0 | RO | 0x0000.000D | UART PrimeCell Identification 0 | 389 |
| 0xFF4 | UARTPCellID1 | RO | 0x0000.00F0 | UART PrimeCell Identification 1 | 390 |
| 0xFF8 | UARTPCellID2 | RO | 0x0000.0005 | UART PrimeCell Identification 2 | 391 |
| 0xFFC | UARTPCellID3 | RO | 0x0000.00B1 | UART PrimeCell Identification 3 | 392 |

## 11.5 Register Descriptions

The remainder of this section lists and describes the UART registers, in numerical order by address offset.

## Register 1: UART Data (UARTDR), offset 0x000

**Important:** Use caution when reading this register. Performing a read may change bit status.

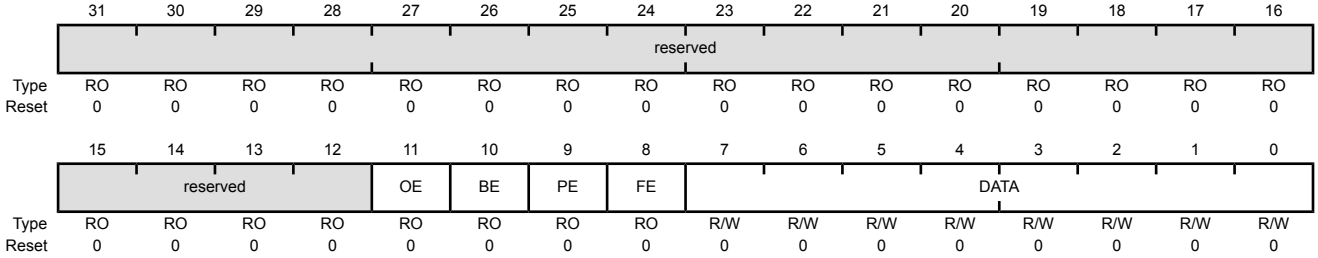
This register is the data register (the interface to the FIFOs).

When FIFOs are enabled, data written to this location is pushed onto the transmit FIFO. If FIFOs are disabled, data is stored in the transmitter holding register (the bottom word of the transmit FIFO). A write to this register initiates a transmission from the UART.

For received data, if the FIFO is enabled, the data byte and the 4-bit status (break, frame, parity, and overrun) is pushed onto the 12-bit wide receive FIFO. If FIFOs are disabled, the data byte and status are stored in the receiving holding register (the bottom word of the receive FIFO). The received data can be retrieved by reading this register.

UART Data (UARTDR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x000
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:12 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 11 | OE | RO | 0 | UART Overrun Error<br>The OE values are defined as follows:<br><br>Value Description<br>0 There has been no data loss due to a FIFO overrun.<br>1 New data was received when the FIFO was full, resulting in data loss. |
| 10 | BE | RO | 0 | UART Break Error<br>This bit is set to 1 when a break condition is detected, indicating that the receive data input was held Low for longer than a full-word transmission time (defined as start, data, parity, and stop bits).<br>In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the received data input goes to a 1 (marking state) and the next valid start bit is received. |
| 9 | PE | RO | 0 | UART Parity Error<br>This bit is set to 1 when the parity of the received data character does not match the parity defined by bits 2 and 7 of the **UARTLCRH** register.<br>In FIFO mode, this error is associated with the character at the top of the FIFO. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 8 | FE | RO | 0 | UART Framing Error<br>This bit is set to 1 when the received character does not have a valid stop bit (a valid stop bit is 1). |
| 7:0 | DATA | R/W | 0 | Data Transmitted or Received<br>When written, the data that is to be transmitted via the UART. When read, the data that was received by the UART. |

## Register 2: UART Receive Status/Error Clear (UARTRSR/UARTECR), offset 0x004

The **UARTRSR/UARTECR** register is the receive status register/error clear register.

In addition to the **UARTDR** register, receive status can also be read from the **UARTRSR** register. If the status is read from this register, then the status information corresponds to the entry read from **UARTDR** prior to reading **UARTRSR**. The status information for overrun is set immediately when an overrun condition occurs.

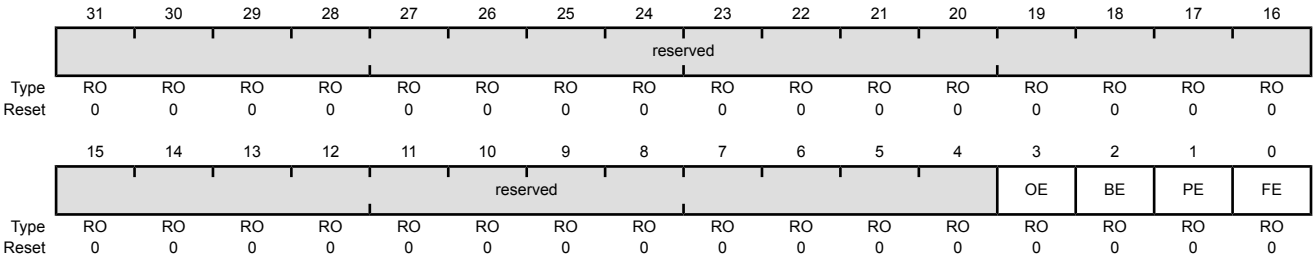
The **UARTRSR** register cannot be written.

A write of any value to the **UARTECR** register clears the framing, parity, break, and overrun errors. All the bits are cleared to 0 on reset.

### Reads

UART Receive Status/Error Clear (UARTRSR/UARTECR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x004
Type RO, reset 0x0000.0000



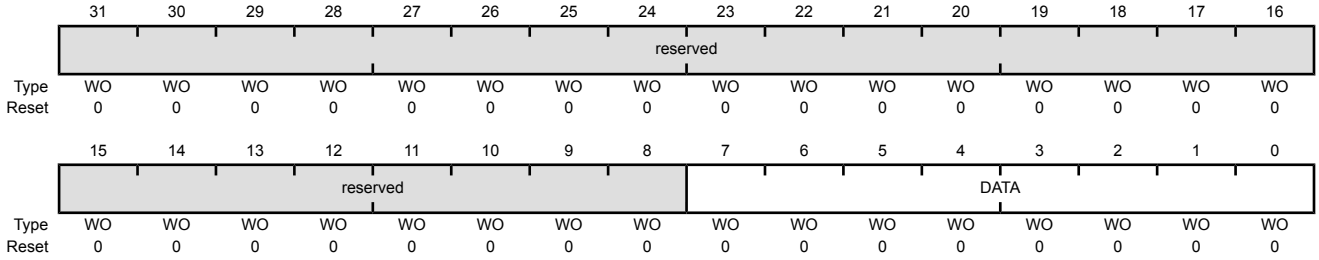

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | OE | RO | 0 | UART Overrun Error<br>When this bit is set to 1, data is received and the FIFO is already full. This bit is cleared to 0 by a write to **UARTECR**.<br>The FIFO contents remain valid since no further data is written when the FIFO is full, only the contents of the shift register are overwritten. The CPU must now read the data in order to empty the FIFO. |
| 2 | BE | RO | 0 | UART Break Error<br>This bit is set to 1 when a break condition is detected, indicating that the received data input was held Low for longer than a full-word transmission time (defined as start, data, parity, and stop bits).<br>This bit is cleared to 0 by a write to **UARTECR**.<br>In FIFO mode, this error is associated with the character at the top of the FIFO. When a break occurs, only one 0 character is loaded into the FIFO. The next character is only enabled after the receive data input goes to a 1 (marking state) and the next valid start bit is received. |
| 1 | PE | RO | 0 | UART Parity Error<br>This bit is set to 1 when the parity of the received data character does not match the parity defined by bits 2 and 7 of the **UARTLCRH** register.<br>This bit is cleared to 0 by a write to **UARTECR**. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | FE | RO | 0 | UART Framing Error<br>This bit is set to 1 when the received character does not have a valid stop bit (a valid stop bit is 1).<br>This bit is cleared to 0 by a write to **UARTECR**.<br>In FIFO mode, this error is associated with the character at the top of the FIFO. |

## Writes

UART Receive Status/Error Clear (UARTRSR/UARTECR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x004
Type WO, reset 0x0000.0000



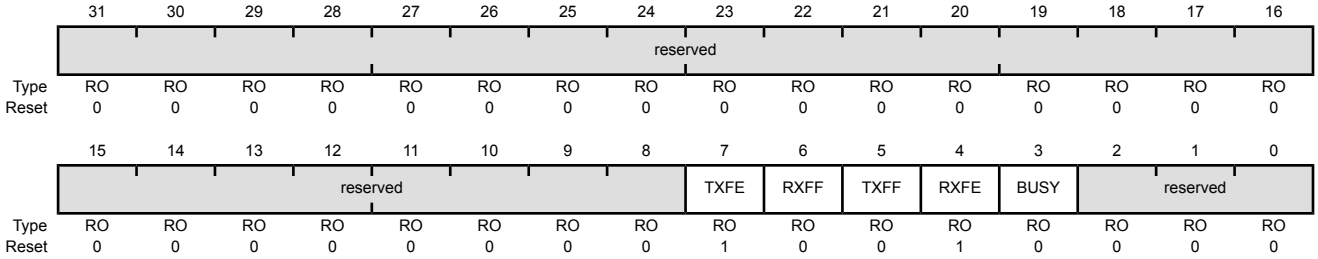

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | WO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DATA | WO | 0 | Error Clear<br>A write to this register of any data clears the framing, parity, break, and overrun flags. |

## Register 3: UART Flag (UARTFR), offset 0x018

The **UARTFR** register is the flag register. After reset, the TXFF, RXFF, and BUSY bits are 0, and TXFE and RXFE bits are 1.

UART Flag (UARTFR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x018
Type RO, reset 0x0000.0090




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7 | TXFE | RO | 1 | UART Transmit FIFO Empty<br>The meaning of this bit depends on the state of the FEN bit in the **UARTLCRH** register.<br>If the FIFO is disabled (FEN is 0), this bit is set when the transmit holding register is empty.<br>If the FIFO is enabled (FEN is 1), this bit is set when the transmit FIFO is empty. |
| 6 | RXFF | RO | 0 | UART Receive FIFO Full<br>The meaning of this bit depends on the state of the FEN bit in the **UARTLCRH** register.<br>If the FIFO is disabled, this bit is set when the receive holding register is full.<br>If the FIFO is enabled, this bit is set when the receive FIFO is full. |
| 5 | TXFF | RO | 0 | UART Transmit FIFO Full<br>The meaning of this bit depends on the state of the FEN bit in the **UARTLCRH** register.<br>If the FIFO is disabled, this bit is set when the transmit holding register is full.<br>If the FIFO is enabled, this bit is set when the transmit FIFO is full. |
| 4 | RXFE | RO | 1 | UART Receive FIFO Empty<br>The meaning of this bit depends on the state of the FEN bit in the **UARTLCRH** register.<br>If the FIFO is disabled, this bit is set when the receive holding register is empty.<br>If the FIFO is enabled, this bit is set when the receive FIFO is empty. |

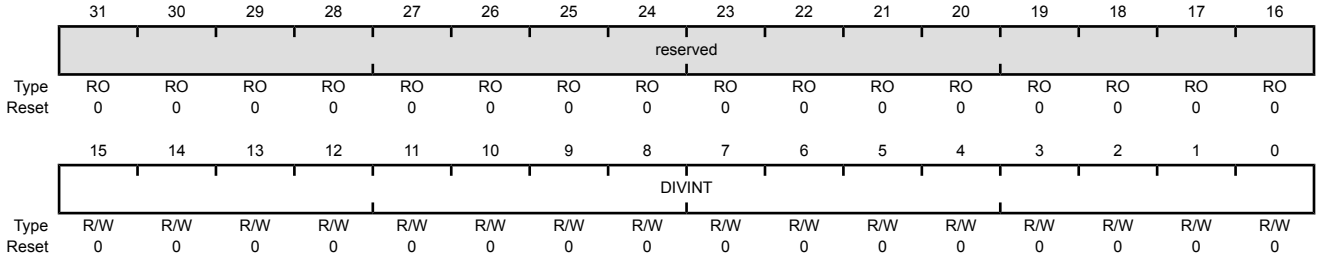
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | BUSY | RO | 0 | UART Busy<br>When this bit is 1, the UART is busy transmitting data. This bit remains set until the complete byte, including all stop bits, has been sent from the shift register.<br>This bit is set as soon as the transmit FIFO becomes non-empty (regardless of whether UART is enabled). |
| 2:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

### Register 4: UART Integer Baud-Rate Divisor (UARTIBRD), offset 0x024

The **UARTIBRD** register is the integer part of the baud-rate divisor value. All the bits are cleared on reset. The minimum possible divide ratio is 1 (when **UARTIBRD**=0), in which case the **UARTFBRD** register is ignored. When changing the **UARTIBRD** register, the new value does not take effect until transmission/reception of the current character is complete. Any changes to the baud-rate divisor must be followed by a write to the **UARTLCRH** register. See “Baud-Rate Generation” on page 356 for configuration details.

UART Integer Baud-Rate Divisor (UARTIBRD)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x024
Type R/W, reset 0x0000.0000



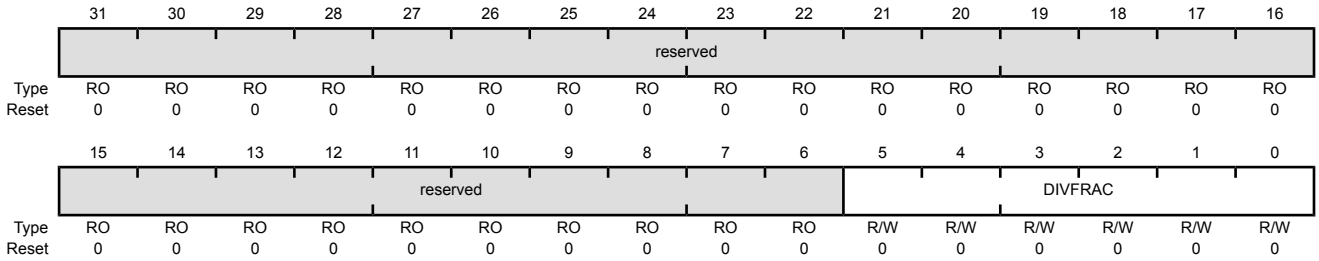

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | DIVINT | R/W | 0x0000 | Integer Baud-Rate Divisor |

## Register 5: UART Fractional Baud-Rate Divisor (UARTFBRD), offset 0x028

The **UARTFBRD** register is the fractional part of the baud-rate divisor value. All the bits are cleared on reset. When changing the **UARTFBRD** register, the new value does not take effect until transmission/reception of the current character is complete. Any changes to the baud-rate divisor must be followed by a write to the **UARTLCRH** register. See “Baud-Rate Generation” on page 356 for configuration details.

UART Fractional Baud-Rate Divisor (UARTFBRD)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x028
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:0 | DIVFRAC | R/W | 0x000 | Fractional Baud-Rate Divisor |

## Register 6: UART Line Control (UARTLCRH), offset 0x02C

The **UARTLCRH** register is the line control register. Serial parameters such as data length, parity, and stop bit selection are implemented in this register.

When updating the baud-rate divisor (**UARTIBRD** and/or **UARTIFRD**), the **UARTLCRH** register must also be written. The write strobe for the baud-rate divisor registers is tied to the **UARTLCRH** register.

UART Line Control (UARTLCRH)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x02C
Type R/W, reset 0x0000.0000


| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7 | SPS | R/W | 0 | UART Stick Parity Select<br>When bits 1, 2, and 7 of **UARTLCRH** are set, the parity bit is transmitted and checked as a 0. When bits 1 and 7 are set and 2 is cleared, the parity bit is transmitted and checked as a 1.<br>When this bit is cleared, stick parity is disabled. |
| 6:5 | WLEN | R/W | 0 | UART Word Length<br>The bits indicate the number of data bits transmitted or received in a frame as follows:<br>Value Description<br>0x3 8 bits<br>0x2 7 bits<br>0x1 6 bits<br>0x0 5 bits (default) |
| 4 | FEN | R/W | 0 | UART Enable FIFOs<br>If this bit is set to 1, transmit and receive FIFO buffers are enabled (FIFO mode).<br>When cleared to 0, FIFOs are disabled (Character mode). The FIFOs become 1-byte-deep holding registers. |
| 3 | STP2 | R/W | 0 | UART Two Stop Bits Select<br>If this bit is set to 1, two stop bits are transmitted at the end of a frame. The receive logic does not check for two stop bits being received. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2 | EPS | R/W | 0 | UART Even Parity Select<br>If this bit is set to 1, even parity generation and checking is performed during transmission and reception, which checks for an even number of 1s in data and parity bits.<br>When cleared to 0, then odd parity is performed, which checks for an odd number of 1s.<br>This bit has no effect when parity is disabled by the PEN bit. |
| 1 | PEN | R/W | 0 | UART Parity Enable<br>If this bit is set to 1, parity checking and generation is enabled; otherwise, parity is disabled and no parity bit is added to the data frame. |
| 0 | BRK | R/W | 0 | UART Send Break<br>If this bit is set to 1, a Low level is continually output on the UnTX output, after completing transmission of the current character. For the proper execution of the break command, the software must set this bit for at least two frames (character periods). For normal use, this bit must be cleared to 0. |

## Register 7: UART Control (UARTCTL), offset 0x030

The **UARTCTL** register is the control register. All the bits are cleared on reset except for the Transmit Enable (TXE) and Receive Enable (RXE) bits, which are set to 1.

To enable the UART module, the UARTEN bit must be set to 1. If software requires a configuration change in the module, the UARTEN bit must be cleared before the configuration changes are written. If the UART is disabled during a transmit or receive operation, the current transaction is completed prior to the UART stopping.

**Note:** The **UARTCTL** register should not be changed while the UART is enabled or else the results are unpredictable. The following sequence is recommended for making changes to the **UARTCTL** register.

1. Disable the UART.
2. Wait for the end of transmission or reception of the current character.
3. Flush the transmit FIFO by disabling bit 4 (FEN) in the line control register (**UARTLCRH**).
4. Reprogram the control register.
5. Enable the UART.

UART Control (UARTCTL)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x030
Type R/W, reset 0x0000.0300

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | RXE | TXE | LBE | reserved | | | | | | UARTEN |
| Type | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | RO | RO | RO | RO | RO | RO | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:10 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 9 | RXE | R/W | 1 | UART Receive Enable<br>If this bit is set to 1, the receive section of the UART is enabled. When the UART is disabled in the middle of a receive, it completes the current character before stopping.<br>**Note:** To enable reception, the UARTEN bit must also be set. |
| 8 | TXE | R/W | 1 | UART Transmit Enable<br>If this bit is set to 1, the transmit section of the UART is enabled. When the UART is disabled in the middle of a transmission, it completes the current character before stopping.<br>**Note:** To enable transmission, the UARTEN bit must also be set. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 7 | LBE | R/W | 0 | UART Loop Back Enable<br>If this bit is set to 1, the UnTX path is fed through the UnRX path. |
| 6:1 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | UARTEN | R/W | 0 | UART Enable<br>If this bit is set to 1, the UART is enabled. When the UART is disabled in the middle of transmission or reception, it completes the current character before stopping. |

## Register 8: UART Interrupt FIFO Level Select (UARTIFLS), offset 0x034

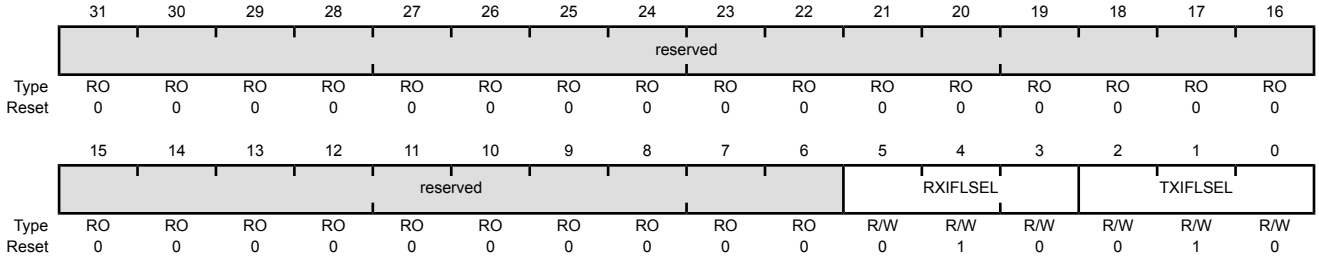
The **UARTIFLS** register is the interrupt FIFO level select register. You can use this register to define the FIFO level at which the TXRIS and RXRIS bits in the **UARTRIS** register are triggered.

The interrupts are generated based on a transition through a level rather than being based on the level. That is, the interrupts are generated when the fill level progresses through the trigger level. For example, if the receive trigger level is set to the half-way mark, the interrupt is triggered as the module is receiving the 9th character.

Out of reset, the TXIFLSEL and RXIFLSEL bits are configured so that the FIFOs trigger an interrupt at the half-way mark.

UART Interrupt FIFO Level Select (UARTIFLS)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x034
Type R/W, reset 0x0000.0012




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5:3 | RXIFLSEL | R/W | 0x2 | UART Receive Interrupt FIFO Level Select<br>The trigger points for the receive interrupt are as follows:<br>Value Description<br>0x0 RX FIFO ≥ ⅛ full<br>0x1 RX FIFO ≥ ¼ full<br>0x2 RX FIFO ≥ ½ full (default)<br>0x3 RX FIFO ≥ ¾ full<br>0x4 RX FIFO ≥ ⅞ full<br>0x5-0x7 Reserved |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 2:0 | TXIFLSEL | R/W | 0x2 | UART Transmit Interrupt FIFO Level Select<br>The trigger points for the transmit interrupt are as follows: |

| Value | Description |
|---|---|
| 0x0 | TX FIFO ≤ ⅞ empty |
| 0x1 | TX FIFO ≤ ¾ empty |
| 0x2 | TX FIFO ≤ ½ empty (default) |
| 0x3 | TX FIFO ≤ ¼ empty |
| 0x4 | TX FIFO ≤ ⅛ empty |
| 0x5-0x7 | Reserved |

## Register 9: UART Interrupt Mask (UARTIM), offset 0x038

The **UARTIM** register is the interrupt mask set/clear register.

On a read, this register gives the current value of the mask on the relevant interrupt. Writing a 1 to a bit allows the corresponding raw interrupt signal to be routed to the interrupt controller. Writing a 0 prevents the raw interrupt signal from being sent to the interrupt controller.

UART Interrupt Mask (UARTIM)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x038
Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | OEIM | BEIM | PEIM | FEIM | RTIM | TXIM | RXIM | reserved | | | |
| Type | RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W | R/W | R/W | R/W | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | OEIM | R/W | 0 | UART Overrun Error Interrupt Mask<br>On a read, the current mask for the `OEIM` interrupt is returned.<br>Setting this bit to 1 promotes the `OEIM` interrupt to the interrupt controller. |
| 9 | BEIM | R/W | 0 | UART Break Error Interrupt Mask<br>On a read, the current mask for the `BEIM` interrupt is returned.<br>Setting this bit to 1 promotes the `BEIM` interrupt to the interrupt controller. |
| 8 | PEIM | R/W | 0 | UART Parity Error Interrupt Mask<br>On a read, the current mask for the `PEIM` interrupt is returned.<br>Setting this bit to 1 promotes the `PEIM` interrupt to the interrupt controller. |
| 7 | FEIM | R/W | 0 | UART Framing Error Interrupt Mask<br>On a read, the current mask for the `FEIM` interrupt is returned.<br>Setting this bit to 1 promotes the `FEIM` interrupt to the interrupt controller. |
| 6 | RTIM | R/W | 0 | UART Receive Time-Out Interrupt Mask<br>On a read, the current mask for the `RTIM` interrupt is returned.<br>Setting this bit to 1 promotes the `RTIM` interrupt to the interrupt controller. |
| 5 | TXIM | R/W | 0 | UART Transmit Interrupt Mask<br>On a read, the current mask for the `TXIM` interrupt is returned.<br>Setting this bit to 1 promotes the `TXIM` interrupt to the interrupt controller. |
| 4 | RXIM | R/W | 0 | UART Receive Interrupt Mask<br>On a read, the current mask for the `RXIM` interrupt is returned.<br>Setting this bit to 1 promotes the `RXIM` interrupt to the interrupt controller. |

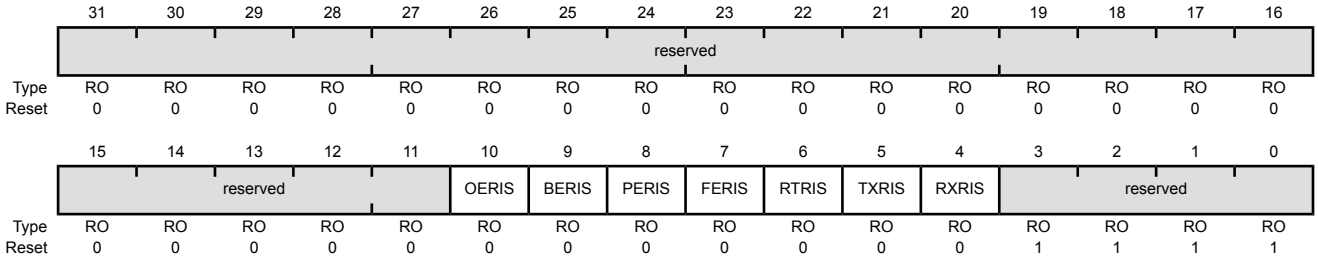
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3:0 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 10: UART Raw Interrupt Status (UARTRIS), offset 0x03C

The **UARTRIS** register is the raw interrupt status register. On a read, this register gives the current raw status value of the corresponding interrupt. A write has no effect.

UART Raw Interrupt Status (UARTRIS)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x03C
Type RO, reset 0x0000.000F



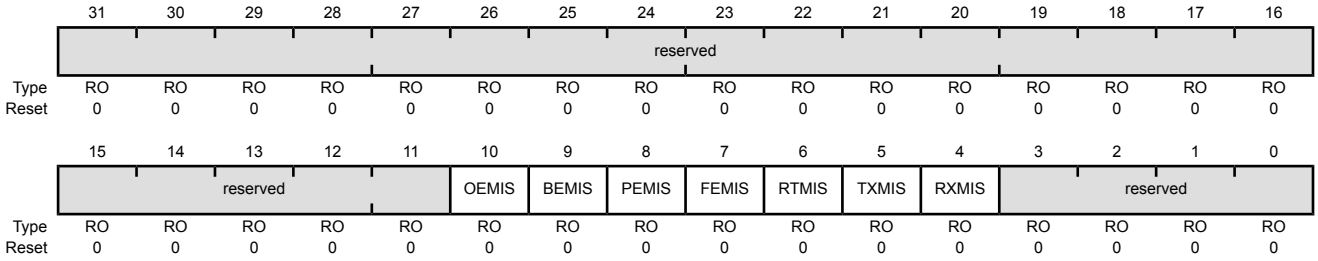

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | OERIS | RO | 0 | UART Overrun Error Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 9 | BERIS | RO | 0 | UART Break Error Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 8 | PERIS | RO | 0 | UART Parity Error Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 7 | FERIS | RO | 0 | UART Framing Error Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 6 | RTRIS | RO | 0 | UART Receive Time-Out Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 5 | TXRIS | RO | 0 | UART Transmit Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 4 | RXRIS | RO | 0 | UART Receive Raw Interrupt Status<br>Gives the raw interrupt state (prior to masking) of this interrupt. |
| 3:0 | reserved | RO | 0xF | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 11: UART Masked Interrupt Status (UARTMIS), offset 0x040

The **UARTMIS** register is the masked interrupt status register. On a read, this register gives the current masked status value of the corresponding interrupt. A write has no effect.

UART Masked Interrupt Status (UARTMIS)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x040
Type RO, reset 0x0000.0000



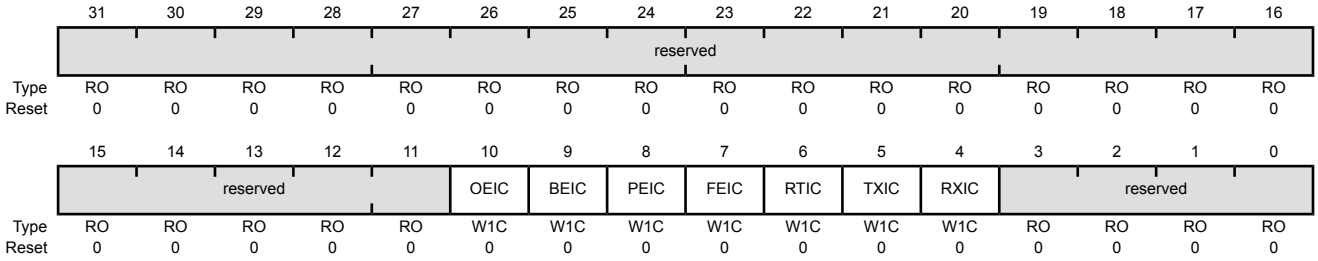

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | OEMIS | RO | 0 | UART Overrun Error Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 9 | BEMIS | RO | 0 | UART Break Error Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 8 | PEMIS | RO | 0 | UART Parity Error Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 7 | FEMIS | RO | 0 | UART Framing Error Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 6 | RTMIS | RO | 0 | UART Receive Time-Out Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 5 | TXMIS | RO | 0 | UART Transmit Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 4 | RXMIS | RO | 0 | UART Receive Masked Interrupt Status<br>Gives the masked interrupt state of this interrupt. |
| 3:0 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 12: UART Interrupt Clear (UARTICR), offset 0x044

The **UARTICR** register is the interrupt clear register. On a write of 1, the corresponding interrupt (both raw interrupt and masked interrupt, if enabled) is cleared. A write of 0 has no effect.

UART Interrupt Clear (UARTICR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0x044
Type W1C, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:11 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 10 | OEIC | W1C | 0 | Overrun Error Interrupt Clear<br>The OEIC values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 9 | BEIC | W1C | 0 | Break Error Interrupt Clear<br>The BEIC values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 8 | PEIC | W1C | 0 | Parity Error Interrupt Clear<br>The PEIC values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 7 | FEIC | W1C | 0 | Framing Error Interrupt Clear<br>The FEIC values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |

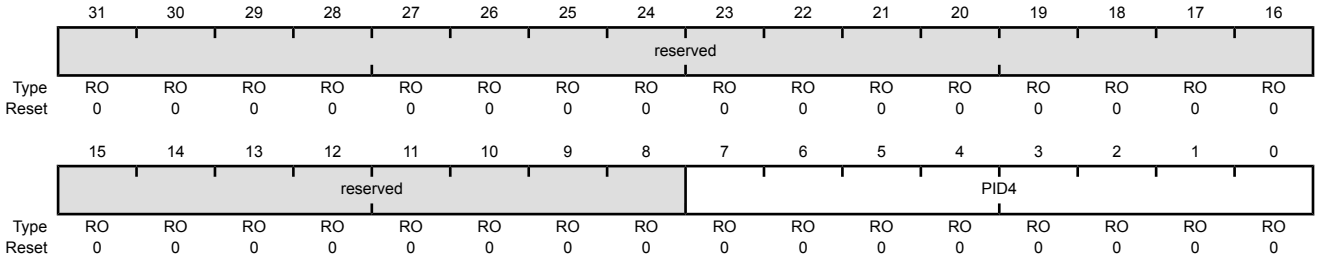
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 6 | RTIC | W1C | 0 | Receive Time-Out Interrupt Clear<br>The RTIC values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 5 | TXIC | W1C | 0 | Transmit Interrupt Clear<br>The TXIC values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 4 | RXIC | W1C | 0 | Receive Interrupt Clear<br>The RXIC values are defined as follows:<br><br>Value Description<br>0 No effect on the interrupt.<br>1 Clears interrupt. |
| 3:0 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |

## Register 13: UART Peripheral Identification 4 (UARTPeriphID4), offset 0xFD0

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 4 (UARTPeriphID4)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFD0
Type RO, reset 0x0000.0000



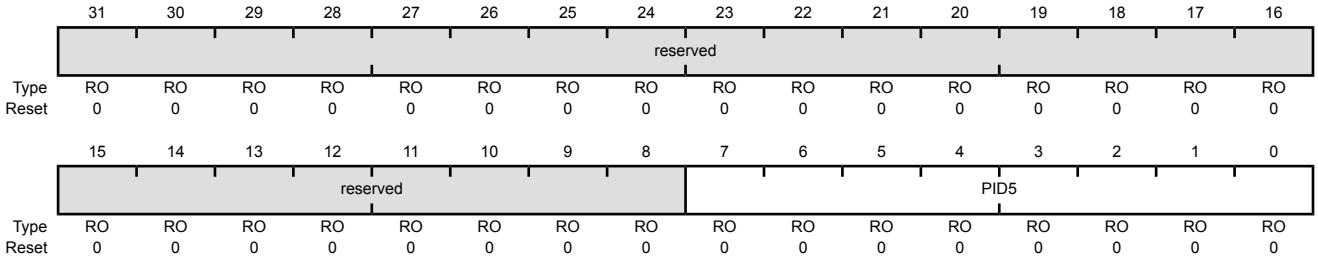

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID4 | RO | 0x0000 | UART Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

## Register 14: UART Peripheral Identification 5 (UARTPeriphID5), offset 0xFD4

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 5 (UARTPeriphID5)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFD4
Type RO, reset 0x0000.0000



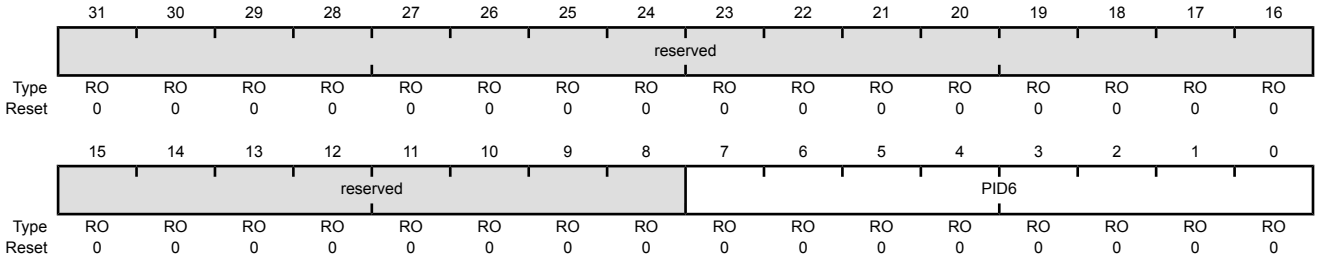

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID5 | RO | 0x0000 | UART Peripheral ID Register[15:8]<br>Can be used by software to identify the presence of this peripheral. |

## Register 15: UART Peripheral Identification 6 (UARTPeriphID6), offset 0xFD8

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 6 (UARTPeriphID6)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFD8
Type RO, reset 0x0000.0000



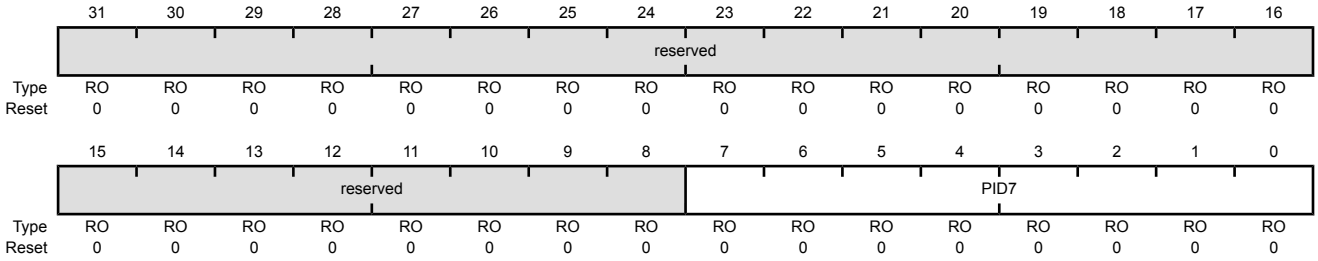

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID6 | RO | 0x0000 | UART Peripheral ID Register[23:16]<br>Can be used by software to identify the presence of this peripheral. |

## Register 16: UART Peripheral Identification 7 (UARTPeriphID7), offset 0xFDC

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 7 (UARTPeriphID7)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFDC
Type RO, reset 0x0000.0000



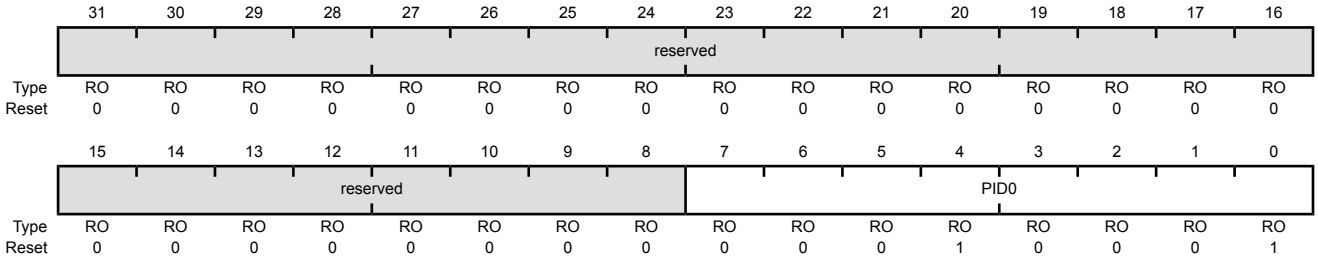

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID7 | RO | 0x0000 | UART Peripheral ID Register[31:24]<br>Can be used by software to identify the presence of this peripheral. |

## Register 17: UART Peripheral Identification 0 (UARTPeriphID0), offset 0xFE0

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 0 (UARTPeriphID0)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFE0
Type RO, reset 0x0000.0011



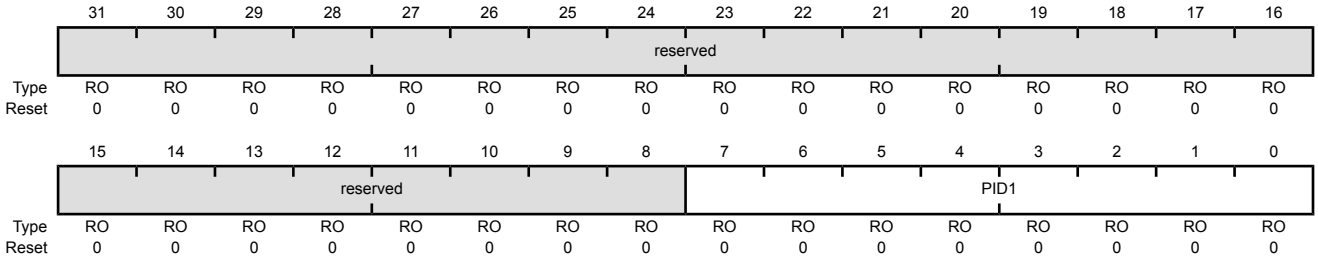

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID0 | RO | 0x11 | UART Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

## Register 18: UART Peripheral Identification 1 (UARTPeriphID1), offset 0xFE4

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 1 (UARTPeriphID1)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFE4
Type RO, reset 0x0000.0000



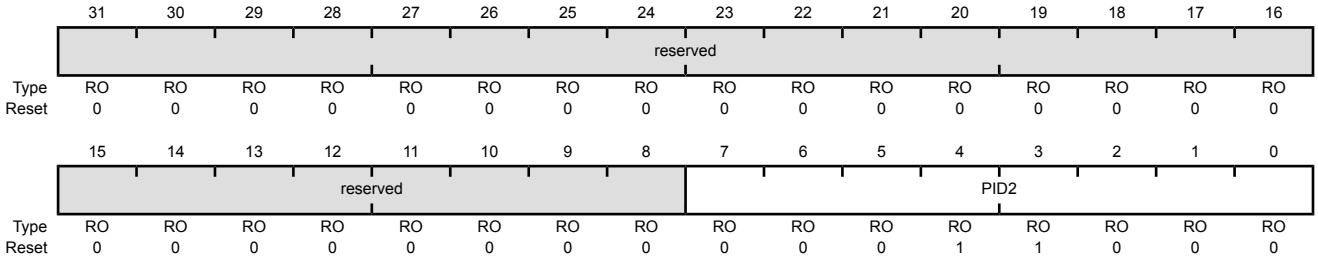

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID1 | RO | 0x00 | UART Peripheral ID Register[15:8]<br>Can be used by software to identify the presence of this peripheral. |

## Register 19: UART Peripheral Identification 2 (UARTPeriphID2), offset 0xFE8

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 2 (UARTPeriphID2)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFE8
Type RO, reset 0x0000.0018



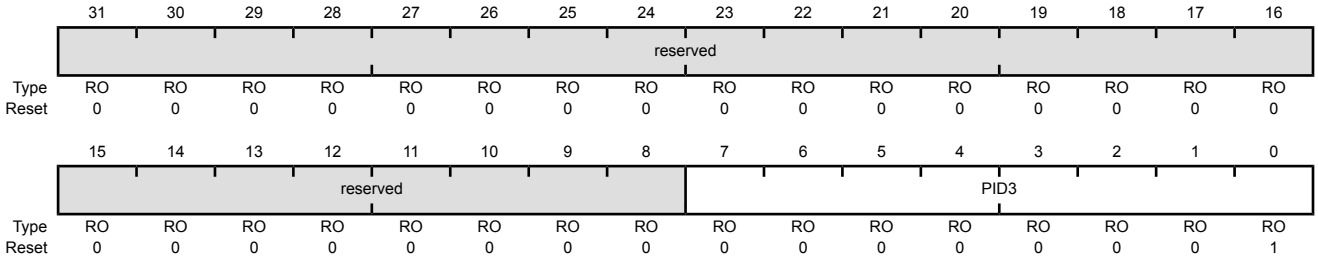

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID2 | RO | 0x18 | UART Peripheral ID Register[23:16]<br>Can be used by software to identify the presence of this peripheral. |

## Register 20: UART Peripheral Identification 3 (UARTPeriphID3), offset 0xFEC

The **UARTPeriphIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART Peripheral Identification 3 (UARTPeriphID3)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFEC
Type RO, reset 0x0000.0001



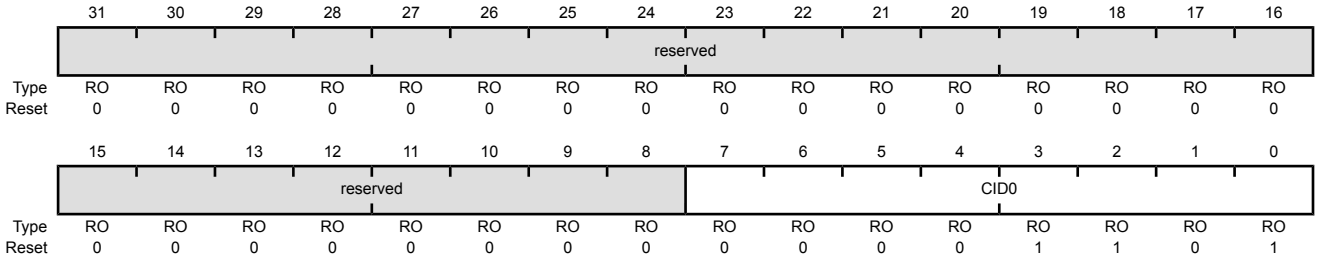

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID3 | RO | 0x01 | UART Peripheral ID Register[31:24]<br>Can be used by software to identify the presence of this peripheral. |

## Register 21: UART PrimeCell Identification 0 (UARTPCellID0), offset 0xFF0

The **UARTPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART PrimeCell Identification 0 (UARTPCellID0)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFF0
Type RO, reset 0x0000.000D



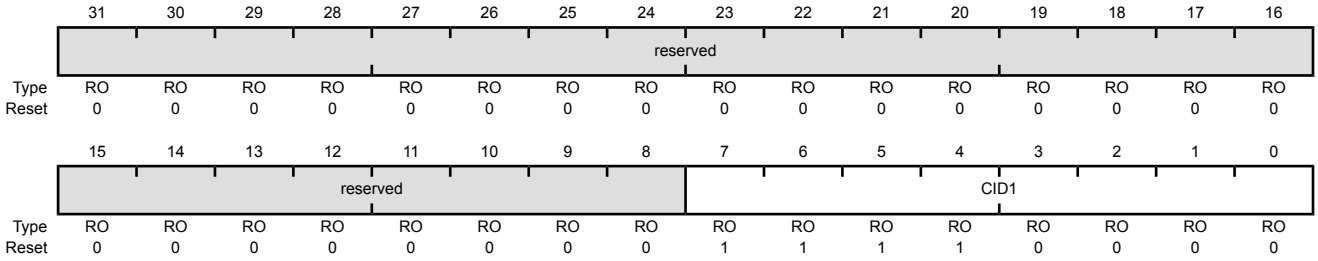

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID0 | RO | 0x0D | UART PrimeCell ID Register[7:0]<br>Provides software a standard cross-peripheral identification system. |

## Register 22: UART PrimeCell Identification 1 (UARTPCellID1), offset 0xFF4

The **UARTPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART PrimeCell Identification 1 (UARTPCellID1)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFF4
Type RO, reset 0x0000.00F0



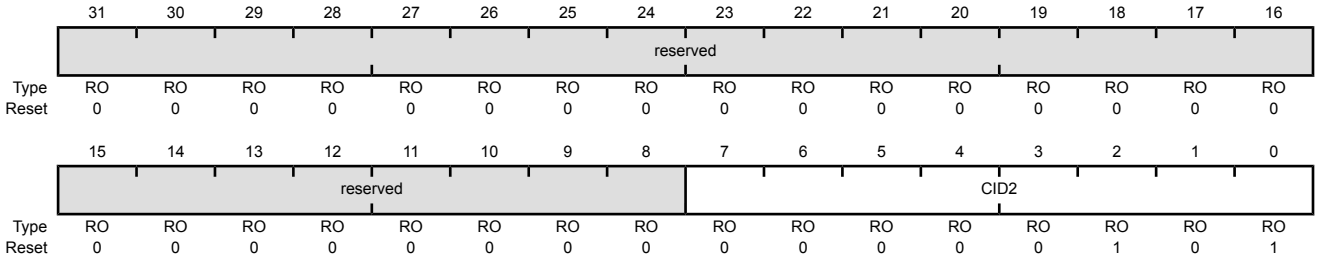

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID1 | RO | 0xF0 | UART PrimeCell ID Register[15:8]<br>Provides software a standard cross-peripheral identification system. |

## Register 23: UART PrimeCell Identification 2 (UARTPCellID2), offset 0xFF8

The **UARTPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART PrimeCell Identification 2 (UARTPCellID2)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFF8
Type RO, reset 0x0000.0005



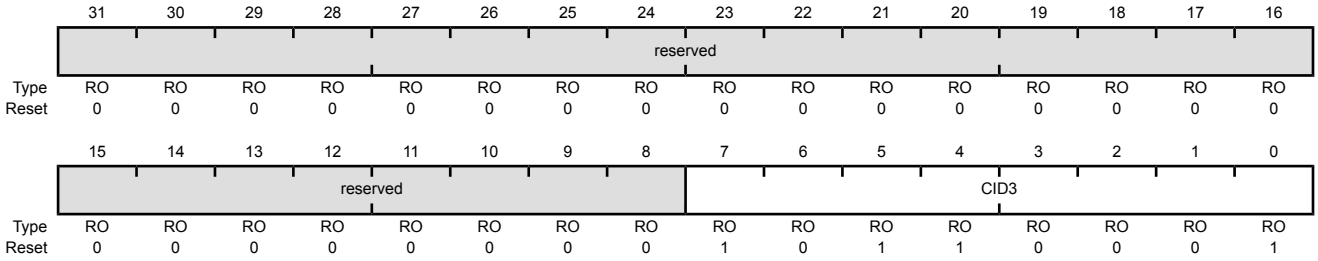

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID2 | RO | 0x05 | UART PrimeCell ID Register[23:16]<br>Provides software a standard cross-peripheral identification system. |

## Register 24: UART PrimeCell Identification 3 (UARTPCellID3), offset 0xFFC

The **UARTPCellIDn** registers are hard-coded and the fields within the registers determine the reset values.

UART PrimeCell Identification 3 (UARTPCellID3)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
Offset 0xFFC
Type RO, reset 0x0000.00B1




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID3 | RO | 0xB1 | UART PrimeCell ID Register[31:24]<br>Provides software a standard cross-peripheral identification system. |

# 12 Synchronous Serial Interface (SSI)

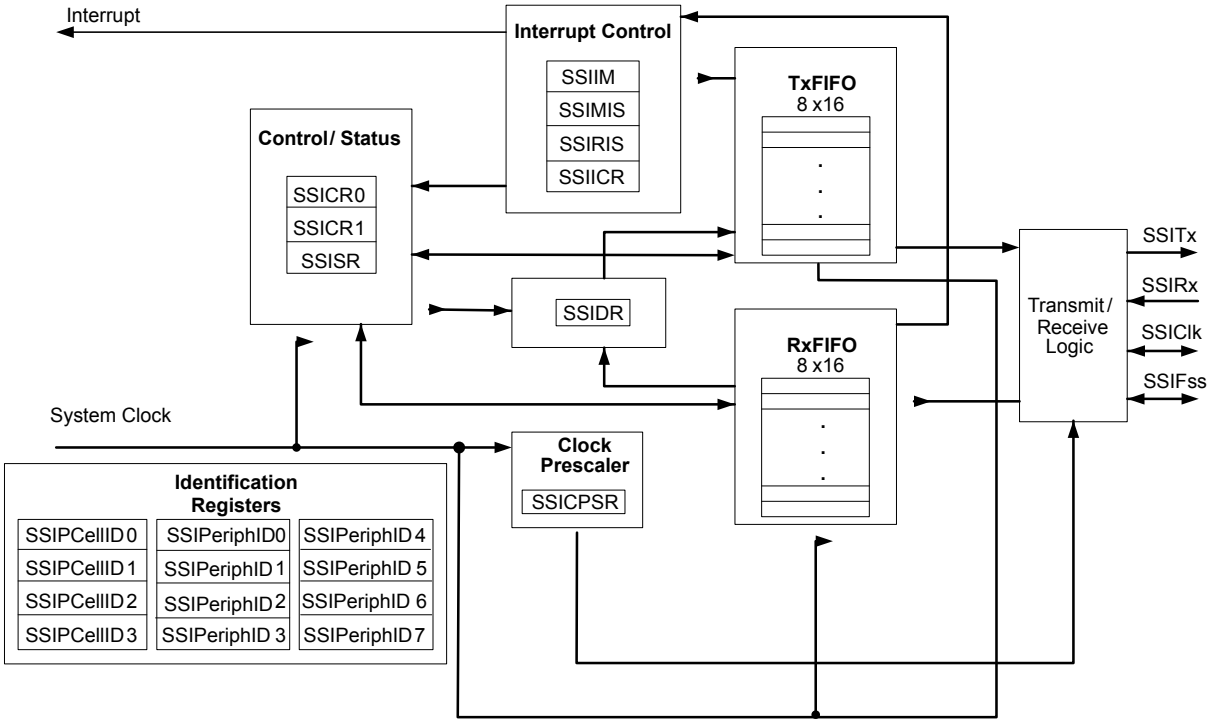
The Stellaris® Synchronous Serial Interface (SSI) is a master or slave interface for synchronous serial communication with peripheral devices that have either Freescale SPI, MICROWIRE, or Texas Instruments synchronous serial interfaces.

The Stellaris SSI module has the following features:

- Master or slave operation
- Programmable clock bit rate and prescale
- Separate transmit and receive FIFOs, 16 bits wide, 8 locations deep
- Programmable interface operation for Freescale SPI, MICROWIRE, or Texas Instruments synchronous serial interfaces
- Programmable data frame size from 4 to 16 bits
- Internal loopback test mode for diagnostic/debug testing

## 12.1 Block Diagram

**Figure 12-1. SSI Module Block Diagram**




## 12.2 Functional Description

The SSI performs serial-to-parallel conversion on data received from a peripheral device. The CPU accesses data, control, and status information. The transmit and receive paths are buffered with

internal FIFO memories allowing up to eight 16-bit values to be stored independently in both transmit and receive modes.

### 12.2.1 Bit Rate Generation

The SSI includes a programmable bit rate clock divider and prescaler to generate the serial output clock. Bit rates are supported to 1.5 MHz and higher, although maximum bit rate is determined by peripheral devices.

The serial bit rate is derived by dividing down the input clock (FSysClk). The clock is first divided by an even prescale value CPSDVSR from 2 to 254, which is programmed in the **SSI Clock Prescale (SSICPSR)** register (see page 412). The clock is further divided by a value from 1 to 256, which is 1 + *SCR*, where *SCR* is the value programmed in the **SSI Control0 (SSICR0)** register (see page 405).

The frequency of the output clock SSIClk is defined by:

```
SSIClk = FSysClk / (CPSDVSR * (1 + SCR))
```

**Note:** For master mode, the system clock must be at least two times faster than the SSIClk. For slave mode, the system clock must be at least 12 times faster than the SSIClk.

See “Synchronous Serial Interface (SSI)” on page 485 to view SSI timing parameters.

### 12.2.2 FIFO Operation

#### 12.2.2.1 Transmit FIFO

The common transmit FIFO is a 16-bit wide, 8-locations deep, first-in, first-out memory buffer. The CPU writes data to the FIFO by writing the **SSI Data (SSIDR)** register (see page 409), and data is stored in the FIFO until it is read out by the transmission logic.

When configured as a master or a slave, parallel data is written into the transmit FIFO prior to serial conversion and transmission to the attached slave or master, respectively, through the SSITx pin.

In slave mode, the SSI transmits data each time the master initiates a transaction. If the transmit FIFO is empty and the master initiates, the slave transmits the 8th most recent value in the transmit FIFO. If less than 8 values have been written to the transmit FIFO since the SSI module clock was enabled using the SSI bit in the **RGCG1** register, then 0 is transmitted. Care should be taken to ensure that valid data is in the FIFO as needed. The SSI can be configured to generate an interrupt or a µDMA request when the FIFO is empty.

#### 12.2.2.2 Receive FIFO

The common receive FIFO is a 16-bit wide, 8-locations deep, first-in, first-out memory buffer. Received data from the serial interface is stored in the buffer until read out by the CPU, which accesses the read FIFO by reading the **SSIDR** register.

When configured as a master or slave, serial data received through the SSIRx pin is registered prior to parallel loading into the attached slave or master receive FIFO, respectively.

### 12.2.3 Interrupts

The SSI can generate interrupts when the following conditions are observed:

- Transmit FIFO service
- Receive FIFO service

- Receive FIFO time-out
- Receive FIFO overrun

All of the interrupt events are ORed together before being sent to the interrupt controller, so the SSI can only generate a single interrupt request to the controller at any given time. You can mask each of the four individual maskable interrupts by setting the appropriate bits in the **SSI Interrupt Mask (SSIIM)** register (see page 413). Setting the appropriate mask bit to 1 enables the interrupt.

Provision of the individual outputs, as well as a combined interrupt output, allows use of either a global interrupt service routine, or modular device drivers to handle interrupts. The transmit and receive dynamic dataflow interrupts have been separated from the status interrupts so that data can be read or written in response to the FIFO trigger levels. The status of the individual interrupt sources can be read from the **SSI Raw Interrupt Status (SSIRIS)** and **SSI Masked Interrupt Status (SSIMIS)** registers (see page 415 and page 416, respectively).

### 12.2.4 Frame Formats

Each data frame is between 4 and 16 bits long, depending on the size of data programmed, and is transmitted starting with the MSB. There are three basic frame types that can be selected:

- Texas Instruments synchronous serial
- Freescale SPI
- MICROWIRE

For all three formats, the serial clock (SSIClk) is held inactive while the SSI is idle, and SSIClk transitions at the programmed frequency only during active transmission or reception of data. The idle state of SSIClk is utilized to provide a receive timeout indication that occurs when the receive FIFO still contains data after a timeout period.

For Freescale SPI and MICROWIRE frame formats, the serial frame (SSIFss ) pin is active Low, and is asserted (pulled down) during the entire transmission of the frame.

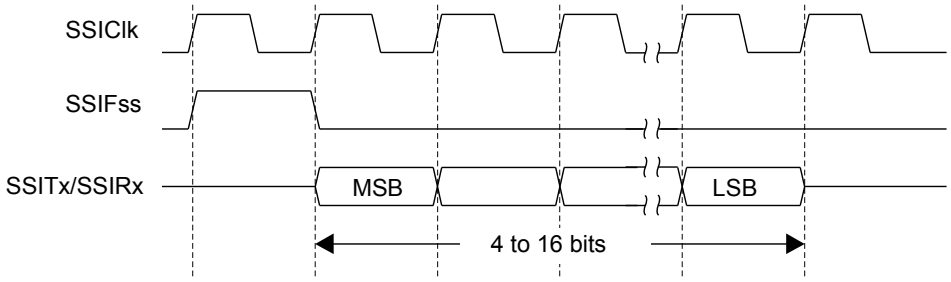
For Texas Instruments synchronous serial frame format, the SSIFss pin is pulsed for one serial clock period starting at its rising edge, prior to the transmission of each frame. For this frame format, both the SSI and the off-chip slave device drive their output data on the rising edge of SSIClk, and latch data from the other device on the falling edge.

Unlike the full-duplex transmission of the other two frame formats, the MICROWIRE format uses a special master-slave messaging technique, which operates at half-duplex. In this mode, when a frame begins, an 8-bit control message is transmitted to the off-chip slave. During this transmit, no incoming data is received by the SSI. After the message has been sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message has been sent, responds with the requested data. The returned data can be 4 to 16 bits in length, making the total frame length anywhere from 13 to 25 bits.

#### 12.2.4.1 Texas Instruments Synchronous Serial Frame Format

Figure 12-2 on page 396 shows the Texas Instruments synchronous serial frame format for a single transmitted frame.

**Figure 12-2. TI Synchronous Serial Frame Format (Single Transfer)**

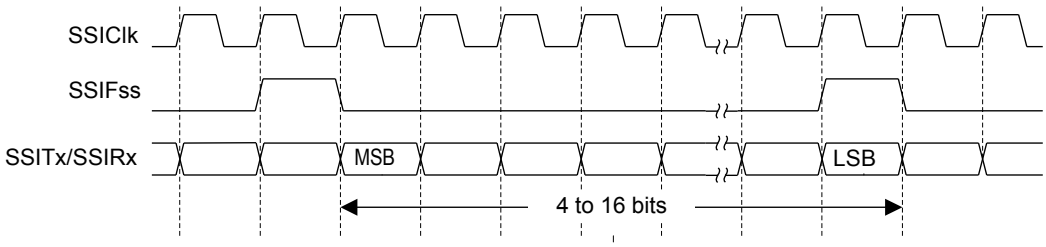



In this mode, SSIClk and SSIFss are forced Low, and the transmit data line SSITx is tristated whenever the SSI is idle. Once the bottom entry of the transmit FIFO contains data, SSIFss is pulsed High for one SSIClk period. The value to be transmitted is also transferred from the transmit FIFO to the serial shift register of the transmit logic. On the next rising edge of SSIClk, the MSB of the 4 to 16-bit data frame is shifted out on the SSITx pin. Likewise, the MSB of the received data is shifted onto the SSIRx pin by the off-chip serial slave device.

Both the SSI and the off-chip serial slave device then clock each data bit into their serial shifter on the falling edge of each SSIClk. The received data is transferred from the serial shifter to the receive FIFO on the first rising edge of SSIClk after the LSB has been latched.

Figure 12-3 on page 396 shows the Texas Instruments synchronous serial frame format when back-to-back frames are transmitted.

**Figure 12-3. TI Synchronous Serial Frame Format (Continuous Transfer)**




### 12.2.4.2 Freescale SPI Frame Format

The Freescale SPI interface is a four-wire interface where the SSIFss signal behaves as a slave select. The main feature of the Freescale SPI format is that the inactive state and phase of the SSIClk signal are programmable through the SPO and SPH bits within the **SSISCR0** control register.

#### *SPO Clock Polarity Bit*

When the SPO clock polarity control bit is Low, it produces a steady state Low value on the SSIClk pin. If the SPO bit is High, a steady state High value is placed on the SSIClk pin when data is not being transferred.

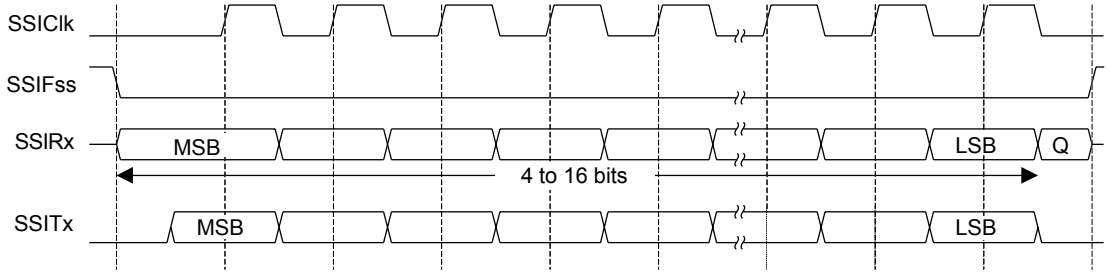
### *SPH Phase Control Bit*

The SPH phase control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge. When the SPH phase control bit is Low, data is captured on the first clock edge transition. If the SPH bit is High, data is captured on the second clock edge transition.

## 12.2.4.3 Freescale SPI Frame Format with SPO=0 and SPH=0

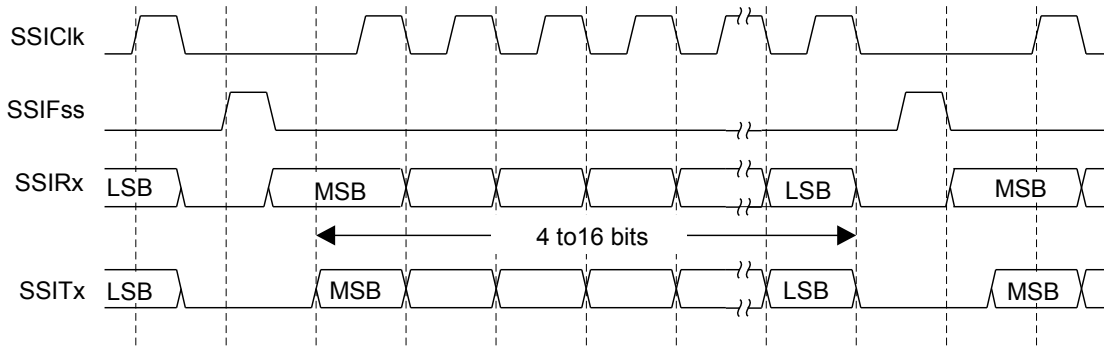
Single and continuous transmission signal sequences for Freescale SPI format with SPO=0 and SPH=0 are shown in Figure 12-4 on page 397 and Figure 12-5 on page 397.

**Figure 12-4. Freescale SPI Format (Single Transfer) with SPO=0 and SPH=0**




**Note:** Q is undefined.

**Figure 12-5. Freescale SPI Format (Continuous Transfer) with SPO=0 and SPH=0**




In this configuration, during idle periods:

- SSIClk is forced Low
- SSIFss is forced High
- The transmit data line SSITx is arbitrarily forced Low
- When the SSI is configured as a master, it enables the SSIClk pad
- When the SSI is configured as a slave, it disables the SSIClk pad

If the SSI is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSIFss master signal being driven Low. This causes slave data to be enabled onto the SSIRx input line of the master. The master SSITx output pad is enabled.

One half SSIClk period later, valid master data is transferred to the SSITx pin. Now that both the master and slave data have been set, the SSIClk master clock pin goes High after one further half SSIClk period.

The data is now captured on the rising and propagated on the falling edges of the SSIClk signal.

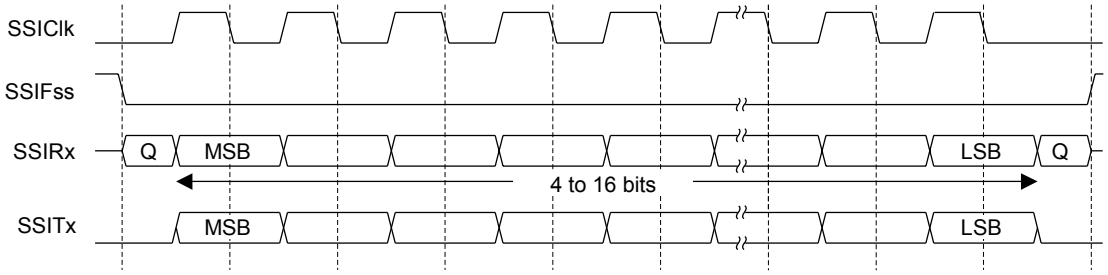
In the case of a single word transmission, after all bits of the data word have been transferred, the SSIFss line is returned to its idle High state one SSIClk period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSIFss signal must be pulsed High between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the SPH bit is logic zero. Therefore, the master device must raise the SSIFss pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSIFss pin is returned to its idle state one SSIClk period after the last bit has been captured.

### 12.2.4.4 Freescale SPI Frame Format with SPO=0 and SPH=1

The transfer signal sequence for Freescale SPI format with SPO=0 and SPH=1 is shown in Figure 12-6 on page 398, which covers both single and continuous transfers.

**Figure 12-6. Freescale SPI Frame Format with SPO=0 and SPH=1**




**Note:** Q is undefined.

In this configuration, during idle periods:

- SSIClk is forced Low
- SSIFss is forced High
- The transmit data line SSITx is arbitrarily forced Low
- When the SSI is configured as a master, it enables the SSIClk pad
- When the SSI is configured as a slave, it disables the SSIClk pad

If the SSI is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSIFss master signal being driven Low. The master SSITx output is enabled. After a further one half SSIClk period, both master and slave valid data is enabled onto their respective transmission lines. At the same time, the SSIClk is enabled with a rising edge transition.

Data is then captured on the falling edges and propagated on the rising edges of the SSIClk signal.

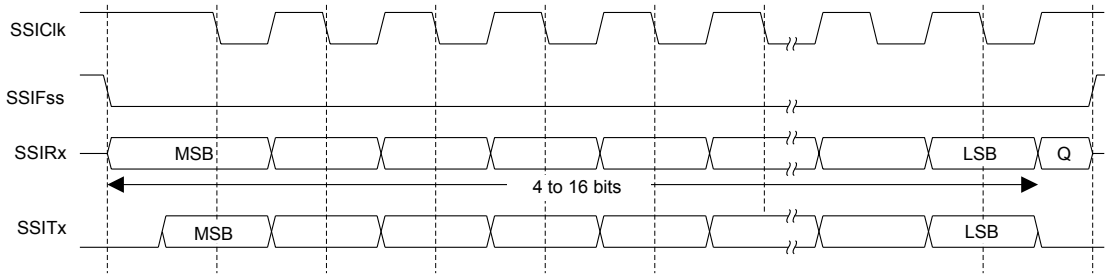
In the case of a single word transfer, after all bits have been transferred, the SSIFss line is returned to its idle High state one SSIClk period after the last bit has been captured.

For continuous back-to-back transfers, the SSIFss pin is held Low between successive data words and termination is the same as that of the single word transfer.

### 12.2.4.5 Freescale SPI Frame Format with SPO=1 and SPH=0

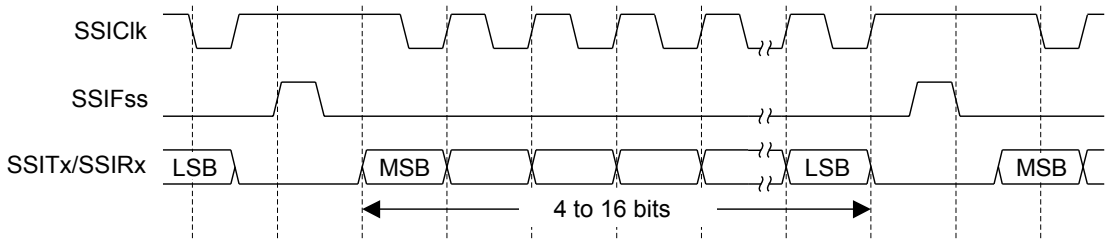
Single and continuous transmission signal sequences for Freescale SPI format with SPO=1 and SPH=0 are shown in Figure 12-7 on page 399 and Figure 12-8 on page 399.

**Figure 12-7. Freescale SPI Frame Format (Single Transfer) with SPO=1 and SPH=0**




**Note:** Q is undefined.

**Figure 12-8. Freescale SPI Frame Format (Continuous Transfer) with SPO=1 and SPH=0**




In this configuration, during idle periods:

- SSIClk is forced High
- SSIFss is forced High
- The transmit data line SSITx is arbitrarily forced Low
- When the SSI is configured as a master, it enables the SSIClk pad
- When the SSI is configured as a slave, it disables the SSIClk pad

If the SSI is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSIFss master signal being driven Low, which causes slave data to be immediately transferred onto the SSIRx line of the master. The master SSITx output pad is enabled.

One half period later, valid master data is transferred to the SSITx line. Now that both the master and slave data have been set, the SSIClk master clock pin becomes Low after one further half SSIClk period. This means that data is captured on the falling edges and propagated on the rising edges of the SSIClk signal.

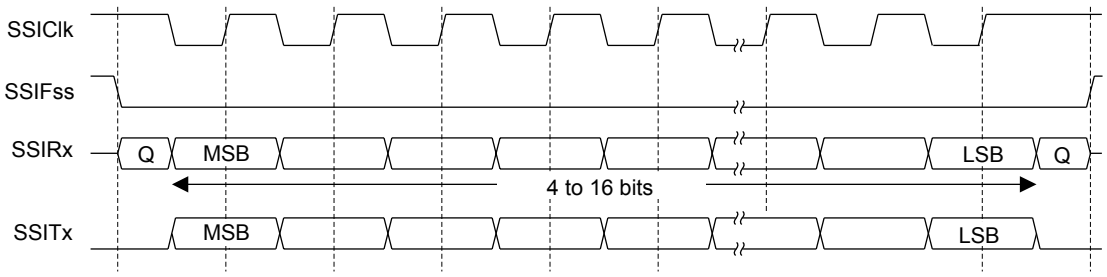
In the case of a single word transmission, after all bits of the data word are transferred, the SSIFss line is returned to its idle High state one SSIClk period after the last bit has been captured.

However, in the case of continuous back-to-back transmissions, the SSIFss signal must be pulsed High between each data word transfer. This is because the slave select pin freezes the data in its serial peripheral register and does not allow it to be altered if the SPH bit is logic zero. Therefore, the master device must raise the SSIFss pin of the slave device between each data transfer to enable the serial peripheral data write. On completion of the continuous transfer, the SSIFss pin is returned to its idle state one SSIClk period after the last bit has been captured.

### 12.2.4.6 Freescale SPI Frame Format with SPO=1 and SPH=1

The transfer signal sequence for Freescale SPI format with SPO=1 and SPH=1 is shown in Figure 12-9 on page 400, which covers both single and continuous transfers.

**Figure 12-9. Freescale SPI Frame Format with SPO=1 and SPH=1**




**Note:** Q is undefined.

In this configuration, during idle periods:

- SSIClk is forced High
- SSIFss is forced High
- The transmit data line SSITx is arbitrarily forced Low
- When the SSI is configured as a master, it enables the SSIClk pad
- When the SSI is configured as a slave, it disables the SSIClk pad

If the SSI is enabled and there is valid data within the transmit FIFO, the start of transmission is signified by the SSIFss master signal being driven Low. The master SSITx output pad is enabled. After a further one-half SSIClk period, both master and slave data are enabled onto their respective transmission lines. At the same time, SSIClk is enabled with a falling edge transition. Data is then captured on the rising edges and propagated on the falling edges of the SSIClk signal.

After all bits have been transferred, in the case of a single word transmission, the SSIFss line is returned to its idle high state one SSIClk period after the last bit has been captured.

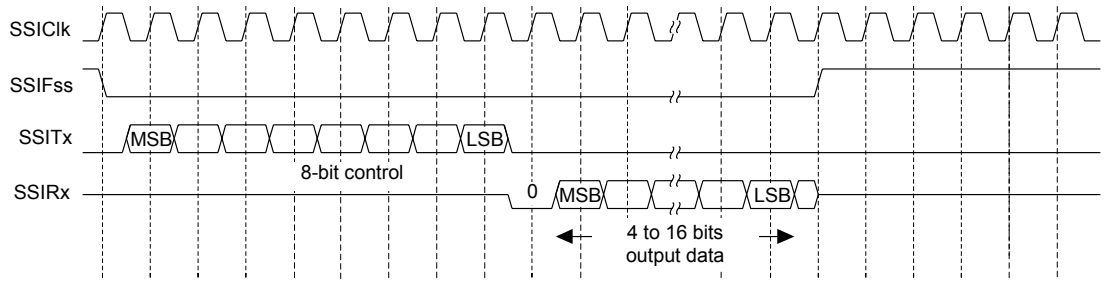
For continuous back-to-back transmissions, the SSIFss pin remains in its active Low state, until the final bit of the last word has been captured, and then returns to its idle state as described above.

For continuous back-to-back transfers, the SSIFss pin is held Low between successive data words and termination is the same as that of the single word transfer.

### 12.2.4.7 MICROWIRE Frame Format

Figure 12-10 on page 401 shows the MICROWIRE frame format, again for a single frame. Figure 12-11 on page 402 shows the same format when back-to-back frames are transmitted.

**Figure 12-10. MICROWIRE Frame Format (Single Frame)**




MICROWIRE format is very similar to SPI format, except that transmission is half-duplex instead of full-duplex, using a master-slave message passing technique. Each serial transmission begins with an 8-bit control word that is transmitted from the SSI to the off-chip slave device. During this transmission, no incoming data is received by the SSI. After the message has been sent, the off-chip slave decodes it and, after waiting one serial clock after the last bit of the 8-bit control message has been sent, responds with the required data. The returned data is 4 to 16 bits in length, making the total frame length anywhere from 13 to 25 bits.

In this configuration, during idle periods:

- SSIClk is forced Low
- SSIFss is forced High
- The transmit data line SSITx is arbitrarily forced Low

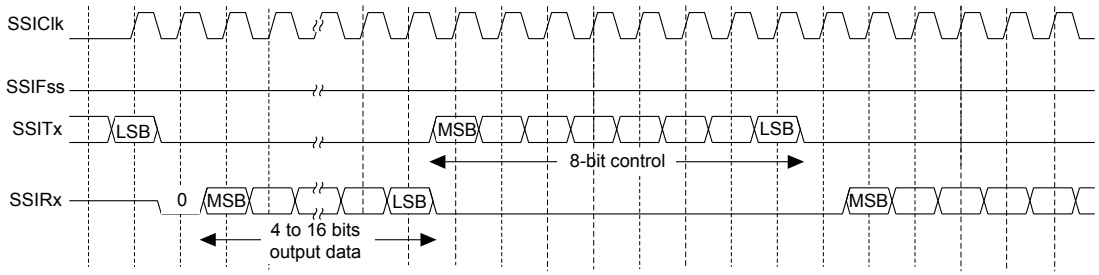
A transmission is triggered by writing a control byte to the transmit FIFO. The falling edge of SSIFss causes the value contained in the bottom entry of the transmit FIFO to be transferred to the serial shift register of the transmit logic, and the MSB of the 8-bit control frame to be shifted out onto the SSITx pin. SSIFss remains Low for the duration of the frame transmission. The SSIRx pin remains tristated during this transmission.

The off-chip serial slave device latches each control bit into its serial shifter on the rising edge of each SSIClk. After the last bit is latched by the slave device, the control byte is decoded during a one clock wait-state, and the slave responds by transmitting data back to the SSI. Each bit is driven onto the SSIRx line on the falling edge of SSIClk. The SSI in turn latches each bit on the rising edge of SSIClk. At the end of the frame, for single transfers, the SSIFss signal is pulled High one clock period after the last bit has been latched in the receive serial shifter, which causes the data to be transferred to the receive FIFO.

**Note:** The off-chip slave device can tristate the receive line either on the falling edge of SSIClk after the LSB has been latched by the receive shifter, or when the SSIFss pin goes High.

For continuous transfers, data transmission begins and ends in the same manner as a single transfer. However, the SSIFss line is continuously asserted (held Low) and transmission of data occurs back-to-back. The control byte of the next frame follows directly after the LSB of the received data from the current frame. Each of the received values is transferred from the receive shifter on the falling edge of SSIClk, after the LSB of the frame has been latched into the SSI.

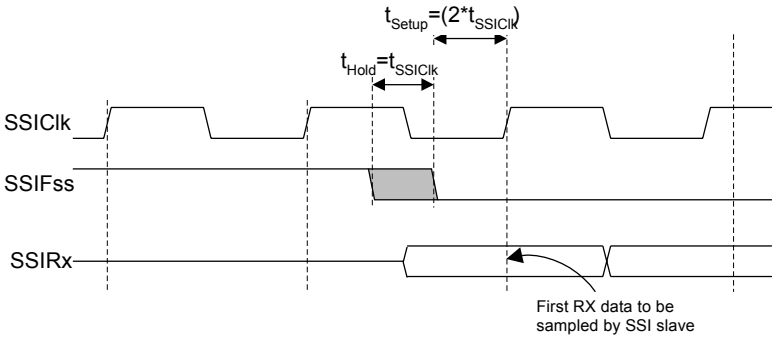
**Figure 12-11. MICROWIRE Frame Format (Continuous Transfer)**




In the MICROWIRE mode, the SSI slave samples the first bit of receive data on the rising edge of SSIClk after SSIFss has gone Low. Masters that drive a free-running SSIClk must ensure that the SSIFss signal has sufficient setup and hold margins with respect to the rising edge of SSIClk.

Figure 12-12 on page 402 illustrates these setup and hold time requirements. With respect to the SSIClk rising edge on which the first bit of receive data is to be sampled by the SSI slave, SSIFss must have a setup of at least two times the period of SSIClk on which the SSI operates. With respect to the SSIClk rising edge previous to this edge, SSIFss must have a hold of at least one SSIClk period.

**Figure 12-12. MICROWIRE Frame Format, SSIFss Input Setup and Hold Requirements**




## 12.3 Initialization and Configuration

To use the SSI, its peripheral clock must be enabled by setting the SSI bit in the **RCGC1** register.

For each of the frame formats, the SSI is configured using the following steps:

1. Ensure that the SSE bit in the **SSICR1** register is disabled before making any configuration changes.
2. Select whether the SSI is a master or slave:
   a. For master operations, set the **SSICR1** register to 0x0000.0000.
   b. For slave mode (output enabled), set the **SSICR1** register to 0x0000.0004.
   c. For slave mode (output disabled), set the **SSICR1** register to 0x0000.000C.
3. Configure the clock prescale divisor by writing the **SSICPSR** register.
4. Write the **SSICR0** register with the following configuration:

- Serial clock rate (SCR)
- Desired clock phase/polarity, if using Freescale SPI mode (SPH and SPO)
- The protocol mode: Freescale SPI, TI SSF, MICROWIRE (FRF)
- The data size (DSS)

5. Enable the SSI by setting the SSE bit in the **SSICR1** register.

As an example, assume the SSI must be configured to operate with the following parameters:

- Master operation
- Freescale SPI mode (SPO=1, SPH=1)
- 1 Mbps bit rate
- 8 data bits

Assuming the system clock is 20 MHz, the bit rate calculation would be:

```
FSSIClk = FSysClk / (CPSDVSR * (1 + SCR))
1x106 = 20x106 / (CPSDVSR * (1 + SCR))
```

In this case, if CPSDVSR=2, SCR must be 9.

The configuration sequence would be as follows:

1. Ensure that the SSE bit in the **SSICR1** register is disabled.
2. Write the **SSICR1** register with a value of 0x0000.0000.
3. Write the **SSICPSR** register with a value of 0x0000.0002.
4. Write the **SSICR0** register with a value of 0x0000.09C7.
5. The SSI is then enabled by setting the SSE bit in the **SSICR1** register to 1.

## 12.4 Register Map

Table 12-1 on page 404 lists the SSI registers. The offset listed is a hexadecimal increment to the register's address, relative to that SSI module's base address:

- SSI0: 0x4000.8000

Note that the SSI module clock must be enabled before the registers can be programmed (see page 191). There must be a delay of 3 system clocks after the SSI module clock is enabled before any SSI module registers are accessed.

**Note:** The SSI must be disabled (see the SSE bit in the **SSICR1** register) before any of the control registers are reprogrammed.

**Table 12-1. SSI Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| 0x000 | SSICR0 | R/W | 0x0000.0000 | SSI Control 0 | 405 |
| 0x004 | SSICR1 | R/W | 0x0000.0000 | SSI Control 1 | 407 |
| 0x008 | SSIDR | R/W | 0x0000.0000 | SSI Data | 409 |
| 0x00C | SSISR | RO | 0x0000.0003 | SSI Status | 410 |
| 0x010 | SSICPSR | R/W | 0x0000.0000 | SSI Clock Prescale | 412 |
| 0x014 | SSIIM | R/W | 0x0000.0000 | SSI Interrupt Mask | 413 |
| 0x018 | SSIRIS | RO | 0x0000.0008 | SSI Raw Interrupt Status | 415 |
| 0x01C | SSIMIS | RO | 0x0000.0000 | SSI Masked Interrupt Status | 416 |
| 0x020 | SSIICR | W1C | 0x0000.0000 | SSI Interrupt Clear | 417 |
| 0xFD0 | SSIPeriphID4 | RO | 0x0000.0000 | SSI Peripheral Identification 4 | 418 |
| 0xFD4 | SSIPeriphID5 | RO | 0x0000.0000 | SSI Peripheral Identification 5 | 419 |
| 0xFD8 | SSIPeriphID6 | RO | 0x0000.0000 | SSI Peripheral Identification 6 | 420 |
| 0xFDC | SSIPeriphID7 | RO | 0x0000.0000 | SSI Peripheral Identification 7 | 421 |
| 0xFE0 | SSIPeriphID0 | RO | 0x0000.0022 | SSI Peripheral Identification 0 | 422 |
| 0xFE4 | SSIPeriphID1 | RO | 0x0000.0000 | SSI Peripheral Identification 1 | 423 |
| 0xFE8 | SSIPeriphID2 | RO | 0x0000.0018 | SSI Peripheral Identification 2 | 424 |
| 0xFEC | SSIPeriphID3 | RO | 0x0000.0001 | SSI Peripheral Identification 3 | 425 |
| 0xFF0 | SSIPCellID0 | RO | 0x0000.000D | SSI PrimeCell Identification 0 | 426 |
| 0xFF4 | SSIPCellID1 | RO | 0x0000.00F0 | SSI PrimeCell Identification 1 | 427 |
| 0xFF8 | SSIPCellID2 | RO | 0x0000.0005 | SSI PrimeCell Identification 2 | 428 |
| 0xFFC | SSIPCellID3 | RO | 0x0000.00B1 | SSI PrimeCell Identification 3 | 429 |

## 12.5 Register Descriptions

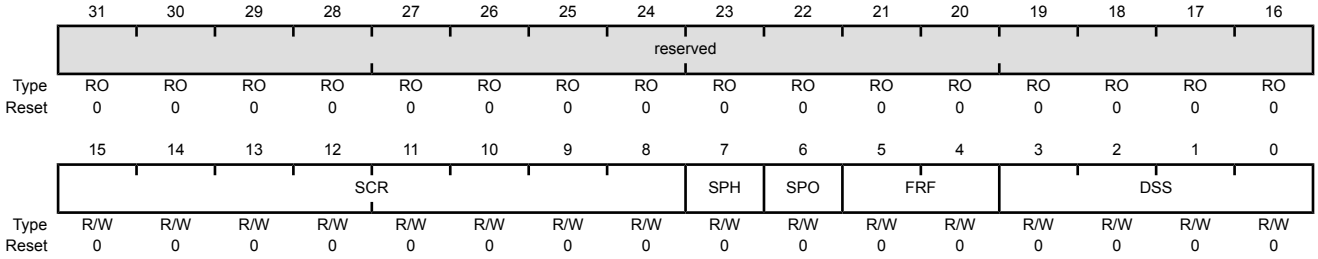
The remainder of this section lists and describes the SSI registers, in numerical order by address offset.

## Register 1: SSI Control 0 (SSICR0), offset 0x000

**SSICR0** is control register 0 and contains bit fields that control various functions within the SSI module. Functionality such as protocol mode, clock rate, and data size are configured in this register.

SSI Control 0 (SSICR0)

SSI0 base: 0x4000.8000
Offset 0x000
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:8 | SCR | R/W | 0x0000 | SSI Serial Clock Rate<br>The value SCR is used to generate the transmit and receive bit rate of the SSI. The bit rate is:<br>`BR=FSSIClk/(CPSDVSR * (1 + SCR))`<br>where CPSDVSR is an even value from 2-254 programmed in the **SSICPSR** register, and SCR is a value from 0-255. |
| 7 | SPH | R/W | 0 | SSI Serial Clock Phase<br>This bit is only applicable to the Freescale SPI Format.<br>The SPH control bit selects the clock edge that captures data and allows it to change state. It has the most impact on the first bit transmitted by either allowing or not allowing a clock transition before the first data capture edge.<br>When the SPH bit is 0, data is captured on the first clock edge transition. If SPH is 1, data is captured on the second clock edge transition. |
| 6 | SPO | R/W | 0 | SSI Serial Clock Polarity<br>This bit is only applicable to the Freescale SPI Format.<br>When the SPO bit is 0, it produces a steady state Low value on the SSIClk pin. If SPO is 1, a steady state High value is placed on the SSIClk pin when data is not being transferred. |
| 5:4 | FRF | R/W | 0x0 | SSI Frame Format Select<br>The FRF values are defined as follows:<br>Value Frame Format<br>0x0 Freescale SPI Frame Format<br>0x1 Texas Instruments Synchronous Serial Frame Format<br>0x2 MICROWIRE Frame Format<br>0x3 Reserved |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3:0 | DSS | R/W | 0x00 | SSI Data Size Select<br>The DSS values are defined as follows: |

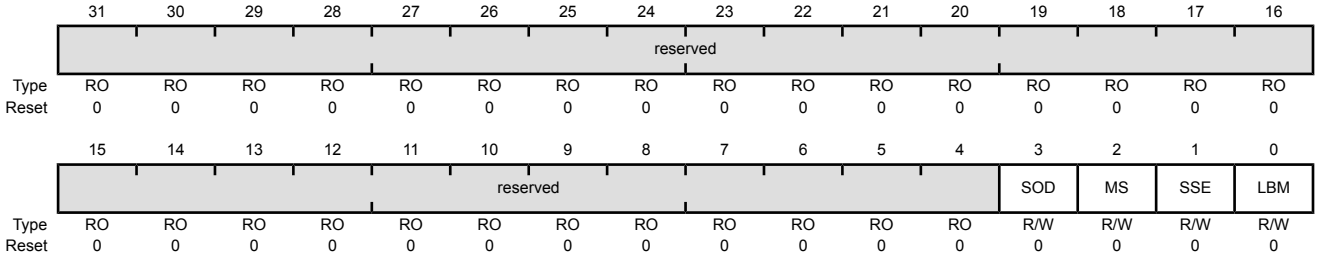
| Value | Data Size |
|---|---|
| 0x0-0x2 | Reserved |
| 0x3 | 4-bit data |
| 0x4 | 5-bit data |
| 0x5 | 6-bit data |
| 0x6 | 7-bit data |
| 0x7 | 8-bit data |
| 0x8 | 9-bit data |
| 0x9 | 10-bit data |
| 0xA | 11-bit data |
| 0xB | 12-bit data |
| 0xC | 13-bit data |
| 0xD | 14-bit data |
| 0xE | 15-bit data |
| 0xF | 16-bit data |

## Register 2: SSI Control 1 (SSICR1), offset 0x004

**SSICR1** is control register 1 and contains bit fields that control various functions within the SSI module. Master and slave mode functionality is controlled by this register.

SSI Control 1 (SSICR1)

SSI0 base: 0x4000.8000
Offset 0x004
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | SOD | R/W | 0 | SSI Slave Mode Output Disable<br>This bit is relevant only in the Slave mode (MS=1). In multiple-slave systems, it is possible for the SSI master to broadcast a message to all slaves in the system while ensuring that only one slave drives data onto the serial output line. In such systems, the TXD lines from multiple slaves could be tied together. To operate in such a system, the SOD bit can be configured so that the SSI slave does not drive the SSITx pin.<br>The SOD values are defined as follows:<br><br>Value Description<br>0 SSI can drive SSITx output in Slave Output mode.<br>1 SSI must not drive the SSITx output in Slave mode. |
| 2 | MS | R/W | 0 | SSI Master/Slave Select<br>This bit selects Master or Slave mode and can be modified only when SSI is disabled (SSE=0).<br>The MS values are defined as follows:<br><br>Value Description<br>0 Device configured as a master.<br>1 Device configured as a slave. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 1 | SSE | R/W | 0 | SSI Synchronous Serial Port Enable<br>Setting this bit enables SSI operation.<br>The SSE values are defined as follows:<br><br>Value Description<br>0 SSI operation disabled.<br>1 SSI operation enabled.<br><br>**Note:** This bit must be set to 0 before any control registers are reprogrammed. |
| 0 | LBM | R/W | 0 | SSI Loopback Mode<br>Setting this bit enables Loopback Test mode.<br>The LBM values are defined as follows:<br><br>Value Description<br>0 Normal serial port operation enabled.<br>1 Output of the transmit serial shift register is connected internally to the input of the receive serial shift register. |

## Register 3: SSI Data (SSIDR), offset 0x008

**Important:** Use caution when reading this register. Performing a read may change bit status.

**SSIDR** is the data register and is 16-bits wide. When **SSIDR** is read, the entry in the receive FIFO (pointed to by the current FIFO read pointer) is accessed. As data values are removed by the SSI receive logic from the incoming data frame, they are placed into the entry in the receive FIFO (pointed to by the current FIFO write pointer).

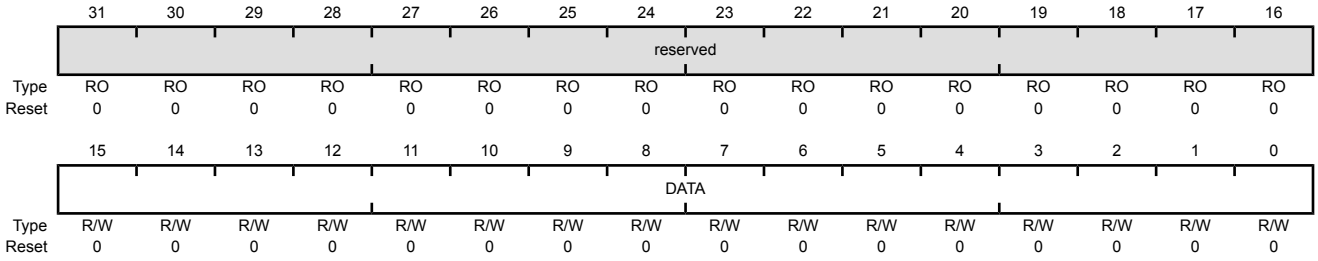
When **SSIDR** is written to, the entry in the transmit FIFO (pointed to by the write pointer) is written to. Data values are removed from the transmit FIFO one value at a time by the transmit logic. It is loaded into the transmit serial shifter, then serially shifted out onto the SSITx pin at the programmed bit rate.

When a data size of less than 16 bits is selected, the user must right-justify data written to the transmit FIFO. The transmit logic ignores the unused bits. Received data less than 16 bits is automatically right-justified in the receive buffer.

When the SSI is programmed for MICROWIRE frame format, the default size for transmit data is eight bits (the most significant byte is ignored). The receive data size is controlled by the programmer. The transmit FIFO and the receive FIFO are not cleared even when the SSE bit in the **SSICR1** register is set to zero. This allows the software to fill the transmit FIFO before enabling the SSI.

SSI Data (SSIDR)

SSI0 base: 0x4000.8000
Offset 0x008
Type R/W, reset 0x0000.0000



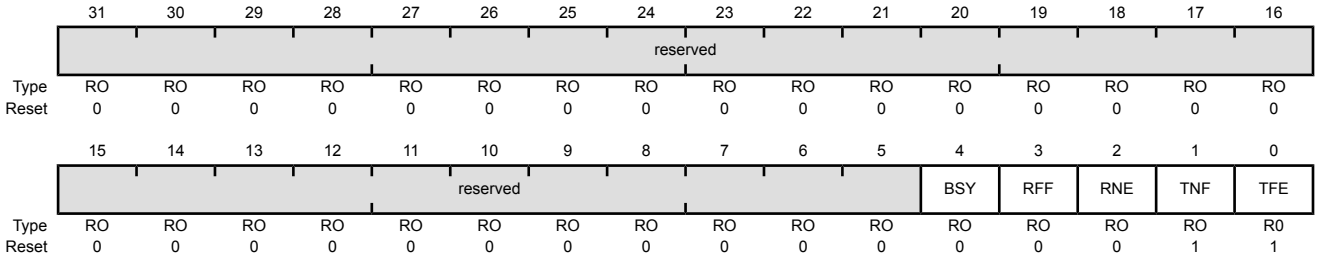

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:16 | reserved | RO | 0x0000 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 15:0 | DATA | R/W | 0x0000 | SSI Receive/Transmit Data<br>A read operation reads the receive FIFO. A write operation writes the transmit FIFO.<br>Software must right-justify data when the SSI is programmed for a data size that is less than 16 bits. Unused bits at the top are ignored by the transmit logic. The receive logic automatically right-justifies the data. |

## Register 4: SSI Status (SSISR), offset 0x00C

**SSISR** is a status register that contains bits that indicate the FIFO fill status and the SSI busy status.

SSI Status (SSISR)

SSI0 base: 0x4000.8000
Offset 0x00C
Type RO, reset 0x0000.0003




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:5 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 4 | BSY | RO | 0 | SSI Busy Bit<br>The BSY values are defined as follows:<br><br>Value Description<br>0 SSI is idle.<br>1 SSI is currently transmitting and/or receiving a frame, or the transmit FIFO is not empty. |
| 3 | RFF | RO | 0 | SSI Receive FIFO Full<br>The RFF values are defined as follows:<br><br>Value Description<br>0 Receive FIFO is not full.<br>1 Receive FIFO is full. |
| 2 | RNE | RO | 0 | SSI Receive FIFO Not Empty<br>The RNE values are defined as follows:<br><br>Value Description<br>0 Receive FIFO is empty.<br>1 Receive FIFO is not empty. |
| 1 | TNF | RO | 1 | SSI Transmit FIFO Not Full<br>The TNF values are defined as follows:<br><br>Value Description<br>0 Transmit FIFO is full.<br>1 Transmit FIFO is not full. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | TFE | R0 | 1 | SSI Transmit FIFO Empty<br>The TFE values are defined as follows:<br><br>Value Description<br>0 Transmit FIFO is not empty.<br>1 Transmit FIFO is empty. |

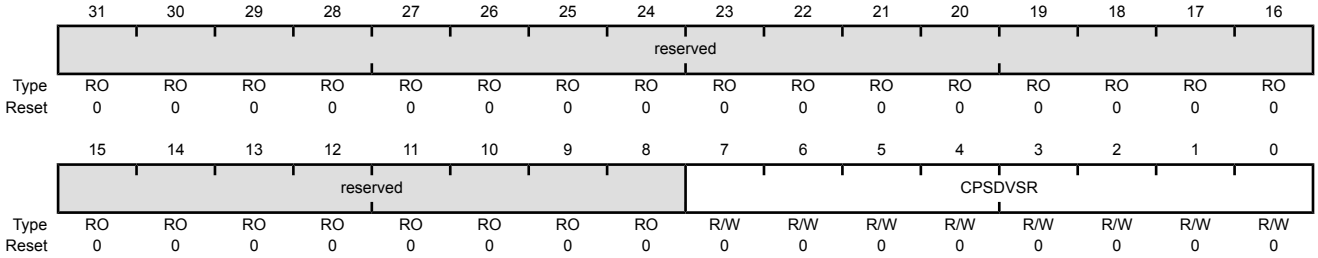
## Register 5: SSI Clock Prescale (SSICPSR), offset 0x010

**SSICPSR** is the clock prescale register and specifies the division factor by which the system clock must be internally divided before further use.

The value programmed into this register must be an even number between 2 and 254. The least-significant bit of the programmed number is hard-coded to zero. If an odd number is written to this register, data read back from this register has the least-significant bit as zero.

SSI Clock Prescale (SSICPSR)

SSI0 base: 0x4000.8000
Offset 0x010
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CPSDVSR | R/W | 0x00 | SSI Clock Prescale Divisor<br>This value must be an even number from 2 to 254, depending on the frequency of SSIClk. The LSB always returns 0 on reads. |

## Register 6: SSI Interrupt Mask (SSIIM), offset 0x014

The **SSIIM** register is the interrupt mask set or clear register. It is a read/write register and all bits are cleared to 0 on reset.

On a read, this register gives the current value of the mask on the relevant interrupt. A write of 1 to the particular bit sets the mask, enabling the interrupt to be read. A write of 0 clears the corresponding mask.

SSI Interrupt Mask (SSIIM)
SSI0 base: 0x4000.8000
Offset 0x014
Type R/W, reset 0x0000.0000

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | | | | |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | reserved | | | | | | | | | | | | TXIM | RXIM | RTIM | RORIM |
| Type | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | RO | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TXIM | R/W | 0 | SSI Transmit FIFO Interrupt Mask<br>The `TXIM` values are defined as follows:<br><br>Value Description<br>0 TX FIFO half-full or less condition interrupt is masked.<br>1 TX FIFO half-full or less condition interrupt is not masked. |
| 2 | RXIM | R/W | 0 | SSI Receive FIFO Interrupt Mask<br>The `RXIM` values are defined as follows:<br><br>Value Description<br>0 RX FIFO half-full or more condition interrupt is masked.<br>1 RX FIFO half-full or more condition interrupt is not masked. |
| 1 | RTIM | R/W | 0 | SSI Receive Time-Out Interrupt Mask<br>The `RTIM` values are defined as follows:<br><br>Value Description<br>0 RX FIFO time-out interrupt is masked.<br>1 RX FIFO time-out interrupt is not masked. |

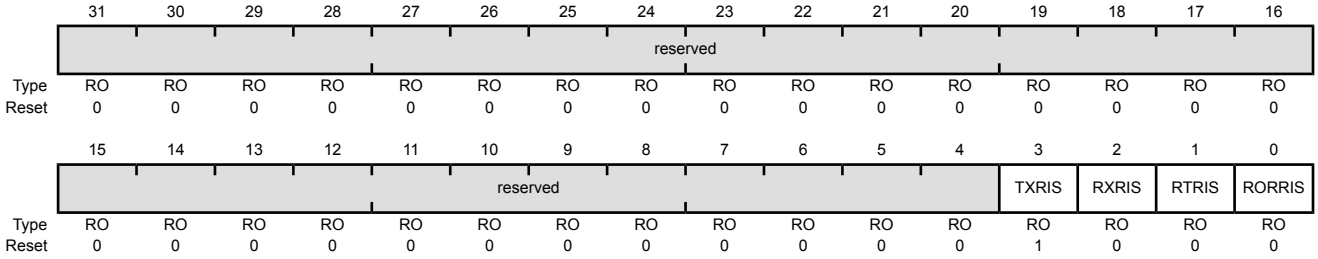
| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | RORIM | R/W | 0 | SSI Receive Overrun Interrupt Mask<br>The RORIM values are defined as follows:<br><br>Value Description<br>0 RX FIFO overrun interrupt is masked.<br>1 RX FIFO overrun interrupt is not masked. |

## Register 7: SSI Raw Interrupt Status (SSIRIS), offset 0x018

The **SSIRIS** register is the raw interrupt status register. On a read, this register gives the current raw status value of the corresponding interrupt prior to masking. A write has no effect.

SSI Raw Interrupt Status (SSIRIS)

SSI0 base: 0x4000.8000
Offset 0x018
Type RO, reset 0x0000.0008



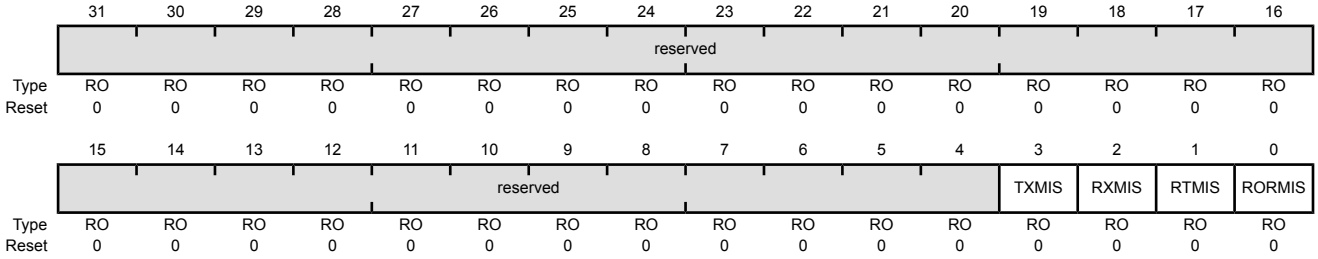

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TXRIS | RO | 1 | SSI Transmit FIFO Raw Interrupt Status<br>Indicates that the transmit FIFO is half full or less, when set. |
| 2 | RXRIS | RO | 0 | SSI Receive FIFO Raw Interrupt Status<br>Indicates that the receive FIFO is half full or more, when set. |
| 1 | RTRIS | RO | 0 | SSI Receive Time-Out Raw Interrupt Status<br>Indicates that the receive time-out has occurred, when set. |
| 0 | RORRIS | RO | 0 | SSI Receive Overrun Raw Interrupt Status<br>Indicates that the receive FIFO has overflowed, when set. |

## Register 8: SSI Masked Interrupt Status (SSIMIS), offset 0x01C

The **SSIMIS** register is the masked interrupt status register. On a read, this register gives the current masked status value of the corresponding interrupt. A write has no effect.

SSI Masked Interrupt Status (SSIMIS)

SSI0 base: 0x4000.8000
Offset 0x01C
Type RO, reset 0x0000.0000



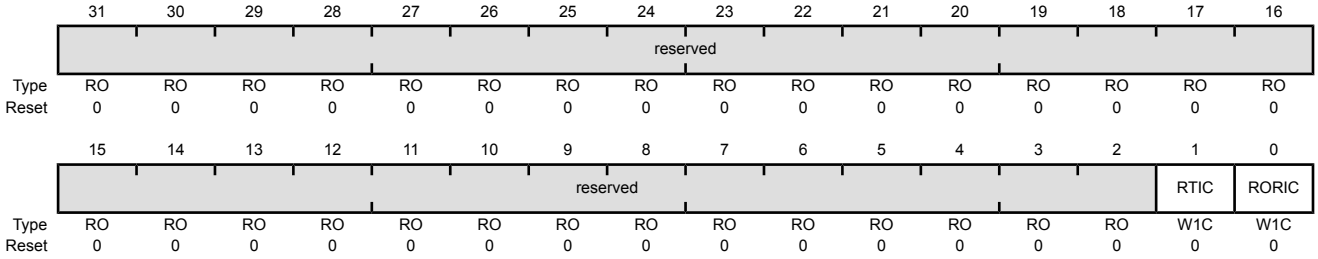

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | TXMIS | RO | 0 | SSI Transmit FIFO Masked Interrupt Status<br>Indicates that the transmit FIFO is half full or less, when set. |
| 2 | RXMIS | RO | 0 | SSI Receive FIFO Masked Interrupt Status<br>Indicates that the receive FIFO is half full or more, when set. |
| 1 | RTMIS | RO | 0 | SSI Receive Time-Out Masked Interrupt Status<br>Indicates that the receive time-out has occurred, when set. |
| 0 | RORMIS | RO | 0 | SSI Receive Overrun Masked Interrupt Status<br>Indicates that the receive FIFO has overflowed, when set. |

## Register 9: SSI Interrupt Clear (SSIICR), offset 0x020

The **SSIICR** register is the interrupt clear register. On a write of 1, the corresponding interrupt is cleared. A write of 0 has no effect.

SSI Interrupt Clear (SSIICR)

SSI0 base: 0x4000.8000
Offset 0x020
Type W1C, reset 0x0000.0000



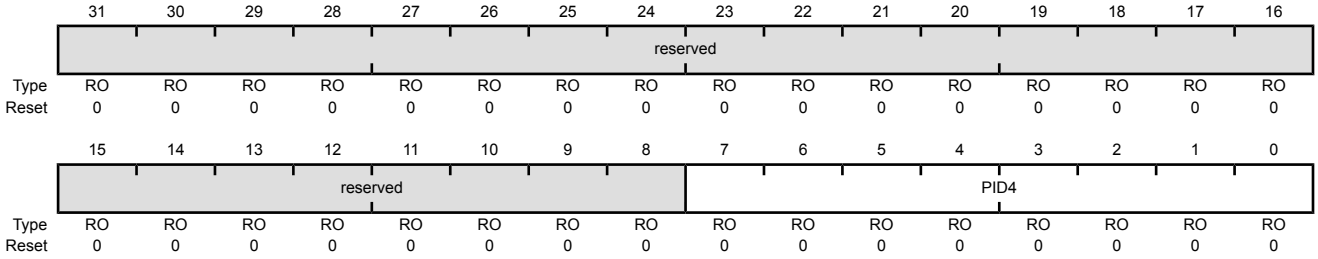

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:2 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 1 | RTIC | W1C | 0 | SSI Receive Time-Out Interrupt Clear<br>The `RTIC` values are defined as follows:<br><br>Value Description<br>0 No effect on interrupt.<br>1 Clears interrupt. |
| 0 | RORIC | W1C | 0 | SSI Receive Overrun Interrupt Clear<br>The `RORIC` values are defined as follows:<br><br>Value Description<br>0 No effect on interrupt.<br>1 Clears interrupt. |

## Register 10: SSI Peripheral Identification 4 (SSIPeriphID4), offset 0xFD0

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 4 (SSIPeriphID4)

SSI0 base: 0x4000.8000
Offset 0xFD0
Type RO, reset 0x0000.0000



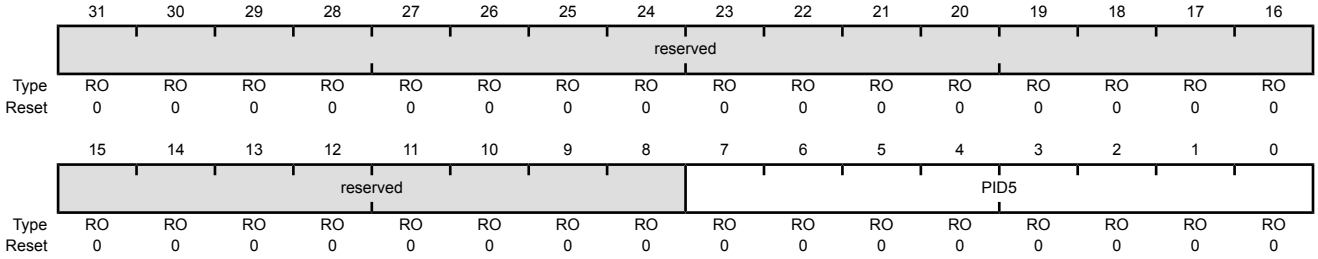

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID4 | RO | 0x00 | SSI Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

### Register 11: SSI Peripheral Identification 5 (SSIPeriphID5), offset 0xFD4

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 5 (SSIPeriphID5)

SSI0 base: 0x4000.8000
Offset 0xFD4
Type RO, reset 0x0000.0000



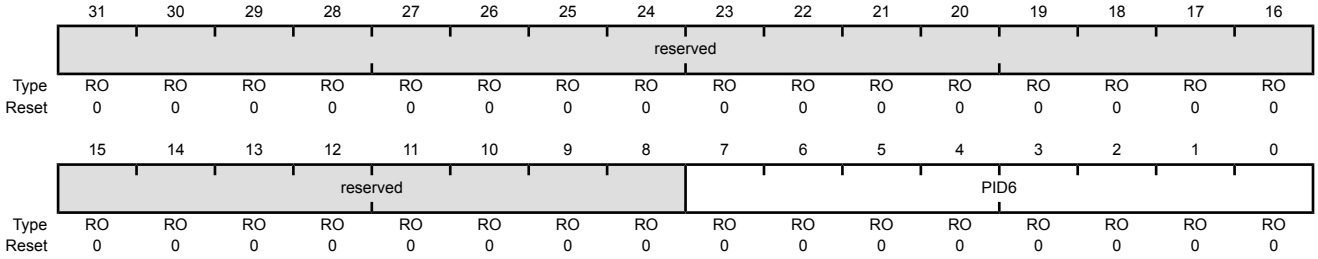

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID5 | RO | 0x00 | SSI Peripheral ID Register[15:8]<br>Can be used by software to identify the presence of this peripheral. |

### Register 12: SSI Peripheral Identification 6 (SSIPeriphID6), offset 0xFD8

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 6 (SSIPeriphID6)

SSI0 base: 0x4000.8000
Offset 0xFD8
Type RO, reset 0x0000.0000



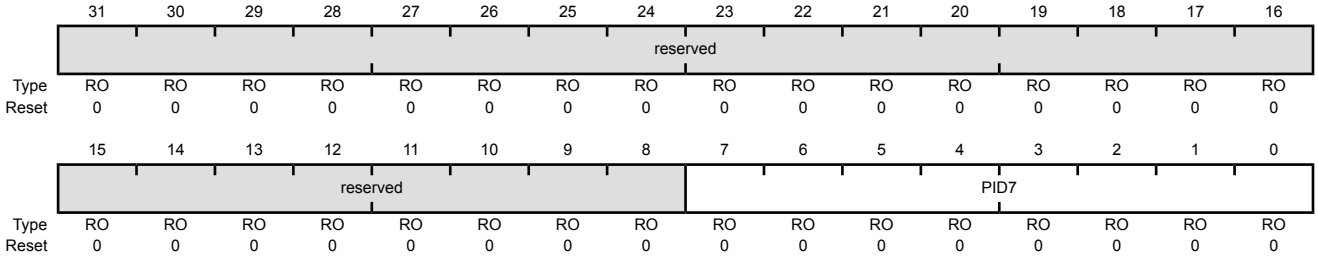

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID6 | RO | 0x00 | SSI Peripheral ID Register[23:16]<br>Can be used by software to identify the presence of this peripheral. |

## Register 13: SSI Peripheral Identification 7 (SSIPeriphID7), offset 0xFDC

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 7 (SSIPeriphID7)

SSI0 base: 0x4000.8000
Offset 0xFDC
Type RO, reset 0x0000.0000



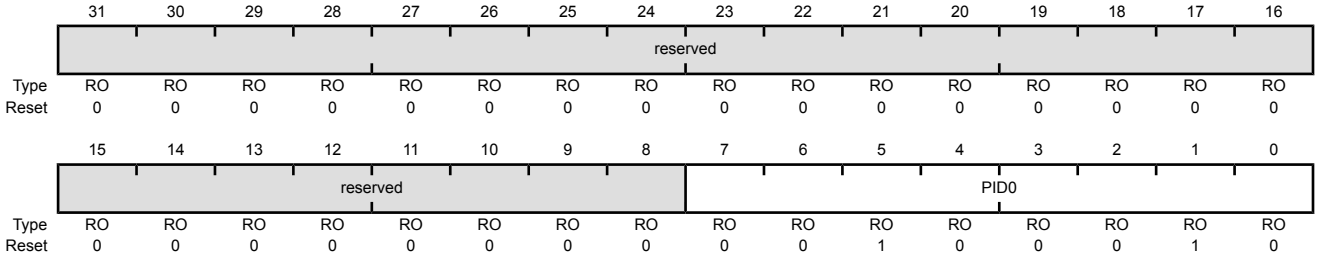

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID7 | RO | 0x00 | SSI Peripheral ID Register[31:24]<br>Can be used by software to identify the presence of this peripheral. |

### Register 14: SSI Peripheral Identification 0 (SSIPeriphID0), offset 0xFE0

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 0 (SSIPeriphID0)

SSI0 base: 0x4000.8000
Offset 0xFE0
Type RO, reset 0x0000.0022



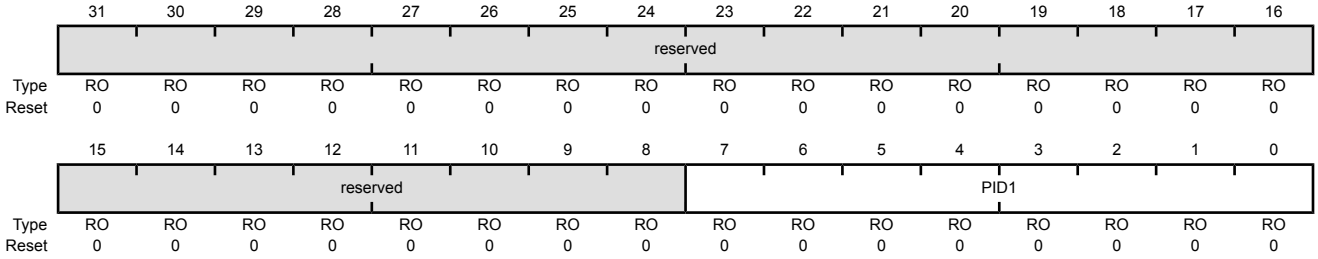

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID0 | RO | 0x22 | SSI Peripheral ID Register[7:0]<br>Can be used by software to identify the presence of this peripheral. |

### Register 15: SSI Peripheral Identification 1 (SSIPeriphID1), offset 0xFE4

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 1 (SSIPeriphID1)

SSI0 base: 0x4000.8000
Offset 0xFE4
Type RO, reset 0x0000.0000



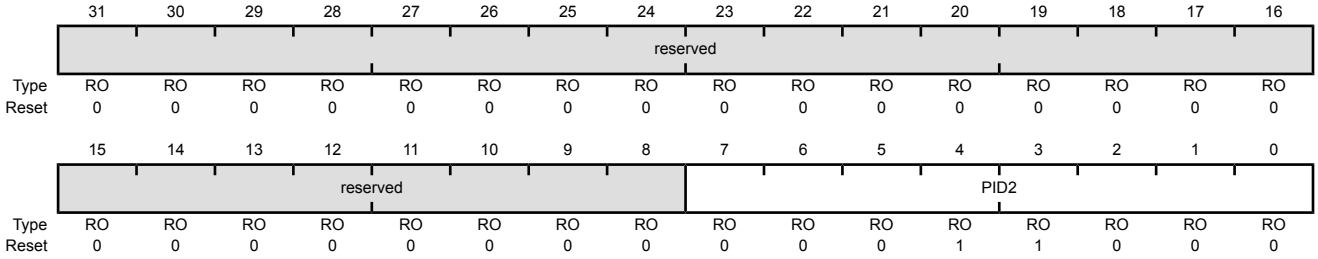

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID1 | RO | 0x00 | SSI Peripheral ID Register [15:8]<br>Can be used by software to identify the presence of this peripheral. |

## Register 16: SSI Peripheral Identification 2 (SSIPeriphID2), offset 0xFE8

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 2 (SSIPeriphID2)

SSI0 base: 0x4000.8000
Offset 0xFE8
Type RO, reset 0x0000.0018



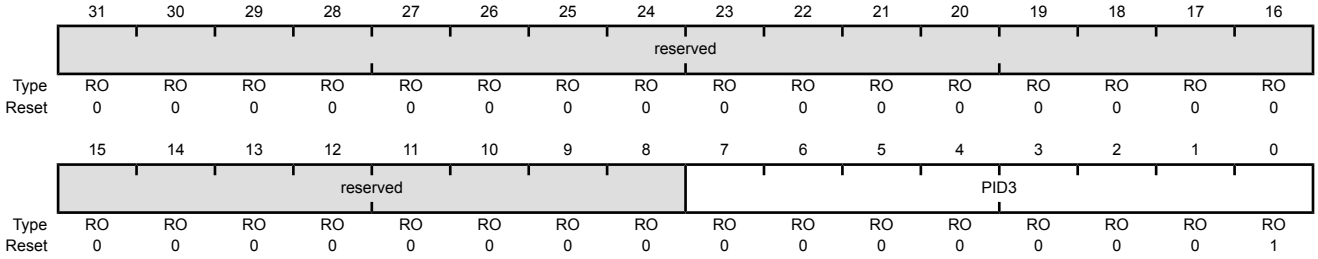

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID2 | RO | 0x18 | SSI Peripheral ID Register [23:16]<br>Can be used by software to identify the presence of this peripheral. |

## Register 17: SSI Peripheral Identification 3 (SSIPeriphID3), offset 0xFEC

The **SSIPeriphIDn** registers are hard-coded and the fields within the register determine the reset value.

SSI Peripheral Identification 3 (SSIPeriphID3)

SSI0 base: 0x4000.8000
Offset 0xFEC
Type RO, reset 0x0000.0001



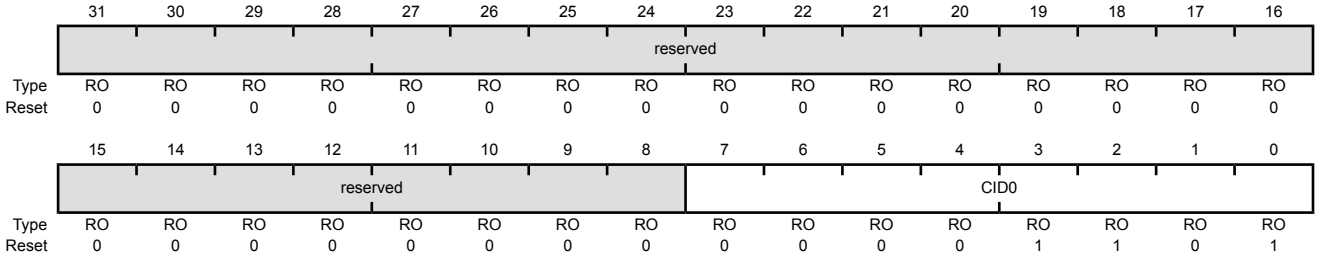

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | PID3 | RO | 0x01 | SSI Peripheral ID Register [31:24]<br>Can be used by software to identify the presence of this peripheral. |

### Register 18: SSI PrimeCell Identification 0 (SSIPCellID0), offset 0xFF0

The **SSIPCellIDn** registers are hard-coded, and the fields within the register determine the reset value.

SSI PrimeCell Identification 0 (SSIPCellID0)

SSI0 base: 0x4000.8000
Offset 0xFF0
Type RO, reset 0x0000.000D



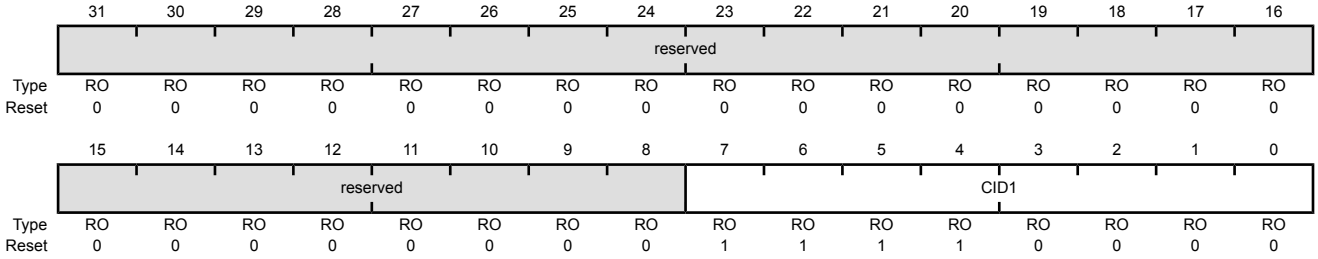

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID0 | RO | 0x0D | SSI PrimeCell ID Register [7:0]<br>Provides software a standard cross-peripheral identification system. |

## Register 19: SSI PrimeCell Identification 1 (SSIPCellID1), offset 0xFF4

The **SSIPCellIDn** registers are hard-coded, and the fields within the register determine the reset value.

SSI PrimeCell Identification 1 (SSIPCellID1)

SSI0 base: 0x4000.8000
Offset 0xFF4
Type RO, reset 0x0000.00F0



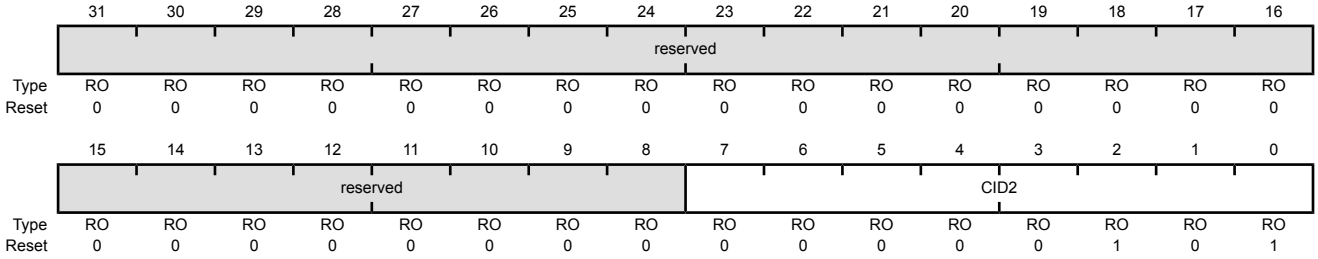

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID1 | RO | 0xF0 | SSI PrimeCell ID Register [15:8]<br>Provides software a standard cross-peripheral identification system. |

## Register 20: SSI PrimeCell Identification 2 (SSIPCellID2), offset 0xFF8

The **SSIPCellIDn** registers are hard-coded, and the fields within the register determine the reset value.

SSI PrimeCell Identification 2 (SSIPCellID2)

SSI0 base: 0x4000.8000
Offset 0xFF8
Type RO, reset 0x0000.0005



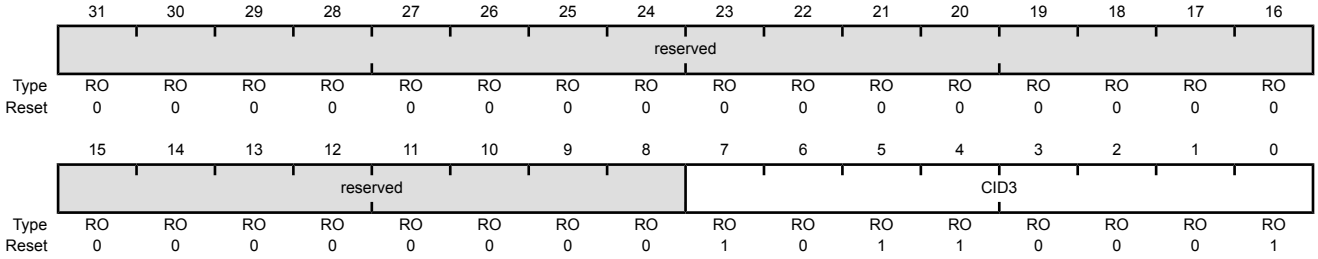

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID2 | RO | 0x05 | SSI PrimeCell ID Register [23:16]<br>Provides software a standard cross-peripheral identification system. |

## Register 21: SSI PrimeCell Identification 3 (SSIPCellID3), offset 0xFFC

The **SSIPCellIDn** registers are hard-coded, and the fields within the register determine the reset value.

SSI PrimeCell Identification 3 (SSIPCellID3)

SSI0 base: 0x4000.8000
Offset 0xFFC
Type RO, reset 0x0000.00B1




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | CID3 | RO | 0xB1 | SSI PrimeCell ID Register [31:24]<br>Provides software a standard cross-peripheral identification system. |

# 13 Inter-Integrated Circuit (I2C) Interface

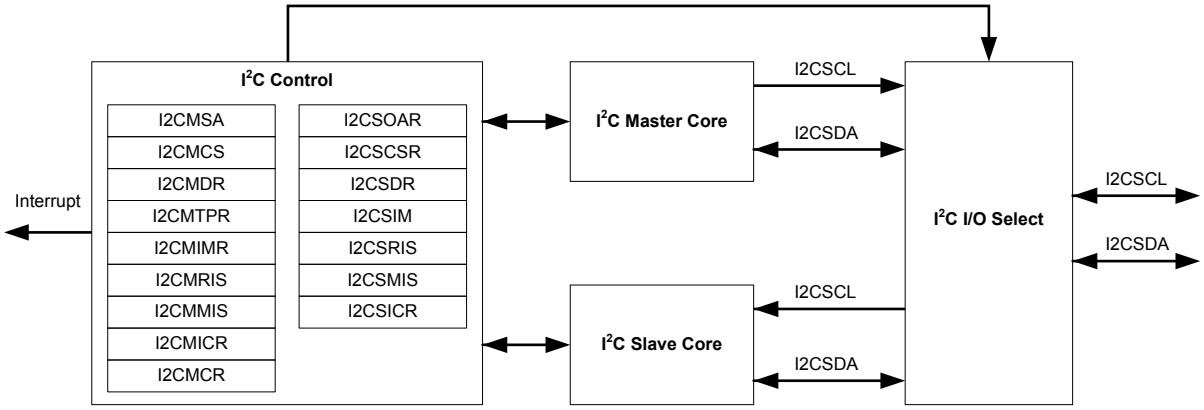
The Inter-Integrated Circuit (I2C) bus provides bi-directional data transfer through a two-wire design (a serial data line SDA and a serial clock line SCL), and interfaces to external I2C devices such as serial memory (RAMs and ROMs), networking devices, LCDs, tone generators, and so on. The I2C bus may also be used for system testing and diagnostic purposes in product development and manufacture. The LM3S828 microcontroller includes one I2C module, providing the ability to interact (both send and receive) with other I2C devices on the bus.

The Stellaris® I2C interface has the following features:

- Devices on the I2C bus can be designated as either a master or a slave
  - Supports both sending and receiving data as either a master or a slave
  - Supports simultaneous master and slave operation
- Four I2C modes
  - Master transmit
  - Master receive
  - Slave transmit
  - Slave receive
- Two transmission speeds: Standard (100 Kbps) and Fast (400 Kbps)
- Master and slave interrupt generation
  - Master generates interrupts when a transmit or receive operation completes (or aborts due to an error)
  - Slave generates interrupts when data has been sent or requested by a master
- Master with arbitration and clock synchronization, multimaster support, and 7-bit addressing mode

## 13.1 Block Diagram

**Figure 13-1. I²C Block Diagram**

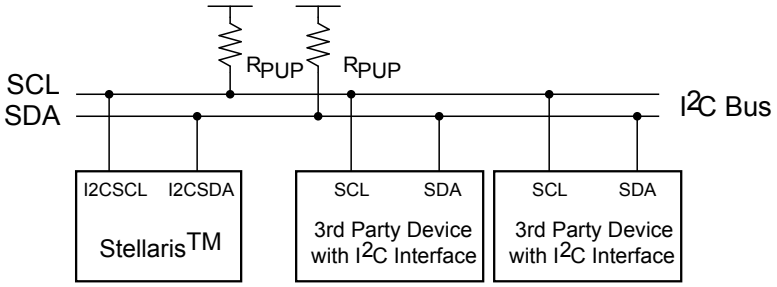



## 13.2 Functional Description

I²C module is comprised of both master and slave functions which are implemented as separate peripherals. For proper operation, the SDA and SCL pins must be connected to bi-directional open-drain pads. A typical I²C bus configuration is shown in Figure 13-2 on page 431.

See "Inter-Integrated Circuit (I²C) Interface" on page 487 for I²C timing diagrams.

**Figure 13-2. I²C Bus Configuration**




### 13.2.1 I²C Bus Functional Overview

The I²C bus uses only two signals: SDA and SCL, named `I2CSDA` and `I2CSCL` on Stellaris microcontrollers. SDA is the bi-directional serial data line and SCL is the bi-directional serial clock line. The bus is considered idle when both lines are High.

Every transaction on the I²C bus is nine bits long, consisting of eight data bits and a single acknowledge bit. The number of bytes per transfer (defined as the time between a valid START and STOP condition, described in "START and STOP Conditions" on page 432) is unrestricted, but each byte has to be followed by an acknowledge bit, and data must be transferred MSB first. When a receiver cannot receive another complete byte, it can hold the clock line SCL Low and force the transmitter into a wait state. The data transfer continues when the receiver releases the clock SCL.

#### 13.2.1.1 START and STOP Conditions

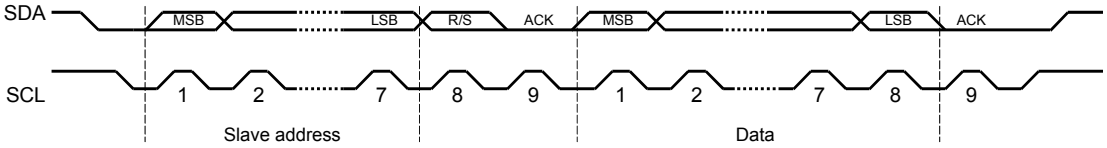
The protocol of the I2C bus defines two states to begin and end a transaction: START and STOP. A High-to-Low transition on the SDA line while the SCL is High is defined as a START condition, and a Low-to-High transition on the SDA line while SCL is High is defined as a STOP condition. The bus is considered busy after a START condition and free after a STOP condition. See Figure 13-3 on page 432.

**Figure 13-3. START and STOP Conditions**


#### 13.2.1.2 Data Format with 7-Bit Address

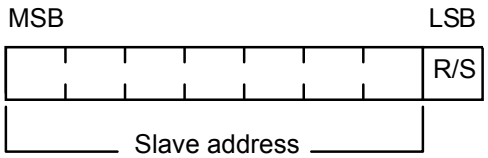
Data transfers follow the format shown in Figure 13-4 on page 432. After the START condition, a slave address is sent. This address is 7-bits long followed by an eighth bit, which is a data direction bit (R/S bit in the **I2CMSA** register). A zero indicates a transmit operation (send), and a one indicates a request for data (receive). A data transfer is always terminated by a STOP condition generated by the master, however, a master can initiate communications with another device on the bus by generating a repeated START condition and addressing another slave without first generating a STOP condition. Various combinations of receive/send formats are then possible within a single transfer.

**Figure 13-4. Complete Data Transfer with a 7-Bit Address**




The first seven bits of the first byte make up the slave address (see Figure 13-5 on page 432). The eighth bit determines the direction of the message. A zero in the R/S position of the first byte means that the master will write (send) data to the selected slave, and a one in this position means that the master will receive data from the slave.

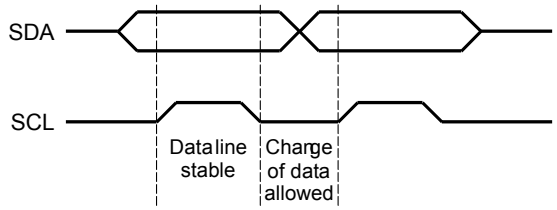
**Figure 13-5. R/S Bit in First Byte**




#### 13.2.1.3 Data Validity

The data on the SDA line must be stable during the high period of the clock, and the data line can only change when SCL is Low (see Figure 13-6 on page 433).

**Figure 13-6. Data Validity During Bit Transfer on the I²C Bus**




#### 13.2.1.4 Acknowledge

All bus transactions have a required acknowledge clock cycle that is generated by the master. During the acknowledge cycle, the transmitter (which can be the master or slave) releases the SDA line. To acknowledge the transaction, the receiver must pull down SDA during the acknowledge clock cycle. The data sent out by the receiver during the acknowledge cycle must comply with the data validity requirements described in “Data Validity” on page 432.

When a slave receiver does not acknowledge the slave address, SDA must be left High by the slave so that the master can generate a STOP condition and abort the current transfer. If the master device is acting as a receiver during a transfer, it is responsible for acknowledging each transfer made by the slave. Since the master controls the number of bytes in the transfer, it signals the end of data to the slave transmitter by not generating an acknowledge on the last data byte. The slave transmitter must then release SDA to allow the master to generate the STOP or a repeated START condition.

#### 13.2.1.5 Arbitration

A master may start a transfer only if the bus is idle. It's possible for two or more masters to generate a START condition within minimum hold time of the START condition. In these situations, an arbitration scheme takes place on the SDA line, while SCL is High. During arbitration, the first of the competing master devices to place a '1' (High) on SDA while another master transmits a '0' (Low) will switch off its data output stage and retire until the bus is idle again.

Arbitration can take place over several bits. Its first stage is a comparison of address bits, and if both masters are trying to address the same device, arbitration continues on to the comparison of data bits.

### 13.2.2 Available Speed Modes

The I²C clock rate is determined by the parameters: `CLK_PRD`, `TIMER_PRD`, `SCL_LP`, and `SCL_HP`.

where:

`CLK_PRD` is the system clock period

`SCL_LP` is the low phase of SCL (fixed at 6)

`SCL_HP` is the high phase of SCL (fixed at 4)

`TIMER_PRD` is the programmed value in the **I²C Master Timer Period (I2CMTPR)** register (see page 451).

The I²C clock period is calculated as follows:

```
SCL_PERIOD = 2*(1 + TIMER_PRD)*(SCL_LP + SCL_HP)*CLK_PRD
```

For example:

```
CLK_PRD = 50 ns
TIMER_PRD = 2
SCL_LP=6
SCL_HP=4
```

yields a SCL frequency of:

```
1/T = 333 Khz
```

Table 13-1 on page 434 gives examples of timer period, system clock, and speed mode (Standard or Fast).

**Table 13-1. Examples of I2C Master Timer Period versus Speed Mode**

| System Clock | Timer Period | Standard Mode | Timer Period | Fast Mode |
|---|---|---|---|---|
| 4 MHz | 0x01 | 100 Kbps | - | - |
| 6 MHz | 0x02 | 100 Kbps | - | - |
| 12.5 MHz | 0x06 | 89 Kbps | 0x01 | 312 Kbps |
| 16.7 MHz | 0x08 | 93 Kbps | 0x02 | 278 Kbps |
| 20 MHz | 0x09 | 100 Kbps | 0x02 | 333 Kbps |
| 25 MHz | 0x0C | 96.2 Kbps | 0x03 | 312 Kbps |
| 33 MHz | 0x10 | 97.1 Kbps | 0x04 | 330 Kbps |
| 40 MHz | 0x13 | 100 Kbps | 0x04 | 400 Kbps |
| 50 MHz | 0x18 | 100 Kbps | 0x06 | 357 Kbps |

### 13.2.3 Interrupts

The I2C can generate interrupts when the following conditions are observed:

- Master transaction completed
- Master arbitration lost
- Master transaction error
- Slave transaction received
- Slave transaction requested

There is a separate interrupt signal for the I2C master and I2C slave modules. While both modules can generate interrupts for multiple conditions, only a single interrupt signal is sent to the interrupt controller.

#### 13.2.3.1 I2C Master Interrupts

The I2C master module generates an interrupt when a transaction completes (either transmit or receive), when arbitration is lost, or when an error occurs during a transaction. To enable the I2C master interrupt, software must set the IM bit in the **I2C Master Interrupt Mask (I2CMIMR)** register. When an interrupt condition is met, software must check the ERROR and ARBLST bits in the **I2C Master Control/Status (I2CMCS)** register to verify that an error didn't occur during the last transaction and to ensure that arbitration has not been lost. An error condition is asserted if the last transaction wasn't acknowledged by the slave. If an error is not detected and the master has not lost arbitration,

the application can proceed with the transfer. The interrupt is cleared by writing a 1 to the IC bit in the **I²C Master Interrupt Clear (I2CMICR)** register.

If the application doesn't require the use of interrupts, the raw interrupt status is always visible via the **I²C Master Raw Interrupt Status (I2CMRIS)** register.

#### 13.2.3.2 I²C Slave Interrupts

The slave module can generate an interrupt when data has been received or requested. This interrupt is enabled by writing a 1 to the DATAIM bit in the **I²C Slave Interrupt Mask (I2CSIMR)** register. Software determines whether the module should write (transmit) or read (receive) data from the **I²C Slave Data (I2CSDR)** register, by checking the RREQ and TREQ bits of the **I²C Slave Control/Status (I2CSCSR)** register. If the slave module is in receive mode and the first byte of a transfer is received, the FBR bit is set along with the RREQ bit. The interrupt is cleared by writing a 1 to the DATAIC bit in the **I²C Slave Interrupt Clear (I2CSICR)** register.

If the application doesn't require the use of interrupts, the raw interrupt status is always visible via the **I²C Slave Raw Interrupt Status (I2CSRIS)** register.

### 13.2.4 Loopback Operation

The I²C modules can be placed into an internal loopback mode for diagnostic or debug work. This is accomplished by setting the LPBK bit in the **I²C Master Configuration (I2CMCR)** register. In loopback mode, the SDA and SCL signals from the master and slave modules are tied together.

### 13.2.5 Command Sequence Flow Charts

This section details the steps required to perform the various I²C transfer types in both master and slave mode.

#### 13.2.5.1 I²C Master Command Sequences

The figures that follow show the command sequences available for the I²C master.

**Figure 13-7. Master Single SEND**

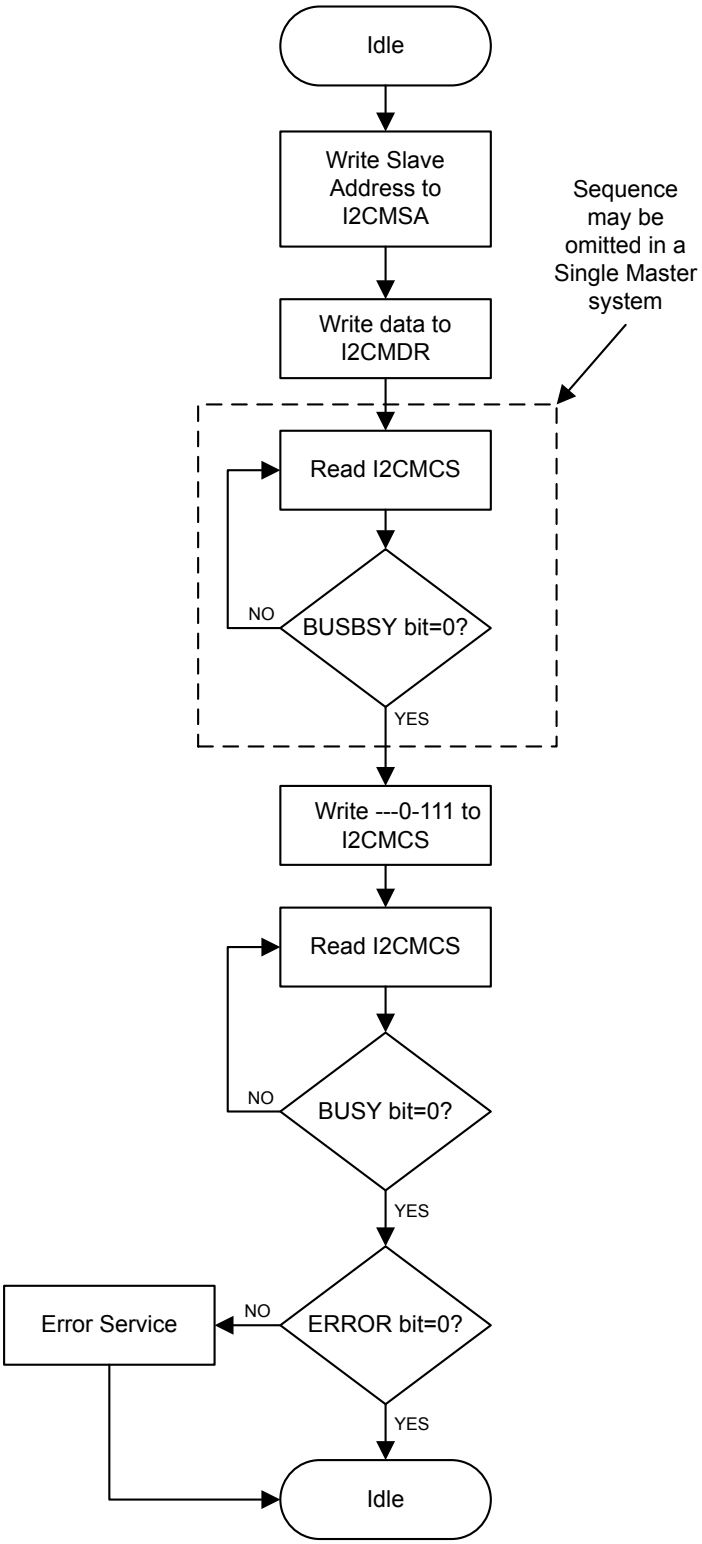


**Figure 13-8. Master Single RECEIVE**

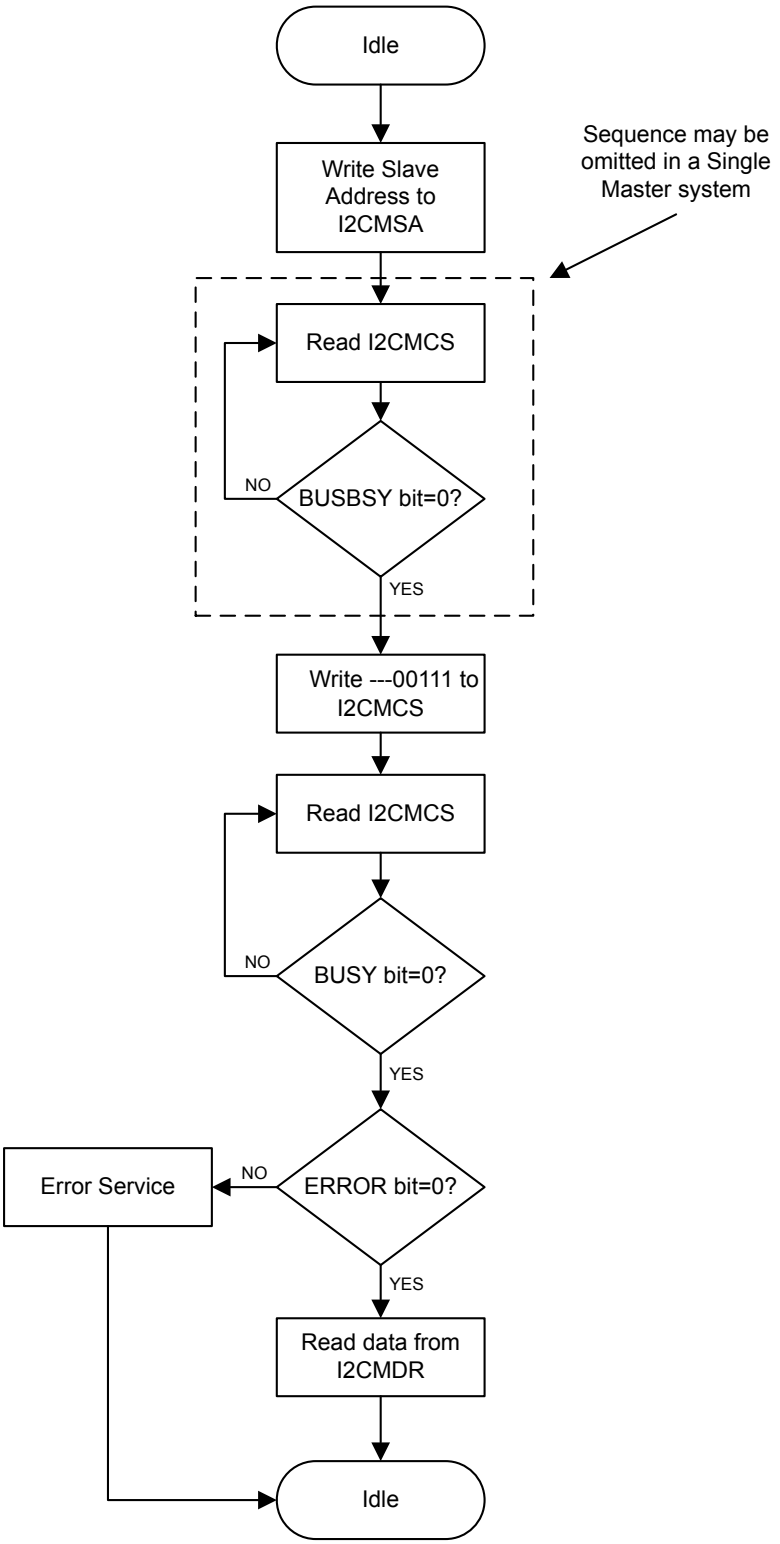


**Figure 13-9. Master Burst SEND**

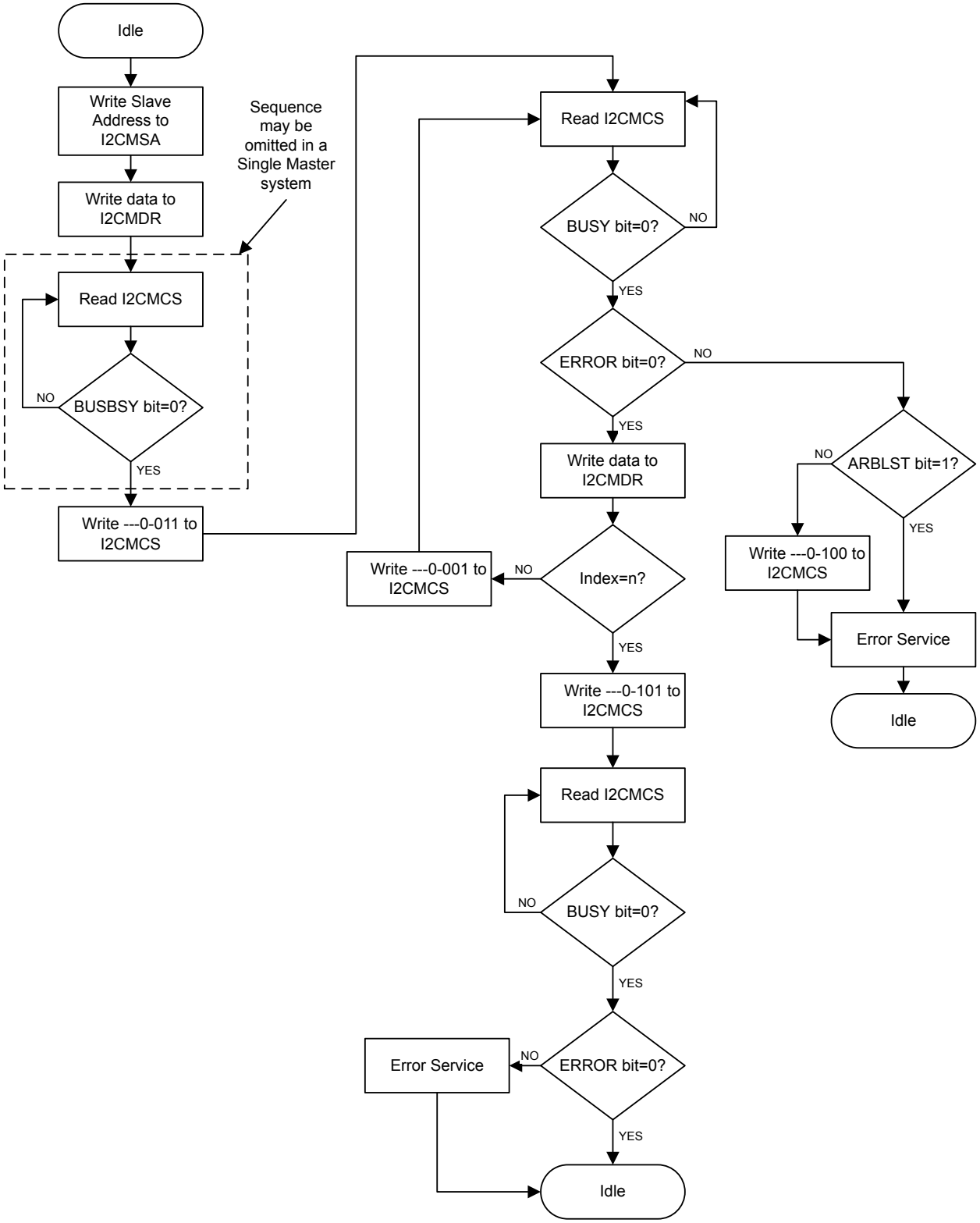


**Figure 13-10. Master Burst RECEIVE**

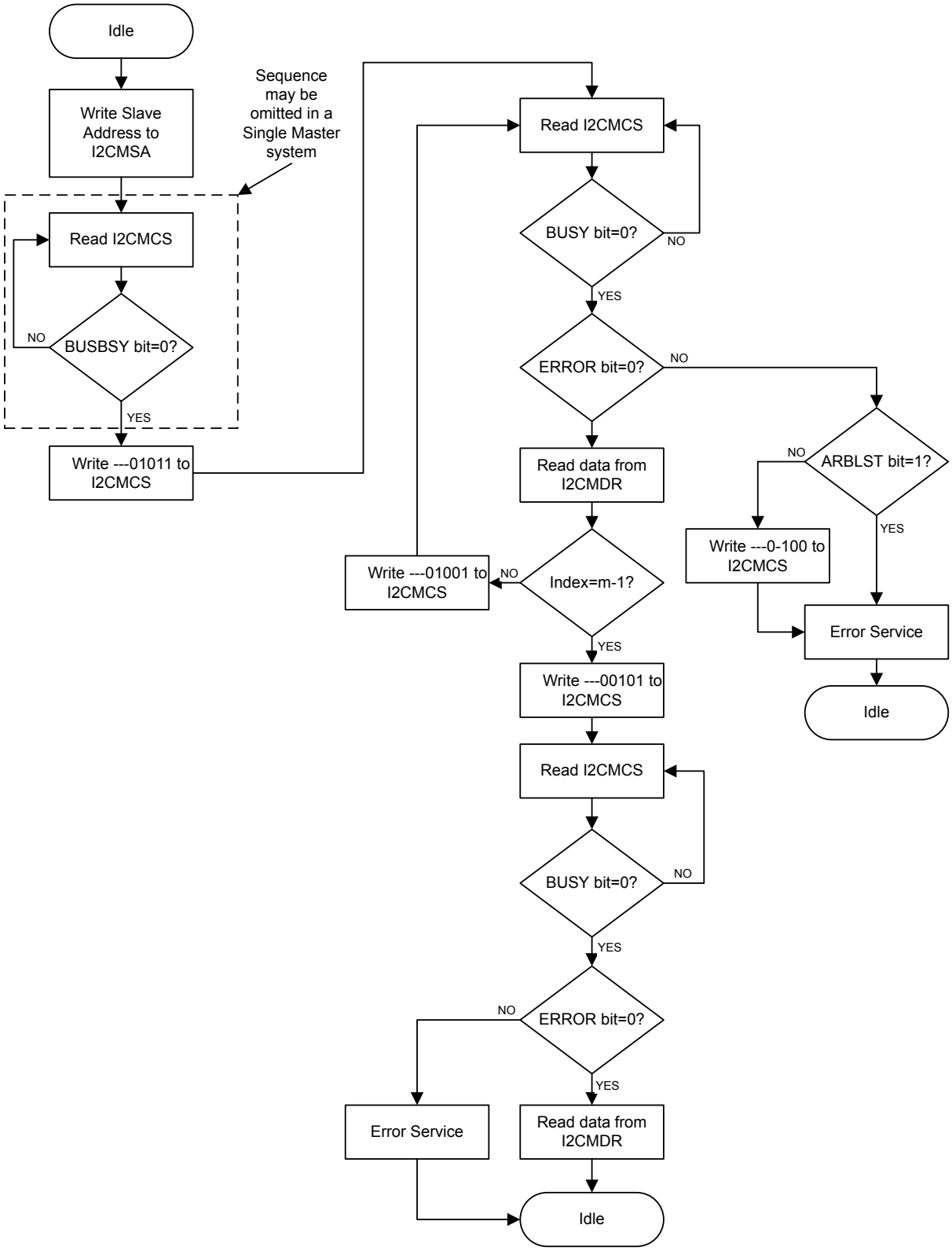


**Figure 13-11. Master Burst RECEIVE after Burst SEND**

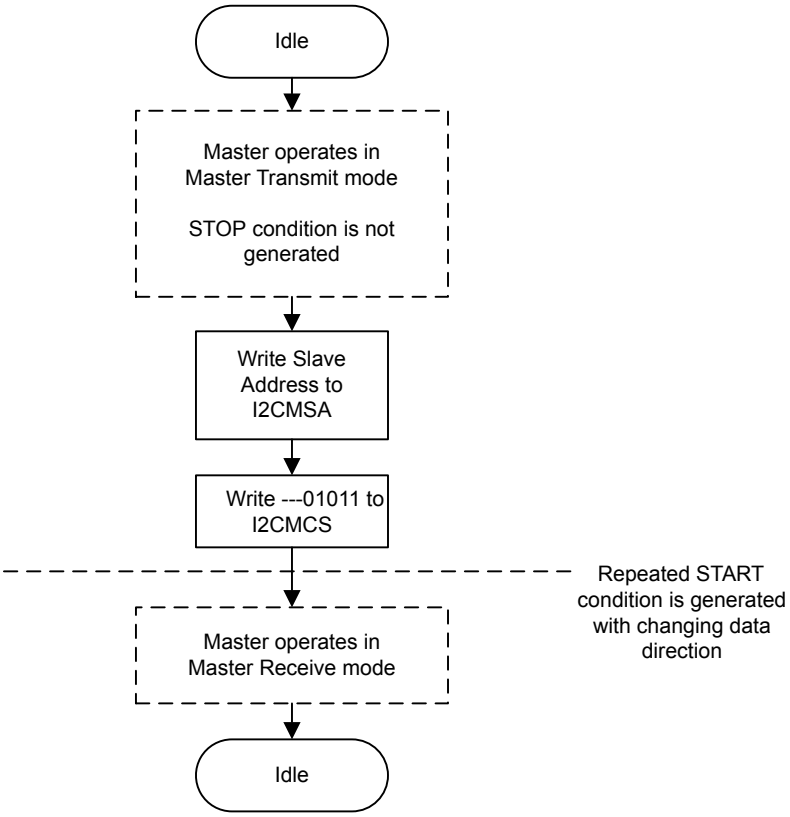
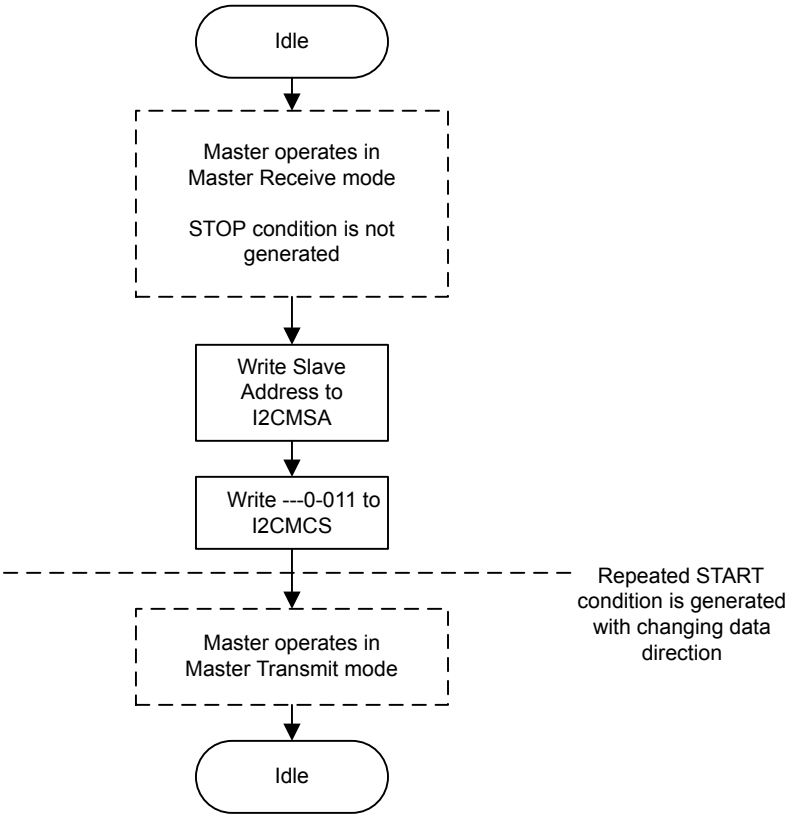


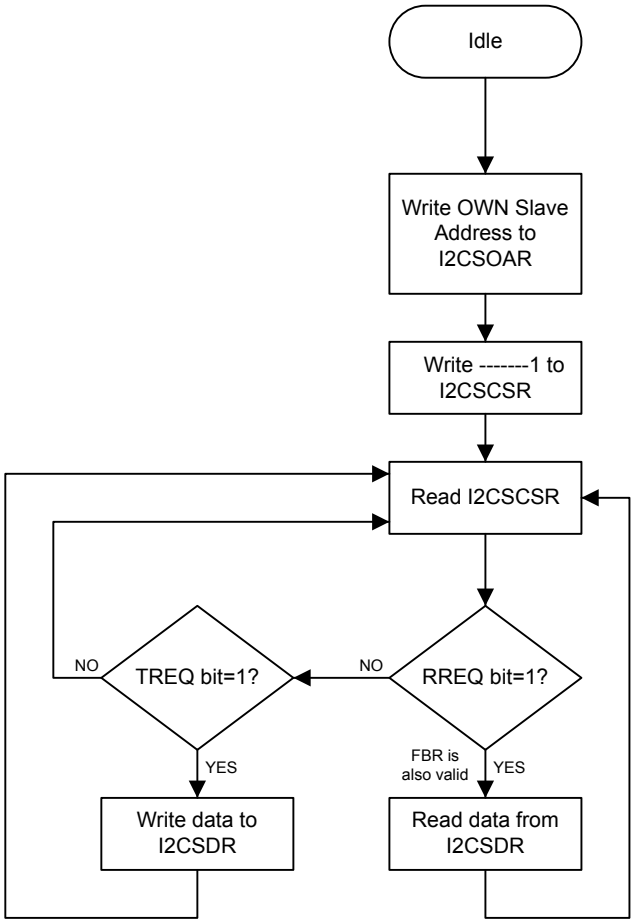
**Figure 13-12. Master Burst SEND after Burst RECEIVE**




### 13.2.5.2 I²C Slave Command Sequences

Figure 13-13 on page 442 presents the command sequence available for the I²C slave.

**Figure 13-13. Slave Command Sequence**




## 13.3 Initialization and Configuration

The following example shows how to configure the I2C module to send a single byte as a master. This assumes the system clock is 20 MHz.

1. Enable the I2C clock by writing a value of 0x0000.1000 to the **RCGC1** register in the System Control module.
2. Enable the clock to the appropriate GPIO module via the **RCGC2** register in the System Control module.
3. In the GPIO module, enable the appropriate pins for their alternate function using the **GPIOAFSEL** register. Also, be sure to enable the same pins for Open Drain operation.
4. Initialize the I2C Master by writing the **I2CMCR** register with a value of 0x0000.0020.
5. Set the desired SCL clock speed of 100 Kbps by writing the **I2CMTPR** register with the correct value. The value written to the **I2CMTPR** register represents the number of system clock periods in one SCL clock period. The TPR value is determined by the following equation:

```
TPR = (System Clock / (2 * (SCL_LP + SCL_HP) * SCL_CLK)) - 1;
TPR = (20MHz / (2 * (6 + 4) * 100000)) - 1;
TPR = 9
```

Write the **I2CMTPR** register with the value of 0x0000.0009.

6. Specify the slave address of the master and that the next operation will be a Send by writing the **I2CMSA** register with a value of 0x0000.0076. This sets the slave address to 0x3B.
7. Place data (byte) to be sent in the data register by writing the **I2CMDR** register with the desired data.
8. Initiate a single byte send of the data from Master to Slave by writing the **I2CMCS** register with a value of 0x0000.0007 (STOP, START, RUN).
9. Wait until the transmission completes by polling the **I2CMCS** register's BUSBSY bit until it has been cleared.

## 13.4 Register Map

Table 13-2 on page 443 lists the $I^2C$ registers. All addresses given are relative to the $I^2C$ base addresses for the master and slave:

- $I^2C$ 0: 0x4002.0000

Note that the $I^2C$ module clock must be enabled before the registers can be programmed (see page 191). There must be a delay of 3 system clocks after the $I^2C$ module clock is enabled before any $I^2C$ module registers are accessed.

The hw_i2c.h file in the StellarisWare® Driver Library uses a base address of 0x800 for the $I^2C$ slave registers. Be aware when using registers with offsets between 0x800 and 0x818 that StellarisWare uses an offset between 0x000 and 0x018 with the slave base address.

**Table 13-2. Inter-Integrated Circuit ($I^2C$) Interface Register Map**

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| **$I^2C$ Master** | | | | | |
| 0x000 | I2CMSA | R/W | 0x0000.0000 | I2C Master Slave Address | 445 |
| 0x004 | I2CMCS | R/W | 0x0000.0000 | I2C Master Control/Status | 446 |
| 0x008 | I2CMDR | R/W | 0x0000.0000 | I2C Master Data | 450 |
| 0x00C | I2CMTPR | R/W | 0x0000.0001 | I2C Master Timer Period | 451 |
| 0x010 | I2CMIMR | R/W | 0x0000.0000 | I2C Master Interrupt Mask | 452 |
| 0x014 | I2CMRIS | RO | 0x0000.0000 | I2C Master Raw Interrupt Status | 453 |
| 0x018 | I2CMMIS | RO | 0x0000.0000 | I2C Master Masked Interrupt Status | 454 |
| 0x01C | I2CMICR | WO | 0x0000.0000 | I2C Master Interrupt Clear | 455 |
| 0x020 | I2CMCR | R/W | 0x0000.0000 | I2C Master Configuration | 456 |

**Table 13-2. Inter-Integrated Circuit (I2C) Interface Register Map *(continued)***

| Offset | Name | Type | Reset | Description | See page |
|---|---|---|---|---|---|
| **I2C Slave** | | | | | |
| 0x800 | I2CSOAR | R/W | 0x0000.0000 | I2C Slave Own Address | 458 |
| 0x804 | I2CSCSR | RO | 0x0000.0000 | I2C Slave Control/Status | 459 |
| 0x808 | I2CSDR | R/W | 0x0000.0000 | I2C Slave Data | 461 |
| 0x80C | I2CSIMR | R/W | 0x0000.0000 | I2C Slave Interrupt Mask | 462 |
| 0x810 | I2CSRIS | RO | 0x0000.0000 | I2C Slave Raw Interrupt Status | 463 |
| 0x814 | I2CSMIS | RO | 0x0000.0000 | I2C Slave Masked Interrupt Status | 464 |
| 0x818 | I2CSICR | WO | 0x0000.0000 | I2C Slave Interrupt Clear | 465 |

## 13.5 Register Descriptions (I2C Master)

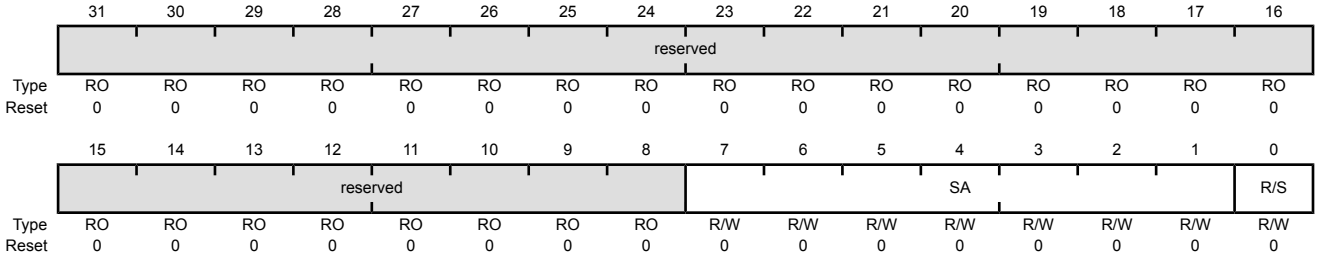
The remainder of this section lists and describes the I2C master registers, in numerical order by address offset. See also “Register Descriptions (I2C Slave)” on page 457.

## Register 1: I²C Master Slave Address (I2CMSA), offset 0x000

This register consists of eight bits: seven address bits (A6-A0), and a Receive/Send bit, which determines if the next operation is a Receive (High), or Send (Low).

I2C Master Slave Address (I2CMSA)

I2C 0 base: 0x4002.0000
Offset 0x000
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:1 | SA | R/W | 0 | I²C Slave Address<br>This field specifies bits A6 through A0 of the slave address. |
| 0 | R/S | R/W | 0 | Receive/Send<br>The R/S bit specifies if the next operation is a Receive (High) or Send (Low).<br><br>Value Description<br>0 Send.<br>1 Receive. |

## Register 2: I2C Master Control/Status (I2CMCS), offset 0x004

This register accesses four control bits when written, and accesses seven status bits when read.

The status register consists of seven bits, which when read determine the state of the I2C bus controller.

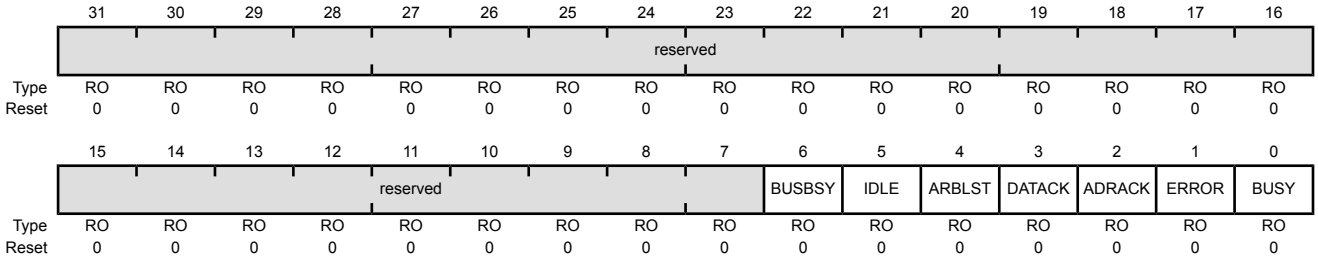
The control register consists of four bits: the RUN, START, STOP, and ACK bits. The START bit causes the generation of the START, or REPEATED START condition.

The STOP bit determines if the cycle stops at the end of the data cycle, or continues on to a burst. To generate a single send cycle, the **I2C Master Slave Address (I2CMSA)** register is written with the desired address, the R/S bit is set to 0, and the Control register is written with ACK=X (0 or 1), STOP=1, START=1, and RUN=1 to perform the operation and stop. When the operation is completed (or aborted due an error), the interrupt pin becomes active and the data may be read from the **I2CMDR** register. When the I2C module operates in Master receiver mode, the ACK bit must be set normally to logic 1. This causes the I2C bus controller to send an acknowledge automatically after each byte. This bit must be reset when the I2C bus controller requires no further data to be sent from the slave transmitter.

### Reads

I2C Master Control/Status (I2CMCS)

I2C 0 base: 0x4002.0000
Offset 0x004
Type RO, reset 0x0000.0000



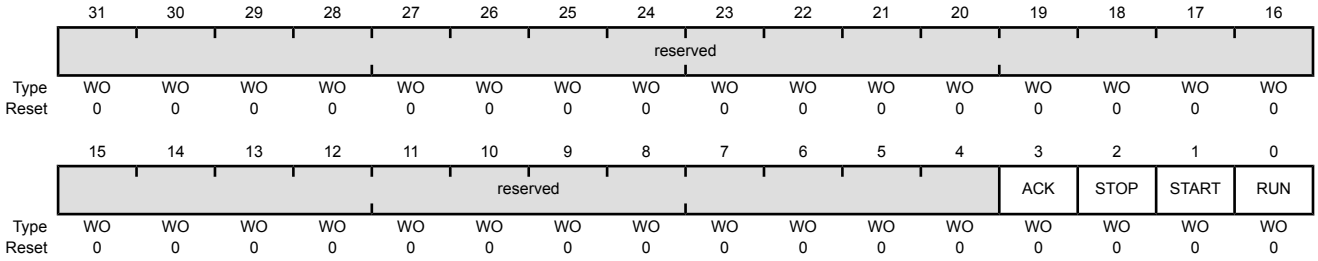

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6 | BUSBSY | RO | 0 | Bus Busy<br>This bit specifies the state of the I2C bus. If set, the bus is busy; otherwise, the bus is idle. The bit changes based on the START and STOP conditions. |
| 5 | IDLE | RO | 0 | I2C Idle<br>This bit specifies the I2C controller state. If set, the controller is idle; otherwise the controller is not idle. |
| 4 | ARBLST | RO | 0 | Arbitration Lost<br>This bit specifies the result of bus arbitration. If set, the controller lost arbitration; otherwise, the controller won arbitration. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 3 | DATACK | RO | 0 | Acknowledge Data<br>This bit specifies the result of the last data operation. If set, the transmitted data was not acknowledged; otherwise, the data was acknowledged. |
| 2 | ADRACK | RO | 0 | Acknowledge Address<br>This bit specifies the result of the last address operation. If set, the transmitted address was not acknowledged; otherwise, the address was acknowledged. |
| 1 | ERROR | RO | 0 | Error<br>This bit specifies the result of the last bus operation. If set, an error occurred on the last operation; otherwise, no error was detected. The error can be from the slave address not being acknowledged or the transmit data not being acknowledged. |
| 0 | BUSY | RO | 0 | $I^2C$ Busy<br>This bit specifies the state of the controller. If set, the controller is busy; otherwise, the controller is idle. When the BUSY bit is set, the other status bits are not valid. |

### Writes

I2C Master Control/Status (I2CMCS)

I2C 0 base: 0x4002.0000
Offset 0x004
Type WO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:4 | reserved | WO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 3 | ACK | WO | 0 | Data Acknowledge Enable<br>When set, causes received data byte to be acknowledged automatically by the master. See field decoding in Table 13-3 on page 448. |
| 2 | STOP | WO | 0 | Generate STOP<br>When set, causes the generation of the STOP condition. See field decoding in Table 13-3 on page 448. |
| 1 | START | WO | 0 | Generate START<br>When set, causes the generation of a START or repeated START condition. See field decoding in Table 13-3 on page 448. |

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 0 | RUN | WO | 0 | I2C Master Enable<br>When set, allows the master to send or receive data. See field decoding in Table 13-3 on page 448. |

**Table 13-3. Write Field Decoding for I2CMCS[3:0] Field (Sheet 1 of 3)**

| Current State | I2CMSA[0] | I2CMCS[3:0] | | | | Description |
|---|---|---|---|---|---|---|
| | R/S | ACK | STOP | START | RUN | |
| Idle | 0 | X[a] | 0 | 1 | 1 | START condition followed by SEND (master goes to the Master Transmit state). |
| | 0 | X | 1 | 1 | 1 | START condition followed by a SEND and STOP condition (master remains in Idle state). |
| | 1 | 0 | 0 | 1 | 1 | START condition followed by RECEIVE operation with negative ACK (master goes to the Master Receive state). |
| | 1 | 0 | 1 | 1 | 1 | START condition followed by RECEIVE and STOP condition (master remains in Idle state). |
| | 1 | 1 | 0 | 1 | 1 | START condition followed by RECEIVE (master goes to the Master Receive state). |
| | 1 | 1 | 1 | 1 | 1 | Illegal. |
| | All other combinations not listed are non-operations. | | | | | NOP. |
| Master Transmit | X | X | 0 | 0 | 1 | SEND operation (master remains in Master Transmit state). |
| | X | X | 1 | 0 | 0 | STOP condition (master goes to Idle state). |
| | X | X | 1 | 0 | 1 | SEND followed by STOP condition (master goes to Idle state). |
| | 0 | X | 0 | 1 | 1 | Repeated START condition followed by a SEND (master remains in Master Transmit state). |
| | 0 | X | 1 | 1 | 1 | Repeated START condition followed by SEND and STOP condition (master goes to Idle state). |
| | 1 | 0 | 0 | 1 | 1 | Repeated START condition followed by a RECEIVE operation with a negative ACK (master goes to Master Receive state). |
| | 1 | 0 | 1 | 1 | 1 | Repeated START condition followed by a SEND and STOP condition (master goes to Idle state). |
| | 1 | 1 | 0 | 1 | 1 | Repeated START condition followed by RECEIVE (master goes to Master Receive state). |
| | 1 | 1 | 1 | 1 | 1 | Illegal. |
| | All other combinations not listed are non-operations. | | | | | NOP. |

**Table 13-3. Write Field Decoding for I2CMCS[3:0] Field (Sheet 1 of 3) *(continued)***

| Current State | I2CMSA[0] | I2CMCS[3:0] | | | | Description |
|---|---|---|---|---|---|---|
| | R/S | ACK | STOP | START | RUN | |
| Master Receive | X | 0 | 0 | 0 | 1 | RECEIVE operation with negative ACK (master remains in Master Receive state). |
| | X | X | 1 | 0 | 0 | STOP condition (master goes to Idle state).[b] |
| | X | 0 | 1 | 0 | 1 | RECEIVE followed by STOP condition (master goes to Idle state). |
| | X | 1 | 0 | 0 | 1 | RECEIVE operation (master remains in Master Receive state). |
| | X | 1 | 1 | 0 | 1 | Illegal. |
| | 1 | 0 | 0 | 1 | 1 | Repeated START condition followed by RECEIVE operation with a negative ACK (master remains in Master Receive state). |
| | 1 | 0 | 1 | 1 | 1 | Repeated START condition followed by RECEIVE and STOP condition (master goes to Idle state). |
| | 1 | 1 | 0 | 1 | 1 | Repeated START condition followed by RECEIVE (master remains in Master Receive state). |
| | 0 | X | 0 | 1 | 1 | Repeated START condition followed by SEND (master goes to Master Transmit state). |
| | 0 | X | 1 | 1 | 1 | Repeated START condition followed by SEND and STOP condition (master goes to Idle state). |
| | All other combinations not listed are non-operations. | | | | | NOP. |

a. An X in a table cell indicates the bit can be 0 or 1.

b. In Master Receive mode, a STOP condition should be generated only after a Data Negative Acknowledge executed by the master or an Address Negative Acknowledge executed by the slave.

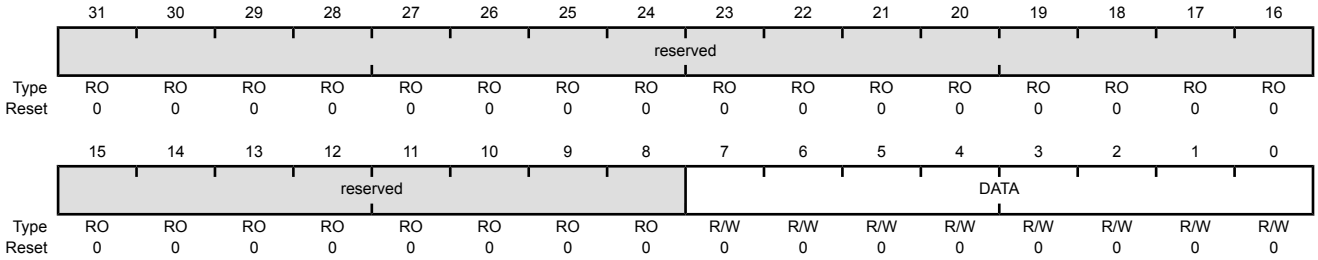
## Register 3: I2C Master Data (I2CMDR), offset 0x008

**Important:** Use caution when reading this register. Performing a read may change bit status.

This register contains the data to be transmitted when in the Master Transmit state, and the data received when in the Master Receive state.

I2C Master Data (I2CMDR)

I2C 0 base: 0x4002.0000
Offset 0x008
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DATA | R/W | 0x00 | Data Transferred<br>Data transferred during transaction. |

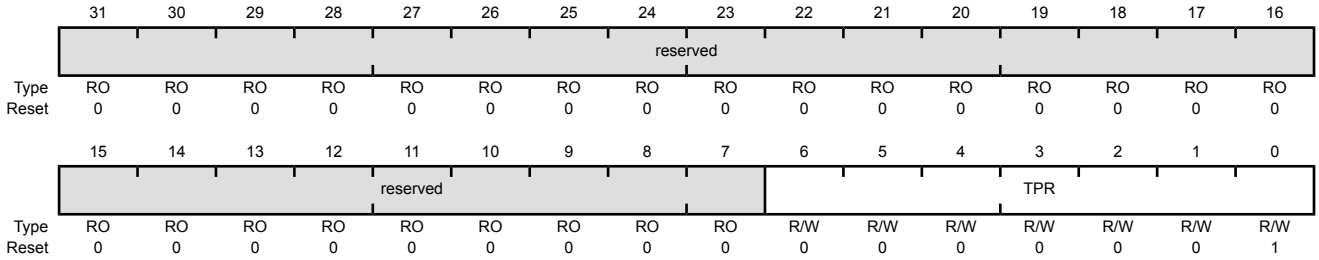
## Register 4: I²C Master Timer Period (I2CMTPR), offset 0x00C

This register specifies the period of the SCL clock.

**Caution – Take care not to set bit 7 when accessing this register as unpredictable behavior can occur.**

I2C Master Timer Period (I2CMTPR)

I2C 0 base: 0x4002.0000
Offset 0x00C
Type R/W, reset 0x0000.0001



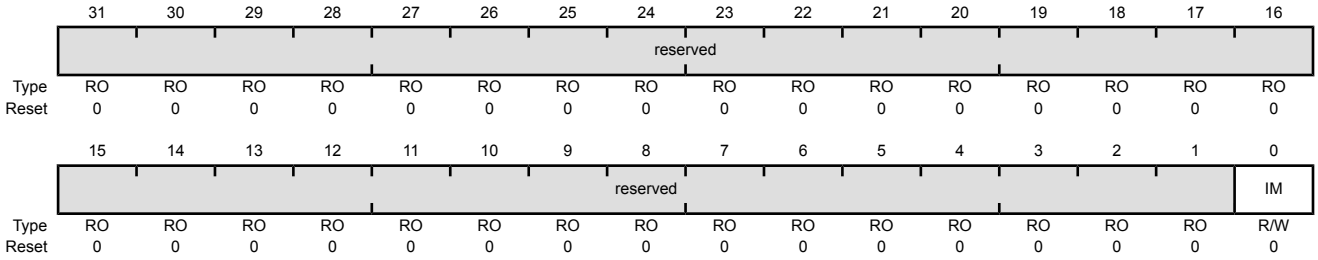

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | TPR | R/W | 0x1 | SCL Clock Period<br>This field specifies the period of the SCL clock.<br>`SCL_PRD = 2*(1 + TPR)*(SCL_LP + SCL_HP)*CLK_PRD`<br>where:<br>SCL_PRD is the SCL line period (I²C clock).<br>TPR is the Timer Period register value (range of 1 to 127).<br>SCL_LP is the SCL Low period (fixed at 6).<br>SCL_HP is the SCL High period (fixed at 4). |

## Register 5: I2C Master Interrupt Mask (I2CMIMR), offset 0x010

This register controls whether a raw interrupt is promoted to a controller interrupt.

I2C Master Interrupt Mask (I2CMIMR)

I2C 0 base: 0x4002.0000
Offset 0x010
Type R/W, reset 0x0000.0000



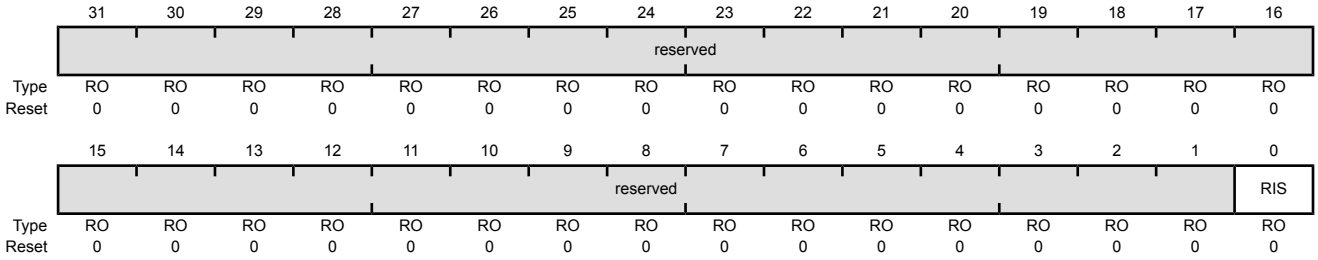

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | IM | R/W | 0 | Interrupt Mask<br>This bit controls whether a raw interrupt is promoted to a controller interrupt. If set, the interrupt is not masked and the interrupt is promoted; otherwise, the interrupt is masked. |

## Register 6: I²C Master Raw Interrupt Status (I2CMRIS), offset 0x014

This register specifies whether an interrupt is pending.

I2C Master Raw Interrupt Status (I2CMRIS)

I2C 0 base: 0x4002.0000
Offset 0x014
Type RO, reset 0x0000.0000



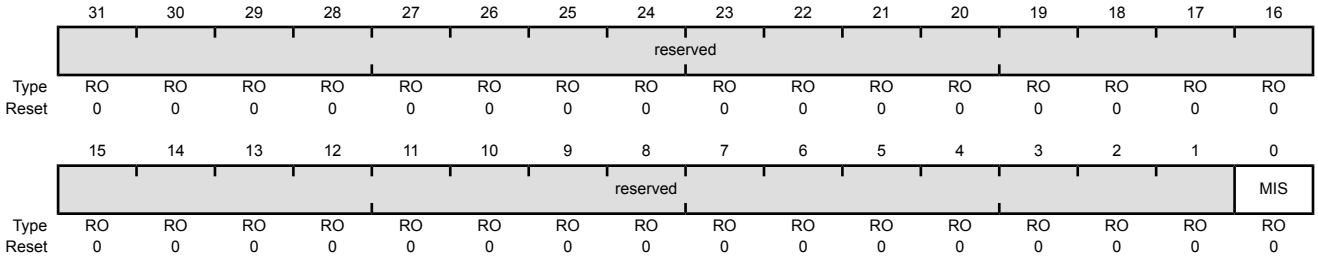

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | RIS | RO | 0 | Raw Interrupt Status<br>This bit specifies the raw interrupt state (prior to masking) of the I²C master block. If set, an interrupt is pending; otherwise, an interrupt is not pending. |

## Register 7: I2C Master Masked Interrupt Status (I2CMMIS), offset 0x018

This register specifies whether an interrupt was signaled.

I2C Master Masked Interrupt Status (I2CMMIS)

I2C 0 base: 0x4002.0000
Offset 0x018
Type RO, reset 0x0000.0000



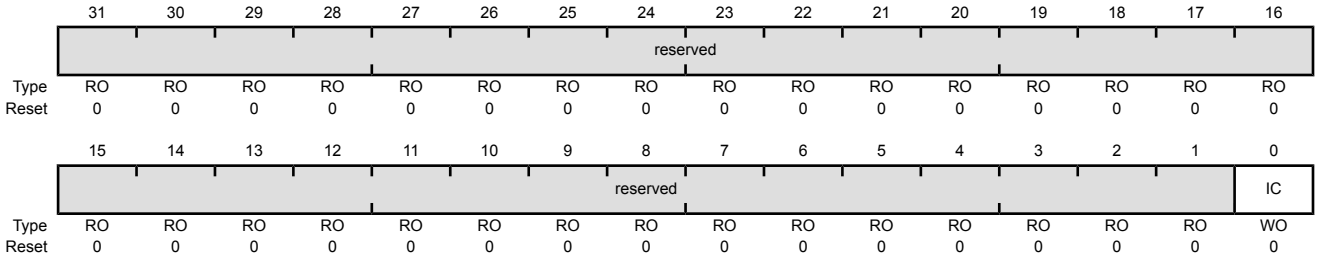

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | MIS | RO | 0 | Masked Interrupt Status<br>This bit specifies the raw interrupt state (after masking) of the I2C master block. If set, an interrupt was signaled; otherwise, an interrupt has not been generated since the bit was last cleared. |

## Register 8: I²C Master Interrupt Clear (I2CMICR), offset 0x01C

This register clears the raw interrupt.

I2C Master Interrupt Clear (I2CMICR)

I2C 0 base: 0x4002.0000
Offset 0x01C
Type WO, reset 0x0000.0000



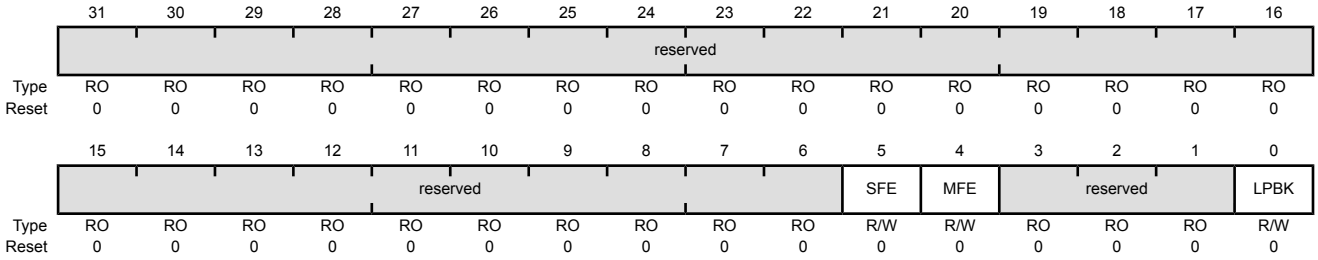

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | IC | WO | 0 | Interrupt Clear<br>This bit controls the clearing of the raw interrupt. A write of 1 clears the interrupt; otherwise, a write of 0 has no affect on the interrupt state. A read of this register returns no meaningful data. |

## Register 9: I2C Master Configuration (I2CMCR), offset 0x020

This register configures the mode (Master or Slave) and sets the interface for test mode loopback.

I2C Master Configuration (I2CMCR)

I2C 0 base: 0x4002.0000
Offset 0x020
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:6 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 5 | SFE | R/W | 0 | I2C Slave Function Enable<br>This bit specifies whether the interface may operate in Slave mode. If set, Slave mode is enabled; otherwise, Slave mode is disabled. |
| 4 | MFE | R/W | 0 | I2C Master Function Enable<br>This bit specifies whether the interface may operate in Master mode. If set, Master mode is enabled; otherwise, Master mode is disabled and the interface clock is disabled. |
| 3:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | LPBK | R/W | 0 | I2C Loopback<br>This bit specifies whether the interface is operating normally or in Loopback mode. If set, the device is put in a test mode loopback configuration; otherwise, the device operates normally. |

## 13.6 Register Descriptions ($I^2C$ Slave)

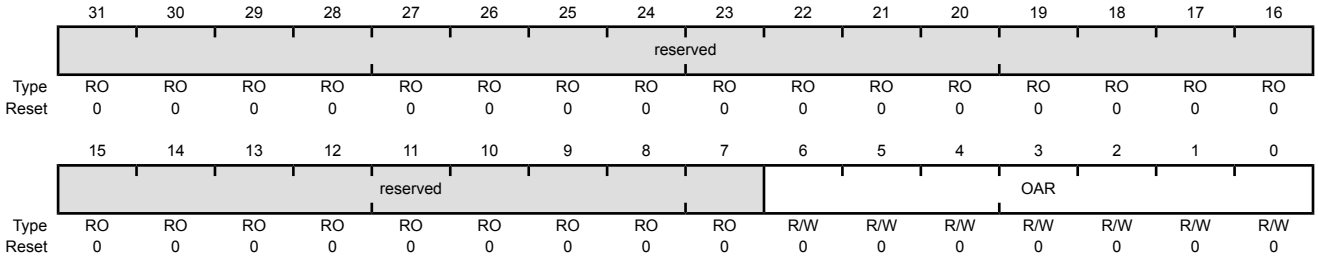
The remainder of this section lists and describes the $I^2C$ slave registers, in numerical order by address offset. See also “Register Descriptions ($I^2C$ Master)” on page 444.

## Register 10: I2C Slave Own Address (I2CSOAR), offset 0x800

This register consists of seven address bits that identify the Stellaris I2C device on the I2C bus.

I2C Slave Own Address (I2CSOAR)

I2C 0 base: 0x4002.0000
Offset 0x800
Type R/W, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:7 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 6:0 | OAR | R/W | 0x00 | I2C Slave Own Address<br>This field specifies bits A6 through A0 of the slave address. |

## Register 11: I²C Slave Control/Status (I2CSCSR), offset 0x804

This register accesses one control bit when written, and three status bits when read.

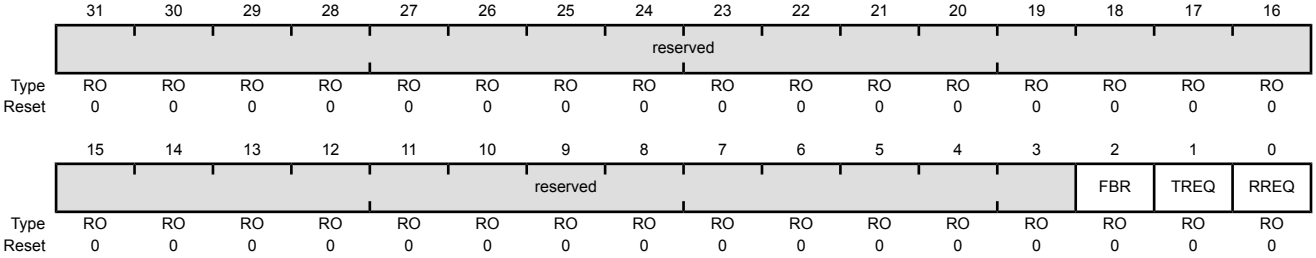
The read-only Status register consists of three bits: the FBR, RREQ, and TREQ bits. The First Byte Received (FBR) bit is set only after the Stellaris device detects its own slave address and receives the first data byte from the I²C master. The Receive Request (RREQ) bit indicates that the Stellaris I²C device has received a data byte from an I²C master. Read one data byte from the **I²C Slave Data (I2CSDR)** register to clear the RREQ bit. The Transmit Request (TREQ) bit indicates that the Stellaris I²C device is addressed as a Slave Transmitter. Write one data byte into the **I²C Slave Data (I2CSDR)** register to clear the TREQ bit.

The write-only Control register consists of one bit: the DA bit. The DA bit enables and disables the Stellaris I²C slave operation.

### Reads

I2C Slave Control/Status (I2CSCSR)

I2C 0 base: 0x4002.0000
Offset 0x804
Type RO, reset 0x0000.0000

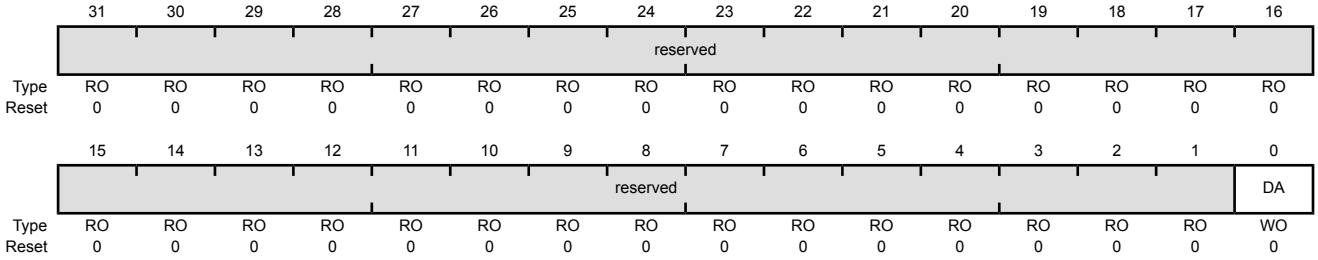



| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:3 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 2 | FBR | RO | 0 | First Byte Received<br>Indicates that the first byte following the slave's own address is received. This bit is only valid when the RREQ bit is set, and is automatically cleared when data has been read from the **I2CSDR** register.<br>**Note:** This bit is not used for slave transmit operations. |
| 1 | TREQ | RO | 0 | Transmit Request<br>This bit specifies the state of the I²C slave with regards to outstanding transmit requests. If set, the I²C unit has been addressed as a slave transmitter and uses clock stretching to delay the master until data has been written to the **I2CSDR** register. Otherwise, there is no outstanding transmit request. |
| 0 | RREQ | RO | 0 | Receive Request<br>This bit specifies the status of the I²C slave with regards to outstanding receive requests. If set, the I²C unit has outstanding receive data from the I²C master and uses clock stretching to delay the master until the data has been read from the **I2CSDR** register. Otherwise, no receive data is outstanding. |

### Writes

I2C Slave Control/Status (I2CSCSR)

I2C 0 base: 0x4002.0000
Offset 0x804
Type WO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | DA | WO | 0 | Device Active<br>Value Description<br>0 Disables the I2C slave operation.<br>1 Enables the I2C slave operation. |

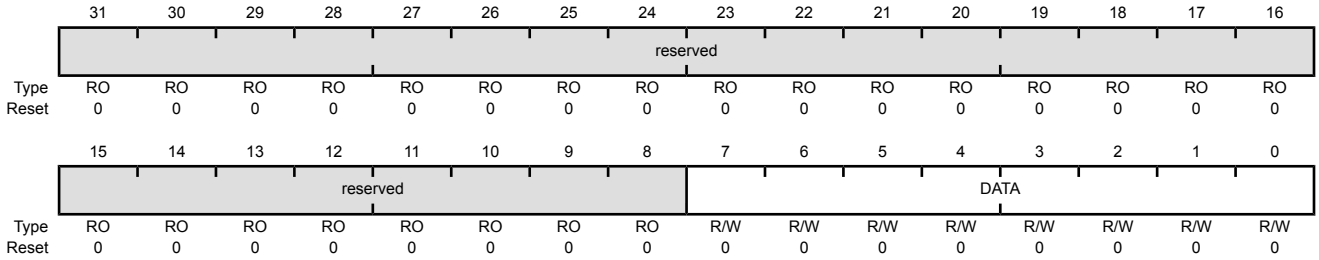
## Register 12: I²C Slave Data (I2CSDR), offset 0x808

**Important:** Use caution when reading this register. Performing a read may change bit status.

This register contains the data to be transmitted when in the Slave Transmit state, and the data received when in the Slave Receive state.

I2C Slave Data (I2CSDR)

I2C 0 base: 0x4002.0000
Offset 0x808
Type R/W, reset 0x0000.0000



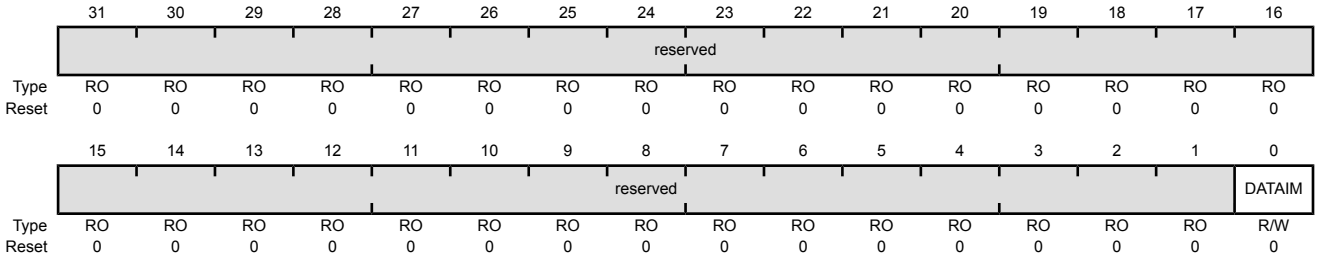

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:8 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 7:0 | DATA | R/W | 0x0 | Data for Transfer<br>This field contains the data for transfer during a slave receive or transmit operation. |

## Register 13: I2C Slave Interrupt Mask (I2CSIMR), offset 0x80C

This register controls whether a raw interrupt is promoted to a controller interrupt.

I2C Slave Interrupt Mask (I2CSIMR)

I2C 0 base: 0x4002.0000
Offset 0x80C
Type R/W, reset 0x0000.0000



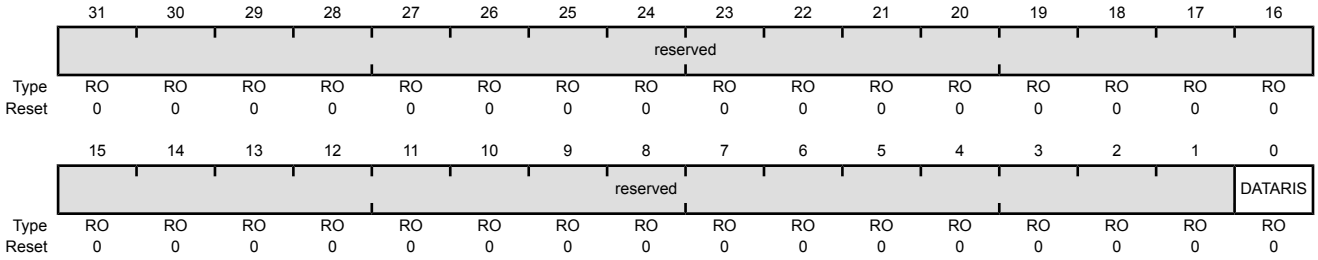

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | DATAIM | R/W | 0 | Data Interrupt Mask<br>This bit controls whether the raw interrupt for data received and data requested is promoted to a controller interrupt. If set, the interrupt is not masked and the interrupt is promoted; otherwise, the interrupt is masked. |

## Register 14: I²C Slave Raw Interrupt Status (I2CSRIS), offset 0x810

This register specifies whether an interrupt is pending.

I2C Slave Raw Interrupt Status (I2CSRIS)

I2C 0 base: 0x4002.0000
Offset 0x810
Type RO, reset 0x0000.0000



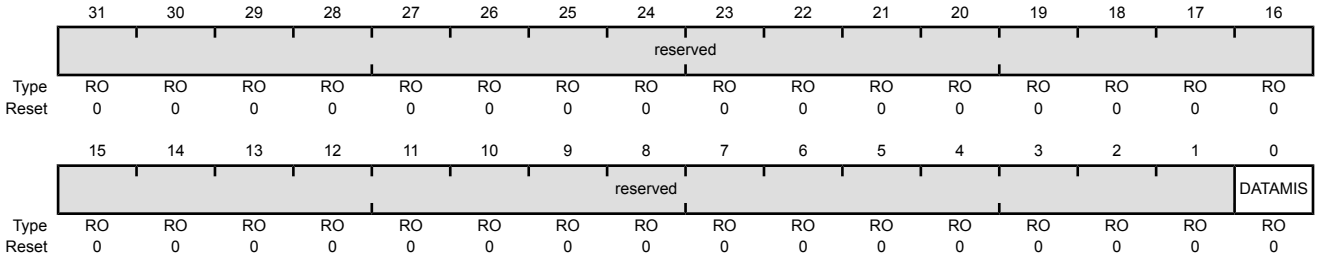

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | DATARIS | RO | 0 | Data Raw Interrupt Status<br>This bit specifies the raw interrupt state for data received and data requested (prior to masking) of the I²C slave block. If set, an interrupt is pending; otherwise, an interrupt is not pending. |

## Register 15: I2C Slave Masked Interrupt Status (I2CSMIS), offset 0x814

This register specifies whether an interrupt was signaled.

I2C Slave Masked Interrupt Status (I2CSMIS)

I2C 0 base: 0x4002.0000
Offset 0x814
Type RO, reset 0x0000.0000



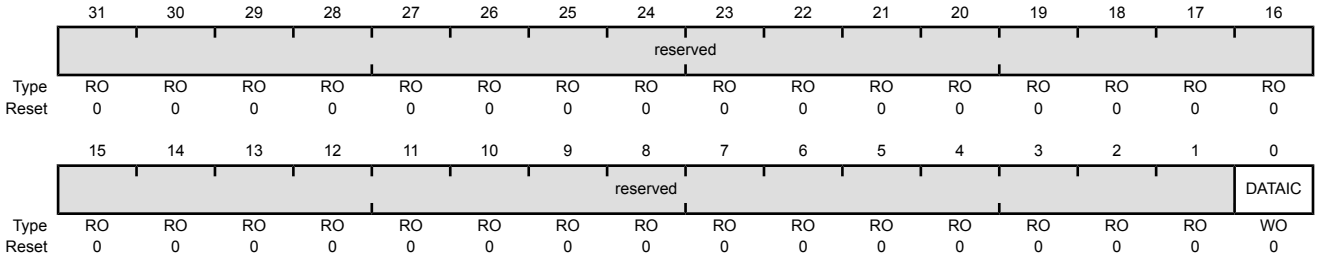

| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | DATAMIS | RO | 0 | Data Masked Interrupt Status<br>This bit specifies the interrupt state for data received and data requested (after masking) of the I2C slave block. If set, an interrupt was signaled; otherwise, an interrupt has not been generated since the bit was last cleared. |

## Register 16: I²C Slave Interrupt Clear (I2CSICR), offset 0x818

This register clears the raw interrupt. A read of this register returns no meaningful data.

I2C Slave Interrupt Clear (I2CSICR)

I2C 0 base: 0x4002.0000
Offset 0x818
Type WO, reset 0x0000.0000




| Bit/Field | Name | Type | Reset | Description |
|---|---|---|---|---|
| 31:1 | reserved | RO | 0x00 | Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation. |
| 0 | DATAIC | WO | 0 | Data Interrupt Clear<br>This bit controls the clearing of the raw interrupt for data received and data requested. When set, it clears the DATARIS interrupt bit; otherwise, it has no effect on the DATARIS bit value. |

# 14 Pin Diagram

The LM3S828 microcontroller pin diagrams are shown below.

**Figure 14-1. 48-Pin QFP Package Pin Diagram**

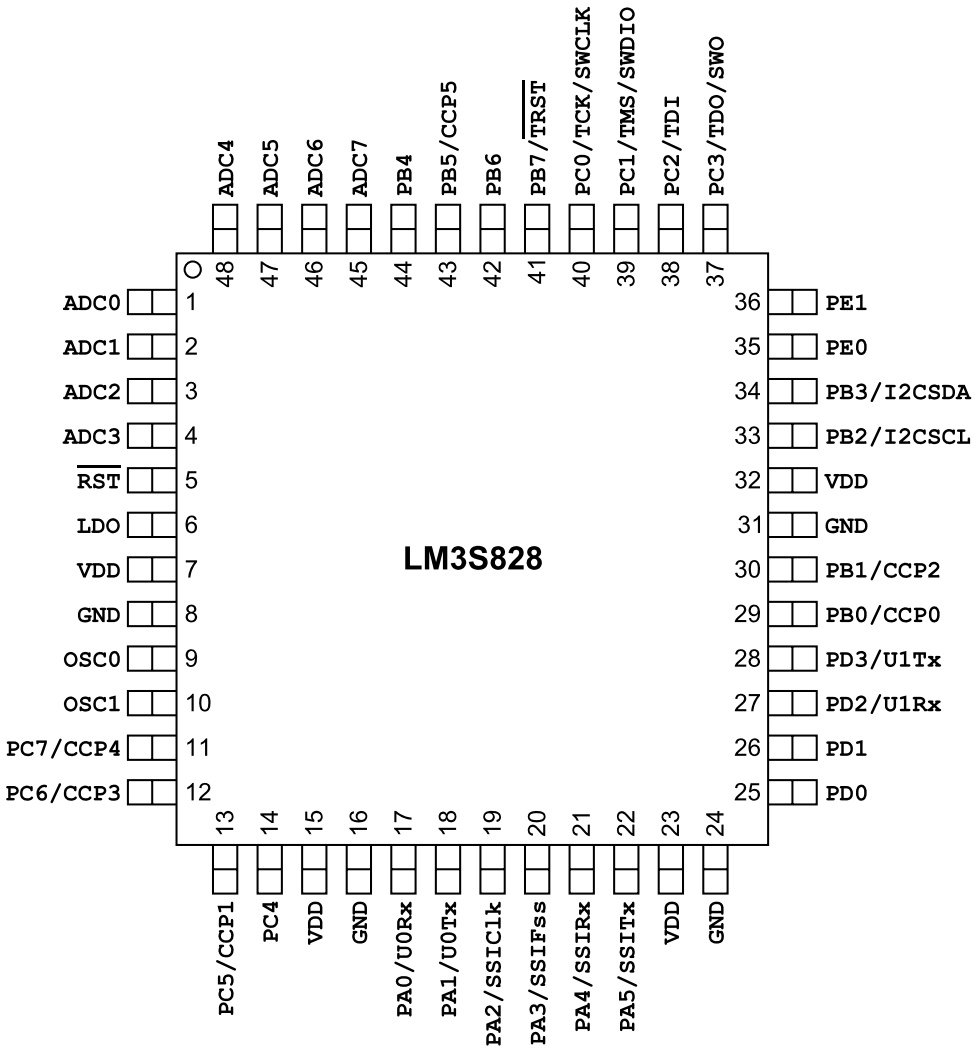
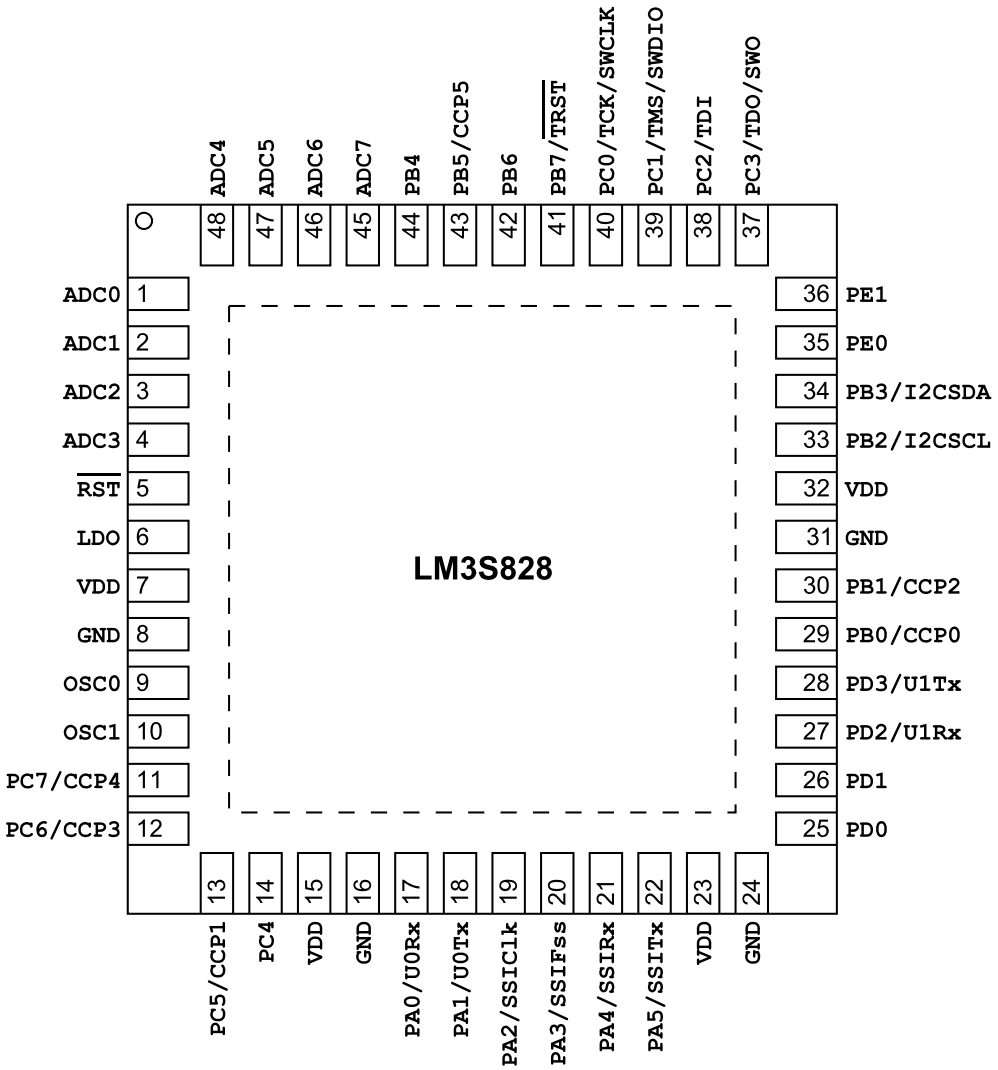


**Figure 14-2. 48-Pin QFN Package Pin Diagram**[1]




[1]The thermal pad must be connected to GND.

# 15 Signal Tables

The following tables list the signals available for each pin. Functionality is enabled by software with the **GPIOAFSEL** register.

**Important:** All multiplexed pins are GPIOs by default, with the exception of the five JTAG pins (PB7 and PC[3:0]) which default to the JTAG functionality.

Table 15-1 on page 468 shows the pin-to-signal-name mapping, including functional characteristics of the signals. Table 15-2 on page 470 lists the signals in alphabetical order by signal name.

Table 15-3 on page 472 groups the signals by functionality, except for GPIOs. Table 15-4 on page 473 lists the GPIO pins and their alternate functionality.

**Note:** All digital inputs are Schmitt triggered.

**Table 15-1. Signals by Pin Number**

| Pin Number | Pin Name | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| 1 | ADC0 | I | Analog | Analog-to-digital converter input 0. |
| 2 | ADC1 | I | Analog | Analog-to-digital converter input 1. |
| 3 | ADC2 | I | Analog | Analog-to-digital converter input 2. |
| 4 | ADC3 | I | Analog | Analog-to-digital converter input 3. |
| 5 | $\overline{\text{RST}}$ | I | TTL | System reset input. |
| 6 | LDO | - | Power | Low drop-out regulator output voltage. This pin requires an external capacitor between the pin and GND of 1 µF or greater. |
| 7 | VDD | - | Power | Positive supply for I/O and some logic. |
| 8 | GND | - | Power | Ground reference for logic and I/O pins. |
| 9 | OSC0 | I | Analog | Main oscillator crystal input or an external clock reference input. |
| 10 | OSC1 | O | Analog | Main oscillator crystal output. Leave unconnected when using a single-ended clock source. |
| 11 | PC7 | I/O | TTL | GPIO port C bit 7. |
| | CCP4 | I/O | TTL | Capture/Compare/PWM 4. |
| 12 | PC6 | I/O | TTL | GPIO port C bit 6. |
| | CCP3 | I/O | TTL | Capture/Compare/PWM 3. |
| 13 | PC5 | I/O | TTL | GPIO port C bit 5. |
| | CCP1 | I/O | TTL | Capture/Compare/PWM 1. |
| 14 | PC4 | I/O | TTL | GPIO port C bit 4. |
| 15 | VDD | - | Power | Positive supply for I/O and some logic. |
| 16 | GND | - | Power | Ground reference for logic and I/O pins. |
| 17 | PA0 | I/O | TTL | GPIO port A bit 0. |
| | U0Rx | I | TTL | UART module 0 receive. |
| 18 | PA1 | I/O | TTL | GPIO port A bit 1. |
| | U0Tx | O | TTL | UART module 0 transmit. |
| 19 | PA2 | I/O | TTL | GPIO port A bit 2. |
| | SSIClk | I/O | TTL | SSI clock. |
| 20 | PA3 | I/O | TTL | GPIO port A bit 3. |
| | SSIFss | I/O | TTL | SSI frame. |

**Table 15-1. Signals by Pin Number *(continued)***

| Pin Number | Pin Name | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| 21 | PA4 | I/O | TTL | GPIO port A bit 4. |
| | SSIRx | I | TTL | SSI receive. |
| 22 | PA5 | I/O | TTL | GPIO port A bit 5. |
| | SSITx | O | TTL | SSI transmit. |
| 23 | VDD | - | Power | Positive supply for I/O and some logic. |
| 24 | GND | - | Power | Ground reference for logic and I/O pins. |
| 25 | PD0 | I/O | TTL | GPIO port D bit 0. |
| 26 | PD1 | I/O | TTL | GPIO port D bit 1. |
| 27 | PD2 | I/O | TTL | GPIO port D bit 2. |
| | U1Rx | I | TTL | UART module 1 receive. |
| 28 | PD3 | I/O | TTL | GPIO port D bit 3. |
| | U1Tx | O | TTL | UART module 1 transmit. |
| 29 | PB0 | I/O | TTL | GPIO port B bit 0. |
| | CCP0 | I/O | TTL | Capture/Compare/PWM 0. |
| 30 | PB1 | I/O | TTL | GPIO port B bit 1. |
| | CCP2 | I/O | TTL | Capture/Compare/PWM 2. |
| 31 | GND | - | Power | Ground reference for logic and I/O pins. |
| 32 | VDD | - | Power | Positive supply for I/O and some logic. |
| 33 | PB2 | I/O | TTL | GPIO port B bit 2. |
| | I2CSCL | I/O | OD | $I^2C$ clock. |
| 34 | PB3 | I/O | TTL | GPIO port B bit 3. |
| | I2CSDA | I/O | OD | $I^2C$ data. |
| 35 | PE0 | I/O | TTL | GPIO port E bit 0. |
| 36 | PE1 | I/O | TTL | GPIO port E bit 1. |
| 37 | PC3 | I/O | TTL | GPIO port C bit 3. |
| | SWO | O | TTL | JTAG TDO and SWO. |
| | TDO | O | TTL | JTAG TDO and SWO. |
| 38 | PC2 | I/O | TTL | GPIO port C bit 2. |
| | TDI | I | TTL | JTAG TDI. |
| 39 | PC1 | I/O | TTL | GPIO port C bit 1. |
| | SWDIO | I/O | TTL | JTAG TMS and SWDIO. |
| | TMS | I/O | TTL | JTAG TMS and SWDIO. |
| 40 | PC0 | I/O | TTL | GPIO port C bit 0. |
| | SWCLK | I | TTL | JTAG/SWD CLK. |
| | TCK | I | TTL | JTAG/SWD CLK. |
| 41 | PB7 | I/O | TTL | GPIO port B bit 7. |
| | $\overline{\text{TRST}}$ | I | TTL | JTAG $\overline{\text{TRST}}$. |
| 42 | PB6 | I/O | TTL | GPIO port B bit 6. |
| 43 | PB5 | I/O | TTL | GPIO port B bit 5. |
| | CCP5 | I/O | TTL | Capture/Compare/PWM 5. |
| 44 | PB4 | I/O | TTL | GPIO port B bit 4. |

### Table 15-1. Signals by Pin Number *(continued)*

| Pin Number | Pin Name | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| 45 | ADC7 | I | Analog | Analog-to-digital converter input 7. |
| 46 | ADC6 | I | Analog | Analog-to-digital converter input 6. |
| 47 | ADC5 | I | Analog | Analog-to-digital converter input 5. |
| 48 | ADC4 | I | Analog | Analog-to-digital converter input 4. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

### Table 15-2. Signals by Signal Name

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| ADC0 | 1 | I | Analog | Analog-to-digital converter input 0. |
| ADC1 | 2 | I | Analog | Analog-to-digital converter input 1. |
| ADC2 | 3 | I | Analog | Analog-to-digital converter input 2. |
| ADC3 | 4 | I | Analog | Analog-to-digital converter input 3. |
| ADC4 | 48 | I | Analog | Analog-to-digital converter input 4. |
| ADC5 | 47 | I | Analog | Analog-to-digital converter input 5. |
| ADC6 | 46 | I | Analog | Analog-to-digital converter input 6. |
| ADC7 | 45 | I | Analog | Analog-to-digital converter input 7. |
| CCP0 | 29 | I/O | TTL | Capture/Compare/PWM 0. |
| CCP1 | 13 | I/O | TTL | Capture/Compare/PWM 1. |
| CCP2 | 30 | I/O | TTL | Capture/Compare/PWM 2. |
| CCP3 | 12 | I/O | TTL | Capture/Compare/PWM 3. |
| CCP4 | 11 | I/O | TTL | Capture/Compare/PWM 4. |
| CCP5 | 43 | I/O | TTL | Capture/Compare/PWM 5. |
| GND | 8<br>16<br>24<br>31 | - | Power | Ground reference for logic and I/O pins. |
| I2CSCL | 33 | I/O | OD | $I^2C$ clock. |
| I2CSDA | 34 | I/O | OD | $I^2C$ data. |
| LDO | 6 | - | Power | Low drop-out regulator output voltage. This pin requires an external capacitor between the pin and GND of 1 µF or greater. |
| OSC0 | 9 | I | Analog | Main oscillator crystal input or an external clock reference input. |
| OSC1 | 10 | O | Analog | Main oscillator crystal output. Leave unconnected when using a single-ended clock source. |
| PA0 | 17 | I/O | TTL | GPIO port A bit 0. |
| PA1 | 18 | I/O | TTL | GPIO port A bit 1. |
| PA2 | 19 | I/O | TTL | GPIO port A bit 2. |
| PA3 | 20 | I/O | TTL | GPIO port A bit 3. |
| PA4 | 21 | I/O | TTL | GPIO port A bit 4. |
| PA5 | 22 | I/O | TTL | GPIO port A bit 5. |
| PB0 | 29 | I/O | TTL | GPIO port B bit 0. |
| PB1 | 30 | I/O | TTL | GPIO port B bit 1. |

**Table 15-2. Signals by Signal Name *(continued)***

| Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|
| PB2 | 33 | I/O | TTL | GPIO port B bit 2. |
| PB3 | 34 | I/O | TTL | GPIO port B bit 3. |
| PB4 | 44 | I/O | TTL | GPIO port B bit 4. |
| PB5 | 43 | I/O | TTL | GPIO port B bit 5. |
| PB6 | 42 | I/O | TTL | GPIO port B bit 6. |
| PB7 | 41 | I/O | TTL | GPIO port B bit 7. |
| PC0 | 40 | I/O | TTL | GPIO port C bit 0. |
| PC1 | 39 | I/O | TTL | GPIO port C bit 1. |
| PC2 | 38 | I/O | TTL | GPIO port C bit 2. |
| PC3 | 37 | I/O | TTL | GPIO port C bit 3. |
| PC4 | 14 | I/O | TTL | GPIO port C bit 4. |
| PC5 | 13 | I/O | TTL | GPIO port C bit 5. |
| PC6 | 12 | I/O | TTL | GPIO port C bit 6. |
| PC7 | 11 | I/O | TTL | GPIO port C bit 7. |
| PD0 | 25 | I/O | TTL | GPIO port D bit 0. |
| PD1 | 26 | I/O | TTL | GPIO port D bit 1. |
| PD2 | 27 | I/O | TTL | GPIO port D bit 2. |
| PD3 | 28 | I/O | TTL | GPIO port D bit 3. |
| PE0 | 35 | I/O | TTL | GPIO port E bit 0. |
| PE1 | 36 | I/O | TTL | GPIO port E bit 1. |
| RST | 5 | I | TTL | System reset input. |
| SSIClk | 19 | I/O | TTL | SSI clock. |
| SSIFss | 20 | I/O | TTL | SSI frame. |
| SSIRx | 21 | I | TTL | SSI receive. |
| SSITx | 22 | O | TTL | SSI transmit. |
| SWCLK | 40 | I | TTL | JTAG/SWD CLK. |
| SWDIO | 39 | I/O | TTL | JTAG TMS and SWDIO. |
| SWO | 37 | O | TTL | JTAG TDO and SWO. |
| TCK | 40 | I | TTL | JTAG/SWD CLK. |
| TDI | 38 | I | TTL | JTAG TDI. |
| TDO | 37 | O | TTL | JTAG TDO and SWO. |
| TMS | 39 | I/O | TTL | JTAG TMS and SWDIO. |
| TRST | 41 | I | TTL | JTAG TRST. |
| U0Rx | 17 | I | TTL | UART module 0 receive. |
| U0Tx | 18 | O | TTL | UART module 0 transmit. |
| U1Rx | 27 | I | TTL | UART module 1 receive. |
| U1Tx | 28 | O | TTL | UART module 1 transmit. |
| VDD | 7<br>15<br>23<br>32 | - | Power | Positive supply for I/O and some logic. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

### Table 15-3. Signals by Function, Except for GPIO

| Function | Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| ADC | ADC0 | 1 | I | Analog | Analog-to-digital converter input 0. |
| | ADC1 | 2 | I | Analog | Analog-to-digital converter input 1. |
| | ADC2 | 3 | I | Analog | Analog-to-digital converter input 2. |
| | ADC3 | 4 | I | Analog | Analog-to-digital converter input 3. |
| | ADC4 | 48 | I | Analog | Analog-to-digital converter input 4. |
| | ADC5 | 47 | I | Analog | Analog-to-digital converter input 5. |
| | ADC6 | 46 | I | Analog | Analog-to-digital converter input 6. |
| | ADC7 | 45 | I | Analog | Analog-to-digital converter input 7. |
| General-Purpose Timers | CCP0 | 29 | I/O | TTL | Capture/Compare/PWM 0. |
| | CCP1 | 13 | I/O | TTL | Capture/Compare/PWM 1. |
| | CCP2 | 30 | I/O | TTL | Capture/Compare/PWM 2. |
| | CCP3 | 12 | I/O | TTL | Capture/Compare/PWM 3. |
| | CCP4 | 11 | I/O | TTL | Capture/Compare/PWM 4. |
| | CCP5 | 43 | I/O | TTL | Capture/Compare/PWM 5. |
| I2C | I2CSCL | 33 | I/O | OD | $I^2C$ clock. |
| | I2CSDA | 34 | I/O | OD | $I^2C$ data. |
| JTAG/SWD/SWO | SWCLK | 40 | I | TTL | JTAG/SWD CLK. |
| | SWDIO | 39 | I/O | TTL | JTAG TMS and SWDIO. |
| | SWO | 37 | O | TTL | JTAG TDO and SWO. |
| | TCK | 40 | I | TTL | JTAG/SWD CLK. |
| | TDI | 38 | I | TTL | JTAG TDI. |
| | TDO | 37 | O | TTL | JTAG TDO and SWO. |
| | TMS | 39 | I/O | TTL | JTAG TMS and SWDIO. |
| | $\overline{\text{TRST}}$ | 41 | I | TTL | JTAG $\overline{\text{TRST}}$. |
| Power | GND | 8<br>16<br>24<br>31 | - | Power | Ground reference for logic and I/O pins. |
| | LDO | 6 | - | Power | Low drop-out regulator output voltage. This pin requires an external capacitor between the pin and GND of 1 µF or greater. |
| | VDD | 7<br>15<br>23<br>32 | - | Power | Positive supply for I/O and some logic. |
| SSI | SSIClk | 19 | I/O | TTL | SSI clock. |
| | SSIFss | 20 | I/O | TTL | SSI frame. |
| | SSIRx | 21 | I | TTL | SSI receive. |
| | SSITx | 22 | O | TTL | SSI transmit. |
| System Control & Clocks | OSC0 | 9 | I | Analog | Main oscillator crystal input or an external clock reference input. |
| | OSC1 | 10 | O | Analog | Main oscillator crystal output. Leave unconnected when using a single-ended clock source. |
| | $\overline{\text{RST}}$ | 5 | I | TTL | System reset input. |

**Table 15-3. Signals by Function, Except for GPIO *(continued)***

| Function | Pin Name | Pin Number | Pin Type | Buffer Type[a] | Description |
|---|---|---|---|---|---|
| UART | U0Rx | 17 | I | TTL | UART module 0 receive. |
| | U0Tx | 18 | O | TTL | UART module 0 transmit. |
| | U1Rx | 27 | I | TTL | UART module 1 receive. |
| | U1Tx | 28 | O | TTL | UART module 1 transmit. |

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

**Table 15-4. GPIO Pins and Alternate Functions**

| IO | Pin Number | Multiplexed Function | Multiplexed Function |
|---|---|---|---|
| PA0 | 17 | U0Rx | |
| PA1 | 18 | U0Tx | |
| PA2 | 19 | SSIClk | |
| PA3 | 20 | SSIFss | |
| PA4 | 21 | SSIRx | |
| PA5 | 22 | SSITx | |
| PB0 | 29 | CCP0 | |
| PB1 | 30 | CCP2 | |
| PB2 | 33 | I2CSCL | |
| PB3 | 34 | I2CSDA | |
| PB4 | 44 | | |
| PB5 | 43 | CCP5 | |
| PB6 | 42 | | |
| PB7 | 41 | TRST | |
| PC0 | 40 | TCK | SWCLK |
| PC1 | 39 | TMS | SWDIO |
| PC2 | 38 | TDI | |
| PC3 | 37 | TDO | SWO |
| PC4 | 14 | | |
| PC5 | 13 | CCP1 | |
| PC6 | 12 | CCP3 | |
| PC7 | 11 | CCP4 | |
| PD0 | 25 | | |
| PD1 | 26 | | |
| PD2 | 27 | U1Rx | |
| PD3 | 28 | U1Tx | |
| PE0 | 35 | | |
| PE1 | 36 | | |

## 15.1 Connections for Unused Signals

Table 15-5 on page 474 show how to handle signals for functions that are not used in a particular system implementation. Two options are shown in the table: an acceptable practice and a preferred practice for reduced power consumption and improved EMC characteristics.

### Table 15-5. Connections for Unused Signals

| Function | Signal Name | Pin Number | Acceptable Practice | Preferred Practice |
|---|---|---|---|---|
| GPIO | All unused GPIOs | - | NC | GND |
| System Control | OSC0 | L11 | NC | GND |
| | OSC1 | M11 | NC | NC |
| | $\overline{\text{RST}}$ | H11 | Pull up as shown in Figure 5-1 on page 151 | Connect through a capacitor to GND as close to pin as possible |

# 16 Operating Characteristics

### Table 16-1. Temperature Characteristics

| Characteristic | Symbol | Value | Unit |
|---|---|---|---|
| Industrial operating temperature range | $T_A$ | -40 to +85 | °C |
| Extended operating temperature range | $T_A$ | -40 to +105 | °C |
| Unpowered storage temperature range | $T_S$ | -65 to +150 | °C |

### Table 16-2. Thermal Characteristics

| Characteristic | Symbol | Value | Unit |
|---|---|---|---|
| Thermal resistance (junction to ambient)[a] | $\Theta_{JA}$ | 50 (48-pin QFP)<br>26 (48-pin QFN) | °C/W |
| Junction temperature[b] | $T_J$ | $T_A + (P \bullet \Theta_{JA})$ | °C |
| Maximum junction temperature | $T_{JMAX}$ | 115 [c] | °C |

a. Junction to ambient thermal resistance $\theta_{JA}$ numbers are determined by a package simulator.

b. Power dissipation is a function of temperature.

c. $T_{JMAX}$ calculation is based on power consumption values and conditions as specified in “Power Specifications” on page 383 of the data sheet.

### Table 16-3. ESD Absolute Maximum Ratings[a]

| Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|
| $V_{ESDHBM}$ | - | - | 2.0 | kV |
| $V_{ESDCDM}$ | - | - | 1.0 | kV |
| $V_{ESDMM}$ | - | - | 100 | V |

a. All Stellaris parts are ESD tested following the JEDEC standard.

# 17 Electrical Characteristics

## 17.1 DC Characteristics

### 17.1.1 Maximum Ratings

The maximum ratings are the limits to which the device can be subjected without permanently damaging the device.

**Note:** The device is not guaranteed to operate properly at the maximum ratings.

**Table 17-1. Maximum Ratings**

| Characteristic[a] | Symbol | Value | Unit |
|---|---|---|---|
| Supply voltage range ($V_{DD}$) | $V_{DD}$ | 0.0 to +3.6 | V |
| Input voltage | $V_{IN}$ | -0.3 to 5.5 | V |
| Maximum current for pins, excluding pins operating as GPIOs | I | 100 | mA |
| Maximum current for GPIO pins | I | 100 | mA |
| Maximum input voltage on a non-power pin when the microcontroller is unpowered | $V_{NON}$ | 300 | mV |

a. Voltages are measured with respect to GND.

**Important:** This device contains circuitry to protect the inputs against damage due to high-static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are connected to an appropriate logic voltage level (for example, either GND or $V_{DD}$).

### 17.1.2 Recommended DC Operating Conditions

**Table 17-2. Recommended DC Operating Conditions**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{DD}$ | Supply voltage | 3.0 | 3.3 | 3.6 | V |
| $V_{IH}$ | High-level input voltage | 2.0 | - | 5.0 | V |
| $V_{IL}$ | Low-level input voltage | -0.3 | - | 1.3 | V |
| $V_{OH}$ | High-level output voltage | 2.4 | - | - | V |
| $V_{OL}$ | Low-level output voltage | - | - | 0.4 | V |
| $I_{OH}$ | High-level source current, $V_{OH}$=2.4 V | | | | |
| | 2-mA Drive | 2.0 | - | - | mA |
| | 4-mA Drive | 4.0 | - | - | mA |
| | 8-mA Drive | 8.0 | - | - | mA |
| $I_{OL}$ | Low-level sink current, $V_{OL}$=0.4 V | | | | |
| | 2-mA Drive | 2.0 | - | - | mA |
| | 4-mA Drive | 4.0 | - | - | mA |
| | 8-mA Drive | 8.0 | - | - | mA |

### 17.1.3 On-Chip Low Drop-Out (LDO) Regulator Characteristics

**Table 17-3. LDO Regulator Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{LDOOUT}$ | Programmable internal (logic) power supply output value | 2.25 | - | 2.75 | V |
| | Output voltage accuracy | - | 2% | - | % |
| $t_{PON}$ | Power-on time | - | - | 100 | µs |
| $t_{ON}$ | Time on | - | - | 200 | µs |
| $t_{OFF}$ | Time off | - | - | 100 | µs |
| $V_{STEP}$ | Step programming incremental voltage | - | 50 | - | mV |
| $C_{LDO}$ | External filter capacitor size for internal power supply | 1.0 | - | 3.0 | µF |

### 17.1.4 GPIO Module Characteristics

**Table 17-4. GPIO Module DC Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $R_{GPIOPU}$ | GPIO internal pull-up resistor | 50 | - | 110 | kΩ |
| $R_{GPIOPD}$ | GPIO internal pull-down resistor | 55 | - | 180 | kΩ |
| $I_{LKG}$ | GPIO input leakage current[a] | - | - | 2 | µA |

a. The leakage current is measured with GND or $V_{DD}$ applied to the corresponding pin(s). The leakage of digital port pins is measured individually. The port pin is configured as an input and the pullup/pulldown resistor is disabled.

### 17.1.5 Power Specifications

The power measurements specified in the tables that follow are run on the core processor using SRAM with the following specifications (except as noted):

- $V_{DD}$ = 3.3 V
- Temperature = 25°C

**Table 17-5. Detailed Power Specifications**

| Parameter | Parameter Name | Conditions | Nom | Max | Unit |
|---|---|---|---|---|---|
| $I_{DD\_RUN}$ | Run mode 1 (Flash loop) | LDO = 2.50 V<br>Code = while(1){} executed in Flash<br>Peripherals = All clock-gated ON<br>System Clock = 50 MHz (with PLL) | 95 | 110 | mA |
| | Run mode 2 (Flash loop) | LDO = 2.50 V<br>Code = while(1){} executed in Flash<br>Peripherals = All clock-gated OFF<br>System Clock = 50 MHz (with PLL) | 60 | 75 | mA |
| | Run mode 1 (SRAM loop) | LDO = 2.50 V<br>Code = while(1){} executed in SRAM<br>Peripherals = All clock-gated ON<br>System Clock = 50 MHz (with PLL) | 85 | 95 | mA |
| | Run mode 2 (SRAM loop) | LDO = 2.50 V<br>Code = while(1){} executed in SRAM<br>Peripherals = All clock-gated OFF<br>System Clock = 50 MHz (with PLL) | 50 | 60 | mA |
| $I_{DD\_SLEEP}$ | Sleep mode | LDO = 2.50 V<br>Peripherals = All clock-gated OFF<br>System Clock = 50 MHz (with PLL) | 19 | 22 | mA |
| $I_{DD\_DEEPSLEEP}$ | Deep-Sleep mode | LDO = 2.25 V<br>Peripherals = All OFF<br>System Clock = MOSC/16 | 950 | 1150 | µA |

### 17.1.6 Flash Memory Characteristics

**Table 17-6. Flash Memory Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $PE_{CYC}$ | Number of guaranteed program/erase cycles before failure[a] | 10,000 | 100,000 | - | cycles |
| $T_{RET}$ | Data retention at average operating temperature of 85˚C (industrial) or 105˚C (extended) | 10 | - | - | years |
| $T_{PROG}$ | Word program time | 20 | - | - | µs |
| $T_{ERASE}$ | Page erase time | 20 | - | - | ms |
| $T_{ME}$ | Mass erase time | - | - | 250 | ms |

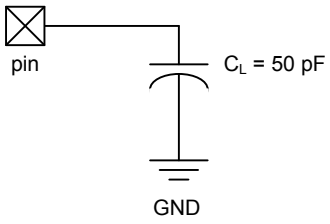
a. A program/erase cycle is defined as switching the bits from 1-> 0 -> 1.

## 17.2 AC Characteristics

### 17.2.1 Load Conditions

Unless otherwise specified, the following conditions are true for all timing measurements. Timing measurements are for 4-mA drive strength.

**Figure 17-1. Load Conditions**




## 17.2.2 Clocks

**Table 17-7. Phase Locked Loop (PLL) Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $f_{ref\_crystal}$ | Crystal reference[a] | 3.579545 | - | 8.192 | MHz |
| $f_{ref\_ext}$ | External clock reference[a] | 3.579545 | - | 8.192 | MHz |
| $f_{pll}$ | PLL frequency[b] | - | 200 | - | MHz |
| $T_{READY}$ | PLL lock time | - | - | 0.5 | ms |

a. The exact value is determined by the crystal value programmed into the XTAL field of the **Run-Mode Clock Configuration (RCC)** register.

b. PLL frequency is automatically calculated by the hardware based on the XTAL field of the **RCC** register.

**Table 17-8. Clock Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $f_{IOSC}$ | Internal oscillator frequency | 7 | 12 | 22 | MHz |
| $f_{MOSC}$ | Main oscillator frequency | 1 | - | 8 | MHz |
| $t_{MOSC\_per}$ | Main oscillator period | 125 | - | 1000 | ns |
| $f_{ref\_crystal\_bypass}$ | Crystal reference using the main oscillator (PLL in BYPASS mode)[a] | 1 | - | 8 | MHz |
| $f_{ref\_ext\_bypass}$ | External clock reference (PLL in BYPASS mode)[a] | 0 | - | 50 | MHz |
| $f_{system\_clock}$ | System clock | 0 | - | 50 | MHz |

a. The ADC must be clocked from the PLL or directly from a 16.667-MHz clock source to operate properly.

### 17.2.2.1 System Clock Specifications with ADC Operation

**Table 17-9. System Clock Characteristics with ADC Operation**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $f_{sysadc}$ | System clock frequency when the ADC module is operating (when PLL is bypassed) | 16 | - | - | MHz |

## 17.2.3 JTAG and Boundary Scan

**Table 17-10. JTAG Characteristics**

| Parameter No. | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| J1 | $f_{TCK}$ | TCK operational clock frequency | 0 | - | 10 | MHz |
| J2 | $t_{TCK}$ | TCK operational clock period | 100 | - | - | ns |

**Table 17-10. JTAG Characteristics *(continued)***

| Parameter No. | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| J3 | $t_{TCK\_LOW}$ | TCK clock Low time | - | $t_{TCK}$ | - | ns |
| J4 | $t_{TCK\_HIGH}$ | TCK clock High time | - | $t_{TCK}$ | - | ns |
| J5 | $t_{TCK\_R}$ | TCK rise time | 0 | - | 10 | ns |
| J6 | $t_{TCK\_F}$ | TCK fall time | 0 | - | 10 | ns |
| J7 | $t_{TMS\_SU}$ | TMS setup time to TCK rise | 20 | - | - | ns |
| J8 | $t_{TMS\_HLD}$ | TMS hold time from TCK rise | 20 | - | - | ns |
| J9 | $t_{TDI\_SU}$ | TDI setup time to TCK rise | 25 | - | - | ns |
| J10 | $t_{TDI\_HLD}$ | TDI hold time from TCK rise | 25 | - | - | ns |
| J11 $t_{TDO\_ZDV}$ | TCK fall to Data Valid from High-Z | 2-mA drive | - | 23 | 35 | ns |
| | | 4-mA drive | | 15 | 26 | ns |
| | | 8-mA drive | | 14 | 25 | ns |
| | | 8-mA drive with slew rate control | | 18 | 29 | ns |
| J12 $t_{TDO\_DV}$ | TCK fall to Data Valid from Data Valid | 2-mA drive | - | 21 | 35 | ns |
| | | 4-mA drive | | 14 | 25 | ns |
| | | 8-mA drive | | 13 | 24 | ns |
| | | 8-mA drive with slew rate control | | 18 | 28 | ns |
| J13 $t_{TDO\_DVZ}$ | TCK fall to High-Z from Data Valid | 2-mA drive | - | 9 | 11 | ns |
| | | 4-mA drive | | 7 | 9 | ns |
| | | 8-mA drive | | 6 | 8 | ns |
| | | 8-mA drive with slew rate control | | 7 | 9 | ns |
| J14 | $t_{TRST}$ | TRST assertion time | 100 | - | - | ns |
| J15 | $t_{TRST\_SU}$ | TRST setup time to TCK rise | 10 | - | - | ns |

**Figure 17-2. JTAG Test Clock Input Timing**

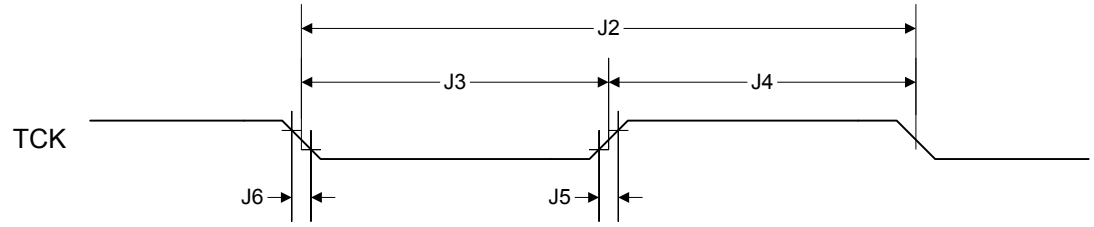


**Figure 17-3. JTAG Test Access Port (TAP) Timing**

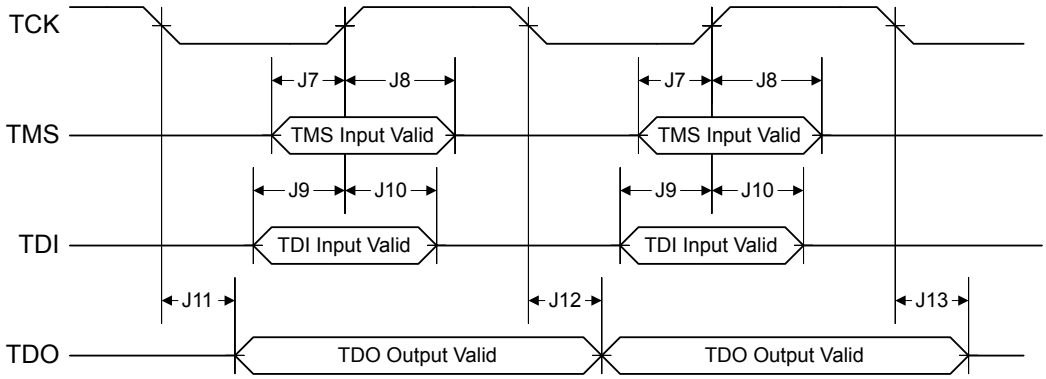
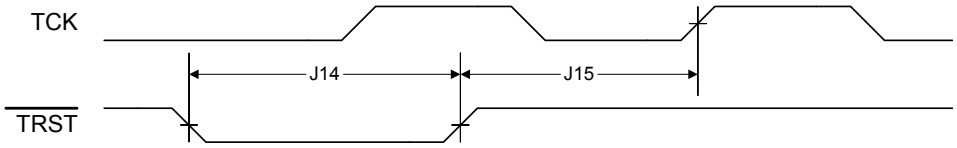



**Figure 17-4. JTAG TRST Timing**




## 17.2.4 Reset

**Table 17-11. Reset Characteristics**

| Parameter No. | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| R1 | $V_{TH}$ | Reset threshold | - | 2.0 | - | V |
| R2 | $V_{BTH}$ | Brown-Out threshold | 2.85 | 2.9 | 2.95 | V |
| R3 | $T_{POR}$ | Power-On Reset timeout | - | 10 | - | ms |
| R4 | $T_{BOR}$ | Brown-Out timeout | - | 500 | - | µs |
| R5 | $T_{IRPOR}$ | Internal reset timeout after POR | 15 | - | 30 | ms |
| R6 | $T_{IRBOR}$ | Internal reset timeout after BOR[a] | 2.5 | - | 20 | µs |
| R7 | $T_{IRHWR}$ | Internal reset timeout after hardware reset (RST pin) | 15 | - | 30 | ms |
| R8 | $T_{IRSWR}$ | Internal reset timeout after software-initiated system reset [a] | 2.5 | - | 20 | µs |
| R9 | $T_{IRWDR}$ | Internal reset timeout after watchdog reset[a] | 2.5 | - | 20 | µs |
| R10 | $T_{IRLDOR}$ | Internal reset timeout after LDO reset[a] | 2.5 | - | 20 | µs |
| R11 | $T_{VDDRISE}$ | Supply voltage ($V_{DD}$) rise time (0 V-3.3 V) | - | - | 100 | ms |

a. 20 * $t_{MOSC\_per}$

**Figure 17-5. External Reset Timing ($\overline{RST}$)**

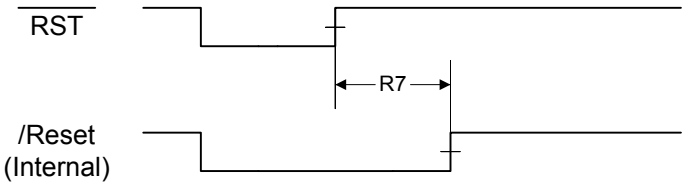



**Figure 17-6. Power-On Reset Timing**

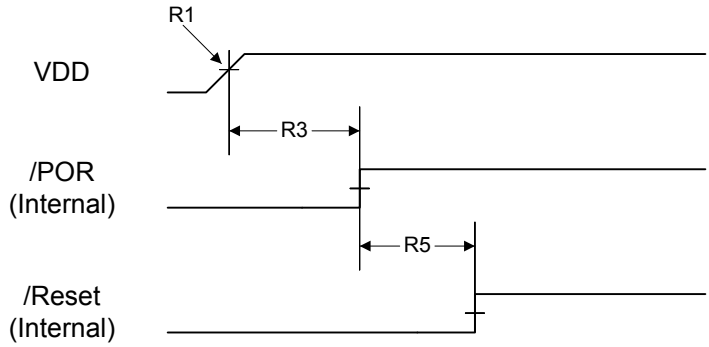



**Figure 17-7. Brown-Out Reset Timing**

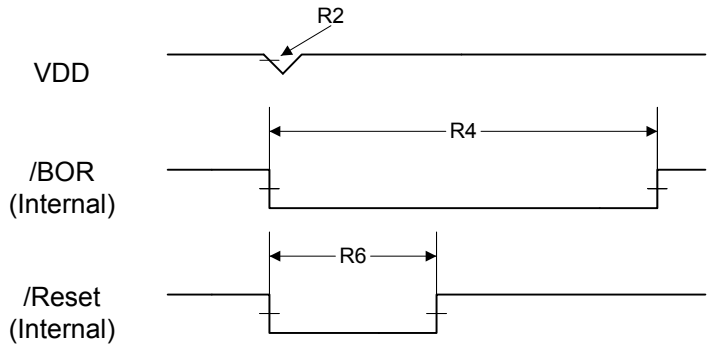


**Figure 17-8. Software Reset Timing**

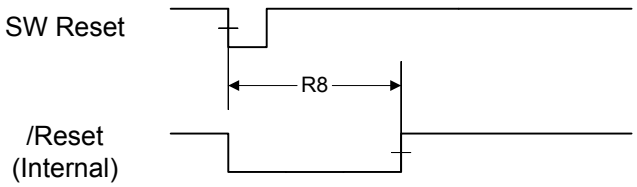



**Figure 17-9. Watchdog Reset Timing**

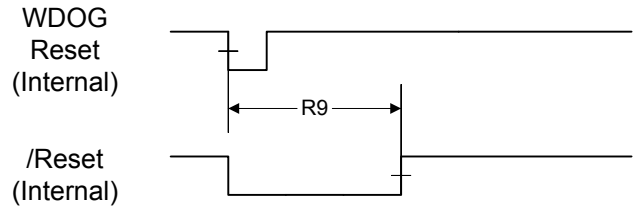
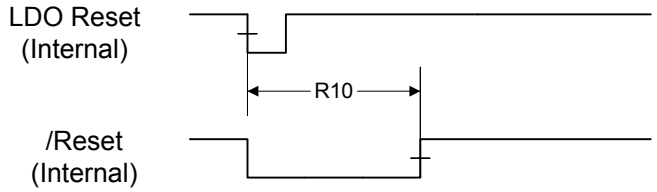



**Figure 17-10. LDO Reset Timing**




## 17.2.5 Sleep Modes

**Table 17-12. Sleep Modes AC Characteristics**[a]

| Parameter No | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| D1 | $t_{WAKE\_S}$ | Time to wake from interrupt in sleep or deep-sleep mode, not using the PLL | - | - | 7 | system clocks |
| D2 | $t_{WAKE\_PLL\_S}$ | Time to wake from interrupt in sleep or deep-sleep mode when using the PLL | - | - | $T_{READY}$ | ms |

a. Values in this table assume the IOSC is the clock source during sleep or deep-sleep mode.

## 17.2.6 General-Purpose I/O (GPIO)

**Note:** All GPIOs are 5 V-tolerant.

### Table 17-13. GPIO Characteristics

| Parameter | Parameter Name | Condition | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| $t_{GPIOR}$ | GPIO Rise Time (from 20% to 80% of $V_{DD}$) | 2-mA drive | - | 17 | 26 | ns |
| | | 4-mA drive | | 9 | 13 | ns |
| | | 8-mA drive | | 6 | 9 | ns |
| | | 8-mA drive with slew rate control | | 10 | 12 | ns |
| $t_{GPIOF}$ | GPIO Fall Time (from 80% to 20% of $V_{DD}$) | 2-mA drive | - | 17 | 25 | ns |
| | | 4-mA drive | | 8 | 12 | ns |
| | | 8-mA drive | | 6 | 10 | ns |
| | | 8-mA drive with slew rate control | | 11 | 13 | ns |

## 17.2.7 Analog-to-Digital Converter

### Table 17-14. ADC Characteristics[a]

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{ADCIN}$ | Maximum single-ended, full-scale analog input voltage | - | - | 3.0 | V |
| | Minimum single-ended, full-scale analog input voltage | 0.0 | - | - | V |
| | Maximum differential, full-scale analog input voltage | - | - | 1.5 | V |
| | Minimum differential, full-scale analog input voltage | 0.0 | - | - | V |
| N | Resolution | | 10 | | bits |
| $f_{ADC}$ | ADC internal clock frequency[b] | 14 | 16.667 | 18 | MHz |
| $t_{ADCCONV}$ | Conversion time[c] | | 1 | | µs |
| $f_{ADCCONV}$ | Conversion rate[c] | | 1041.667 | | k samples/s |
| $t_{LT}$ | Latency from trigger to start of conversion | - | 2 | - | system clocks |
| $I_L$ | ADC input leakage | - | - | ±3.0 | µA |
| $R_{ADC}$ | ADC equivalent resistance | - | - | 10 | kΩ |
| $C_{ADC}$ | ADC equivalent capacitance | 0.9 | 1.0 | 1.1 | pF |
| $E_L$ | Integral nonlinearity error | - | - | ±3 | LSB |
| $E_D$ | Differential nonlinearity error | - | - | ±2 | LSB |
| $E_O$ | Offset error | - | - | +6[d] | LSB |
| $E_G$ | Full-scale gain error | - | - | ±3 | LSB |
| $E_{TS}$ | Temperature sensor accuracy | - | - | ±5 | °C |

a. The ADC reference voltage is 3.0 V. This reference voltage is internally generated from the 3.3 VDDA supply by a band gap circuit.

b. The ADC must be clocked from the PLL or directly from an external clock source to operate properly.

c. The conversion time and rate scale from the specified number if the ADC internal clock frequency is any value other than 16.667 MHz.

d. The offset error listed above is the conversion result with 0 V applied to the ADC input.

**Figure 17-11. ADC Input Equivalency Diagram**

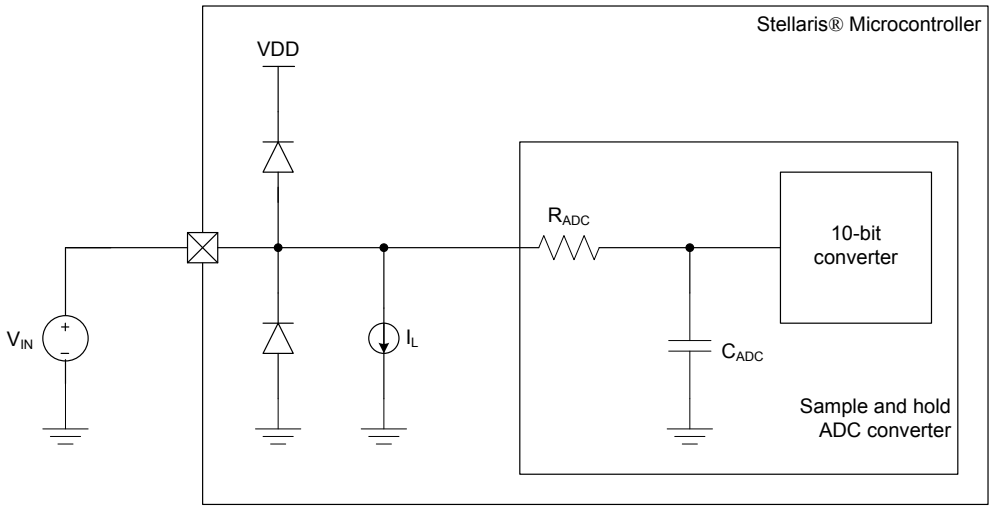



**Table 17-15. ADC Module Internal Reference Characteristics**

| Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|
| $V_{REFI}$ | Internal voltage reference for ADC | - | 3.0 | - | V |
| $E_{IR}$ | Internal voltage reference error | - | - | ±2.5 | % |

## 17.2.8 Synchronous Serial Interface (SSI)

**Table 17-16. SSI Characteristics**

| Parameter No. | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| S1 | $t_{clk\_per}$ | SSIClk cycle time | 2 | - | 65024 | system clocks |
| S2 | $t_{clk\_high}$ | SSIClk high time | - | 0.5 | - | t clk_per |
| S3 | $t_{clk\_low}$ | SSIClk low time | - | 0.5 | - | t clk_per |
| S4 | $t_{clkrf}$ | SSIClk rise/fall time[a] | - | 6 | 10 | ns |
| S5 | $t_{DMd}$ | Data from master valid delay time | 0 | - | 1 | system clocks |
| S6 | $t_{DMs}$ | Data from master setup time | 1 | - | - | system clocks |
| S7 | $t_{DMh}$ | Data from master hold time | 2 | - | - | system clocks |
| S8 | $t_{DSs}$ | Data from slave setup time | 1 | - | - | system clocks |
| S9 | $t_{DSh}$ | Data from slave hold time | 2 | - | - | system clocks |

a. Note that the delays shown are using 8-mA drive strength.

**Figure 17-12. SSI Timing for TI Frame Format (FRF=01), Single Transfer Timing Measurement**

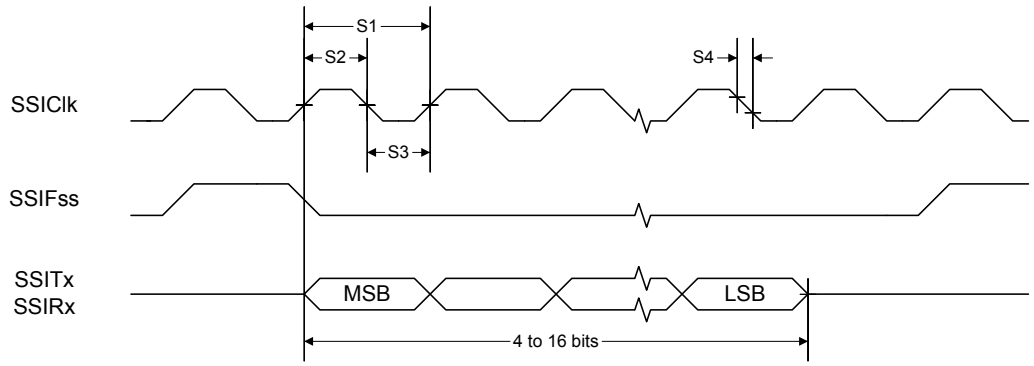



**Figure 17-13. SSI Timing for MICROWIRE Frame Format (FRF=10), Single Transfer**

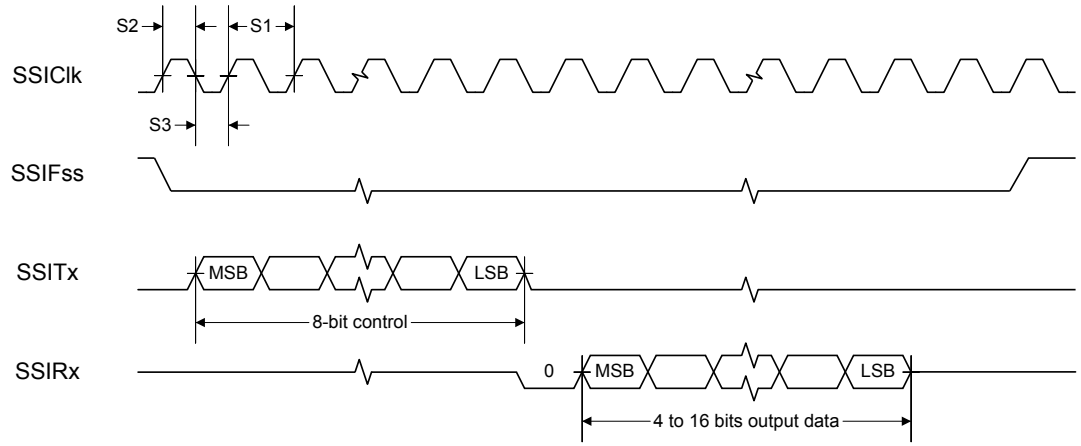
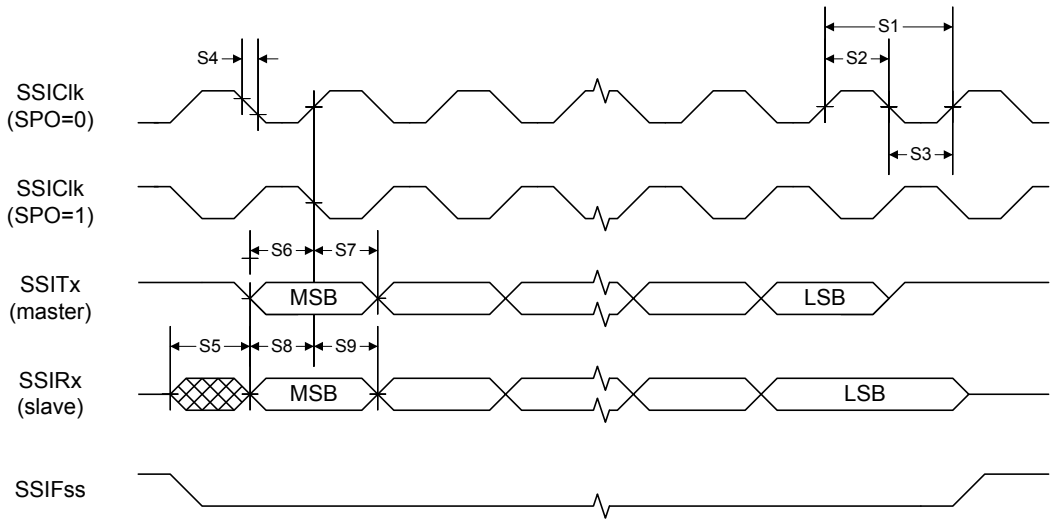


**Figure 17-14. SSI Timing for SPI Frame Format (FRF=00), with SPH=1**




## 17.2.9 Inter-Integrated Circuit ($I^2C$) Interface

**Table 17-17. $I^2C$ Characteristics**

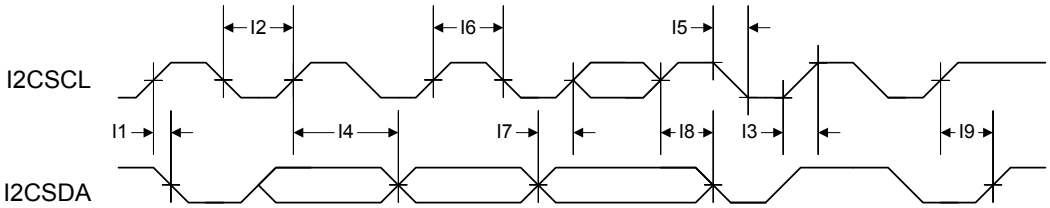
| Parameter No. | Parameter | Parameter Name | Min | Nom | Max | Unit |
|---|---|---|---|---|---|---|
| I1[a] | $t_{SCH}$ | Start condition hold time | 36 | - | - | system clocks |
| I2[a] | $t_{LP}$ | Clock Low period | 36 | - | - | system clocks |
| I3[b] | $t_{SRT}$ | I2CSCL/I2CSDA rise time ($V_{IL}$ =0.5 V to $V_{IH}$ =2.4 V) | - | - | (see note b) | ns |
| I4[a] | $t_{DH}$ | Data hold time | 2 | - | - | system clocks |
| I5[c] | $t_{SFT}$ | I2CSCL/I2CSDA fall time ($V_{IH}$ =2.4 V to $V_{IL}$ =0.5 V) | - | 9 | 10 | ns |
| I6[a] | $t_{HT}$ | Clock High time | 24 | - | - | system clocks |
| I7[a] | $t_{DS}$ | Data setup time | 18 | - | - | system clocks |
| I8[a] | $t_{SCSR}$ | Start condition setup time (for repeated start condition only) | 36 | - | - | system clocks |
| I9[a] | $t_{SCS}$ | Stop condition setup time | 24 | - | - | system clocks |

a. Values depend on the value programmed into the TPR bit in the **$I^2C$ Master Timer Period (I2CMTPR)** register; a TPR programmed for the maximum I2CSCL frequency (TPR=0x2) results in a minimum output timing as shown in the table above. The $I^2C$ interface is designed to scale the actual data transition time to move it to the middle of the I2CSCL Low period. The actual position is affected by the value programmed into the TPR; however, the numbers given in the above values are minimum values.

b. Because I2CSCL and I2CSDA are open-drain-type outputs, which the controller can only actively drive Low, the time I2CSCL or I2CSDA takes to reach a high level depends on external signal capacitance and pull-up resistor values.

c. Specified at a nominal 50 pF load.

**Figure 17-15. I²C Timing**



# A Serial Flash Loader

## A.1 Serial Flash Loader

The Stellaris® serial flash loader is a preprogrammed flash-resident utility used to download code to the flash memory of a device without the use of a debug interface. The serial flash loader uses a simple packet interface to provide synchronous communication with the device. The flash loader runs off the crystal and does not enable the PLL, so its speed is determined by the crystal used. The two serial interfaces that can be used are the UART0 and SSI0 interfaces. For simplicity, both the data format and communication protocol are identical for both serial interfaces.

## A.2 Interfaces

Once communication with the flash loader is established via one of the serial interfaces, that interface is used until the flash loader is reset or new code takes over. For example, once you start communicating using the SSI port, communications with the flash loader via the UART are disabled until the device is reset.

### A.2.1 UART

The Universal Asynchronous Receivers/Transmitters (UART) communication uses a fixed serial format of 8 bits of data, no parity, and 1 stop bit. The baud rate used for communication is automatically detected by the flash loader and can be any valid baud rate supported by the host and the device. The auto detection sequence requires that the baud rate should be no more than 1/32 the crystal frequency of the board that is running the serial flash loader. This is actually the same as the hardware limitation for the maximum baud rate for any UART on a Stellaris device which is calculated as follows:

```
Max Baud Rate = System Clock Frequency / 16
```

In order to determine the baud rate, the serial flash loader needs to determine the relationship between its own crystal frequency and the baud rate. This is enough information for the flash loader to configure its UART to the same baud rate as the host. This automatic baud-rate detection allows the host to use any valid baud rate that it wants to communicate with the device.

The method used to perform this automatic synchronization relies on the host sending the flash loader two bytes that are both 0x55. This generates a series of pulses to the flash loader that it can use to calculate the ratios needed to program the UART to match the host's baud rate. After the host sends the pattern, it attempts to read back one byte of data from the UART. The flash loader returns the value of 0xCC to indicate successful detection of the baud rate. If this byte is not received after at least twice the time required to transfer the two bytes, the host can resend another pattern of 0x55, 0x55, and wait for the 0xCC byte again until the flash loader acknowledges that it has received a synchronization pattern correctly. For example, the time to wait for data back from the flash loader should be calculated as at least 2*(20(bits/sync)/baud rate (bits/sec)). For a baud rate of 115200, this time is 2*(20/115200) or 0.35 ms.

### A.2.2 SSI

The Synchronous Serial Interface (SSI) port also uses a fixed serial format for communications, with the framing defined as Motorola format with SPH set to 1 and SPO set to 1. See "Frame Formats" on page 395 in the SSI chapter for more information on formats for this transfer protocol. Like the UART, this interface has hardware requirements that limit the maximum speed that the SSI clock can run. This allows the SSI clock to be at most 1/12 the crystal frequency of the board running

the flash loader. Since the host device is the master, the SSI on the flash loader device does not need to determine the clock as it is provided directly by the host.

## A.3 Packet Handling

All communications, with the exception of the UART auto-baud, are done via defined packets that are acknowledged (ACK) or not acknowledged (NAK) by the devices. The packets use the same format for receiving and sending packets, including the method used to acknowledge successful or unsuccessful reception of a packet.

### A.3.1 Packet Format

All packets sent and received from the device use the following byte-packed format.

```
struct
{
 unsigned char ucSize;
 unsigned char ucCheckSum;
 unsigned char Data[];
};
```

| | |
|---|---|
| ucSize | The first byte received holds the total size of the transfer including the size and checksum bytes. |
| ucChecksum | This holds a simple checksum of the bytes in the data buffer only. The algorithm is Data[0]+Data[1]+…+ Data[`ucSize`-3]. |
| Data | This is the raw data intended for the device, which is formatted in some form of command interface. There should be `ucSize`–2 bytes of data provided in this buffer to or from the device. |

### A.3.2 Sending Packets

The actual bytes of the packet can be sent individually or all at once; the only limitation is that commands that cause flash memory access should limit the download sizes to prevent losing bytes during flash programming. This limitation is discussed further in the section that describes the serial flash loader command, COMMAND_SEND_DATA (see “COMMAND_SEND_DATA (0x24)” on page 492).

Once the packet has been formatted correctly by the host, it should be sent out over the UART or SSI interface. Then the host should poll the UART or SSI interface for the first non-zero data returned from the device. The first non-zero byte will either be an ACK (0xCC) or a NAK (0x33) byte from the device indicating the packet was received successfully (ACK) or unsuccessfully (NAK). This does not indicate that the actual contents of the command issued in the data portion of the packet were valid, just that the packet was received correctly.

### A.3.3 Receiving Packets

The flash loader sends a packet of data in the same format that it receives a packet. The flash loader may transfer leading zero data before the first actual byte of data is sent out. The first non-zero byte is the size of the packet followed by a checksum byte, and finally followed by the data itself. There is no break in the data after the first non-zero byte is sent from the flash loader. Once the device communicating with the flash loader receives all the bytes, it must either ACK or NAK the packet to indicate that the transmission was successful. The appropriate response after sending a NAK to the flash loader is to resend the command that failed and request the data again. If needed, the host may send leading zeros before sending down the ACK/NAK signal to the flash loader, as the

flash loader only accepts the first non-zero data as a valid response. This zero padding is needed by the SSI interface in order to receive data to or from the flash loader.

## A.4 Commands

The next section defines the list of commands that can be sent to the flash loader. The first byte of the data should always be one of the defined commands, followed by data or parameters as determined by the command that is sent.

### A.4.1 COMMAND_PING (0X20)

This command simply accepts the command and sets the global status to success. The format of the packet is as follows:

```
Byte[0] = 0x03;
Byte[1] = checksum(Byte[2]);
Byte[2] = COMMAND_PING;
```

The ping command has 3 bytes and the value for COMMAND_PING is 0x20 and the checksum of one byte is that same byte, making Byte[1] also 0x20. Since the ping command has no real return status, the receipt of an ACK can be interpreted as a successful ping to the flash loader.

### A.4.2 COMMAND_GET_STATUS (0x23)

This command returns the status of the last command that was issued. Typically, this command should be sent after every command to ensure that the previous command was successful or to properly respond to a failure. The command requires one byte in the data of the packet and should be followed by reading a packet with one byte of data that contains a status code. The last step is to ACK or NAK the received data so the flash loader knows that the data has been read.

```
Byte[0] = 0x03
Byte[1] = checksum(Byte[2])
Byte[2] = COMMAND_GET_STATUS
```

### A.4.3 COMMAND_DOWNLOAD (0x21)

This command is sent to the flash loader to indicate where to store data and how many bytes will be sent by the COMMAND_SEND_DATA commands that follow. The command consists of two 32-bit values that are both transferred MSB first. The first 32-bit value is the address to start programming data into, while the second is the 32-bit size of the data that will be sent. This command also triggers an erase of the full area to be programmed so this command takes longer than other commands. This results in a longer time to receive the ACK/NAK back from the board. This command should be followed by a COMMAND_GET_STATUS to ensure that the Program Address and Program size are valid for the device running the flash loader.

The format of the packet to send this command is a follows:

```
Byte[0] = 11
Byte[1] = checksum(Bytes[2:10])
Byte[2] = COMMAND_DOWNLOAD
Byte[3] = Program Address [31:24]
Byte[4] = Program Address [23:16]
Byte[5] = Program Address [15:8]
Byte[6] = Program Address [7:0]
Byte[7] = Program Size [31:24]
```

```
Byte[8] = Program Size [23:16]
Byte[9] = Program Size [15:8]
Byte[10] = Program Size [7:0]
```

### A.4.4 COMMAND_SEND_DATA (0x24)

This command should only follow a COMMAND_DOWNLOAD command or another COMMAND_SEND_DATA command if more data is needed. Consecutive send data commands automatically increment address and continue programming from the previous location. The caller should limit transfers of data to a maximum 8 bytes of packet data to allow the flash to program successfully and not overflow input buffers of the serial interfaces. The command terminates programming once the number of bytes indicated by the COMMAND_DOWNLOAD command has been received. Each time this function is called it should be followed by a COMMAND_GET_STATUS to ensure that the data was successfully programmed into the flash. If the flash loader sends a NAK to this command, the flash loader does not increment the current address to allow retransmission of the previous data.

```
Byte[0] = 11
Byte[1] = checksum(Bytes[2:10])
Byte[2] = COMMAND_SEND_DATA
Byte[3] = Data[0]
Byte[4] = Data[1]
Byte[5] = Data[2]
Byte[6] = Data[3]
Byte[7] = Data[4]
Byte[8] = Data[5]
Byte[9] = Data[6]
Byte[10] = Data[7]
```

### A.4.5 COMMAND_RUN (0x22)

This command is used to tell the flash loader to execute from the address passed as the parameter in this command. This command consists of a single 32-bit value that is interpreted as the address to execute. The 32-bit value is transmitted MSB first and the flash loader responds with an ACK signal back to the host device before actually executing the code at the given address. This allows the host to know that the command was received successfully and the code is now running.

```
Byte[0] = 7
Byte[1] = checksum(Bytes[2:6])
Byte[2] = COMMAND_RUN
Byte[3] = Execute Address[31:24]
Byte[4] = Execute Address[23:16]
Byte[5] = Execute Address[15:8]
Byte[6] = Execute Address[7:0]
```

### A.4.6 COMMAND_RESET (0x25)

This command is used to tell the flash loader device to reset. This is useful when downloading a new image that overwrote the flash loader and wants to start from a full reset. Unlike the COMMAND_RUN command, this allows the initial stack pointer to be read by the hardware and set up for the new code. It can also be used to reset the flash loader if a critical error occurs and the host device wants to restart communication with the flash loader.

```
Byte[0] = 3
Byte[1] = checksum(Byte[2])
Byte[2] = COMMAND_RESET
```

The flash loader responds with an ACK signal back to the host device before actually executing the software reset to the device running the flash loader. This allows the host to know that the command was received successfully and the part will be reset.

# B Register Quick Reference

| 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 |
| **The Cortex-M3 Processor** | | | | | | | | | | | | | | | |
| **R0, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R1, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R2, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R3, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R4, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R5, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R6, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R7, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R8, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R9, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R10, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R11, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **R12, type R/W, , reset -** (see page 46) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **SP, type R/W, , reset -** (see page 47) | | | | | | | | | | | | | | | |
| SP | | | | | | | | | | | | | | | |
| SP | | | | | | | | | | | | | | | |
| **LR, type R/W, , reset 0xFFFF.FFFF** (see page 48) | | | | | | | | | | | | | | | |
| LINK | | | | | | | | | | | | | | | |
| LINK | | | | | | | | | | | | | | | |
| **PC, type R/W, , reset -** (see page 49) | | | | | | | | | | | | | | | |
| PC | | | | | | | | | | | | | | | |
| PC | | | | | | | | | | | | | | | |

| 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 |
| **PSR, type R/W, , reset 0x0100.0000** (see page 50) | | | | | | | | | | | | | | | |
| N | Z | C | V | Q | ICI / IT | | THUMB | | | | | | | | |
| ICI / IT | | | | | | | | | | ISRNUM | | | | | |
| **PRIMASK, type R/W, , reset 0x0000.0000** (see page 54) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | PRIMASK |
| **FAULTMASK, type R/W, , reset 0x0000.0000** (see page 55) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | FAULTMASK |
| **BASEPRI, type R/W, , reset 0x0000.0000** (see page 56) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | BASEPRI | | | | | | | |
| **CONTROL, type R/W, , reset 0x0000.0000** (see page 57) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | ASP | TMPL |
| **Cortex-M3 Peripherals**<br>**System Timer (SysTick) Registers**<br>Base 0xE000.E000 | | | | | | | | | | | | | | | |
| **STCTRL, type R/W, offset 0x010, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | COUNT |
| | | | | | | | | | | | | | CLK_SRC | INTEN | ENABLE |
| **STRELOAD, type R/W, offset 0x014, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | RELOAD | | | | | | | |
| RELOAD | | | | | | | | | | | | | | | |
| **STCURRENT, type R/WC, offset 0x018, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | CURRENT | | | | | | | |
| CURRENT | | | | | | | | | | | | | | | |
| **Cortex-M3 Peripherals**<br>**Nested Vectored Interrupt Controller (NVIC) Registers**<br>Base 0xE000.E000 | | | | | | | | | | | | | | | |
| **EN0, type R/W, offset 0x100, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | INT | | | | | | | | | | | | | |
| INT | | | | | | | | | | | | | | | |
| **DIS0, type R/W, offset 0x180, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | INT | | | | | | | | | | | | | |
| INT | | | | | | | | | | | | | | | |
| **PEND0, type R/W, offset 0x200, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | INT | | | | | | | | | | | | | |
| INT | | | | | | | | | | | | | | | |
| **UNPEND0, type R/W, offset 0x280, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | INT | | | | | | | | | | | | | |
| INT | | | | | | | | | | | | | | | |
| **ACTIVE0, type RO, offset 0x300, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | INT | | | | | | | | | | | | | |
| INT | | | | | | | | | | | | | | | |
| **PRI0, type R/W, offset 0x400, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI1, type R/W, offset 0x404, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |

| 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 |
| **PRI2, type R/W, offset 0x408, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI3, type R/W, offset 0x40C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI4, type R/W, offset 0x410, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI5, type R/W, offset 0x414, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI6, type R/W, offset 0x418, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **PRI7, type R/W, offset 0x41C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| INTD | | | | | | | | INTC | | | | | | | |
| INTB | | | | | | | | INTA | | | | | | | |
| **SWTRIG, type WO, offset 0xF00, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | INTID | | | | |

## Cortex-M3 Peripherals

### System Control Block (SCB) Registers

Base 0xE000.E000

| 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 |
| **CPUID, type RO, offset 0xD00, reset 0x410F.C231** | | | | | | | | | | | | | | | |
| IMP | | | | | | | | VAR | | | | CON | | | |
| PARTNO | | | | | | | | | | | | REV | | | |
| **INTCTRL, type R/W, offset 0xD04, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| NMISET | | | PENDSV | UNPENDSV | PENDSTSET | PENDSTCLR | | ISRPRE | ISRPEND | | | | | VECPEND | |
| VECPEND | | | | RETBASE | | | | | | VECACT | | | | | |
| **VTABLE, type R/W, offset 0xD08, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | BASE | OFFSET | | | | | | | | | | | | |
| OFFSET | | | | | | | | | | | | | | | |
| **APINT, type R/W, offset 0xD0C, reset 0xFA05.0000** | | | | | | | | | | | | | | | |
| VECTKEY | | | | | | | | | | | | | | | |
| ENDIANESS | | | | | PRIGROUP | | | | | | | | SYSRESREQ | VECTCLRACT | VECTRESET |
| **SYSCTRL, type R/W, offset 0xD10, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | SEVONPEND | | SLEEPDEEP | SLEEPEXIT | |
| **CFGCTRL, type R/W, offset 0xD14, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | STKALIGN | BFHFNMIGN | | | | DIV0 | UNALIGNED | | MAINPEND | BASETHR |
| **SYSPRI1, type R/W, offset 0xD18, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | USAGE | | | | | | | |
| BUS | | | | | | | | MEM | | | | | | | |
| **SYSPRI2, type R/W, offset 0xD1C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| SVC | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| **SYSPRI3, type R/W, offset 0xD20, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| TICK | | | | | | | | PENDSV | | | | | | | |
| | | | | | | | | DEBUG | | | | | | | |

| 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 |
| **SYSHNDCTRL, type R/W, offset 0xD24, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | USAGE | BUS | MEM |
| SVC | BUSP | MEMP | USAGEP | TICK | PNDSV | | MON | SVCA | | | | USGA | | BUSA | MEMA |
| **FAULTSTAT, type R/W1C, offset 0xD28, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | DIV0 | UNALIGN | | | | | NOCP | INVPC | INVSTAT | UNDEF |
| BFARV | | | BSTKE | BUSTKE | IMPRE | PRECISE | IBUS | MMARV | | | MSTKE | MUSTKE | | DERR | IERR |
| **HFAULTSTAT, type R/W1C, offset 0xD2C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| DBG | FORCED | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | VECT | |
| **MMADDR, type R/W, offset 0xD34, reset -** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| **FAULTADDR, type R/W, offset 0xD38, reset -** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |

## Cortex-M3 Peripherals

## Memory Protection Unit (MPU) Registers

Base 0xE000.E000

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **MPUTYPE, type RO, offset 0xD90, reset 0x0000.0800** | | | | | | | | | | | | | | | |
| | | | | | | | | IREGION | | | | | | | |
| DREGION | | | | | | | | | | | | | | | SEPARATE |
| **MPUCTRL, type R/W, offset 0xD94, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | PRIVDEFEN | HFNMIENA | ENABLE |
| **MPUNUMBER, type R/W, offset 0xD98, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | NUMBER | | |
| **MPUBASE, type R/W, offset 0xD9C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | VALID | | REGION | | |
| **MPUBASE1, type R/W, offset 0xDA4, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | VALID | | REGION | | |
| **MPUBASE2, type R/W, offset 0xDAC, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | VALID | | REGION | | |
| **MPUBASE3, type R/W, offset 0xDB4, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | VALID | | REGION | | |
| **MPUATTR, type R/W, offset 0xDA0, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | XN | | AP | | | | | TEX | | | S | C | B |
| SRD | | | | | | | | | | SIZE | | | | | ENABLE |
| **MPUATTR1, type R/W, offset 0xDA8, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | XN | | AP | | | | | TEX | | | S | C | B |
| SRD | | | | | | | | | | SIZE | | | | | ENABLE |
| **MPUATTR2, type R/W, offset 0xDB0, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | XN | | AP | | | | | TEX | | | S | C | B |
| SRD | | | | | | | | | | SIZE | | | | | ENABLE |
| **MPUATTR3, type R/W, offset 0xDB8, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | XN | | AP | | | | | TEX | | | S | C | B |
| SRD | | | | | | | | | | SIZE | | | | | ENABLE |

| 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 |
| **System Control**<br>Base 0x400F.E000 | | | | | | | | | | | | | | | |
| **DID0, type RO, offset 0x000, reset -** (see page 161) | | | | | | | | | | | | | | | |
| | | VER | | | | | | | | | | | | | |
| MAJOR | | | | | | | | MINOR | | | | | | | |
| **PBORCTL, type R/W, offset 0x030, reset 0x0000.7FFD** (see page 163) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| BORTIM | | | | | | | | | | | | | | BORIOR | BORWT |
| **LDOPCTL, type R/W, offset 0x034, reset 0x0000.0000** (see page 164) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | VADJ | | | | | |
| **RIS, type RO, offset 0x050, reset 0x0000.0000** (see page 165) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | PLLLRIS | CLRIS | IOFRIS | MOFRIS | LDORIS | BORRIS | PLLFRIS |
| **IMC, type R/W, offset 0x054, reset 0x0000.0000** (see page 166) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | PLLLIM | CLIM | IOFIM | MOFIM | LDOIM | BORIM | PLLFIM |
| **MISC, type R/W1C, offset 0x058, reset 0x0000.0000** (see page 167) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | PLLLMIS | CLMIS | IOFMIS | MOFMIS | LDOMIS | BORMIS | |
| **RESC, type R/W, offset 0x05C, reset -** (see page 168) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | LDO | SW | WDT | BOR | POR | EXT |
| **RCC, type R/W, offset 0x060, reset 0x0780.3AC0** (see page 169) | | | | | | | | | | | | | | | |
| | | | | ACG | | SYSDIV | | | USESYSDIV | | | | | | |
| | | PWRDN | OEN | BYPASS | PLLVER | XTAL | | | | OSCSRC | | IOSCVER | MOSCVER | IOSCDIS | MOSCDIS |
| **PLLCFG, type RO, offset 0x064, reset -** (see page 172) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| OD | | F | | | | | | | | | R | | | | |
| **DSLPCLKCFG, type R/W, offset 0x144, reset 0x0780.0000** (see page 173) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | IOSC |
| **CLKVCLR, type R/W, offset 0x150, reset 0x0000.0000** (see page 174) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | VERCLR |
| **LDOARST, type R/W, offset 0x160, reset 0x0000.0000** (see page 175) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | LDOARST |
| **DID1, type RO, offset 0x004, reset -** (see page 176) | | | | | | | | | | | | | | | |
| VER | | | | FAM | | | | PARTNO | | | | | | | |
| | | | | | | | | TEMP | | | PKG | | ROHS | QUAL | |
| **DC0, type RO, offset 0x008, reset 0x001F.001F** (see page 178) | | | | | | | | | | | | | | | |
| SRAMSZ | | | | | | | | | | | | | | | |
| FLASHSZ | | | | | | | | | | | | | | | |
| **DC1, type RO, offset 0x010, reset 0x0001.33BF** (see page 179) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | ADC |
| MINSYSDIV | | | | | | MAXADCSPD | | MPU | | TEMPSNS | PLL | WDT | SWO | SWD | JTAG |
| **DC2, type RO, offset 0x014, reset 0x0007.1013** (see page 181) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| | | | I2C0 | | | | | | | | SSI0 | | | UART1 | UART0 |
| **DC3, type RO, offset 0x018, reset 0xBFFF.0000** (see page 183) | | | | | | | | | | | | | | | |
| 32KHZ | | CCP5 | CCP4 | CCP3 | CCP2 | CCP1 | CCP0 | ADC7 | ADC6 | ADC5 | ADC4 | ADC3 | ADC2 | ADC1 | ADC0 |
| | | | | | | | | | | | | | | | |

| 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 |
| **DC4, type RO, offset 0x01C, reset 0x0000.001F** (see page 185) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| **RCGC0, type R/W, offset 0x100, reset 0x00000040** (see page 186) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | ADC |
| | | | | | | MAXADCSPD | | | | | | WDT | | | |
| **SCGC0, type R/W, offset 0x110, reset 0x00000040** (see page 188) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | ADC |
| | | | | | | MAXADCSPD | | | | | | WDT | | | |
| **DCGC0, type R/W, offset 0x120, reset 0x00000040** (see page 190) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | ADC |
| | | | | | | | | | | | | WDT | | | |
| **RCGC1, type R/W, offset 0x104, reset 0x00000000** (see page 191) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| | | | I2C0 | | | | | | | | SSI0 | | | UART1 | UART0 |
| **SCGC1, type R/W, offset 0x114, reset 0x00000000** (see page 193) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| | | | I2C0 | | | | | | | | SSI0 | | | UART1 | UART0 |
| **DCGC1, type R/W, offset 0x124, reset 0x00000000** (see page 195) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| | | | I2C0 | | | | | | | | SSI0 | | | UART1 | UART0 |
| **RCGC2, type R/W, offset 0x108, reset 0x00000000** (see page 197) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| **SCGC2, type R/W, offset 0x118, reset 0x00000000** (see page 198) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| **DCGC2, type R/W, offset 0x128, reset 0x00000000** (see page 199) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| **SRCR0, type R/W, offset 0x040, reset 0x00000000** (see page 200) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | ADC |
| | | | | | | | | | | | | WDT | | | |
| **SRCR1, type R/W, offset 0x044, reset 0x00000000** (see page 201) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | TIMER2 | TIMER1 | TIMER0 |
| | | | I2C0 | | | | | | | | SSI0 | | | UART1 | UART0 |
| **SRCR2, type R/W, offset 0x048, reset 0x00000000** (see page 202) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | GPIOE | GPIOD | GPIOC | GPIOB | GPIOA |
| **Internal Memory**<br>**Flash Memory Control Registers (Flash Control Offset)**<br>Base 0x400F.D000 | | | | | | | | | | | | | | | |
| **FMA, type R/W, offset 0x000, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| OFFSET | | | | | | | | | | | | | | | |
| **FMD, type R/W, offset 0x004, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **FMC, type R/W, offset 0x008, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| WRKEY | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | COMT | MERASE | ERASE | WRITE |

| 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 |
| **FCRIS, type RO, offset 0x00C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | PRIS | ARIS |
| **FCIM, type R/W, offset 0x010, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | PMASK | AMASK |
| **FCMISC, type R/W1C, offset 0x014, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | PMISC | AMISC |
| **Internal Memory**<br>**Flash Memory Protection Registers (System Control Offset)**<br>Base 0x400F.E000 | | | | | | | | | | | | | | | |
| **USECRL, type R/W, offset 0x140, reset 0x31** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | USEC | | | | | | | |
| **FMPRE, type R/W, offset 0x130, reset 0xBFFF.FFFF** | | | | | | | | | | | | | | | |
| DBG | READ_ENABLE | | | | | | | | | | | | | | |
| READ_ENABLE | | | | | | | | | | | | | | | |
| **FMPPE, type R/W, offset 0x134, reset 0xFFFF.FFFF** | | | | | | | | | | | | | | | |
| PROG_ENABLE | | | | | | | | | | | | | | | |
| PROG_ENABLE | | | | | | | | | | | | | | | |
| **General-Purpose Input/Outputs (GPIOs)**<br>GPIO Port A base: 0x4000.4000<br>GPIO Port B base: 0x4000.5000<br>GPIO Port C base: 0x4000.6000<br>GPIO Port D base: 0x4000.7000<br>GPIO Port E base: 0x4002.4000 | | | | | | | | | | | | | | | |
| **GPIODATA, type R/W, offset 0x000, reset 0x0000.0000** (see page 228) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | DATA | | | | | | | |
| **GPIODIR, type R/W, offset 0x400, reset 0x0000.0000** (see page 229) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | DIR | | | | | | | |
| **GPIOIS, type R/W, offset 0x404, reset 0x0000.0000** (see page 230) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | IS | | | | | | | |
| **GPIOIBE, type R/W, offset 0x408, reset 0x0000.0000** (see page 231) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | IBE | | | | | | | |
| **GPIOIEV, type R/W, offset 0x40C, reset 0x0000.0000** (see page 232) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | IEV | | | | | | | |
| **GPIOIM, type R/W, offset 0x410, reset 0x0000.0000** (see page 233) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | IME | | | | | | | |
| **GPIORIS, type RO, offset 0x414, reset 0x0000.0000** (see page 234) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | RIS | | | | | | | |
| **GPIOMIS, type RO, offset 0x418, reset 0x0000.0000** (see page 235) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | MIS | | | | | | | |
| **GPIOICR, type W1C, offset 0x41C, reset 0x0000.0000** (see page 236) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | IC | | | | | | | |

| 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 |
| **GPIOAFSEL, type R/W, offset 0x420, reset -** (see page 237) | | | | | | | | | | | | | | | |
| | | | | | | | | AFSEL | | | | | | | |
| **GPIODR2R, type R/W, offset 0x500, reset 0x0000.00FF** (see page 239) | | | | | | | | | | | | | | | |
| | | | | | | | | DRV2 | | | | | | | |
| **GPIODR4R, type R/W, offset 0x504, reset 0x0000.0000** (see page 240) | | | | | | | | | | | | | | | |
| | | | | | | | | DRV4 | | | | | | | |
| **GPIODR8R, type R/W, offset 0x508, reset 0x0000.0000** (see page 241) | | | | | | | | | | | | | | | |
| | | | | | | | | DRV8 | | | | | | | |
| **GPIOODR, type R/W, offset 0x50C, reset 0x0000.0000** (see page 242) | | | | | | | | | | | | | | | |
| | | | | | | | | ODE | | | | | | | |
| **GPIOPUR, type R/W, offset 0x510, reset 0x0000.00FF** (see page 243) | | | | | | | | | | | | | | | |
| | | | | | | | | PUE | | | | | | | |
| **GPIOPDR, type R/W, offset 0x514, reset 0x0000.0000** (see page 244) | | | | | | | | | | | | | | | |
| | | | | | | | | PDE | | | | | | | |
| **GPIOSLR, type R/W, offset 0x518, reset 0x0000.0000** (see page 245) | | | | | | | | | | | | | | | |
| | | | | | | | | SRL | | | | | | | |
| **GPIODEN, type R/W, offset 0x51C, reset 0x0000.00FF** (see page 246) | | | | | | | | | | | | | | | |
| | | | | | | | | DEN | | | | | | | |
| **GPIOPeriphID4, type RO, offset 0xFD0, reset 0x0000.0000** (see page 247) | | | | | | | | | | | | | | | |
| | | | | | | | | PID4 | | | | | | | |
| **GPIOPeriphID5, type RO, offset 0xFD4, reset 0x0000.0000** (see page 248) | | | | | | | | | | | | | | | |
| | | | | | | | | PID5 | | | | | | | |
| **GPIOPeriphID6, type RO, offset 0xFD8, reset 0x0000.0000** (see page 249) | | | | | | | | | | | | | | | |
| | | | | | | | | PID6 | | | | | | | |
| **GPIOPeriphID7, type RO, offset 0xFDC, reset 0x0000.0000** (see page 250) | | | | | | | | | | | | | | | |
| | | | | | | | | PID7 | | | | | | | |
| **GPIOPeriphID0, type RO, offset 0xFE0, reset 0x0000.0061** (see page 251) | | | | | | | | | | | | | | | |
| | | | | | | | | PID0 | | | | | | | |
| **GPIOPeriphID1, type RO, offset 0xFE4, reset 0x0000.0000** (see page 252) | | | | | | | | | | | | | | | |
| | | | | | | | | PID1 | | | | | | | |
| **GPIOPeriphID2, type RO, offset 0xFE8, reset 0x0000.0018** (see page 253) | | | | | | | | | | | | | | | |
| | | | | | | | | PID2 | | | | | | | |
| **GPIOPeriphID3, type RO, offset 0xFEC, reset 0x0000.0001** (see page 254) | | | | | | | | | | | | | | | |
| | | | | | | | | PID3 | | | | | | | |

| 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 |
| **GPIOPCellID0, type RO, offset 0xFF0, reset 0x0000.000D** (see page 255) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID0 | | | | | | | |
| **GPIOPCellID1, type RO, offset 0xFF4, reset 0x0000.00F0** (see page 256) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID1 | | | | | | | |
| **GPIOPCellID2, type RO, offset 0xFF8, reset 0x0000.0005** (see page 257) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID2 | | | | | | | |
| **GPIOPCellID3, type RO, offset 0xFFC, reset 0x0000.00B1** (see page 258) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID3 | | | | | | | |
| **General-Purpose Timers**<br>Timer0 base: 0x4003.0000<br>Timer1 base: 0x4003.1000<br>Timer2 base: 0x4003.2000 | | | | | | | | | | | | | | | |
| **GPTMCFG, type R/W, offset 0x000, reset 0x0000.0000** (see page 271) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | GPTMCFG | | |
| **GPTMTAMR, type R/W, offset 0x004, reset 0x0000.0000** (see page 272) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TAAMS | TACMR | TAMR | |
| **GPTMTBMR, type R/W, offset 0x008, reset 0x0000.0000** (see page 274) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TBAMS | TBCMR | TBMR | |
| **GPTMCTL, type R/W, offset 0x00C, reset 0x0000.0000** (see page 276) | | | | | | | | | | | | | | | |
| | TBPWML | TBOTE | | TBEVENT | | TBSTALL | TBEN | | TAPWML | TAOTE | RTCEN | TAEVENT | | TASTALL | TAEN |
| **GPTMIMR, type R/W, offset 0x018, reset 0x0000.0000** (see page 279) | | | | | | | | | | | | | | | |
| | | | | | CBEIM | CBMIM | TBTOIM | | | | | RTCIM | CAEIM | CAMIM | TATOIM |
| **GPTMRIS, type RO, offset 0x01C, reset 0x0000.0000** (see page 281) | | | | | | | | | | | | | | | |
| | | | | | CBERIS | CBMRIS | TBTORIS | | | | | RTCRIS | CAERIS | CAMRIS | TATORIS |
| **GPTMMIS, type RO, offset 0x020, reset 0x0000.0000** (see page 282) | | | | | | | | | | | | | | | |
| | | | | | CBEMIS | CBMMIS | TBTOMIS | | | | | RTCMIS | CAEMIS | CAMMIS | TATOMIS |
| **GPTMICR, type W1C, offset 0x024, reset 0x0000.0000** (see page 283) | | | | | | | | | | | | | | | |
| | | | | | CBECINT | CBMCINT | TBTOCINT | | | | | RTCCINT | CAECINT | CAMCINT | TATOCINT |
| **GPTMTAILR, type R/W, offset 0x028, reset 0xFFFF.FFFF** (see page 285) | | | | | | | | | | | | | | | |
| | | | | | | | | TAILRH | | | | | | | |
| | | | | | | | | TAILRL | | | | | | | |
| **GPTMTBILR, type R/W, offset 0x02C, reset 0x0000.FFFF** (see page 286) | | | | | | | | | | | | | | | |
| | | | | | | | | TBILRL | | | | | | | |
| **GPTMTAMATCHR, type R/W, offset 0x030, reset 0xFFFF.FFFF** (see page 287) | | | | | | | | | | | | | | | |
| | | | | | | | | TAMRH | | | | | | | |
| | | | | | | | | TAMRL | | | | | | | |
| **GPTMTBMATCHR, type R/W, offset 0x034, reset 0x0000.FFFF** (see page 288) | | | | | | | | | | | | | | | |
| | | | | | | | | TBMRL | | | | | | | |

| 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 |
| **GPTMTAPR, type R/W, offset 0x038, reset 0x0000.0000** (see page 289) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | TAPSR | | | | | | | |
| **GPTMTBPR, type R/W, offset 0x03C, reset 0x0000.0000** (see page 290) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | TBPSR | | | | | | | |
| **GPTMTAPMR, type R/W, offset 0x040, reset 0x0000.0000** (see page 291) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | TAPSMR | | | | | | | |
| **GPTMTBPMR, type R/W, offset 0x044, reset 0x0000.0000** (see page 292) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | TBPSMR | | | | | | | |
| **GPTMTAR, type RO, offset 0x048, reset 0xFFFF.FFFF** (see page 293) | | | | | | | | | | | | | | | |
| TARH | | | | | | | | | | | | | | | |
| TARL | | | | | | | | | | | | | | | |
| **GPTMTBR, type RO, offset 0x04C, reset 0x0000.FFFF** (see page 294) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| TBRL | | | | | | | | | | | | | | | |
| **Watchdog Timer**<br>Base 0x4000.0000 | | | | | | | | | | | | | | | |
| **WDTLOAD, type R/W, offset 0x000, reset 0xFFFF.FFFF** (see page 299) | | | | | | | | | | | | | | | |
| WDTLoad | | | | | | | | | | | | | | | |
| WDTLoad | | | | | | | | | | | | | | | |
| **WDTVALUE, type RO, offset 0x004, reset 0xFFFF.FFFF** (see page 300) | | | | | | | | | | | | | | | |
| WDTValue | | | | | | | | | | | | | | | |
| WDTValue | | | | | | | | | | | | | | | |
| **WDTCTL, type R/W, offset 0x008, reset 0x0000.0000** (see page 301) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | RESEN | INTEN |
| **WDTICR, type WO, offset 0x00C, reset -** (see page 302) | | | | | | | | | | | | | | | |
| WDTIntClr | | | | | | | | | | | | | | | |
| WDTIntClr | | | | | | | | | | | | | | | |
| **WDTRIS, type RO, offset 0x010, reset 0x0000.0000** (see page 303) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | WDTRIS |
| **WDTMIS, type RO, offset 0x014, reset 0x0000.0000** (see page 304) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | WDTMIS |
| **WDTTEST, type R/W, offset 0x418, reset 0x0000.0000** (see page 305) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | STALL | | | | | | | | |
| **WDTLOCK, type R/W, offset 0xC00, reset 0x0000.0000** (see page 306) | | | | | | | | | | | | | | | |
| WDTLock | | | | | | | | | | | | | | | |
| WDTLock | | | | | | | | | | | | | | | |
| **WDTPeriphID4, type RO, offset 0xFD0, reset 0x0000.0000** (see page 307) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID4 | | | | | | | |
| **WDTPeriphID5, type RO, offset 0xFD4, reset 0x0000.0000** (see page 308) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID5 | | | | | | | |
| **WDTPeriphID6, type RO, offset 0xFD8, reset 0x0000.0000** (see page 309) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID6 | | | | | | | |

| 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 |
| **WDTPeriphID7, type RO, offset 0xFDC, reset 0x0000.0000** (see page 310) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID7 | | | | | | | |
| **WDTPeriphID0, type RO, offset 0xFE0, reset 0x0000.0005** (see page 311) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID0 | | | | | | | |
| **WDTPeriphID1, type RO, offset 0xFE4, reset 0x0000.0018** (see page 312) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID1 | | | | | | | |
| **WDTPeriphID2, type RO, offset 0xFE8, reset 0x0000.0018** (see page 313) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID2 | | | | | | | |
| **WDTPeriphID3, type RO, offset 0xFEC, reset 0x0000.0001** (see page 314) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID3 | | | | | | | |
| **WDTPCellID0, type RO, offset 0xFF0, reset 0x0000.000D** (see page 315) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID0 | | | | | | | |
| **WDTPCellID1, type RO, offset 0xFF4, reset 0x0000.00F0** (see page 316) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID1 | | | | | | | |
| **WDTPCellID2, type RO, offset 0xFF8, reset 0x0000.0005** (see page 317) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID2 | | | | | | | |
| **WDTPCellID3, type RO, offset 0xFFC, reset 0x0000.00B1** (see page 318) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID3 | | | | | | | |
| **Analog-to-Digital Converter (ADC)**<br>Base 0x4003.8000 | | | | | | | | | | | | | | | |
| **ADCACTSS, type R/W, offset 0x000, reset 0x0000.0000** (see page 328) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | ASEN3 | ASEN2 | ASEN1 | ASEN0 |
| **ADCRIS, type RO, offset 0x004, reset 0x0000.0000** (see page 329) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | INR3 | INR2 | INR1 | INR0 |
| **ADCIM, type R/W, offset 0x008, reset 0x0000.0000** (see page 330) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | MASK3 | MASK2 | MASK1 | MASK0 |
| **ADCISC, type R/W1C, offset 0x00C, reset 0x0000.0000** (see page 331) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | IN3 | IN2 | IN1 | IN0 |
| **ADCOSTAT, type R/W1C, offset 0x010, reset 0x0000.0000** (see page 332) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | OV3 | OV2 | OV1 | OV0 |
| **ADCEMUX, type R/W, offset 0x014, reset 0x0000.0000** (see page 333) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| EM3 | | | | EM2 | | | | EM1 | | | | EM0 | | | |
| **ADCUSTAT, type R/W1C, offset 0x018, reset 0x0000.0000** (see page 336) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | UV3 | UV2 | UV1 | UV0 |
| **ADCSSPRI, type R/W, offset 0x020, reset 0x0000.3210** (see page 337) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | SS3 | | | | SS2 | | | | SS1 | | | | SS0 | |

| 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 |
| **ADCPSSI, type WO, offset 0x028, reset -** (see page 339) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | SS3 | SS2 | SS1 | SS0 |
| **ADCSAC, type R/W, offset 0x030, reset 0x0000.0000** (see page 340) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | AVG | |
| **ADCSSMUX0, type R/W, offset 0x040, reset 0x0000.0000** (see page 341) | | | | | | | | | | | | | | | |
| | MUX7 | | | | MUX6 | | | | MUX5 | | | | MUX4 | | |
| | MUX3 | | | | MUX2 | | | | MUX1 | | | | MUX0 | | |
| **ADCSSCTL0, type R/W, offset 0x044, reset 0x0000.0000** (see page 343) | | | | | | | | | | | | | | | |
| TS7 | IE7 | END7 | D7 | TS6 | IE6 | END6 | D6 | TS5 | IE5 | END5 | D5 | TS4 | IE4 | END4 | D4 |
| TS3 | IE3 | END3 | D3 | TS2 | IE2 | END2 | D2 | TS1 | IE1 | END1 | D1 | TS0 | IE0 | END0 | D0 |
| **ADCSSFIFO0, type RO, offset 0x048, reset -** (see page 346) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | DATA | | | | | | | | | |
| **ADCSSFIFO1, type RO, offset 0x068, reset -** (see page 346) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | DATA | | | | | | | | | |
| **ADCSSFIFO2, type RO, offset 0x088, reset -** (see page 346) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | DATA | | | | | | | | | |
| **ADCSSFIFO3, type RO, offset 0x0A8, reset -** (see page 346) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | DATA | | | | | | | | | |
| **ADCSSFSTAT0, type RO, offset 0x04C, reset 0x0000.0100** (see page 347) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | FULL | | | | EMPTY | | HPTR | | | | TPTR | | |
| **ADCSSFSTAT1, type RO, offset 0x06C, reset 0x0000.0100** (see page 347) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | FULL | | | | EMPTY | | HPTR | | | | TPTR | | |
| **ADCSSFSTAT2, type RO, offset 0x08C, reset 0x0000.0100** (see page 347) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | FULL | | | | EMPTY | | HPTR | | | | TPTR | | |
| **ADCSSFSTAT3, type RO, offset 0x0AC, reset 0x0000.0100** (see page 347) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | FULL | | | | EMPTY | | HPTR | | | | TPTR | | |
| **ADCSSMUX1, type R/W, offset 0x060, reset 0x0000.0000** (see page 348) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | MUX3 | | | | MUX2 | | | | MUX1 | | | | MUX0 | | |
| **ADCSSMUX2, type R/W, offset 0x080, reset 0x0000.0000** (see page 348) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | MUX3 | | | | MUX2 | | | | MUX1 | | | | MUX0 | | |
| **ADCSSCTL1, type R/W, offset 0x064, reset 0x0000.0000** (see page 349) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| TS3 | IE3 | END3 | D3 | TS2 | IE2 | END2 | D2 | TS1 | IE1 | END1 | D1 | TS0 | IE0 | END0 | D0 |
| **ADCSSCTL2, type R/W, offset 0x084, reset 0x0000.0000** (see page 349) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| TS3 | IE3 | END3 | D3 | TS2 | IE2 | END2 | D2 | TS1 | IE1 | END1 | D1 | TS0 | IE0 | END0 | D0 |
| **ADCSSMUX3, type R/W, offset 0x0A0, reset 0x0000.0000** (see page 351) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | MUX0 | |

| 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 |
| **ADCSSCTL3, type R/W, offset 0x0A4, reset 0x0000.0002** (see page 352) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TS0 | IE0 | END0 | D0 |
| **ADCTMLB, type R/W, offset 0x100, reset 0x0000.0000** (see page 353) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | LB |
| **Universal Asynchronous Receivers/Transmitters (UARTs)**<br>UART0 base: 0x4000.C000<br>UART1 base: 0x4000.D000 | | | | | | | | | | | | | | | |
| **UARTDR, type R/W, offset 0x000, reset 0x0000.0000** (see page 361) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | OE | BE | PE | FE | DATA | | | | | | | |
| **UARTRSR/UARTECR, type RO, offset 0x004, reset 0x0000.0000 (Reads)** (see page 363) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | OE | BE | PE | FE |
| **UARTRSR/UARTECR, type WO, offset 0x004, reset 0x0000.0000 (Writes)** (see page 363) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | DATA | | | | | | | |
| **UARTFR, type RO, offset 0x018, reset 0x0000.0090** (see page 365) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | TXFE | RXFF | TXFF | RXFE | BUSY | | | |
| **UARTIBRD, type R/W, offset 0x024, reset 0x0000.0000** (see page 367) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| DIVINT | | | | | | | | | | | | | | | |
| **UARTFBRD, type R/W, offset 0x028, reset 0x0000.0000** (see page 368) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | DIVFRAC | | | | | |
| **UARTLCRH, type R/W, offset 0x02C, reset 0x0000.0000** (see page 369) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | SPS | WLEN | | FEN | STP2 | EPS | PEN | BRK |
| **UARTCTL, type R/W, offset 0x030, reset 0x0000.0300** (see page 371) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | RXE | TXE | LBE | | | | | | | UARTEN |
| **UARTIFLS, type R/W, offset 0x034, reset 0x0000.0012** (see page 373) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | RXIFLSEL | | | TXIFLSEL | | |
| **UARTIM, type R/W, offset 0x038, reset 0x0000.0000** (see page 375) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | OEIM | BEIM | PEIM | FEIM | RTIM | TXIM | RXIM | | | | |
| **UARTRIS, type RO, offset 0x03C, reset 0x0000.000F** (see page 377) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | OERIS | BERIS | PERIS | FERIS | RTRIS | TXRIS | RXRIS | | | | |
| **UARTMIS, type RO, offset 0x040, reset 0x0000.0000** (see page 378) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | OEMIS | BEMIS | PEMIS | FEMIS | RTMIS | TXMIS | RXMIS | | | | |
| **UARTICR, type W1C, offset 0x044, reset 0x0000.0000** (see page 379) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | OEIC | BEIC | PEIC | FEIC | RTIC | TXIC | RXIC | | | | |
| **UARTPeriphID4, type RO, offset 0xFD0, reset 0x0000.0000** (see page 381) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID4 | | | | | | | |

| 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 |
| **UARTPeriphID5, type RO, offset 0xFD4, reset 0x0000.0000** (see page 382) | | | | | | | | | | | | | | | |
| | | | | | | | | PID5 | | | | | | | |
| **UARTPeriphID6, type RO, offset 0xFD8, reset 0x0000.0000** (see page 383) | | | | | | | | | | | | | | | |
| | | | | | | | | PID6 | | | | | | | |
| **UARTPeriphID7, type RO, offset 0xFDC, reset 0x0000.0000** (see page 384) | | | | | | | | | | | | | | | |
| | | | | | | | | PID7 | | | | | | | |
| **UARTPeriphID0, type RO, offset 0xFE0, reset 0x0000.0011** (see page 385) | | | | | | | | | | | | | | | |
| | | | | | | | | PID0 | | | | | | | |
| **UARTPeriphID1, type RO, offset 0xFE4, reset 0x0000.0000** (see page 386) | | | | | | | | | | | | | | | |
| | | | | | | | | PID1 | | | | | | | |
| **UARTPeriphID2, type RO, offset 0xFE8, reset 0x0000.0018** (see page 387) | | | | | | | | | | | | | | | |
| | | | | | | | | PID2 | | | | | | | |
| **UARTPeriphID3, type RO, offset 0xFEC, reset 0x0000.0001** (see page 388) | | | | | | | | | | | | | | | |
| | | | | | | | | PID3 | | | | | | | |
| **UARTPCellID0, type RO, offset 0xFF0, reset 0x0000.000D** (see page 389) | | | | | | | | | | | | | | | |
| | | | | | | | | CID0 | | | | | | | |
| **UARTPCellID1, type RO, offset 0xFF4, reset 0x0000.00F0** (see page 390) | | | | | | | | | | | | | | | |
| | | | | | | | | CID1 | | | | | | | |
| **UARTPCellID2, type RO, offset 0xFF8, reset 0x0000.0005** (see page 391) | | | | | | | | | | | | | | | |
| | | | | | | | | CID2 | | | | | | | |
| **UARTPCellID3, type RO, offset 0xFFC, reset 0x0000.00B1** (see page 392) | | | | | | | | | | | | | | | |
| | | | | | | | | CID3 | | | | | | | |
| **Synchronous Serial Interface (SSI)**<br>SSI0 base: 0x4000.8000 | | | | | | | | | | | | | | | |
| **SSICR0, type R/W, offset 0x000, reset 0x0000.0000** (see page 405) | | | | | | | | | | | | | | | |
| | | | | | | | | SCR | | | | | | SPH | SPO |
| FRF | | DSS | | | | | | | | | | | | | |
| **SSICR1, type R/W, offset 0x004, reset 0x0000.0000** (see page 407) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | SOD | MS | SSE | LBM |
| **SSIDR, type R/W, offset 0x008, reset 0x0000.0000** (see page 409) | | | | | | | | | | | | | | | |
| DATA | | | | | | | | | | | | | | | |
| **SSISR, type RO, offset 0x00C, reset 0x0000.0003** (see page 410) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | BSY | RFF | RNE | TNF | TFE |
| **SSICPSR, type R/W, offset 0x010, reset 0x0000.0000** (see page 412) | | | | | | | | | | | | | | | |
| | | | | | | | | CPSDVSR | | | | | | | |
| **SSIIM, type R/W, offset 0x014, reset 0x0000.0000** (see page 413) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TXIM | RXIM | RTIM | RORIM |

| 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 |
| **SSIRIS, type RO, offset 0x018, reset 0x0000.0008** (see page 415) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TXRIS | RXRIS | RTRIS | RORRIS |
| **SSIMIS, type RO, offset 0x01C, reset 0x0000.0000** (see page 416) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | TXMIS | RXMIS | RTMIS | RORMIS |
| **SSIICR, type W1C, offset 0x020, reset 0x0000.0000** (see page 417) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | RTIC | RORIC |
| **SSIPeriphID4, type RO, offset 0xFD0, reset 0x0000.0000** (see page 418) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID4 | | | | | | | |
| **SSIPeriphID5, type RO, offset 0xFD4, reset 0x0000.0000** (see page 419) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID5 | | | | | | | |
| **SSIPeriphID6, type RO, offset 0xFD8, reset 0x0000.0000** (see page 420) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID6 | | | | | | | |
| **SSIPeriphID7, type RO, offset 0xFDC, reset 0x0000.0000** (see page 421) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID7 | | | | | | | |
| **SSIPeriphID0, type RO, offset 0xFE0, reset 0x0000.0022** (see page 422) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID0 | | | | | | | |
| **SSIPeriphID1, type RO, offset 0xFE4, reset 0x0000.0000** (see page 423) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID1 | | | | | | | |
| **SSIPeriphID2, type RO, offset 0xFE8, reset 0x0000.0018** (see page 424) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID2 | | | | | | | |
| **SSIPeriphID3, type RO, offset 0xFEC, reset 0x0000.0001** (see page 425) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | PID3 | | | | | | | |
| **SSIPCellID0, type RO, offset 0xFF0, reset 0x0000.000D** (see page 426) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID0 | | | | | | | |
| **SSIPCellID1, type RO, offset 0xFF4, reset 0x0000.00F0** (see page 427) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID1 | | | | | | | |
| **SSIPCellID2, type RO, offset 0xFF8, reset 0x0000.0005** (see page 428) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID2 | | | | | | | |
| **SSIPCellID3, type RO, offset 0xFFC, reset 0x0000.00B1** (see page 429) | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | CID3 | | | | | | | |
| **Inter-Integrated Circuit ($I^2C$) Interface**<br>**$I^2C$ Master**<br>I2C 0 base: 0x4002.0000 | | | | | | | | | | | | | | | |
| **I2CMSA, type R/W, offset 0x000, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | SA | | | | | | | R/S |

| 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 |
| **I2CMCS, type RO, offset 0x004, reset 0x0000.0000 (Reads)** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | BUSBSY | IDLE | ARBLST | DATACK | ADRACK | ERROR | BUSY |
| **I2CMCS, type WO, offset 0x004, reset 0x0000.0000 (Writes)** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | ACK | STOP | START | RUN |
| **I2CMDR, type R/W, offset 0x008, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | DATA | | | | | | | |
| **I2CMTPR, type R/W, offset 0x00C, reset 0x0000.0001** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | TPR | | | | | | |
| **I2CMIMR, type R/W, offset 0x010, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | IM |
| **I2CMRIS, type RO, offset 0x014, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | RIS |
| **I2CMMIS, type RO, offset 0x018, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | MIS |
| **I2CMICR, type WO, offset 0x01C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | IC |
| **I2CMCR, type R/W, offset 0x020, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | SFE | MFE | | | | LPBK |

## Inter-Integrated Circuit ($I^2C$) Interface

### $I^2C$ Slave

I2C 0 base: 0x4002.0000

| 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 |
| **I2CSOAR, type R/W, offset 0x800, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | OAR | | | | | | |
| **I2CSCSR, type RO, offset 0x804, reset 0x0000.0000 (Reads)** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | FBR | TREQ | RREQ |
| **I2CSCSR, type WO, offset 0x804, reset 0x0000.0000 (Writes)** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | DA |
| **I2CSDR, type R/W, offset 0x808, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | DATA | | | | | | | |
| **I2CSIMR, type R/W, offset 0x80C, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | DATAIM |
| **I2CSRIS, type RO, offset 0x810, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | DATARIS |
| **I2CSMIS, type RO, offset 0x814, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | DATAMIS |

| 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 |
| **I2CSICR, type WO, offset 0x818, reset 0x0000.0000** | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | DATAIC |

# C Ordering and Contact Information

## C.1 Ordering Information

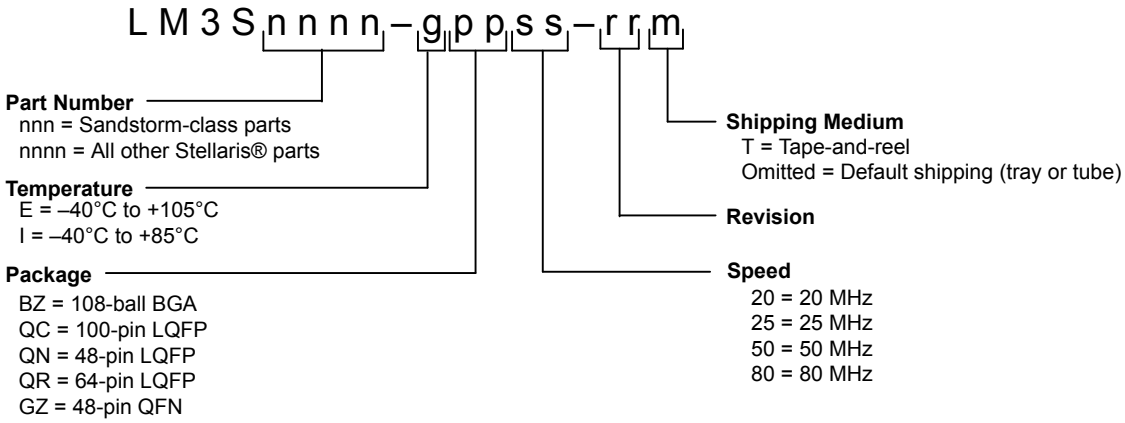



**Table C-1. Part Ordering Information**

| Orderable Part Number | Description |
|---|---|
| LM3S828-IQN50-C2 | Stellaris® LM3S828 Microcontroller Industrial Temperature 48-pin LQFP |
| LM3S828-IQN50-C2T | Stellaris LM3S828 Microcontroller Industrial Temperature 48-pin LQFP Tape-and-reel |
| LM3S828-EQN50-C2 | Stellaris LM3S828 Microcontroller Extended Temperature 48-pin LQFP |
| LM3S828-EQN50-C2T | Stellaris LM3S828 Microcontroller Extended Temperature 48-pin LQFP Tape-and-reel |
| LM3S828-IGZ50-C2 | Stellaris LM3S828 Microcontroller Industrial Temperature 48-pin QFN |
| LM3S828-IGZ50-C2T | Stellaris LM3S828 Microcontroller Industrial Temperature 48-pin QFN Tape-and-reel |
| LM3S828-EGZ50-C2 | Stellaris LM3S828 Microcontroller Extended Temperature 48-pin QFN |
| LM3S828-EGZ50-C2T | Stellaris LM3S828 Microcontroller Extended Temperature 48-pin QFN Tape-and-reel |

## C.2 Part Markings

The Stellaris microcontrollers are marked with an identifying number. This code contains the following information:

- The first line indicates the part number. In the example figure below, this is the LM3S6965.
- In the second line, the first seven characters indicate the temperature, package, speed, and revision. In the example below, this is an Industrial temperature (I), 100-pin LQFP package (QC), 50-MHz (50), revision A2 (A2) device.
- The remaining characters contain internal tracking numbers.


## C.3 Kits

The Stellaris Family provides the hardware and software tools that engineers need to begin development quickly.

- Reference Design Kits accelerate product development by providing ready-to-run hardware and comprehensive documentation including hardware design files
- Evaluation Kits provide a low-cost and effective means of evaluating Stellaris microcontrollers before purchase
- Development Kits provide you with all the tools you need to develop and prototype embedded applications right out of the box

See the website at www.ti.com/stellaris for the latest tools available, or ask your distributor.

## C.4 Support Information

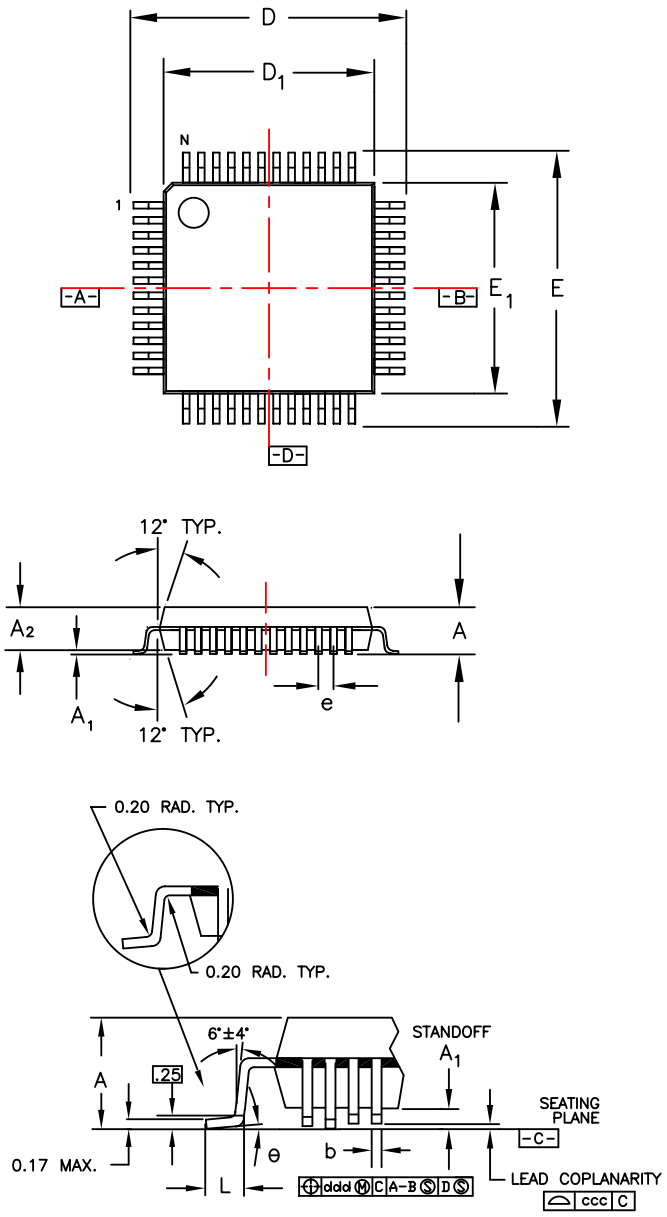
For support on Stellaris products, contact the TI Worldwide Product Information Center nearest you: http://www-k.ext.ti.com/sc/technical-support/product-information-centers.htm.

# D Package Information

## D.1 48-Pin LQFP Package

### D.1.1 Package Dimensions

**Figure D-1. 48-Pin LQFP Package**



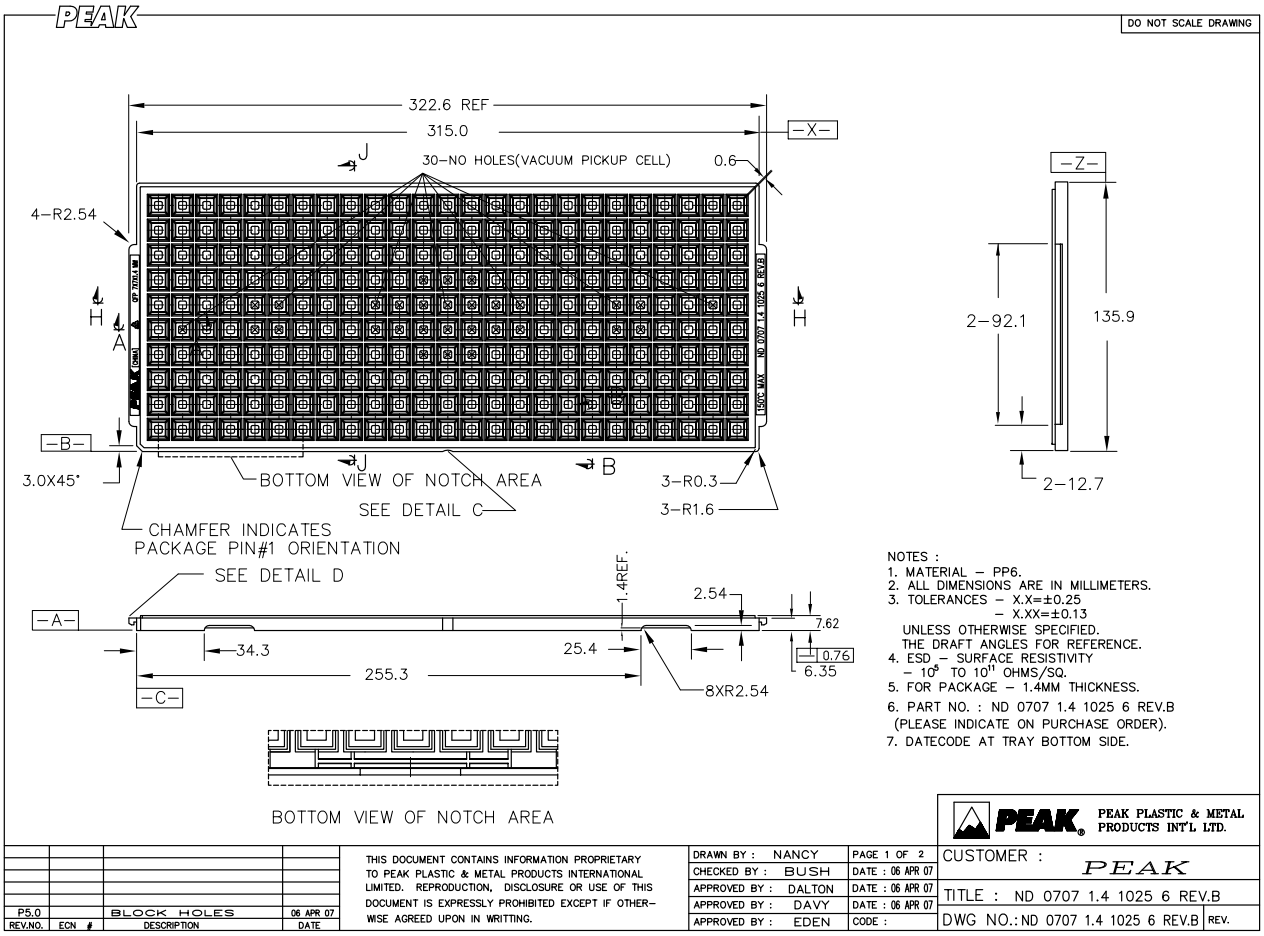

**Note:** The following notes apply to the package drawing.

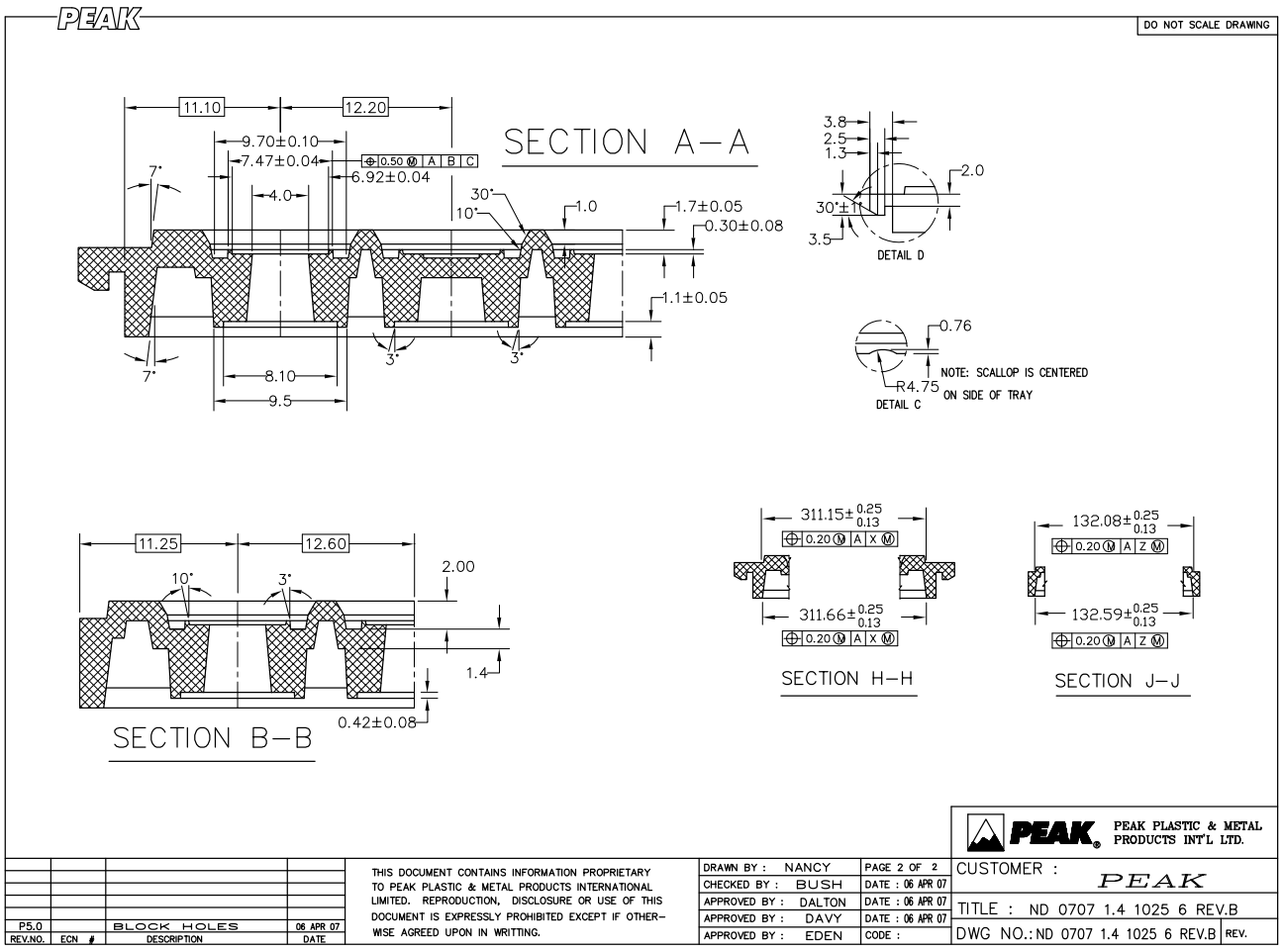
1. All dimensions are in mm.
2. Dimensions shown are nominal with tolerances indicated.
3. Foot length "L" is measured at gage plane 0.25 mm above seating plane.
4. L/F: Eftec 64T Cu or equivalent, 0.127 mm (0.005") thick.

| Symbol | Package Type | | Note |
|---|---|---|---|
| | 48LD LQFP | | |
| | MIN | MAX | |
| A | - | 1.60 | |
| $A_1$ | 0.05 | 0.15 | |
| $A_2$ | - | 1.40 | |
| D | 9.00 | | |
| $D_1$ | 7.00 | | |
| E | 9.00 | | |
| $E_1$ | 7.00 | | |
| L | 0.60 | | |
| e | 0.50 | | |
| b | 0.22 | | |
| theta | 0° - 7° | | |
| ddd | 0.08 | | |
| ccc | 0.08 | | |
| JEDEC Reference Drawing | | | MS-026 |
| Variation Designator | | | BBC |

## D.1.2 Tray Dimensions

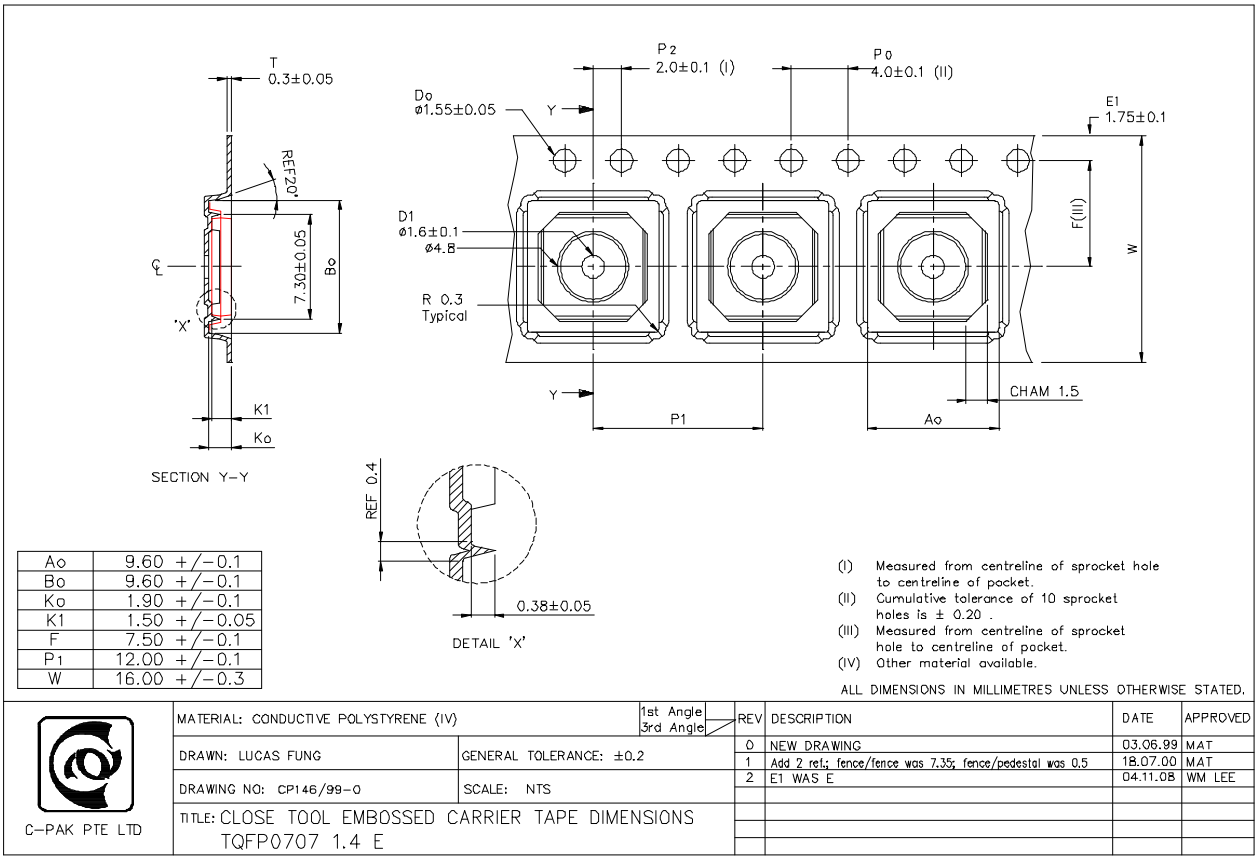
**Figure D-2. 48-Pin LQFP Tray Dimensions**





## D.1.3 Tape and Reel Dimensions

**Figure D-3. 48-Pin LQFP Tape and Reel Dimensions**



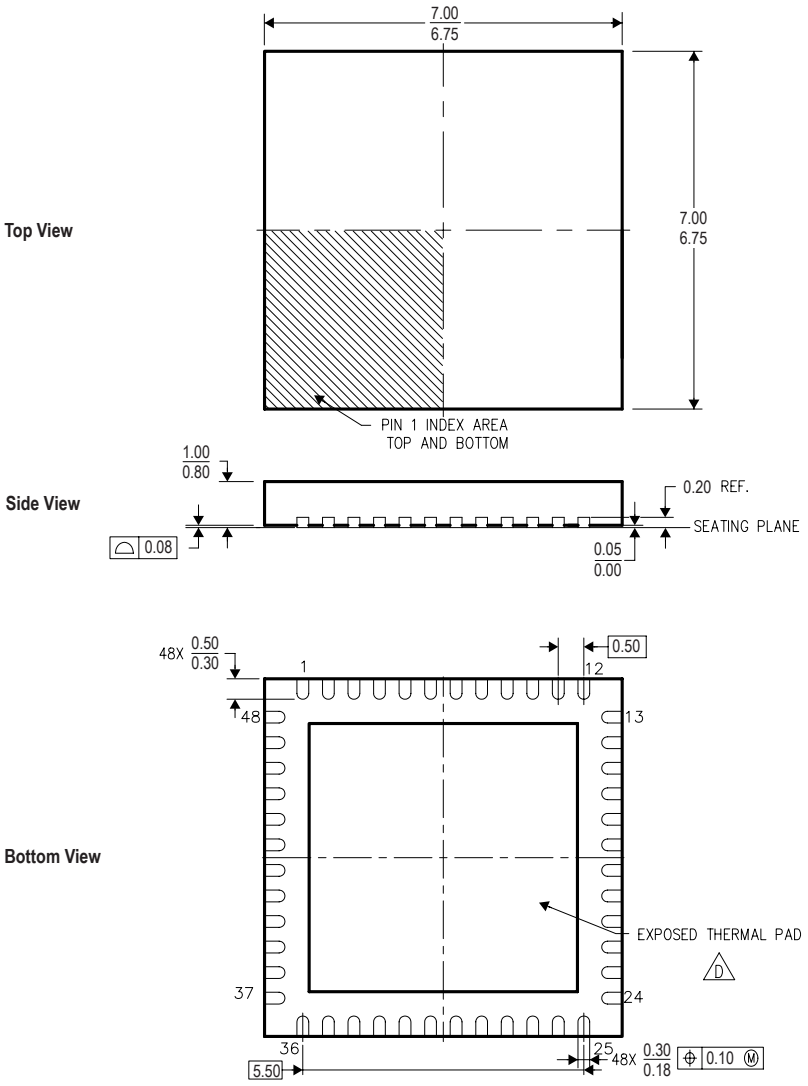

| Ao | 9.60 +/-0.1 |
|---|---|
| Bo | 9.60 +/-0.1 |
| Ko | 1.90 +/-0.1 |
| K1 | 1.50 +/-0.05 |
| F | 7.50 +/-0.1 |
| P1 | 12.00 +/-0.1 |
| W | 16.00 +/-0.3 |

| REV | DESCRIPTION | DATE | APPROVED |
|---|---|---|---|
| 0 | NEW DRAWING | 03.06.99 | MAT |
| 1 | Add 2 ref.; fence/fence was 7.35; fence/pedestal was 0.5 | 18.07.00 | MAT |
| 2 | E1 WAS E | 04.11.08 | WM LEE |

# D.2 48-Pin QFN Package

## D.2.1 Package Dimensions

**Figure D-4. 48-Pin QFN Package**




NOTES: A. All linear dimensions are in millimeters. Dimensioning and tolerancing per ASME Y14.5M-1994.
B. This drawing is subject to change without notice.
C. Quad Flatpack, No-leads (QFN) package configuration.
D. The package thermal pad must be soldered to the board for thermal and mechanical performance. In addition, the pad must be connected to GND.
E. Falls within JEDEC MO-220.

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

| **Products** | | **Applications** | |
|---|---|---|---|
| Audio | www.ti.com/audio | Communications and Telecom | www.ti.com/communications |
| Amplifiers | amplifier.ti.com | Computers and Peripherals | www.ti.com/computers |
| Data Converters | dataconverter.ti.com | Consumer Electronics | www.ti.com/consumer-apps |
| DLP® Products | www.dlp.com | Energy and Lighting | www.ti.com/energy |
| DSP | dsp.ti.com | Industrial | www.ti.com/industrial |
| Clocks and Timers | www.ti.com/clocks | Medical | www.ti.com/medical |
| Interface | interface.ti.com | Security | www.ti.com/security |
| Logic | logic.ti.com | Space, Avionics and Defense | www.ti.com/space-avionics-defense |
| Power Mgmt | power.ti.com | Transportation and Automotive | www.ti.com/automotive |
| Microcontrollers | microcontroller.ti.com | Video and Imaging | www.ti.com/video |
| RFID | www.ti-rfid.com | Wireless | www.ti.com/wireless-apps |
| RF/IF and ZigBee® Solutions | www.ti.com/lprf | | |

**TI E2E Community Home Page** e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265